A Model-Based Approach to Reliability Certification of Services

Claudio A. Ardagna, Ernesto Damiani, Ravi Jhawar, Vincenzo Piuri
DTI – Università degli Studi di Milano
Crema (CR), 26013, Italy
Email: firstname.lastname@unimi.it

***Abstract*—We present a reliability certification scheme in which services are modeled as discrete-time Markov chains. A machine-readable certificate is issued to the service after validating its reliability properties, and validity of the certificate is verified using constant run-time monitoring. In addition, we present a solution that allows users to search and select services with a given set of reliability properties. Our solution is integrated within existing Service-Oriented Architectures (SOAs), and allows validation of users' preferences both at discovery-time and at run-time.**

***Index Terms*—Markov Chains, Reliability Certification, SOA, Web Services**

I. Introduction

The increasing demand for flexibility and extensibility in software reuse and integration has resulted in a wide adoption of web services and SOA applications. The availability of a range of published services by different providers, coupled with XML-based web service protocols, form a digital ecosystem that allows the design and orchestration of business processes across organization boundaries [8], [15]. While the benefits are immense, this dynamic paradigm of building software has changed the dimension and scope of risks in business process development and, as a result, users are increasingly concerned about failures that may affect functional and non-functional properties of their applications. In this context, trustworthiness of services is a critical factor for users, and raises the need of adapting current development, verification, validation, and certification techniques to the SOA vision [5], [7], [9]. In particular, the definition of assurance techniques increasing users' confidence that a service complies with their reliability requirements becomes of utmost importance.

An emerging paradigm to address this issue is based on certification of services [2], [6]. This approach, in contrast to other software certification solutions (e.g., Common Criteria [10]) that provide human-readable certificates, relies on machine-readable certificates that can be used both at discovery-time and at run-time. In this paper, we focus on reliability certification where reliability properties of a service are validated by means of formal model-checking. In particular, we use Markov models to establish whether the service supports a given reliability property with a given assurance level. The result of a property validation is a machine-readable certificate that represents the reasons why the service supports a reliability property and serves as a proof to the users that appropriate reliability mechanisms have been considered while building it. Our certification scheme also provides a mechanism where users can search and select services with a certificate proving a set of reliability properties. To complement the dynamic nature of SOA, properties in the certificates are continuously verified by means of run-time monitoring.

The paper is organized as follows. Section II describes the reference scenario and basic concepts on reliability certification. Section III presents an approach to service modeling. Section IV describes our two-phase reliability certification process. Section V discusses how to integrate the proposed certification process within SOA to effectively match services' certificates with users' requirements. Section VI summarizes the related work, and Section VII presents our conclusions.

II. Reference Scenario and Basic Concepts

We describe the reference scenario and some basic concepts on reliability certification.

A. Reference Scenario

We consider a highly dynamic and distributed service-based infrastructure which involves the following main parties.

- *Certification Authority*: The trusted entity that certifies reliability properties of services.
- *Service Provider*: The entity that implements a service, and engages with the certification authority to obtain a certificate for its service.
- *Client*: The entity that establishes a business relationship with one or more service providers, and uses a set of certified services to implement its business process.
- *Service Discovery*: A registry of published services that is enhanced to support reliability certification.

In this paper, as an example, we consider a service provider offering a web-based storage service which allows its clients to store and retrieve data over the Internet. A client can integrate the storage service in its business process by accessing the interface published by the service provider. The service implements a *result* `write`(*data*, *metadata*) operation that takes the *data* and *metadata* as input from the client, stores them on the remote server, and returns the *result* (*success* or *failure*) of the operation. The data can then be retrieved by the client by sending a query to the remote server using the *data* `read`(*query*) operation, and can be deleted using the *result* `delete`(*metadata*) operation. When a client implements

its business process using the storage service, the functional correctness of the business process strongly depends on the storage service and, as a consequence, reliability of the storage service becomes of paramount importance to the client. In this context, a reliability certificate can serve as an effective means of assurance to the client, by providing a proof that the storage service supports a given set of reliability properties.

B. Basic Concepts

A service provider implements its service using a set of reliability mechanisms, and engages with the certification authority in a process that *i)* validates the reliability properties of the implemented service, and *ii)* issues a certificate to the service provider based on the results of the validation tests. To validate the reliability properties of a given service, the certification scheme must define a hierarchy of reliability properties to be certified, and a policy that contains all conditions necessary to assess and prove that a given reliability property holds for a service.

Hierarchy of reliability properties: We define a set of abstract properties to represent the general purpose reliability requirements of the service under certification. Examples of abstract properties are reliability, availability, integrity, and durability. A concrete property p=$(\hat{p}, A)$ enriches an abstract property $\hat{p}$ with a set A of attributes that refer to the type of failures the service proves to tolerate, the reliability mechanisms used to realize the service, or to a specific configuration of reliability mechanisms that characterizes the service to be certified. For each attribute $a{\in}A$, a partial (or total) order relationship $\preceq_a$ can be defined on its domain Ω_a, and $v(a)$ represents the value of a. If an attribute does not contribute to a property configuration, its value is not specified. In general, property attributes represent service provider's claims on the reliability of its service (e.g., when a=fault_type, $v(a)$ can be "crash failure" or "programming error" or "byzantine failure").[1] A hierarchical ordering of reliability properties can then 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,$p_j{\in}\mathcal{P}$, $p_i{\preceq_P}p_j$ if *i)* $p_i.\hat{p}$=$p_j.\hat{p}$ and *ii)* $\forall a{\in}A$ either $v_i(a)$ is not specified or $v_i(a){\preceq_a}v_j(a)$. In other words, $p_i{\preceq_P}p_j$ means that a service certified for p_j always holds p_i. For instance, given p_1=(reliability,{mechanism=redundancy, level=2, fault_type=server_crashes, failure_detection=heartbeat_test}) and p_2=(reliability,{mechanism=redundancy, level=3, fault_type=server_crashes, failure_detection=heartbeat_test, max_recovery_time=15ms}), $p_1{\preceq_P}p_2$.

Policy: A certification scheme must verify that a reliability property p is supported by the service. To this aim, we first define a policy $Pol(p){\rightarrow}\{c_1, c_2, \ldots, c_m\}$ that contains all conditions $c_1, \ldots, c_m$ necessary to prove that p holds for the service. We then enhance the model of the service with $Pol(p)$ to validate and prove p (see Section III). We note that proving a concrete property is equivalent to validating the reliability mechanisms used to implement the service. Based on this observation, we can define a policy corresponding to each property configuration, where each condition c_i in the policy defines a relationship derived by attributes in A. For instance, for a reliability property p=(reliability,{mechanism=redundancy, level=3, fault_type=server_crashes, failure_detection=heartbeat_test, max_recovery_time=15ms}), a policy can be defined with conditions: c_1:no_of_server_instances$\geq$3, c_2:recovery_time$\leq$15ms. Here, each condition c_i is of the form c:$\{a\ op\ v\}$, where op specifies a relationship on an attribute a and its expected value v based on the reliability property to be certified and reliability mechanisms used to implement the service. In this context, a reliability certificate is granted to the service when it satisfies all the conditions in the policy and proves that it holds a reliability property p with a given level of assurance (see Section IV).

[1]Note that, the fault model is equivalent to the attacker model in terms of security properties of a service.

III. Service Modeling

The modeling of services as finite state automata has been used in the past to estimate and improve reliability, as well as to test and certify security properties of services (e.g., [1], [3], [5], [9]). The certification scheme presented in this paper uses state-based, discrete-time Markov models that combine the failure behavior and system architecture of the service to validate and certify a given set of reliability properties. We represent the control-flow of a service in the form of a graph G=(N, E) where each node $n{\in}N$ corresponds to a state of the service, and a direct edge $(n_i, n_j){\in}E$ represents the transfer of control from node n_i to node n_j. The transition from one state to another is assumed to follow the Markov property, regardless of the point in time at which the transition occurs. The graph G is enriched with *i)* policy conditions associated to each state transition, and *ii)* two absorbing states C and F representing the states of correct output and failure, respectively. From a certification point of view, state C is reached when the service satisfies the policy Pol, while state F is reached in case of a failure or policy violation. The transition probability Pr_{ij} to satisfy the policy condition is associated with each directed edge (n_i, n_j), and the probability R_i to remain fail-free is associated with each node n_i. In this context, similar to [4], R_iPr_{ij} represents the probability that execution of a service in state n_i will produce the correct results, and transfer the control to state n_j. The transition from the final state n_k to the correct state C, having probability R_k, is observed if the service satisfies all conditions in Pol (with no failures). We note that there is an implicit transition of probability $1{-}\sum_{j=1}^{k} R_iPr_{ij}$ from each node $n_i{\neq}n_k$ to F representing a failure or violation of the policy condition at that node. The transition from n_k to F has probability $1{-}R_k$. A model deduced from the graph G is defined as follows.

Definition 1 (Service model)**.** The model of a service is a 7-tuple $\{n_1, N_{\mathcal{I}}, n_k, C, F, \overset{c_{ij}}{\rightarrow}, R_i Pr_{ij}\}$, where: n_1 is the initial state; $N_{\mathcal{I}}$ represents all intermediate states $n_2, \ldots, n_{k-1}$; n_k is the final execution state; C is the final correct state; F is the final failure state; $\overset{c_{ij}}{\rightarrow}$ represents a transition between two nodes (n_i, n_j) labeled with a policy condition $c_{ij} \in Pol(p)$; and $R_i Pr_{ij}$ is the probability that the service execution provides the correct results and satisfies the policy conditions in state n_i, and moves to state n_j.

A model is represented by a transition matrix Q' as follows.

$$Q' = \begin{array}{c} \\ C \\ F \\ n_1 \\ n_2 \\ \vdots \\ n_k \end{array} \begin{array}{c} \begin{array}{cccccc} C & F & n_1 & n_2 & \ldots & n_k \end{array} \\ \begin{pmatrix} 1 & 0 & 0 & 0 & \ldots & 0 \\ 0 & 1 & 0 & 0 & \ldots & 0 \\ 0 & 1 - \sum_{j=1}^{k} R_1 Pr_{1j} & R_1 Pr_{11} & R_1 Pr_{12} & \ldots & R_1 Pr_{1k} \\ 0 & 1 - \sum_{j=1}^{k} R_2 Pr_{2j} & R_2 Pr_{21} & R_2 Pr_{22} & \ldots & R_2 Pr_{2k} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ R_k & 1 - R_k & 0 & 0 & \ldots & 0 \end{pmatrix} \end{array}$$

We note that, for the sake of clarity, policy conditions are not reported in Q'. The certification authority uses the matrix Q' to estimate the extent to which the service satisfies the reliability property p by following the approach presented in [4]. 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 2 (Assurance level)**.** The assurance level L of a service deployed in a specific environment is the probability that it satisfies a policy $Pol(p)$ and holds a reliability 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 n_1 will reach the final execution state n_k, and transit from n_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 1^{st} 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.

Example 1. Figure 1 shows an example of a Markov model enhanced with policy conditions, representing the write operation of the storage service. The service starts from the initial state n_1 when it receives a valid input during a *result* `write`(*data*, *metadata*) operation. In this state, failures may happen due to some unexpected errors in input management (e.g., request overflow and timeout), resulting in a direct

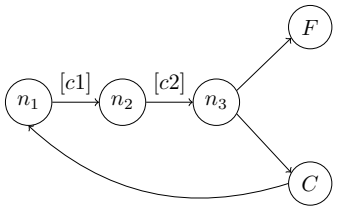

Fig. 1. An example of a model of the storage service for the write operation

(n_1, F) transition. For the service to move from the input state n_1 to n_2, the policy condition [c1] calls a method to check if the data provided by the user have been managed correctly, and if the index is successfully generated.[2] In state n_2, data, metadata, and index are stored across all redundant storage servers. The corresponding policy condition [c2] checks if *success* is returned by all servers. A policy violation/failure in this state results in a transition to the failure state F. State n_3 represents the final output state of the service for a write operation. An implicit transition from n_3, to the final correct state C happens with a probability R_3, resulting in a fail-free operation. Let us assume that fail-free probabilities of the nodes computed by the certification authority are: $R_1 = 0.99$, $R_2 = 0.94$, and $R_3 = 0.97$, and transition probabilities between nodes are $Pr_{12} = 0.93$ and $Pr_{23} = 0.95$. The corresponding transition matrix Q' is

$$Q' = \begin{array}{c} \\ C \\ F \\ n_1 \\ n_2 \\ n_3 \end{array} \begin{array}{c} \begin{array}{ccccc} C & F & n_1 & n_2 & n_3 \end{array} \\ \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0.0793 & 0 & 0.9207 & 0 \\ 0 & 0.107 & 0 & 0 & 0.893 \\ 0.97 & 0.03 & 0 & 0 & 0 \end{pmatrix} \end{array}$$

and the matrix μ is:

$$\mu = (I - Q)^{-1} = \begin{pmatrix} 1 & 0.9207 & 0.8222 \\ 0 & 1 & 0.893 \\ 0 & 0 & 1 \end{pmatrix}$$

Here, $\mu_{1,3}$=0.8222. The probability of the storage service to satisfy a reliability property is L=0.8222∗0.97=0.7975.

IV. Reliability Certification Process

The task of certifying reliability of services involves specification of reliability properties to be certified, definition of efficient policies that can sufficiently prove that each property holds, and deduction of a model to validate and verify policies against the real service implementation. In this section, we present our certification scheme which is designed as a two-phase process. The first phase validates the reliability properties of a service before it is actually deployed in a system (offline), and issues a certificate to the service based on initial validation results. The second phase monitors the certified properties of each service at run-time (online). For

[2]For the sake of simplicity, the model of the service does not consider exception management.

simplicity, in the following, we assume a certification process which proves (and awards certificates with) a single property.

A. Offline Phase

The offline phase starts when a service provider requests the certification authority to issue a certificate for a reliability property p to its service. The model used to validate the property p of the service can be either generated by the certification authority itself, or directly provided by the service provider.[3] After verifying that the model conforms to the real service implementation, the certification authority selects a policy $Pol(p)$ based on p and identifies the reliability mechanisms used to implement the service. The certification authority then enhances the model with policy conditions, and defines a validation function to verify that the service satisfies all the conditions. We assume that each condition is specified as a boolean valued predicate. The validation function is formally defined as follows.

Definition 3 (Validation function)**.** A validation function f:$(s, p, Pol(p), M, k)$→$\{true, false\}$ takes the service s under evaluation, the reliability property p to be validated, the policy $Pol(p)$ selected by the certification authority, the model M of the service, and an index k that refers to the service execution that triggers policy verification as input, and returns $true$ when all conditions in $Pol(p)$ corresponding to p are satisfied with respect to M, $false$ otherwise, as output.

We note that the model of the service, the reliability property, and the policy 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 reliability property of the service under different contexts (e.g., using fault injection).

Example 2. In Figure 2, we extend the model in Figure 1 with the stateful implementation of a three-replica storage service to certify p=(reliability,{mechanism=redundancy, level=3, fault_type=server_crashes, failure_detection=heartbeat_test, max_recovery_time=15ms}). Based on this model, the certification authority validates p by performing a sequence of tests. At each test iteration for the write operation, the service starts from state n_1 and, if the condition [c1] is verified using f, it transits to the sub-state n_{2a} where the service sends the request to all three replicated storage servers. We note that, [c1] in this example is equivalent to [c1] in Example 1. At this point, the service moves to state n_{2b} if at least a server crash is detected by the heartbeat test (i.e., f verifies [c2]); otherwise, service transits to state n_{2c} when all servers are fail-free (f verifies [c3]). In state n_{2b}, the service invokes recovery mechanisms, and a transition to n_{2d} is observed only if the server crash is recovered in at most 15ms and *success* is returned by all three servers ([c4]); otherwise, the service moves to the failure state F. A transition (n_{2c}, n_{2d}) is observed if *success* is returned by all storage servers ([c5]). In the transition from n_{2d} to the final execution state n_3, f performs an integrity check by comparing the hash values of data from all three servers ([c6]). To conclude, f returns $true$ (i.e., move to C) if all conditions are satisfied, and returns $false$ (i.e., move to F), otherwise.

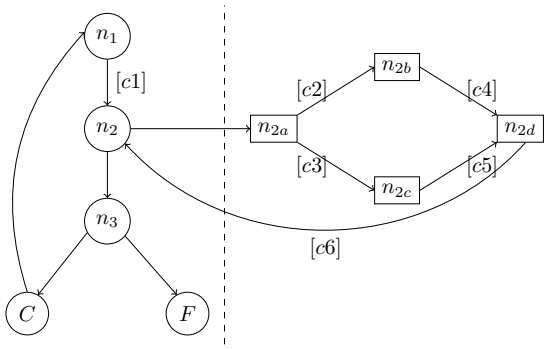

Fig. 2. An example of a model representing a stateful implementation of the storage service for the write operation

The results of the validation function are used by the certification authority to estimate the values of R_i and Pr_{ij} in Definition 1, and L in Definition 2. To this aim, we introduce a *frequency log* that maintains f's results. A frequency log is a list of triplets $(k, \{v_k\}, n_i)$, where: k is the index of the test request in Definition 3; $\{v_k\}$ represents the attribute value causing a transition to F; and n_i is the state of the service model in which a transition to F is observed. We note that $\{v_k\}$ and n_i are empty if f returns $true$. Each probability $R_i Pr_{ij}$ can then be calculated, using the frequency log, as the number of successful transitions from n_i to n_j over the total number of test requests reaching n_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, suppose that the service fails to recover from a server crash in state n_{2b}; the frequency log registers k, the state n_{2b}, and the snapshot of changed real-attribute values, such as, no_of_server_instances=2. We note that, in Example 2, the probability value $R_2 Pr_{23}$ is deduced from the results of the validation function f in states n_{2a}−n_{2d}.

The assurance level L in Definition 2 is then quantified by performing the matrix operations described in Section III, and used to characterize the reliability property of a service. A reliability certificate is issued to the service, if the value of L is above a certain predefined threshold $\mathcal{T}$, and is of the form $\mathcal{C}(p, M, L)$, where: *i)* p represents the reliability property supported by the service; *ii)* M represents the model that is used to validate p. M serves as a basis for the certification authority to prove to the client that the service satisfies the conditions defined in $Pol(p)$ and supports p; *iii)* L represents the assurance level with which the service supports the reliability property p.

[3]We note that the service model is neither used to estimate the functional behavior of the service nor the overall reliability, but to validate reliability properties supported by the service and to consequently generate a certificate.

B. 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 reliability certificate issued to the service, since, in dynamic complex digital ecosystems, reliability properties may change over time resulting in outdated certificates. For example, reliability of the service may be affected 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 service is deployed to monitor the reliability property of the service using its model.

Since a Markov model generates all possible states of the service, the number of states can be extremely large. However, to monitor reliability properties of a service, we do not need the complete model. We therefore propose that the certification authority derives a lightweight reduced model by reducing the original one, while maintaining its accuracy. A reduced Markov model is formally defined as follows.

Definition 4 (Reduced Markov model)**.** A reduced Markov model $\tilde{M}$, which is derived from the original model M, is of the form $\tilde{M}=\{n_1, \tilde{N_\mathcal{I}}, n_k, C, F, \overset{c_{ij}}{\rightarrow}, R_i Pr_{ij}\}$, such that, $|\tilde{M}(\tilde{N_\mathcal{I}})|<|M(N_\mathcal{I})|$ and, for all validation tests, *i)* $f(s,p,Pol(p),\tilde{M},k)$=$f(s,p,Pol(p),M,k)$ and *ii)* the frequency logs for M and $\tilde{M}$ are consistent.

We note that the frequency logs are consistent if, for each entry $(k,\{v_k\},n_i)$ in M and $(\tilde{k},\{\tilde{v}_k\},\tilde{n}_i)$ in $\tilde{M}$, k=$\tilde{k}$, $\{v_k\}$=$\{\tilde{v}_k\}$, and n_i=$\tilde{n}_i$ or $\tilde{n}_i$ is a combination of states including n_i. For example, states n_{2b} and n_{2c} of the original model M in Figure 2 can be combined to a single state $\tilde{n}_{2bc}$ to obtain a reduced Markov model $\tilde{M}$. In $\tilde{M}$, service moves from n_{2a} to $\tilde{n}_{2bc}$ following a combination of [c2] and [c3], and from $\tilde{n}_{2bc}$ to n_{2d} following a combination of [c4] and [c5]. We note that, the transition $(\tilde{n}_{2bc},F)$ is a combination of transitions (n_{2b},F) and (n_{2c},F) in M. The results of f using $\tilde{M}$ are then the same as the ones obtained using M.

At run-time, evaluation body performs monitoring of the service executions by obtaining real-attribute values using $\tilde{M}$. A reliability certificate issued to a service remains valid if its real-attribute values satisfy all conditions in the policy with a given level of assurance. For each service execution validated using $f(s,p,Pol(p),\tilde{M},k)$, the probability values in the original service model M (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 $L\geq\mathcal{T}$. To this aim, as in the offline phase, we use the *frequency log*, maintaining f's results, within the evaluation body. We note that the source of failure or policy violation in M can be precisely located using the frequency log and $\tilde{M}$. We also note that the update of the matrix Q' is not done in real time, but periodically, to preserve system performance.

By analogy with the offline phase, we extend the notion of L to support the validation process of the certification authority, and define a random variable L_t to characterize the reliability property of a service at run-time. 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 the matrix Q' updated using the reduced Markov model $\tilde{M}$ and the frequency log. The 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. This implies that $L_t\geq\mathcal{T}$, that is, the assurance value of the service during run-time is still greater than a predefined threshold value, and the reliability property 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 not, the certification authority either revokes the certificate, or updates the property in the certificate, based on the new value L_t and the reliability property p.

V. Reliability Certificate Matching

We aim to provide a solution where services can be searched and selected at run-time based on client's reliability requirements. To achieve this, our service discovery component provides an extended service registry to support the matching and comparison processes. The matching process performs a check in the service registry to obtain the set of services that meet client's preferences on reliability properties, whereas the comparison process performs a ranking of services within the set generated by the matching process to support a client in selecting the most appropriate service.

In the matching process, a client c first defines its preferences $R_c(p_c,L_c)$ in terms of requirements on the reliability property p_c and the assurance level L_c a service should possess. A service discovery engine then performs an automatic matching of client's requirements against the reliability certificates of services in the registry, and returns the set of services satisfying the required property with a given level of assurance. Let us consider a service discovery with a set of services, each one having a certificate of the form $C_s(p_s,M_s,L_s)$, where p_s is a reliability property, M_s is the service model, and L_s is the assurance level of the service obtained by policy validation using M_s. The matching process returns a set S of services for which $p_c\preceq_P p_s \wedge L_c\leq L_s$.

The comparison process takes S as input and generates an ordering of services. The goal of this phase is to rank the shortlisted set S of services based on a client's preferences so as to facilitate it in selecting the best service. Given two services s_i,$s_j\in S$, the ordering of services is performed based on the hierarchical relationship among reliability properties (e.g., $p_i\preceq_P p_j$) and assurance level values (e.g., $L_i\leq L_j$). We note that, in some cases, there can be inconsistencies in the comparison (e.g., $p_i\preceq_P p_j$ but $L_j<L_i$). In this context, we define a precedence in which the property is more important than the assurance level. A (partially) ordered set S' of services is returned to the client as the output of the comparison phase.

Example 3. Consider a service discovery having services s_1, s_2, s_3, with certificates C_1, C_2, C_3, respectively, where:

C_1=[(reliability,{mechanism=redundancy; level=3; fault_type=server_crashes}), M_1, 0.95],
C_2=[(reliability,{mechanism=redundancy; level=4; fault_type=software_errors}), M_2, 0.98], and
C_3=[(reliability,{mechanism=redundancy; level=2; fault_type=server_crashes}), M_3, 0.90].
Then, consider that a client's preferences are:
R_c=(reliability,{mechanism=redundancy;level$>$2},$\geq$0.90).
The set S of services returned by the matching process is $\{s_1, s_2\}$. Service s_3 is not selected because $p_c \not\preceq_P p_s$, that is, redundancy level of s_3 is $\leq$2. The comparison process ranks $\{s_1, s_2\}$ and returns $\{s_2, s_1\}$ since $p_1 \preceq_P p_2 \wedge L_1 \leq L_2$.

We extend our matching and comparison processes to complement our certification scheme and provide a two-phase matching and comparison solution. In the first phase, a static matching and comparison is performed when the client sends a request to the service discovery. The second phase starts when the client selects a service $s_i \in S'$. In this phase, service discovery performs constant monitoring of the certificate status for s_i. If a certificate update is observed (i.e., $L_i < \mathcal{T}$), a comparison of client's preferences against the updated certificate $C_u(p_u, M_u, L_u)$ is performed. If $p_c \not\preceq_P p_u \vee L_u < L_c$, the service discovery triggers the matching and comparison processes to generate a new S' for the client; otherwise, the service continues uninterrupted.

VI. Related Work

Modeling of software components has been used to improve the reliability and to validate the correctness and security of software systems. An interesting work [14] relevant to this paper proposes a model-based approach to identify reliable ways of designing the interactions in a system, where each interaction is associated with the probability of its success. In [4], the authors put forward the idea that the reliability of a software system depends on the reliability of its components and the probabilistic distribution of their utilization. A simple Markov model is then used to measure the software reliability and calculate the effects of failures on the system with respect to a user environment. Other Markovian model-based approaches have been proposed to evaluate system reliability and to generate the evidence proving it (e.g., [11], [13]). In contrast to Markov models, Petri nets are often employed to verify the feasibility and correctness of business process configurations (e.g., [16]). Although these models are effective in establishing functional requirements of service compositions, they seemingly overkill our goals of validating and monitoring reliability properties of individual services and add complexity in determining assurance level values. The work most relevant to our proposal [2] presents a test-based reliability certification scheme for services that provides an a priori validation of services based on underlying reliability patterns, and a posteriori test on service reliability using a set of metrics. Differently from the above works, our proposal provides a certification scheme based on formal modeling to prove reliability properties of services at a given level of assurance. This scheme also supports matching and comparison of services based on user's requirements.

VII. Conclusions

We presented a reliability certification scheme in which a machine-readable certificate is issued to a service after validating its reliability properties using a discrete-time Markov model. The service is then continuously monitored at run-time using a reduced model to verify the validity of the issued certificate. Furthermore, we proposed a solution that allows clients to search and select services with a given set of reliability properties and ensures that client's preferences are maintained at run-time. Our future work will consider the formulation of an enhanced approach to quantify the assurance level and provide certificate updates, and a more complete evaluation of the proposed solution.

Acknowledgments

This work was partly funded by the European Commission under the project ASSERT4SOA (contract n. FP7-257351), and by the Italian Ministry of Research within the PRIN 2008 project PEPPER (2008SY2PH4).

References

[1] M. Anisetti, C. Ardagna, and E. Damiani, "Fine-grained modeling of web services for test-based security certification," in *Proc. of SCC 2011*, Washington, DC, USA, July 2011.

[2] I. Buckley, E. Fernandez, M. Anisetti, C. Ardagna, and E. Damiani, "Towards pattern-based reliability certification of services," in *Proc. of DOA-SVI 2011*, Hersonissos, Crete, Greece, October 2011.

[3] G. Canfora and M. di Penta, "Service-oriented architectures testing: A survey," *Software Engineering: International Summer Schools, ISSSE 2006-2008*, vol. 1, pp. 78–105, 2009.

[4] R. C. Cheung, "A user-oriented software reliability model," *IEEE Transactions on Software Engineering*, vol. 6, pp. 118–125, March 1980.

[5] V. Cortellessa and V. Grassi, "Reliability modeling and analysis of service-oriented architectures," in *Test and Analysis of Web Services*, 2007, pp. 339–362.

[6] E. Damiani, C. Ardagna, and N. E. Ioini, Eds., *"Open Source Systems Security Certification"*. New York, NY, USA: Springer, 2009.

[7] E. Damiani, S. De Capitani di Vimercati, S. Paraboschi, and P. Samarati, "Securing SOAP e-services," *International Journal of Information Security*, vol. 1, no. 2, pp. 100–115, February 2002.

[8] T. Erl, *Service-Oriented Architecture: Concepts, Technology, and Design*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2005.

[9] L. Frantzen, J. Tretmans, and R. d. Vries, "Towards model-based testing of web services," in *Proc. of WS-MaTe 2006*, Palermo, Italy, June 2006.

[10] D. Herrmann, *Using the Common Criteria for IT security evaluation*. Auerbach Publications, 2002.

[11] S. M. Iyer, M. K. Nakayama, and A. V. Gerbessiotis, "A markovian dependability model with cascading failures," *IEEE Transactions on Computers*, vol. 58, pp. 1238–1249, September 2009.

[12] R. Jhawar, V. Piuri, and M. Santambrogio, "A comprehensive conceptual system-level approach to fault tolerance in cloud computing," in *IEEE SysCon 2012*, Vancouver, BC, Canada, March 2012.

[13] J. K. Muppala, M. Manish, and T. K. S., "Markov dependability models of complex systems: Analysis techniques," *In Ozekici, S. (ed.) Reliability and Maintenance of Complex Systems, NATO ASI Series F: Computer and Systems Sciences*, vol. 154, pp. 442–486, 1996.

[14] S. Mustafiz, X. Sun, J. Kienzle, and H. Vangheluwe, "Model-driven assessment of system dependability," *Software and System Modeling*, vol. 7, no. 4, pp. 487–502, 2008.

[15] M. P. Papazoglou, "Web services and business transactions," *World Wide Web*, vol. 6, pp. 49–91, March 2003.

[16] W. van der Aalst, N. Lohmann, and M. La Rosa, "Ensuring correctness during process configuration via partner synthesis," *Journal of Information Systems*, vol. 37, no. 6, pp. 574–592, September 2012.