J Reliable Intell Environ (2015) 1:87–100
DOI 10.1007/s40860-015-0009-z

ORIGINAL ARTICLE

Using Bayesian networks for highly available cloud-based web applications

Stefano Marrone[1]

Received: 28 August 2015 / Accepted: 30 September 2015 / Published online: 31 October 2015

Abstract Bayesian networks have demonstrated their capability in several applications spanning from reasoning under uncertainty in artificial intelligence to dependability modelling and analysis. This paper focuses on the use of this language for allocating cloud resources to maximise service dependability. This objective is accomplished by the definition of a model-driven approach able to guide the software engineering to define a cloud infrastructure (applications, services, virtual and concrete resources) using a semi-automated process. This process exploits both high-level languages such as UML as well as Bayesian networks. Using all their features (backward analysis, ease of usage, low analysis time), Bayesian networks are used in this process as a driver for the optimization, learning and estimation phases. The paper discusses all the issues that the application of Bayesian networks in the proposed process arises.

Keywords Model-driven engineering · Dependability evaluation · Service dependability · Cloud resource allocation

1 Introduction

Last years have seen the rise of the cloud computing paradigm as a key factor to improve efficiency and scalability in the ICT industry. The decoupling of the offered services from the IT infrastructures and the possibility to optimise the allocation of the first onto the second have opened for new business ideas.[1] Entrepreneurs are more and more willing to implement highly available and scalable web applications to improve the volume of their business: cloud computing paradigm is one of the keys to pursue this goal. The IT industry and the research community have already faced with the problem of developing highly dependable applications. Dependability is one of the most sensitive features of cloud computing as reported in some white papers and research surveys [1,2]: in particular, availability is considered a first-class citizen since the great loss of money during service downtimes.

The problem to pass from an availability-oriented specification to an effective deployment of virtual resources has been quite studied in scientific literature; however, this problem is often worsened by the capability of cloud-based service and infrastructure to reuse the greatest part of existing software artefacts and design/programming paradigms. This paper focuses on the joint usage of service-oriented architecture (SOA) paradigm and cloud computing infrastructure: SOA-based applications that use cloud-based services represent a clear example of modern distributed and stratified software architecture. These architectures are built following a system-of-systems logic which also generates the problem of finding a compositional modelling and designing methodology. From a modelling perspective, this need is reflected by the exigency of aggregating (sub-)models to cope with the explosion of their extension by exploiting the same compositional logic. Guaranteeing a high-availability level for cloud infrastructures as well as for SOA-based applications separately does not bring to an optimal design of the software system that holistic approaches would allow.

✉ Stefano Marrone
stefano.marrone@unina2.it

[1] Dipartimento di Matematica e Fisica, Seconda Universitá di Napoli, viale Lincoln 5, 81100 Caserta, Italy

[1] https://www.whitehouse.gov/sites/default/files/omb/egov/digital-government/digital-government.html.

Furthermore, designing an optimal highly available system is only the starting point. Environmental and operational conditions change and, hence, the hypotheses made when the system has been designed and deployed can fall. Big "black swan" changes also can occur radically mutating the behaviour of systems. Antifragility is a modern mathematical theory and set of engineering practices which aim at designing robust systems by embracing the change through flexibility and learning [3].

The main objective of the paper is to define an approach for the integrated modelling of web applications and cloud services and infrastructures. These models serve to the software engineer to pass from a specification of the availability requirements of the software infrastructure to an allocation of these services onto a physical infrastructure and to an evaluation of the achieved availability level.

To achieve this objective, the paper provides an automatable methodology whose core is the usage of a Bayesian network (BN) model which drives the optimization, the search for availability bottlenecks and the overall evaluation of availability at both levels (cloud infrastructure and SOA-based application).

To make the approach usable, this paper also uses model-driven engineering. This technique has been described having not fully realised its great potential [4]: however, it still remains a valid tool specifically when used to generate formal models. The paper provides an automatable methodology whose first step requires the definition of a high-level description of cloud services, underlying software components, as well as the description of user level availability requirements. Moreover, a description of the application orchestrating the services is also modelled. A Bayesian network (BN) model is generated starting from this composed high-level model.

This work constitutes the extension of another published paper [5]. The original contribution added in this paper is constituted by a deeper description of the mechanisms by which BNs are used for analysis, optimization and learning. From a high-level point of view, the methodology has been extended by dealing with the description of the application that orchestrates cloud services. Some elements of the availability modelling methodology for SOA-based services are taken from other published resources [6] but they are here completely re-adapted and re-framed to the chosen modelling languages.

This paper is framed in SPECS, an ongoing EU research project of the FP7-ICT programme [7]. The aim of SPECS is to respond to the cloud community need of improving comprehensibly the state-of-the-art in cloud computing security by creating, promoting and exploiting a user-centric framework and a platform dedicated to offer security-as-a-service using an SLA-based approach. In the SPECS project, a typical SLA life cycle can be characterised by three main phases that are cyclically conducted: negotiation, monitoring and enforcement. While this work focuses on availability that is only one of the security-oriented features, the proposed approach is general enough to be extended in the future to other security concerns.

The rest of the paper is structured as follows. Section 2 provides both an overview of the related research as well as short background on needed information. Section 3 depicts the approach at a glance and shows the details of the different models. Section 4 describes the structure of the BN model at the centre of the approach while Sect. 5 focuses on the optimization matters by which highly available configurations are found and on how the BN model impacts on this phase. Section 6 ends the paper showing future research developments.

2 Background and related works

This section gives some basic information needed to understand what is introduced in this paper: Bayesian networks, model-driven engineering and service-oriented paradigm. Moreover, a review of the literature is given to highlight similar works. First, an overview of the SPECS project objectives and scope is provided.

2.1 The SPECS project

In the context of the SPECS project, a typical SLA life cycle can be characterised by three main phases that are cyclically conducted (see Fig. 1): negotiation, monitoring and enforcement.

In the negotiation phase, the SLA is not fully defined, and the customer(s) and provider(s) conduct a negotiation process on requirements/services to find agreement on what the SLA should effectively offer. The customers evaluate the trade-off across service specifications (base services and options), and the corresponding performance and costs. Service providers, instead, have to evaluate the services requested, matching them to what can be actually granted; moreover, they also need to evaluate the risks related to incorrect evaluation. During the monitoring phase, a signed SLA is checked for its actual degree of conformance or for penalties if in violation. Note that at the state-of-the-art, no systematic

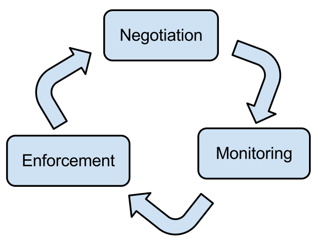


Fig. 1 SLA life cycle

SLA-monitoring solutions exist to support users. This is typically a cumbersome manual process. From the service provider view, monitoring implies two different activities: (a) verifying that the SLAs are respected via access to the underlying infrastructure that is inaccessible to end users, and (b) generating alerts before SLAs are broken, to activate remedial actions. The final step of the SLA life cycle is the enforcement, where the actions needed to respect the SLA (i.e., to keep a sustained level of security) are effectively taken. This may imply the activation of software modules, the acquisition of resources (in the correct amount), but possibly even the dynamic reconfiguration of resources after an alert is generated.

The three phases are correlated: the negotiation cannot be performed without taking in consideration how SLA can be granted, i.e., how the enforcement will take place. Enforcement needs monitoring to evaluate the real state of the solution before applying the predefined policies and procedures, while monitoring needs the results of negotiation to know what to monitor and which alerts should be generated.

2.2 Bayesian networks

Bayesian networks (BNs) [8] provide a graphical representation of a joint probability distribution over a set of random variables with a possible mutual causal relationship: the network organised as a directed acyclic graph (DAG) whose nodes represent random variables and arcs represent causal influences between pair of nodes. A conditional probability distribution is defined for each node in the network: in the common case of discrete random variables, the conditional probability function is often represented by a table (conditional probability table, CPT). Bayesian networks have been extensively included in a lot of scientific works in the field of system dependability and risk prediction [9] as well as network security [10].

Let X and Y be two BN nodes where X is an ancestor of Y, i.e., there is a path from X to Y. Three different indexes can be in general computed by solving the BN model [8]:

- *Prior probability* $\Pr(Y = y)$: the probability that Y has the value y in absence of any observations;
- *Posterior probability* $\Pr(Y = y|X = x)$: the probability that Y has the value y when the value x is observed for the variable X;
- *Likelihood* $\Pr(X = x|Y = y)$: the probability that X has the value x when the value y is observed for the variable Y.

2.3 Model-driven engineering

The unified modelling language (UML) is a well-known general purpose standardised modelling language for software system specification. UML is equipped with a profiling mechanism that allows to customise UML for a particular domain or platform. The UML profiling is actually a lightweight meta-modelling technique to extend UML: in such a way, the standard semantics of UML model elements could be refined in a strictly additive manner. Stereotypes, tags and object constrain language (OCL) constraints are the extension mechanisms used to define a UML profile. In particular, a stereotype extends one or more UML meta-classes and can be applied to those UML model elements that are instantiations of the extended meta-classes.

The UML profile for modelling and analysis of real-time and embedded systems (MARTE) is an OMG-standard profile that customises UML for the modelling and analysis of non-functional properties (NFP) of real-time embedded systems, such as timing or performance-related properties. MARTE provides a general analysis framework called general quantitative analysis model (GQAM) sub-profile. A key feature of MARTE is the NFP framework used to define new NFP data-types that are necessary to the definition of a specific analysis domain sub-profile. The dependability analysis and modelling (DAM) [11] profile is a specialisation of MARTE-GQAM to enable dependability analysis. DAM is conceived by following a systematic approach, which consists of two main steps: firstly, a dependability domain model is built; later, the model is mapped onto UML extensions (i.e., stereotypes, tags and OCL constraints). MARTE-DAM has also been extended towards the maintainability and proved its capability to apply in effective model-driven processes [12,13].

The automation of the workflow underlying these approaches is based on the development and the usage of proper model transformations which are key techniques in model-driven engineering (MDE) [14]. A transformational approach generally requires the chaining of model-to-model (M2M) and model-to-text (M2T) transformations.

2.4 Service-oriented paradigm

The service-oriented computing paradigm provides a methodological foundation for the design of complex distributed applications by integrating existing components, namely services. A service is accessible independently from its implementation details and is designed to be reused. Services are commonly implemented by web service technology that allows services to be discovered, integrated and used on the Internet, by running them on a server (that can run simultaneously more web services). Services are meant to be integrated, knowing their interfaces, in business processes (BP) or workflows (WF). Among the several techniques available for such integration, BPEL (BP execution language) [15] is the most widespread solution and took the role of standard language for services orchestration. Orchestration is one of the two main strategies for service integration and is based on the idea that an application consists of a centrally specified BP or WF that operates all services involved and is run by a specific executor. Other approaches raise the level of abstraction of BPEL preferring a modelling approach rather than a programming oriented one: these approaches contain business process modelling notation (BPMN), UML, software and systems process engineering metamodel (SPEM), etc.

2.5 Related works

The problem of defining a proper and effective allocation of VMs onto physical machines (PMs) has already been studied in the literature, and several results are available [16–18], also in the context of funded/open source research projects as mOSAIC[2] and Eucalyptus.[3]

Some works use energy to guide the allocation of VMs by pursuing energy efficiency [19,20]. The work of Dougherty et al., while oriented to energy efficiency, also describes a model-driven approach, but it does not use standard modelling languages such as UML [21]. The dependability of services provided in the cloud has received recent attention by the scientific community: Huang et al. propose a method based on reliability evaluation by means of series–parallel composition [22] while Yanagisawa et al. deal with this problem by defining a mixed integer programming model [23]. Frincu and Craciun propose a multi-objective optimisation approach for high-availability and cost-effective VMs allocation [24].

The work of Zambon et al. presents many similarities to the approach here proposed [25]; the main differences are: (1) the approach proposed here tries to overcome the limitations of the formalisms used in [25] (i.e., fault trees and reliability block diagrams); (2) the perspective of the approach here proposed is not limited to planning activities but also to monitoring and enforcement actions; (3) the proposed approach is user oriented. While other approaches focus on traditional straight-forward software development [26], in this work, model-driven principles and techniques are chosen to raise the level of abstraction and to improve the participation of the stakeholders in all phases of the cloud service life cycle.

Furthermore, two other works present some interesting features applicable to this paper. De Florio introduces some fault tolerance patterns (e.g. redoing, diversity and redundancy) for software-based systems that can be used as basic mechanisms to explore the design space of possible allocations of cloud services to VMs [27]. From the same work, the concept of distance-to-failure (dtof) can be improved in this paper by calculating it on the probability of failure (i.e., the residual probability of having a failure considered the observed faults). Another work is applicable to this paper; it focuses on some mechanisms able to guarantee software reliability through data integrity both at the design-time and during all the run-time phases of a software system [28].

3 The methodological approach

Figure 2 depicts the approach at a glance. There are two stages: the specification of an allocation problem and the allocation itself. The model is instead specified by means of three layers: application, service and architecture.

The application contains the specification of the workflow of activities and external services that are invoked to provide complex tasks. All the typical constructs of a programming/workflow specification language may be used: variables, control structures (choice, loops, etc.). Moreover, according to the principles of the SOA approach, complex applications rely on the invocation of external services which are accessed over the Internet. At the second layer, there is the specification of the services which are to allocate achieving high-availability levels. Some services may depend on others. At the third stage, a high-level architecture is given containing the software infrastructure and components needed to provide each specified service: in this view, the components needed to implement a service and their inter-dependencies are highlighted. To the allocation stage also belongs another layer: the infrastructure layer which is responsible for mapping the software components described in the architecture layer to physical/virtual machines hosting the components.

The passage between the specification and the allocation stages is realised using a formal model which guides the process of determination of the values of the many parameters

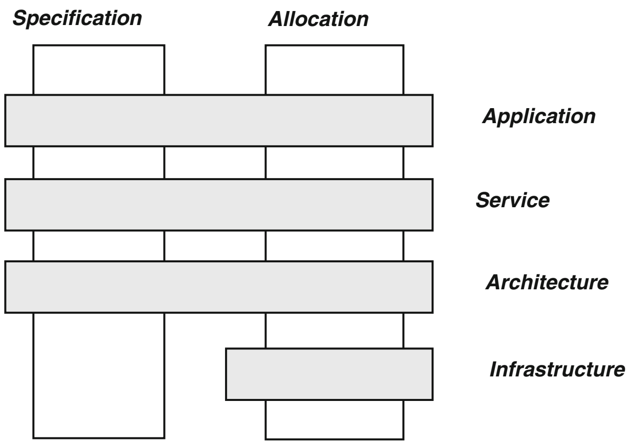


Fig. 2 Overview of the approach

[2] http://www.mosaic-cloud.eu/.

[3] https://www.eucalyptus.com/.

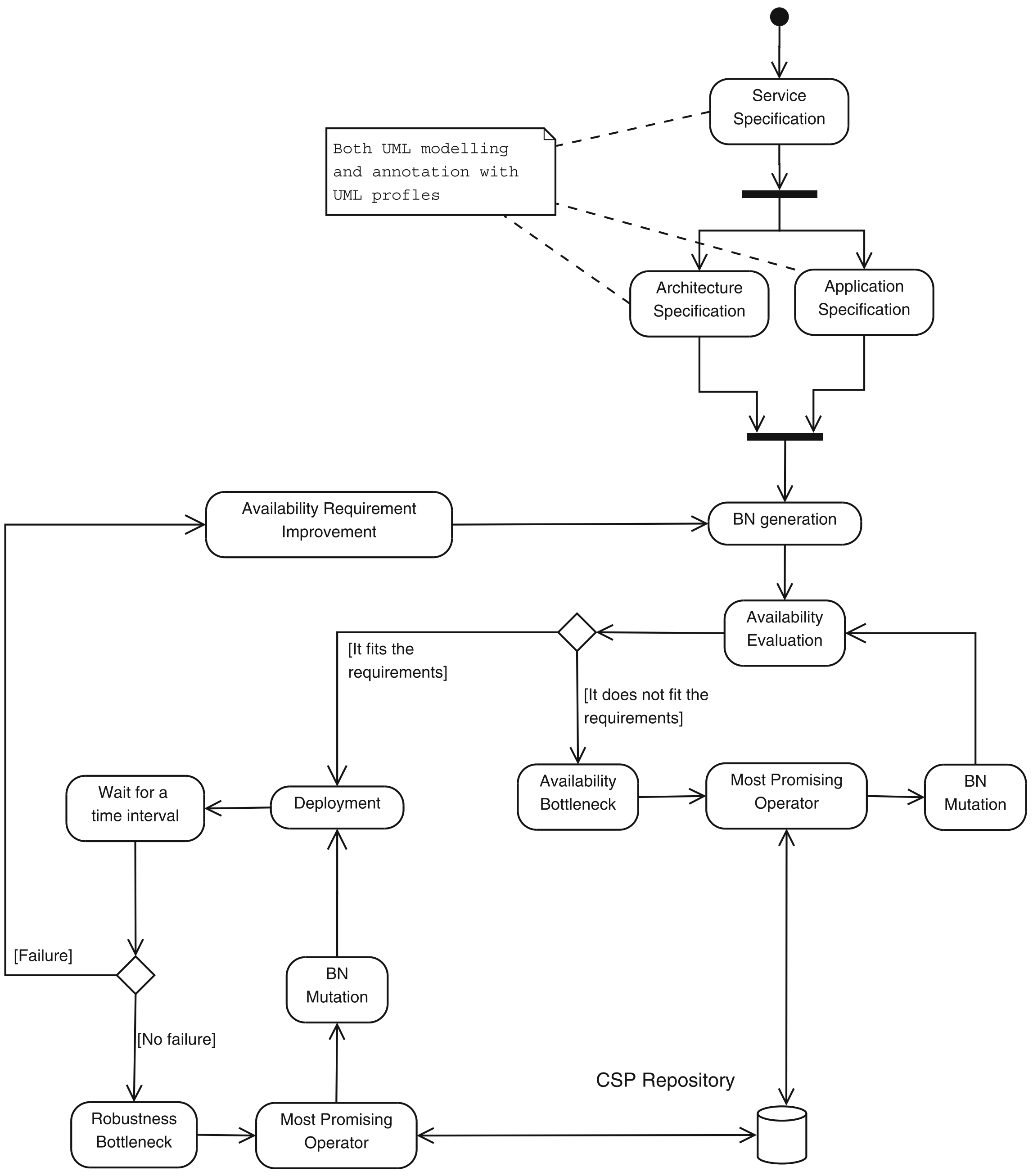


Fig. 3 Workflow of the process activities

involved in the specification phase. This choice is made to maximise the expected availability of the sensitive items and to minimise the overall cost.

Figure 3 depicts the detailed workflow of the activities expressed as a UML activity diagram.

The first phases of the workflow are related to the creation of the high-level model (application, service and architecture views). The order of specification of these views is the following: the service layer should be specified first, and then the architecture layer goes deeper into the structure of each service while the application composes specified services. These two last activities may be done in parallel. Please note that by annotating the model with proper UML structures (UML profiles, stereotypes and tagged values), it is possible to capture the model itself, the availability requirements and the sensitive parameters, values of which are subject of optimization.

Now, the passage between the specification and the allocation phases is developed as follows: first a model transformation is in charge of generating a BN model from the specification model. Once a first BN model has been generated, the workflow enters a loop: first the model is evaluated (availability evaluation) by calculating the availability metric; then, this value is compared with the availability requirement asked by the user. If the achieved availability is higher than the requirement, the candidate BN model is chosen for deployment: in this phase, a high-level model is returned to the final user explaining all the details of the found solution. If the requirement is not fulfilled, the model is then analysed trying to understand which is the most sensitive availability parameter (availability bottleneck), then a mutation operator is chosen (most promising operator) and then a new model is generated changing something in the model to improve it (BN mutation). To optimise allocation, further data are needed: the CSP repository database contains data about known CPSs, the availability they offer and the related costs.

Once the model has been deployed and then the design of the system has been completed, the workflow continues with an infinite loop aimed at continuously improving the system by raising the level of the robustness. By collecting data on the functioning of the system, the workflow can filter these data and analyse the model by simulating most probable failures or operational workload. This optimization phase is devoted to the improvement of the robustness and, hence, probable weaknesses under robustness aspect are evaluated (robustness bottleneck). Then, the operator that could bring the greatest improvement to the model is detected (most promising operator) and applied, and hence the new model is generated (BN mutation). This notwithstanding, following a failure, the availability requirements may change and then the first loop is re-entered.

It is important to underline that while the first part of the approach is typical of model-driven/model-based design and optimisation approaches, the second (infinite) loop of the methodology is oriented to a non-ending process which allows the system improve its robustness by learning from experience. In other words, this is an “antifragile workflow” since continuous robustness improvement is possible by exploiting experience learnt during the operational life cycle phase [29,30].

3.1 The UML model

Since the main focus of the paper is not on the UML modelling topics, few details are reserved for the construction of the UML model. This model has different views according to the different aspects it wants to capture: in fact, it has to describe the different layers of the specification stage. Each layer is well described by a UML diagram.

Figure 4 gives an overview of such models. Figure 4a shows the service layer where all the services, including the final application, are specified. One of the most natural way to model it in UML is using a UML use case diagram (UCD) where the application service «`include`»s or «`use`»s other services. One or more services may be further specified by linking to the related use case a UML activity diagram (AD) which can be used to specify the application: using choice, merge, fork and join nodes, complex applications can be specified (see Fig. 4b). Figure 4c finally depicts how services can be implemented by hierarchies of software components running on virtual machines.

Moreover, to capture the description of all the parameters, the UML model can be annotated using a proper UML profile. According to our goal to capture dependability and availability requirements as well as some performance indices, MARTE and MARTE-DAM profiles will be mainly used. These additional elements also serve to link the model elements among them.

Tables 1, 2 and 3 highlight the stereotypes and the tagged values used, respectively, for service, application and architecture layers.

An important modelling rule is constituted by the usage of the notation of the value specification language (VSL) of MARTE and MARTE-DAM. Using this notation, the modeller can specify important model variables also differentiating which are requirements, constants or parameters of the model. Variables in general are indicated by prefixing its name by the $ symbol. Furthermore, indicating that the value of a variable is a requirement is possible by specifying `source = req` (see the availability of `A` service in Fig. 4 as an example). The second category, the constants, are specified by the clause `source = pred` (an example is constituted by the availability of `C1` and `C2` components). The last category, the model parameters, is identified by non-specifying the kind of the `source` tagged value: two examples are constituted by the *rep* value of the `A1` step of by the *resMult* tagged value of the `C1` component.

4 The BN model

This Section shows the structure and the semantics of the BN model. Since the great number of the elements to translate and the complexity of the original model, the BN model should be structured in blocks. Hence, we have a sort of macro-architecture of the BN model and a micro-architecture: the first organises the different BN submodels that are related to the different layers while the second is in charge of translating single model elements.

Figure 5 depicts the general architecture of the model proposing a layered architecture where the nodes of one layer may depend only on the nodes of the same layer or on the nodes of the previous layer.

In the following we present the BN micro-architectures of the single layers. It is important to underline that the BN

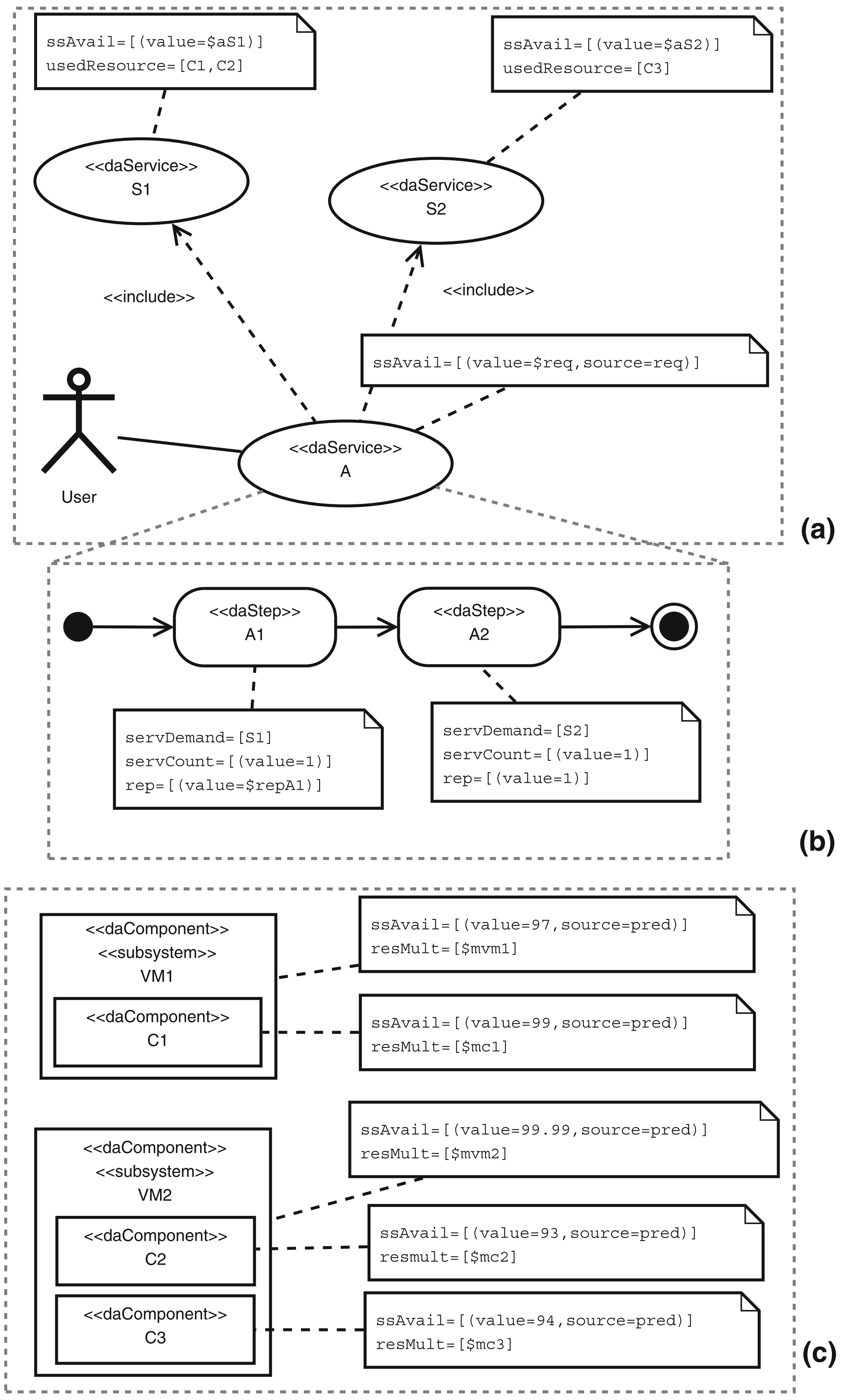


Fig. 4 General structure of the UML specification model: **a** application, **b** service and **c** architecture

Table 1 MARTE/MARTE-DAM stereotypes and tags for the service layer

Stereotype	Tags: type	Meanings
daService	ssAvail: NFP_Percentage[*]	Steady-state availability of the service
	usedResource: Resource[*]	Resources used by the service (inherited from MARTE). This list contains the components which implement the service

Table 2 MARTE/MARTE-DAM stereotypes and tags for the application layer

Stereotype	Tags: type	Meanings
daStep	servDemand: GaRequestedService [*]	A set of operations requested by the step, such as calls to interface operations (inherited from MARTE). This tagged value needs to specify the list of the service called by the step
	servCount: NFP_Real[*]	A set of values for the number of requests to the operations given in servDemand (inherited from MARTE)
	rep: Integer	The actual number of repetitions of an operation or loop (inherited from MARTE)

Table 3 MARTE/MARTE-DAM stereotypes and tags for architecture layer

Stereotype	Tags: type	Meanings
daComponent	ssAvail: NFP_Percentage[*]	Steady-state availability of the service
	resMult: Integer	Multiplicity of the resource (inherited from MARTE): number of replicas of the component

model is constituted by binary BN nodes having `ok` and `ko` values: `ok` means that the node (representing a service or a component) is running while `ko` means that the node has failed.

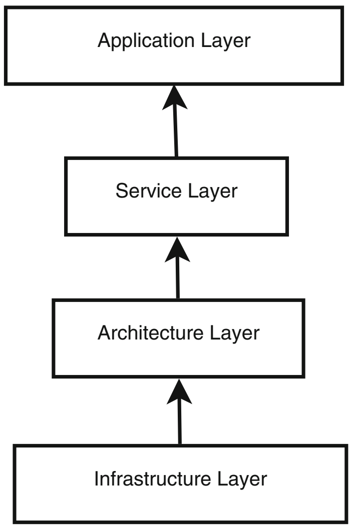


Fig. 5 Macro-architecture of the BN model

4.1 Application micro-architecture

The application micro-architecture is in charge of translating the different workflow constructs to obtain the structural availability of the SOA-based application. This mapping is inspired by the work in [6] but several differences are present: (1) the previous work starts from BPEL program rather than UML AD, (2) the work here is framed in a wider context where application, service and hw/sw architecture are optimised in a holistic way, (3) while the translation in [6] generates a Fault Tree (FT) model, this work generates BN models which have a greater modelling power and allows to model some situations not allowed in FTs (complex failure logics, multivalued variables, common mode of failures, etc.). In the following, some elementary constructs of the UML Activity Diagrams are mapped aiming at illustrating the potentiality of the approach:

- *Sequence* when two «`daStep`»s are in a sequence, the availability of the entire process is determined by the OR of the failing events of the single activities. If at least one activity fails, the entire sequence fails (Fig. 6a);
- *Fork* when two «`daStep`»s are executed in parallel, it is expected that both of them are needed to accomplish the whole process. Hence, the availability of the entire process is determined by the OR of the failing events of the single activities (Fig. 6b);
- *Call* when a «`daStep`» calls a service (as indicated in the *servDemand* tagged value), its availability depends on how many attempts are made. If the value of the *rep* tagged value is *n* (i.e., there are *n* attempts of calling a service), we can model this situation with *n* BN nodes which are the parents of the *Call* node. Each of these nodes is a replica of the node representing the

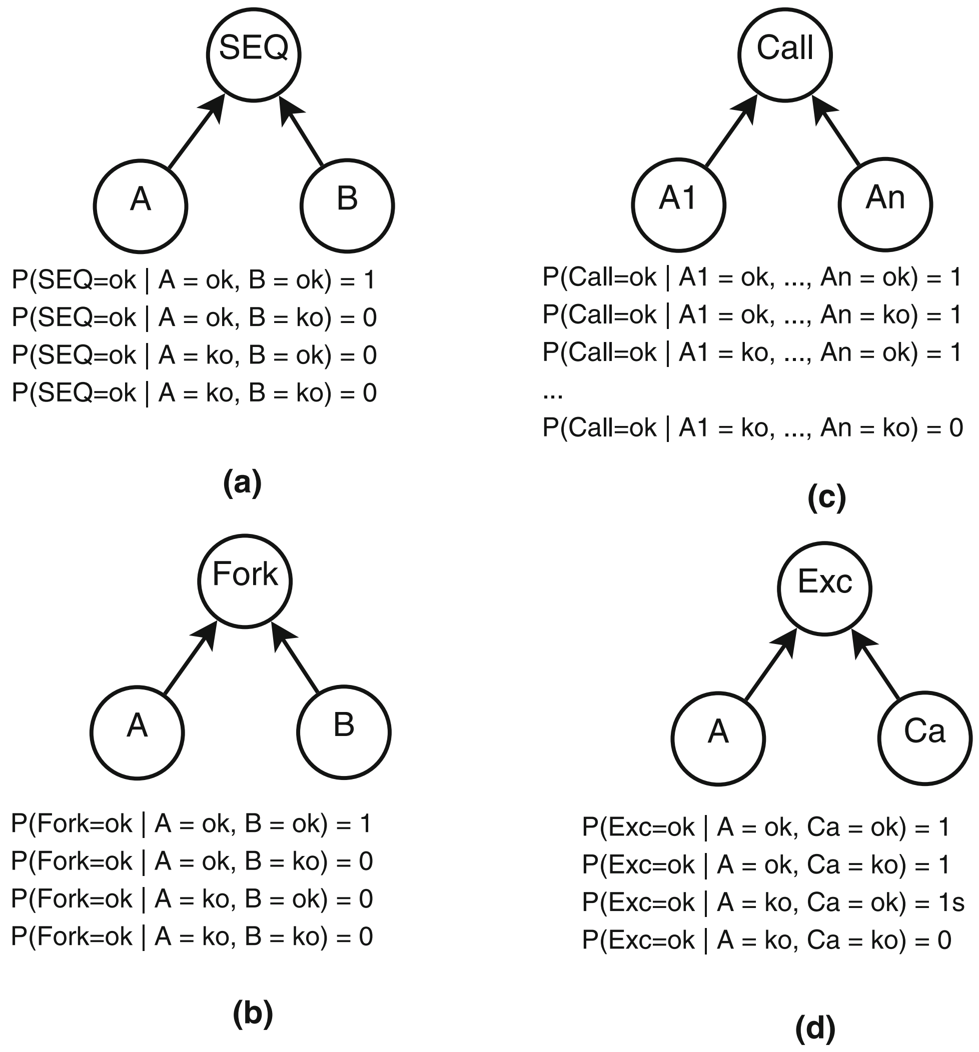

SEQ
A
B
P(SEQ=ok | A = ok, B = ok) = 1
P(SEQ=ok | A = ok, B = ko) = 0
P(SEQ=ok | A = ko, B = ok) = 0
P(SEQ=ok | A = ko, B = ko) = 0
(a)
Call
A1
An
P(Call=ok | A1 = ok, ..., An = ok) = 1
P(Call=ok | A1 = ok, ..., An = ko) = 1
P(Call=ok | A1 = ko, ..., An = ok) = 1
...
P(Call=ok | A1 = ko, ..., An = ko) = 0
(c)
Fork
A
B
P(Fork=ok | A = ok, B = ok) = 1
P(Fork=ok | A = ok, B = ko) = 0
P(Fork=ok | A = ko, B = ok) = 0
P(Fork=ok | A = ko, B = ko) = 0
(b)
Exc
A
Ca
P(Exc=ok | A = ok, Ca = ok) = 1
P(Exc=ok | A = ok, Ca = ko) = 1
P(Exc=ok | A = ko, Ca = ok) = 1s
P(Exc=ok | A = ko, Ca = ko) = 0
(d)

Fig. 6 Micro-architecture of the BN application submodel

called service. Hence, the failure of the entire call is the AND of the failures of all the service invocations (Fig. 6c);

- *Exception Handling* an activity A may raise an exception and another activity (Ca) can catch it. In this situation, only when both the activities have failed, the exception handling activity fails (Fig. 6d).

By combining these mappings, it is possible to create a complex BN model from a workflow.

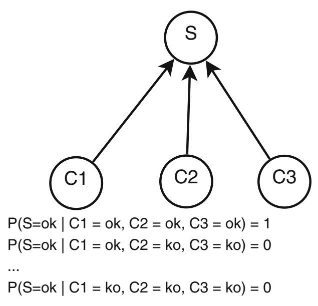

S
C1
C2
C3
P(S=ok | C1 = ok, C2 = ok, C3 = ok) = 1
P(S=ok | C1 = ok, C2 = ko, C3 = ko) = 0
...
P(S=ok | C1 = ko, C2 = ko, C3 = ko) = 0

Fig. 7 Micro-architecture of the BN service submodel

4.2 Service micro-architecture

The micro-architecture of the Service BN submodel is simpler since only the case of the service invocation is considered. Figure 7 depicts the case when a service *S* is implemented by three components *C1*, *C2* and *C3*. In this case, only when all the components are up, the service is up.

4.3 Architecture micro-architecture

The micro-architecture of the Architecture BN submodel mainly deals with the levels of hierarchy and redundancy of the components. Two patterns are described in Fig. 8. Figure 8a shows when a component C1 (with a *p1* probability of autonomous failure) includes a `C2` component. In this case when `C2` is down, `C1` is ko; otherwise, it has a *p1*

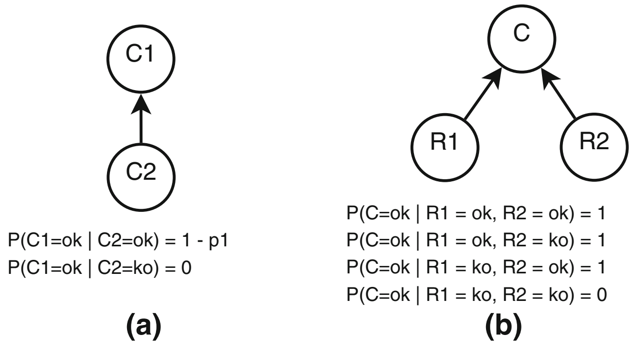

C1
C2
P(C1=ok | C2=ok) = 1 - p1
P(C1=ok | C2=ko) = 0
(a)
C
R1
R2
P(C=ok | R1 = ok, R2 = ok) = 1
P(C=ok | R1 = ok, R2 = ko) = 1
P(C=ok | R1 = ko, R2 = ok) = 1
P(C=ok | R1 = ko, R2 = ko) = 0
(b)

Fig. 8 Micro-architecture of the BN architecture submodel

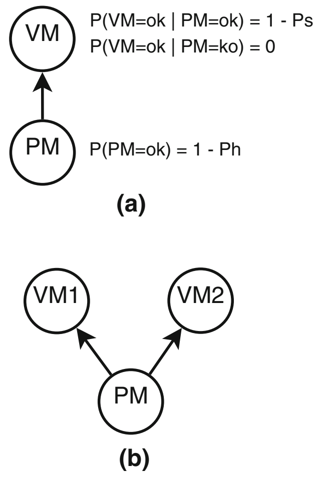


Fig. 9 Micro-architecture of the BN infrastructure submodel

probability of failure. The second case is when a component has more than one replica: in this case, all the replicas are running to improve the fault tolerance: the component fails when all the replicas are down. The example in Fig. 8b is related to a component C that has two replicas R1 and R2.

4.4 Infrastructure micro-architecture

The infrastructure micro-architecture has the role of adding physical hosting infrastructure to the model to understand how the service, the components and the virtual machines are allocated to a physical infrastructure also using the *CSP Repository*. Two BN fragments are shown in Fig. 9. More specifically, Fig. 9a shows the hosting of a virtual machine VM (with *Ps* probability of autonomous failures) by a physical machine PM (with *Ph* probability of failure): the schema is similar to the one presented in Fig. 8a. A refinement of this schema is in Fig. 9b where two VMs are hosted by a single PM: in this case the PM node becomes a common mode of failure for both virtual machines.

For the sake of clarity, here we report the BN model following the example first depicted in Fig. 4. This BN model is illustrated in Fig. 10.

5 Optimization

This Section gives some details on the optimisation phase: both the optimisation of the availability and of the robustness are characterised by the same "meta-schema". For the sake of clarity, we illustrate these techniques on the BN model in Fig. 10. According to the workflow in Fig. 3, the optimisation activities are structured as in the Algorithm 1.[4]

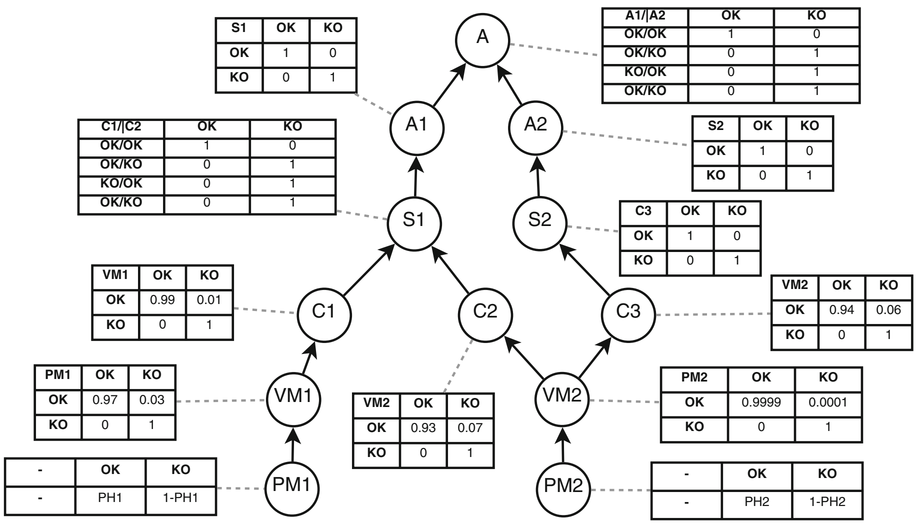
Algorithm 1 Optimization pseudo-code (availability)

```
Require: $bn \in \mathcal{BN}$
Require: $req \in \Re$
Require: $KB$
Ensure: $bn \in \mathcal{BN}$
  $N^* \leftarrow nodes(bn) - apps(bn)$
  $avail \leftarrow prior(bn)$
  while $(avail < req) \wedge (freevm(bn) = \emptyset)$ do
    $i \leftarrow 0$
    for all $X \in N^*$ do
      $cause_i \leftarrow likelihood(X)$
      $i++$
    end for
    $i_{max} \leftarrow i \mid cause_i = max(cause)$
    $X_{max} \leftarrow N^*_i$
    $bmo \leftarrow getBMO(KB, cause)$
    $bn \leftarrow bmo(cause, bn)$
    $bn \leftarrow purge(bn)$
    $N^* \leftarrow nodes(bn) - apps(bn)$
    $avail \leftarrow prior(bn)$
  end while
```

First, let us define some functions and symbols used in this algorithm:

- $\mathcal{BN}$ is the set of all the possible BN models;
- $bn \in \mathcal{BN} \mid bn = (N, E)$ is a Bayesian Network model;
- $\mathcal{P}(N)$ is the set of all the subsets of N;
- $parent : N \longrightarrow \mathcal{P}(N)$ is the function that returns the parent nodes of a given BN node, $parent(X) = \{Y \in N \mid (Y, X) \in E\}$;
- *label* is a function which tags each node of a BN model according to its nature, i.e., the element of the high-level model that generates the node: $label : N \longrightarrow \{app, step, comp, serv, vm, pm\}$. These labels represent the nodes, respectively, related to: «daService» associated to an Activity Diagram, «daStep», «daComponent», «daService», «daComponent» also tagged with the UML's stereotype «subsystem», physical machines.
- *nodes* is a function that returns all the nodes of the Bayesian Network, $nodes(bn) = N \; bn \in \mathcal{BN}$;

[4] Notwithstanding the algorithm refers to the availability optimization, the robustness optimisation follows a very similar schema with only few differences.



S1	OK	KO
OK	1	0
KO	0	1

A1/|A2	OK	KO
OK/OK	1	0
OK/KO	0	1
KO/OK	0	1
OK/KO	0	1

C1/|C2	OK	KO
OK/OK	1	0
OK/KO	0	1
KO/OK	0	1
OK/KO	0	1

S2	OK	KO
OK	1	0
KO	0	1

C3	OK	KO
OK	1	0
KO	0	1

VM1	OK	KO
OK	0.99	0.01
KO	0	1

VM2	OK	KO
OK	0.94	0.06
KO	0	1

PM1	OK	KO
OK	0.97	0.03
KO	0	1

VM2	OK	KO
OK	0.93	0.07
KO	0	1

PM2	OK	KO
OK	0.9999	0.0001
KO	0	1

-	OK	KO
-	PH1	1-PH1

-	OK	KO
-	PH2	1-PH2

Fig. 10 BN model of the example

- *apps* is a function that returns all the nodes representing the applications of the BN model, $apps(bn) = \{M \in N \mid label(M) = app\}\, bn \in \mathcal{BN}$[5];
- $prior : \mathcal{BN} \longrightarrow \Re$ is so that $prior(bn) = Pr(apps(bn) = ok)$. This function executes the prior analysis on a BN model;
- $likelihood : \mathcal{BN} \times N \longrightarrow \Re$ is so that $likelihood(bn, X) = Pr(X = ko \mid apps(bn) = ko)$. This function executes the likelihood analysis on the model;
- $freevm : \mathcal{BN} \longrightarrow \mathcal{P}(N)$ returns the nodes representing the non allocated virtual machines: $freevm(bn) = \{M \in N | label(M) = vm \wedge \nexists (G, M) \in E\ with\, label(G) = pm\}, bn \in \mathcal{BN}$;
- $purge : \mathcal{BN} \longrightarrow \mathcal{BN}$ is a function that deletes all the non-connected nodes, i.e. all the nodes that appears neither as source nor destination of any arc.

Let us now analyse in detail this pseudo-program. The first lines are devoted to detect the availability/robustness bottlenecks of the system. This is possible using the likelihood analysis of BNs. By querying the model in this way, we can ask the following question: "given a failure of the system, which is the most probable cause?". Hence, by supposing that the application fails (`A = ko`), this query evaluates:

$$Pr(X = ko \mid A = ko)\ \ \forall X \in N - \{A\}$$

This represents the likelihood function as we said in Sect. 2.2. Once the most probable cause has been detected, the nature of this cause determines which is the next step to improve the model. First, let us enumerate a list of possible *BN mutation operators* (BMO). A BMO is a function which mutates an input BN model generating a new BN model:

$$\text{BMO} : \mathcal{BN} \longrightarrow \mathcal{BN}$$

The set of all the BMOs counts four types of operators:

- *insertion* INS: this operator extracts the information about a CSP from the CSP Repository that is not already represented in the BN model. Then, the operator allocates a VM (represented by the X node) to this CSP by creating a new BN node (PM) and an arc from PM to X;
- *allocation* ALL: this operator does not create a new node for a physical machine but simply creates an arc from an existing node labelled as *pm* to a node in $freevm(bn)$. In other words, it allocates an unallocated VM to an existing PM;
- *migration* MIG: this operator changes the allocation of a VM onto the physical infrastructure. Practically, it deletes the arc from a node PM1 to a node VM and creates another from PM2 to VM;
- *redundancy*: the ratio of this operator is to duplicate a resource and/or the execution of an action to improve the level of fault tolerance. It can be applied in two contexts:
- *component* RED_A: a component C and all the included components are duplicated;
- *step* RED_B: an action of an application is repeated more than one time;

The choice of the most proper BMO to apply is done according to the value of *label* function applied to the most probable failure cause node (*cause* in the algorithm). The $getBMO$ function is in charge of accomplishing such task: it uses the *cause* node of the BN model and the KB knowledge base in which there is the algorithm for choosing the right rule; $getBMO$ returns the operator to use. Different

[5] We suppose just for simplicity that there is only one "app" node for the BN.

KBs may affect the performance and the quality of the final result of the optimisation algorithm. We propose here some `if-then` rules that may constitute the core of such a KB:

- **Rule1**: $freevm(bn) \neq \emptyset \Longrightarrow bmo = \{INS, ALL\}$;
- **Rule2**: $(freevm(bn) = \emptyset) \wedge (label(cause) = step) \Longrightarrow bmo = \{RED_B\}$;
- **Rule3**: $(freevm(bn) = \emptyset) \wedge (label(cause) \in \{comp, vm\}) \Longrightarrow bmo = RED_A$;
- **Rule4**: $(freevm(bn) = \emptyset) \wedge (label(cause) = serv) \Longrightarrow (bmo = RED_A) \wedge (cause \leftarrow parents(cause))$;
- **Rule5**: $(freevm(bn) = \emptyset) \wedge (label(cause) = pm) \Longrightarrow bmo = MIG$.

Please note that some kind of non-determinism is present in the KB. As an example, the first rule allows to use the insertion or the allocation operators. The choice can be determined by further usage policy of the CSP Repository: one can prefer first to use all the CSPs in the DB (preferring the INS operator) or first to saturate a CSP to its maximum capacity (preferring the ALL operator). The choice of which form the KB has to assume is subject to different factors. In this paper, for the sake of simplicity, we chose the simple structure of an `if-then` rule set. This notwithstanding, a BN model can be used for the KB. The advantages of using such a formalism stay in the possibility to have (a) a probabilistic rule set capable of expressing which is the degree of belief associated with a selected rule (b) using traditional and assessed BN learning algorithms to improve and evolve the KB during the operation of the system.

As already said, the application of the BMO functions allows to pass from a BN model to a new BN model. Starting from the BN model in Fig. 10, the exemplary application of all these rules are depicted in the following.

Figure 11 depicts the case of the application of Rule1.

Figure 12 shows the case of the application of Rule2 and Rule4 since, in both cases, the effect is the duplication of a software component (`C1` in this case).

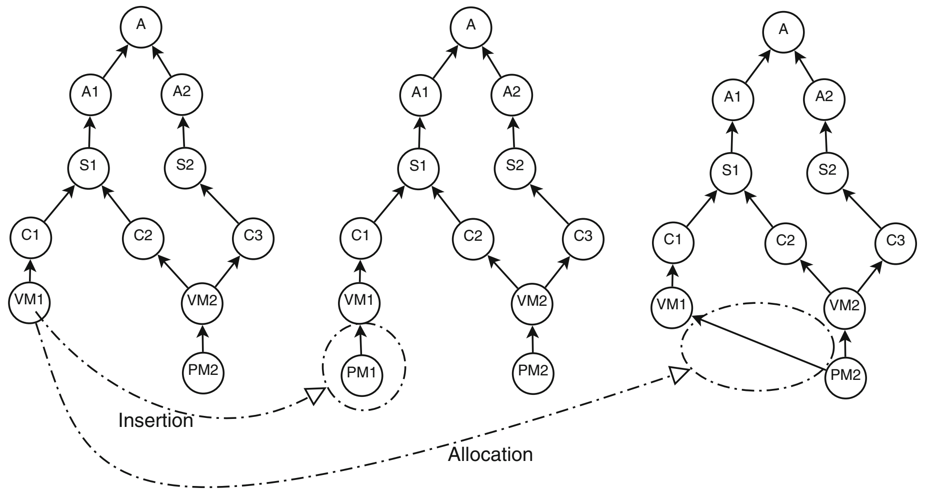


Fig. 11 Application of Rule1

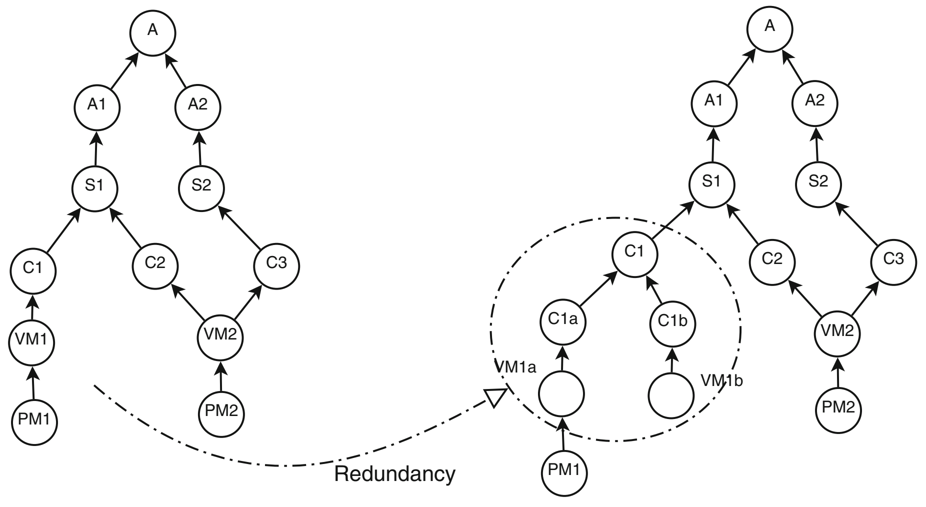


Fig. 12 Application of Rule2/Rule4

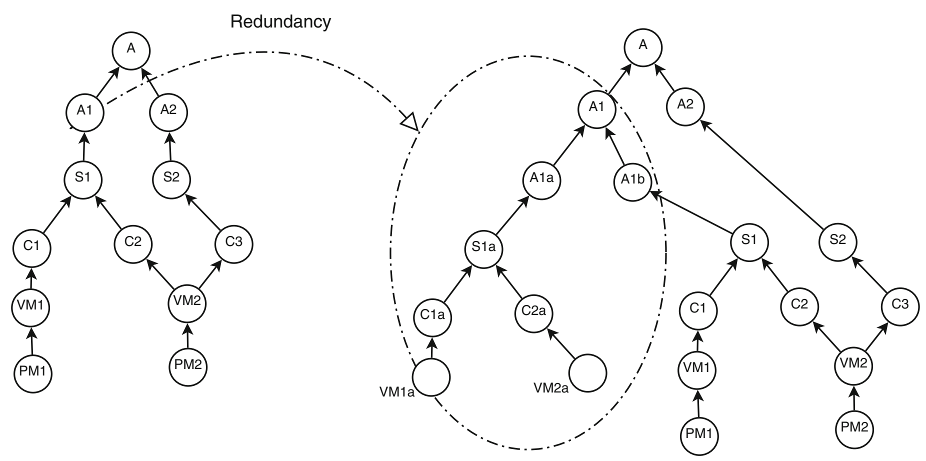


Fig. 13 Application of Rule3

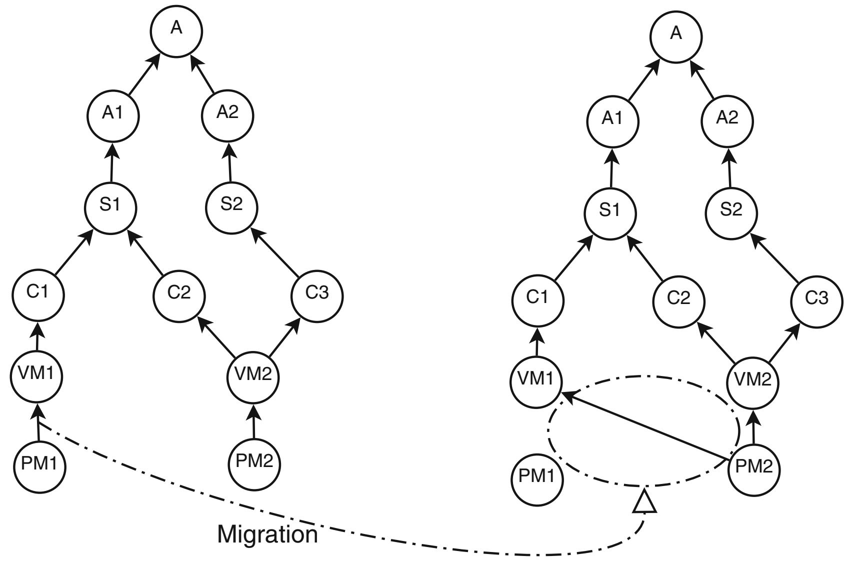


Fig. 14 Application of Rule5

Figure 13 illustrates when the redundancy is applied to an application step (application of Rule3). In this case, it is possible to see how the duplication of the nodes is propagated from upper to lower levels.

Figure 14 reports the case of the application of Rule5 when one virtual machine, `VM1` in this case, migrates from `PM1` to `PM2`.

After created a new BN model, some refinements can be possible. An example is constituted by eliminating all the non-connected nodes, i.e., nodes that are neither parents nor children of any other node. This operation can be necessary after the application of a migration BMO (see Fig. 14).

6 Conclusions and future works

This paper has shown how holistically to improve the availability and the robustness of cloud-based web application. The category of the information systems considered counts web applications, provided services, software architectures, guested virtual machines as well as hosting physical machines. The approach, which starts from annotated UML models, is based on the use of a Bayesian Network model as the founding brick in an optimisation algorithm. By modelling and mapping all the high-level elements into BN nodes and their relationships, the different analyses that are possible to make on the BN model, can address the search of optimal allocation and replication of the resources.

The paper is strongly centred on the notion of antifragility and its related themes. According to the formula “Antifragility = Elasticity + Resilience + Machine Learning”, this paper tackles with all of these topics by addressing a flexible evolving and experience-learning approach.

The limitations of this work are that it catches the theoretical aspects of the approach but does not discuss a real implementation of the optimisation engine notwithstanding the optimisation logic is expressed in pseudo-code. Further research efforts will be devoted to: (1) implement and assess the approach, (2) explore the usage of the BN formalism to express the optimisation KB and (3) improve the approach also considering learning techniques by which KB may evolve in time.

References

1. Dekker M, Hogben G (2011) Survey and analysis of security parameters in cloud slas across the european public sector. Tech. rep. European Network and Information Security Agency
2. Kritikos K, Pernici B, Plebani P, Cappiello C, Comuzzi M, Benrernou S, Brandic I, Kertész A, Parkin M, Carro M (2013) A survey on service quality description. ACM Comput Surv 46(1):1:1–1:58. doi:10.1145/2522968.2522969
3. Taleb N (2013) Philosophy: 'antifragility' as a mathematical idea. Nature 494(7438):430. doi:10.1038/494430e
4. Schmidt DC (2006) Guest editor's introduction: model-driven engineering. Computer 39(2):25–31. doi:10.1109/MC.2006.58
5. Marrone S, Nardone R (2015) Automatic resource allocation for high availability cloud services. Procedia Computer Sci 52:980–987. doi:10.1016/j.procs.2015.05.176 (The 6th International Conference on Ambient Systems, Networks and Technologies (ANT-2015), the 5th International Conference on Sustainable Energy Information Technology (SEIT-2015))
6. Iacono M, Marrone S (2014) Model-based availability evaluation of composed web services. J Telecommun Inf Technol 2014(4):5–13
7. Rak M, Suri N, Luna J, Petcu D, Casola V, Villano U (2013) Security as a service using an SLA-based approach via SPECS. In: IEEE 5th International Conference on Cloud Computing Technology and Science (CloudCom), vol 2, pp 1–6. doi:10.1109/CloudCom.2013.165
8. Pearl J (1988) Probabilistic reasoning in intelligent systems: networks of plausible inference. Morgan Kaufmann Publishers Inc., San Francisco
9. Weber P, Medina-Oliva G, Simon C, Iung B (2012) Overview on bayesian networks applications for dependability, risk analysis and maintenance areas. Eng Appl Artif Intell 25(4):671–682. doi:10.1016/j.engappai.2010.06.002
10. Frigault M, Wang L (2008) Measuring network security using bayesian network-based attack graphs. In: Computer Software and Applications, 2008. COMPSAC '08. 32nd Annual IEEE International, pp 698–703. doi:10.1109/COMPSAC.2008.88
11. Bernardi S, Merseguer J, Petriu DC (2011) A dependability profile within MARTE. Softw Syst Model 10(3):313–336
12. Bernardi S, Flammini F, Marrone S, Mazzocca N, Merseguer J, Nardone R, Vittorini V (2013) Enabling the usage of UML in the verification of railway systems: the DAM-rail approach. Reliab Eng Syst Saf 120:112–126
13. Bernardi S, Flammini F, Marrone S, Merseguer J, Papa C, Vittorini V (2011) Model-driven availability evaluation of railway control systems. In: Computer Safety, Reliability, and Security, LNCS, vol 6894, pp 15–28. Springer, New York
14. Bézivin J (2004) In search of a basic principle for model driven engineering. Novatica J Special Issue 5(2):21–24. http://www.dei.isep.ipp.pt/~alex/publico/mde/up5-2Bezivin.pdf
15. Alves A et al (2007) Web services business process execution language version 2.0 (OASIS Standard). WS-BPEL TC OASIS. http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html
16. Calheiros R, Ranjan R, Beloglazov A, De Rose C, Buyya R (2011) Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. Softw Pract Exp 41(1):23–50
17. Ezugwu Absalom E, Buhari SM, Junaidu SB (2013) Virtual machine allocation in cloud computing environment. Int J Cloud Appl Comput (IJCAC) 3(2):47–60
18. Xiao Z, Song W, Chen Q (2013) Dynamic resource allocation using virtual machines for cloud computing environment. IEEE Trans Parallel Distrib Syst 24(6):1107–1117
19. Beloglazov A, Abawajy J, Buyya R (2012) Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing. Future Gener Computer Syst 28(5):755–768
20. Quang-Hung N, Thoai N, Son N (2014) EPOBF: energy efficient allocation of virtual machines in high performance computing cloud. In: Transactions on Large-Scale Data and Knowledge-Centered Systems XVI, LNCS, pp 71–86
21. Dougherty B, White J, Schmidt D (2012) Model-driven autoscaling of green cloud computing infrastructure. Future Gener Computer Syst 28(2):371–378
22. Huang C, Chen J, Zhang L, Zhu Q (2013) Architecting dependable virtual computing system with considering error propagation. J Comput Inf Syst 9(4):1593–1601
23. Yanagisawa H, Osogami T, Raymond R (2013) Dependable virtual machine allocation. In: INFOCOM, 2013 Proceedings IEEE, pp 629–637. doi:10.1109/INFCOM.2013.6566848
24. Frincu ME, Craciun C (2011) Multi-objective meta-heuristics for scheduling applications with high availability requirements and cost constraints in multi-cloud environments. In: 2011 4th IEEE International Conference on Utility and Cloud Computing (UCC), pp 267–274. IEEE (2011)
25. Zambon E, Etalle S, Wieringa RJ (2012) A2thos: availability analysis and optimisation in slas. Int J Netw Manag 22(2):104–130. doi:10.1002/nem.790
26. Ardagna D, di Nitto E, Mohagheghi P, Mosser S, Ballagny C, D'Andria F, Casale G, Matthews P, Nechifor CS, Petcu D, Gercke A, Sheridan C (2012) Modaclouds: a model-driven approach for the design and execution of applications on multiple clouds. In: 2012 ICSE Workshop on Modeling in Software Engineering (MISE), pp 50–56. doi:10.1109/MISE.2012.6226014
27. De Florio V (2010) Software assumptions failure tolerance: role, strategies, and visions. In: Architecting Dependable Systems VII, LNCS, vol 6420, pp 249–272. doi:10.1007/978-3-642-17245-8_11
28. De Florio V, Blondia C (2008) On the requirements of new software development. Int J Bus Intell Data Min 3(3):330–349. doi:10.1504/IJBIDM.2008.022138
29. De Florio V (2015) On resilient behaviors in computational systems and environments. J Reliab Intell Environ 1(1):33–46. doi:10.1007/s40860-015-0002-6
30. Florio VD (2014) Antifragility = elasticity + resilience + machine learning models and algorithms for open system fidelity. Procedia Computer Science 32:834–841. doi:10.1016/j.procs.2014.05.499 (The 5th International Conference on Ambient Systems, Networks and Technologies (ANT-2014), the 4th International Conference on Sustainable Energy Information Technology (SEIT-2014))