A stochastic modeling approach for an efficient dependability evaluation of large systems with non-anonymous interconnected components

Silvano Chiaradonna
and Felicita Di Giandomenico
ISTI-CNR, Pisa, Italy
e-mail: {silvano.chiaradonna,
felicita.digiandomenico}@isti.cnr.it

Giulio Masetti
Computer Science Department
University of Pisa, Italy
and ISTI-CNR, Pisa, Italy
e-mail: giulio.masetti@isti.cnr.it

***Abstract*—This paper addresses the generation of stochastic models for dependability and performability analysis of complex systems, through automatic replication of template models. The proposed solution is tailored to systems composed by large populations of similar non-anonymous components, interconnected with each other according to a variety of topologies. A new efficient replication technique is presented and its implementation is discussed. The goal is to improve the performance of simulation solvers with respect to standard approaches, when employed in the modeling of the addressed class of systems, in particular for loosely interconnected system components (as typically encountered in the electrical or transportation sectors). Effectiveness of the new technique is demonstrated by comparison with a state of the art alternative solution on a representative case study.**

I. Introduction and related work

Stochastic model-based approaches are popular means to perform system dependability analysis. A variety of modeling formalisms and model solution techniques, typically automated in commercially available tools, have been developed since decades to assist the modeler's activity. Although a system model is a representation at an appropriate degree of abstraction of the real system under analysis, as necessary to satisfy requirements on computational feasibility of the analysis itself, the growing complexity and size of modern and future systems pose continuous challenges. Modularity and composition at model level are key principles commonly adopted, but reflected at solution level only under specific conditions (e.g., peculiar symmetries or convenient hierarchical structures). Unfortunately, unaffordable execution time to obtain results from model solution is an observable obstacle when dealing with system size representative of the reality in many sectors, thus confining the analysis to limited system configurations that, consequently, generate limited interest by system utilizers in the assessment results. Of course, raising the level of system abstraction so to match the capability of available modeling and evaluation methods is always a possibility. But then the drawback is the risk of too poor accuracy of the obtained analysis results, if the attainable representation is too abstract to properly account for the specificities requested by the analysis purpose. Certainly, when dealing with critical systems employed in vital sectors such as critical infrastructures, accuracy of the assessment is a key issue. In particular, dependability related properties (such as reliability, safety and performability) are primarily relevant in such contexts and assessing whether a designed-implemented system fulfills them at a pre-defined level of accuracy is an essential, but very challenging activity. By boosting efficient simulation-based evaluation, we aim at enhancing accuracy of dependability analysis of (potentially large) critical systems in realistic scenarios.

The study started from the observation that many systems of interest from the dependability and resilience perspective are actually composed by large populations of similar non-anonymous components. By exploiting such observation, this paper presents a modeling strategy well suited to obtain efficient simulation-based evaluations in such contexts.

In the literature, modeling of system components replication has been mainly addressed in the form of anonymous replication, well suited to represent a population of identical components, each one interacting with all the others or completely independent, following the approach originally presented in [1]. This configuration favors the application of the strong lumping theorem [2], enhancing the state-space generation based solvers [3], as well as simulation-based solvers [4]. The key point is the possibility to exploit the trivial "all or nothing" symmetry that characterizes the complete directed graph of interactions among components. For identical components connected in arbitrary ways, symmetry exploitation methods [2] have been adopted. In case of the "all or nothing" interconnection pattern, other approaches focus on syncronization [5], [6], in particular when stochastic Petri nets dialects are employed. Alternatively, behavioral aspects [7], such as bisimulation, have been pursued by the stochastic process algebra community.

The need to cope with modeling of different *but similar* components, e.g., when the topological position of a specific system component has an impact on its parameters setting, triggered solutions based on non-anonymous replication. In [8], [9] a solution, referred in the following as *State-Sharing* (*SS*) approach, is proposed. It is a general solution for

simulation-based solvers, but its efficiency is limited because it relies on the pessimistic approach of a complete dependency graph among all the replicas. Instead, the great majority of real-world systems are typically composed by many loosely interconnected components according to regular topologies (tree, mesh, cycle, etc). A population of different but similar components is also considered in [10], where a graph of connections is employed to construct a well shaped matrix in which differences among components correspond to non-zero off-diagonal entries. Similarities help the numerical solver in gaining performance, being the computation based on a linear system solution that is sensitive to the number of non-zero off-diagonal elements. In [10], there is no template model and no replication, and the definition and composition of submodels is manually performed.

In general, modelling a large population of different inter-dependent components, without using a generic template as a building block in the definition of the overall model, requires to define and compose a large number of different models, one for each component. Automated procedures are certainly desirable both from the efficiency point of view and to enhance correctness of the developed model.

Taking an approach based on the modular composition of template models through replication and join operators, similarly to the *SS* approach, in this paper we propose a new mechanism to non-anonymous replication, named *Dependency-Aware Replication* (*DARep*), that greatly improves on efficiency when simulation-based solvers are adopted. The idea is to exploit the real dependency graph characterizing the interactions among the system entities, usually not a complete graph structure, to reduce the solution time. In fact, it is expected that dependency awareness will benefit the simulation operations, including the simulator initialization step. Promoting higher efficiency is greatly relevant, since it makes possible to enlarge the size of the systems that can be analyzed and better represent scenarios at the required level of accuracy (especially crucial for dependability critical applications).

Very recently, the authors also developed an alternative approach, presented in [11], which specifically addresses non-anonymous replication of loosely interconnected components adopting model-based simulation in Möbius. This strategy, named channel-sharing approach, uses a single channel shared among all the replicas to exchange values of the state variables following the actual system topology. Although based on the same principle of exploiting real dependencies as in *DARep*, it has been conceived as a Möbius modeling mechanism for system scenarios with low dependency degrees in presence of large number of replicas. These are no more the premises for the new *DARep* solution.

The rest of the paper is organized as follows. Section II briefly introduces the category of systems under analysis and their characteristics. Section III describes the formalism and tool used to define and evaluate the proposed approach, which is then presented in Section IV. To gain insights on concrete benefits, extensive comparisons with the *SS* approach are carried on in Section VI, adopting a simple but representative case study previously presented in Section V. The obtained results show the superiority of the newly introduced approach in all the investigated scenarios. Conclusions are finally drawn in Section VII. To help the reading, a table of acronyms is also included in Appendix A.

II. Logical architecture of targeted systems

The reference system category for this study is composed by a large number of components, grouped according to well known structures, e.g., tree, cycle, mesh, etc. The connection among two or more components induces a dependency among the behaviour, and so in general the state, of the involved components. Therefore, in the following, the terms connection and dependency are used as synonymous, when referred to the relation among components.

Cyber-physical systems in critical sectors, such as transportation, electricity, water and oil, fit well in the addressed system category. For example, an electrical grid encompasses, among its components, a number of collection points called busses, which are different but similar. The main aspects that determine differences among them include the position occupied in the grid topology and the number and kind of attached electrical equipments for energy production or consumption. When abstracting the bus component for analysis purpose, a *generic* bus component can be assumed and a template model for it can be built, which is then replicated through an indexing function to model all the *specific* bus components included in the system, each with its individual peculiarities. Therefore, although they all adhere to the structure and behaviour of the generic bus component, each bus replica is non-anonymous and needs to be specifically distinguished through an index.

The solution proposed in this paper is meant to assist the modeler in modeling a generic component and then automatically build different *but similar* components, connected according to a predefined topology. Of course, the definition of the logical architecture of the system under analysis in terms of types and number of components to be accounted for in the models, as well as the abstraction level to be adopted for their representation, strongly depends on which is the goal of the analysis. As a simple exemplification, if the interest is in determining the overall system failure rate, the system abstraction level is in general different from the case where the objective is to quantify the impact of failure propagation among specific system components.

In general terms, the logical architecture of the given reference system can be seen as composed by:

- A large number of connected components (called specific components).
- One or more generic components. Each generic component groups all the specific components having common characteristics, i.e., homogeneous system components, which, although different, share the same behaviour, structure and parameters. This means that the template model built for the generic component is adequate to represent the set of its specific components.

- A topology that defines the connections among the generic or specific components. For the purpose of the *DARep* mechanism, for each generic component, only the dependencies existing among its specific components are of interest, so in this paper the focus is restricted to them.

Without loosing in generality, but for the sake of simplicity, in the following only one generic component with its n specific connected components is considered. In fact, extending to h generic components, simply implies simultaneous application of the *DARep* mechanism to the h template models, each one representing a generic component. In order to allow automatic generation of the non-anonymous replicas, the structure, behavior and parameters of the generic component have to be defined as a function of the index i of the replica, with $i = 0, 1, \ldots, n-1$. Each specific component is characterized by the value of the index i of the replica and it is referred as "component i". The state of each component i is represented by v state variables

$$SV_{i,0}, SV_{i,1}, \ldots, SV_{i,v-1}$$

with v being the same for each replica. The values of the state variables can be discrete and continuous. A specific component sc_i can depend on state variables of other components according to a predefined topology. Of course, at the same time other components can depend on state variables of sc_i. Therefore, we distinguish two kinds of state variables. Those that impact on, or are impacted by, other specific components, called *dependency-related state variables*, indicated with $SV_{i,0}, SV_{i,1}, \ldots, SV_{i,m-1}$, where $m \leq v$, being m the same for each replica. The other $v-m$ state variables are referred as non dependency-related state variables, or *local state variables*. The *dependency degree* of the component i, called $d_i \in \{0, 1, \ldots, n-1\}$, indicates that the structure, behavior and parameters of the component i depend on dependency-related state variables of d_i other components. The list of those components from which the component i depends on is called $D_i = \{j_0, j_1, \ldots, j_{d_i-1}\}$. If $d_i = 0$ then the component i does not depend on any other components, although the state of the component i can impact on the state of the component j, if $i \in D_j$. When $d_i = 0$ and $\nexists j \mid i \in D_j$ the component i is said to be independent. From a topological point of view, D_i, for $i = 0, \ldots, n-1$, defines an oriented graph that represents how the n components are connected and how they depend on each other to form the overall system. An independent component corresponds to a disconnected node of the graph, while a system where there is full connectivity among its components results in a complete graph.

III. Formalism and Tools

The Möbius modeling framework [12], implemented by the tool Möbius [13], is used to define and evaluate the models expressing the proposed approach. Möbius is a powerful modular environment that supports multiple modeling formalisms and multiple solution techniques.

The Möbius modeling formalism we used for our replication approach is the Stochastic Activity Networks (SAN) formalism [14], a stochastic extension of Petri nets based on four primitives: places, activities (transitions), input gates, and output gates. Primitive data types of the programming language C++, like short, float, double, including structures and arrays, are represented by special places, called "extended places". Input gates define both the enabling condition of an activity and the marking changes occurring when the activity completes. The output gates define the marking changes occurring when the activity completes, but they are randomly chosen at completion of the activity from a probability distribution function, defined by "cases" associated to the activity. The SAN primitives are expressed by C++ code.

Composed models are obtained through two compositional operators, based on the sharing of state variables [1]:

- *Join*, that composes, i.e., brings together two or more (composed or atomic) submodels, and
- *Rep*, that automatically constructs identical copies (replicas) of a (composed or atomic) submodel.

A state variable can be either local to each submodel, if it cannot be directly accessed by other submodels, or shared among submodels or replicas, if the submodels or replicas can directly access that state variable. Both *Join* and *Rep* operators support automated sharing, based on an all-or-none sharing strategy, meaning that an automatically shared state variable can be either shared among all submodels or it is local to each submodels only. In contrast to *Rep*, the *Join* operator supports the sharing of state variables among different subsets of the composed submodels, but each state variable has to be manually defined for each submodel and manually shared using the graphical user interface.

The *Rep* and *Join* operators are defined at level of Abstract Functional Interface (AFI) [12], [15], a common interface between model formalisms and solvers that allows formalism-to-formalism and formalism-to-solver interactions. At AFI level, places and activities (transitions) of SAN correspond to state variables and actions, respectively. For the sake of simplicity, the proposed atomic and composed models are defined using the SAN notation (places and activities). All the formalisms and solvers supported by Möbius are based on, and defined in terms of, C++ code. Thus, the tool supports external C++ data structures, statically defined at compilation time, and can include and link external C++ libraries. A template model represents a group of homogeneous components, as described in Section II. It is an atomic or composed generic model used as a building block in the definition of the overall system model.

Möbius is not an open source software and the AFI level is not accessible via APIs, thus we introduce an additional layer of abstraction between the modeler and Möbius that addresses directly the non-anonymous replication. In particular, our approach is based on XQuery [16], the W3C recommended language for manipulating xml files, XQilla [17], an XQuery and XPath 2 library written in C++, and self-written C++ classes and function templates.

IV. THE *DARep* APPROACH

A. The General Approach

The proposed *DARep* approach models automatically the interactions among similar components represented by non-anonymous replication of a given template model, as described in Section II. The *DARep* approach takes advantage of the topology of dependencies among system components, by sharing the state variables of each replica among only those replicas that need to exchange values of their state variables. The goal is to improve the performance of the simulation solvers with respect to the standard *SS* approach, which uses the automatic replication based on the all-or-none sharing strategy, that is either a state variable is shared among all the replicas or it is local to each replica. Since it is rarely the case that this sharing strategy naturally fits the system under analysis, extra computation time and possibly intricate data structures are necessary to properly manage the correct dependency relations at model definition and solution. To overcome this drawback, the *DARep* approach extends the *Rep* composition operator in order to: 1) generate automatically the replicas of a template model and the state variable associated to each replica, 2) share the state variables among different subsets of the replicas. Thus, it merges the advantages of the *Join* and *Rep* compositional operators.

In more details, the *DARep* approach models similar components and the interactions among them in the following steps (listed in order of execution):

1. For each different dependency-related state variable, manual or automatic definition of the related dependency topology.
2. Manual definition of the template model that represents the generic component.
3. Automatic generation of n atomic models, one for each replica of the template, including only the dependency-related state variables shared among the other replicas, in accordance with the actual dependencies topology.
4. Automatic generation of the composed model defined by the *Join* operator that joins all the atomic models generated at step 3 and shares among different subsets of them all the dependency-related state variables automatically defined at step 3.

The primitives of the formalism used to define the template model (e.g., the SAN formalism) have to include the following two functions:

- *Index()*, that is applied to each atomic template model to get the actual index of the replica. It is replaced, in each atomic model generated at step 3, by the actual index of the modeled replica.
- *Deps(j)*, that is applied to each dependency-related state variable defined in the template model to access to: 1) the actual dependency-related state variable of the current replica, if $j = 0$, and 2) the actual dependency-related state variable of the j-th replica of the list of replicas (components) from which the current replica depends on or that impact on the the current replica, if $j > 0$.

These functions are needed to define a template model as a function of the index of the replica and to access to the dependency-related state variables associated to each replica, in accordance with the actual dependencies topology.

To make more concrete the proposed method, in Section IV-B, the implementation of the above listed steps in the Möbius framework is presented. However, we underline that it is a general approach applicable to any modeling and evaluation environment that supports both composition of submodels based on sharing of state variables and the definition of the above two functions *Index()* and *Deps()*. Of course, specific implementation details could be different, in accordance with what the adopted modeling environment offers.

In the *DARep* approach, the composed model is automatically defined with the *Join* operator, at step 4, in order to share, for each dependency-related state variable defined in the template model, the minimum set of state variables among the minimum set of the replicas. More precisely, for each dependency-related state variable, each replica i requires to share with the other replicas only $d_i + 1$ state variables, i.e. the state variable associated to the replica i and the state variables, associated to other replicas, from which the replica i depends on or that depend on the replica i.

To perform comparison with the state of the art *SS* approach, we briefly recall how this last works, as presented in [8], [9]. *SS* relies on a template SAN model replicated by the *Rep* operator to model similar components and the interactions among them. For each dependency-related state variable of the modeled component, an array of state variables, i.e. one state variable for each replica, is defined in the template model. Differently from the *DARep* approach, *SS* does not exploit the dependency topology and the array is shared among all the replicas of the template model. The template defines an additional state variable local to each replica representing the index of the replica. Using the index in the template, each replica can access to the the entries of the array representing the state variable of the other replicas.

Compared to *SS*, a reduction in the time overhead is expected for *DARep*, both during the initialization of the simulation solver and during the execution of the simulation batches. However, *DARep* introduces a time overhead at generation time of the atomic and composed models (steps 3 and 4) and at compilation time, due to the number n of the atomic models and to the size of the composed model. In particular, for very high values of n and d_i, the time required for the generation and compilation of the composed model could have a relevant impact on the efficiency of the model evaluation.

To understand the impact of these phenomena on the efficiency of the approaches, at varying both the number of considered replicas and the dependency degree, comparisons between the *DARep* approach and the *SS* approach have been performed. To this purpose, both approaches have been implemented in the Möbius modeling environment and the obtained results are shown and discussed in SECTION VI.

B. The Möbius-Based Implementation

The above general steps of the *DARep* approach are detailed in this Section, resorting to the SAN formalism and the Möbius framework.

The template model defined at step 2 is an atomic SAN model, where either plain places or extended places are used to represent respectively one or more dependency-related state variables of the modeled generic component. In particular, struct-type place with m fields or array-type extended place with m entries can be both used to model m different dependency-related state variables associated to a generic component.

The functions *Index()* and *Deps(j)* of *DARep* approach are implemented respectively by two new C++ functions, that can be only used in the SAN template model as follow:

- *SAN*DARep*::sanname::Index()*, and
- *SAN*DARep*::sanname::placename–>Deps(j)*.

where *SANDARep* and *sanname* are C++ namespaces introduced to avoid names conflicting with Möbius C++ code. In particular, a different namespace *sanname* is defined for each template SAN *sanname*. The statement *SAN*DARep*::sanname::Index()* is used in the template model *sanname* to get the index of the replica. It is replaced, in each atomic SAN model generated at step 3, by the actual index of the modeled replica. The statement *SAN*DARep*::sanname::placename–>Deps(j)* refers to: 1) the dependency-related place (state variable) *placename* associated to the current replica i of the template *sanname*, if $j = 0$; 2) the dependency-related place (state variable) *placename* associated to j-th replica of the list D_i shared with the current replica i, if $j > 0$; 3) the list of all dependency-related places (state variables) *placename* defined in the current SAN and associated to the replicas listed in D_i, if j is omitted. The method *Deps()*, without the index j, can be used to pass the list of references to dependency-related places to each user defined C++ function, that can read or update the values of these places. In each atomic SAN model generated at step 3, corresponding to replica i, the statement *SAN*DARep*::sanname::placename–>Deps(j)* is replaced by

```
SANDARep::sanname::rep(j).placename(),
```

where a C++ object *rep* calls the actual method that returns the reference to the dependency-related place *placename* defined in the generated SAN and associated to j-th replica of the list D_i. Moreover, in each SAN primitive where this statement is used (e.g., in the enabling condition of an input gate), all the actual names of the dependency-related places *placename* associated to all replicas of the list D_i are included in the primitive through a call to a dummy empty function having all these names as arguments. This is needed because the dependencies among SAN primitives and places in the Möbius tool are statically defined when the C++ code describing the model is generated, based on the names of the places. Thus, for example, an enabling condition defined by a statement that accesses to a place by reference, like the above statement

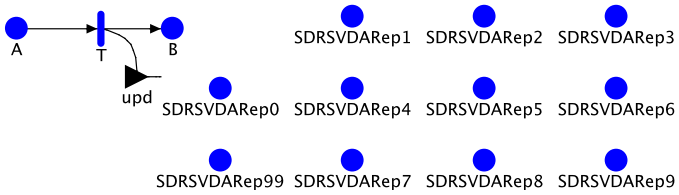


Fig. 1. SAN model *SANSANDARep99* generated by the *DARep* approach from the generic template *SAN* of Figure 5.

generated by the *DARep* approach, is checked at each update of the place only if the enabling condition includes also the name of the place.

At step 3, the C++ code (definition of classes and initialization of objects and constants), used to implement this method and that depends on the dependencies topology, is automatically generated and included in each submodel. In particular, the code to initialize the object *rep* with the list of the pointers to the dependency-related places associated to each replica listed in D_i, is included in the field "Custom Initialization" of each *SAN*, thus the C++ data structures are set before the model evaluation starts.

An example of definition of output gate in a SAN template *mysan* including the dependency-related place *S* and the local place *B* is the following:

```
B->Mark()=
  SANDARep::mysan::S->Deps(0)->Mark();
updateS(SANDARep::mysan::S->Deps(),
  SANDARep::mysan::Index());
```

where to the extended place *B* is assigned the value of the dependency-related place *S* associated to the current replica, and the function *updateS()* receives the list of all the dependency-related places *S* of the other replicas that depend on the current replica and updates them based on the index of the current replica *SANDARep::mysan::Index()*.

The i-th atomic SAN model, automatically generated at step 3, represents the i-th replica. For each dependency-related state variable (place) $SV_{*,y}$, defined for a generic component in the template SAN model, the dependency-related place $SV_{i,y}$ associated to the specific replica i, referred in the template SAN by *SAN*DARep*::sanname::*$SV_{*,y}$*–>Deps(0)*, is automatically generated in the SAN. In addition, all the places $SV_{x,y}$, for $x \in D_i$, that depend on or have impact on it, are automatically generated in the SAN. The places are referred in the template SAN by *SAN*DARep*::sanname::*$SV_{*,y}$*–>Deps(j)*, for $1 \leq j \leq d_i$, where j is the position of the the index x in D_i. For example, Figure 1 shows the SAN model *SANSANDARep99* (the name of the model is obtained merging the name *SAN* of the template, the string *SANDARep* and the index of the replica) representing the 100-th replica (the index of first replica is 0) of the generic template depicted in Figure 5 and presented in Section V. This model is generated by the *DARep* approach for $n = 100$ and $d = 10$.

Figure 2 depicts the left part of the composed model *SANSANDARep* (the name of the model is obtained merging

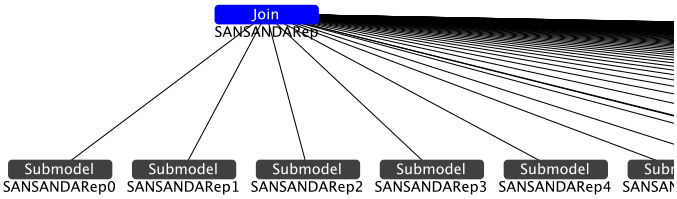


Fig. 2. Composed model *Comp* generated by the *DARep* approach from the generic template *SAN* of Figure 5.

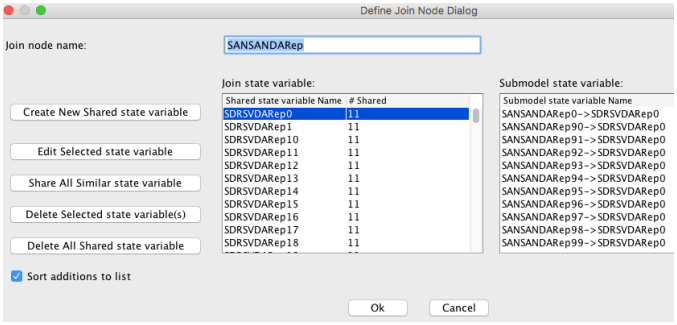


Fig. 3. A snapshot of the "Define Node Join Dialog" of the Möbius tool for the composed model *SANSANDARep* generated by the *DARep* approach from the generic template *SAN* of Figure 5.

the name *SAN* of the template with the string *SANDARep*) automatically generated at step 4, for $n = 100$ and $d = 10$, from the generic template depicted in Figure 5.

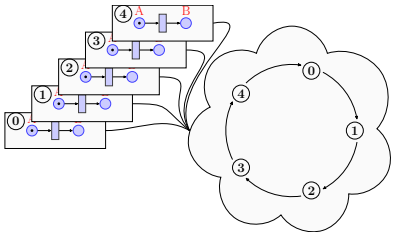
Figure 3 is a snapshot of the "Define Node Join Dialog" of the Möbius tool for the composed model *SANSANDARep*. It shows the list automatically generated of all the dependency-related places $SDRSVDARep_i$ (the name of each place is obtained merging the name *S* of the place, the string *DRSVDARep* and the index of the replica associated to the place) associated to each replica of the template, that are shared among subsets of submodels (replicas) in accordance with the actual dependencies topology. In particular, for the first place $SDRSVDARep_0$, the list of the submodels (replicas) that share the place is shown.

Each model generated at steps 3 and 4 is defined with an *XML* file, automatically generated with the XQilla tool using the dependency topology described with an *XML* input file defined at step 1. Each time the template SAN model undergoes updates at step 2 that imply changes in the number of replicas n or the dependency topology, the steps 3 and 4 must be repeated, to update the resulting *xml* files and C++ files. Consequently the overall model must be compiled again.

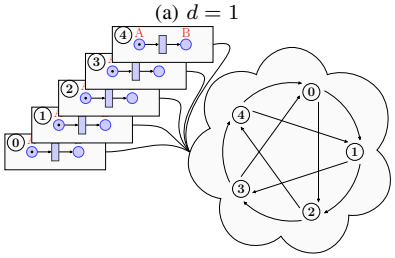
V. Case Study

As already mentioned in Section II, to illustrate the concrete application of our proposed replication mechanism we consider a system composed by n different, but similar components, interconnected through a specified topology which involves a varying number of components (including the two extremes of having each component connected with only another one, to the case where each component is connected with all the other ones). Although simple, our case study is effective in demonstrating the features and benefits of the *DARep* mechanism, since it fully represents the logical architecture of targeted systems described in Section II and can be considered as a basis to be easily extended and adapted to represent a great variety of real contexts.

Similarly to the system logical structure adopted in the study in [18], let's consider n working stations dedicated to perform the same task in parallel. At every time instant, each station can be either *working* or *failed*, and the change of status takes place after an exponentially distributed random time. The failure of a station implies a reconfiguration of the workload assigned to the other stations, to continue accomplishing the tasks of the failed station. Just before failing, a station redirects its tasks to one or more other stations it is connected with, i.e. neighbouring stations according to the dependency topology. The stations that receive new tasks increase their workload, implying also a change of their failure rate.



(a) $d = 1$



(b) $d = 2$

Fig. 4. Logical structure of the case study for $n = 5$ and two dependency scenarios: $d = 1$ in (a) and $d = 2$ in (b).

This system is modeled as a pure death process [19] with monotone load sharing [20]. To reflect the impact of failures of neighbouring components on the failure rate of component i, this last is determined according to the following expression:

$$\lambda_i = const \cdot \left((i \bmod 10 + 1) + \sum_{j \in D_i} \mathbb{1}_{\{\text{component } j \text{ is failed}\}} \right)$$

where the first addendum of the sum in parenthesis accounts for the failure rate of component i in isolation, and the second addendum for the increment due to the failure of neighbouring components it depends on.

The workload of failed stations is transferred to the d cyclically following working stations, determined according to the following expression:

$$D_i = \{i + 1 \bmod n, \ldots, i + d \bmod n\}$$

Figure 4 illustrates a system configuration of 5 components in two simple dependency scenarios: $d = 1$ and $d = 2$. Then,

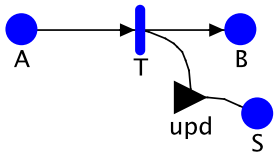


Fig. 5. Case study generic SAN model.

should for example component 0 fail, its workload is transferred to component 1, which also increases its failure rate, in the scenario with $d = 1$. In case of scenario with $d = 2$, the workload originally assigned to the failed component 0 is transferred to components 1 and 2, which also increase their failure rates.

In this case study, the same dependency degree is assumed for each component, i.e., $d_i = d$ holds for each i, although different degrees could be easily accommodated.

To gradually investigate the impact of the different dimensions impacting on the performance of the studied replication mechanisms, the representation of the workload reconfiguration process assumes in our study only one dependency-related state variable ($m = 1$). Some dialects of Stochastic Petri Nets, including Stochastic Well-Formed Nets [21] and SANs, offer arbitrary typed tokens or places, thus having only one dependency-related state variable allows anyhow to model a rich information exchange among components. The SAN model implementing the generic component (the working station) is depicted in Figure 5. The model consists of two places, *A* and *B*, the transition *T* and the dependency-related place *S*. When a token is in *A* the component is working, when a token is in *B* the component is failed, and the two alternatives are mutually exclusive. At the beginning, each replica initializes $S{->}Deps(0){->}Mark()$ with its local load. Whenever a replica fails, via the Output Gate *upd*, it increments each Place in $S{->}Deps()$ by 1. Transition *T* has then an exponentially distributed time and an index dependent rate equal to $const \cdot S{->}Deps(0){->}Mark()$.

Note that Figure 5 depicts only the structural part of the case study template SAN model, leaving the logic inside λ_i and D_i.

VI. Evaluation Results

To demonstrate the effectiveness of the proposed approach, a comparison of the performance results of the Möbius simulator obtained by both *DARep* and *SS* approaches has been conducted. To this purpose, the terminating Möbius simulator [12] has been used to evaluate at each execution different measures of dependability (reward variables) for the proposed case study, like the cumulated time a component stays in a specific state and the probability that a component is failed at time t. As a form of validation, for some sample models, it has been verified that the number of stable states obtained with both approaches is the same and it is equal to the theoretical prediction. In addition, also the results obtained for the defined measures have been the same for both *DARep* and *SS* approaches. Different reward structures [5], [19] over different set of markings have a different impact on simulation times. Thus, to improve accuracy and fairness of the comparison, a high number of reward variables (around 50) has been considered in the study. The evaluated measures span from indicators relative to individual components (such as the probability of failure or the MTTF of a specific system entity), to indicators relative to portions of the system, up to the overall system (such as the system MTTF). However, since here the analysis focuses on the comparison of the performance of the two approaches, details on these measures and the obtained results are out of the scope of this paper.

Each execution of the terminating Möbius simulator is defined for a specific setting of all the parameters of the considered models (corresponding to an experiment in the Möbius terminology). Each execution of the terminating simulator starts initializing the data structures, then runs k batches (replications in Möbius notation) with $k \geq 1$.

The following performance measures have been considered:

- $\tau(k)$: Total amount of *CPU* time, in seconds, used by one execution of the Möbius simulator that runs k batches, with $k \geq 1$.
- τ_{init} or $\tau(0)$: The amount of *CPU* time, in seconds, used by one execution of the Möbius simulator to initialize the data structures of the simulator. This is the *CPU* time used by the simulator to output the string "SIMULATOR::Preparing to run()". The definition of τ_{init} as a function of $\tau(k)$ is: $\tau_{init} = \tau(1) - (\tau(2) - \tau(1)) = 2\tau(1) - \tau(2)$, where $\tau(2) - \tau(1)$ is the total amount of *CPU*, in seconds, used by one execution of the Möbius simulator to run a batch.
- $\Delta\tau(k)$: Difference between run time and initialization time:
$$\Delta\tau(k) = \tau(k) - \tau_{init}$$
- $\mathcal{C}\tau(k)$: Compared simulation performance (pure number) between *SS* and *DARep*, defined as follows:
$$\mathcal{C}\tau(k) = \frac{\Delta\tau^{DARep}(k)}{\Delta\tau^{SS}(k)}$$
where $k > 0$ and the superscripts "*SS*" and "*DARep*" refer to *State-Sharing* and *Dependency-Aware Replication* approaches, respectively.

The considered *CPU* time includes both user and system *CPU* times. As already stated, only one template model is assumed with its n replicas, and the number of dependency-related state variables is also one ($m = 1$). Then, to exercise the approaches in relevant contexts, the scenarios generated as combinations of the following values for n, d and k, have been considered:

- number n of replicas ranging from 10 to 1000,
- dependency degree d varying from 1 (minimum connectivity) to 500,
- number of batches k varying from 1 to 10000. The value of k impacts on the precision of the obtained results. Although 10000 showed an appropriate value to assure convergence satisfying the selected requirement (confidence interval of width less than 10^{-5}) for most of

the measures we considered, we also observed that for some of them 50000 batches have been necessary.

Simulations were sequentially performed on Intel(R) Core(TM) i7-5960X with fixed 3.50 GHz CPU, 20M cache and 32GB RAM, and an up to date GNU/Linux Operating System. Each $\tau(k)$ has been evaluated 10 times and the arithmetic mean is presented in Tables I to IV, VI and VII. It is important to notice that, for the *SS* approach, models compilation times are negligible, being constituted by a single atomic SAN and one composed model, while for the *DARep* approach component models compilation time can be relevant. In particular, if $n \leq 100$ then atomic SANs compilation times and composed model compilation time can take few minutes, while for $n \approx 1000$ and $d \approx 500$ composed model compilation time can grow up to about 2.8 hours. For $d \approx 999$ composed model compilation time can take several hours and the C++ source code can reach $250Mbyte$ of size. However, composed model re-compilations are needed *only* when interdependencies are modified, thus – once fixed the topology of the system – the modeler can change or define new measures and studies without having to re-compile.

Tables I and II show initialization times for both *SS* and *DARep*, for different values of n and d. All the table entries in which $d \geq n$ are empty because $d < n$ by definition. As expected, since the *SS* approach works always considering the pessimistic case of totally interdependent replicas, τ_{init} remains constant at varying the value of d. Of course, the value of τ_{init} grows at the increasing of n: it is about 0.23 seconds for $n = 100$, and about 560 seconds for $n = 1000$. On the contrary, τ_{init} of the *DARep* approach changes at varying d, but the assumed values are always negligible compared with τ_{init} of the *SS* approach.

TABLE I
τ_{init} FOR THE *SS* APPROACH.

		d			
		1	9	99	500
n	10^1	0.010	0.010		
	10^2	0.229	0.238	0.242	
	10^3	565.736	560.015	561.493	562.433

TABLE II
τ_{init} FOR THE *DARep* APPROACH.

		d			
		1	9	99	500
n	10^1	0.011	0.010		
	10^2	0.011	0.011	0.130	
	10^3	0.043	0.136	2.250	84.643

Tables III and IV show, for $k = 1000$, the difference between run and initialization times (in seconds) for $n = 10, 100, 1000$ and $d = 1, 9, 99, 500$, respectively for *SS* and *DARep*. Although the values shown by *DARep* are small and significantly lower than the corresponding ones of *SS*, it can be observed the different trend of the two approaches with respect to d. In fact, while the impact of d on $\Delta\tau(1000)$ is small in the table relative to *SS*, in the case of *DARep* the value of $\Delta\tau(1000)$ for $d = 500$ is about 1.6 times the value for $d = 1$. This is not surprising, since *SS* always works under the implicit assumption of maximum interconnection among component replicas, so its sensitivity to variation of d is minimal, while *DARep* is significantly influenced by d, given the applied principle of considering only real replicas interdependencies. With respect to increasing values of n, as expected the results obtained for $\Delta\tau(1000)$ increase for both approaches. However, *DARep* can be more than one order of magnitude faster than *SS* when $n = 100$ and d up to 9 and when $n = 1000$.

TABLE III
$\Delta\tau(1000)$ IN SECONDS FOR THE *SS* APPROACH.

		d			
		1	9	99	500
n	10^1	0.087	0.102		
	10^2	9.203	9.197	9.357	
	10^3	1613.246	1723.666	1732.983	1754.996

TABLE IV
$\Delta\tau(1000)$ IN SECONDS FOR THE *DARep* APPROACH.

		d			
		1	9	99	500
n	10^1	0.015	0.022		
	10^2	0.774	0.817	0.972	
	10^3	104.939	109.797	131.580	167.160

Elaborating on the results shown in Tables III and IV, in Table V the comparisons between $\Delta\tau(1000)$ of *DARep* and $\Delta\tau(1000)$ of *SS* are shown. The improvement in efficiency brought by *DARep* is very significant, especially at increasing the value of n: *DARep* is between 0.67 and 1.19 orders of magnitude faster than *SS*.

TABLE V
COMPARISONS $\mathcal{C}\tau(1000)$.

		d			
		1	9	99	500
n	10^1	0.175	0.213		
	10^2	0.084	0.089	0.104	
	10^3	0.065	0.064	0.076	0.095

Tables VI and VII show, for $d = 10$, differences between run and initialization times (in seconds) for $n = 100, 1000$ and $k = 1, 10, 100, 1000, 10000$, for *SS* and *DARep* respectively. As expected, $\Delta\tau(k)$ is (about linearly) proportional to k, for both *DARep* and *SS*. Also for this performance indicator, there is a strong superiority of *DARep* over *SS*, especially in the scenarios with high values of n (systems of larger size, which are those we specifically address) and of k (to assure more accurate results, as typically requested in dependability analysis). For $n = 1000$ and $d = 10$, the mean run times are $\tau^{SS}(10000) = 574.032 + 13085.148 = 13659.180$ seconds (about 3.8 hours) and $\tau^{DARep} = 0.138 + 1067.821 = 1067.959$ seconds (about 17.8 minutes).

TABLE VI
$\Delta\tau(k)$ IN SECONDS FOR THE *SS* APPROACH WHEN $d = 10$.

		k				
		10^0	10^1	10^2	10^3	10^4
n	10^2	0.020	0.095	0.937	9.208	91.584
	10^3	5.396	20.326	180.629	1675.024	13085.148

TABLE VII
$\Delta\tau(k)$ IN SECONDS FOR THE *DARep* WHEN $d = 10$.

		k				
		10^0	10^1	10^2	10^3	10^4
n	10^2	0.001	0.011	0.089	0.811	8.081
	10^3	0.111	1.120	10.976	109.692	1067.821

Finally, Table VIII presents comparisons between $\Delta\tau(k)$ of *DARep* and $\Delta\tau(k)$ of *SS* with respect to values shown in Tables VI and VII. The obtained values for $\mathcal{C}\tau(k)$ clearly point out that *DARep* is again about one order of magnitude faster than *SS*.

TABLE VIII
$\mathcal{C}\tau(k)$ WHEN d IS FIXED AT 10.

		k				
		10^0	10^1	10^2	10^3	10^4
n	10^2	0.045	0.116	0.095	0.088	0.088
	10^3	0.021	0.055	0.061	0.065	0.082

VII. CONCLUSIONS

Moving from considerations on the need to promote efficient dependability and performability model-based analysis to properly address the increasing size of modern and future critical and complex systems, this paper developed a novel replicator operator for non-anonymous replication in systems composed by large populations of interconnected components, when simulation-based solvers are used. Although the demand for non-anonymous replication comes from a variety of key sectors for our society and economy, the available solutions show significant limitations in terms of efficiency when copying with real-size systems. The major principle the new solution, *DARep*, is based on is to exploit the actually existing dependencies among components of the system under analysis, instead of relying on the pessimistic situation of point-to-point connections as assumed by the already existing *SS* approach. The underlying principle is that making use of topology information is expected to reduce the operations performed by the solver, both at initialization and simulation time. Although the approach is general and applicable to any modeling and evaluation environment that supports automatic replication and composition of submodels based on sharing of state variables, in this work we adopted the Möbius framework as implementation environment. Then, to quantify the extent of the expected gain in performance and to better understand the interplay of the peculiarities of both solutions, an evaluation study involving both the *State-Sharing* and *Dependency-Aware Replication* approaches has been conducted using the terminating simulator of Möbius. Indicators representative of the execution and initialization time of the simulator have been computed on a simple but representative case study. Several scenarios, characterized by different values of the number of replicas, the dependency degree and the number of simulation batches, have been investigated.

The simulation results demonstrate the superiority of the newly introduced *Dependency-Aware Replication* method with respect to the *State-Sharing* competitor in all the considered scenarios.

Extensions of the presented study are foreseen in several directions. An already on-going activity is to progress on the performance evaluation of *Dependency-Aware Replication*, investigating system configurations with higher numbers of n and d. Although the analyses performed so far are the most representative of realistic applications scenarios, stressing further these parameters would allow understanding the applicability limits of the approach, both from a theoretical perspective and in view of potential future needs.

Another immediate advancement direction is the adoption of the new approach in more complex system scenarios, as offered by the power grid sector. It would require relaxing the simplistic assumption on having just one template model with equal dependency degree among all its replicas, which has been made in this paper to easy the presentation of the *DARep* solution. However, this is also a context where the dependency degree d is very small compared to the number of system components, so the benefits of using *DARep* instead of *SS* are expected to be very significant, according to the results presented in the previous section. In fact, focusing on the bus component, which is the major grid component, and examining some reference grids used in other studies, and precisely the IEEE118, IEEE300 testbed [22], [23] and the Illinois Center for a Smarter Electric Grid's Texas synthetic grid [24], the dependency degrees are numbers between 2 and 3 on average, with maximum value of 16 for the configuration with 2000 buses.

Certainly, taking advantage of the results obtained so far and the understanding of the dynamics that generated them, another line of extension would be to enhance the *DARep* mechanism towards making it applicable to a wider class of systems, e.g. where the dependency-related state variables may change for different subsets of dependency-related replicas (that is, the parameter m is no more constant for all the replicas, as in the current version).

Also, implementing the *DARep* approach in a different modeling environment would provide further feedback on the implications of the underlying technological choices provided by the adopted modeling and solution tool. Although Möbius is a widely used framework for dependability and performance analysis, it is not the only one and extending the proposed replication mechanism to other stochastic model-based frameworks would help modelers familiar with them. One example can be GreatSPN [21], [25].

Finally, a more long term objective would be to define and making native in the adopted evaluation tool a new

non-anonymous replica operator, based on the principle of state sharing only among interdependent replicas. This would certainly improve efficiency with respect to solutions that can be built by a tool utilizer on top of offered operators/features, as done so far. Compared to *DARep*, such a smart replicator operator would avoid resorting to the many files now needed, which require increasing (and possibly prohibitive) compilation time at increasing the number of replicas and the dependency degree they show.

APPENDIX

SAN	Stochastic Activity Networks
AFI	Abstract Functional Interface
SV	State Variable
SS	*State-Sharing*
DARep	*Dependency-Aware Replication*
n	Number of system components
m	Number of dependency-aware state variables
h	Number of generic components
d_i	Dependency degree of component *i*
d	Constant dependency degree
D_i	List of those components from which the component *i* depends on
k	Number of simulation runs (batches)
$\tau(k)$	Möbius CPU run time with k batches
$\Delta\tau(k)$	Difference between Möbius run and initialization time with k batches
$\mathcal{C}\tau(k)$	Compared simulation performance between *DARep* and *SS* with k batches
i, j, x, y	System components indeces

REFERENCES

[1] W. H. Sanders and J. F. Meyer, "A unified approach for specifying measures of performance, dependability and performability," in *Dependable Computing for Critical Applications, Vol. 4 of Dependable Computing and Fault-Tolerant Systems*, A. Avizienis and J. Laprie, Eds. Springer Verlag, 1991, pp. 215–237.

[2] P. Buchholz, "Exact and ordinary lumpability in finite Markov chains," *J. of Appl. Probab.*, vol. 31, no. 1, pp. 59–75, 1994.

[3] S. Derisavi, P. Kemper, and W. H. Sanders, "Symbolic state-space exploration and numerical analysis of state-sharing composed models," *Linear Algebra and its Applications, Special Issue on the Conf. on the Numerical Solution of Markov Chains 2003*, vol. 386, pp. 137–166, 2004.

[4] W. H. Sanders and R. S. Freire, "Efficient simulation of hierarchical stochastic activity network models," *Discrete Event Dynamic Systems*, vol. 3, no. 2, pp. 271–299, 1993.

[5] V. V. Lam, P. Buchholz, and W. H. Sanders, "A component-level path-based simulation approach for efficient analysis of large Markov models," in *37th Conf. on Winter Simulation*, M. E. Kuhl, N. M. Steiger, F. B. Armstrong, and J. A. Joines, Eds., Orlando, Florida, 2005, pp. 584–590.

[6] L. Brenner, P. Fernandes, A. Sales, and T. Webber, "A framework to decompose GSPN models," in *Applications and Theory of Petri Nets 2005*, ser. LNCS, G. Ciardo and P. Darondeau, Eds. Springer-Verlag, 2005, vol. 3536, pp. 128–147.

[7] J. Hillston, *Compositional Markovian Modelling Using a Process Algebra*. Boston, MA: Springer US, 1995, pp. 177–196.

[8] S. Chiaradonna, P. Lollini, and F. Di Giandomenico, "On a modeling framework for the analysis of interdependencies in electric power systems," in 37^{th} *Annu. IEEE/IFIP Int. Conf. on Dependable Syst. and Netw. (DSN 2007)*, Edinburgh, UK, June 2007, pp. 185–195.

[9] F. Flammini, *Critical Infrastructure Security: Assessment, Prevention, Detection, Response*, ser. Information & communication technologies. WIT Press, 2012.

[10] G. Ciardo and K. S. Trivedi, "A decomposition approach for stochastic reward net models," *Perform. Eval.*, vol. 18, no. 1, pp. 37–59, 1993.

[11] G. Masetti, S. Chiaradonna, and F. Di Giandomenico, "Model-based simulation in Mobius: an efficient approach targeting loosely interconnected components," forthcoming Computer Performance Engineering: 13th European Workshop (EPEW), Berlin, Germany, Sep. 2017.

[12] D. D. Deavours, G. Clark, T. Courtney, D. Daly, S. Derisavi, J. M. Doyle, W. H. Sanders, and P. G. Webster, "The Möbius framework and its implementation," *IEEE Trans. on Softw. Eng.*, vol. 28, no. 10, pp. 956–969, 2002.

[13] T. Courtney, S. Gaonkar, K. Keefe, E. W. D. Rozier, and W. H. Sanders, "Möbius 2.3: An extensible tool for dependability, security, and performance evaluation of large and complex system models," in 39^{th} *Annu. IEEE/IFIP Int. Conf. on Dependable Syst. and Netw. (DSN 2009)*, Estoril, Lisbon, Portugal, 2009, pp. 353–358.

[14] W. H. Sanders and J. F. Meyer, "Stochastic activity networks: Formal definitions and concepts," in *Lectures on formal methods and performance analysis: first EEF/Euro summer school on trends in computer science, Berg en Dal, The Netherlands, July 3-7, 2000, Revised Lectures*, ser. LNCS, E. Brinksma, H. Hermanns, and J. P. Katoen, Eds. Springer-Verlag, 2001, vol. 2090, pp. 315–343.

[15] J. M. Doyle, "Abstract model specification using the Möbius modeling tool," Master's thesis, University of Illinois at Urbana-Champaign, 2000.

[16] M. Brundage, *XQuery: The XML Query Language*. Pearson Higher Education, 2004.

[17] Oracle, Decisionsoft Ltd, Parthenon Computing Ltd, Y. Cai, G. Feinberg, L. Foutz, B. Kolpackov, A. Massari, and J. Snelson, "XQilla," http://xqilla.sourceforge.net/HomePage, 2011.

[18] L. Xing and G. Levitin, "Reliability of systems subject to failures with dependent propagation effect," *IEEE Trans. of Systems, Man, and Cybernetics: Systems*, vol. 43, 2013.

[19] K. S. Trivedi, *Probability and Statistics with Reliability, Queueing and Computer Science Applications*, 2nd ed. New York: John Wiley & Sons, 2002.

[20] B. M. Amari, *Handbook of Performability Engineering*. Springer, 2008.

[21] M. Beccuti and G. Franceschinis, "Efficient simulation of stochastic well-formed nets through symmetry exploitation," in *2012 Winter Simulation Conference (WSC)*, Berlin, Germany, 2012, pp. 1–13.

[22] R. Christie, "IEEE 118-bus test case," http://icseg.iti.illinois.edu/ieee-118-bus-system, 1993.

[23] M. Adibi and IEEE Test Systems Task Force, "IEEE 300-bus test case," http://icseg.iti.illinois.edu/ieee-300-bus-system, 1993.

[24] Illinois Center for a Smarter Electric Grid (ICSEG), "June 2016 texas 2000 syntetic test case," http://icseg.iti.illinois.edu/synthetic-power-cases/texas2000-june2016, 2016.

[25] S. Baarir, M. Beccuti, D. Cerotti, M. D. Pierro, S. Donatelli, and G. Franceschinis, "The GreatSPN tool: recent enhancements," *ACM SIGMETRICS Perform. Eval. Rev., Spec. Issue on Tools for computer performance modeling and reliability analysis*, vol. 36, no. 4, pp. 4–9, 2009.