International Journal of Performability Engineering, Vol.7, No. 5, September 2011, pp. 417-428.

Printed in India

MDWNsolver: A Framework to Design and Solve Markov Decision Petri Nets

MARCO BECCUTI*[1], GIULIANA FRANCESCHINIS[2] AND SERGE HADDAD[3]

[1]*Univ. di Torino, Italy*
[2]*Univ.del Piemonte Orientale, Italy*
[3]*LSV, ENS Cachan, France*

(Received on December 02, 2010 and revised on March 22, 2011)

Abstract: *MDWNsolver* is a framework for system modeling and optimization of performability measures based on Markov Decision Petri Net (MDPN) and Markov Decision Well-formed Net (MDWN) formalisms, two Petri Net extensions for high level specification of Markov Decision Processes (MDP). It is integrated in the GreatSPN suite which provides a GUI to design MDPN/MDWN models. From the analysis point of view, *MDWNsolver* uses efficient algorithms that take advantage of system symmetries, thus reducing the analysis complexity. In this paper the *MDWNsolver* framework features and architecture are presented, and some application examples are discussed.

Keywords: *Markov decision process, dependability optimization tool, Markov decision well-formed nets.*

1. Introduction

The *Markov Decision Process (MDP)* formalism [12] can be used for modeling systems which exhibit both non deterministic and probabilistic behavior (*e.g.,* distributed systems, resource management systems, *etc,).* Being a low level formalism, it is rather hard to directly use MDPs to model complex systems. Some high level MDP specification formalisms have been proposed in the literature to overcome this problem (*e.g.,* Stochastic Transition Systems [9], Dynamic Decision Network [10], Reactive Modules [1], *etc,*); in this context, the originality of Markov Decision Petri Net (MDPN) and Markov Decision Well-formed Net (MDWN) [5] high level formalisms is that they allow to describe the system in terms of its components and their interactions. As a consequence, the models are more compact and manageable; in particular, it is possible to define a complex non deterministic or probabilistic behavior as a composition of simpler non deterministic or probabilistic steps (that take zero time). From the analysis point of view, the MDWN formalism inherits the efficient algorithms originally devised for WNs, allowing to automatically taking advantage of the model symmetries to reduce the analysis complexity.

This paper presents a framework, integrated in the *GreatSPN* suite [14]: MDPN/MDWN models can be designed using the *GreatSPN* GUI, and solved by means of specific new modules integrated in the distribution. These modules transform an MDPN/MDWN model expressed as a pair of non-deterministic and probabilistic subnets plus a reward function specification into an MDP model and then solve such MDP deriving an optimal strategy.

The paper is organized as follows: Section 2 briefly recalls the MDP, MDPN and MDWN formalisms. In Section 3, the main features of the framework are explained, and

Corresponding author's email: beccuti@di.unito.it

its architecture is outlined. In Section 4 and 5, some examples of MDPN and MDWN models are presented, together with some analysis results obtained with the *MDWNsolver* and some considerations on the efficiency of the MDWN solution. Section 6 concludes the paper.

2. Background

In this section we recall briefly the MDP, MDPN and MDWN formalisms which will be used in this paper; the reader can find more details in [5].

2.1 Markov Decision Process

MDP [12] is a well-know formalism providing a simple mathematical model to express optimization problems in random environments. In particular, a discrete time finite MDP[1] is an extension of a Markov Chain which allows non deterministic choices/actions, and rewards function expressing a target function to be minimized /maximized. For every non deterministic action allowed in a given state a reward/cost and a transition probability distribution are defined. Hence, the evolution of an MDP can be described as an alternation of non deterministic transitions (actions) and probabilistic transitions.

Solving an MDP consists in finding an optimal strategy (optimal action to be chosen in each state) *w.r.t.* a given reward function.

2.2 Markov Decision Petri Net and Markov Decision Well-formed Net Formalisms

MDPN was first introduced in [5] as a high level formalism to specify MDPs. The main features of MDPNs are the possibility to specify the general behavior as a composition of the behavior of several concurrent components (some of which are subject to local non deterministic choice, and are thus called controllable, while the others are called non controllable); moreover any non deterministic or probabilistic transition of an MDP can be composed by a set of non deterministic or probabilistic steps, each one involving a subset of components.

An MDPN model is composed of two parts; both specified using the PN formalism with priorities associated with transitions: the PN^{nd} subnet and the PN^{pr} subnet (describing the non deterministic (ND) and probabilistic (PR) behavior respectively). The two subnets share the set of places, while having disjoint transition sets. In both subnets the transitions are partitioned into *run* and *stop* subsets, and each transition has an associated set of components involved in its firing (in the PN^{nd} only controllable components can be involved). Transitions in PN^{pr} have a “weight” attribute, used to compute the probability of each firing sequence. Firing of *run* transitions represent intermediate steps in an ND/PR transition at the MDP level, while *stop* transitions represent the final step in an ND/PR MDP transition, for all components involved in it. An MDPN model behavior alternates between ND transition sequences and PR transition sequences, initially starting from an ND state. The PR sequences are determined according to the PN^{pr} structure, start with a PR state reached by an ND state, and include exactly one stop transition for each component; the ND sequences are determined by the PN^{nd} structure, start from an ND state reached by a PR state, and include exactly one stop transition for each controllable

[1] In the rest of this paper we will use MDP to indicate a discrete time finite MDP.

component plus possibly a *global stop* transition. Moreover, in the MDPN formalism we can specify a reward/cost function, called ***rs()*** associated with every system state and one, called ***rt(),*** associated with every non deterministic transition; the global reward function is obtained by summing up a state reward function and an action reward function.

The generation of the MDP corresponding to a given MDPN has been described in [5]: it consists of (1) a composition step, merging the two subnets in a single net, (2) the generation of the RG of the composed net, (3) two reduction steps transforming each PR and ND sequence in the RG into a single MDP transition.

MDWN [5] extends the MDPN formalism with color: the PN^{pr} and PN^{nd} subnets are specified using Well-formed Nets (WN) [7] and a subset of the color classes is used to represent the system components. The transitions are still partitioned into *run* and *stop* subsets (with the same semantics defined in the MDPN), and each transition firing involves a set of components identified by the transition color instance. MDWNs enable the modeler to specify in a concise way similar components, obtaining a more compact and readable model; it is always possible to derive an equivalent MDPN applying an unfolding algorithm. From an analysis point of view, the generation of the MDP corresponding to an MDWN follows the same two steps already explained for MDPN, but in this case the Symbolic Reachability Graph (SRG) [7] approach developed for the WN formalism can be adapted to produce a smaller MDP *w.r.t.* the original one.

3. *MDWNsolver* Features and Architecture

MDWNsolver consists of a module that builds the MDP corresponding to a given MDWN or MDPN model, and produces an output suitable for the MDP analysis by means of an MDP solver built upon the *graphMDP* library [13]; it may be adapted to interact (at the MDP analysis level) with other tools like *e.g.,* ZMDP, allowing to derive both optimal and suboptimal strategies, or PRISM, featuring the computation of properties expressed in PCTL through efficient model checking algorithms.

The architecture of *MDWNsolver* is depicted in Figure 1. The user must specify two subnets (***Prob_net*** and ***ND_net***) by means of the *GreatSPN* GUI, representing the probabilistic and non deterministic behavior of the model. A special annotation is used to associate sets of *components* with transitions, and to distinguish between run and stop transitions. In case MDWN models are used, the components are represented by means of a *color class*: this is useful when the system under study comprises several similarly behaving components, and should be used when the system structure and behavior exhibit a certain degree of symmetry that can be exploited to achieve a more compact representation, and - what is most important - to reduce the model transformation cost as well as the MDP solution cost. Different priorities can be assigned to transitions: this allows to avoid useless interleavings when deriving the MDP model, and to force a correct ordering of probabilistic or non deterministic intermediate (immediate) steps. In addition the ***RewardSpec*** file must be prepared: it is a textual file where the reward function to be optimized is specified according to a given grammar.

The transformation process consists of four steps: (1) the non deterministic and probabilistic subnets are modified by the ***MDWN2WN*** module that adds some places and two (timed) transitions; (2) the resulting new subnets (***Prob_netM*** and ***ND_netM***) are composed through the *algebra* module of *GreatSPN*; (3) from the obtained PN/WN the (S)RG is generated using the module ***MDWN(S)RG***, that produces also two files containing the list of the non deterministic transition sequences (the MDP actions) and markings description (the MDP states), needed to compute the value of the reward function associated with the MDP states and actions; (4) module ***RG2MDP***, generates the final MDP: the states of the MDP correspond to the *tangible* states produced by the previous module, the MDP actions and the subsequent probabilistic transitions, correspond to the *maximal immediate non deterministic/probabilistic paths* respectively, departing from the non deterministic/probabilistic tangible markings and reaching probabilistic/non deterministic tangible markings. In order to make the MDP solution more efficient, the reduction algorithm selects among the actions that connect the same tangible states, that with minimal (or maximal, depending on the optimization problem) reward value. The MDP file is produced in an efficient format which is accepted in input by the ***MDP*** solver module (based on the *graphMDP* library), that produces the optimal strategy and corresponding optimal reward value.

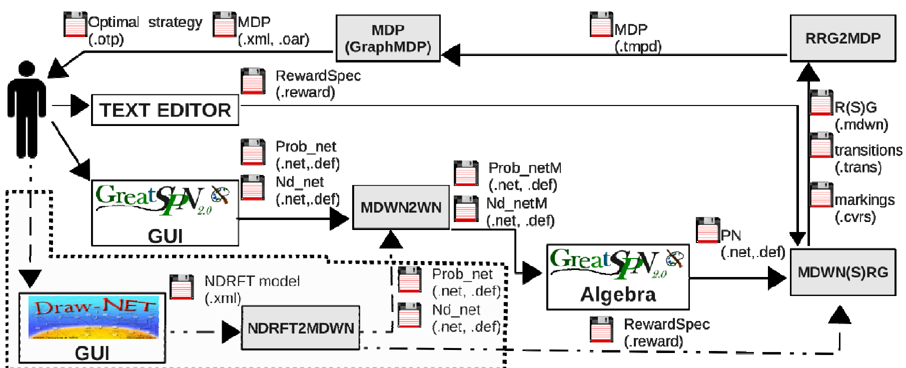

Figure 1: *MDWNsolver* Architecture

The implementation of the ***MDWN(S)RG*** module derives from the *WN(S)RG* module of *GreatSPN*: the main difference is that it performs already the first step of probabilistic paths reduction, so that the resulting (S)RG does not contain the intermediate probabilistic markings: large part of the code is reused from *WN(S)RG*, hence future improvements in *WN(S)RG* will be inherited. In particular, the SRG approach is applied to MDWN to reduce the number of generated states (and hence the size of the final MDP): it exploits the model symmetries, without introducing any approximation, thanks to the lumpability property of the MDP corresponding to the ordinary RG.

A detailed example of model specification and solution procedures are presented in Section 4 for a simple example; the state space reduction due to the exploitation of symmetries is shown on more complex examples in Section 5.

4. A Simple Example of MDPN and MDWN

In this section, we show how the *MDWNsolver* works on a simple example; more complex examples are discussed in the next section. Let us consider a system with two identical components, that can be in service (UP) or out of service (DOWN), and a centralized recovery system (decision maker), that can apply different repair policies. The recovery system must decide whether a given down component must be assigned a repair resource (to restore it to the UP state) or not. We consider the case where there is only one repair resource, so that the components cannot be repaired in parallel. The goal of the study is to find the optimal strategy that reduces the costs incurred by the system when the components break down: a penalty ($C_{penalty}$) is paid at each time unit if both components are down, moreover each time a repair activity starts, a repair cost (C_{repair}) is charged.

Figures 2 and 3 show an MDPN model for this system. In particular the former shows the probabilistic behavior of the two identical components; while the latter shows the possible actions of the centralized recovery system (assigning or not the repair resources to DOWN components).

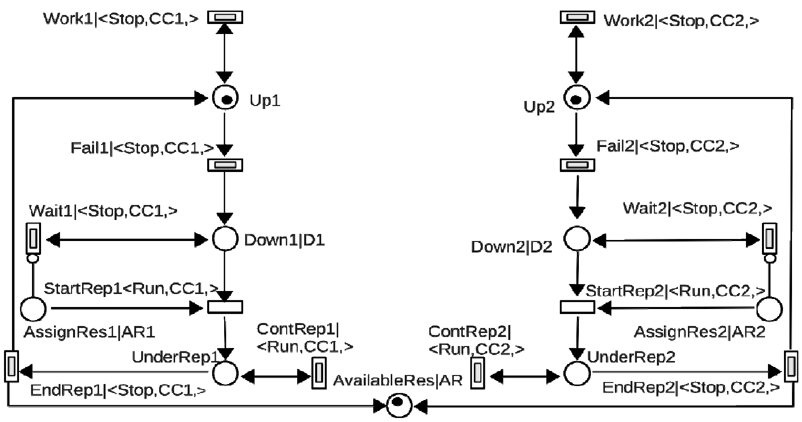

Figure 2: Example of MDPN Probabilistic Net

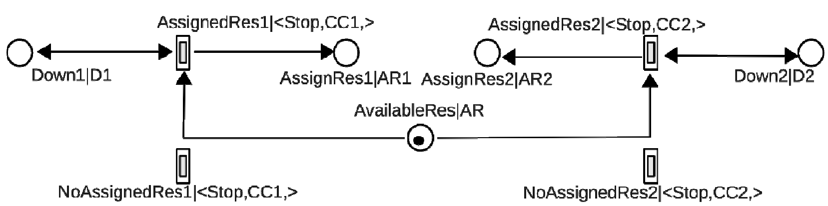

Figure 3: Example of MDPN non-Deterministic Net

The component lists are specified through the *GreatSPN* GUI by defining appropriate parameters with reserved names: a two letters prefix distinguishes among controllable (CC), non controllable (NC) and global (GL) components (in the example CC1 and CC2 are defined). By properly annotating the model, *stop* and *run* transitions can be identified, and the components involved in each firing are specified: these annotations are integrated in the "tag" attribute of transitions, concatenated to the transition name after the | separator. For example the annotation *StartRep2|<Run,CC2,>* means that transition *StartRep2* is a *run* transition involving only controllable component *CC2*, while the annotation *Fail1|<Stop,CC1,>* means that transition *Fail1* is a *stop* transition involving component *CC1*. Since places can be shared between the non deterministic and probabilistic subnet, these must be identified through a common label (concatenated with the place name) in the two models: in the example this is the case for places *AvailableRes*, $AssignRes_i$, and $Down_i$, $i=1,2$ identified as shared places by the suffixes $|AR$, $|AR_i$, $|D_i$ respectively. Places with these labels appear in both the subnets.

A token in place Up_i, $i=1,2$ means that component i is in service. The firing of transition $Fail_i$ with probability $1\text{-}P_{work}$ corresponds to component i failure and moves the token in $Down_i|D_i$. The repair of component i starts firing the *run* transition $StartRep_i$

when the decision maker has assigned a repair resource to that component putting a token into place $AssignRes_i|AR_i$. In each time unit an ongoing repair process can finish, represented by the firing of *stop* transition $EndRep_i$, with probability $1\text{-}P_{repair}$, or can go on, represented by the firing of *stop* transition $ContRep_i$, with probability P_{repair}.

In the non deterministic net the *stop* transitions $AssignedRes_i$ and $NoAssignedRes_i$ model the choice to activate or not the repair of a down component.

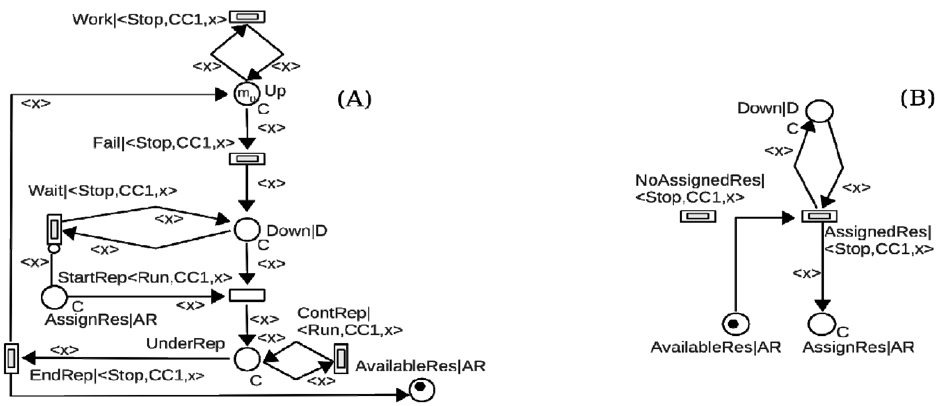

Figure 4: Example of MDWN Probabilistic Net (A) and Non-Deterministic Net (B)

The reward function associated with this model, defining the optimization problem, is:

T AssignedRes1 -1
T AssignedRes2 -1
F -100 (Down2|D1 = 1 && Down2|D2 = 1)

where, the first two items represent the cost associated with each repair action, while the last item expresses the penalty paid for the whole system being inactive (all components down). The *MDWNsolver* expects to find the reward function in a separate file, expressed according to a given grammar (see the manual in [16]).

When the system comprises sets of identical components, as is the case in the example, the MDWN formalism should be preferred since it allows a more compact and parametric definition of the model, since the behavior of each component type appears only once in the model. In Figure 4, the MDWN model for this system is depicted, where the list of controllable components contains only one element, *CC1*, that is associated with the color class *C* containing the identifiers of the two identical components. The annotations of the MDWN models are a bit more complex because it is required to specify the (tuple of) transition color elements (variables) that are used to identify one component within a set of identical ones. In the example of Figure 4, variable is *x* and the transition annotation must specify a component type followed by the variable(s) that are instantiated upon transition firing to select one specific component of that type; *e.g.*, the tag *StartRep|<Run,CC1,x,>* denotes a *run* transition involving component *x* among the components of type *CC1*. Special annotations can be used to associate more than one component in the same class with a given transition.

From the two models, we can derive an MDP and solve it, activating the sequence of modules described in Section 2. For each MDP state, the corresponding optimal action is reported as a sequence of non deterministic transition instances (*e.g., NoAssignedRes(a1); NoAssignedRes(a2)*). The size of the final MDP in general is smaller when the MDWN model is used, due to the exploitation of symmetries (SRG). The gain increases with the cardinality of the classes of similarly behaving components. As a consequence, the optimal strategy encoding is more compact and parametric (expressed using the symbolic markings and symbolic transition instances notation, representing equivalence classes of states and transitions). For instance, symbolic action *NoAssignedRes(C1); AssignRes(C2)* in a state where place *Up* contains <*C2*> and place *Down* contains <*C1*> represents the decision of assigning the repair resource to the component that is *Down*: different assignments of actual component identifiers to parameters *C1* and *C2* allow to obtain specific states and corresponding optimal action.

5. Interesting Applications of MDPN and MDWN

In this section, we present some interesting MDPN/MDWN application examples, giving a flavor of the type of optimization problems that can be dealt with this formalism, and discussing the model sizes that the tool can currently manage.

The first example, presented in [2], concerns a Wireless Sensor Network (WSN) monitoring system, that has to track a moving object within a building composed of *F* floors; each floor is partitioned in *Z* zones, each containing a fixed number *S* of sensors.

In this context, the MDWN was used to find an optimal trade off between the power consumption and the object tracking reliability; the power saving was achieved by periodically powering off some of the nodes for a given time interval (up to *C* time units long). The cost function to be optimized includes both the penalty due to losing track of the monitored object, and the cost of battery consumption; the possible non deterministic actions correspond to the choice of a set of nodes to be powered off and the respective sleeping time. The number of states is quite large, even for a relatively small system: to mitigate the complexity, the optimization problem has been solved on several simplified models, each representing only one floor in details; the computed optimal power management strategy has been simulated on a complete and more detailed model, to estimate the interesting performability measures, including energy consumption.

Table 1: The State Space Size of the WSN Monitoring Model in [2], where S is the number of sensors/zone, Z the number of zones/floor, C the maximum sleep time; number of floors F=3.

	MDWN			MDP (SRG)	
S,Z,C	**|RG|**	**|SRG|**	**$Time_{SRG}$**	**States**	**Time**
2,3,1	19,253	6,356	5s	144	0s
2,3,2	80,272	24,475	56s	380	5s
2,3,3	229,661	67,001	75s	825	26s
2,3,4	527,768	149,708	341s	1,575	6m
2,3,5	1,050,757	292,324	609s	2,744	20m
3,3,1	920,981	55,508	119s	420	6s
3,3,2	7,818,304	379,840	992s	1,600	17m
3,3,3	37,737,589	1,623,725	52m	4,725	34m
4,3,1	45,246,989	345,200	862s	975	201s

Table 1 shows the state space size and solution time as a function of the system parameters (*S, Z, C*) for a three floors model: the solution is feasible only for a limited number of sensors. In details, the first column reports the experiment parameters, the second, third and fourth columns report the number of ordinary states (RG size – derived from the SRG, not from direct computation) and symbolic states (SRG size), and the SRG generation time. The last two columns show the number of states of the reduced MDP and its generation and solution time. The results shown in this table shows the effectiveness of the SRG method in mitigating the state space explosion: a good level of reduction is achieved (*e.g.,* for case 4,3,1, the reduction factor (|RG|/|SRG|) is 131), moreover the SRG growth is smoother than the RG one (*e.g.,* moving from configuration 2,3,3 to 3,3,3 the SRG size grows by a factor 24 while the RG by factor 164).

Another interesting application of *MDWNsolver* is the computation of the optimal repair policy of systems specified by means of Non deterministic Repairable Fault Trees (NdRFT) or Parametric NdRFT (ParNdRFT), indeed an NdRFT/ParNdRFT model can be automatically translated into an MDPN/MDWN [4,6].

Here we present a model inspired to the *Multiprocessors system* in [8]. The system structure is shown Figure 5 (top left): it comprises two parts: the disk access (*DA*) and the CPU-Memory (*CM*) subsystem. The former unit is composed by two disks *D1, D2* in mirroring (RAID-1) and a bus (DBUS); while the latter unit comprises two processing units: *PU1* and *PU2*. Each processing unit includes a processor *Pi* and three redundant banks of local memory *Mi1-3*. Moreover, the two processing units share a global memory *SM* composed by two redundant memory banks *R1, R2*.

Figure 5 (right) shows, the NdRFT model for this system: the Fault Tree structure represents the Boolean function specifying which combinations of basic fault events (leaves) lead to the fault of each subsystems (internal nodes) and of the whole system (root - TE). In particular, the system (TE) fails if the *DA* or the *CM* subsystem fails. The *DA* fails if both the disks fail or the bus fails; while the *CM* fails only if both *PU1* and *PU2* fail. Each *PUi* fails if its processor or all its local memory banks and the global memory fail. Finally *SM* fails if both memory banks are not accessible (due to a faulty memory or bus).

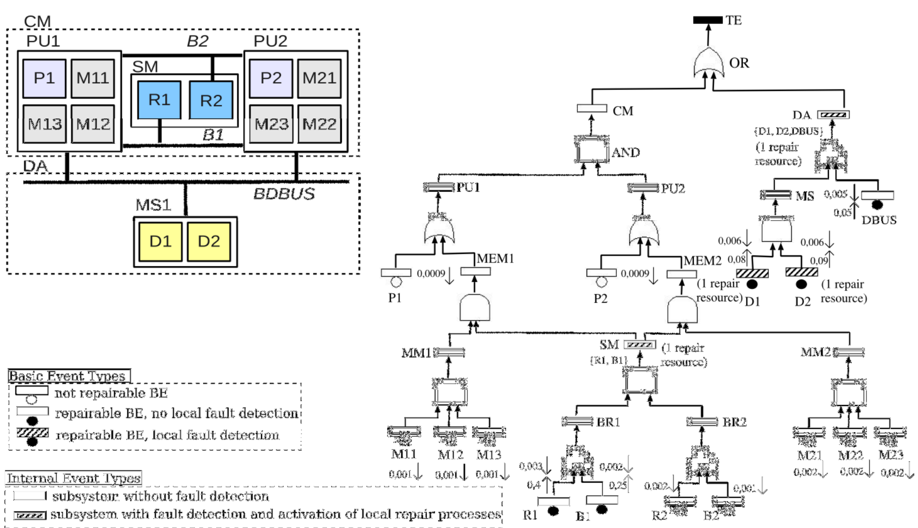

Figure 5: Example of NdRFT for a Multiprocessors System

The NdRFT model includes information on the fault rates (downward arrows) and on the possible repair actions that can be performed on the system components, and their rates (upward arrows): five basic components of the multiprocessors system can be repaired: *R1, B1, D1, D2, DBUS*. Their repair process can be activated either upon detection of an *SM fault* (*R1* and *B1*), or when a fault is detected in *DA*, (*D1 and D2* or *DBUS*), but also immediately when a fault is detected in a disk D_i. In our case study we suppose that only one repair resource is available and only one resource is required to perform each repair process.

In [4,6], it has been shown how an NdRFT can be automatically translated into an MDPN, where the cost function may include both the cost for the system (or subsystem) being down per time unit, and the repair cost. The dashed part at the bottom left of Figure 1 shows the software components that allow to design the NdRFT model (DrawNet GUI) and to translate it into an MDPN. The PN^{pr} and PN^{nd} subnets resulting from the translation of the multiprocessor NdRFT have 26 places 24 transitions overall; the PN^{pr} subnet models the system components behavior, while the PN^{nd} subnet represents the choice of which failed component has to be repaired at any time. For this model we have computed the optimal repair policy that minimizes the *TE* probability at time *t* (defining a constant positive cost per time unit for all the states where the whole system has failed, and no repair cost). The RG of the MDPN model obtained from the NdRFT has 586.826 states and it has been generated in 88 seconds, while the underlying MDP has 8.875 states and it has been generated and solved in 11 minutes (Intel Centrino Duo 2.4GHz, 2GiB RAM).

The computed optimal repair policy is not trivial even if the system has only five repairable components, since when more repairable components have failed, their repair order must be dynamically chosen according to the whole system state.
The optimal repair policy is shown in Table 2; where the first three columns represent the state of subsystems *CM, DA,* and *SM*, while the last column shows the corresponding optimal repair order. For instance if all subsystems have failed then the optimal repair order is *B1, R1, DBUS, D1, D2*, while if only *CM* is working then the optimal repair order is *DBUS, D1, B1, R1, D2*.

Table 2: The Repair Order corresponding to the Optimal Repair Policy

CM	DA	SM	Repair Order
Working	Failed	Failed/Working	DBUS,D1,D2,B1,R1
Failed/Working	Working	Failed	B1,R1,D1,D2
Working	Failed	Failed	DBUS,D1,B1,R1,D2
Failed	Failed	Failed	B1,R1,DBUS,D1,D2

In order to illustrate the performance of the optimal repair strategy we have computed the corresponding *TE* probability at time *t* solving the DTMC obtained from the MDP by fixing the action to take in every state according to the computed optimal strategy and we have compared it with that obtained using the following state independent repair strategies: 1) always repair first all the failed components in subsystem *CM*; 2) always repair first all the failed components in subsystem *DA*.

The obtained *TE* probabilities at time *t* with $400 \le t < 10000$ are plotted in Figure 6; as expected the curve representing the *TE* probability associated with the optimal strategy lays below those obtained when applying the state independent repair strategies.

It is interesting to observe that despite the multiprocessor model is structurally symmetric, the symmetry is not reflected in the failure and repair rates, as a consequence in this case, it is not possible to apply the SRG state space reduction method.

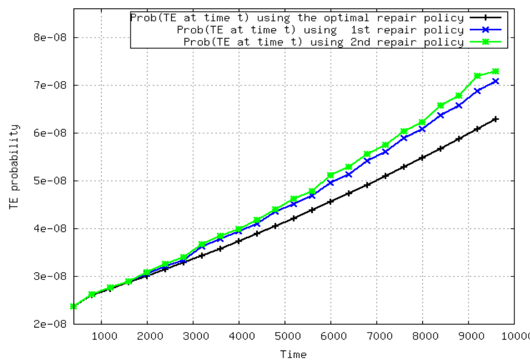

Figure 6: TE Probability at Time t for Different Repair Strategies

When instead also the fault/repair rates are uniform for replicated components, the SRG technique can be applied (the ParNdRFT formalism has been defined to represent in a compact and parametric form systems with symmetric structure and rates). In [6] an MDWN model automatically generated from a ParNdRFT model is illustrated: it represents an Active Heat Rejection System composed by a parametric number of thermal units, each composed by one source and one heat component. Each thermal unit belongs to one of three types (U1, U2 or U3) that have different parameters concerning fault occurrence probability and repair costs, and different possible repair actions. The failure of a thermal unit occurs when its source or its heat component fail; while the whole system fails when all its thermal units fail. The MDWN model of this example has been used to compute the repair strategy minimizing the probability of a system fault at time *t*.

Table 3 shows the state space size and solution time for this MDWN model, as a function of the number of thermal units for each type. The first column shows the experiment parameters, while the following two groups of four columns refer to the RG- versus SRG approach. For each approach the state space size, its generation time, the corresponding MDP size and its generation and solution time are reported.

Table 3: State Space Size and Computation Time of the Active Heat Rejection System model used in [4] with varying the number of sub-components of types U1,U2,U3.

	RG approach				SRG approach			
|U1|,|U2|,|U3|	**|RG|**	**$Time_{RG}$**	**$\|MDP_{RG}\|$**	**$Time_{MDP}$**	**|SRG|**	**$Time_{SRG}$**	**$\|MDP_{SRG}\|$**	**$Time_{MDP}$**
1,1,1	3,189	0s	389	0s	1,572	1s	389	0s
2,1,1	35,555	5s	937	5s	15,246	47s	579	0s
2,2,1	453,257	230s	7,754	11m	228,917	168s	3,143	4s
2,2,2	2,919,999	67m	32,558	2h	784,945	200s	16,222	3m
2,2,3	83,524,010	---	---	---	10,280,241	5h	52.271	2h

6. Conclusion

In this paper we have presented the *MDWNsolver* framework, able to generate an MDP from an MDPN/MDWN specification: this contribution extends an earlier two pages communication [3]; *w.r.t.* that prototype several optimizations on the solver have significantly improved its performance. The advantage of the proposed *MDWNsolver* is the possibility to express in a quite easy way MDP models using a high level language, supporting a component based specification with the possibility to put in evidence and exploit symmetries, and a way of specifying multi-step actions (composed of component-oriented sub-actions) and multi step probabilistic evolution. To the best of our knowledge the other tools supporting a high level specification language for MDPs do not include all the above mentioned features: for instance PRISM [11] allows to specify a system by composition of modules (resembling our notion of component), but at each time step there can be either a synchronized action of a subset of modules, followed by a one-step probabilistic state change, or an action can be performed by only one module, again followed by a probabilistic state change, so that modeling the concurrent evolution of independent components within each time step requires some effort. The experiments performed up to now with the *MDWNsolver* in different application domains have shown that the current prototype can handle models with RG or SRG of up to 10,000,000 states: in all the considered cases the resulting MDP structure had less than 55,000 states (which is also a limit to find the optimal strategy without running out of memory with the current solver). These are at the moment the limit sizes that can be managed in reasonable time, and without running out of memory, on a machine with an Intel Core Duo T7500 2.20 GHz processor and 2GiB RAM, with Linux. Although the time required to generate and solve the MDP depends on several factors (not only the number of states) the time required in our experiments to generate the MDP from the MDPN/MDWN model and solve it, were comprised between a few seconds for models with a (S)RG of a few thousands states and an MDP of a few hundreds of states, to some hours, for models with a (S)RG of several millions of states and an MDP of up to fifty thousand states. The *MDWNsolver* is distributed with the *GreatSPN* tool: it can be downloaded from [15].

References

[1] Alur, R., and T. Henzinger. *Reactive Modules*. Formal Methods in System Design 1999; 15(1):7–48.

[2] Beccuti, M., D. Codetta-Raiteri, and G. Franceschinis. *Multiple Abstraction Levels in Performance Analysis of WSN Monitoring Systems.* In Proc. of the WSNperf (Satellite Workshop of VALUETOOLS09), Pisa, Italy, October 2009. ICST.

[3] Beccuti, M., D. Codetta-Raiteri, G. Franceschinis, and S. Haddad. *A Framework to Design and Solve Markov Decision Well-formed Net models.* In Proc. of the 4th IEEE Int. Conf. on Quantitative Evaluation of Systems (QEST'07), 165–166, Edinburgh, Scotland, UK, September 2007. IEEE Computer Society Press.

[4] Beccuti, M., D. Codetta-Raiteri, G. Franceschinis, and S. Haddad. *Non Deterministic Repairable Fault Trees for Computing Optimal Repair Strategy*. In Proc. of the 3rd Int. Conf. on Performance Evaluation, Methodologies and Tools (VALUETOOLS'08), Athens, Greece, October 2008. ICST.

[5] Beccuti, M., G. Franceschinis, and S. Haddad. *Markov Decision Petri Net and Markov Decision Well-formed Net Formalisms.* Proc of the 28th Int. Conference on Applications and Theory of Petri Nets and other Models of Concurrency. LNCS Vol. 4546, 43-62. 2007.

[6] Beccuti, M., D. Codetta-Raiteri, G. Franceschinis and S. Haddad. *Parametric NdRFT for the Derivation of Optimal Repair Strategies*. Proceedings of the 39th International Conference on Dependable Systems and Networks (DSN-2009), Estoril, Lisbon, Portugal, 29 June-2 July 2009. IEEE Computer Society Press.

[7] Chiola, G., C. Dutheillet, G. Franceschinis, and S. Haddad. *Stochastic Well-formed Coloured Nets for Symmetric Modelling Applications*. IEEE Transactions on Computers, Nov. 1993; 42(11):1343–1360.

[8] Codetta-Raiteri, D. *Extended Fault Trees Analysis supported by Stochastic Petri Nets*. Ph.D. Thesis, Univ. degli Studi di Torino, Torino, Italia, 2005.
[9] de Alfaro, L. *Stochastic Transition Systems*. In 9th Int. Conf. on Concurrency Theory, LNCS Vol. 1466, 423–438. Springer, 1998.
[10] Dean, T., and M. P. Wellman. *Planning and Control*. Morgan Kaufmann, 1991.
[11] Hinton, A., M. Kwiatkowska, G. Norman, and D. Parker. *PRISM: A Tool for Automatic Verification of Probabilistic Systems.* In 12th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems, LNCS Vol. 3920, 441–444. Springer, 2006.
[12] Puterman, M.L. *Markov Decision Processes.* Discrete Stochastic Dynamic Programming, Wiley, Chichester, 2005.
[13] GraphMDP Web Page. `http://www.cert.fr/dcsd/cd/teichteil/.`
[14] Baarir, S., M. Beccuti, D. Cerotti, M. De Pierro, S. Donatelli and G. Franceschinis. *The GreatSPN Tool: Recent Enhancements. ACM Performance Evaluation Review Special Issue on Tools for Performance Evaluation*, 2009; 36:(4): 4–9.
[15] GreatSPN Web Page: `http://www.di.unito.it/~greatspn`
[16] MDWNsolver Web page: `http://www.di.unito.it/~greatspn/MDWNsolver/`

Marco Beccuti received Ph.D. degree in Computer Science in 2008 from University of Torino in ``cotutela'' with the Universite' Paris Dauphine. From January 2008 to December 2008, he was a research assistant at Consorzio Nazionale Interuniversitario per le Telecomunicazioni (CNIT). He is currently temporary researcher at University of Torino and member of Consorzio Nazionale Interuniversitario per le Telecomunicazioni (CNIT). His main research interests are in the fields of Petri net and Markov Decision theory and applications, performance evaluation, parallel discrete-event simulation, parallel architectures, and he is the (co-)author of more than twenty papers published in proceedings or journals.

Giuliana Franceschinis received the Ph.D. degree in computer science from the University of Torino in 1992. From 1992 to 1998 she was an assistant professor at the Computer Science Department of the University of Torino, Italy. From 1998 to 2002, she was an associate professor at the Università del Piemonte Orientale in Alessandria, Italy; since 2002 she is full professor in the same University. She teaches computer architecture, operating systems, and simulation. Her current research interests are in the areas of dependability and performance evaluation of computer and communication systems and in stochastic Petri nets theory and applications. She has published several papers in international conference proceedings and journals, and is a coauthor of the book "Modelling with Generalized Stochastic Petri Nets". She participates in several international research cooperation initiatives and projects.

Serge Haddad is a former student at the Ecole Normale Supérieure de Cachan. He received the M.Sc. degree in mathematics in 1977 from the University of Orsay and the M.Sc. and Ph.D. degrees in computer science in 1983 and 1987, respectively, from the University of Paris 6. He is currently a full professor at the Ecole Normale Supérieure de Cachan. His research interests include quantitative verification with emphasis on timed and stochastic systems and applications to software engineering.