Efficient solution of extended Multiple-Phased Systems

Elvio Gilberto Amparore
Dipartimento di Informatica
Università di Torino
Torino, Italy
amparore@di.unito.it

Susanna Donatelli
Dipartimento di Informatica
Università di Torino
Torino, Italy
donatelli@di.unito.it

ABSTRACT

Multiple-Phased Systems (MPS) are systems whose behaviour can be split in a set of successive periods, called phases. We concentrate on the Phase Petri nets (PPN) formalism for the representation of state-based MPS, whose state space is a Markov Regenerative Process (MRP). MPS have been traditionally evaluated using an ad-hoc method. In this paper we show how the recently developed *Component Method* for general MRP deals with MPS. The application of this method also allows for more general MPS system, namely where the phases can be interrupted or influenced by the behaviour of the modeled system The advantages of this approach is assessed on a Scheduled Maintenance Systems model using two tools, DEEM and GreatSPN.

1. INTRODUCTION AND STATE OF ART

Multiple-Phased Sytems (MPS) [1] are systems whose behaviour can be split in a set of successive periods, called phases. Also called Phased Mission System (PMS) in other works, as in [2], since they can easily describe systems in which the behaviour is described as a mission, structured into multiple *phases*, each described by a different duration, system configuration, desired task, etc. MPS have been shown to be useful also in modelling systems with scheduled maintenance [3] (Scheduled Maintenance Systems - SMS). In a MPS, the standard question is to compute the reliability of the system, i.e. the probability that the system survives the mission, but optimization also plays a role [4], especially for SMS to determine the best maintenance policy, as well as sensitivity analysis [3, 5] to allow to reason about the structure of the mission and their parameters.

There has been a significant amount of work, especially in the late nineties and also more recently, on MPS. Different techniques have been used for modelling and solving these systems, from combinatorial methods like reliability blocks and fault trees to state-based techniques. Combinatorial approaches based on reliability blocks and fault trees have been reported for example in [6, 7]. State-based approaches build Markovian systems either through an ad-hoc language as in EHARP [6] or through a high level modelling formalism as Stochastic Petri Nets as in [8]. It is well known what are the relative advantages and disadvantages of combinatorial methods over state-based ones, but when the objective of the MPS analysis goes beyond the computation of the reliability of the system at the end of the mission, state-based approaches can provide a plus, like computing the probability of the system states at time t or associate a reward structure to the system states.

The use of a high level formalism based on Petri nets, as promoted in [8], seems to have a good trade-off between the modelling power (class of systems that can be specified) and the amount of human intervention in the definition of the model, as claimed in the same paper.

In this paper we start from the class of MPS generated by Stochastic Petri Nets as in [2]: the phases are described as a Petri net (the *Phase Net*) which only includes deterministic transitions and immediate ones, with exactly one deterministic transition enabled in each phase; And the system behaviour is again a Petri nets (the *System Net*) that only includes exponential and immediate transitions. Transition enabling and firing rates in the System Net may depend on the marking of the Phase Net and this permits a description of the system that is compact (a single net) but that is able to describe a behaviour that may differ from phase to phase.

An MPS net is indeed a *Deterministic and Stochastic Petri Net* (DSPN) [9, 10], or more precisely, a non-ergodic DSPN, since phases (the mission) always come to an end. DSPNs have the requirement that at most one deterministic transition is enabled in any one state, which is indeed true for MPS. The stochastic process of a DSPN is a *Markov Regenerative Process* (MRP) [11] which has been widely used in the modelling community since it can describe more complex behaviours than a continuous time Markov chain, while still allowing an analytical solution.

A MRP is a stochastic process defined by a sequence of time instants called *renewal times* in which the process loses its memory, i.e. the age of non-exponential (general) events is 0. The behaviour between these points is then described by a time-limited stochastic process, and we consider MRP in which the time limited stochastic process is a CTMC. MRPs have been studied extensively in the past [10, 12], and many solid analysis techniques exists.

The limitations imposed on the Phase net allow Mura et al. to show in [2] that the solution of the MPS can be decomposed in a sequence of transient solutions of Markov chains. The DEEM tool [1] implements this method. Although not

VALUETOOLS 2016, October 25-28, Taormina, Italy

DOI 10.4108/eai.25-10-2016.2267064

recent, DEEM can still be considered the state of the art tool for MPS definition and solution.

In [13] a component-based solution technique for non-ergodic MRP has been defined. The MRP process is decomposed into smaller components that are solved in isolation with the cheapest possible technique. Indeed there are cases in which the solution of a component is equivalent to the transient solution of a CTMC. The method has been implemented inside the GreatSPN tool [14] as a solver for non-ergodic DSPNs and it is also part of the model-checker MC4CSLTA [15] of the stochastic logic CSL^{TA} [16].

This paper shows that the Component Method applied to MPS nets identifies the same components and has the same time complexity as the ad-hoc techniques developed for MPS and implemented in DEEM. Moreover we show that the Component Method can solve an extended class of MPS, that we shall term X-MPS. This extension has a practical relevance since it lifts many of the limitation imposed on MPS, for example an event in the system net may now stop and/or restart a phase, and the Phases net can also include exponential transitions.

On the other side the memory requirement of the DEEM solver is smaller, since the state spaces of the single phases are built in isolation and the state space of phase i is built, used and destroyed before the one for the $i+1$ phase. Basically DEEM builds the state space by chunks, only when needed, in a sort of on-the-fly solution. The Component Method is a general technique for non-ergodic MRPs and DSPNs, and therefore cannot build the system phase by phase since it does not know about phases.

An important aspect of DEEM is that it is specifically designed for MPS, so it has a lot of interesting features that makes the modeller activity simpler (like the separation of the description of the phases and of the basic system, the heavy use of marking dependencies and the rich reward structure). The result of this investigation indicates that a good way to follow is the integration of the best MPS characteristics of DEEM and GreatSPN in a single tool.

The paper develops as follows: Section 2 defines the necessary background on the Component Method for MRP solution (in 2.1) and on MPS and the DEEM tool (in 2.2). Section 3 shows the application of the Component Method to MPS, enlightening the extension to the current MPS DEEM models in Section 4. Section 5 studies the efficacy of the proposed technique through a set of numerical experiments on a standard and extended MPS models. Finally Section 6 concludes the paper and outlines future possible extensions and integration activities.

2. BACKGROUND

2.1 Component based solution of MRP

We assume the reader is familiar with the definition of Markov renewal sequence and of (time-homogeneous) Markov regenerative process built on a Markov Renewal sequence. A full description can be found in [17] and in [10] for a definition and notation of MRP related to DSPN. In this paper we use German's notation as in [10] and [13]. In MRP the process behaviour among regeneration points is a discrete-time Markov chain, called the *embedded Markov chain* (EMC), while the process behaviour between two regeneration points is described by a continuous-time process, called the *subordinated process*. In this paper we consider only the class of MRP where the subordinated process is a CTMC.

A MRP can be represented as a discrete event system (like in [18]) with a finite state space, where in each state a general event g is taken from a set G. As the time flows, the age of g being enabled is kept, until either g fires ($\mathbf{\Delta}$ event), or a Markovian transition, concurrent with g, fires. Markovian events may actually disable g (preemptive event, or $\bar{\mathbf{Q}}$ event), clearing its age, or keep g running with its accumulated age (non-preemptive event, or $\mathbf{Q}$ event). Matrix $\mathbf{Q}$ accounts for the rates of the exponential transitions whose firing does not disable any general (deterministic) transition; Matrix $\bar{\mathbf{Q}}$ accounts for the rates of the exponential transitions whose firing disables a general (or deterministic) one; And matrix $\mathbf{\Delta}$ has, for each entry $\mathbf{\Delta}_{i,j}$, the probability of ending in state j when the general transition fires in state i.

The *embedded Markov chain* (EMC) $\mathbf{P}$ is defined in terms of the $\mathbf{Q}, \bar{\mathbf{Q}}$, and $\mathbf{\Delta}$ in a standard manner (see [10] for example), but even if the three matrices are sparse, matrix $\mathbf{P}$ is usually dense and expensive to compute, due to the matrix exponential terms. This problem has been solved in [10] with the *matrix-free* technique (actually $\mathbf{P}$-free), based on the idea that $\mathbf{P}$ can be substituted by a function of the $\mathbf{Q}, \bar{\mathbf{Q}}$, and $\mathbf{\Delta}$ (sparse) matrices of the MRP. These functions are used in vector×matrix products with $\mathbf{P}$ so that they can be computed without the need of constructing and storing $\mathbf{P}$.

A non-ergodic EMC allows to distinguish transient and recurrent states, and specialized solution methods can be devised to compute the probability distribution of recurrent states. The work in [13, 19] introduces an efficient steady-state solution for non-ergodic MRPs, called *Component Method*, which computes the outgoing probability flow from transient to recurrent subsets of states, called *components*. In graph terms components corresponds to the union of one or more strongly connected components (SCC) of the MRP state space and they must satisfy the requirement that the component graph (the condensed graph in which each component is condensed in a single node) is a directly acyclic graph (DAG). The method is also given in matrix-free terms, so the components and their characteristic matrices are computed directly on the state space and on the $\mathbf{Q}, \bar{\mathbf{Q}}$, and $\mathbf{\Delta}$ matrices of the MRP, and there is no need to compute the expensive and dense matrix $\mathbf{P}$.

The basic idea of the Component Method is readily explained unrealistically assuming that matrix $\mathbf{P}$ is available. If the MRP is non-ergodic it is indeed possible to rearrange the order of its states so that the EMC matrix $\mathbf{P}$ is in upper-triangular form (the *reducible normal form*, or RNF):

$$\mathbf{P} = \begin{bmatrix} \mathbf{T}_1 & & & & \mathbf{F}_1 \\ & \ddots & & & \ddots \\ & & \mathbf{T}_k & & \mathbf{F}_k \\ & & & \mathbf{R}_{k+1} & \\ & & & & \ddots \\ & & & & \mathbf{R}_m \end{bmatrix} \tag{1}$$

Matrix $\mathbf{P}$ has $k \geq 0$ transient subsets and $(m-k)$ recurrent subsets of states, with $m > k$. Let $\mathcal{S}_i \subseteq \mathcal{S}$ be the set of states in subset i, hereafter called the *component* i. Alternatively, the upper triangular form of $\mathbf{P}$ can be seen as if the $\mathcal{S}_i$ subsets form a DAG of components. A standard way to build an RNF for a non-ergodic process is to take as $\mathcal{S}_i$ sets the set of *strongly connected components* (SCC) [20] of the graph built by considering $\mathbf{P}$ as an adjacency matrix. Transient

SCCs are the $\mathcal{S}_i$ for $i \leq k$, and bottom SCC (BSCC) are the $\mathcal{S}_i$ for the recurrent classes ($k < i \leq m$). Indeed SCCs are the finest partition of $\mathbf{P}$ that result in an acyclic set of components. A convenient numbering of the $\mathcal{S}_i$ can then be found visiting in topological order the component graph.

When $\mathbf{P}$ is in RNF, the steady-state probability of being in the recurrent states can be computed using the outgoing probability vectors $\boldsymbol{\mu}_i$. The *outgoing probability* vector $\boldsymbol{\mu}_i$ gives for each state $s \in (\mathcal{S} \setminus \mathcal{S}_i)$ the probability of reaching s in one jump while leaving $\mathcal{S}_i$. Vector $\boldsymbol{\mu}_i$ is given by:

$$\boldsymbol{\mu}_i = \Big(\boldsymbol{\alpha}_i + \sum_{j<i}(\mathbf{I}_i \cdot \boldsymbol{\mu}_j)\Big) \cdot (\mathbf{I} - \mathbf{T}_i)^{-1} \cdot \mathbf{F}_i, \quad i \leq k \qquad (2)$$

where $\boldsymbol{\alpha}_i$ is the vector of initial probability for the states in $\mathcal{S}_i$. We adopt the *matrix filter* notation of [10] where: $\mathbf{I}^g$ is the matrix derived from the identity matrix of size $|\mathcal{S}|$ where each row corresponding to states that do not enable g are set to zero; And $\mathbf{I}^E$ is the same for states that do not enable any general transition (exponential states). Since matrix inversion is usually expensive, a product of a generic vector $\mathbf{u}$ with $(\mathbf{I} - \mathbf{T}_i)^{-1}$ can be reformulated as a linear equations system $\mathbf{x} \cdot (\mathbf{I} - \mathbf{T}_i) = \mathbf{u}$. This system can be computed iteratively using vector×matrix products with $\mathbf{T}_i$. Each vectors $\boldsymbol{\mu}_i$ may depend on the previous $(i-1)$ outgoing probability vectors, implicitly defining a computational order. Given the $\boldsymbol{\mu}_i$ vectors, the steady state probability of the recurrent subsets $\mathcal{S}_i$ is given by:

$$\boldsymbol{\pi}_i = \Big(\boldsymbol{\alpha}_i + \sum_{j=1}^{k}(\mathbf{I}_i \cdot \boldsymbol{\mu}_j)\Big) \cdot \lim_{n\to\infty}(\mathbf{R}_i)^n, \quad k < i \leq m \qquad (3)$$

The Component Method computes first equation (2) for all transient components, taken in an order that respects the condition $j < i$ of the formula, and then computes the probability for the recurrent subsets based on equation (3).

When the $\mathbf{P}$ matrix is available in RNF, the computation of (2) and (3) is straightforward.

In [13] the matrix-free method is generalized for the case of the reducible normal form of Eq. 1 giving rise to a *matrix-free component method*, Component Method for short. This generalization provides: 1) a derivation of the m subsets $\mathcal{S}_i$ (components) which is based only on $\mathbf{Q}$, $\bar{\mathbf{Q}}$ and $\boldsymbol{\Delta}$; 2) the matrix-free form of the sub-terms $\mathbf{T}_i$, $\mathbf{F}_i$ and $\mathbf{R}_i$, to be used in Eq. (2) and (3). The computation of the products of a generic vector $\mathbf{u}$ with the matrix-free form of $\mathbf{T}_i$ and $\mathbf{F}_i$ is given by:

$$\mathbf{u}\mathbf{T}_i = \mathbf{I}_i \cdot (\mathbf{a}_{\widehat{i}}(\mathbf{u}) + \mathbf{b}_{\widehat{i}}(\mathbf{u}) + \mathbf{c}_i(\mathbf{u})) \qquad (4)$$

$$\mathbf{u}\mathbf{F}_i = (\mathbf{I} - \mathbf{I}_i) \cdot (\mathbf{a}_{\widehat{i}}(\mathbf{u}) + \mathbf{b}_{\widehat{i}}(\mathbf{u}) + \mathbf{c}_i(\mathbf{u})) \qquad (5)$$

where $\mathbf{I}_i$ is a matrix derived from $\mathbf{I}$ where rows of states not in $\mathcal{S}_i$ are zeroed, $\mathbf{I}_{\widehat{i}}$ is the same for the augmented set $\widehat{\mathcal{S}}_i$, and $\boldsymbol{\Delta}_{\widehat{i}} = \mathbf{I}_{\widehat{i}} \cdot \boldsymbol{\Delta}$, $\bar{\mathbf{Q}}_{\widehat{i}} = \mathbf{I}_{\widehat{i}} \cdot \bar{\mathbf{Q}}$. The *augmented set* $\widehat{\mathcal{S}}_i$ of component i is the closure over $\mathbf{Q}$ of the states $\mathcal{S}_i$ of components i, that is to say the augmented set of a component is made by the states of the components plus all the states that can be reached through Markovian events that do not disable any general (that is to say all states that are reachable before the next regeneration point).

The vector terms $\mathbf{a}_{\widehat{i}}(\mathbf{u})$, $\mathbf{b}_{\widehat{i}}(\mathbf{u})$ and $\mathbf{c}_i(\mathbf{u})$ are:

$$\mathbf{a}_{\widehat{i}}(\mathbf{u}) = \mathbf{u} \cdot \Big(\sum_{g\in G} \mathbf{I}^g_{\widehat{i}} \cdot \mathrm{e}^{\mathbf{Q}_{\widehat{i}}\,\delta^g}\Big) \cdot \boldsymbol{\Delta}_{\widehat{i}}$$

$$\mathbf{b}_{\widehat{i}}(\mathbf{u}) = \mathbf{u} \cdot \Big(\sum_{g\in G} \mathbf{I}^g_{\widehat{i}} \cdot \int_0^{\delta^g} \mathrm{e}^{\mathbf{Q}_{\widehat{i}}x}\,\mathrm{d}x\Big) \cdot \bar{\mathbf{Q}}_{\widehat{i}}$$

$$\mathbf{c}_i(\mathbf{u}) = \mathbf{u} \cdot \Big(\mathbf{I}^E_i - \mathrm{diag}^{-1}(\mathbf{Q}^E_i)\mathbf{Q}^E_i\Big)$$

with $\mathbf{Q}^E_i = \mathbf{I}_i \cdot \mathbf{Q}^E = \mathbf{I}_i \cdot \mathbf{I}^E \cdot \mathbf{Q}$. These terms describe how the process evolves between two regeneration points. Vector $\mathbf{a}_{\widehat{i}}(\mathbf{u})$ and $\mathbf{b}_{\widehat{i}}(\mathbf{u})$ are the probability distribution of the next regeneration state reached with the firing of the general event ($\mathbf{a}$), or with the preemption of it ($\mathbf{b}$). Vector $\mathbf{c}_i(\mathbf{u})$ is the probability distribution of the next regeneration state when there are no general events enabled in the starting state. Note that the computation of $\mathbf{a}$ and $\mathbf{b}$ on a subset $\mathcal{S}_i$ of states has to consider all the states in the augmented set $\widehat{\mathcal{S}}_i$. Eq. (4) and (5) assumes that the subsets $\{\mathcal{S}_i\}$ have been identified.

The advantage of working at the component level is not only the trivial one of solving many small models instead of a single much bigger one, but that the cheapest available solution for each component can be used. It is sometimes possible that the component's solution do not bear the full complexity of the solution of a MRP.

Although it is intuitive and usually true that solving many small matrices is less expensive than solving a big one, in the matrix-free Component Method this is not always true. Indeed as the solution of a single components may actually require to consider states that are not part of that components (due to the construction of the augmented set), so that often it is convenient to aggregate small components into a bigger one. The work in [19, 21] already identifies a need for aggregating the components into bigger ones, and often the performance of the algorithm depends on the number, size, and solution complexity of the components. The aggregation is defined through a set of rules, to decide which components can be aggregated together, and could be done using a greedy heuristic (fast) or integer linear programming (ILP) [22] (optimal but slower). The Component Method, including the component optimization and the greedy heuristics, have been implemented as part of the GreatSPN solver for non-ergodic DSPNs. GreatSPN [14] is a tool for Petri net definition and analysis developed mainly at the University of Torino during the last 30 years. GreatSPN has been recently renovated to include a new Java-based interface with colored and plain token game simulation, model checking for branching and stochastic logics, a new solver for DSPN, and additional facilities for model composition and for performing multiple experiments. GreatSPN, including the solution used in this paper, is available upon request visiting its web page [23].

2.2 Background: MPS and DEEM

In this paper we consider definition and notation of MPS in [1, 8], in which the mission is divided into m phases of deterministic duration δ_i, and the system behavior in each phase is described by a CTMC with state space $\mathcal{S}_i$ and rate matrix $\mathbf{Q}_i$. The MPS is defined using a restricted class of DSPN called *Phased Petri net* (PPN) [24].

In a PPN the system is described by two Petri nets [25]: a *phase net* (PhN), which defines the phase structure and

a *system net* (SN), which describes the stochastic behavior of the system components during each phase of the mission. Marking dependent rates and guards can be used to make the SN behaviour dependent from a specific phase, and the probability of immediate transitions in the PhN can depend from the marking of the SN: this allows the choice of the next phase to depend upon the state of the SN.

A number of limitations have been imposed on the PhN and SN and on their dependency, partly to provide a guideline to the modeller, and partly to allow for a decomposable solution [1,2]: *L1*) The PhN can only have deterministic and immediate transitions - no exponential delays are allowed, *L2*) the reachability graph of the PhN should be a DAG (directed acyclic graph), *L3*) each phase is described by a single tangible marking of the PhN which enables a single deterministic transition, *L4*) an event occurring in the SN cannot disable a phase (the firing of a transition in the SN cannot preempt a deterministic transition). Note that condition 2 implies that the whole stochastic process is non-ergodic.

Under the above limitations the work in [2] observes that the time at which the deterministic transitions of the PhN fire correspond to regeneration instants, and the markings after the firing are markings in which the system regenerates. Moreover since there is no preemption of the deterministic and the state space of the PhN is acyclic, it is possible to devise a decomposed solution approach, from regeneration point to regeneration point, approach that we explain in the following according to the schema described in [1].

The first step consists in generating the state space of the PhN. By definition each phase i is characterized by a single marking m_i and by a deterministic transition of delay δ_i, therefore it is possible to solve the whole system as follows (at the beginning $i = 0$):

- Compute the state space $\mathcal{S}_i$ of the SN when the PhN is in state m_i and build the associated rate matrix $\mathbf{Q}_i$
- Compute the transient solution at time δ_i of the CTMC specified by $\mathbf{Q}_i$ (which gives the distribution of the markings of the SN at the time phase i ends)
- Compute the $\mathbf{\Delta}_{i,j}$ matrices, the branching probability matrix of size $|\mathcal{S}_i| \times |\mathcal{S}_j|$ of going to the states of phase j at the end of phase i. This requires to build the state spaces of the SN for two successive phases
- The solution at time δ_i of $\mathcal{S}_i$ is mapped, through $\mathbf{\Delta}_{ij}$ to the initial probability of the successive components

The above sequence is repeated for successive phases until the target time t of the analysis has been reached. If t is greater than the sum of the duration of all the phases (complete duration of the mission) this corresponds to the distribution of the absorbing states of the system at the end of the mission. If t is smaller, let's say between the end of phase $i-1$ and i then $\mathbf{Q}_i$ is solved at time $t - t'$, where t' is the sum of all the duration of the previous phases.

The theory summarized above has been implemented inside the prototype tool DEEM [1]. Although not recent, DEEM can still be considered the state of the art tool for MPS definition and solution based on Petri nets. DEEM is not only a solver but also has a GUI for the definition of the PhN and of the SN, including facilities for the definition of marking dependent rates, transitions' guards and a reward structure.

For what concerns the time complexity, if there are N phases DEEM generates and computes the transient solution of N CTMCs. The space complexity is determined by the need of building N state spaces for the SN (one per phase) and the corresponding N CTMCs, and $N-1$ state spaces for pairs of successive phases (of size $|\mathcal{S}_i| + |\mathcal{S}_j|$) to build the $\mathbf{\Delta}_{i,j}$ matrices.

3. COMPONENT METHOD VS DEEM

From now on we use the MPS term to mean a MPS model defined through a DSPN that follows the PPN limitations *L1-L4*. Since a MPS is a non-ergodic DSPN, it makes sense to compare the solution approach of DEEM with the Component Method,. The comparison is organized as a set of questions for which to provide an answer. Given an MPS: *Q1*) do the two techniques consider the same components? *Q2*) do they solve the components with techniques of comparable complexity? *Q3*) do they use comparable amounts of memory?

A first step is to better understand the characteristics of the MRP produced by a MPS. If we consider the embedded Markov chain transition matrix $\mathbf{P}$ in RNF form, what are the block matrices involved? Since the state space of the PhN forms a DAG, we can certainly find a RNF form for the embedded Markov Chain transition matrix $\mathbf{P}$, in which the states of phase i corresponds to the subset i in the matrix of Eq. 1, as it was shown in [2] that the end of a phase corresponds to a regeneration point. For ease the notation each $\mathbf{F}_i$ matrix is split into multiple $\mathbf{F}_{i,j}$ submatrices, with $i < j \leq m$. We can then observe that:

$$\begin{aligned} \mathbf{T}_i &= \mathbf{0} \\ \mathbf{F}_{i,j} &= \mathrm{e}^{\mathbf{Q}_i \delta_i} \cdot \mathbf{\Delta}_{i,j} \\ \mathbf{R}_i &= \mathbf{I}_i - \mathrm{diag}^{-1}(\mathbf{Q}_i) \cdot \mathbf{Q}_i \end{aligned} \tag{6}$$

each $\mathbf{T}_i$ will be zero, since $\mathbf{T}_i$ is the submatrix of the probability of remaining in $\mathcal{S}_i$ at the end of the phase, but this is not allowed since the PhN is a DAG. The $\mathbf{F}_{i,j}$ matrices describe the probability of moving from the start of phase i to the start of phase j, and this involves the solution at time δ_i and the probability among phases (terms $\mathrm{e}^{\mathbf{Q}_i \delta_i}$ and $\mathbf{\Delta}_{i,j}$). The $\mathbf{R}_i$ terms corresponds to the recurrent terminal phases (phases with no successors), and the steady state solution can be applied. Note that, when DEEM solves an MPS at time t also the terminal phases are solved in transient.

A second step for the comparison is to recall the component classification of the Component Method and the associated computational costs when general transitions are restricted to deterministic ones. Table 1, re-elaborated from [13, Table 2], follows the classification of components in [22] and defines three component types.

Class C_E identifies components in which there is no state that enables a general transition. Their solution requires a fixed point iteration over the portion of the $\mathbf{Q}$ matrix that corresponds to $\mathcal{S}_i$. The $\mathbf{a}$ and $\mathbf{b}$ terms of (4) are zero, and a product with $(\mathbf{I}_i - \mathbf{T}_i)^{-1}$ has the same cost of a steady state solution of a CTMC.

Class C_g identifies components in which 1) there is a single deterministic g enabled; and 2) no preemption of g, if any, keeps the system inside the component. Their solution cost is that of a transient solution at time δ_g of a CTMC, over the augmented set $\widehat{\mathcal{S}}_i$. Since any preemptive Markovian or deterministic event takes the system out of the component, then $\mathbf{T}_i$ is empty, and $(\mathbf{I}_i - \mathbf{T}_i)^{-1}$ reduces to $\mathbf{I}_i$.

Finally class C_M identifies components in which either different general transitions are enabled (of course in different states) or there are preemptive events that keep the system inside the component. The process structure of the component is a MRP.

Coming back to the questions, to answer to *Q1* (do the two techniques consider the same components?) we should consider that DEEM generates the CTMC of each phase in isolation, doing a state space generation in which the marking of the PhN is kept fixed and equal to the marking that enables the deterministic transition of phase i. Let g be such a transition. In the Component Method the criteria behind the definition of an optimal partition of the MRP [13, 22] is that, starting from the initial definition of components as SCCs, two components are aggregated together only if they belong to the same complexity class and if the resulting complexity class is the same, given that the component graph after the aggregation is still a DAG. The component construction will indeed put together all states that enable g, whether they correspond to one or more SCCs, since all those SCC are of class C_g, and their union is still of class C_g, since no preemption of the deterministic transitions can take place in a MPS and the firing of the deterministic takes the system outside of all SCCs that enable g (since g is not enabled any longer).

The answer to *Q2* (time complexity) is straightforward given that there is a component per phase, and the component is of type C_g. The $\boldsymbol{\mu}_g$ vector of Eq. 2 is therefore computed with a transient solution at time δ_g, that is to say at the time the phase ends, as does the solver of DEEM.

The answer to *Q3* (memory) shows a difference in the maximum amount of storage used. Both techniques computes the same $\mathbf{Q}_i$ and $\boldsymbol{\Delta}_{ij}$ matrices, but the Component Method builds all of them beforehand, to be able to compute the components, while DEEM can exploit the knowledge of the PhN state space to build the SN of the phases one by one and the $\boldsymbol{\Delta}_{ij}$ using pair of phases. So the maximum amount of memory for the Component Method is the size of the whole system, and for DEEM is the maximum size of a pair of SN of two consecutive phases. The difference is linear in the number of phases.

In summary we can say that the more general method (the Component Method) when applied to the MPS with the limitations required by DEEM performs the same computations as DEEM which has instead an ad-hoc solver for MPS. The Component Method uses at most N times more memory than DEEM.

4. EXTENDED MPS MODELS

The Component Method is a general technique for non ergodic DSPN, so we can envision to lift all the limitations imposed by MPS and simply keep the requirement that the DSPN has to be non-ergodic. But this is an approach that may not be very useful for a designer that wants to use Petri nets to study a phased system, for which a more structured approach in the description of the system can be useful. We propose therefore to define an extended class of MPS, called X-MPS by simply lifting the limitation imposed on MPS.

Definition 1. An X-MPS is a non-ergodic DSPN composed of two nets, a Phase net, which is a DSPN, and a System Net, which is a GSPN [26] (only exponential and immediate transitions with associated priorities and inhibitor arcs).

X-MPS models can be solved efficiently with the Component Method, by devising the structure of the solution directly at the MRP level. Note that to ensure non-ergodicity in an easy manner we can re-insert limitation *L2* (the reachability graph of the PhN should be a DAG), but it is not required if the modeller is expert enough in Petri net modelling and, for example, stops a phase as a consequence of a SN event.

Lifting limitation *L1* and *L3* allows a richer description of phases in which phase duration may include some non-deterministic duration and more complex behaviour than a single marking, as in MPS. Removing limitation *L4* allows the SN to stop and restart a phase before its end, or to stop a phase and go to the next one (for example in a scheduled maintenance system a failure of a certain severity in the SN can interrupt the scheduled maintenance phase to start a more extensive maintenance activity).

The modelling power of X-MPS are shown in the next section through a number of examples of a SMS of increased complexity.

5. NUMERICAL RESULTS

In this section we consider an example of a *Scheduled Maintenance System* (SMS) model, inspired by [3], and represented using the PPN formalism. We shall first consider it in a standard MPS and then we show how it could be modified into an X-MPS to account for more complex features.

Fig. 1 shows the model of the SMS drawn with DEEM (upper part), which explicitly supports PPN, and with GreatSPN (lower part), in which the SMS is modeled as a DSPN. The SMS model represents an alternation between a factory work phase and a maintenance phase. The PhN and the SN are drawn into separate boxes, both in DEEM and in GreatSPN. The PhN models the alternation between work and maintenance, for *NP* consecutive cycles. During the production, raw pieces are loaded from a warehouse, are transformed following a sequence of steps with one of the M available machines, and are then moved back in a storage as finished products. Machines may break, and are therefore subject to a continuous maintenance that is scheduled at fixed intervals during the maintenance phase. In addition, workers have S spare parts to fix the machines between each maintenance cycle, parts that are replenished during maintenance.

In DEEM, each transition may have a guard condition that determines in which phase it is available. This feature, part of the PPN formalism, is not available in GreatSPN. Thus, the GreatSPN model has additional test/inhibitor arcs to model such guards.

Table 2 shows a comparison of the MPS method with the Component Method. The comparison is done for an increasing value of the parameter *NP*, which controls the number of phases. The table reports the PPN parameters, and the results of the MPS solver of DEEM and of the Component Method. For DEEM, the table reports the total number of phases, the maximum number of states per phase, and the total time. For the Component Method, the table reports the total number of states, the number and types of components obtained, and the total time. The last two columns report the expected number of completed products and failures of the machines after running for *NP* phases, computed with both tools, that, as expected, compute the same values.

Table 1: Classification of MRP components.

Class	*Component identification*	*Component characteristics*	*Cost of computing Eq.* (2)
C_E	No general enabled.	CTMC.	Steady state CTMC solution.
C_g	$\bar{\mathbf{Q}}_{i,i} = 0 \wedge \mathbf{\Delta}_{i,i} = 0$; g enabled and no internal $\bar{\mathbf{Q}}/\mathbf{\Delta}$ events.	CTMC under general event g.	Transient CTMC solution at time δ^g.
C_M	$\bar{\mathbf{Q}}_{i,i} \neq 0 \vee \mathbf{\Delta}_{i,i} \neq 0$, g enabled, internal $\bar{\mathbf{Q}}/\mathbf{\Delta}$ events.	MRP.	Steady state MRP solution.
	Mixed enabling of general events		

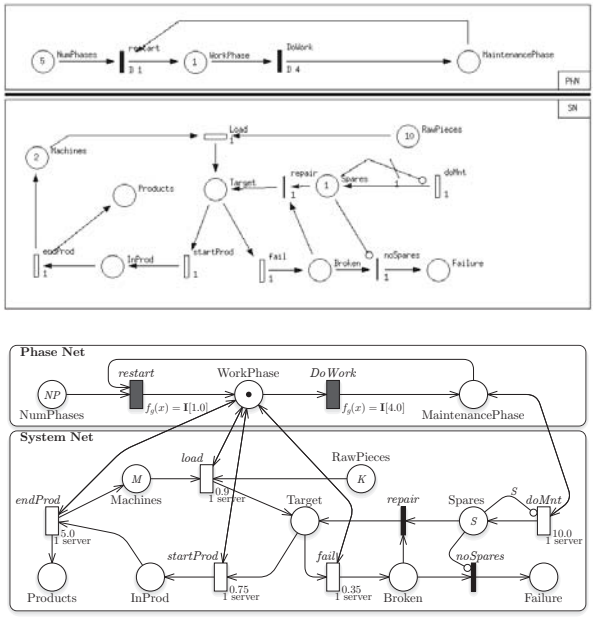

Figure 1: Scheduled Maintenance System drawn in DEEM (above, as a PPN) and in GreatSPN (below, as a DSPN).

The first observation is that DEEM execution times are much slower than expected. This is presumably due to a combination of two factors: DEEM is a prototype implementation, moreover it has been executed inside a virtual machine. For what concerns the memory requirements, from the upper table it is clear that GreatSPN builds the whole state space in a single shot (column States), while DEEM builds the state space of one phase at a time. Actually the maximum memory required by DEEM is not 389 but $2 * 389$ since two phases are needed to build the $\mathbf{\Delta}_{ij}$ matrices, but what is relevant is that this value does not increase with the number of phases.

A second observation is that, as shown in Section 3, the Component Method computes a number of components of type C_g which is equal to the number of phases, plus a final C_E component that includes all absorbing states. In this terminal component no deterministic or exponential is enabled (indeed matrix $\mathbf{Q}_i$, $\bar{\mathbf{Q}}_i$, and $\mathbf{\Delta}_i$ are empty).

The lower part of Table 2 reports in more details the structure of the phases and of the components for the two methods. Since the components correspond to the MPS phases, they are listed together. For the Component Method the component size is split into the states used in the solution (column tangible states) and the states of the frontier (state reached by a C_g component after the firing of g.

Table 2: MPS method compared to Component Method.

M=3, S=1			MPS method (DEEM)			Component method (GreatSPN)						
Model time	NP	K	Num phases	max states per phase	Time	States	C_E	C_g	C_M	Time	E[Failures]	E[Products]
14	2	10	5	389	39.9	2257	1	5	0	0.2	0.74	4.81
24	4	10	9	389	73.0	3813	1	9	0	0.2	1.11	7.38
34	6	10	13	389	118.6	5369	1	13	0	0.3	1.38	9.27
44	8	10	17	389	237.8	6925	1	17	0	0.4	1.55	10.55
54	10	10	21	389	372.8	8481	1	21	0	0.6	1.62	11.20
64	12	10	25	389	465.1	10037	1	25	0	0.8	1.65	11.41

Details for the phases/components of case NP=2, K=10:

	MPS method		Component method			
Phase num	tangible states	non-zero entries	tangible states	frontier states	total comp. states	type
1 Work	312	539	312	312	624	C_g
2 Maintenance	389	1166	389	389	778	C_g
3 Work	389	1230	389	389	778	C_g
4 Maintenance	389	1166	389	389	778	C_g
5 Work	389	1230	389	389	778	C_g
6 (system blocked)	-	-	389	0	389	C_E

X-MPS model of the SMS.

The original SMS model of interest has some additional rules that cannot be expressed as MPS, but corresponds to X-MPS. Since phases have deterministic duration, it may happen that the maintenance phase starts when the machines are processing some item. The maintenance phase needs to wait for their completion before starting. This makes the PhN dependent on a condition of the SN, which is not expressible in standard MPS. In addition maintenance could take a varying quantity of time, and thus is not represented anymore with a fixed-delay transition, but by the exponential transiton *restart*. Fig. 2 shows the Petri nets of the modified SMS as an X-MPS form, drawn with GreatSPN.

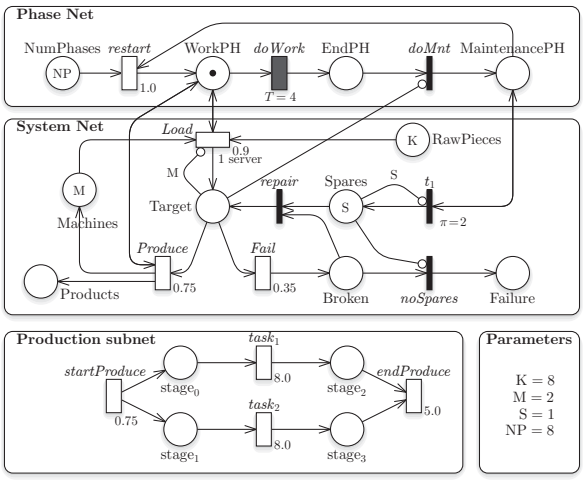

Figure 2: SMS extended with delayed maintenance rule.

Transition *Produce* is actually a subnet, shown in the lower part of Fig. 2), that models two parallel stages of the production. The delayed start of the maintenance phase is modeled using the inhibitor arc from place Target (and any place in the production subnet) to the transition *doMnt*.

The SMS in X-MPS form is tested in four different configurations. In configuration (A) a fixed number of phases is performed before the system stops. Configuration (B) is similar, runs are performed for an increasing number of both pieces and phases. Configuration (C) allows for an unbounded number of maintenance cycles: place *NumPhases* has been removed and the net is modified so that the work phase is interrupted when there are no more raw pieces. Tests are done on a varying number of parallel working machines and spare parts available to compute the probability distribution of machines into the *Failure* place at the time the system has consumed 10 raw pieces. Finally, configuration (D) has both unbounded phases and pieces, and computes the distribution of the states when all machines have failed. The net of configuration (D) can also be used to tests the duration of the *maintenance free operating period* (MFOP) of the system [24, 27]. Table 3 shows the results of the four configurations of the SMS for varying parameters.

Table 3: Component method on the extended SMS model.

(A) Finite number of phases set to NP=8:

				Standard		Component method			
K	States	SCC	BSCC	Iter.	Time	C_E	C_g	C_M	Time
20	16146	16146	99	30	23.8	9	9	0	2.9
40	32906	32906	199	31	52.6	9	9	0	6.1
60	49666	49666	299	31	79.8	9	9	0	9.1
80	66426	66426	399	31	108.4	9	9	0	12.3
100	83186	83186	499	31	134.1	9	9	0	15.5

(B) Increasing number of phases NP and pieces K, four machines:

					Standard		Component method			
K	NP	States	SCC	BSCC	Iter.	Time	C_E	C_g	C_M	Time
10	5	29877	29877	79	26	70.7	6	6	0	7.0
20	10	128407	128407	169	54	596.6	11	11	0	31.7
30	15	294237	294237	259	206	4943.6	16	16	0	74.6
40	20	527367	527367	349	-	-	21	21	0	136.4
50	25	827797	827797	439	-	-	26	26	0	215.5

(C) Unbound number of phases, waits for K=10 pieces completed:

					Standard		Component method			
M	S	States	SCC	BSCC	Iter.	Time	C_E	C_g	C_M	Time
10	2	830	467	13	25	1.0	9	9	18	2.6
10	3	3055	1796	17	30	6.4	10	9	18	18.1
10	4	8542	5219	23	42	31.4	10	9	18	76.2
20	2	1770	967	23	42	3.6	19	19	38	10.9
20	3	6865	3906	27	59	27.0	20	19	38	84.6
20	4	20432	12059	33	84	149.9	20	19	38	395.9

(D) Closed system, unbounded number of phases, ends with *Failure*:

				Standard		Component method			
K	States	SCC	BSCC	Iter.	Time	C_E	C_g	C_M	Time
20	5123	293	1	57	25.7	0	3	4	15.4
40	10683	613	1	85	81.7	0	3	4	65.8
60	16243	933	1	119	177.5	0	3	4	146.1
80	21803	1253	1	171	356.5	0	3	4	246.0
100	27363	1573	1	213	565.9	0	3	4	391.1

Note that, if we release the condition of delayed maintenance and of exponential duration of transition *restart*, configurations (A) and (B) simplify to an MPS, but configurations (C) and (D) are X-MPS system, since they have an unbounded number of phases. As we shall see, this type of net generates components of class C_M, which cannot be modeled using regular MPS.

Each table reports the test parameters, the number of MRP states, and the number of SCCs and BSCCs in the non-ergodic state space $\mathcal{S}$. Since we cannot compare with DEEM, we compare the matrix-free Component Method with a standard matrix-free solution of MRP [28] ("Standard" columns), that solves the whole system as a single component. Finally, the last four columns report the number of components obtained using the method described in this paper, divided per class type, and the time to compute the steady state distribution of absorbing states.

Case (A) is a non-ergodic MRP where the entire process can be decomposed into a constant number of C_E and C_g components (half of the phases have an exponential duration). Case (B) is similar, but the number of components grows with the parameters. In these two cases the Component Method behaves significantly better than the standard matrix-free one. Note that the aggregation of SCC into components has a significant impact: these systems have hundreds of thousands of SCCs: treating them one at a time would result in prohibitive matrix management costs. Both (A) and (B) are totally irreversible nets (no loop among states), as can be seen by the fact that the number of SCCs is the same as the number of MRP states.

Case (C) is a MRP where the aggregated components form an intermix of components of all three classes, including the C_M. The number of C_M components is rather high and this is a typical case in which the Component Method costs more than the standard matrix-free method, due to the cost of matrix management and to some inefficiency of the implementation. Case (D) has a structure made with few components, where C_M ones are still present in a small number. The method performance are similar to that of case (C), except that the smaller number of C_M components makes the overall solution process more efficient than the standard matrix-free .

6. CONCLUSIONS AND FUTURE WORK

This paper extends the class of MPS systems that can be efficiently solved in an analytical manner thanks to the Component Method for non-ergodic MRPs. It also shows that, when applied to standard MPS behaves similarly to the ad-hoc MPS solver implemented in DEEM, despite using more memory. The Component Method implemented in GreatSPN also allows for transitions with (a limited number of different) general distributions, as far as there is at most one enabled in each state: a possible future extension is to include generally distributed transitions into the PhN definition.

We have seen that DEEM saves memory by building the state space of SN one phase at a time: recent work on on-the fly generation of MRP in the context of the MC4CSLTA model-checker, which is part of GreatSPN, can be a line to follow for the on-the-fly generation of the MPS state space.

Although we believe that the extensions introduced in this paper are of practical significance, as the possibility of an unbounded number of phases, the introduction of exponential delays in the PhN or the possibility for the SN to disable a phase, we should remark that certain features of DEEM not available in GreatSPN also have a practical relevance, in particular the separation in the definition of the PhN and SN net, the guards (to avoid too many crossing arcs), the reward structure and the computation of the probability at time t of the whole MPS. This last feature is strictly related to the choice of including only deterministic timings in the

PhN. The ideal MPS tool should include features of both tools, and in particular it would be very useful to have a tool in which the user can choose his/her own trade-off among modelling power and computable performance indices.

7. REFERENCES

[1] A. Bondavalli, S. Chiaradonna, F. Di Giandomenico, and I. Mura, "Dependability modeling and evaluation of multiple-phased systems using DEEM," *IEEE Transactions on Reliability*, vol. 53, no. 4, pp. 509–522, 2004.

[2] I. Mura and A. Bondavalli, "Markov Regenerative Stochastic Petri Nets to Model and Evaluate the Dependability of Phased Missions," *IEEE Transactions on Computers*, vol. 50, no. 12, pp. 1337–1351, 2001.

[3] A. Bondavalli, I. Mura, and K. S. Trivedi, *Dependability Modelling and Sensitivity Analysis of Scheduled Maintenance Systems*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 7–23.

[4] Z. Peng, Y. Lu, and A. Miller, "Uncertainty analysis of phased mission systems with probabilistic timed automata," in *7th IEEE International Conference on Prognostics and Health Management (PHM'16)*, 2016.

[5] H. Choi, V. Mainkar, and K. S. Trivedi, "Sensitivity analysis of Deterministic and Stochastic Petri Nets," in *International Workshop on Modeling, Analysis, and Simulation On Computer and Telecommunication Systems (MASCOTS)*, San Diego, USA, 1993, pp. 271–276.

[6] A. K. Somani, J. A. Ritcey, and S. H. Au, "Computationally-efficient phased-mission reliability analysis for systems with variable configurations," *IEEE Transactions on Reliability*, vol. 41, no. 4, pp. 504–511, 1992.

[7] L. Xing and S. Amari, "Reliability of Phased-mission Systems," in *Handbook of Performability Engineering*, K. Misra, Ed. Springer London, 2008, pp. 349–368.

[8] I. Mura, A. Bondavalli, X. Zang, and K. S. Trivedi, "Dependability modeling and evaluation of Phased Mission Systems: a DSPN approach," in *Int. Conference on Dependable Computing for Critical Applications*. IEEE Press, 1999, pp. 299–318.

[9] M. Ajmone Marsan and G. Chiola, "On Petri nets with deterministic and exponentially distributed firing times," in *Advances in Petri Nets*, ser. Lecture Notes in Computer Science, vol. 266/1987. Springer Berlin / Heidelberg, 1987, pp. 132–145.

[10] R. German, *Performance Analysis of Communication Systems with Non-Markovian Stochastic Petri Nets*. New York, NY, USA: John Wiley & Sons, Inc., 2000.

[11] R. Pyke, *Markov Renewal Processes with Finitely Many States*. New York: Columbia University, 1959. [Online]. Available: http://books.google.it/books?id=ZFn7ygAACAAJ

[12] W. J. Stewart, *Probability, Markov chains, queues, and simulation : the mathematical basis of performance modeling*. Princeton (N.J.), Oxford: Princeton University Press, 2009.

[13] E. G. Amparore and S. Donatelli, "A component-based solution for reducible markov regenerative processes," *Performance Evaluation*, vol. 70, no. 6, pp. 400 – 422, 2013.

[14] E. G. Amparore, G. Balbo, M. Beccuti, S. Donatelli, and G. Franceschinis, *30 Years of GreatSPN*. Cham: Springer International Publishing, 2016, pp. 227–254.

[15] E. G. Amparore and S. Donatelli, "MC4CSL$^{\text{TA}}$: an efficient model checking tool for CSL$^{\text{TA}}$," in *International Conference on Quantitative Evaluation of Systems*. Los Alamitos, CA, USA: IEEE Computer Society, 2010, pp. 153–154.

[16] S. Donatelli, S. Haddad, and J. Sproston, "Model checking timed and stochastic properties with CSL$^{\text{TA}}$," *IEEE Transactions on Software Engineering*, vol. 35, no. 2, pp. 224–240, 2009.

[17] K. S. Trivedi, *Probability and Statistics with Reliability, Queuing, and Computer Science Applications*. 605 Third Avenue, New York, USA: John Wiley and Sons, Ltd., 2002.

[18] C. G. Cassandras and S. Lafortune, *Introduction to Discrete Event Systems*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006.

[19] E. G. Amparore and S. Donatelli, "A component-based solution method for non-ergodic Markov Regenerative Processes," in *Computer Performance Engineering*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2010, vol. 6342, pp. 236–251.

[20] R. Tarjan, "Depth-first search and linear graph algorithms," in *Proceedings of the 12th Annual Symposium on Switching and Automata Theory (SWAT 1971)*. Washington, DC, USA: IEEE Computer Society, 1971, pp. 114–121.

[21] E. G. Amparore and S. Donatelli, "Improving and assessing the efficiency of the MC4CSL$^{\text{TA}}$ model checker," in *Computer Performance Engineering*, ser. LNCS. Springer Berlin Heidelberg, 2013, vol. 8168, pp. 206–220.

[22] ——, "Optimal aggregation of components for the solution of Markov Regenerative Processes," to appear in QEST2016, August 2016.

[23] U. of Torino, "The greatspn tool homepage," http://www.di.unito.it/~greatspn/index.html.

[24] S. Chew, S. Dunnett, and J. Andrews, "Phased mission modelling of systems with maintenance-free operating periods using simulated petri nets," *Reliability Engineering & System Safety*, vol. 93, no. 7, pp. 980 – 994, 2008, bayesian Networks in Dependability.

[25] H. Choi, V. G. Kulkarni, and K. S. Trivedi, "Markov regenerative stochastic Petri nets," *Performance Evaluation*, vol. 20, no. 1-3, pp. 337–357, 1994.

[26] M. Ajmone-Marsan, G. Balbo, G. Conte, S. Donatelli, and G. Franceschinis, *Modelling with Generalized Stochastic Petri Nets*. Wiley Series in Parallel Computing, John Wiley and Sons, 1995.

[27] C. Hockley, "Design for success," *Journal of Aerospace Engineering*, vol. 212, no. 6, pp. 371–378, 1998.

[28] R. German, "Iterative analysis of Markov regenerative models," *Performance Evaluation*, vol. 44, pp. 51–72, April 2001.