Chapter 11

Tensor methods and stochastic Petri nets

Serge Haddad , Patrice Moreaux[1]

1. Introduction

To get the stationary distribution of Markovian models we have to solve a system of equations whose the number of equations and unknowns equal the number of states of the Markov chain. Various methods allow us to get this distribution by lowering the computation complexity. For a specific structure of Petri net (studied in chapter 9), the stationary distribution admits a product-form representation that we can compute form the invariants of the net. With stochastic Petri nets, (chapter 10), we substitute an aggregated chain to the Markov chain. The stationary distribution of this new chain leads straightforwardly to the non aggregated distribution. We present now a method based on systems decomposition.

As we show in section 2, the applicability conditions of this method are expressed at the Markov chainlevel: the state space is a Cartesian product of subspaces and transitions between states are either locales (modifying only one component of the state), or else synchronized (in the converse case). Then, the infinitesimal generator may be written as an expression operands of which are matrices indexed by states of the subspaces and operators are the tensor sum and the tensor product. Tensor[2] algebra properties ensure that the generator

[1]LSV, École normale supérieure de Cachan, Cachan, France, (haddad@lsv.ens-cachan.fr)
LISTIC, Polytech'Savoie, Université de Savoie BP 80439 74944 Annecy le Vieux Cedex, France, (patrice.moreaux@univ-savoie.fr),

[2]The reader will find the two set of names Kronecker algebra, product, etc. and tensor algebra, product, etc. in the literature. We use the term tensor in this chapter.

matrix product may be obtained *from submatrices only*. Hence, product based iterative resolution methods may be used without computing the generator. Time and memory complexities are then linear with respect to the sizes of the subspaces.

This method was first proposed by [PLA 85, PLA 91] for stochastic automata networks (SAN), an model equivalent to composition of Markov chains. Then, it was extended to synchronous composition of generalised stochastic Petri nets (GSPN) by [DON 94] (i.e. with transitions fusion). The main advantage of the GSPN model is the factorization of synchronized transitions of the Markov chain on the net transitions reducing significantly the size of the tensorial expression. Whatever the application model, the Cartesian product of the subspaces is always super-space of the state space which minimizes complexity savings (in a substantial way in worst cases). To apply the tensorial method to asynchronous composition of nets (through place fusion), each subnet must have an abstract view of the other subnets [BUC 93] to build finite subspaces. Following this idea, we dive the state space into an union of Cartesian products of subspaces which leads to a decomposition of the generator in blocks of submatrices where each submatrix is defined by a tensor expression [CAM 97]. This decomposition benefits from tensor algebra at the submatrix level and allows us to cope with the ratio the state space and its super-space. We have the same decomposition of the state space of a stochastic Petri net with phase type distribution (PH-SPN) between markings and descriptors of the residual distributions [DON 98]. In this case, we show that the state space is *exactly* the union of the Cartesian products. This result leads to regard PH-SPN as one of the most efficient solution to model general distributions. After presenting the tensor analysis of stochastic Petri nets in section 3, we study in the last section the combination of aggregation and tensor decomposition with the help of the stochastic well formed Petri net model. The two approaches are not orthogonal and two difficulties must be overcome before using them simultaneously. On the on ehand, modeling the system may force the designer to choose between a stochastic well formed Petri net (SWN) and a SPN composition. On the other hand, if the modeling comes out on SWN composition, most often *the tensor composition* of the aggregated chains of the SWN is not an aggregation of the chain of the composed SWN. Therefore, the authors of this chapter, have exhibited sufficient syntactic conditions for asynchronous and synchronous composition of SWNs.

For simplicity, we restrict ourself to compute steady-state distribution of the models. However, the applicability of these methods extends most often to the transient analysis. At last, since some results are rather technical, we refer the reader to the references for a detailed mathematical exposition.

2. Synchronized Markov chains

The principle of decomposition methods consists of exploiting the distinction between local actions and actions with global impact, translating this distinction at the level of the matrices of the Markov chains of the stochastic processes modeling entities carrying out these actions.

In this part, we study a Markov chain X with values in the space $S = \prod_{k=1}^{K} S_k$, each[3] S_k being of cardinality n_k. Hence $X = (X_1, X_2, \ldots, X_K)$. we order S with respect to the *lexicographical* order of the components: a state $s = (s_1, \ldots, s_K)$ of S, component s_k of which has index i_k, has for global index the integer

$$i = \sum_{k=1}^{K-1} (i_k - 1) \left(\prod_{j=k+1}^{K} n_j \right) + i_K$$

In the following, we identify states and integers when no confusion may arise and i is identified to its writing $(i_1, \ldots, i_K)$ in the "multi-base" $(n_1, \ldots, n_K)$ [DAV 81]. Also, the projection of i on S_k is denoted by i_k (by extension).

Then, for each transition τ of X, we can identify the involved components in the modification of the state, named the domain of τ, and the others components. This allows us to distinguish transitions modifying only one component k without taking other components into account (subspace S_k local event), from the transitions modifying several components and/or expecting a given state in several components (synchronized event in several S_k).

Definition 1 (Synchronized continuous time Markov chain) *A synchronized continuous time Markov chain (CTMC) is a CTMC on a space* $S = \prod_{k=1}^{K} S_k$. *A transition* τ *is fully defined by its rate* $\lambda(\tau)$, *its domain* $dom(\tau) = \prod_{k \in K(\tau) = \{i_1, \ldots, i_m\}} S_k$, *its input constraints* $s_{i_1}, \ldots, s_{i_m}$ *and its output constraints* $s'_{i_1}, \ldots, s'_{i_m}$. *For all states* s *and* s', $s \xrightarrow{\tau} s'$ *iff*

- *the projections of* s *and* s' *on* $dom(\tau)$ *are equal to its input and output constraints;*
- *the projections of* s *and* s' *on the complementary domain of* τ $(\prod_{k \in K \setminus K(\tau)} S_k)$ *are equal.*

A transition τ *is* local *(to* X_k*) iff its domain is reduced to only one subspace* (S_k). *A transition* τ *is* a synchronisation transition *iff it is not local.*

[3] For ease of writing, K denotes, depending on the context, the integer K, as here, or the set of the K integers $\{1, \ldots, K\}$ if no confusion may arise.

Hence, a local transition may modify only one component of a state. Let us note that the $\mathbf{Q}[s, s']$ element of the generator of X is then, for $s \neq s'$: $\mathbf{Q}[s, s'] = \sum_{\tau, s \xrightarrow{\tau} s'} \lambda(\tau)$.

In the following, all matrices are real valued. We denote by $\mathcal{M}_{n,p}$ the set of $n \times p$ matrices and we set $\mathcal{M}_n = \mathcal{M}_{n,n}$.

2.1. *Tensor product*

The tensor product allows us to get a tensor expression of the transition probabilities matrices of discrete time processes with K components. But, it also translates at the generators level of the CTMC of the continuous time processes, the impact of of synchronization transitions which produce simultaneously state modifications in several subsystems. This is mainly this property that we will use since we will study continuous time models.

Definition 2 (Tensor product) *Let be* $\mathbf{A} \in \mathcal{M}_{n_1,p_1}$ *and* $\mathbf{B} \in \mathcal{M}_{n_2,p_2}$. *The tensor product* ($\bigotimes$) *of* $\mathbf{A}$ *and* $\mathbf{B}$ *is the matrix* $\mathbf{C} \in \mathcal{M}_{n_1 n_2, p_1 p_2}$*:*

$$\mathbf{C} = \mathbf{A} \bigotimes \mathbf{B} \quad \textit{with} \quad c_{ij} = a_{i_1 j_1} b_{i_2 j_2}$$

where $i = (i_1, i_2)$ *in the multi-base* (n_1, n_2) *and* $j = (j_1, j_2)$ *in* (p_1, p_2).

Example 1 If

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{pmatrix} \quad \text{and} \quad \mathbf{B} = \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \\ b_{31} & b_{32} \\ b_{41} & b_{42} \end{pmatrix}$$

$$\mathbf{A} \bigotimes \mathbf{B} = \left(\begin{array}{cc|cc|cc} a_{11}b_{11} & a_{11}b_{12} & a_{12}b_{11} & a_{12}b_{12} & a_{13}b_{11} & a_{13}b_{12} \\ a_{11}b_{21} & a_{11}b_{22} & a_{12}b_{21} & a_{12}b_{22} & a_{13}b_{21} & a_{13}b_{22} \\ a_{11}b_{31} & a_{11}b_{32} & a_{12}b_{31} & a_{12}b_{32} & a_{13}b_{31} & a_{13}b_{32} \\ a_{11}b_{41} & a_{11}b_{42} & a_{12}b_{41} & a_{12}b_{42} & a_{13}b_{41} & a_{13}b_{42} \\ \hline a_{21}b_{11} & a_{21}b_{12} & a_{22}b_{11} & a_{22}b_{12} & a_{23}b_{11} & a_{23}b_{12} \\ a_{21}b_{21} & a_{21}b_{22} & a_{22}b_{21} & a_{22}b_{22} & a_{23}b_{21} & a_{23}b_{22} \\ a_{21}b_{31} & a_{21}b_{32} & a_{22}b_{31} & a_{22}b_{32} & a_{23}b_{31} & a_{23}b_{32} \\ a_{21}b_{41} & a_{21}b_{42} & a_{22}b_{41} & a_{22}b_{42} & a_{23}b_{41} & a_{23}b_{42} \end{array}\right)$$

The definition of the tensor product is straightforwardly extended to K matrices:

$$\bigotimes_{k=1}^{K} \mathbf{M}_k = \mathbf{M}_1 \bigotimes \cdots \bigotimes \mathbf{M}_K = \mathbf{M} \quad \text{with} \quad m_{ij} = \prod_{k=1}^{K} m_{i_k j_k}$$

where $i = (i_1, \ldots, i_K)$ in $(n_1, \ldots, n_K)$ and $j = (j_1, \ldots, j_K)$ in $(p_1, \ldots, p_K)$.

The fundamental interest in tensor expression of the generator $\mathbf{Q}$ of a Markov chain is the saving in required memory to store $\mathbf{Q}$. If matrices $\mathbf{M}_k$ and $\mathbf{M}$ are dense, saving is obvious ($\sum n_k^2$ versus $\prod n_k^2$) and it remains important in the case, frequently encountered in the SPN context, of sparse matrices, or even very sparse. However, this saving involves more complex solution algorithms for $\boldsymbol{\pi}.\mathbf{Q} = \mathbf{0}$ than for direct representations of $\mathbf{Q}$.

Computation of $\boldsymbol{\pi}.\mathbf{A} \bigotimes \mathbf{B}$

Due to the size of $\mathbf{Q}$, iterative resolution methods of $\boldsymbol{\pi}.\mathbf{Q} = \mathbf{0}$ (with a sparse representation of $\mathbf{Q}$) are the only usable ones. Among these methods, we mainly use (ordered by increasing convergence speed in most cases) the power method, the Jacobi and the Gauss-Seidel algorithms [STE 94], which do not modify $\mathbf{Q}$. For instance, with the power method, the n_{th} step computes $\boldsymbol{\pi}^{(n+1)} = \boldsymbol{\pi}^{(n)}(\mathbf{I} + \frac{1}{\beta}\mathbf{Q})$ where $\beta > \max_i |q_{i,i}|$. In all these techniques, the key point is the efficient computation of vector-matrix products, in the shape of $\mathbf{x}.\bigotimes_{k=1}^{K} \mathbf{A}^{(k)}$. Two kinds of methods are used.

The first one, introduced in [PLA 85] is based on two technical elements. On the one hand, permutations denoted by σ_k (with associated matrices $\mathbf{M}_{\sigma_k}$) called "perfect shuffles" [DAV 81] reorder vector components. On the other hand, the relation $\mathbf{x}.\bigotimes_{k=1}^{K} \mathbf{A}^{(k)} = \mathbf{x}.\prod_{k=1}^{K} \mathbf{M}_{\sigma_k}^T.(\mathbf{I}_{\overline{n}_k} \bigotimes \mathbf{A}^{(k)}).\mathbf{M}_{\sigma_k}$ transforms a K terms tensor product into K ordinary products ($\overline{n}_k = \frac{n}{n_k}$ and $\mathbf{M}_{\sigma_k}^T$ is the transpose of $\mathbf{M}_{\sigma_k}$).

The second method translates into the code, the relation $a_{i,j} = a^{(1)}_{i_1,j_1} a^{(2)}_{i_2,j_2} \ldots a^{(K)}_{i_K,j_K}$ using expression of the indexes i and j in the multi-base $(n_1, \ldots, n_K)$: $i = (\cdots((i_1 - 1)n_2 + (i_2 - 1))n_3 \cdots)n_K + i_K$. Algorithm 1 corresponds to this method for the computation of the product $\mathbf{x}.\mathbf{A} \bigotimes \mathbf{B}$. The body of the external loop (on i) computes the contribution of x_i to $\mathbf{y}$. Each of the internal loops completes the computation of the product $a_{i_1,j_1} b_{i_2,j_2}$. The most internal loop finishes the computation multiplying with x_i. At the same time, each of the two loop levels contributes to the computation of the index l_2 of the component of the vector $\mathbf{y}$ to be modified.

Algorithm 1 (Computation of $\mathbf{y} = \mathbf{x}.\mathbf{A} \bigotimes \mathbf{B}$)

```
begin
    Initialize y to 0
    For i from 1 to n = n1.n2 do      /* i = (i1, i2), j = (j1, j2) */
        For each j1 such that a_{i1,j1} ≠ 0 do
            l1 ← (j1 − 1).n2          /* also avoid the product in the internal loop */
            c1 ← a_{i1,j1}            /* speed up accesses in the internal loop */
            For each j2 such that b_{i2,j2} ≠ 0 do
                l2 ← l1 + j2 − 1
                y_{l2} ← y_{l2} + x_i.c1.b_{i2,j2}
            done
        done
    done
end
```

Complexity analysis of these methods [BUC 97, FER 98] show that, as a general rule, the additional cost of the tensor methods increases with the sparsity of the matrices (with models derived from SPN, matrices are always sparse, and sparsely stored). More precisely, let us denote by $\eta(\mathbf{M})$ the number of non null elements of a matrix $\mathbf{M}$ and $\alpha_k = \frac{\eta(A^{(k)})}{n_k}$ the filling ratio of $\mathbf{A}^{(k)}$. For simplicity, we assume that all α_k have the same value α. The complexity (number of floating point operations) of the computation of the product $\mathbf{y} = \mathbf{x}.\mathbf{A}$ (with $\mathbf{A} = \bigotimes_{k=1}^{K} \mathbf{A}^{(k)}$) is in the order of $O(\eta(\mathbf{A})) = n.\alpha^K$ when $\mathbf{A}$ is stored explicitly. If $\mathbf{A}$ is stored implicitly through matrices $\mathbf{A}^{(k)}$, the complexity of the direct computation of $\mathbf{y}$ is in the order of $O(K.\eta(\mathbf{A}))$. We can check that the complexity of the perfect shuffle method is in the order of $O(n.K.\alpha)$. Type two methods have a complexity in $O(K.\eta(\mathbf{A}))$ for very sparse matrices, and in $O(\eta(\mathbf{A}))$ for sparse matrices. Thus, for sparse matrices, or even very sparse matrices, the additional cost of tensor decompositions for the resolution remains low.

2.2. *Tensor sum and continuous time Markov chains*

The tensor sum allows us to express the generator of the CTMC of K components processes, with those of its components. It translates the autonomous behavior of the components resulting from local transitions.

Definition 3 (Tensor sum) *Let* $\mathbf{A} \in \mathcal{M}_n$ *and* $\mathbf{B} \in \mathcal{M}_p$ *be two* square *ma-*

trices. The tensor sum ($\bigoplus$) of $\mathbf{A}$ *and* $\mathbf{B}$ *is the matrix* $\mathbf{C} \in \mathcal{M}_{n.p}$*:*

$$\mathbf{C} = \mathbf{A} \bigoplus \mathbf{B} = \mathbf{A} \bigotimes \mathbf{I}_p + \mathbf{I}_n \bigotimes \mathbf{B}$$

Hence:

$$c_{ij} = \begin{cases} a_{i_1 j_1} + b_{i_2 j_2} & \text{if} \quad i_1 = j_1 \text{ and } i_2 = j_2 \\ b_{i_2 j_2} & \text{if} \quad i_1 = j_1 \text{ and } i_2 \neq j_2 \\ a_{i_1 j_1} & \text{if} \quad i_1 \neq j_1 \text{ and } i_2 = j_2 \\ 0 & \text{if} \quad i_1 \neq j_1 \text{ and } i_2 \neq j_2 \end{cases}$$

Example 2 If

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \quad \text{and} \quad \mathbf{B} = \begin{pmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{pmatrix}$$

$$\mathbf{A} \bigoplus \mathbf{B} = \left(\begin{array}{ccc|ccc} a_{11}+b_{11} & b_{12} & b_{13} & a_{12} & & \\ b_{21} & a_{11}+b_{22} & b_{23} & & a_{12} & \\ b_{31} & b_{32} & a_{11}+b_{33} & & & a_{12} \\ \hline a_{21} & & & a_{22}+b_{11} & b_{12} & b_{13} \\ & a_{21} & & b_{21} & a_{22}+b_{22} & b_{23} \\ & & a_{21} & b_{31} & b_{32} & a_{22}+b_{33} \end{array}\right)$$

We also extend the tensor sum to K matrices $\mathbf{M}_k$:

$$\bigoplus_{k=1}^{K} \mathbf{M}_k = \sum_{k=1}^{K} \bigotimes_{1 \leq k' < k} \mathbf{I}_{n_{k'}} \bigotimes \mathbf{M}_k \bigotimes_{k < k' \leq K} \mathbf{I}_{n_{k'}} = \sum_{k=1}^{K} \mathbf{I}_{l_k} \bigotimes \mathbf{M}_k \bigotimes \mathbf{I}_{u_k}$$

with: $l_k = \prod_{k'<k} n_{k'}$ and $u_k = \prod_{k'>k} n_{k'}$.

Local transitions of a synchronized CTMC generate, in the expression of the generator $\mathbf{Q}$, a tensor sum rendering the independence of these transitions, and each *synchronization* transition generates a tensor product rendering the simultaneous modification of several components. The fundamental theorem below is the base for structured descriptions of generators used for SANs and composition of stochastic Petri nets.

Theorem 4 (Generator of a synchronized CTMC) *Let* $X = (X_1, \ldots, X_K)$ *be a synchronized CTMC and* $\mathcal{T}_s$ *be the set of its synchronization transitions. The generator of* X *is*

$$\mathbf{Q} = \bigoplus_{k=1}^{K} \mathbf{Q}'_k + \sum_{\tau \in \mathcal{T}_s} \lambda(\tau) \left[\bigotimes_{k=1}^{K} \mathbf{C}_k(\tau) - \bigotimes_{k=1}^{K} \mathbf{A}_k(\tau) \right] \qquad [1]$$

where $\mathbf{Q}'_k$ *is the restriction of* $\mathbf{Q}$ *to local transitions of* S_k *and, for* $\tau \in \mathcal{T}_s$*, with rate* $\lambda(\tau)$*, such that* $i \xrightarrow{\tau} j$*:*

$$\begin{array}{ll} \mathbf{C}_k(\tau) = \mathbf{A}_k(\tau) = \mathbf{I}_{n_k} & \text{if } S_k \text{ is not in the domain of } \tau \\ \mathbf{C}_k(\tau) = \mathbf{1}_{n_k}(i_k, j_k) \quad \text{and} \quad \mathbf{A}_k(\tau) = \mathbf{1}_{n_k}(i_k, i_k) & \text{otherwise and if } \tau_k(i_k) = j_k \end{array}$$

where $\mathbf{1}_n(j, j')$ *is the* $\mathcal{M}_n$ *matrix all terms of which are null except the one with index* (j, j') *(equal to 1), and* τ_k *is the projection of* τ *on* S_k *(we identify a state* s *with its index* i *and we identify its components* s_k *with their index* i_k *in* S_k*).*

Matrices $\mathbf{I}_{n_k}$ "spread", in $\mathbf{Q}$, jumps in each S_k where τ produces a state modification. Matrices $\mathbf{A}_k(\tau)$ ensure the diagonal compensation of the $\mathbf{C}_k(\tau)$ such that $\mathbf{Q}$ is a generator.

Proof
Translating the state modifications due to transitions of X, we get

$$\mathbf{Q} = \sum_{\tau} \lambda(\tau) \left[\bigotimes_{k=1}^{K} \mathbf{C}_k(\tau) - \bigotimes_{k=1}^{K} \mathbf{A}_k(\tau) \right].$$

Decomposing $\mathbf{Q}$ according to local and synchronization transitions, we get $\mathbf{Q} = \mathbf{Q}^l + \mathbf{Q}^s$. $\mathbf{Q}^l$ refers only local transitions. Then, we group for each k these transitions (termed k-local transitions):

$$\mathbf{Q}^l = \sum_{k} \sum_{\tau,\ k-\text{local}} \bigotimes_{k'<k} \mathbf{I}_{k'} . \lambda(\tau) . \left[\mathbf{C}_k(\tau) - \mathbf{A}_k(\tau) \right] . \bigotimes_{k'>k} \mathbf{I}_{k'}$$

We note that we can factorize the two tensor products and we then recognize the tensor sum of the theorem. ♢

The most important interest of the fundamental theorem lies in the computation of the steady-state solution (and also the transient solution) of the continuous time Markov chain. Starting from [1], we can use a lot of numerical solution methods of the equation $\boldsymbol{\pi}.\mathbf{Q} = \mathbf{0}$ exploiting this expression, without explicitly computing $\mathbf{Q}$. In contrast, we only use $\mathbf{Q}'_k, \mathbf{C}_k$ and $\mathbf{A}_k$ matrices, generally far smaller than $\mathbf{Q}$, without significantly increasing computation times of the steady-state solution (see section 2.1).

Let us note that if the CTMC has no synchronization transition, we have an independent CTMC composition and the expression [1] reduces to a tensor sum.

Principle of two levels methods

Theorem 4 applies when the state space is a subset of a Cartesian product of local spaces. This result is extended in the following way. At a first level, a super-set of the state space is partitioned with respect to an equivalence relation $\mathcal{R}$. Each equivalence class is then a Cartesian product of local subspaces. Reordering states with respect to equivalence classes of $\mathcal{R}$, $\mathbf{Q}$ may be seen as a block $(\mathbf{Q}[\widehat{m}, \widehat{m}'])$ matrix, where $\widehat{m}$ is the class of m. Each $\mathbf{Q}[\widehat{m}, \widehat{m}']$ matrix is then written in the form of a tensor expression analogous to [1].

During the nth iteration of the resolution with an iterative method, the sub-vector $\boldsymbol{\pi}^{(n+1)}_{\widehat{m}}$ of the product $\boldsymbol{\pi}^{(n)}.\mathbf{Q}$ is computed with the expression:

$$\boldsymbol{\pi}^{(n+1)}_{\widehat{m}} = \sum_{\widehat{m}'} \boldsymbol{\pi}^{(n)}_{\widehat{m}'}.\mathbf{Q}[\widehat{m}', \widehat{m}]$$

Then, the tensor method applies to each term of this sum.

Including the state space in an union of Cartesian products noticeably reduces the number of *fictitious* states added to the effective state space. However, for a general model, it is difficult to exhibit an equivalence relation $\mathcal{R}$ defined at the syntactic level. The reader may refer to section 3.3 for an application example of this approach.

3. Tensor algebra and SPN

We can remark that in the expression [1], we have usually a very large number of matrices $\mathbf{C}_k$ each having only one non null term. Hence, we look for conditions on *higher level* models (stochastic automata networks, stochastic Petri nets, ...) so that their underlying generators have a tensor expression with state transition factorization corresponding to the same high level event.

A tensor decomposition of states renders a system structure made up of several interacting subsystems, each one having a more or less large behavioral autonomy. The model is then built taking this structure into account with conditions to be enforced to get a factorized tensor expression.

We detail below application of tensor methods to decomposable GSPN and to PH-SPN.

The Petri net model is essentially a "flat" model and the reachability graph is a priori and unstructured graph. Numerous approaches have devised structuring methods for Petri nets to structure the reachability graph. Among them,

composition of subnets allows us to take into account the two fundamental coordination mechanisms between systems. We speak about *decomposition* into subnets if we try conversly to composition, to define subnets of a global net. From the Markovian models point of view, the two approaches are identical. This is not the case at the structural level of Petri nets: composition is usually more complex to define formally. Since we focus on tensor methods, we adopt the decomposition point of view.

3.1. *Synchronous decomposition of generalised stochastic Petri nets*

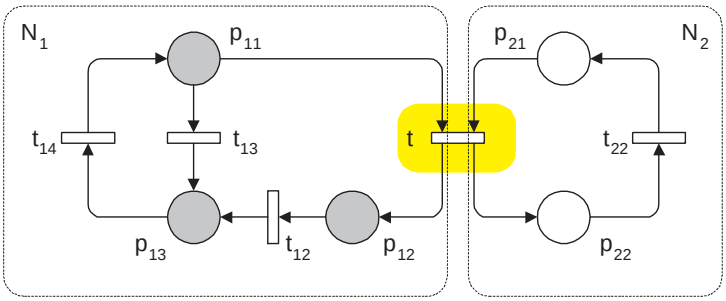

Figure 1: Synchronous decomposition of GSPN

Rendez-vous synchronization, or synchronous coordination, as the ADA language rendez-vous, corresponds to an event arising simultaneously in the sub-systems. In Petri nets, it is modeled by fusion of transitions of each subnet. Introduced by [DON 94], the synchronous decomposition corresponds to a given partition of the places of the net into subsets $P_1, \ldots, P_K$. Each subset induces a classification of the transitions: a transition is k-local iff its entry and output places are in P_k; a transition is termed a synchronization transition otherwise. For instance, in figure 1, the net is deocmposed through place subsets $P_1 = \{p_{11}, p_{12}, p_{13}\}$ and $P_2 = \{p_{21}, p_{22}\}$, and the only synchronization transition is t. So we define K subnets, each consisting of places P_k and of all transitions and arcs bound to P_k. These subnets have common transitions (the synchronization ones, t in the example) and they are called superposed generalised stochastic Petri nets (SGSPN). We denote by TS the set of synchronization transitions, TS_k the transitions of TS which are in the kth subnet and $m_k[t_{(k)}\rangle m'_k$ the fact that t, considered as a transition of $\mathcal{N}_k$, is enabled in m_k and produces the marking m'_k.

The interest of this decomposition is twofold. On the one hand, it allows us to express the reachability set of the net as a subset of the Cartesian product of the reachability sets of the K subnets, termed the potential reachability set (PRS) of the net. Obviously, this presupposes that each subnet stays bounded when it is studied in isolation. On the other hand, we can show, inspecting possible firings, that the generator of the underlying continuous time Markov chain of the net is a "submatrix" of a tensor expression involving only matrices which may be computed in each subnet in isolation. Hence, we have the following result.

Theorem 5 (Generator of the synchronous composition of GSPN) [DON 94] *The generator of the CTMC of the net $\mathcal{S}$, synchronous composition of the $\mathcal{S}_k$, is a submatrix of*

$$\mathbf{Q}' = \bigoplus_{k=1}^{K} \mathbf{Q}'_k + \sum_{t \in TS} \mathbf{w}[t] \left[\bigotimes_{k=1}^{K} C_k(t) - \bigotimes_{k=1}^{K} A_k(t) \right] \qquad [2]$$

with, if $t \notin TS_k$:

$$\mathbf{C}_k(t) = \mathbf{A}_k(t) = \mathbf{I}_{n_k}$$

and if $t \in TS_k$:

$$c_k(t)_{m_k, m'_k} = 1 \text{ if } m_k[t_{(k)}\rangle m'_k \text{ and } 0 \text{ otherwise}$$

$$a_k(t)_{m_k, m'_k} = \begin{cases} \sum_{m''_k \neq m_k} c_k(t)_{m_k, m''_k} & \text{if} \quad m'_k = m_k \\ 0 & \text{if} \quad m'_k \neq m_k \end{cases}$$

and where matrices $\mathbf{Q}'_k$ are the elements of the generators of the CTMC of tangible markings of the $\mathcal{S}_k$ in isolation, that is to say only taking into account local transitions of $\mathcal{N}_k$.

We see that firings of synchronization transitions are *factorized* to get a tensor expression, elements of which are autonomously computable in *each subnet*.

Submatrix means that non null terms of $\mathbf{Q}$ for a given pair of states are equal to those of $\mathbf{Q}'$ and that if m is reachable in $\mathcal{S}$, then $q'_{m,m'} = 0$ if m' is unreachable in $\mathcal{S}$. This theorem may be extended to GSPN with immediate transitions which are not synchronization transitions: coefficients are modified to take into account immediate firing sequences which may occur after a synchronized firing.

The schema of the elementary computation algorithm of a performance measure is then the following (RG_k is the reachability graph of the subnet $\mathcal{S}_k$):

1. for $k = 1, \ldots, K$, compute RG_k

2. for $k = 1, \ldots, K$, compute the matrix $\mathbf{Q}'_k$ from RG_k, using only local transitions

3. for each $t \in TS$
   for $k = 1, \ldots K$, compute $\mathbf{C}_k(t)$ and $\mathbf{A}_k(t)$ from RG_k:
   compute $c_k(t)_{m_k, m'_k}$
   compute $a_k(t)_{m_k, m'_k}$ as the diagonal compensation of the $(c_k(t)_{m_k, m'_k})$

4. compute the performance measures via the tensor expression of $\mathbf{Q}'$

This last calculus never uses directly $\mathbf{Q}'$, but in contrast matrices $\mathbf{Q}'_k$, $\mathbf{C}_k$ and $\mathbf{A}_k$.

In the general case, $\mathbf{Q}'$ has more non null terms than $\mathbf{Q}$ for two reasons: on the one hand, building of the TRG of $\mathcal{S}_k$ generates markings which are not projection of a global reachable marking; on the other hand, these unreachable local markings may build up global states (falsely) enabling a synchronization transition.
This problem of the deviation –which may be very large– between the potential and the actual reachability sets is at the present time handled in two ways. At the marking analysis level, we define "macro-views" of the subnet markings. Then, we can apply the so called two levels method that we detail below for the asynchronous composition, which is the most suited context for this approach. At the level of numerical resolution methods for $\boldsymbol{\pi}.\mathbf{Q} = \mathbf{0}$, the problem is to design algorithms allowing an efficient encoding of the reachability set in the potential reachability set [KEM 95, KEM 96] or to directly store the reachability set exploiting its structure [MIN 99].

Works carried out these last years on application of tensor decomposition methods to GSPN have virtually eliminated problems on the generator storage. These studies show that the algorithmic problem is now mainly the storage of the solution vector (vector of floating numbers with as many components as tangible markings, that is some around ten millions on a workstation today) and the possibility to use someone or other iterative method to solve large linear systems [CIA 99].

In most of the studies, the synchronization transitions are not immediate. This corresponds to the fact that an immediate synchronization transition will not be "visible" in the generator of the CTMC of the net. It is still possible [CIA 96] to get a tensor expression, but significantly more complex, with immediate synchronization transition.

3.2. *Asynchronous decomposition of generalised stochastic Petri nets*

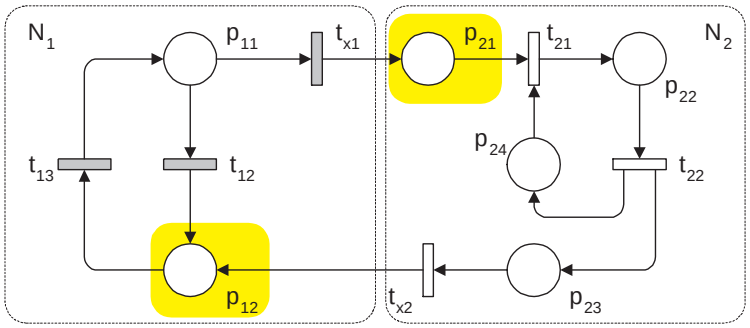

Figure 2: Asynchronous decomposition of GSPN

The asynchronous coordination, such as sending a message from the subsystem A to the subsystem B, models that an event arises in B, after arising in A. This kind of coordination is translated in Petri nets, by interface places receiving tokens from transitions controlled by other subnets. Thus, the asynchronous decomposition, studied initially by [BUC 92] corresponds to partition the set of transitions of the net into subsets $T_1, \ldots, T_K$. Each subset leads to a classification of places: a place is k-local iff it is only connected to transitions from T_k. A place is an interface place in the other case. A transition with output places being interface places of other subnets is a synchronization transition. For instance, in figure 2, the net is decomposed via subsets $T_1 = \{t_{x1}, t_{12}, t_{13}\}$ and $T_2 = \{t_{21}, t_{22}, t_{x2}\}$ of transitions. The two interface places are p_{21} and p_{12} and the two synchronization transitions are t_{x1} and t_{x2}. We define in this way K subnets, each one comprising the transitions of T_k, the k-local places and the corresponding arcs. In most of the proposed models, interface places connected to T_k are added, with associated arcs, to the kth subnet.

We notice immediately the fundamental difference with the synchronous composition: the subnets cannot be studied in isolation since they exchange tokens with other subnets. To take advantage of tensor methods, we are lead to introduce the notion of "environment" of a subnet summarising interactions of this subnet with the remainder of the net. But this environment may just be defined as soon as we have, for each subnet, a subnet summing up its own behavior, what we call an "abstract view" of a subnet. Hence, the approach is the following: for each subnet, we define an abstract view; we then build K extended subnets $\overline{\mathcal{N}}_k$ made up of a subnet $\mathcal{N}_k$ and of the abstract views of the $K-1$ other subnets (see figure 3 representing the extended subnet $\overline{\mathcal{N}}_1$);

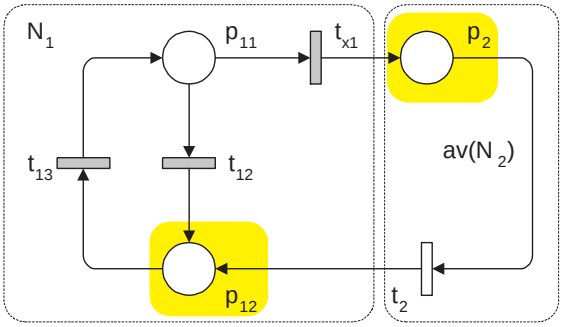

Figure 3: Extended subnet of the subnet $\mathcal{N}_1$ of figure 2

we study each extended subnet in isolation; the reachability set of the initial net is in bijection with a subset of the Cartesian product of the reachability sets of the $\overline{\mathcal{S}}_k$. The generator of the initial net may also be written as a tensor expression using elements of the $\overline{\mathcal{S}}_k$. In practice, we get abstract views in a structural way [CAM 97] or a posteriori from the markings [BUC 93].

The notion of abstract view obviously leads to reduction of the potential state space thanks to two levels methods introduced in section 2. The principle is to build, generally with the help of abstract views of the subnets, an abstract global view of a marking which leads to a partition of the markings of the net into subsets owning the same abstract global view.

3.3. *Tensor analysis of phase-type Petri nets*

Another application domain of tensor methods to SPN consists of phase-type distribution nets (see chapter 9) with transitions having various semantics (multiple/single-server, enabling/age memory, etc.). In this case, due to phase-type distributions, the Markov chain has a much larger size than the one of the reachability graph. Tensor methods allows us to maintain a reasonable ratio between these two sizes. Let us remain that a Markovian state is made up of a marking of the net and of the set of states of the phase-type distributions (descriptors), that is a tuple $(m, d_1, \ldots, d_K)$ if we have K phase-type distribution transitions.

The idea [DON 98] is to apply a two levels method (c.f. theorem 4). The abstract view of a Markovian state, termed extended marking, consists of its marking and of the number of interrupted clients in each service of phase-type distributions (there may be several number of interrupted clients for the same

marking). To this end, we build the extended reachability graph which holds transitions between extended markings.

It must be emphasized that, in this specific case, we get under slightly restrictive technical conditions not detailed here, the *exact description* of the Markovian state space denoted by MS:

$$\mathrm{MS} = \biguplus_{\mathbf{ei} \in \mathcal{EI}} [\mathbf{ei}] \times \mathrm{D}_1(\mathrm{ei}_1) \times \cdots \times \mathrm{D}_H(\mathrm{ei}_H) \qquad [3]$$

where **ei** is an extended marking, [**ei**] is the set of corresponding markings (of the net), $\mathcal{EI}$ is the set of all extended markings and $\mathrm{D}_h(\mathrm{ei}_h)$ is the set of possible states of the phase-type distribution of the transition t_h under conditions **ei**.

Moreover, and this is also noticeable, we can derive the ergodicity properties of the continuous time Markov chain of the net from its structural characteristics.

At last, the block matrices of the generator of the net are tensor expressions made up of terms which can be computed from each subspace. The reader will find exact expressions in [DON 98].

4. Tensor decomposition of stochastic well formed Petri nets

To extend the application sphere of performance evaluation methods to more and and complex systems, it is appealing to combine Markovian aggregation methods and tensor decomposition methods. In this way, we hope to get an aggregated CTMC being a "tensor composition" of smaller aggregated CTMC. The phases of the method are then:

- build a decomposition of the state space E, giving $E \subseteq E' = \prod_{k=1}^{K} E_k$;
- use an aggregation method satisfying the strong aggregation condition (see chapter 10) for each of the CTMC $(E_k, \mathbf{Q}_k)$, leading to $\widetilde{E_k} = \{E_k^{(j)} \mid j = 1, \ldots, n_k\}$ with generators $\widetilde{\mathbf{Q}_k}$;
- build the product $(\widetilde{E}' = \prod_{k=1}^{K} \widetilde{E_k}, \widetilde{\mathbf{Q}} = f(\widetilde{\mathbf{Q}_1}, \ldots, \widetilde{\mathbf{Q}_K}))$ of the aggregated CTMC and define the aggregated image $\widetilde{E} \subseteq \widetilde{E}'$ of E.

Unfortunately, as a general rule, $(\widetilde{E}', \widetilde{\mathbf{Q}})$ *is not* a super-set of an exact aggregation of $(E, \mathbf{Q})$. So, we are led [HAD 95, HAD 96b] to find conditions under which such a combination of methods is feasible. In this section, we explain the problems raised by this approach and the results obtained.

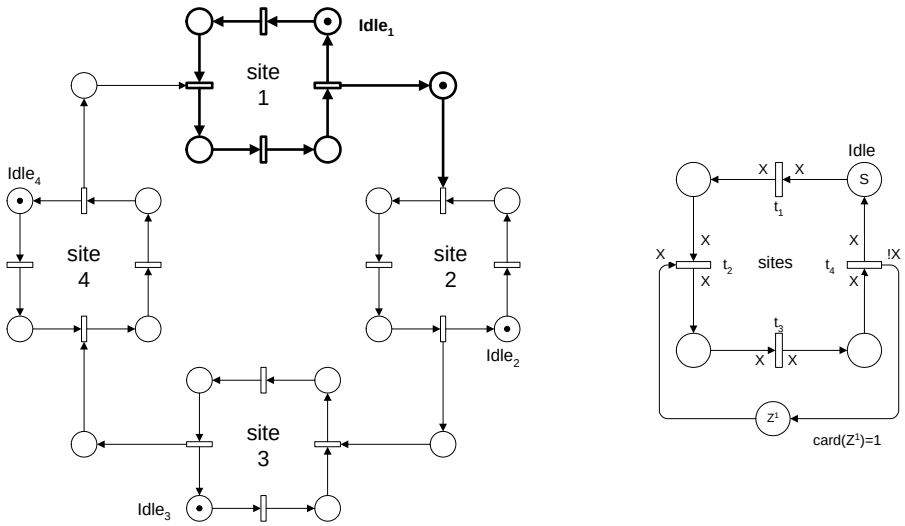

Figure 4: GSPN and SWN of a virtual token ring

4.1. *Problems*

Two fundamental problems arise for a combination of these methods: the first one regards the specification of the studied system and the second one concerns the used resolution method.

4.2. *The specification problem*

If the system to be modeled owns synchronization between the same kind of entities (server processes for instance), we can build a "synchronized product" of submodels, each one modeling the behavior of one entity; but, since we model *each entity* with *one submodel*, there is no entity class and we cannot introduce aggregation.

An elementary example of this kind of situation is a system of sites running sequential code with a critical section the execution of which is allocated in a cyclic way to each site (virtual token ring). The GSPN and the SWN of such a system (with 4 sites) are presented in figure 4: starting from the Idle state, each site execute an initial work (transition t_1), then waits for the mutual exclusion token to go on (transition t_2). When the critical section is over, the site releases the mutual exclusion token (transition t_4) and goes back idle. In

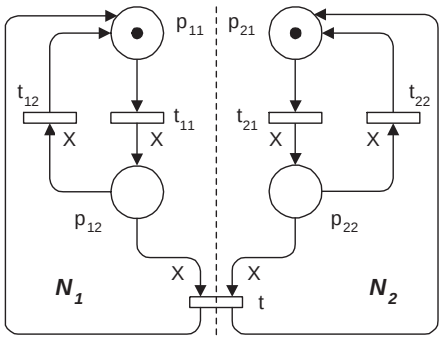

Figure 5: Synchronous composition of SWN with memory

the SWN model, we have only one base class, C_s, for the sites. The marking S means that all sites are idle in the initial state and the marking Z^1 (dynamic subclass) shows that this place holds *some token* of the color class C_s.

As we see in this example, with Petri net modeling, synchronization between objects of the same kind is translated by "folding" the uncolored net into a *single* colored net. W note that we could also decompose the net in four GSPN (one of them is drawn with thick lines) and an asynchronous composition.

We summarize this situation in the term *internal synchronization*, since it is a matter of synchronization between objects of the kind. Conversely, we will say that the system exhibits an *external synchronization* if there are synchronization between entities of different kinds.

So, in the internal synchronization case, we have to choose between a model decomposition into submodels (SAN or synchronized GSPN) and a possible aggregation (SWN).

4.3. *The resolution problem*

A system with external synchronization may be modeled as a SWN $\mathcal{N}$, *synchronous or asynchronous composition* of K SWN $\mathcal{N}_k$: in this way, we hope to make use of an aggregation at the level of each subnet and to apply the tensor (de)composition for the global net.

Let us give an example of such a composition with a system built from two subsystems: in each subsystem, the activity begins with the choice of the kind of task to be done (represented by the class C); then, the task may be

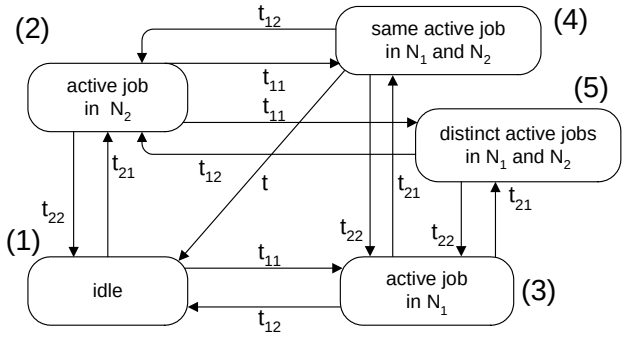

Figure 6: Synchronous composition of SWN with memory: SRG of $\mathcal{S}$

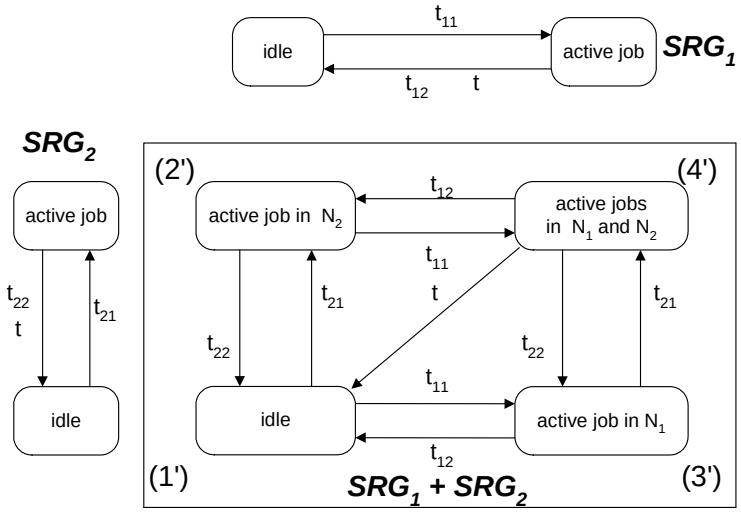

Figure 7: Synchronous composition of SWN with memory: SRG of $\mathcal{S}_1$, $\mathcal{S}_2$ and SRG of their "synchronized product"

completed either in an autonomous way in each subsystem, or else jointly by both, for the same kind of task in the two subsystems. The SWN of figure 5 is a model of this system. The net is the composition of two SWN $\mathcal{N}_1$ and $\mathcal{N}_2$ through t: tasks are carried out either autonomously in $\mathcal{N}_1$ through t_{12} (resp. $\mathcal{N}_2$ through t_{22}), or else, for the same kind of task (which is translated by the same X function for arcs linking p_{12} and p_{22} to t) in both nets through t. The color domain of p_{12} and p_{22} is the class C and the color domain of p_{11} and p_{21} is the neutral color. The color domain of all transitions is C. We give in figure 6 the symbolic reachability graph (SRG) of $\mathcal{S}$ and, in figure 7, the SRG of $\mathcal{S}_1$, $\mathcal{S}_2$ and of their "synchronized product", in an informal way.

The autonomous work in $\mathcal{N}_1$ (resp. $\mathcal{N}_2$) is translated in figure 6 by the firing sequences (t_{11}, t_{12}) (resp. (t_{21}, t_{22})). We also note that t is enabled in *only one symbolic marking (4)* which represents markings with the *same* kind of available objects in p_{12} and p_{22}, while in the marking (5), p_{12} and p_{22} hold *different* kinds of objects.

In contrast, in figure 7, we have only one "symbolic marking" (4') with tokens in p_{12} and p_{22}: the relative identity of these tokens (defined by the firings of t_{11} and t_{21}) is lost and (4') is the gathering of (4) and (5), which is a wrong aggregation since (4) enables t but (5) does not.

We call such transitions (t), *synchronization transitions with memory.*

We state that, as a general rule, we cannot use a straightforward extension of the composition of GSPN to *solve* the initial CTMC because the composition, i.e. the sum of the graphs[4] of the aggregates given by the symbolic reachability graphs of the $\mathcal{S}_k$ *is not an aggregation of the CTMC of the whole model* satisfying the aggregation condition of Kemeny and Snell [KEM 60] (chapter 10, Proposition 5).

This situation is mainly due to the fact that the firing of a synchronization transition generates modifications of markings which will forbid some admissible color permutations, that is to say, they will reduce the possible symmetries in each subnet. We can say that the (stochastic) transition system must "memorize" these firings, and a direct composition of such systems does not allow it.

Let us emphasize that this memory problem is a general one and we encounter it with all models (SWN, SAN, GSPN, ...): for instance, [PLA 85] introduces auxiliary automata to store specific states of the system. In contrast, we wish not to modify the initial net to apply a decomposition.

At last, even if we succeed in defining a "synchronized product" of SWN,

[4]The sum of $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ is the graph $G = G_1 + G_2 = (V_1 \times V_2, E)$ with $((u_1, u_2), (v_1, v_2)) \in E$ iff $(((u_1, v_1) \in E_1)$, and $((u_2 = v_2)$, or $((u_2, v_2) \in E_2))$, or $(((u_2, v_2) \in E_2)$, and $((u_1 = v_1)$, or $((u_1, v_1) \in E_1))$.

it remains to express the firing rate of external transitions from information given by their SRG.

4.4. *A tensor decomposition method for SWN*

Figure 8 summarizes the approach we made us of, and places it in its context. To compute performances measures of modeled systems, the basic method (left branch of the diagram) computes the reachability graph of the net (arcs "G" -Graph building-) et derives Q (arcs "M" -Markov chain-) from it.

The two methods, exposed above and in chapter 10 are:

- the tensor decomposition method (right branch of the diagram, with RG_k, $\mathbf{Q}'$);
- the aggregation method (SRG branch, $\widetilde{\mathbf{Q}}$) used by the SWN model.

The validity of the expressions of the new generator ($\widetilde{\mathbf{Q}}$, $\mathbf{Q}'$) for these two methods (arcs "C" -Consistancy-) has been established.

The proposed approach consists in combining aggregation and decomposition (central branch of figure 8). We have three points to study:

1. build *modified* SWN, that is to say extended, (denoted by $\overline{\mathcal{N}_k}$) of the subsystems, allowing us to memorize the synchronization;
2. derive an expression of the generator $\overline{\mathbf{Q}}'$ of the underlying CTMC of the composition of the SRG (denoted by $\overline{SRG}_k$) of the ($\overline{\mathcal{S}}_k$), analogous to the ones obtained for GSPN;
3. prove that this expression is a "super-matrix" of an aggregation of $\mathbf{Q}$.

In this context, we have defined kinds of synchronous and asynchronous compositions for which we give explicit building methods of the extended subnets (point 1). We also get an expression of the generator $\overline{\mathbf{Q}}'$ with adapted matrices $\mathbf{A}_k(t)$ and $\mathbf{C}_k(t)$ for each case (point 2), and we prove consistency of these expressions (point 3).

4.5. *Application in the asynchronous case*

For simplicity, we expose only the asynchronous case. We first present the extension method of subnets: we clarify the structure of the synchronization,

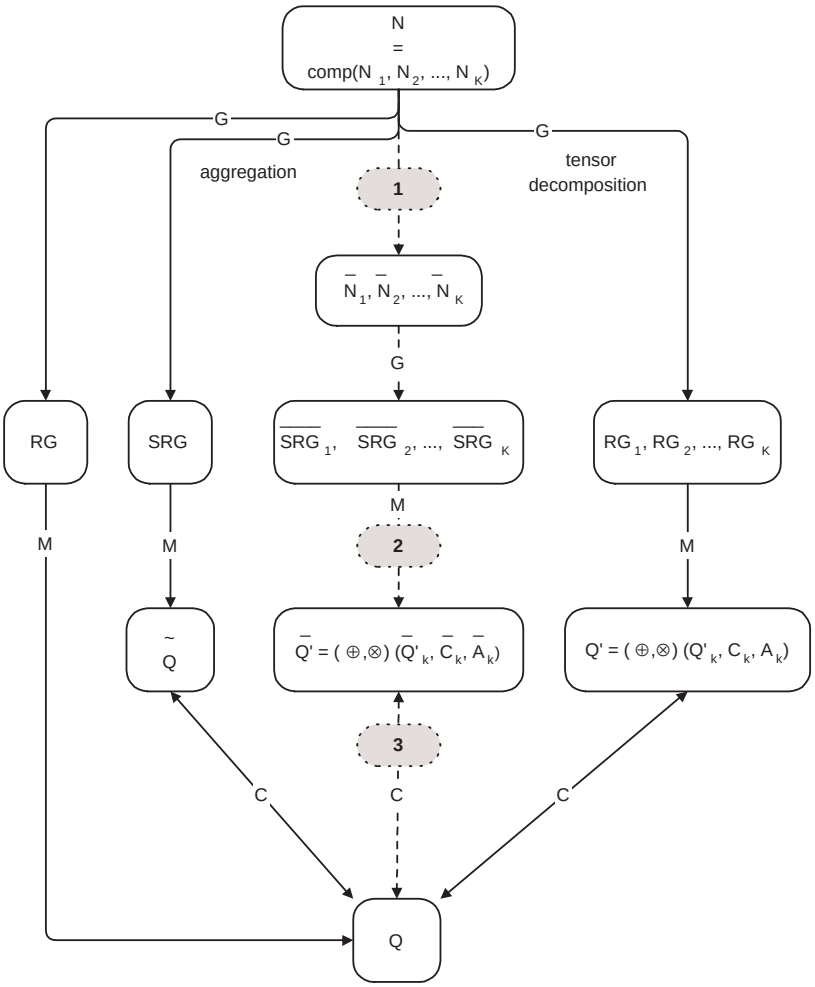

Figure 8: Approach and context of the decomposition of SWN

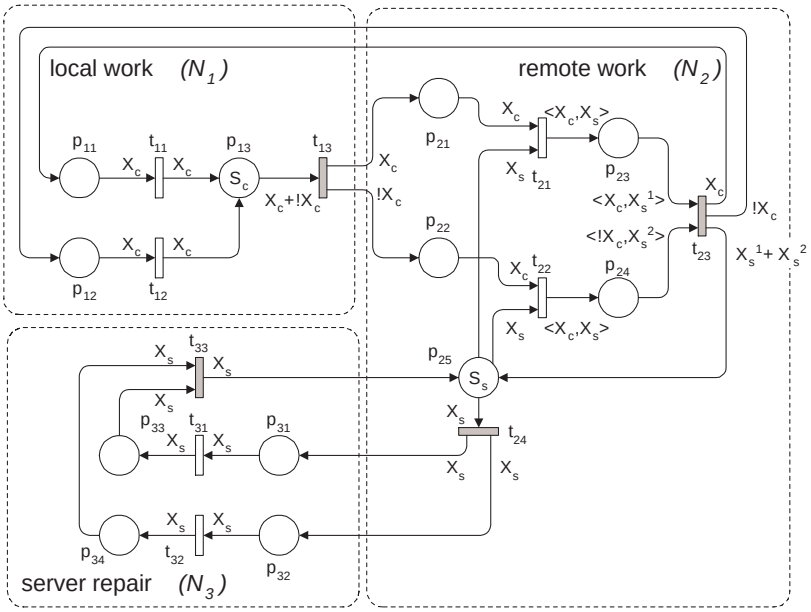

Figure 9: Example of asynchronous decomposition of SWN

and then, we point out how to build extended SWN which will be studied in isolation. Next, we define *syntactic conditions*, that is to say at the structural level of the net (color domains of places and transitions, arc functions, flows, ...), satisfied by many nets decomposable into subnets with some "autonomy", for which the tensor composition of aggregates may apply.

Finally, we give an overview of the computation algorithms and associated consistency proofs. We refer the reader to [HAD 96a, HAD 97, MOR 96] for detailed expositions of results presented below.

Here, we first have to build an extension of subnet allowing us both to study it in isolation taking its environment into account, and to define an (asynchronous) decomposition of subnets. This is done through the definition of an *abstract view* of each subnet: each color class modeling entities "moving" from one subnet to another one is called a *global class* and the abstract view of a subnet $\mathcal{N}_k$ is made up of one place for each global class of $\mathcal{N}_k$, and of the set of transitions TS_k modified accordingly.
Let us emphasize that this abstract view is *formally defined*, in contrast with other works on this subject, and so it may be automatically computed, thanks to symbolic partial flows (see chapter 7). The extension of a given $\mathcal{N}_k$ is then $\mathcal{N}_k$ enlarged with the abstract views of all other subnets.

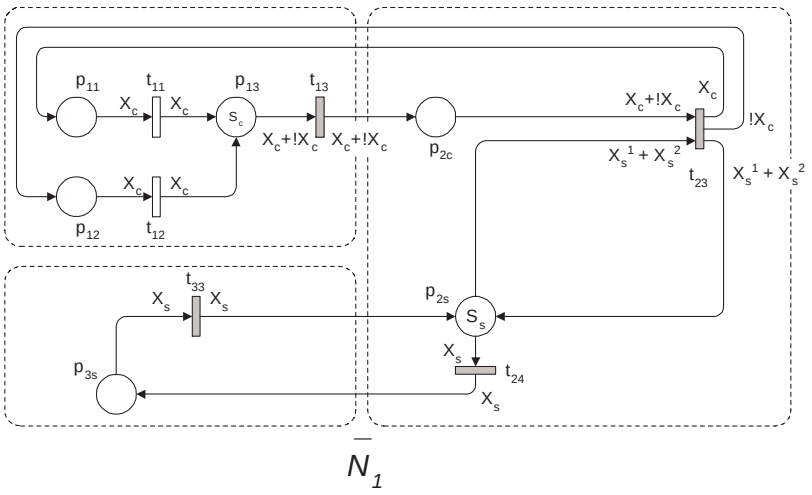

Figure 10: Extension $\overline{\mathcal{N}}_1$ of the net $\mathcal{N}_1$ of figure 9

Figure 9 is an example of asynchronous composition of stochastic well formed Petri nets. $\mathcal{N}$ is made up of three parts and models a client (local work) - server (remote work) with a repair device (server repair). Clients are initially in the local part (place p_{13}) and servers are in the remote site (place p_{25}). Clients send requests *neighbor paired* (variables X and $!X$, the client class C_c is ordered). Requests are served at the remote site (transitions t_{21}, t_{22} and t_{23}) by the servers (class C_s). A server may fail: in this case, it must be repaired through two tasks in the repair site (transitions t_{31}, t_{32} and t_{33}).

Thus, the basic color classes are C_c and C_s and the color domains of places and transitions (not given in the figure for clarity) are:
- C_c for p_{11}, p_{12}, p_{13}, p_{21}, p_{22}, t_{11}, t_{12} and t_{13};
- C_s for p_{25}, p_{31}, p_{32}, p_{33}, p_{34}, t_{24}, t_{31}, t_{32} and t_{33};
- $C_c \times C_s$ for p_{23}, p_{24}, t_{21} and t_{22};
- $C_c \times C_s^2$ for t_{23}.

Figure 10 gives the extension $\overline{\mathcal{N}}_1$ of $\mathcal{N}_1$. We see that, in the abstract view of $\mathcal{N}_2$, we have two places (p_{2c} and p_{2s}) for colors C_c and C_s. The corresponding partial flows of $\mathcal{N}$ are:

$$\begin{aligned} f_{2c} &= X_c.p_{21} + X_c.p_{23} + X_c.p_{22} + X_c.p_{24} \\ f_{2s} &= X_s.p_{25} + X_s.p_{23} + X_s.p_{24} \end{aligned}$$

In the abstract view of $\mathcal{N}_3$, we have only one place (p_{3s}) for C_s. The associated

partial flow is:

$$f_{3s} \quad = \quad X_s.p_{31} + X_s.p_{33}$$

Transitions t_{13}, t_{23}, t_{24} and t_{33} are also modified according to the definition of an abstract view.

We can define a set of syntactic conditions which allows us to use the method [HAD 97, MOR 96]. Intuitively, these conditions render three properties:

- on the one hand, all activities must keep their identity when going from one subnet to another one; for instance, we find for output of t_{23}, one client X and its successor $!X$, which are in its input places p_{23} and p_{24}. Generally speaking, this condition is expressed by a property binding the flows of the abstract view to the colors of the control places of the synchronization transitions;

- on the other hand, there may be only *transfer* of activities between subnets, but no creation nor destruction; for instance, the arc functions of t_{24}, induce the transfer of only one server to the repair service;

- finally, at any time, activities of a given global color are restricted to only one subnet; the flow $X_c.p_{11} + X_c.p_{12} + X_c.p_{13} + f_{2c}$ in $\mathcal{N}$ ensures this property for any color of the client class (thus, this condition is ensured by symbolic flows).

4.5.1. *Algorithm for computing performance measures*

The matrix $\mathbf{Q}$ is a submatrix of

$$\mathbf{Q}' = \bigoplus_{k=1}^{K} \mathbf{Q}'_k + \sum_{t \in TS} \sum_{d} \mathbf{w}[t](d) \left[\bigotimes_{k=1}^{K} \mathbf{C}_k(t,d) - \bigotimes_{k=1}^{K} \mathbf{A}_k(t,d) \right] \qquad [4]$$

where $\mathbf{w}[t](d)$ is the rate[5] of the transition t and d is a choice of static subclasses for the symbolic firings of t. This formula extends the relation [2] for GSPN composition. The computation algorithm of $\mathbf{Q}'$ has same steps as the one presented in section 3.1; however, it must be adapted to colored firings, which implies a careful analysis of the firings of synchronization transitions.

[5]We assume that the rate is independent of the marking.

4.5.2. *Sketch of proof of the algorithm*

We denote by $\overline{SRG}_k$ (resp. by $\overline{SRS}_k$), the SRG (resp. SRS) of $\overline{\mathcal{N}}_k$ and by $\overline{\mathcal{M}}_k$ its symbolic markings. $\overline{XSRS}$ is the Cartesian product of the $\overline{SRS}_k$ and its elements are denoted by $\overline{\mathcal{M}}$ (hence $\overline{\mathcal{M}} = (\overline{\mathcal{M}}_k)_{k=1,\ldots,K}$). We first show[6] that the reachability set RS of the net is a subset of $\{(M_k)_{k\in K} \mid \exists \overline{\mathcal{M}} \in \overline{XSRS}$ such that $\forall k \in K,\ \overline{M}_k \in \overline{\mathcal{M}}_k\}$ which may be seen as a "desaggregated" state space of $\overline{XSRS}$.

Then, we prove that $\mathcal{A}(M) = \overline{\mathcal{M}} \in \overline{XSRS}$ is an aggregation function on RS which ensures exact aggregation.

Finally, we show that the transition rate (in the sense of the Markov chain) from a state $\overline{\mathcal{M}}$ of $\overline{XSRS}$ to another state $\overline{\mathcal{M}}'$ is indeed given by the proposed algorithm.

The proof of the first two points is established in several steps [MOR 96]:

- definition of a set of *semantic conditions*, that is to say at the marking level;
- verification of the strong aggregation condition as a consequence of the previous semantic conditions (this is the main part of the proof);
- proof that the syntactic aggregation conditions imply these semantic conditions.

At last, a minutely detailed examination of the firing colors of the synchronization transitions allows us to prove that the generator of the aggregated CTMC is a submatrix of the matrix $\mathbf{Q}'$ of the algorithm.

Introducing an intermediate semantic level is justified by two reasons:

- semantic conditions help us looking for syntactic conditions: for each of the formers, we try to establish a syntactic translation;
- for a given set of semantic conditions, we may find several sets of syntactic conditions for specific classes of nets. In this way, we reduce the consistency proof to the derivation of the semantic conditions from these new syntactic conditions.

[6] The structure of the proof is analogous in the asynchronous and the synchronous cases.

5. Conclusion

Tensor methods are based on a decomposition of discrete event systems into synchronized subsystems. Applying these methods to various classes of stochastic Petri nets takes advantage of structural and behavioral properties of these nets. For GSPN, we compose several subnets in a synchronous or an asynchronous manner. For phase-type distribution nets, we describe the state space and the generator of the Markov chain in a tensor way, taking into account the possible states of the phase-type activities. In all cases, subtle implementation techniques allow us to exploit at best the tensor properties, thus increasing the size of the systems we are able to analyze. At last, combination of tensor methods and methods based on behavioral symmetries (stochastic well formed Petri nets), takes advantage of the two kinds of complexity reduction when these two methods are non incompatible.

References

[BUC 92] P. Buchholz. A hierarchical view of GCSPNs and its impact on qualitative and quantitative analysis. *Journal of Parallel and Distributed Computing*, 15(2):207–224, 1992.

[BUC 93] P. Buchholz. Hierarchies in colored GSPNs. In *Proc. of the 14th International Conference on Application and Theory of Petri Nets*, number 691 in LNCS, pages 106–125, Chicago, Illinois, USA, June 1993. Springer–Verlag.

[BUC 97] P. Buchholz, G. Ciardo, S. Donatelli and P. Kemper. Complexity of Kronecker operations on sparse matrices with applications to solution of Markov models. Technical report 97-66, ICASE, Institute for Computer Applications in Science and Engineering, NASA/Langley Research Center, Hampton, VA, USA, 1997.

[CAM 97] J. Campos, M. Silva and S. Donatelli. Structured solution of stochastic DSSP systems. In *Proc. of the 7th International Workshop on Petri Nets and Performance Models*, pages 91–100, Saint-Malo, France, June 3–6 1997. IEEE Computer Society Press.

[CIA 96] G. Ciardo and M. Tilgner. On the use of Kronecker operators for the solution of generalized stochastic Petri nets. Technical report 96-35, ICASE, Institute for Computer Applications in Science and Engineering, NASA/Langley Research Center, Hampton, VA, USA, May 1996.

[CIA 99] G. Ciardo and A. Miner. A data structure for the efficient Kronecker solution of GSPNs. In *Proc. of the 8th Int. Workshop on Petri nets and performance models (PNPM99)*, pages 22–31, Zaragoza, Spain, September 8–10 1999. IEEE Comp. Soc. Press.

[DAV 81] M. Davio. Kronecker products and shuffle algebra. *IEEE Transactions on Computers*, 30(2):116–125, 1981.

[DON 94] S. Donatelli. Superposed generalized stochastic Petri nets: definition and efficient solution. In Robert Valette, ed, *Proc. of the 15th International Conference on Application and Theory of Petri Nets*, number 815 in LNCS, pages 258–277, Zaragoza, Spain, June 20–24 1994. Springer–Verlag.

[DON 98] S. Donatelli, S. Haddad and P. Moreaux. Structured characterization of the Markov chains of phase-type SPN. In *Proc. of the 10th International Conference on Computer Performance Evaluation. Modelling Techniques and Tools (TOOLS'98)*, number 1469 in LNCS, pages 243–254, Palma de Mallorca, Spain, September 14–18 1998. Springer–Verlag.

[FER 98] P. H. L. Fernandes. *Méthodes numériques pour la solution de systèmes markoviens à grand espace d'états*. Thèse, Institut National Polytechnique de Grenoble, Grenoble, France, février 1998.

[HAD 95] S. Haddad and P. Moreaux. Evaluation of high level Petri nets by means of aggregation and decomposition. In *Proc. of the 6th International Workshop on Petri Nets and Performance Models*, pages 11–20, Durham, NC, USA, October 3–6 1995. IEEE Computer Society Press.

[HAD 96a] S. Haddad and P. Moreaux. Aggregation and decomposition for performance evaluation of synchronous product of high level Petri nets. Document du Lamsade 96, LAMSADE, Université Paris Dauphine, Paris, France, September 1996.

[HAD 96b] S. Haddad and P. Moreaux. Asynchronous composition of high level Petri nets: a quantitative approach. In *Proc. of the 17th International Conference on Application and Theory of Petri Nets*, number 1091 in LNCS, pages 193–211, Osaka, Japan, June 24–28 1996. Springer–Verlag.

[HAD 97] S. Haddad and P. Moreaux. Aggregation and decomposition for performance evaluation of asynchronous product of high level Petri nets. Document du Lamsade 102, LAMSADE, Université Paris Dauphine, Paris, France, May 1997.

[KEM 60] J. G. Kemeny and J. L. Snell. *Finite Markov Chains*. V. Nostrand, Princeton, NJ, 1960.

[KEM 95] P. Kemper. Numerical analysis of superposed GSPNs. In *Proc. of the 6th International Workshop on Petri Nets and Performance Models*, pages 52–61, Durham, NC, USA, October 3–6 1995. IEEE Computer Society Press.

[KEM 96] P. Kemper. Reachability analysis based on structured representations. In *Proc. of the 17th International Conference on Application and Theory of Petri Nets*, number 1091 in LNCS, pages 269–288, Osaka, Japan, June 24-28 1996. Springer–Verlag.

[MIN 99] A.S. Miner and G. Ciardo. Efficient reachability set generation and storage using decision diagrams. In *Proc. of the 20th International Conference on Application and Theory of Petri Nets*, number 1639 in LNCS, pages 6–25, Williamsburg, VA, USA, June 21–25 1999. Springer–Verlag.

[MOR 96] P. Moreaux. *Structuration des chaînes de Markov des réseaux de Petri stochastiques. Décomposition tensorielle et agrégation*. Thèse, Université Paris Dauphine, Paris, France, 11 décembre 1996.

[PLA 85] B. Plateau. On the stochastic structure of parallelism and synchronization models for distributed algorithms. In *Proc. of the 1985 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, pages 147–154, Austin, Texas, USA, August 1985. ACM.

[PLA 91] B. Plateau and J.M. Fourneau. A methodology for solving Markov models of parallel systems. *Journal of parallel and distributed computing*, 12:370–387, 1991.

[STE 94] W. J. Stewart. *Introduction to the numerical solution of Markov chains*. Princeton University Press, USA, 1994.

Index