Petri Nets Compositional Modeling and Verification of Flexible Manufacturing Systems

Paolo Ballarini[1], Hilal Djafri[2], Marie Duflot[1], Serge Haddad[2], Nihal Pekergin[1]
LACL[1], Université Paris-Est Créteil, France
LSV[2], ENS-Cachan, France
Email: {paolo.ballarini,duflot,nihal.pekergin}@u-pec.fr, {djafri,haddad}@lsv.ens-cachan.fr

***Abstract*—Flexible Manufacturing Systems (FMS) are amongst the most studied types of systems, however due to their increasing complexity, there is still room for improvement in their modeling and analysis. In this paper we consider the design and the analysis of stochastic models of FMS in two complementary respects. First we describe a (stochastic) Petri Nets based compositional framework which enables to model an FMS by combination of an arbitrary number of basic components. Second we demonstrate how *classical* transient-analysis of manufacturing systems, including *reliability* and *performability* analysis, can be enriched by application of a novel, sophisticated stochastic logic, namely the Hybrid Automata Stochastic Logic (HASL). We demonstrate the proposed methodology on an FMS example.**

I. Introduction

Analysis of Flexible Manufacturing Systems (FMS). FMS have been introduced in order to optimize different criteria of manufacturing systems. For instance, one wants to reduce the number of workstations in order to efficiently manage crashes, increase the productivity and the flexibility, etc. In such a context, a critical issue consists in evaluating these criteria and comparing different architectures before selecting the appropriate one. This implies to resort to formal models and evaluation methods.

Modeling FMS with Petri Nets. The Petri net formalism is applied in numerous application areas. Compared to other formalisms, Petri nets are appropriate to model concurrent activities (each one described by a finite automaton) sharing resources (described by additional places) and communicating via synchronization (described by transitions). Since FMS present such characteristics, they are a good candidate to be modeled and analyzed with the help of Petri nets [WD98].

Indeed several approaches have been undertaken differing w.r.t. their goals and the kind of nets used for modeling. When one is interested in qualitative properties of FMS like deadlock prevention, modeling is based on structural subclasses of Petri nets allowing to design efficient algorithms [ECM95]. When one is interested in performance of FMS, there are (at least) two possible modeling approaches: either to substitute discrete quantities by continuous ones leading to an hybrid Petri net [BGS01] or to represent the uncertainties related to the FMS behavior by distribution probabilities leading to a stochastic Petri net [MBC$^+$95], [LCGH93]. Here we follow the latter approach.

Limitations of Petri net modeling of FMS. There are two drawbacks of Petri nets w.r.t. the modeling. First there are no net operators that would lead to a compositional modeling. In [BDK01], Petri nets have been extended with operators. This is an interesting theoretical approach but the subnets are not viewed as components. For instance, they do not own an interface and an internal part. Second, the syntax and semantics of nets may prevent modelers used to their dedicated formalism to switch to Petri nets.

Steady-state and transient-analysis of FMS. Although the vast majority of FMS stochastic modeling studies have been focused on the analysis of *steady-state*-based measures (such as, for example, *throughput*, *productivity*, *makespan*) the relevance of *transient-analysis* of FMS models has been extensively demonstrated [NV94]. For instance, as soon as faults are modeled, transient measures like the time until FMS stopping are interesting. Furthermore it is well known that for systems presenting regenerative points (like idle states), every steady-state measure may be obtained by averaging the corresponding transient measure between two occurrences of a regenerative point.

Our contribution. In [BDD$^+$11], we have presented a framework with a dedicated prototype tool COSMOS for analyzing complex systems modeled by stochastic Petri nets via quantitative model-checking of formulas specified in an expressive language HASL. Here we show an high-level modeling approach for FMS developed over this framework presenting the following features.

- **A compositional framework targeted to FMS modeling.** Our framework allows to build arbitrarily large/complex models of FMS by composition of basic elements representing the elementary parts of a FMS. Following an object-oriented approach, we start with three generic classes: the load unit class, the machine class and the transportation class. Then the modeler specializes these classes in order to express the characteristics of his specific architecture. This specialization concerns both the qualitative features like the routing policy of a transporter and the quantitative features like the loading time of a unit. By instantiating such classes into components and gluing them through their interface, he finally produces the FMS architecture. During the modeling stage, Petri net patterns associated with

specialized classes are automatically generated. Then these patterns are duplicated to reflect the components corresponding to the classes and linked via place merging to obtain the final stochastic Petri net. Observe that the net is managed internally so that one does not require any Petri net knowledge from the user.

- **A set of formal properties customized for analysis of FMS.** As discussed before, several specific qualitative and quantitative properties of FMS are relevant. Our framework includes a set of FMS-oriented properties described in natural language including the appropriate parameters w.r.t. the property. Once the user fixes the parameters, an HASL formula is automatically generated and later evaluated over the SPN corresponding to the FMS. Since HASL is very expressive the specification of almost all relevant properties for FMS does not present any difficulty.

Organization. In section II, we introduce the syntax and semantics of SPNs. Then we present our modeling approach in section III. We illustrate the approach on a toy example in section **??** and the adequation of HASL formulae for analysis of FMS in section IV. Finally we conclude and give perspectives to this work in section V.

II. Stochastic Petri Nets

Since the semantics of a stochastic Petri net (SPN) is a discrete-event stochastic process (DESP), we briefly remind what is a DESP. A DESP is given by a state space $\mathcal{S}$ and two families of random variables $\{S_n\}$ and $\{T_n\}$ indexed by $\mathbb{N}$. S_0 is the initial state and $T_0 = 0$ is the initial time. For every $n > 0$, $S_n \in \mathcal{S}$ is the state after the occurrence of the nth event and $T_n \in \mathbb{R}_{\geq 0}$ is the occurrence time of this event.

In order to equip Petri nets with DESP as semantic, we need to add information to the net. More precisely we must fix the three policies ruling the behaviour of the net: the server policy, the choice policy and the memory policy. The server policy is related to the delay before firing a transition in a marking. So we associate a distribution with every marking (state of the net) and transition that represents the random delay before the firing of a transition. The choice policy is related to the next transition to be fired. We choose a race-based policy meaning that one of the transitions with the shortest delay should be selected. In order to solve the problem of equal delays, we add (marking-dependent) priorities to transitions and in case of both equal delays and priorities we add (marking-dependent) weights to transitions in order to perform a random choice. Finally, the sampled delay of each enabled transition different from the selected one is decremented by the minimal delay (i.e. enabling memory policy). This leads to the following definition.

Definition 1: $\mathcal{N} = (P, T, W^-, W^+, I, dist, prio, wght, m_0)$ a *stochastic Petri net* is defined by:

- P the finite set of *places*. A *marking* is an item m of $\mathbb{N}^P$ and $m(p)$ is the corresponding marking of place p;
- T the finite set of *transitions* with $P \cap T = \emptyset$;
- W^- (resp. W^+) the backward (resp. forward) incidence matrix from $P \times T$ to $\mathbb{N}$;
- I the inhibitor matrix from $P \times T$ to $\mathbb{N}_{>0} \cup \{\infty\}$;
- $dist$ the distribution mapping from $\mathbb{N}^P \times T$ to the set of probability distributions over $\mathbb{R}_{\geq 0}$;
- $prio$ the priority mapping from $\mathbb{N}^P \times T$ to $\mathbb{N}$;
- $wght$ the weighting mapping from $\mathbb{N}^P \times T$ to $\mathbb{R}_{>0}$;
- m_0 is the initial marking.

A Petri net is graphically represented by a bipartite graph. The vertices are the places (circles) and transitions (rectangles). There is an arc from place p (resp. transition t) to t (resp. p) labelled by $W^-(p,t)$ (resp. $W^+(p,t)$) if $W^-(p,t) > 0$ (resp. $W^+(p,t) > 0$). There is an inhibitor arc from place p to t (denoted by an arc ending in a circle) if $I(p,t) < \infty$. The labels equal to one are omitted. The marking of a place is represented by tokens (small black circles) in the place. Figures 3 and 4 are typical examples of Petri nets.

Usual notions for Petri nets equally apply in the context of SPNs. A transition t is *enabled* in a marking m if $\forall p \in P,\ (m(p) \geq W^-(p,t) \wedge m(p) < I(p,t))$. The *firing* of a transition t enabled in m leads to marking m' defined by $\forall p \in P\ m'(p) = m(p) - W^-(p,t) + W^+(p,t)$.

Let us describe the stochastic process associated with an SPN. The state space $\mathcal{S}$ is defined by a marking say m and a family of (remaining) durations indexed by the transitions, say dur_t such that $dur_t = \infty$ iff t is disabled in m. Assume that $S_n \equiv (m, \{dur_t\}_{t \in T})$ and T_n are given then:

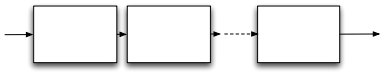
- Let $dmin \equiv \min(dur_t)$. If $dmin = \infty$ then $S_{n+1} = S_n$ and $T_{n+1} = T_n$ (the current marking is a deadlock).
- Otherwise $T_{n+1} = T_n + dmin$. Now let T' be the subset of transitions t with $dur_t = dmin$ and (equal) highest priority for m (i.e. $prio(m,t)$). Whenever T' is not a single element, sample a random choice between transitions of T' with probabilities proportional to $wght(m,t)$.
- Let $t^* \in T'$ be the selected transition. Then $S_{n+1} \equiv (m', \{dur'_t\}_{t \in T})$ is defined by:
  - m' is the marking reached by the firing of t^* in m.
  - Let t be a transition enabled in m'. If $t \neq t^*$ and t was already enabled in m then $dur'_t = dur_t - dmin$. Otherwise sample dur'_t w.r.t. the distribution $dist(m', t)$.

The significant measures for an SPN are related to the *random execution path* associated with the DESP. Transient measures fix some time horizon τ and can be defined w.r.t. the random marking at time τ like the (mean) marking of some place or can be defined all along the whole path like the (mean) maximal marking of some place along the path. τ may be fixed to be the time that an event happens, like the failure of a component etc. Steady-state measures assume that the DESP will asymptotically reach some marking distribution and express properties of this distribution. For models with regenerative points, the finite random execution paths between consecutive regenerative points can be used to derive steady-state measures. Most of qualitative and performability measures can be expressed using formulas of our language HASL (see subsection IV-A).

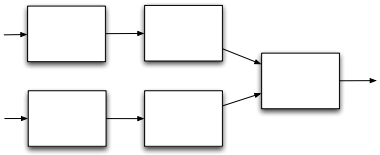
III. Compositional FMS modeling using Petri Nets

We introduce a stochastic Petri net (SPN) based modeling framework for FMS, by means of which a model of an

FMS can be obtained through assembling of the desired combination of *basic components*. FMS are, by nature, heterogenous systems which may differ from one another both in the functionality of components as well as on the workflow of the production process. The compositional framework we introduce is designed to cope with the heterogeneity of FMS. For example, components for modeling of a simple *linear FMS* (whose workflow is drafted in Figure 1(a)), whereby the end product is obtained by processing of a single type of raw material through a line of machines connected by a conveyor belt, will be (internally) different from components for modeling of a more complex FMS (e.g. workflow drafted in Figure 1(b)), involving multiple types of material (m_a and m_b), machine selection (e.g. workpiece w'_1 outputted by machine M_1 is delivered to either M_3 or M_5) and workpieces combination (e.g. workpieces w''_1 and w''_2 are combined by machine M_6).



(a) A simple linear FMS with single type of raw material



(b) An FMS with 2 types of raw materials, machine-selection and workpiece-combination

Fig. 1. Different types of workflow for FMSs

A. Principles of the proposed approach

Goal. A framework for modeling and analysis of complex systems is well suited to the user when this framework allows the user to model his system as he has designed it and to analyze it with logic formulas and/or performance indices which are directly meaningful to him. This has led us to the following choices in the context of FMSs. First the modeling we propose is component-based as FMS are explicitly obtained by assembling different functional parts. In addition, the granularity of the components correspond to the architectural decomposition of the FMS meaning that there is a one to one mapping between the model components and the business components. Although our internal formalism is an SPN, the user can fully model his FMS without specifying any Petri net. Similarly, while the formula language supported by the framework is very expressive, a set of specific formula patterns corresponding to usual FMS analyses is proposed to the user. Since these patterns can be instantiated by fixing different parameters, this yields a simple and flexible way to check the model.

An overall view. The FMS modeling framework is based on the following principles:

1) It lies on the following basic component classes: *Load Unit* (representing the loading of raw materials into the system), *Machine* (representing the various phases of the actual manufacturing of workpieces) and *Transportation* (representing the movement of materials/workpieces). These component classes have different attributes reflecting their functional properties, that may be quite complex. For instance routing policy may be fixed or state-dependent and in the latter case may depend on the occupation of buffers or occurrences of failures.
2) First the modeler specializes the basic classes by fixing the values of their attributes. As in object-oriented approaches, it allows to reuse the specialized classes for different architectures of FMS sharing some identical component types.
3) Then the modeler instantiates these specialized classes in (named) components.
4) At last these components are combined using the names to bind variables occurring in the interface of the class. For instance, assume that the definition of a transporter class involves some machine variable, say X, producing the inputs of the transporter. Then, when the transporter is instantiated as a component, variable X is substituted by a machine name.

An internal view. When the set of specialized classes are specified through a (natural-language-like) syntax, the corresponding SPN subnets are automatically generated from such specifications. The SPN subnets can be seen as boxes partitionned in an *interface* and an *internal structure*. SPN components interface consists of *local* and *imported places* arranged on the edge of a box. Local places (denoted as non-filled-in circles) represent relevant aspects of a component's state (that may be imported by other components). Imported places (denoted as filled-in circles) represent relevant aspects of external components that influence the importing component behavior.

Naming of places and transitions is essential for the assembling of FMS. The name of local places can be viewed as local identifiers. Depending on their role in the component, the name can be predefined, as $idle$ in Figure 3, or composed by a predefined word followed by an identifier provided by the user, like in_a1 in the same figure. Here $a1$ corresponds to the name of a product and in means that this product is an input of the machine. Names of imported places are built by prefixing a local name by a variable like $X3.in\_a1$ in Figure 4. Observe that the set of variables occurring in a subnet corresponds to the components that will communicate with a component of this class. Since the variable are typed by their class, the compilation stage checks that the interfaces intended to be linked are compatible. Transition names are handled like local place names.

When the modeler instantiates a class into one or several (named) components, he must provide a component name per variable occurring in this class. At the net level, we need one class subnet copy per component of this class. The names of local places and transitions are prefixed by the name of the component while the name of imported places is obtained by

3 types of material ($a1, a2, a3$) are loaded in the FMS. Items of type ai ($1 \leq i \leq 3$) are loaded according to (delay) distribution ld_ai. An SLU1 class has a finite buffer of size s. When s items are present in the buffer loading is interrupted, and it is automatically restarted as soon as one item is withdrawn from the output buffer of SLU1.

TABLE I
EXAMPLE OF (INFORMAL) SPECIALIZATION OF THE LOAD UNIT CLASS

substituting the component names to the variables. Now the assembling of the whole net is straightforward: it consists in merging places with identical names.

In the following subsections, we illustrate the specialization of the three basic classes and the associated subnets.

B. Modeling the load unit

The Load Unit (LU) class represents the process through which raw materials are loaded (from the "external world") into an FMS. **LU's parameters** are: *i) set of loaded materials:* the type of raw-materials the whole production system depends upon; in the case of a closed-system, the number of items for each material type *ii) size of buffers:* the size of the output buffers of the LU component; *iii) loading times:* the distribution of the loading time for each type of material.

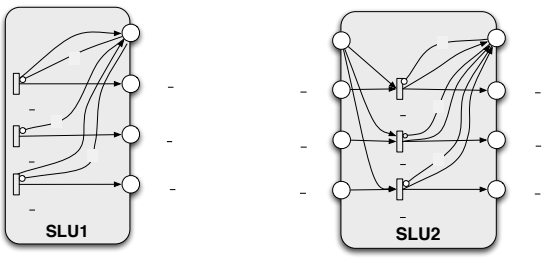


Fig. 2. Internal structure of multi-material, buffered specialized Load Unit class for *open system* (left) and *closed system* (right)

An example of LU specialization (for an *open system*) for a FMS part named SLU1 is given in Table I. The corresponding SPN component is depicted in Figure 2 (left), (Figure 2 (right) instead corresponds to a *closed system*). The **interface of the (*open system*) SLU1 class** consists of 4 (output) places corresponding to the items loaded into the FMS: place out containing the total of loaded items, place out_ai containing type ai loaded items. The **interface of the SLU1 class** consists of 3 timed-transitions $load\_ai$ ($1 \leq i \leq 3$), representing loading of each type of piece. Note that since transitions ld_ai have no input places the underlying model is inherently infinite-state if we allow $s = \infty$. In case of a *closed system* (Figure 2 right) the interface contains 4 additional (input) places. They represent the initial amount of material: total amount (place in) and type ai amount (place in_ai) [1]. The control on the fullness of the $s \geq 1$ sized buffer is achieved through the inhibitor arcs connecting each "loading" transition with the (output) place out. An SLU1 class with infinite buffer capacity is obtained by removing inhibitor arcs from the components in Figure 2.

[1]Note that in a *closed system* the input places of the LU component will be "connected" with the output places of the machine(s) which delivers the end product of the FMS.

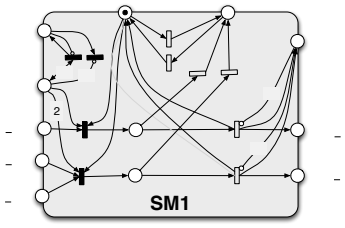


Fig. 3. A specialized class of machine

C. Modeling the machine

The machine class (M) describes the behavior of machines processing materials/workpieces. **M's parameters** are: *i) set of input materials:* the type of materials/workpieces processed by the machine; *ii) buffers dimension:* the dimension ($\in \mathbb{N} \cup \{\infty\}$) of the input and output buffers of M; *iii) processing times:* the distribution of the processing time for each type of material/workpiece processed by M. This is given by a 3-tuple: first parameter is the set of input materials, the second is the set of produced workpieces, the third one is the distribution for this production. *iv) failure/repair times:* the distribution of failure and repairing times.

We now illustrate a possible specialization SM1 (described in table II) of the machine class. The corresponding SPN component is depicted in Figure 3.

The ***interface of the SM1 class*** consists of 10 places (Figure 3): a 1-safe place $idle$ indicating the idle state of SM1 ; a 1-safe place $down$ indicating whether SM1 is down; a 1-safe place $full$ indicating that the input buffer of the machine is full; place in (out) representing the total number of items in the input (output) buffer of SM1 and places in_ai, $1 \leq i \leq 3$ (out_bj, $1 \leq j \leq 2$) indicating the number of items of type ai (bj) material (workpiece) in the input (output) buffer of SM1. In case of a machine producing a single type of workpiece from a single type of material, the interface of the corresponding subclass would simply consist of places $idle$, $full$, $down$, in and out.

Let us describe ***internal structure of SM1***. Transition $pra1$ and $pra2a3$ respectively represent processing of output pieces $b1$ and $b2$. Their associated distributions are obtained following the user specification. The control on the $s_o \geq 1$ sized output buffer is achieved through the inhibitor arcs connecting each "processing" transition with the (output) place out. Failures are modeled by three transitions: $fail_1$, $fail_2$ and $fail_3$ corresponding to a failure occuring respectively when the machine is idle, processing piece $b1$ or $b2$. Transition $repair$ models the repairing process.

Many other specializations of the machine class can be considered, for instance a machine could simultaneously process multiple inputs and outputs, or different materials could have different separate buffers.

D. Modeling the transportation unit

The transporter class (T) describes a transportation unit moving materials/workpieces from (a set of) *source nodes* to

Machine class SM1 processes 3 types of pieces: $a1, a2, a3$. *It can process a* $a1$ *piece resulting in a type* $b1$ *piece. Another process consists in combining* $a2$ *and* $a3$ *resulting in* $b2$. *SM1 class is prone to failure and repairing. After a repairing, the unfinished workpieces are lost.* *The input (resp. output) buffer sizes are denoted* s_i *and* s_o

TABLE II
EXAMPLE OF (INFORMAL) SPECIALIZATION OF THE MACHINE CLASS

(a set of) *target nodes* of an FMS. Source and target nodes of a T component can be either machines or other transporters[2].

At the level of generic class T, the parameters are untyped. The types will be defined during the specialization. We now describe them informally. **T's parameters** are: *i) level of freedom* of the unit, specifying whether the trajectory is fixed or subject to change depending on the needs. *ii) transportation policy* expressing when the transporter decides to move and where. *iii) delivery time* depending on the materials to be delivered and the location (initial or destination). Depending on the nature of the transportation unit, new parameters will appear in the specialized class. We now illustrate a possible specialization ST1 (described in table III) of the transporter class. The corresponding SPN component is depicted in Figure 4. Note that almost all places in the interface are represented in grey as they are *imported* from other SPNs.

The ***interface of the ST1 class*** is composed of: i) 3 input places from machine $X1$, one for the total number of pieces and one for each type ($a1$ and $a2$). ii) five output places representing the input buffers of the destination machines, and the number of pieces of each type in each machine iii) six controlling places stating whether the destination machines are full, idle and down. iv) Finally place $idle$ indicates whether the transporter is free.

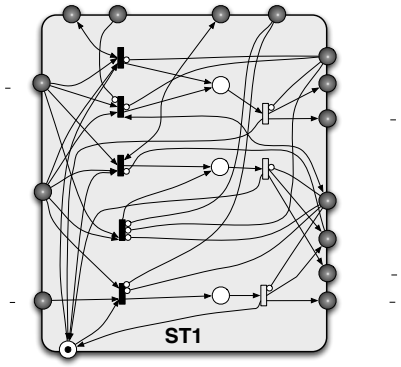


Fig. 4. An example of *transporter class* employing selective policies for moving of pieces from multi-typed machine X_1 to limited-size buffered target machines X_2, X_3

The ***internal structure of ST1*** describes the delivery policy. The three internal places correspond to (from top to bottom) the $a1$ pieces to be delivered to $X2$, the $a1$ pieces to be delivered to $X3$ and the $a2$ pieces to be delivered to $X3$. The black transitions named $snd1$ to $snd4$ correspond to the delivering possibilities for $a1$ pieces. They are controlled by

[2]machine-to-transporter movements are useful when modeling *continuous* transportation system such as, for example, conveyor belts transportation or rail-guided AGVs.

Transporter class $ST1$ *moves workpieces* $a1$ *and* $a2$ *from machine* $X1$ *to machines* $X2$ *and* $X3$ *(to be instanciated during the linking phase).* $a1$ *pieces can reach either* $X2$ *or* $X3$*;* $a2$ *pieces must reach* $X3$. *delivery time from* $X1$ *to* $X2$ *for* $a1$ *follows distribution* $D1$ *delivery time from* $X1$ *to* $X3$ *for* $a1$ *follows distribution* $D2$ *delivery time from* $X1$ *to* $X3$ *for* $a2$ *follows distribution* $D3$ *if* $X3$ *is idle and not full then deliver* $a1$ *to* $X3$, *else if* $X2$ *is idle and not full, then deliver* $a1$ *to* $X2$, *else if* $X3$ *is up and not full, then deliver* $a1$ *to* $X3$, *else if* $X2$ *is up and not full then deliver* $a1$ *to* $X2$.

TABLE III
EXAMPLE OF (INFORMAL) SPECIALIZATION OF THE TRANSPORTER CLASS

controlling places through inhibitor or regular arcs. Note that, in order to simplify the figure, the return time of the transporter is abstracted away.

IV. FINE-GRAINED TRANSIENT-ANALYSIS OF FMS

A. Hybrid Automata Stochastic Logic

In order to describe and verify interesting properties of FMS, we use an expressive logic called HASL, introduced in [BDD$^+$11]. This logic is based on two components: first an extension of timed automata, called linear hybrid automata, that will synchronize with the SPN in order to precisely select a set of timed paths, then an expression based on moments of path random variables is defined and evaluated on the system.

The first component of a HASL formula is a restriction of hybrid automata [ACHH92], namely synchronized Linear Hybrid Automata (LHA). LHA extend the Deterministic Timed Automata (DTA) used to describe properties of Markov chain models [DHS09], [CHKM09]. Simply speaking, LHA are automata whose set of *locations* is associated with a n-tuple X of real-valued variables (called data variables) that evolve with a linear rate depending on the location of the automaton and on the current state of the SPN (through special functions called *indicators*). Our model also uses *constraints*, which describe the conditions for an edge to be traversed, and *updates*, which describe the actions taken on the data variables on traversing an edge. Both constraints and updates are more general than their timed automata counterpart as they allow for linear combinations of data variable values (possibly multiplied by indicators). The automaton has two types of transitions: autonomous, *i.e.* time-triggered (or rather variable-triggered), labeled with $\sharp$, that take place as soon as a constraint is satisfied, and synchronized *i.e.* triggered by the SPN and take place when an event occurs in the SPN. Due to the determinism ensured on the hybrid automaton, the synchronisation between the SPN and the LHA leads to a stochastic process[3].

An HASL automaton expresses requirements associated to the specification process. The link between the SPN (corresponding implementation) and HASL is made through state indicators and transition labels (see examples below).

Example. The automata of figure 5 illustrate some of the possibilities of HASL. The first one has two variables: x_1 is in fact a clock, reset at the occurrence of the first failure (in figure 3, fail should label all three transitions whose name is prefixed by $fail$), and x_2 is a counter (rate 0) that counts the number of objects processed (transition labelled out) between

[3]Details can be found in [BDD$^+$11]

the two first failures. The second automaton has two variables counting the global time (x_1) and the time in state Mthre (x_2). The automaton changes state from Init to Mthre depending on whether the number of tokens in a specified buffer reaches a threshold or not. Here on the example of figure **??**, the condition $m(M2.in) > s$ with s equal to a chosen threshold is a good candidate for indicator **thre**. After k time units, the execution reaches state End and terminates. In the third automaton, x_2 counts the number of pieces arrived so far, and x_1 the number of pieces, arrived among the k first ones, that are still waiting. The execution terminates when all of the k first pieces are being/have been served (x_1=0). Since both variables have rate 0 in each state, the rates are omitted in the figure.

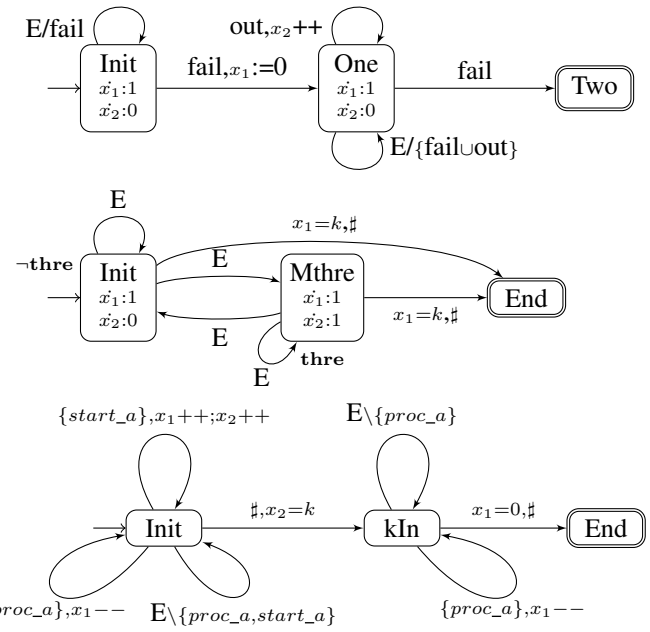


Fig. 5. Three LHAs to compute interesting measures on FMS

Expressions. The second component of a HASL formula is an expression related to the automaton. Such an expression, denoted Z, is based on moments of a path random variable Y and is defined as follows. First y is an arithmetic expression built on top of LHA data variables and constants. Then Y is a path dependent expression built on top of basic path random variables such as $last(y)$ (resp. $min(y)$, $max(y)$)i.e. the last (resp. minimum, maximum) value of y along a synchronizing path, $int(y)$ (i.e. the integral over time along a path) and $avg(y)$ (the average value of y along a path). Finally Z, the actual target of HASL verification, is an arithmetic expression built on top of the first moment of Y ($E[Y]$), and thus allowing for the consideration of diverse significant characteristics of Y including, for example, expectation, variance and covariance. Ensuring that, with probability 1, the system (SPN + LHA) will reach a final state, the expression Z associated with the formula may be evaluated with expectations defined w.r.t. the distribution of a random path *conditioned by acceptance of the path*. In other words, the LHA $\mathcal{A}$ both calculates the relevant measures during the execution and selects the relevant executions for computing the expectations. This evaluation gives the result of the formula $(\mathcal{A}, Z)$ for an SPN $\mathcal{S}$.

Example. Given the first LHA of figure 5, the expected throughput between the two first failures corresponds to expression $E(last(x_2)/last(x_1))$. If we slightly modify it by considering state One as a final state, we can compute the mean time to first failure by $E(last(x_1))$ and its variance (which is often a critical parameter) by $E(last(x_1)^2) - E(last(x_1))^2$. If we consider the second LHA, the expected value of the average time (within k time units) that the input buffer of machine M is full can be computed using $E(last(x_2))$ and the ratio of the time it is full is $E(last(x_2)/last(x_1))$. For automaton 3 we can express the expected value of the mean waiting time for k products using expression $E(int(x_1)/k)$.

This logic extends the transient properties that can be expressed and verified using other stochastic logics (such as CSL, CSRL, asCSL, CSL$^{\text{TA}}$,...) both capturing probabilistic properties of standard probabilistic model checking and also enabling to express more complex performance evaluation measures, coupled with a more precise selection of paths.

B. Expressing qualitative and quantitative properties of FMS

The steady-state analysis has been the focus of many performance studies for manufacturing systems. Traditionally we are interested in customer average measures like mean fabrication time for a kind of product, time average measures like mean number of raw materials in a buffer. The relevance of transient measures for manufacturing systems has been emphasized in [NV94]. In FMSs the arrivals of raw materials to feed the input buffers, and the extraction of finished products from output buffers may be bursty. This means there may be high-activity and low-activity periods due to some external reasons like logistic problems. For such cases it is really important to observe transient behaviors which may be radically different from equilibrium (steady-state) behaviors. For instance, the buffers must be dimensioned by considering high-activity periods, and the throughput (mean number of finished products) during low-activity may be important. We can state here the case when the setting of FMS is changed, the time until the system reaches a stationary regime may be long and it may be important to observe this transient period. In FMS, the components are prone to failures or human interventions that may provoke the unavailability of some parts of the system. Such phenomena may lead to a deadlock situation or to a complete unavailability of the system. For such cases only transient measures provide some lights on FMS properties.

Using the HASL formalism, we can express interesting quantitative measures on FMS. We give here several examples. First we can characterize (and evaluate) properties related to the occupation of finite-capacity buffers, like the blocking probability for a machine, the mean time to fill $x\%$ of buffers, the mean number of pieces in buffers during a given time interval. These measures are important for an appropriate dimensioning of buffers. In order to evaluate the efficiency

of the underlying FMS we are also interested in the measures related to throughput (mean number of produced workpieces per time unit), and make-span (average production time for a given production workflow). We can state for instance the probability that a certain number of workpieces are produced during a given time interval, the average time to produce a given number of workpieces. The reliability measures when some components are prone to failures can be also considered like the Mean Time To Failure (MTTF) of a component or whole system, throughput of a given production workflow between the first and the second failure.

In addition, steady state measures can also be obtained by transient analysis when the system admits regeneration points. Indeed, the steady state measure is then the average measure between two regeneration points.

C. Automatic Generation of properties for FMS

Just as we did not want to assume that a modeller knows the Petri net formalism, this modeller should be able to verify different properties without any knowledge about hybrid automata. The goal of the automatic generation of properties for FMS is to hide this formalism and to let the user chose a property to verify in an intuitive way. The user selects a property pattern in a predefined list (for example the mean time to fill $x\%$ of a buffer), selects the appropriate parameters (the desired buffer and the percentage) and the HASL formula (automaton + expression) is generated automatically. This generation is possible and efficient since, as we consider a predefined list of relevant properties, there is no combinatory explosion and the translation is relatively simple.

An important point to mention is that the generation of SPNs for the FMS model and of the automaton for the HASL formula are linked, since the SPN needs to be coherent in terms of labels on events and indicators.

V. Conclusion

We have presented here a compositional modeling framework for flexible manufacturing systems using stochastic Petri nets. The FMS is modeled piecewise by specifying the classes of components to be used (loading unit, transporters, machines), specifying their parameters (type of raw material needed/produced, size of input/output buffers, transporting policy, ...) and then combining all these components.

In order to evaluate these FMS, we then use a stochastic logic named HASL. This logic enables a precise selection of succesful path by synchronizing the SPN with an hybrid automaton, and then a quantitative evaluation using an expression that can express both model checking (the probability of the set of winning paths, ...) and performance evaluation (mean waiting time, ...) measures.

These two steps are meant to be facilitated for the modeler. The goal is to generate both the SPN for the FMS and the automaton + expression for the formula in an automated way, not requiring the modeler to be familiar to either type of models. We aim at an analysis that is both formal, using the COSMOS tool for evaluating HASL formulas on FMS, and user oriented, providing the user with an easy way to describe his model and specify the useful formulas.

As numerous biological systems analysis rely on complex stochastic processes, we plan to undertake a similar work in this area.

References

[ACHH92] R. Alur, C. Courcoubetis, T. A. Henzinger, and P.-H. Ho. Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems. In *Hybrid Systems*, volume 736 of *LNCS*, pages 209–229. Springer, 1992.

[BDD+11] P. Ballarini, H. Djafri, M. Duflot, S. Haddad, and N. Pekergin. HASL : an expressive language for statistical verification of stochastic models. In *Proc. 5th Int. ICST Conf. on Performance Evaluation Methodologies and Tools*, 2011. To appear.

[BDK01] E. Best, R. Devillers, and M. Koutny. *Petri net algebra*. Monographs in Theoretical Computer Science. Springer, 2001.

[BGS01] F. Balduzzi, A. Giua, and C. Seatzu. Modelling and Simulation of Manufacturing Systems with First-Order Hybrid Petri Nets. *Inter. J. of Production Research*, 39(2):255–282, 2001.

[CHKM09] T. Chen, T. Han, J.-P. Katoen, and A. Mereacre. Quantitative model checking of continuous-time Markov chains against timed automata specifications. In *Proc. Symp. on Logic in Computer Science (LICS)*, pages 309–318. IEEE, 2009.

[DHS09] S. Donatelli, S. Haddad, and J. Sproston. Model checking timed and stochastic properties with CSL^{TA}. *IEEE Trans. on Software Engineering*, 35:224–240, 2009.

[ECM95] J. Ezpeleta, J.M. Colom, and J. Martinez. A Petri net based deadlock prevention policy for flexible manufacturing systems. *IEEE Trans. on robotics and automation*, 4(2):173–184, 1995.

[LCGH93] C. Lindemann, G. Ciardo, R. German, and G. Hommel. Performability modeling of an automated manufacturing system with deterministic and stochastic Petri nets. In *ICRA (3)*, pages 576–581, 1993.

[MBC+95] M. Ajmone Marsan, G. Balbo, G. Conte, S. Donatelli, and G. Franceschinis. *Modelling with Generalized Stochastic Petri Nets*. J. Wiley, 1995.

[NV94] Y. Narahari and N. Viswanadham. Transient analysis of manufacturing systems performance. *IEEE Trans. on Robotics and Automation*, 10(2):230 –244, 1994.

[WD98] J. Wang and Y. Deng. Incremental modeling and verification of flexible manufacturing systems. *J. of Intelligent Manufacturing*, 4, 1998.