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 IV and the adequation of HASL formulae for analysis of FMS in section V. Finally we conclude and give perspectives to this work in section VI.

**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 \( S \) 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 \( n > 0 \), \( S_n \in S \) is the state after the occurrence of the \( n \)th 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 to be 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 \( S \) 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 \( S_n = (m, \{dur_t\}_{t \in T}) \) 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. \( priom(t, m) \)). Whenever \( T' \) is not a single element, sample a random choice between transitions of \( T' \) with probabilities proportional to \( wght(t, m) \).
- Let \( t^* \in T' \) be the selected transition. Then \( S_{n+1} = (m', \{dur_t\}_{t \in T}) \) is defined by:
  - \( m' \) is the marking reached by the firing of \( t^* \) in \( m \).
  - \( t \) was 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 \( \tau \) and can be defined w.r.t. the random marking at time \( \tau \) 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. \( \tau \) 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 V-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, heterogeneous 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 $M_6$).

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.

**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:

![Different types of workflow for FMSs](image)
TABLE I
EXAMPLE OF (INFORMAL) SPECIALIZATION OF THE LOAD UNIT CLASS

substituting the component specialization 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 ≤ i ≤ 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 = ∞. 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 ≥ 1 sized output 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.

1Note 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.

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 \ {∞}) 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 ≤ i ≤ 3 (out Bj, 1 ≤ j ≤ 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_i ≥ 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: fail1, fail2 and fail3 corresponding to a failure occurring 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
The input (resp. output) buffer sizes are denoted $s_1$ and $s_o$.

<table>
<thead>
<tr>
<th>TABLE II</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXAMPLE OF (INFORMAL) SPECIALIZATION OF THE MACHINE CLASS</td>
</tr>
</tbody>
</table>

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

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 ($a_1$ and $a_2$). 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.

The internal structure of ST1 describes the delivery policy. The three internal places correspond to (from top to bottom) the $a_1$ pieces to be delivered to $X_2$, the $a_1$ pieces to be delivered to $X_3$ and the $a_2$ pieces to be delivered to $X_3$. The black transitions named $snd_1$ to $snd_4$ correspond to the delivering possibilities for $a_1$ pieces. They are controlled by controlling places through inhibitor or regular arcs. Note that, in order to simplify the figure, the return time of the transporter is abstracted away.

<table>
<thead>
<tr>
<th>TABLE III</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXAMPLE OF (INFORMAL) SPECIALIZATION OF THE TRANSPORTER CLASS</td>
</tr>
</tbody>
</table>

![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 $X_2$, $X_3$]

![Fig. 5. Workflow of the modeled conveyor-belt FMS with 3 machines and 2 raw-materials](image)

To demonstrate the application of the compositional SPN framework we consider an example of FMS (push production) system taken from [AMBC*95] (chapter 8). Such FMS consists of: a loading unit $LU_1$ and three machines $M_1, M_2$ and $M_3$ arranged in a linear fashion according to the workflow depicted in Figure 5. The FMS treats two types of material, namely $a$ and $b$, which are progressively transformed into workpieces and eventually in the final products (i.e. the output of $M_3$). The transportation medium is a conveyor belt, which we assume to consist of 4 adjacent belt segments (segment $LU_1-M_1$, segment $M_1-M_2$, segment $M_2-M_3$ and segment $M_3-LU_1$, each one commanded by a separate engine) The first machine in the line, i.e. $M_1$, processes only material of type $a$ (thus type $b$ items arriving at $M_1$ are bypassed to $M_2$); machine $M_2$ processes both type $b$ raw material and pieces $a'$ outputted by $M_1$; finally $M_3$ processes only pieces $b'$ outputted by $M_2$. The SPN components corresponding to such FMS are depicted in Figure 6. They consist of an LU-component (i.e. $LU_1$), three M-component (i.e. $M_1, M_2, M_3$) and four T-components (i.e. $T_1, T_2, T_3, T_4$).

In this example we assume the 4 segments of the conveyor belt to behave in a blocking fashion: if on reaching of a certain position an item cannot be unloaded (because the destination machine’s buffer is full or because the next segment is not empty) then the belt (segment) blocks. Note that from a behavioral point of view the conveyor’s segments can be distinguished into: those performing delivering of an item (to a machine) and bypassing (to the next segment) i.e. component $T_1$ and $T_3$, and those performing delivering without bypassing, i.e. component $T_2$ and $T_4$. To better understand such a
V. 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 $\tau$, 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.

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 7 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 the two first failures. The second automaton has two variables counting the global time ($x_1$) and the time in state $\text{Mthre}$ ($x_2$). The automaton changes state from $\text{Init}$ to $\text{Mthre}$ depending on whether the number of tokens in a specified buffer reaches a threshold or not. Here on the example of figure 6, the condition $m(M2.in) > s$ with $s$ equal to a chosen threshold is a good candidate for indicator $\text{thre}$. 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.

![Diagram](image_url)

**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 \( A \) 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 \((A, Z)\) for an SPN \( S \).

**Example.** Given the first LHA of figure 7, 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, CSLA,...) 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 combinatorial 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.

VI. 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 successful 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


