Exploiting Formal Methods to Simplify the Modeling of Flexible Manufacturing Systems

Joceleide D. C. Mumbelli Marcelo Rosa Patrik B. Schettert Marcelo Teixeira

***Resumo*—*Flexible Manufacturing Systems* (FMS) tend to be large, complex and to depend on an increasingly numerous and intricate set of requirements. To be properly controlled, a FMS requires software engineers to combine thousands of programming rules, empirically addressing parallelism, concurrency, etc. Yet, in the end there is no quality guarantees about the imagined solution, as human-centered development practices depend on the designer's inspiration to be conducted. In contrast, formal approaches for FMS control (such as the *Supervisory Control Theory*), allow to automatically calculate optimal operational sequences to industrial processes, but they require a modeling structure as input. This itself can be infeasible to be obtained for large and complex processes. In this paper, we present an example of a FMS for which an optimal control logic would depend on an intricate structure of models that could make a solution infeasible to be derived by hands. Then, we describe and apply a formal alternative that can substantially reduce modeling effort, while leading to an equivalent resulting model, which can then be used as input to control synthesis algorithms.**

***Index Terms*—Flexible Manufacturing Systems, Formal Modeling, Complex Programming.**

I. Introduction

A *Manufacturing System* (MS) refers to an industrial production process through which materials are transformed into products by integrating people, equipments and technology [1]. When a MS is context sensitive, i.e., it reacts to changes occurred in the factory and starts to behave differently, in different situations, then the systems is said to be *flexible* and the whole structure is called *Flexible Manufacturing System* (FMS) [2]. Nowadays, FMSs represent an opportunity for shifting from fixed to customized production and, when associated to computational technology, e.g., web and intelligence, they lead to modern advanced methods for industry.

In spite of the practical relevance, the current standards of industrial processes suggest that FMSs tend to be large, complex and to depend on an increasingly numerous and intricate set of requirements. This makes it difficult for a FMS to be approached using traditional programming. To be properly coordinated, a FMS requires software engineers to combine thousands of programming rules, empirically addressing parallelism, concurrency, etc. Yet, in the end there is no quality guarantees about the imagined solution, as human-centered development practices depend on the designer's inspiration to be conducted. In contrast, formal approaches for FMSs coordination (such as the *Supervisory Control Theory* [3]), allow to automatically calculate their optimal operational sequences. However, those methods require a modeling structure as input, which itself can be infeasible to handle for large and complex processes.

The authors are with the Academic Department of Informatics (DAINF) of the Federal University of Technology - Paraná (UTFPR), Pato Branco, Brasil ({joceleide, marcelorosa, patrik}@alunos.utfpr.edu.br, marceloteixeira@utfpr.edu.br).

In this paper, we present an example of a FMS for which an optimal coordination logic would depend on an intricate structure of models. We start by showing how the system could be modeled by using the conventional theory of *Languages* and *Automata* [4], [5]. Using this initial structure of models, it will be illustrated how complex can become the task of coordinating the system if we consider the reasonable possibility of manufacturing more than one type of product in parallel. The case will be incrementally illustrated until the modeling task can be considered unfeasible.

Then, we present a formal alternative based on *Extended Automata*, that can substantially reduce modeling effort producing more concise and readable models, while keeping equivalent resulting behavior, which can then be used as input to control synthesis algorithms. A step-guide to the use of Extended Automata to model industrial processes is also provided and illustrated in the context of the same example of FMS, which allows the approaches to be compared.

The paper is structures as follows: Section II introduces some preliminaries on FMS modeling and coordination; Section III presents an example that motivates the alternative modeling method described in Section IV. Section V proposes a methodology for the modeling of complex FMSs, which is followed by an example. Conclusions and perspectives as discussed in Section VI.

II. Preliminaries

FMSs are maybe the most representative example of a class of systems called *Discrete Event Systems* (DESs) [4]. DESs have in common the fact that their transitions are not guided by the time, but by *events* that occur irregularly, progressing the systems in many different and unpredictable manners. It is conceivable that modeling DESs is also different from modeling time-dependent systems. While the latter can be addressed, for example, by using differential equations, SEDs are more naturally modeled by state-transition diagrams, as described next.

A. FMS Conventional Modeling

Languages are formalisms that can be used to describe discrete-event behaviors [4]. Their basic structure are *events*, which are taken from a finite *alphabet* Σ, where Σ^* denotes the set of all finite *strings* generated by events from Σ, including

the *empty string* ε. A subset $L \subseteq \Sigma^*$ is called a *language*. The *prefix-closure* of a language L is $\overline{L} = \{\, s \in \Sigma^* \mid st \in L \text{ for some } t \in \Sigma^* \,\}$.

A language L it is said to be *regular* if, and only if, it can be recognized by finite-state automata [6]. In industry, regularity is of interest, as it delimits a class of languages that are suitable for computational processing, i.e., they can be represented by automata that occupy finite memory when stored in a computer [4].

A *Finite State Automata* (FA) define a simple and powerful framework to formally recognize languages that describe discrete behaviors and their properties. A FA can be formally represented as a 5-tuple $\mathrm{G} = \langle \Sigma, Q, q^\circ, Q^\omega, \rightarrow \rangle$, where: Σ is the alphabet of events; Q is the set of states; $q^\circ \in Q$ is the initial state; $Q^\omega \subseteq Q$ is the subset of marked states (complete tasks); $\rightarrow \subseteq Q \times \Sigma \times Q$ is the state transition relation.

For two any states $q_1, q_2 \in Q$, we denote by $\mathsf{q}_1 \xrightarrow{\sigma} \mathsf{q}_2$, a transition from the state q_1 to q_2 with the event $\sigma \in \Sigma$. $\mathrm{G} \xrightarrow{s} q$ denotes that a string s is possible in the FA G. Two languages can be defined from G: $L(\mathrm{G}) = \{s \in \Sigma^* \mid \mathrm{G} \xrightarrow{s} q \in Q\}$; $L^\omega(\mathrm{G}) = \{s \in \Sigma^* \mid \mathrm{G} \xrightarrow{s} q \in Q^\omega\}$. $L(\mathrm{G})$ is the *generated language*, containing all strings possible in G, while $L^\omega(\mathrm{G})$ is the *marked language*, i.e., the set of strings leading to marked states.

Two FA, $\mathrm{A} = \langle \Sigma_\mathrm{A}, Q_\mathrm{A}, q^\circ_\mathrm{A}, Q^\omega_\mathrm{A}, \rightarrow_\mathrm{A} \rangle$ and $\mathrm{B} = \langle \Sigma_\mathrm{B}, Q_\mathrm{B}, q^\circ_\mathrm{B}, Q^\omega_\mathrm{B}, \rightarrow_\mathrm{B} \rangle$, can be synchronously composed by:

$$\mathrm{A} \parallel \mathrm{B} = \langle \Sigma_\mathrm{A} \cup \Sigma_\mathrm{B}, Q_\mathrm{A} \times Q_\mathrm{B}, (q^\circ_\mathrm{A}, q^\circ_\mathrm{B}), Q^\omega_\mathrm{A} \times Q^\omega_\mathrm{B}, \rightarrow \rangle,$$

in which:

- $(q_A, q_B) \xrightarrow{\sigma} (q'_A, q'_B)$, if $\sigma \in \Sigma_\mathrm{A} \cap \Sigma_\mathrm{B}$;
- $(q_A, q_B) \xrightarrow{\sigma} (q'_A, q_B)$, if $\sigma \in \Sigma_\mathrm{A} \setminus \Sigma_\mathrm{B}$;
- $(q_A, q_B) \xrightarrow{\sigma} (q_A, q'_B)$, if $\sigma \in \Sigma_\mathrm{B} \setminus \Sigma_\mathrm{A}$.

In $\parallel$, the events shared by the two FA are synchronized, while other events are interleaved. Fig. 1 illustrates the use of the operator $\parallel$ to compose two FA A and B.

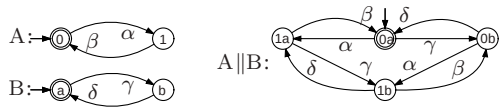

Fig. 1. Example of synchronous composition of FA.

For a set of FA $\mathrm{G}^i = \langle \Sigma_i, Q_i, q^\circ_i, Q^\omega_i, \rightarrow_i \rangle$, for $i = 1, \cdots, n$, a global model $\mathrm{G} = \langle \Sigma_\mathrm{G}, Q_\mathrm{G}, q^\circ_\mathrm{G}, Q^\omega_\mathrm{G}, \rightarrow_\mathrm{G} \rangle$ can be obtained by the composition $\mathrm{G} = \|^n_{i=1} \mathrm{G}^i$, such that $\Sigma_\mathrm{G} = \bigcup^n_{i=1} \Sigma_i$. Compositions are particularly useful to allow systems to be modularly expressed.

B. FMS Coordination

In the context of FMS automation, an important step is to properly (optimally) program the operational sequences (coordination logic) for the system to be commanded under control. This is a discrete event-driven activity that can be addressed by empirical programming, or it can be alternatively supported by formal methods and automated software engineering.

From the synchronous composition ($\|$) of automata (FA or EFA), it becomes possible to modularly design an entire system model (also known as plant) G by composing m subsystem models $\mathrm{G} = \|^m_{i=1} \mathrm{G}^i$. Similarly, control rules can be imposed by specifications models E^j which are individually expressed and composed afterwards to form $\mathrm{E} = \|^n_{j=1} \mathrm{E}^j$. This step-by-step procedure tends to facilitate modeling tasks. When combined, G and E lead to a structure $\mathrm{K} = \mathrm{G} \,\|\, \mathrm{E}$, such that $L^\omega(\mathrm{K}) \subseteq L^\omega(\mathrm{G})$, that reflects the desired behavior under control. This structure can then be translated to implementable notations using automated tools for code generation [7].

When a FMS involves partially-controllable behaviors, it has to be controlled in a way to recognize the impossibility of disabling some its events. The *Supervisory Control Theory* (SCT) [3] is a formal alternative that allows to extract, from a FSM model K, its least restrictive, nonblocking and controllable sub-model K', which can then be used for implementation purposes. In this case, $L(\mathrm{K}')$ is associated to the specific sub-language that more closely approximates from $L(\mathrm{K})$, preserving controllability with respect to $L(\mathrm{G})$ without violating any requirement in E.

In spite of the advanced alternatives for calculating control solutions for FMSs, such as SCT, its several extensions and related tooling support, remark that computing K' depends on having models G and E. This itself is a human-centered task that depends on the designer. In this paper, we argue that some FMSs may involve a complex combination of events, such that G and/or E can be too much difficult to be manually structured (see example next). For these cases, we propose in Section IV an alternative that can substantially reduce modeling effort while preserving the same behavior of K.

III. An Example

The main problem approached in this paper is dimensioned in this section. For that, we use an example of a real industrial process, in the context of the experimental manufacturing cell XC241, produced by *Exsto Tecnologia* [8], which is illustrated in Fig. 2.

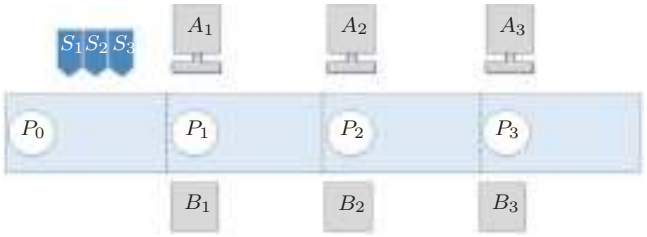

Fig. 2. Example of a manufacturing cell

The process is composed by sensors positioned such that they identify the size of workpieces arriving at a manufacturing line, by a conveyor that carries workpieces throughout the process , and by actuators that remove them from the conveyor to output buffers. Arriving workpieces are assumed to have 3 different sizes: *small* (sensor S_1), *regular* (sensor S_2) and *large* (sensor S_3), which also defines their separation by the

TABLE I
NOTATION FOR THE EVENTS COMPOSING THE PLANT MODEL

Component	*Event*	*Description*
Sensors	$s_i, i = 1, 2, 3$	sensor identifying a workpiece of the size 1 (small),2 (regular) or 3 (large)
Manipulators	s_{Ai}, $i = 1, 2, 3$	Event activating the actuator A_1, A_2 or A_3
	f_{Ai}, $i = 1, 2, 3$	Event identifying that the actuator A_1, A_2 or A_3 is retreated
Conveyor	s_C	Conveyor activation
	f_C	Conveyor deactivation

respective actuator A_1, A_2 or A_3 to the respective buffer B_1, B_2 or B_3. The conveyor behavior is sectored into 4 positions in a way to facilitate the manufacturing process. The initial position (P_0) is neutral and it is used to receive external workpieces. The other positions (P_1, P_2 and P_3) coincide with the respective actuator action point. Table I summarizes the process components and their respective events that will be used to design the plant model.

In Figure 3 the components S_1, S_2, S_3, A_1, A_2, A_3 and Conveyor are respectively modeled by the automata G^i, $i = 1, \cdots 7$, such that the composite system plant is given by $G = \|_{i=1}^{7} G^i$.

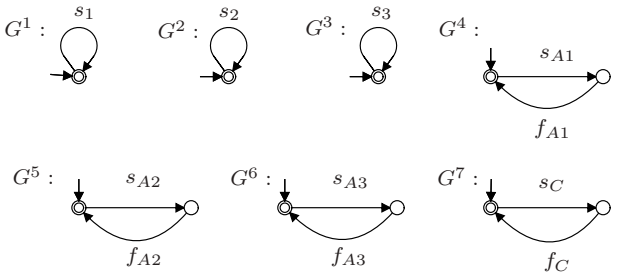

Fig. 3. Modular version of the system plant $G = \|_{i=1}^{7} G^i$

The three sensors are modeled by self-loops in the automata G^1, G^2 and G^3. Note that the activation of a sensor usually leads it back to the previous (initial) state, i.e., the sensor signal happens, is has to be recognized in terms of string composition, but there is no state-change. The other components are modeled by simple two-state automata representing their beginning and conclusion of operation.

In order to restrict the system plant to a given behavior expected under control, the following objective is assumed in the context of a specification E: "*Each workpiece must be separated according to its size*".

Designing E could be a trivial task if it was assumed that a single type of workpiece arrives in the system, or that a single workpiece is processed at a time. Nevertheless, this does not always match the needs of industry for productivity. Usually, manufacturing systems are required to admit concurrency and parallelism for multiple types of items, which substantially complexifies modeling tasks. Next, we incrementally illustrate how difficult the task of modeling E can become. We start by a case (E^1) that admits 1 single type of workpiece and then we show cases for more than one type. An automaton modeling E^1 is presented in Figure 4.

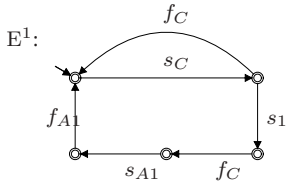

Fig. 4. Specification model for the separation of a single type of workpiece

Specification E^1 prevents the actuator A_1 to be activated (event s_{A1}) before a small workpiece is identified on the conveyor (event s_1) and the conveyor is positioned correctly (event f_C). We remark that the conveyor position is not controlled by modeling. This is coordinated at implementation time using encoder settings. We just take advantage of its events start/stop.

Note that this control logic involves a few states and it can be trivially expressed by the engineer. However, if we slightly reconfigure the control objective E, it can be shown that the modeling becomes substantially more complex. In order to analyze this impact, assume that two different types of workpieces can now be admitted in the system, small and regular. Small workpieces are still stored in the buffer B_1 while regular workpieces are separated in B_2. Now, the model E^1 is extended to absorb this new control requirement and the result is the specification E^2 shown in Figure 5.

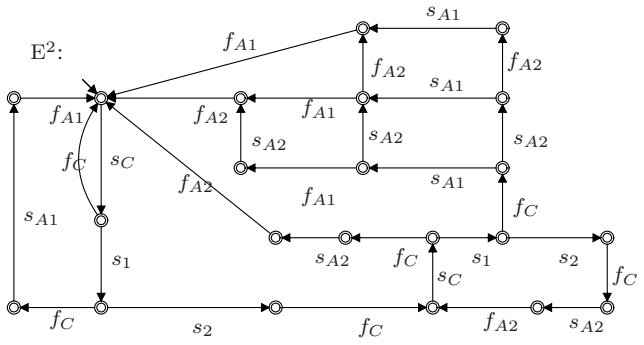

Fig. 5. Specification model for the separation of two types of workpieces

Specification E^2 prevents the actuator A_1 to be activated (event s_{A1}) before a small workpiece is identified on the

conveyor (event s_1) and the conveyor is positioned correctly in front the actuator A_1. In parallel, it also prevents the actuator A_2 to be activated (event s_{A2}) before a regular workpiece is identified on the conveyor (event s_2) and the conveyor is positioned correctly in front the actuator A_2.

For example, after a string $s_C s_1 s_2 f_C s_C s_1 f_C$ all events (belonging to the model) are disabled, except s_{A1} and s_{A2}. This means that, after the system identifies a regular and small workpiece, and both are positioned in front the correct buffer, the only action possible is the actuators activation.

It is conceivable that the modeling of E^2 is substantially more complex that the previous specification E^1. In fact, E^2 deals with the possibility of workpieces arrive in different sizes, in different orders. It also admits that after a workpiece arrives, another may or may not arrive and thus the conveyor has to behave differently. It turns out that mapping all possible combinations directly in the specification model becomes a nontrivial engineering task.

The number of states could be used to compare the models E^1 and E^2 and to dimension modeling effort. While the first involves only 5 states, the second requires to structure 22 states. Remark that this modeling task is not systematic as for example it is to express a buffer behavior. On the contrary, this construction requires an intricate reasoning about the concurrent nature of the events.

Assuming that a third type of workpiece was to be admitted in the system, a possible specification model E^3 would require 72 states to equivalently replace the 22-automaton E^2, and so on. Such complexity would become even more remarkable if the plant was to be modified in order to embody new components. The presented example, for instance, has support to also separate workpieces according to the material they are made, which has been prevented in this paper to keep the example illustrative. We remark that this non-systematic work depends exclusively of the engineer in order to map all possible contexts of the system, which can be a barrier for a control solution to be derived using FA as modeling foundation.

From the composition $\mathrm{K} = G \parallel \mathrm{E}^2$ we obtain an automaton with 44 states when two different types of workpiece are placed on the conveyor and separated by the control system. This model represents the desired behavior of the system when it is under control and it is used as the input for synthesis algorithms in order to ensure minimally restrictive and non-blocking behavior.

Next we present an alternative to the use of AF in modeling. It is shown that engineering effort can be substantially reduced while the same control objectives can be expressed. A methodology to the use of this approach on FMS control problems is also introduced.

IV. FMS Extended Modeling

Sometimes, modeling manufacturing processes using ordinary automata may become a too much complex task. By nature, FA are limited in expressiveness, particularly when dealing with data dependency, which requires to memorize long sequences of events until a given modeling decision can be taken. In this sense, the literature has extended FA in several ways to simplify the automatic control of FMSs. Timed [9], [10], modular [11], [12] and refined [13], [14] extensions, for example, have shown to be useful to simplify control synthesis. In this paper, we are more concentrated on the previous step, that is, on the modeling that enables for automatic control.

This section subsumes results in [15], [16], [17], [14] about *Extended Finite-State Automata* (EFA), a particular version of FA extended with formulas associated to transitions. A *formula* is a logical structure implemented using variables and constants. A *variable* v is an entity associated with a finite domain $\mathrm{dom}(v)$ and an initial value $v^\circ \in \mathrm{dom}(v)$.

The set of all variables $v_i,\ i = 0, \cdots, n$ is denoted by $V = \{v_0, \ldots, v_n\}$ and the domain of V is denoted by $\mathrm{dom}(V) = \mathrm{dom}(v_0) \times \cdots \times \mathrm{dom}(v_n)$, i.e., the domain of V is a set that combines all domains of individual variables. An element from dom(V) is written as $\bar{v} = (\bar{v}_0, \ldots, \bar{v}_n) \in \mathrm{dom}(V)$ with $\bar{v}_i \in \mathrm{dom}(v_i)$ and it is called a *valuation*, i.e., a valuation is a structure that assigns to each variable $v \in V$ a value belonging to its domain.

For example, let $V = \{a, b, c\}$ be a set of variables with $\mathrm{dom}(a) = \mathrm{dom}(b) = \mathrm{dom}(c) = \{0, \cdots, 10\}$. A possible valuation (in the deterministic case) for dom(V) could be, for example, $(0, 0, 0)$, which is also the initial valuation. Another could be $(2, 7, 1)$, etc. For Boolean variables, a valuation assigns only *true* or *false* values.

A second set of variables, called *next-state variables* and denoted by $V' = \{\, v' \mid v \in V \,\}$ with $\mathrm{dom}(V') = \mathrm{dom}(V)$, is used to describe how variables are updated by transitions.

For example, let x be a variable with domain $\mathrm{dom}(x) = \{0, \ldots, 5\}$ and initial value $x^\circ = 0$. A transition with update $x' = x + 1$ changes the variable x by adding 1 to its current value, if it currently is less than 5. Otherwise (if $x = 5$) the transition is disabled and no updates are performed. Another possibility is to write the formula $x' = \min(x+1, 5)$, in which case the transition remains enabled when $x = 5$. The update $x = 3$ disables a transition unless $x = 3$ in the current state, and allows all possible next-state values of x. Differently, the update $x' = 3$ always enables its transition, and the value of x in the next state is forced to be 3.

Formally, an EFA is described by a 6-tuple $M_{\mathrm{v}} = \langle \Sigma, V, Q, Q^\circ, Q^\omega, \rightarrow \rangle$, where:

- Σ is the alphabet of events;
- $V = \{v_1, \ldots, v_n\}$ is the set of variables;
- Q is the finite set of states;
- $Q^\circ \subseteq Q$ is the set of initial states;
- $Q^\omega \subseteq Q$ is the set of marked states;
- $\rightarrow \subseteq Q \times \Sigma \times \mathcal{F} \times Q$ is the state transition relation, where $\mathcal{F}$ is the set of Boolean formulas over $V \cup V'$.

The term $x \overset{\sigma:p}{\rightarrow} y$ denotes the presence of a transition in M_{v}, from state x to state y with event $\sigma \in \Sigma$ and update $p \in \mathcal{F}$.

An EFA M_{v} can also be interpreted (*unfolded* interpretation) as an ordinary FA $\mathrm{G} = \langle \Sigma, Q_{\mathrm{G}}, Q^\circ_{\mathrm{G}}, Q^\omega_{\mathrm{G}}, \rightarrow \rangle$ where:

- $Q_{\mathrm{G}} = Q \times \mathrm{dom}(V)$;

- $Q^{\circ}_{\mathrm{G}} = Q^{\circ} \times \{(v^{\circ}_1, \ldots, v^{\circ}_n)\}$;
- $Q^{\omega}_{\mathrm{G}} = Q^{\omega} \times \mathrm{dom}(V)$;
- $\rightarrow$ is such that $(x, \bar{v}) \xrightarrow{\sigma} (y, \bar{v}')$ for $\bar{v}, \bar{v}' \in \mathrm{dom}(V)$, if there exists $x \xrightarrow{\sigma:p} y$ such that $p(\bar{v}, \bar{v}') = true$.

The unfolded state set Q_{G} includes the values of the variables as part of each state. The unfolded transition relation is defined based on the transition relation of M_{v}, by taking into account the conditions imposed by the updates on the variable values. The unfolded transition relation is extended to strings in Σ^* by $(x, \bar{v}) \xrightarrow{\varepsilon} (x, \bar{v})$ for all $(x, \bar{v}) \in Q_G$ and $(x, \bar{v}) \xrightarrow{s\sigma} (x'', \bar{v}'')$ if $(x, \bar{v}) \xrightarrow{s} (x', \bar{v}') \xrightarrow{\sigma} (x'', \bar{v}'')$ for some $(x', \bar{v}') \in Q_{\mathrm{G}}$.

We denote by $M_{\mathrm{v}} \xrightarrow{s} (x, \bar{v})$ that the state $(x, \bar{v})$ can be reached from the initial state of M_{v}. Finally, the *open-loop behavior* and the *marked behavior* of M_{v} are the languages

$$L(M_{\mathrm{v}}) = \{\, s \in \Sigma^* \mid M_{\mathrm{v}} \xrightarrow{s} (x, \bar{v}) \in Q_{\mathrm{G}} \,\} \;;$$
$$L^{\omega}(M_{\mathrm{v}}) = \{\, s \in \Sigma^* \mid M_{\mathrm{v}} \xrightarrow{s} (x, \bar{v}) \in Q^{\omega}_{\mathrm{G}} \,\} \;.$$

Composition of EFA is well defined from the standard synchronous composition of FA [14]. The only difference is that now formulas are combined by conjunction. Thus, the same notation $\|$ is used to refer to both cases, as it is clear to the context. Given two EFA $\mathrm{A_v} = \langle \Sigma_A, V_A, Q_A, Q^{\circ}_A, Q^{\omega}_A, \rightarrow_A \rangle$ and $\mathrm{B_v} = \langle \Sigma_B, V_B, Q_B, Q^{\circ}_B, Q^{\omega}_B, \rightarrow_B \rangle$, the *synchronous composition* of A_{v} and B_{v} is $\mathrm{A_v} \parallel \mathrm{B_v} = \langle \Sigma_A \cup \Sigma_B, V_A \cup V_B, Q_A \times Q_B, Q^{\circ}_A \times Q^{\circ}_B, \rightarrow \rangle$, where:

- $(x_A, x_B) \xrightarrow{\sigma:p_A \wedge p_B} (y_A, y_B)$ if: $\sigma \in \Sigma_A \cap \Sigma_B$, $x_A \xrightarrow{\sigma:p_A}_A y_A$, and $x_B \xrightarrow{\sigma:p_B}_B y_B$;
- $(x_A, x_B) \xrightarrow{\sigma:p_A} (y_A, x_B)$ if: $\sigma \in \Sigma_A \setminus \Sigma_B$ and $x_A \xrightarrow{\sigma:p_A}_A y_A$;
- $(x_A, x_B) \xrightarrow{\sigma:p_B} (x_A, y_B)$ if: $\sigma \in \Sigma_B \setminus \Sigma_A$ and $x_B \xrightarrow{\sigma:p_B}_B y_B$.

That is, shared events between two EFA are synchronized, while other events are interleaved. In addition, the updates are combined by conjunction. Determinism can also be defined to an EFA $M_{\mathrm{v}} = \langle \Sigma, V, Q, Q^{\circ}, Q^{\omega}, \rightarrow \rangle$ as follows:

- *Q-determinism*: when $|Q^{\circ}| = 1$ and, for any two transitions $x \xrightarrow{\sigma:p_1} y_1$ and $x \xrightarrow{\sigma:p_2} y_2$ implies $y_1 = y_2$, for all $x, y_1, y_2 \in Q, \sigma \in \Sigma$ and $p_1, p_2 \in \mathcal{F}$.
- *V-determinism*: when $(x, \bar{v}) \xrightarrow{\sigma} (y, \bar{w}) \wedge (x, \bar{v}) \xrightarrow{\sigma} (y, \bar{w}')$ implies $\bar{w} = \bar{w}'$, for all $x, y \in Q, \sigma \in \Sigma$ and $\bar{v}, \bar{w}, \bar{w}' \in \mathrm{dom}(V)$.

In words, Q-Determinism of EFA requires (i) a single departure point (initial state) and (ii) two any transitions with the same event always lead to the same state, does not matter how many satisfiable updates are conjuncted. In EFA, Q-Determinism makes more sense when combined to V-Determinism, i.e., when besides reaching the same state, transitions with the same event also update variables deterministically. In this paper, we assume determinism of variable assignments and states.

V. A STEP-GUIDE TO THE USE AFEs

For many practical problems, it may be easier to model constraints for a DES plant using a logical formula than structuring a large automaton. In general, this is in more tuned to the human perception about the problem to be handled. This approach simply imposes a control rule without requiring to memorize long sequences of events in the system as, in the case of EFA, memory is stored in variables, which are part of the plant behavior.

However, in order to be possible restricting a DES plant by simple formulas, using variable values, such values have to first be properly addressed by the plant model, i.e., EFA modeling the plant has to express the semantic of updates appropriately on variables. This includes a sequence of tasks which are not exactly systematic. In fact, this is an engineering task that depends on manual effort. Even though, some steps emerge naturally to the engineer and they are characterized next, attempting to somehow help the designer to address complex modeling by EFA.

(i) *Identifying variables*: Including a variable to a system model is justified face a modeling problem. As a variable provides more information about the system, it tends to facilitate modeling tasks. To identify which variable the system needs, one has to identify first which event, when associated to the variable, can update it so that necessary (extra) information on the system is provided.

(ii) *Defining each variable domain*: Identified which variable the system model needs, one has to define the structure of such variable. In this paper, we assume that variables assume only finite domains. Properly defining a variable domain implies to recognize which and how many values will be possibly assigned to the variable. For example, if a variable is Boolean, then the domain belongs to the set $\{0, 1\}$.

(iii) *Designing control specifications*: Once a variable is declared and its domain is defined, one already can use its values to design control specifications. In EFA, this is done by simply embedding constrains on transitions labeled by a given event. Constraints have the form of logical conditions, which is also known as *guards*. Summarizing, a control specification is now expressed by a logical formula embedded on a transitions labeled by the event that one aims to control.

(iv) *Identifying additional variables*: Sometimes, declaring a variable directly simplifies a given modeling task, but this indirectly implies in declaring additional variables. This situation occurs when the value to be assigned to a variable depends on another value, from other variable. In FMSs, for example, this is a common situation that emerges from processes that include data dependency among components of a manufacturing line.

(v) *Enriching the plant model*: The final step to properly combine variables to automata consists in implementing their updates. Remark that the whole framework is useless without this step. In fact, any constraint would never reach its purpose if the variable would remain unchanged. A variable update is a logical formula similar to a guard, which is also embedded on a transition. However, instead

of testing a condition on the occurrence of an event, a update replaces a variable value by another value. In this way, an update is always executed when the event over which it is implemented occurs, and it never disables the transition. Remark also that, while a constraint is implemented on a specific model to form the specification version, an update depends on the events of the plant. For that reason, updates are naturally implemented as part of the plant model, the formulas are associated to the transitions of the plant and variable values become part of the plant behavior.

The activity diagram in Fig. 6 illustrates the main idea of the proposed step-guide.

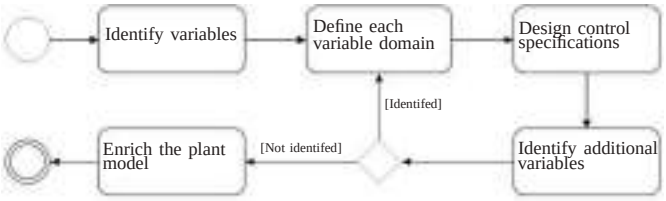

Fig. 6. Activities composing the proposed step-guide

Remark that, as soon as a variable is target as "necessary" to a modeling task (step (i)) and its domain is defined (step (ii)), the specification model can be already designed (step (iii)). In fact, the engineer is in this case anticipating that such variable value will be available whenever it is required. It remains so to make this variable value in fact available in the plant model to serve (to be tested by) the specification when it requires. This is achieved in the step (v), whose construction eventually depends on another variables (step (iv)) in order to complement a given memory to be composed. It is usual, in EFA modeling, that every component in the system is represented by a particular variable. This allow the component to start a new job after each job is finished, without waiting for control decisions.

Section V-A illustrates the use of the proposed step-guide by an example.

A. An Improved Solution to the Example

This section applies the step-guide proposed in Section V, in conjunction to the concepts described in Section IV, to reproduce the modeling of the example illustrated in Section III. For the sake of clarity, we consider just a case where two types of workpieces are received in the system (specification E^2), although any finite number of types can be recognized by the system without loosing generality.

The goal to be pursued next is to derive a coordination model for the example, such that it is equivalent to the results in Section III, however it is designed in such a way that modeling effort is reduced. We consider number of states as a measure for modeling effort, which seems to be practically reasonable.

1) Step (i):: The first action to design the example using AFE, according to the step-guide (i), is to identify possible variables which, when combined to the system model, could provide useful information to simplify a given modeling task, in this case let us assume that our modeling problem is to design specification E^2. Note that modeling E^2 requires information about the type of each workpiece positioned on the conveyor, in front the actuator. This allows to define whether or not to activate the actuator. In order to provide such information, we define two variables, p_1 and p_2, each representing the respective sector of the conveyor, storing information about the type of workpiece that is currently occupying such sector.

2) Step (ii):: The second action according to the step-guide (ii) is to determine the domain of each variable. Each value possible in a variable domain is associated to a particular type of workpiece present on the conveyor. Remark that the combination of all variable values define the state of the system. In the example, each variable is declared with a domain 3, i.e., $dom(p_1) = dom(p_2) = \{0, 1, 2\}$, such that the domain 0 is used as initial value, i.e., $p_1^\circ = p_2^\circ = 0$, and domains 1 and 2 are used respectively to identify a workpiece of the type 1 and 2. Thus, the meaning of variables assignments is as follows:

- $p_i = 1$: there is a workpiece of the type 1 on the sector i of the conveyor;
- $p_i = 2$: there is a workpiece of the type 2 on the sector i of the conveyor; for $i = 1, 2$.

3) Step (iii):: Now, we take advantage of the variable values to design a specification E_v that equivalent models E^2 (see Figure 5), however in a simpler fashion. Since information on the workpieces are now stored in variables, E^2 can be designed in such a way that traces of events no longer need to be memorized, i.e., coordination is now imposed by constraints implemented on the variable values. This is modeling task that can naturally be conducted in a modular fashion. For the example, E^2 can now be expressed by the set of models described as follows, which are then composed afterwards to form E_v:

- E_v^{1a}, E_v^{1b} and E_v^{1c}: Conveyor, sensor S_1 and sensor S_2 are sequenced, in this order;
- E_v^2 and E_v^3: Conveyor and actuators are mutually excluded, i.e., conveyor is disabled when a workpiece type i is positioned in P_i, $i = 1, 2$, and actuators A_i are disabled when conveyor is working;
- E_v^4: Actuators A_1 and A_2 are disabled while there is no workpiece on the respective position of the conveyor.

The textual version of each specification can be expressed by the EFA shown in Figure 7.

Specifications E_v^{1a}, E_v^{1b} and E_v^{1c} aim to control sensors activation in sequence (1,2), but only after the conveyor moves a workpiece in front of them. This is the way they behave in the system and it has to be properly mapped by the models.

Model E_v^2 was developed to ensure that the conveyor will be activated only when there are no workpieces in front of their

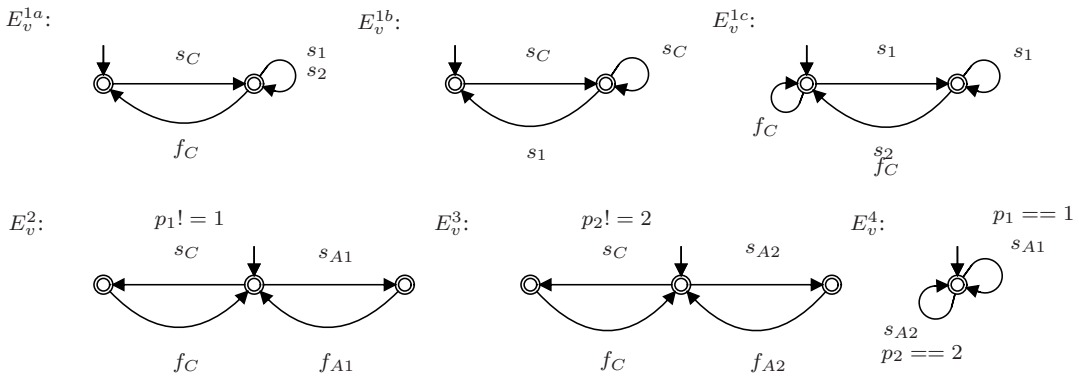

Fig. 7. Modular version of the specification $E_v = E_v^{1a} \parallel E_v^{1b} \parallel E_v^{1c} \parallel E_v^2 \parallel E_v^3 \parallel E_v^4$, modeling E^2 with EFA

respective buffers. Plus, the actuator cannot be activated while the conveyor is enabled. For that, the event s_C is enabled in the initial state as long as the condition $p_1! = 1$ is evaluated to *true*, which means that there is no workpiece in front of A_1. In addition, this model ensures that activation of the conveyor and actuator A_1 is mutually exclusive, i.e., once activated the actuator (event s_{A1}), the conveyor can only be enabled after the actuator finishes (event f_{A1}). The same idea holds for specification E_v^3 mutually excluding conveyor and actuator A_2.

Specification E_v^4 is responsible for effectively separating the workpieces to the correct buffer. For that, it tests the guard $p_1 == 1$ with the event s_{A1}, and $p_2 == 2$ with the event s_{A2}. When any of these guards is evaluated to *true*, this means that there is a workpiece of the type i, in front the position i, for $i = 1, 2$ and, therefore, the actuator is allowed to start.

An important remark to be pointed out is that EFA specifications impose control rules individually, without concerning to the other constraints. E_v^4, for example, controls the separation of workpieces without concerning to mutual exclusion or other aspects of coordination. Note that this differs from FA modeling. See Fig. 5, for example, where separating a workpiece requires to trace it from the start point to the control action point, considering all possible intermediate combinations of workpieces in the system (including empty positions in the conveyor). This memory has to be designed by the same FA. In EFA, differently, such memory is stored in variables, which imposes no similar burden to the designer. Summarizing, for practical purposes EFA specifications only make sense in conjunction, but they can be designed modularly.

4) Step iv:: Now, we define the final set V of variables. So far it has been defined 2 variables, p_1 and p_2, and this has been enough to model the specification E_v. However, it remains to be analyzed whether or not p_1 and p_2 can be properly updated by the plant without no additional variable modeling the conveyor.

Observe that, in order assign correct values to p_1 and p_2, one has to trace each workpiece since it arrives in the system. From the process in Fig. 2, workpieces arrive in the position P_0 of the conveyor. Thus, in order to assign the proper workpiece identification to p_1 and p_2, on has to know the identification itself, which is done by the sensors in P_0.

Consider now to define a complementary variable p_0 which receives workpieces identification from the sensors and transfers this to p_1 and p_2 when the conveyor moves. So, now, the new set of variables is given by $V = \{p_0, p_1, p_2\}$. By revisiting the step (ii) (see step-guide interaction in Fig. 6) we define $dom(p_0) = dom(p_1) = dom(p_2) = \{0, 1, 2\}$ and also $p_0^\circ = p_1^\circ = p_2^\circ = 0$, such that $p_0 = 0$ indicates that there is no workpiece on the sector 0 of the conveyor; and $p_0 = 1$ or $p_0 = 2$ indicates that there is a workpiece of the respective type 1 or 2 on the sector 0 of the conveyor.

5) Step v:: The final step of our example is to enrich the plant model with a logic that updates variables so as to follow the system as it evolves. This step is actually which impose semantic value to variables and allows variable values to be tested by constraints. Updates are implemented on transitions of the plant model and are responsible to memorize every possible state of the system. For the example, we propose a semantic of updates as shown in Fig. 8.

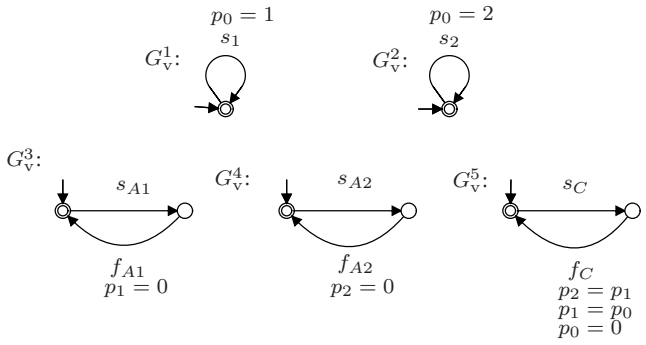

Fig. 8. EFA version $G_{\mathrm{v}} = \|_{i=1}^5 G_{\mathrm{v}}^i$ of the plant model

EFA G_{v}^1 and G_{v}^2 express the behavior of the sensors that identify the type of each workpiece and store such identification in the variable p_0 which represents the position P_0 of the conveyor.

Then, G_{v}^3 and G_{v}^4 model the actuators A_1 and A_2, respectively, and they are composed by the events *start* (s_{A1} and s_{A2}) and *finish* (f_{A1} and f_{A2}). Whenever an actuator finishes,

the value of the respective variable (p_1 or p_2) is reset to 0, meaning that the workpiece has just been removed from the conveyor and the respective position (1 or 2) is now empty.

Finally, G_{v}^{5} models the conveyor behavior by the events *start* (s_C) and *finish* (f_C). This model plays an essential role to the example, as it is responsible to transfer the variable values combination (including all positions and workpieces) from a state to another of the system.

For example, when a workpiece is identified in P_0, the variable p_0 receives its identification. Then, when the conveyor is activated, it has to reset p_0 in order to make the position P_0 ready to receive a new workpiece (update $p_0 = 0$). However, the value carried by p_0 cannot be lost when it is reset, i.e., it has to be transfered previously to p_1 (update $p_1 = p_0$). The same idea applied to the update $p_2 = p_1$.

In this way, historical records of workpieces can be kept for all positions combined, in an appropriate computational fashion. This facilitates parallel processing, besides to allow taking control decisions anytime along the process execution without having to structure large automata.

6) Example Overview:: From the composition $K_v = G_{\mathrm{v}} \parallel E_v$ one obtains an EFA version for the system under control. It can be checked (by comparing languages, for example) that K_v has the same behavior as K, even though it has been designed by automata with at most 3 states, while K contains a model with 22 states. Another way to compare K_v to K is by generating its ordinary version, i.e., its FA version. Both tests are supported by automated tools such as *Supremica* [7].

Remark that the 3-state structures (worst case) in K_v would be the same for any number of workpieces to be manufactured, or conveyor positions to be considered. In fact, the only difference would be the variable domain and respective adjusts on updates and guards, which is reasonably simple to be done. On the other hand, the FA version of the same problem would suffer with the state-space increasing whenever the process expands. Furthermore, it is also valid to be mentioned that complexity faced by combining long sequences of FA events are now replaced by the implementation of formulas, which is in general more tuned to the human perception and it is modular fashion, differently from the conventional modeling method with FA.

VI. Conclusions

This paper exploits an important limitation faced by engineers when coordinating complex Flexible Manufacturing Systems using conventional theory of *Languages* and *Automata* as modeling formalism. An example of a real system for which an optimal coordination logic would depend on an intricate structure of models is presented. Then, an advanced modeling method based on variable values is presented, together with a methodology to facilitate its adoption in industry. The benefits of the approach, compared to the conventional theory, is illustrated by the same example, where the coordination problem is solved in a much simpler way.

Although the presented approach is illustrated by a particular case, we argue that it can be naturally extended to any other real industrial process. One expects that the presented approach can help to migrate theoretical results on modeling and control synthesis to industry practices. This would allow the development of advanced techniques and tooling to handle large manufacturing systems through more concise and readable models, which is more tuned to industry profile.

We remark that, while extended automata are indeed helpful to simplify modeling tasks, they actually do not reduce computational effort. In fact, variable values are implicit states, which are taken into account for processing. In general, extended automata are expected to be processed with equivalent cost as conventional automata. We aim to present a more precise analysis on computational complexity of extended automata in our future research. Besides this, we also aim to generalize the method to other modeling contexts in the future.

References

[1] B. Esmaeilian, S. Behdad, and B. Wang, "The evolution and future of manufacturing: A review," *Journal of Manufacturing Systems*, vol. 39, pp. 79 – 100, 2016.

[2] G. Chryssolouris, *Manufacturing Systems: Theory and Practice*, 2nd ed., 2005.

[3] P. J. Ramadge and W. M. Wonham, "The control of discrete event systems," *Discrete Event Dynamic Systems*, vol. 77, pp. 81–98, 1989.

[4] C. G. Cassandras and S. Lafortune, *Introduction to Discrete Event Systems*, 2nd ed. NY: Springer Science, 2008.

[5] J. E. Hopcroft, J. D. Ullman, and R. Motwani, *Introduction to Automata Theory, Languages and Computation*, 2nd ed. Addison Wesley, 2001.

[6] L. H.R. and P. C.H., *Elements of the Theory of Computation*, 2nd ed. NJ: Prentice-Hall, 1998.

[7] K. Akesson, M. Fabian, H. Flordal, R. Malik, A. Vahidi, M. Skoldstam, and G. Cengic, *Supremica*, 2014. [Online]. Available: http://www.supremica.org/

[8] E. Tecnologia, "Manufacturing process xc241," http://www.exsto.com.br, accessed: 2016-08-01.

[9] B. A. Brandin and W. M. Wonham, "Modular supervisory control of timed discrete-event systems," *IEEE Conference on Decision and Control*, pp. 2230–2235, 1993.

[10] S. Ware and R. Su, "Time optimal synthesis based upon sequential abstraction and its application to cluster tools," *IEEE Transactions on Automation Science and Engineering*, 2016, to appear.

[11] M. H. D. Queiroz and J. E. R. Cury, "Synthesis and implementation of local modular supervisory control for a manufacturing cell," in *Proceedings of the 6th International Workshop on Discrete Event Systems*. Zaragoza, Spain: IEEE Computer Society, 2002, pp. 377–382.

[12] R. Malik and M. Teixeira, "Modular supervisor synthesis for extended finite-state machines subject to controllability," in *2016 13th International Workshop on Discrete Event Systems (WODES)*, May 2016, pp. 91–96.

[13] J. E. R. Cury, M. H. de Queiroz, G. Bouzon, and M. Teixeira, "Supervisory control of discrete event systems with distinguishers," *Automatica*, vol. 56, pp. 93 – 104, 2015.

[14] M. Teixeira, R. Malik, J. Cury, and M. de Queiroz, "Supervisory control of des with extended finite-state machines and variable abstraction," *IEEE Trans. on Automatic Control*, vol. 60, no. 1, pp. 118–129, 2014.

[15] Y. Chen and F. Lin, "Modeling of discrete event systems using finite state machines with parameters," in *IEEE Int. Conf. on Control Applications*, Anchorage, 2000, pp. 941–946.

[16] M. Sköldstam, K. Åkesson, and M. Fabian, "Modeling of discrete event systems using finite automata with variables," in *46th Conf. on Decision and Control*, 2007, pp. 3387–3392.

[17] L. Ouedraogo, R. Kumar, R. Malik, and K. Akesson, "Nonblocking and safe control of discrete-event systems modeled as extended finite automata," *IEEE Trans. on Automation Science and Engineering*, vol. 8, no. 3, pp. 560–569, July 2011.