Tackling Uncontrollability in the Specification and Performance of Manufacturing Systems

van Putten, B.J.C.

Award date:
2018

Tackling Uncontrollability in the Specification and Performance of Manufacturing Systems

Berend Jan Christiaan van Putten
b.j.c.v.putten@gmail.com
Eindhoven University of Technology, Eindhoven, The Netherlands

***Abstract*—One of the challenges in the specification and analysis of models for the design of manufacturing systems is the occurrence of events outside the control of the supervisor. Uncontrollable events are typically encountered in the description of (user) inputs, external disturbances, and exceptional behavior. This paper introduces an approach for the specification and performance analysis of manufacturing systems with partially-controllable behavior on two abstraction levels. Finite automata in terms of system activities are used to model the high abstraction level, where uncontrollability is modeled by the presence of uncontrollable events. Uncontrollability on the lower abstraction level is treated as an external disturbance to the automata. This is modeled using variables, which are added to the high-level automata in an extended finite automaton. For performance analysis, game-theoretic methods are employed to determine a guarantee to the lower-bound system performance. This result is also used in a new method to automatically compute a throughput-optimal controller which is robust to the uncontrollable behavior.**

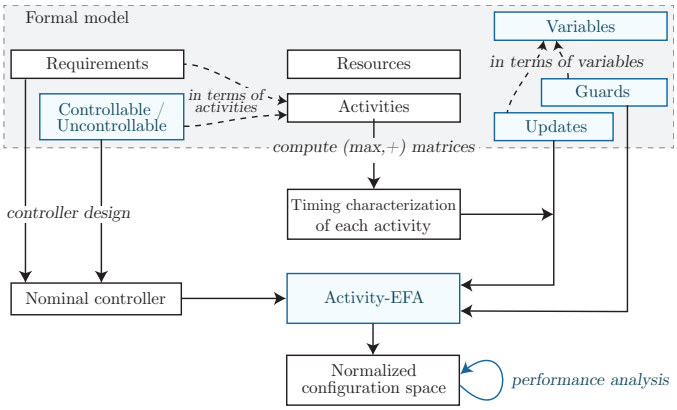


Fig. 1. Overview of the modeling approach to handle uncontrollability. Changes compared to the Activity formalism of [4] are marked blue.

I. Introduction

Over the past decades, increasing complexity of manufacturing systems has driven the development of model-based systems engineering (MBSE) and supervisory control methods to aid in the design process [1], [2]. Executable models created by these methods allow engineers to test and adjust the system before it is built. This increases design flexibility and potentially improves system performance, and reduces time-to-market. In controller design, the developed models can be used for automatic generation of supervisory controllers using controller synthesis techniques [3]. These controllers must ensure functional correctness with respect to the specifications, and provide optimal control decisions in terms of relevant performance criteria. A major challenge in the specification and analysis of models in these methods is the inclusion of system behavior outside the influence of the controller. For example, products may enter the manufacturing system with varying time intervals, or actions on a different control layer may require a response from the supervisor. In the design phase, it is useful to assess functional correctness and achievable performance under the influence of such behavior, while it is desirable for a controller to make correct and optimal control decisions when these events occur.

This paper introduces an approach, shown in Fig. 1, for the specification and performance analysis of manufacturing systems with partially-controllable behavior. It is based on the recently developed Activity model formalism [4], which enables the compositional specification of functionality and timing of manufacturing systems. Functionality in the Activity formalism is specified on two abstraction levels: high-level *activities*, and low-level *actions*. An activity expresses some deterministic functionality, such as picking up and moving a product, in terms of the individual actions and dependencies between the actions. *Resources* provide the services to execute actions; an activity consisting of multiple actions may require more than one resource for its execution. Functional design is carried out on the level of activities by the use of finite automata (*requirements*), which describe the order in which activities can be executed. The key benefit of this abstraction is a much smaller state-space for controller design compared to the case where all actions are modeled explicitly [4]. The internal structure of activities allows timing expressed on the level of actions to be collected into a $(max, +)$ timing characterization of each activity. By orthogonally adding timing characterizations to the state space, a $(max, +)$ state space can be constructed for performance analysis. Existing optimal cycle ratio algorithms [5] can be used to find a throughput-optimal controller which dispatches activities in an optimal ordering. The Activity modeling formalism is already in use within ASML[1], the world-leading manufacturer of lithography machines for the semiconductor industry, to formalize the specification of the product handling part of their machines. It acts as the semantic underpinning of a domain-specific language (DSL) which allows domain engineers to model a complete system.

[1]www.asml.com

Our modeling approach expands on the Activity formalism in two fundamental ways. First, it defines explicit uncontrollability of system behavior by partitioning activities into controllable and uncontrollable (to the supervisor) activities. This is a common concept from supervisory control theory [6], [7]. As uncontrollable activities may influence the performance that can be achieved, performance can no longer be guaranteed by looking at the controller alone. To overcome this, we use game-theoretic methods [8] to determine a guarantee to the lower bound system performance that can be achieved given the occurrence of uncontrollable behavior. Further, we use this result in a new method to automatically compute a throughput-optimal controller which is robust to the uncontrollable behavior.

The second fundamental change of our approach is the inclusion of (uncontrollable) feedback from lower abstraction levels into the level of the supervisor. We do this by augmenting the finite automaton with variables, obtaining an extended finite automaton (EFA) [9], [10]. Timing from the Activity formalism, captured by the resource timestamps and their change upon the execution of an activity, can be naturally expressed as variables and update functions in the EFA. This approach enables the specification of functionality based on information induced by the actions, including the timed state of the system. It also provides for concise and unified specification of functionality and timing in a single model.

The remainder of the paper is structured as follows. In Section II we introduce the Activity modeling formalism in more detail and outline several concepts and definitions that are used in our modeling approach. That approach is described in Section III, where we present the methods that can be used to express and model uncontrollability. We subsequently turn to optimization of a partially-controllable system in Section IV, where we describe methods to determine guarantees for system performance and to automatically compute an optimal controller that is robust to uncontrollable behavior. All introduced modeling and optimization methods are put to use in Section V, where we model and optimize an example Dice Factory system. We discuss the similarities and differences between our work and related works in Section VI. Section VII concludes the paper and describes future extensions which can be investigated.

II. Preliminaries

In this section, several of the concepts and definitions that are used in our modeling approach are introduced. We start by outlining the concepts of the Activity language [4] to the extent that is relevant for its application in the modeling concepts we present in the next section. The reader is encouraged to explore the details of the Activity language in its original publication.

A. Activity Modeling Formalism

The Activity formalism views a system as a set $\mathcal{P}$ of *peripherals* which can execute *actions* from set $\mathcal{A}$. Peripherals are aggregated into *resources* in a set $\mathcal{R}$, which can be *claimed* or *released* for the execution of some task. Given these semantics, *activities* are used to describe a deterministic procedure of co-ordinated actions, essentially aggregating them to describe a common functional operation. Activities are modeled as directed acyclic graphs (DAGs) which define the actions involved and the dependencies between them.

Definition 1 (Activity (From [4]))**.** *An activity is a DAG $(N, \rightarrow)$, consisting of a set N of nodes and a set $\rightarrow \subseteq N \times N$ of dependencies. The mapping function $M : N \rightarrow \mathcal{A} \times \mathcal{P} \cup \mathcal{R} \times \{cl, rl\}$ associates a node to either a pair (a, p) referring to an action executed on a peripheral, or to a pair (r, v) with $v \in \{cl, rl\}$, referring to the claim or release of a resource.*

A number of constraints exist on the activity structure to ensure proper resource claiming, such as that each resource is claimed and released no more than once, and that each action on a resource is preceded by the claim of that resource and succeeded by the release of it.

For the purpose of performance analysis, timing is added to the static semantics. The *execution time* of a node is given by the mapping function $T : N \rightarrow \mathbb{R}_{\geq 0}$. For the execution time of a node n with $M(n) = (a, p)$ for some $a \in \mathcal{A}, p \in \mathcal{P}$, a fixed execution time $T(n) = T(a)$ is assumed. The execution time of a node corresponding to the claim or release of a resource is 0.

Given the execution time of nodes, he dynamic semantics of an activity are concisely captured using $(max, +)$ algebra. $(max, +)$ algebra uses two operators, *max* and *addition*, which correspond conveniently to two essential characteristics of the execution of an activity: *synchronization* while a node waits for all its dependencies to finish, and *delay* as the node takes an amount of time to execute. Since activities are executed by the system resources, the timed state of the system can be expressed by a set of clocks indicating the availability time of the resources. These clock values are collected in the *resource availibility vector* $\gamma_R : \mathcal{R} \rightarrow \mathbb{R}^{-\infty}$, whose elements express when each resource $r \in \mathcal{R}$ is available and can thus be claimed for the execution of an activity. Here, we use $\mathbb{R}^{-\infty} = \mathbb{R} \cup \{-\infty\}$.

Given a vector γ_R this allows the start and completion times of the nodes of an activity to be uniquely defined.

Definition 2 (Start and completion time of a node (From [4]))**.** *Given activity $Act = (N, \rightarrow)$ and resource availability vector γ_R, the start time $start(n)$ and completion time $end(n)$ for each node $n \in N$ are given by*

$$start(n) = \begin{cases} \gamma_R(r), & \text{if } M(n) = (r, cl) \\ & \text{for some } r \in \mathcal{R}, \\ \max\limits_{n_{in} \in Pred(n)} end(n_{in}), & \text{otherwise}, \end{cases}$$

$$end(n) = start(n) + T(n),$$

where $Pred(n) = \{n_{in} \in N \mid n_{in} \rightarrow n\}$ is the set of predecessor nodes of node n.

As the dynamic semantics of an activity $Act = (N, \rightarrow)$ are uniquely defined by N, $\rightarrow$ and timing function T, the step change of timestamp vector γ_R when an activity is executed can be defined.

Definition 3 (Dynamics of an activity execution (From [4])). *Given activity $Act = (N, \rightarrow)$, timing function T, and resource availability vector γ_R, the new resource availability vector after execution of Act is given by γ'_R whose elements are given by*

$$\gamma'_R(r) = \begin{cases} \gamma_R(r), & \text{if } r \notin R(Act), \\ end(n), & \text{if } r \in R(Act) \wedge M(n) = (r, rl) \\ & \text{for some } n \in N, \end{cases}$$

where $R(Act) = \{r \in \mathcal{R} \mid (\exists n \in N \mid M(n) = (r, cl))\}$ is the set of resources used by Act.

The dynamics of an activity can be equivalently expressed using a $(max, +)$ matrix multiplication of vector γ_R with an activity matrix $\mathbf{M}_{Act}$, called the $(max, +)$ *characterization* of activity Act. An algorithm for automatically computing the $(max, +)$ characterizations can be found in [11]. The new resource availability vector after execution of an activity is computed in this way by $\gamma'_R = \mathbf{M}_{Act} \otimes \gamma_R$. Here, $\gamma'_i = \max_{k=1}^{m} (M_{ik} + \gamma_k)$ is the new value of the i-th element of γ_R after the update, where m is the size of γ_R. The dynamics of the execution of a sequence of activities is then given by repeated matrix multiplication.

Functional requirements of the system are expressed in terms of the order in which activities can be executed, modeled by compositional finite automata with transitions labeled with activities. Supervisory control synthesis [6] can be used to obtain an Activity-FSM, which contains all allowed activity sequences. As the supervisor is not aware of the inner structure of an activity, it may influence the order in which activities are executed, but not the order of the actions within an activity.

Definition 4 (Activity-FSM (From [4])). *An Activity-FSM F on $\mathcal{A}ct$ is a tuple $\langle L, \mathcal{A}ct, \delta, l_0 \rangle$ where L is a finite, nonempty set of locations, $\mathcal{A}ct$ is a nonempty set of activities, $\delta \subseteq L \times \mathcal{A}ct \times L$ is the transition relation, and l_0 is the initial location. Let $l \xrightarrow{Act} l'$ be a shorthand for $(l, Act, l') \in \delta$.*

From the Activity-FSM, a $(max, +)$ state space can be constructed which records the different timed configurations of the system. The $(max, +)$ state space can be used for performance analysis of the system. As future behavior is affected only by the relative timing differences of the resources, and not by their absolute offset from the initial configuration, the resource availability vector is normalized in each configuration. This results in the normalized $(max, +)$ state space. Using $\|\gamma\| = \max_i \gamma_i$ to denote the maximum element of a vector γ, the normalized resource availability vector is given by $\gamma_R^{norm} = \gamma_R - \|\gamma_R\|$. We use $\mathbf{0}$ to denote a vector with all zero-valued entries.

Definition 5 (Normalized $(max, +)$ state space (From [4])). *Given Activity-FSM $\langle L, \mathcal{A}ct, \delta, l_0 \rangle$, resource set $\mathcal{R}$, and a $(max, +)$ matrix $\mathbf{M}_{Act}$ for each $Act \in \mathcal{A}ct$, the normalized $(max, +)$ state space is defined as a 3-tuple*

$$\langle C, c_0, \Delta \rangle,$$

where:

- *$C = L \times \mathbb{R}^{-\infty^{|\mathcal{R}|}}$ is a set of configurations consisting of a location and a normalized resource availability vector,*
- *$c_0 = \langle l_0, \mathbf{0} \rangle$ is the initial configuration,*
- *$\Delta \subseteq C \times \mathbb{R} \times \mathcal{A}ct \times C$ is the labeled transition relation consisting of the transitions in the set*

$$\{\langle \langle l, \gamma_R \rangle, \|\gamma'_R\|, Act, \langle l', \gamma_R'^{norm} \rangle \rangle \mid l \xrightarrow{Act} l' \wedge \gamma'_R = \mathbf{M}_{Act} \otimes \gamma_R\}$$

B. Extended Finite Automata

An extended finite automaton (EFA) [10] is a finite automaton augmented with *variables*. *Guard expressions* and *update functions* can be added which restrict behavior depending on the variable values, and update the variable values during a transition, respectively. An EFA can be viewed as a 7-tuple,

$$\langle L, \Sigma, \delta, V, G, U, \langle l_0, v_0 \rangle \rangle,$$

where L is the location set, Σ is the alphabet, $\delta \subseteq L \times G \times \Sigma \times U \times L$ is the state transition relation, V is a set of variables, G is a collection of boolean expressions over the variables, U is a collection of value assignments to the variables of the form $v := e$, where e is an expression over the variables, and $\langle l_0, v_0 \rangle$ are the initial location and variable values. We denote a valuation of the variables by v and the collection of all valuations by *Val*(V). Resolving the guard expressions and update functions, the explicit transition relation [10] in an EFA is given by

$$\hat{\delta} = \{(l, v, \sigma, l', v') \in L \times \mathit{Val}(V) \times \Sigma \times L \times \mathit{Val}(V) \mid (\exists (l, g, \sigma, u, l') \in \delta \mid g(v) \wedge v' = u(v))\}.$$

C. Ratio Games

Ratio games have been proposed for the computation of robust controllers in supervisory control problems [8], [12]. We will use these to compute a robust optimal supervisor which is throughput-optimal given a partially-controllable system.

A ratio game is a two-player infinite game, played on a finite double-weighted directed graph, where each edge has two associated non-negative weights $w_1(e)$ and $w_2(e)$. Each turn of the game constitutes a move by one of the players of a pebble on a vertex in the game over an edge to an adjacent vertex. The two players have opposite goals: whereas one player wants to maximize the ratio of the sum of weights w_1 and w_2 in the limit of an infinite play, the other player wants to minimize it.

Definition 6 (Ratio game graph (From [8])). *A ratio game graph Γ is a 5-tuple $\langle V, E, w_1, w_2, \langle V_0, V_1 \rangle \rangle$, where $G^\Gamma = \langle V, E, w_1, w_2 \rangle$ is a weighted directed graph and $\langle V_0, V_1 \rangle$ partitions V in a set V_0, belonging to player 0, and V_1, belonging to player 1. Weighted graph G^Γ consists of a finite set V of vertices, a set $E \subseteq V \times V$ of edges, and weight functions $w_1, w_2 : E \rightarrow \mathbb{R}_{\geq 0}$ which assign two nonnegative real weights to edges.*

G^Γ is assumed to be *total*, meaning that for all $v \in V$, there exists a $v' \in V$ such that $(v, v') \in E$. Note that although V is

partitioned in disjoint subsets V_0 and V_1 for players 0 and 1, the game graph does not need to be bipartite. This means that a player may sometimes make multiple consecutive moves, i.e. there may exist $(v, v') \in E$ with both v and v' in either V_0 or V_1.

In an *infinite game*, two players move a pebble along the edges of the graph for infinitely many rounds. Taking V^ω as the set of infinite sequences over V, a *play* $\pi = v_0 v_1 ... \in V^\omega$ is an infinite sequence of vertices such that $(v_i, v_{i+1}) \in E$ for all $i \geq 0$. A *ratio game* [12] is an infinite game played on a ratio game graph. The *ratio* of a play π is defined as

$$Ratio(\pi) = \lim_{m \to \infty} \lim_{l \to \infty} \frac{\sum_{i=m}^{l} w_1(v_i, v_{i+1})}{1 + \sum_{i=m}^{l} w_2(v_i, v_{i+1})}.$$

A *memoryless strategy* σ_i for player i is a function $\sigma : V_i \to V$ which defines a unique successor v' for every vertex $v \in V_i$ such that $(v, v') \in E$. A play is *consistent* with memoryless strategy σ of player i if $v_{j+1} = \sigma(v_j)$ for all $j \geq 0 \wedge v_j \in V_i$. Strategy σ_i of player i is considered *optimal* if it achieves for all $v \in V$ the highest (lowest) ratio of a play starting in v that is consistent with σ_i given the worst-case strategy by the opponent player. It is shown in [8] that in a ratio game there exist strategies σ_0 for player 0 and σ_1 for player 1 for which the ratios of consistent plays are equal and optimal. The optimal (maximum) ratio that player 0 can achieve is thus equal to the optimal (minimum) ratio that player 1 can achieve.

III. Modeling for Uncontrollability

In this section, the concepts which we use to model uncontrollability in the Activity formalism are addressed. First, we show how the occurrence of activities outside the influence of the controller can be expressed. We then introduce the Activity-EFA, which extends an Activity-FSM with variables, that is used to model functionality based on system feedback.

As a running motivating example, consider a paint robot which paints products in a manufacturing line. Following an initialization procedure, unpainted products enter the system. They may occasionally do so at a misaligned position. When this is the case, the paint robot must first make an alignment move to the product before it can start painting. In what follows, we will use this motivating example to highlight how each of the introduced concepts can be used in a model. The resulting automaton of this example is shown in Figure 2.

A. Controllability of activities

In supervisory control theory, a transition in a transition system is typically labeled with an *event* from a nonempty *alphabet* [6], [7]. To express events whose occurrence the supervisor cannot influence, but can observe and may wish to respond to, the alphabet is in these theories partitioned into a set of controllable events, and a set of uncontrollable events. An example of a controllable event is the command to turn on the paint nozzle of the paint robot, while an uncontrollable event may be a sensor signaling that a product has arrived at the paint station. In the Activity formalism, these events constitute actions, which are aggregated to form deterministic activities. We consider uncontrollability at the level of activities, so that we retain the deterministic nature of an activity. Variations on the action level can be modeled on the activity level as separate activities.

In the Activity-FSM, each transition is labeled with an activity. We express uncontrollability in the same way as in supervisory control theory. The nonempty set $\mathcal{A}ct$ of activities is partitioned into a set $\mathcal{A}ct^c$ of controllable activities, and a set $\mathcal{A}ct^u$ of uncontrollable activities. An example of a controllable activity in the paint robot example is the operation of painting a product (Paint), while a product entering the system at either an aligned (Enter_OK) or a misaligned (Enter_NOK) position are uncontrollable activities.

A supervisory controller is said to be *controllable* with respect to the system it acts on, if it does not attempt to disable the occurrence of any uncontrollable event which is otherwise admissible in the system. It may, however, disable a controllable event leading up to the uncontrollable event, effectively preventing the uncontrollable event from becoming admissible in the system.

B. Activity-EFA

Timing analysis in the Activity formalism is achieved by adding to the functional Activity-FSM a vector consisting of resource availability times, and rules which describe their change when a transition is taken, i.e. an activity is executed. This bares a close resemblance to an EFA, which augments an automaton with variables and update functions which operate on the variables when a transition is taken. EFAs further allow the definition of guards that describe conditions for the execution of a transition based on the variable values. We gain this expressivity by combining the components of the formal model into an EFA.

Definition 7 (Activity-EFA)**.** *Given activity set $\mathcal{A}ct$, an Activity-EFA is an extended finite automaton $\langle L, \mathcal{A}ct, \delta, V, G, U, \langle l_0, v_0 \rangle \rangle$ in which $\mathcal{A}ct$ is the alphabet.*

The Activity-EFA is an augmentation of an ordinary automaton, whose transitions are labeled with activities, with guard expressions and update functions. Transitions in the Activity-EFA are enabled only when the associated guard expressions evaluate *true*. Upon taking a transition, the values of the variables are updated in accordance with the associated update function. As shorthand for a transition in the Activity-EFA, we write $l \overset{Act}{\to}_{g/u} l'$, where $l, l' \in L$, $Act \in \mathcal{A}ct$, $g \in \mathcal{G}$ and $u \in \mathcal{U}$. In case g is absent, we take that g is *true*. If u is absent, the value of the variables is carried over, i.e. $v' := v$.

Variables. A finite set of variables can be defined in V which augment the state of the system. In the timing analysis of the Activity formalism, the timed state of the system is expressed in terms of resource availability timestamps in a vector γ_R. For each resource $r \in \mathcal{R}$, we now assume a variable τ_r representing the availability time of that resource, which we do not normalize for now. This enables the specification of functionality based on the timing information they provide. As $(max, +)$ algebra is used for the computation of the availability times, all τ_r are defined on the domain $\mathbb{R}^{-\infty}$.

In addition to the timing variables, other variables can be defined. These can be of a type that is useful to their application. In the paint robot example, a useful variable may express the real-valued position (*Pos*) at which a product entered the manufacturing line. To distinguish between variables which contain timing information and all other variables, we collect the timing variables τ_r for all $r \in \mathcal{R}$ in the set V^τ. All other variables go in the set V^o. The complete set of variables is then given by $V = V^\tau \dot\cup V^o$. When it is convenient to refer to the timing variables in vector form, we will continue to denote this by γ_R.

Updates. An update maps the values of variables to their new values when a transition is taken. Updates are written as

$$v' := u(v), \text{ where } v, v' \in \mathit{Val}(V). \tag{1}$$

The values of resource timestamps in V^τ are updated by $(max, +)$ multiplication. Given a vector γ_R of timing variables and some matrix $\mathbf{M}$, a timing update is written as $u^\tau(\gamma_R) = \mathbf{M} \otimes \gamma_R$. For any transition in the Activity-EFA, multiplication matrix $\mathbf{M}$ is given by the $(max, +)$ characterization $\mathbf{M}_{Act}$ of the corresponding activity. These can be automatically computed from the structure of the activity and timing function T on the nodes of the activity, as pointed out in Section II-A.

Updates of variables in V^o, on the other hand, are supplied by the modeler. We denote these updates by $u^o(v)$. In the paint robot example, two updates are defined for position variable *Pos*. The first update assigns the coordinate of the aligned position. This update is applied to the transition labeled with Enter_OK. The second update assigns the coordinate of the misaligned position, and is applied to the transition labeled with Enter_NOK. Given the separate definitions of updates for timing variables and other variables, an update in the Activity-EFA always consists of two parts: the update of timing variables, given by u^τ, and the update of ordinary variables, given by u^o.

Guards. Guards are written as Boolean expressions $g(v)$ over the variables, which map a valuation $v \in \mathit{Val}(V)$ of the variables to a Boolean value *true* or *false*. In the paint robot example, guards expressed on the Paint and Align activities ensure that the robot makes an alignment move whenever a product enters at an insufficiently aligned position.

Guards affect the supervisory control properties of the EFA as they have the power to disable a transition. Methods found in the literature [13] can be used to verify controllability and nonblockingness of the EFA. In case the Activity-EFA without guards is known to be controllable, it is easily seen that a sufficient condition for controllability is the absence of guards on uncontrollable transitions.

As before, we use the Activity-EFA to construct a state space in terms of the system configurations, which can be used for performance analysis.

Definition 8 (Configuration space)**.** *Given an Activity-EFA $\langle L, \mathcal{A}ct, \delta, V, G, U, \langle l_0, v_0\rangle\rangle$, the configuration space is defined as a 3-tuple*

$$\langle C, c_0, \Delta\rangle, \tag{2}$$

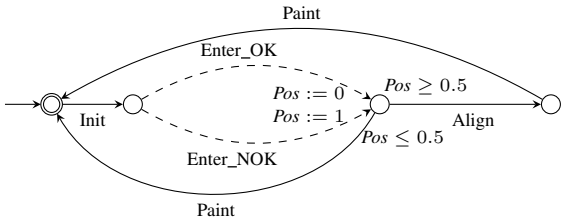


Fig. 2. Automaton for a paint robot with controllable and uncontrollable transitions, variable *Pos*, guards, and updates.

where:

- $C = L \times V$ *is the (possibly infinite) set of configurations consisting of a location and a collection of values;*
- $c_0 = \langle l_0, v_0\rangle$ *is the initial configuration;*
- $\Delta \subseteq C \times \mathbb{R} \times \mathcal{A}ct \times C$ *is the transition relation consisting of the transitions in the set*

$$\{(\langle l, v\rangle, \theta, Act, \langle l', v'\rangle) \mid (\exists l \xrightarrow{Act}_{g/u} l' \in \delta \mid g(v) \wedge v' = u(v) \wedge \theta = \|\gamma'_R\| - \|\gamma_R\|)\},$$

in which γ_R is a vector of the timing values in v, before the transition, and γ'_R is the corresponding vector in v', after the transition.

A configuration refers to both a location in the EFA and a valuation of the variables. When a transition is taken, the variable values in the new configuration are updated according to the specified update function. The timing variables are used in this definition to determine *transit time* θ of a transition, which expresses by how much the largest resource availability time has shifted by the execution of an activity. The transit time is used in performance analysis to determine the duration of activity sequences.

Variables in V may be defined on a possibly infinite domain, potentially leading to an infinite configuration space $L \times V$. For example, consider a counter variable which is incremented for each finished product in a cyclic production process. Many performance analysis and optimization algorithms, including the ratio games we wish to employ, require a finite solution space to terminate [8], [14]. By instead defining variables on a finite domain, we can ensure a finite configuration space for performance analysis. This does not restrict our modeling efforts in many practical situations. An infinite counter variable may be replaced by a counter which can only go up to a feasible number of products in a shift. Even values from sensors can normally be expressed on a finite domain, as sensors have a finite precision and their values are typically discretized. We denote the set of variables defined on a finite domain by $\hat{V}$.

Normalization of the timing variables, as used in the construction of the normalized $(max, +)$ state space (Definition 5), serves the same purpose. Using normalization, it has been shown that the state space remains finite under the condition that each cycle contains a dependency on each of the resources at least once, i.e. no part of the system evolves completely independently of another part [15]. We denote a valuation of

the variables in $\hat{V}$ before normalization of the timing variables by $\hat{v}$, and after normalization of the timing variables by $\hat{v}^{norm}$.

Normalization affects the expression of guards on timing variables, as the meaning of their absolute values is no longer clear. It is easily shown that the relative difference between timing variables is invariant under normalization. Suppose we take the difference between the values $v_{\tau,1}$ and $v_{\tau,2}$ of two timing variables in the same valuation v and apply normalization using a value $\|\gamma\|$, then

$$\begin{aligned} v_{\tau,1}^{norm} - v_{\tau,2}^{norm} &= (v_{\tau,1} - \|\gamma\|) - (v_{\tau,2} - \|\gamma\|) \\ &= v_{\tau,1} - v_{\tau,2}. \end{aligned}$$

A meaningful guard involving timing variables is therefore an expression over the difference between two of these variables. We define $W^\tau = \{v_i - v_j \mid v_i, v_j \in V^\tau\}$ as the set of all differences between the members of the set of timing variables. A meaningful guard is then limited to expressions over the extended set of variables $W^\tau \cup \hat{V}^o$. We designate the set of such guards as $\hat{G}$.

A finite state space can now be constructed for performance analysis of the system.

Definition 9 (Normalized configuration space)**.** *Given an Activity-EFA $\langle L, \mathcal{A}ct, \delta, \hat{V}, \hat{G}, U, \langle l_0, \hat{v}_0\rangle\rangle$, the normalized configuration space is defined as a 3-tuple*

$$\langle \hat{C}, c_0, \Delta\rangle, \tag{3}$$

where:

- *$\hat{C} = L \times \hat{V}$ is the finite set of configurations consisting of a location and a collection of values;*
- *$\hat{c}_0 = \langle l_0, \hat{v}_0\rangle$ is the initial configuration;*
- *$\Delta \subseteq \hat{C} \times \mathbb{R} \times \mathcal{A}ct \times \hat{C}$ is the transition relation consisting of the transitions in the set*

$$\begin{aligned} \{(\langle l, \hat{v}\rangle, \theta, Act, \langle l', \hat{v}'^{norm}\rangle) \mid (\exists l \xrightarrow{Act}_{\hat{g}/u} l' \in \delta \mid \\ \hat{g}(v) \wedge \hat{v}' = u(\hat{v}) \wedge \theta = \|\gamma'_R\|)\}, \end{aligned}$$

in which γ'_R is the non-normalized vector of timing values in v', after the transition.

Since we are interested in system descriptions with a finite number of configuration, in what follows we will drop the circumflex in our definitions and assume that each Activity-EFA and configuration space is defined in accordance with Definition 9.

Notice that the original Activity-FSM and its timing characterization are embedded in the definition of an Activity-EFA. To see this, given an Activity-FSM $\langle L, \mathcal{A}ct, \delta, l_0\rangle$, resource set $\mathcal{R}$, and $(max, +)$ matrices $\mathbf{M}_{Act}$ for each $Act \in \mathcal{A}ct$, construct the Activity-EFA $\langle L, V^\tau, \mathcal{A}ct, \emptyset, \mathcal{U}, \delta', \langle l_0, \mathbf{0}\rangle\rangle$, where

- $V = V^\tau$ contains the timing variables for each $r \in \mathcal{R}$,
- $\mathcal{U} = \{\mathbf{M}_{Act} \otimes \gamma_R \mid Act \in \mathcal{A}ct\}$,
- $\delta' = \{(l, \mathit{true}, Act, u, l') \mid (l, Act, l') \in \delta \wedge u = \mathbf{M}_{Act} \otimes \gamma_R\}$,

Any Activity-FSM may thus equivalently be expressed as an Activity-EFA and gain the ability to express functionality based on timing or other variables.

IV. Robust Optimal Supervisor Design

A reachable cycle in the configuration space relates to the periodic execution of the system. Each transition in the configuration space is assigned a *transit time* relating to the duration of an activity. A second value is assigned to each transition, which we call the *reward* of the transition. The transit time of a cycle equals the sum of the transit times of the transitions of the cycle, and the reward of a cycle equal the sum of the rewards. We define the *cycle ratio* as its reward divided by its transit time[2]. Optimal cycle ratio algorithms [5], [15] can be used to find cycles in the system with the highest value, or the lowest value for the cycle ratio. These cycles relate to best-case, or worst-case system performance.

The partitioning of activities into controllable activities and uncontrollable activities allows us to consider new scenarios. It is likely that we wish to design a controller which attains the highest achievable performance by directing system execution to an optimal cycle. It may do so by determining the choice of controllable activities. Uncontrollable activities act as a disturbance, which may aid system performance if they lead to a higher cycle ratio, or hurt performance if they lead to a lower cycle ratio. Viewed this way, the highest cycle ratio that can be achieved by the controller, given worst-case uncontrollable disturbances, determines a guarantee to the lower bound of periodic system performance.

The challenge is now to find a controller which makes optimal choices. We are interested in two results: i) a guarantee to the achievable lower-bound system performance, and ii) an optimal controller which is robust to uncontrollable behavior, i.e. it makes optimal choices regardless of the occurrence of uncontrollable disturbances. The structure of this challenge corresponds well to ratio games, as introduced in Section II-C. Recall that the result of a ratio game are optimal strategies for player 1 and player 2, given that one player aims to achieve the highest ratio, and the other player aims to achieve the lowest ratio. In our challenge, player 1 is the controller, who controls the moves on edges with a controllable activity, and whose aim is to achieve the highest cycle ratio. Player 2 is the environment, who controls the moves on edges with an uncontrollable activity and aims to achieve the lowest cycle ratio. Optimal strategies then relate to the the best achievable controller performance given worst-case disturbances.

A ratio games requires a graph whose vertices are partitioned in vertices controller by player 1, and vertices controlled by player 2. The graph structure of the configuration space is typically not partitioned in this way, as from any location there may exist both controllable and uncontrollable transitions to another configuration. We must therefore introduce a partitioning. Given that a characteristic property of uncontrollable behavior is that it may not be blocked by a controller, we assume a priority of uncontrollable behavior over controllable behavior. In our game setting, this can be interpreted as the rule that at each turn in which both players can make a move, the

[2]In a manufacturing system, *throughput* refers to the number of produced products per unit of time. In this context, we can assign a reward of value 1 to each edge associated with an activity that delivers a finished product. The cycle ratio then equals the throughput of the system when operating on that cycle.

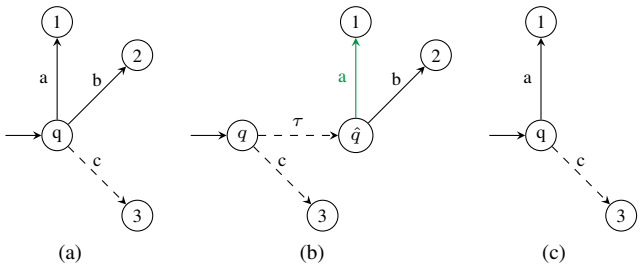


Fig. 3. Partitioning of a transition system with competing controllable and uncontrollable players. Fig. 3a shows the system before partitioning. Fig. 3b shows the system after partitioning with the green arrow indicating the optimal controller choice. Fig. 3c shows the resulting robust optimal controller.

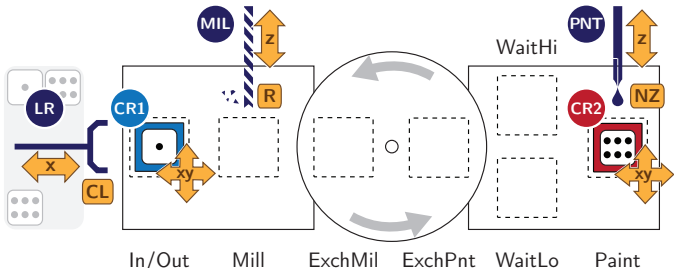


Fig. 4. Top-down view of the Dice Factory example manufacturing system with two carts (CR1 and CR2), two production stages (MIL and PNT) and a load robot (LR). The resource names are shown in colored circles, peripheral names are shown by orange labels, and the names of positions with a dashed outline are shown below or above the position.

player who commands the uncontrollable moves may make a move first. He may equally choose to pass his turn to the other player, at which point the player in charge of the controllable moves may make a move.

We model this using the concept of *forcible events* [16] as encountered in a supervisory control context, where forcible events may momentarily prevent the occurrence of *preemptible events*. Consider the example system of Fig. 3a, where from state q there are two controllable transitions a and b, and an uncontrollable transition c. We introduce a new uncontrollable activity $\tau \notin \mathcal{A}ct$ which is the 'pass a turn' activity. No update function is assigned to this activity and variable values carry over when it is executed. In the game graph, the reward and transit time of τ are both 0. For a location q in which both a controllable and an uncontrollable transition are possible, we introduce a new location $\hat{q}$ and transition $(q, \tau, \hat{q})$ so that it can be reached via the 'pass a turn' move. Transitions from location q labeled with a controllable activity are removed from location q and added to location $\hat{q}$. In the example, transition $(q, a, 1)$ becomes $(\hat{q}, a, 1)$ and transition $(q, b, 2)$ becomes $(\hat{q}, b, 2)$. Uncontrollable transitions are maintained at the original location. We end up with the partitioned system shown in Fig. 3b.

Ratio game algorithms [8] can be applied to the partitioned configuration space where rewards and transit times are assigned as weights w_1 and w_2 to the transitions. A guarantee to the achievable periodic system performance is given by the ratio of a reachable cycle consistent with the computed optimal strategies of both players. The absence of a reward or transit time for τ ensure that its introduction does not affect the computed ratio.

A play consistent with the strategies of both players gives us one periodic sequence of activities. For our controller, we are interested in all activity sequences in which the controller follows an optimal strategy for *any* strategy of the uncontrollable environment. To compute this controller, we start with the partitioned configuration space S and optimal controller strategy σ_c. Recall that σ_c defines a unique successor $c' = \sigma(c)$ for each $c \in C^{ctb}$, where C^{ctb} contains the controllable locations in the partitioned configuration space. From each location in the partitioned configuration space, we remove the outgoing controllable transitions that are not consistent with the computed strategy, such that $c' = \sigma(c)$ for every remaining transition $(c, \theta, Act, c') \in \Delta$ of S where $c \in C^{ctb}$. All uncontrollable transitions are retained since we want keep all possibilities of uncontrollable behavior. As we do not observe the artificial activity τ in our original system, we use the natural projection [16] $P : \mathcal{A}ct^{\tau} \to \mathcal{A}ct$, where $\mathcal{A}ct^{\tau} = \mathcal{A}ct \cup \{\tau\}$, to project out τ from the configuration space. This effectively erases all transitions labeled with τ and maps the remaining transitions of each artificially introduced location q' back to their original location q. Finally, we can compute the *reachable subautomaton* [16] to obtain the supervisory controller which is minimal, controllable, and throughput-optimal. This result is shown in Fig. 3c. We call this the *robust throughput-optimal controller*, as it determines throughput-optimal controller choices while respecting the occurrence of uncontrollable behavior.

V. Example: Dice Factory

In this section we show how our approach is used for the specification and performance analysis of a manufacturing system. As an example we take the fictional dice factory shown in Fig. 4, which produces high-grade game dice in two stages. During the first stage, holes are milled in a side of an unfinished die in a pattern corresponding to the value of that side. The milled die is then transported to the second stage via an exchange disc, where the milled holes are filled with a colored paint. This system is a simplification of the product processing within an ASML TWINSCAN machine, modeled using similar resources and constraints.

We model the example system in two steps. First, the nominal system is introduced which describes the system components and a simple production cycle. To this we add two refinements which introduce different forms of uncontrollable behavior. This shows how we can easily adapt existing or new models to include uncontrollability.

A. System description

Our example system contains five resources. Two carts (CR1 and CR2) are used to transport dice in the system. A load robot (LR) is used to put an unfinished die from the input buffer onto a waiting cart, and to pick a processed die from a cart and put it in the output buffer. The final two resources are processing stations. The mill station (MIL) mills a number of holes in a

TABLE I
SET OF ACTIVITIES OF THE NOMINAL EXAMPLE SYSTEM

CartExchange	Move_ToMill_⋆	Move_ToInOut_⋆
Mill_⋆	Move_MillToExch_⋆	PutOnOutput_⋆
Paint_⋆	Move_ToPaint_⋆	
PickFromInput_⋆	Move_PaintToExch_⋆	

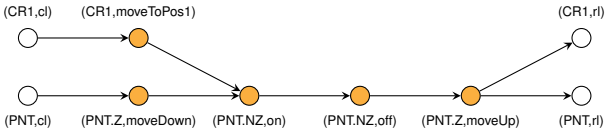


Fig. 5. Activity Paint_CR1 showing nodes and dependencies. Release and claim nodes are represented by a clear circle, while action nodes are colored yellow. An incoming arrow represents a dependency on the source node of the arrow.

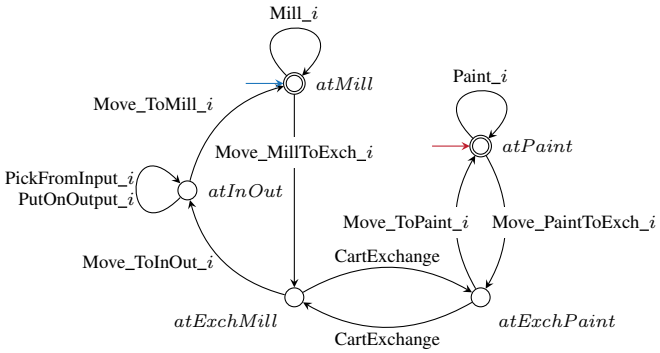


Fig. 6. Plant automaton of cart i, where i can be 1 or 2. The initial locations of carts 1 and 2 are marked by a blue and a red arrow, respectively.

side of a die, corresponding to the pattern on that side. The paint station (PNT) fills the holes with a colored substance.

The two functionally equivalent carts have a single peripheral: an undercarriage (XY) which moves the cart to a position in the XY-plane. The load robot has two peripherals: an arm (X) which moves to and from a waiting cart, and a clamp which grips the die when the arm is moving. The mill station has two peripherals, an R-motor (R) which rotates the mill cutter and a Z-motor (Z) which moves the mill cutter up and down. The paint station has a nozzle (NZ) which turns on and off to release paint and a Z-motor which moves the nozzle to and from the die. Note that the mill and paint stations are fixed in the XY-plane and the carts move during processing to correctly position the die for a drill or paint operation.

Each die is processed in the system by following the same life cycle. An unfinished die is put on either cart 1 or cart 2 by the load robot. We assume no preference to which cart picks up a die. The cart then transports the die to the mill station where it is milled. Upon finishing, the cart moves to the exchange disc, on which it moves to the paint stage. The cart then transports the die to the paint station where it is painted. The cart subsequently moves to the exchange disc for the reverse cart exchange operation. The cart exchange always moves the two carts simultaneously; the cart on the mill side is moved to the paint side, and vice-versa. Finally, the cart transports the finished die to the In/Out position where it is offloaded by the load robot.

B. Activities

In the nominal system, there are two production activities, Mill and Paint, which operate on either cart 1 or cart 2. The CartExchange activity works on both carts as they are moved simultaneously. Finally, there are several Move_ activities by cart 1 and cart 2 which move the respective cart from one position to another. The labeled positions are shown in Fig. 4 as dashed cart outlines; the WaitHi and WaitLo positions are not used in the nominal system.

Since the two carts are modeled as separate resources, we define individual activities for each cart. As they are functionally equivalent, we define their activities once and use the following notation to refer to the activity executed by cart 1 or cart 2. Given a cart activity Act and cart identifiers $I = \{\text{CR1}, \text{CR2}\}$, the set $Act\_\star = \{Act\_i \mid i \in I\}$ contains activity Act executed by cart 1 and by cart 2.

All activities are defined by specifying the resources, actions, and dependencies between actions as outlined in section II. As an example, consider activity Paint_CR1 for cart 1, shown in Fig. 5, which paints a die on cart 1. The full set of activities of the nominal system is listed in Table I. The individual activity structures can be found in Appendix A.

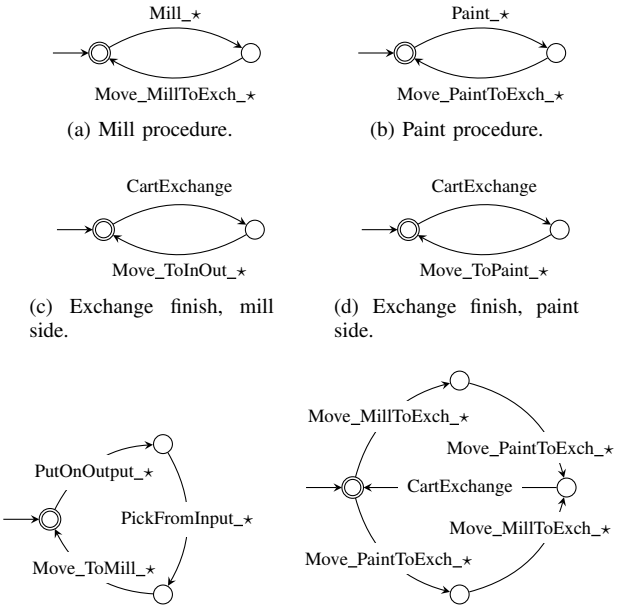
C. Allowed activity sequences

For the specification of allowed activity sequences, we model the system as a set of automata with transitions labeled with the activities. We first define a set of *plant* automata which define all admissible, if not necessarily desired, behavior. These are complemented with *requirements* which enforce the functional specification of our system. Multiparty synchronization is used between the automata in the set so that execution of shared events is synchronous.

The plant model of the nominal system consists of two automata, for carts 1 and 2. These automata are equal except for the initial locations. Cart 1 is initially on the mill stage at location $atMill$, while cart 2 starts on the drill stage at location $atPaint$. The automaton of a cart is shown in Fig. 6, where the initial locations of carts 1 and 2 are highlighted.

The plant model is complemented by six requirements which enforce the correct sequence of activities in the nominal production cycle. These requirements are shown in Fig. 7. Requirements 7a and 7b ensure that a die is processed by one mill or paint operation before it is moved along. Requirements 7c and 7d ensure that the carts are not exchanged consecutively before a mill or paint operation has taken place. Requirement 7e avoids product collisions by specifying that a die must be put on the output buffer before a new die can be picked up for processing. Requirement 7f ensures that the cart exchange cannot take place before both carts are ready to be exchanged.

The components of the nominal system are now specified and may be used to construct an Activity-FSM with all nominal activity sequences, or a normalized $(max, +)$ state space for performance analysis. Because we are interested in the specification and performance of a system including



(a) Mill procedure. (b) Paint procedure. (c) Exchange finish, mill side. (d) Exchange finish, paint side. (e) Input-output sequence. (f) Exchange entry procedure.

Fig. 7. Requirement automata which enforce the nominal production life cycle.

uncontrollable behavior, we will instead continue to model these aspects.

D. Refining with uncontrollable behavior

For the first refinement, consider that the system must produce multiple patterns depending on the value of the side of a die that is processed. The pattern affects the duration of the mill and paint operations, as well as the end position of a cart which moves during these operations. We assume that the choice of a value is unknown to the controller until a mill or paint operation is executed, for instance because it is determined by a different controller.

For the second refinement, we introduce a change to the entry procedure to the exchange disc. As a motivation, a systems engineer might want to increase the system lifetime by reducing unnecessary load on the bearings of the exchange disc. Because the mill and paint operations may finish at different times, it is desirable that the cart which finishes first waits away from the exchange disc until the other cart has finished its operation. Two wait positions, WaitHi and WaitLo, are appointed beside the exchange disc so that a cart may move to the wait position to which it is closest, as indicated in Fig. 4.

For reasons of brevity, we consider two patterns, producing a side with either value 1 or value 6. For the same reason, we only consider the refinements on the Paint side; the mill operation and exchange disc entry procedure on the Mill side remain as in the nominal system. We model the refinements by making changes to the activities and plants, defining variables and corresponding updates, and by adding requirements, which use guards in terms of the variables, to enforce the refined system behavior.

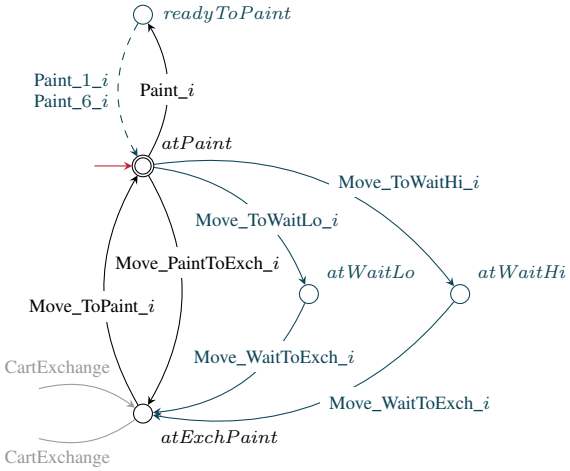


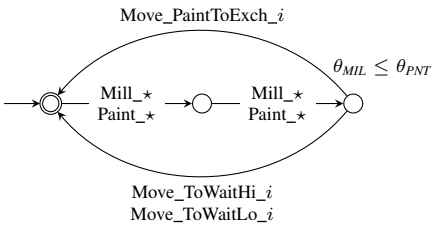
Fig. 8. Refinement of the cart plant, showing only the refined Paint stage. The remainder of the cart plant is unchanged from the nominal plant in Fig. 6. New locations and edges compared to the nominal plant are marked blue.

1) Plant and activities: The cart plant is modified on the Paint side to allow for the new system behavior. The refined Paint side of the cart automaton is shown in Fig. 8. We model the first refinement, the uncontrollable choice of a Paint_1 or Paint_6 operation, by splitting the paint operation into controllable and uncontrollable parts. Controllable activity Paint_i starts the operation and is followed by one of two uncontrollable activities, Paint_1_i or Paint_6_i, which finish the operation. This way, the controller may choose when to start a paint operation, but has no control over which paint operation is subsequently executed. To reflect this, activity Paint_i contains no actions so that it does not affect the resource availability times. Activities Paint_1_i and Paint_6_i contain the actions which constitute the paint operation, where Paint_6_i paints more holes and thus has a larger transit time.

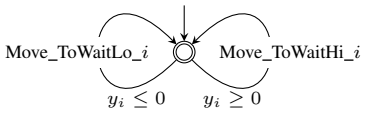
For the second refinement, relating to the cart exchange entry procedure, two wait positions are added as locations to the plant, as also seen in Fig. 8. From the paint location, the plant now allows a move immediately to the exchange disc, a move to the high wait position, or a move to the low wait position. We model the move from either wait location to the exchange disc using a single activity, as the wait locations are located at equal distance from the exchange disc and either move therefore has the same duration. The newly introduced activities can all be found in Appendix A; the full cart plant automaton of the refined system can be found in Appendix B.

2) Variables and updates: Since the system contains five resources, there are five $(max, +)$ variables available with timed information. We denote these by θ_{MIL}, θ_{PNT}, θ_{CR1}, θ_{CR2} and θ_{LR} according to the respective resource identifiers. The update of $(max, +)$ variables after a transition is given by the $(max, +)$ matrix of the corresponding activity, as explained in Section III-B.

We add two additional variables which represent the real-valued y-coordinates of cart 1 and cart 2, which we denote by y_1 and y_2. As the system is modeled using a finite number of positions for the carts, these variables can taken on a finite number of values. For their update functions, we consider the



(a) Paint side timed choice requirement with timed guard.



(b) Paint side wait location requirement with guards on the position of cart i.

Fig. 9. Refinement requirements.

activities on the Paint side which affect the y-coordinate of a cart. As the new y-coordinate is determined solely by the execution of an activity, we apply the corresponding value as a fixed assignment to each transition labeled with the activity. We assume the following values in cm: for Paint_1_i, $y_i = 0.0$; for Paint_6_i, $y_i = -0.5$; for Move_ToWaitHi_i, $y_i = 3.0$; for Move_ToWaitLo_i, $y_i = -3.0$; for all activities ending with ToExch_i, which move a cart to the exchange disc, $y_i = 0.0$.

3) Requirements and guards: We specify two new requirements to ensure the system behaves according to the new exchange disc entry procedure. The first requirement, shown in Fig. 9a, compares the clock values of the Mill and Paint stations, encoded in θ_{MIL} and θ_{PNT}, to determine which operation finishes first. If the paint operation finishes before the mill operation, a direct move to the Paint side of the exchange disc is disabled, leaving only a move via one of the wait positions. If it finishes later than the Mill operation, no preference for a move is specified. In Section V-F the optimal choice for this move is determined.

To ensure that the clock values relate to the current cycle, the comparison is preceded by two Paint or Mill operations, in arbitrary order, before the clock comparison is made. This additional ordering of activities does not affect timed system behavior, as the mill and paint activities are executed in parallel on separate resources. Requiring the mill activity to take precedence does therefore not delay a subsequent move on the Paint side.

In this case, the mill station and paint station resources are used only once during a cycle. We may therefore access their clocks at any later time during the cycle and consistently receive the time at which their operation finished. In general, this is not the case. For example, the cart resources are used during several activities, and the meaning of the value of their clocks for use in requirements can quickly become ambiguous. In these cases, it is useful to introduce a new resource, which we call a *virtual resource*, which can be claimed and released during an activity where the availability time of a resource is of interest. As the clock of the virtual resource is decoupled from the original resource, we can access its value at a later time, even if the clock of the original resource changes with subsequent activities. This allows for concise and meaningful requirements using clock values.

The second requirement, shown in Fig. 9b, ensures that a cart moves to the correct wait position depending on the value of its y-coordinate. This requirement specifies only which wait position is allowed; it does not specify if a move to a wait position is at all enabled. As the move to a wait position is only available after a paint operation, the value of the y-coordinate is determined by the end position of the Paint_1 and Paint_6 activities. For the values of y given earlier, this means that after activity Paint_6, a cart may only move to the lower waiting position, whereas after activity Paint_1, a cart may move to either wait position. Note that we could have alternatively modeled this behavior explicitly using event-based requirements, by disabling the move to a disallowed wait position after the Paint_1 or the Paint_6 activity. However, we would have to add new requirements each time we wanted to add a paint operation. We would also need to remember to update these requirements if any of the y-coordinate parameters changed. Using the guard-based requirement, safe functionality is guaranteed for any paint operation or parameter value as long as updates of variables y_i are correctly specified, which can be easily automated in the implementation of a domain-specific language (DSL).

Finally, the existing requirements must be updated for the newly introduced plant behavior. In the requirement shown in Fig. 7b, we replace activity Move_PaintToExch_⋆ by three activities Move_PaintToExch_⋆, Move_ToWaitLo_⋆ and Move_ToWaitHi_⋆. In the requirement shown in Fig. 7f, the two transitions labeled with Move_PaintToExch_⋆ are replaced by transitions labeled with two activities Move_PaintToExch_⋆ and Move_WaitToExch_⋆. The full set of requirement automata of the refined system can be found in Appendix B.

E. Constructing the Activity-EFA

Given the components, we construct the Activity-EFA in two steps. We start by computing an intermediary FSM which contains all allowed activity sequences *without considering variables*. For this, we use supervisory controller synthesis [6] on the plants and requirements which have been stripped of guards and updates. The synthesized FSM acts as an intermediary which we can augment to form the Activity-EFA. We choose for this two-step approach as the interleaving of states and variable values can quickly lead to state-space explosion, leading to issues with scalability of the synthesis algorithm [10], [17]. We add back the guards and updates to the transitions of the synthesized FSM to construct the Activity-EFA.

To compute the intermediary FSM, we strip guards and updates from transitions in the plants and requirements (Figures 6 to 9) before synthesis. The use of synthesis guarantees that

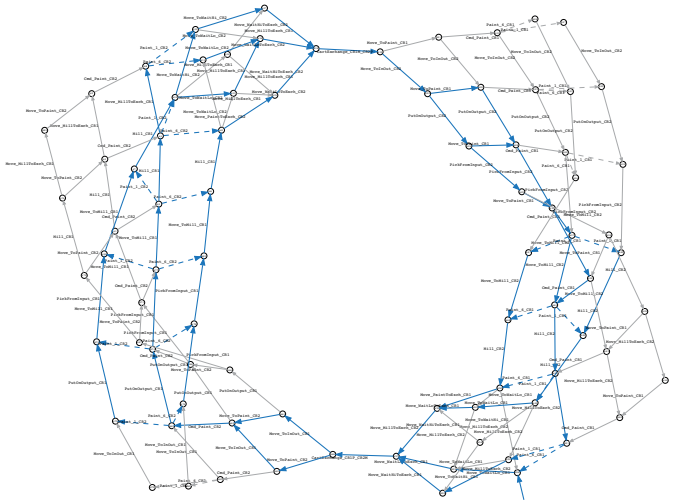


Fig. 10. Automaton representation of the constructed Activity-EFA (81 locations and 146 transitions). This representation focusses on the activity sequences and omits guards and updates. Activity sequences consistent with a cycle of the robust optimal controller are marked blue.

functional requirements are respected and that the synthesized system is *controllable* and *nonblocking*. Controllability ensures that the synthesized FSM does not disable uncontrollable events from happening that could have occurred in the uncontrolled plants. Nonblockingness acts as a guarantee to system progress, as there is an allowed activity sequence from every reachable state to a marked (safe) state. The intermediary FSM is more permissive compared to the modeled requirements, since guards that were specified on requirements have been disregarded. We must therefore add these components back.

The guards in Fig. 9 are specified on transitions of modular automata, which were also used (without the guards) to synthesize the intermediary FSM. It is not trivial to manually find the corresponding transitions in this intermediary system. Since we use multiparty synchronization, we can compute the synchronous product [16] of the synthesized FSM and the automata containing guards, which automatically adds the guards to the corresponding transitions.

Updates in this example are defined on activities, which facilitates application to the Activity-EFA. This is done by applying each update to the transitions labeled with the corresponding activity. In case updates should be defined on transitions rather than activities, we can again compute a synchronous product to find matching transitions. In that case, care should be taken as multiple updates may attempt to set different values to the same variable, for which several solutions can be found in the literature [9], [10].

F. Performance analysis

Having constructed the Activity-EFA in two steps, we compute the normalized configuration space and use the methods of Section IV to compute a robust throughput-optimal controller. The Activity-EFA is shown in Fig. 10 with the optimal control strategy marked blue. Each uncontrollable transition from a location in the controller is also in the controller, and

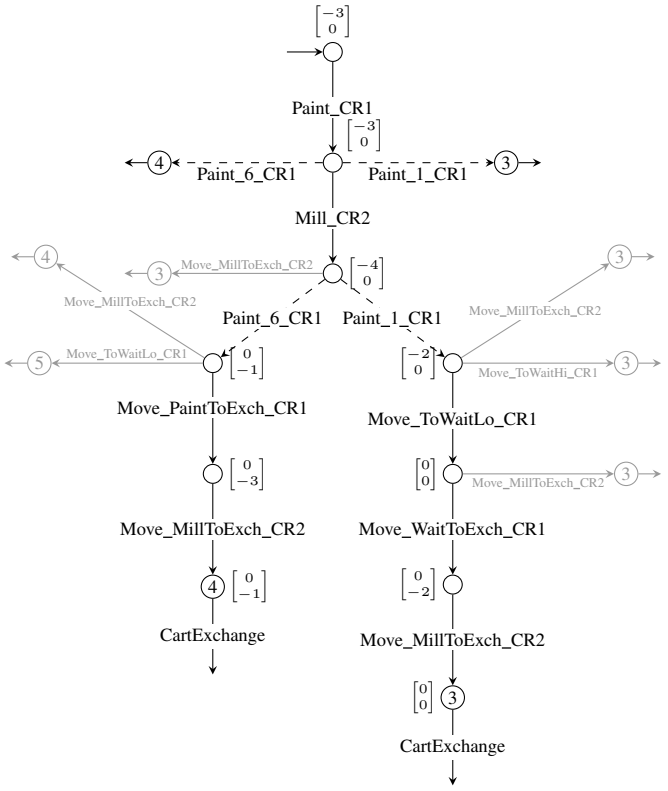


Fig. 11. Subautomaton of the computed controller showing activity sequences leading up to a cart exchange, overlayed on the normalized configuration space (shown in grey). The vector next to a configuration denotes the timing values of carts 1 and 2, $[\theta_{CR1}, \theta_{CR2}]^\intercal$. The number in each final node denotes the shortest transit time of a path through that node from the indicated initial configuration to the cart exchange.

after each uncontrollable transition, there is an optimal control strategy.

To illustrate robustness of the controller to uncontrollable behavior, consider the choice on the Paint side to move either directly, or via a wait position to the exchange disc. This choice impacts performance in case the Paint side operation finishes later than the Mill side operation. For this example, we choose appropriate timing functions T of the actions, such that a) Paint_1_i finishes 2 [s] before Mill_i, b) Paint_6_i finishes 1 [s] later than Mill_i, and c) a move to the exchange disc via a wait position takes 2 [s] longer than a move directly onto the exchange disc.

The activity sequences leading up to a cart exchange on the throughput-optimal cycle of the controller are shown in Fig. 11. Also shown are alternative sequences which were allowed in the configuration space before optimization. We start from a configuration where cart 1 is on the paint station and cart 2 is on the mill station. For both uncontrollable activities Paint_1_CR1 and Paint_6_CR1, an optimal path is determined by the controller. In case activity Paint_6_CR1 is executed, a direct move onto the exchange disc from the paint side results in a faster sequence. In case activity Paint_1_CR1 is executed, there is no performance gain from choosing a move via either of the wait positions, or a direct move.

VI. Related Work

Timed Petri nets are a commonly encountered modeling mechanism for manufacturing systems, which also allow for uncontrollable transitions [18]. Performance analysis of timed Petri nets can be achieved by associating delay bounds with each place in the net [19]. Petri nets are generally well suited for the design and analysis of systems operating on multiple resources, as they can naturally express concurrent execution of system behavior. Studies which consider uncontrollable transitions generally focus on finding minimally-restrictive controllers [20] or deadlock-avoidance [21], rather than timed performance. Recently, an effort was made in [22] to incrementally compute an optimal control sequence using timed Petri nets, with knowledge of the probability of occurrence of some uncontrollable events. This makes it suitable for a real-time control solution of a known system, but not as much for performance analysis in the design phase. An alternative approach which considers both uncontrollability and timed performance is proposed in [23]. Here, uncontrollability is replaced by a control cost of an event, and optimization aims to minimize control cost and cycle time of the closed loop net simultaneously.

Other approaches for specification or performance analysis of partially-controllable systems include the use of probabilistic automata and stochastic scheduling. In [24], probabilistic automata are combined with extended finite automata to find uncontrollable alternative working sequences with corresponding occurrence probabilities. A version of the A* search algorithm [25] is used to find the shortest cycle time. Stochastic scheduling [26] considers uncertainty only in the duration of activities by assuming stochastic operation times. Controller synthesis methods can be found [27] to compute a controller with an expected optimal performance. What these methods have in common is the assumption of a priori knowledge of the probabilities of duration or occurrence of uncontrollable system behavior.

Game theory applied to timed automata [28] has been used for the synthesis of a controller which ensures safe behavior, as well reachability of a final (goal) state. Timed automata have also been used in [29] to solve a scheduling problem using shortest path algorithms. Using timed automata poses challenges with respect to scaling for large systems, as timing of individual actions must be considered during analysis. This is avoided by the use of $(max, +)$ timing characterizations of activities.

VII. Conclusions

This paper introduces a new approach to model uncontrollability of functionality in manufacturing systems. It is based on the recently developed Activity modeling formalism [4], which is used to specify the functionality and timing of manufacturing systems at a high level of abstraction using deterministic activities. The Activity formalism allows for the scheduling of desirable sequences of activities on multiple resources by means of $(max, +)$ algebra. Our approach extends the specification by the introduction of uncontrollability at the level of activities, as commonly found in a supervisory control context [7]. Functional dependency at activity level on feedback from lower abstraction levels is realized by converting the formal specification into an Activity-EFA. The Activity-EFA contains the allowed activity sequences along with guard expressions which limit behavior depending on the values of variables. We introduce a new performance analysis method which employs game-theoretic methods to provide a guarantee to system performance, and to automatically compute a robust, throughput-optimal controller given a partially controllable system. The methods are illustrated on an example manufacturing system.

There are a number of directions in which our approach can be extended. First, the use of extended finite automata could be broadened to include the initial controller design, which currently uses regular automata. By being able to refer to variables and states of (other) automata, EFAs typically allow a more concise specification [10], [30]. It may be possible to devise a combined approach with the variables, guards and updates which we add at a later stage in our approach. Second, we want to investigate if data-based synthesis techniques [9] can be used in the construction of the EFA. This would provide automatic guarantees on important supervisory control properties, such as *controllability* and *nonblockingness*. Third, there are various other performance analysis and optimization techniques that can be investigated which cope with uncontrollability in different ways. For batch production systems, the use of finite two-player games and different performance metrics, such as makespan, could be investigated. When the probability of occurrence of uncontrollable behavior is known, stochastic scheduling methods [26] can be employed which may further optimize performance. Finally, we want to investigate the application of our approach on an industrial-size case. The approach we present can easily be adopted in a domain-specific language which already uses the semantics of the Activity formalism.

References

[1] J. A. Estefan, "Survey of model-based systems engineering (mbse) methodologies," INCOSE MBSE Initiative, Tech. Rep., 2008.

[2] C. Steimer, J. Fischer, and J. C. Aurich, "Model-based design process for the early phases of manufacturing system planning using sysml," *Procedia CIRP*, vol. 60, pp. 163–168, 2017, ISSN: 22128271. DOI: 10.1016/j.procir.2017.01.036. [Online]. Available: http://dx.doi.org/10.1016/j.procir.2017.01.036.

[3] J. C. M. Baeten, J. M. van de Mortel-Fronczak, and J. E. Rooda, "Integration of supervisory control synthesis in model-based systems engineering," *Proc. of Special Int. Conf. on Complex Systems: Synergy, of Control, Communications and Computing*, pp. 167–178, 2011.

[4] B. Van Der Sanden, J. Bastos, J. Voeten, M. Geilen, M. Reniers, T. Basten, J. Jacobs, and R. Schiffelers, "Compositional specification of functionality and timing of manufacturing systems," in *2016 Forum on Specification and Design Languages (FDL)*, Bremen, Germany, 2016.

[5] A. Dasdan, "Experimental analysis of the fastest optimum cycle ratio and mean algorithms," *ACM Transactions on Design Automation of Electronic Systems*, vol. 9, no. 4, pp. 385–418, 2004, ISSN: 10844309. DOI: 10.1145/1027084.1027085.

[6] P. J. Ramadge and W. M. Wonham, "Supervisory control of a class of discrete event processes," *SIAM Journal on Control and Optimization*, vol. 25, no. 1, pp. 206–230, 1987, ISSN: 0363-0129. DOI: 10.1137/0325013. [Online]. Available: http://epubs.siam.org/doi/10.1137/0325013.

[7] C. G. Cassandras and S. Lafortune, *Introduction to Discrete Event Systems*. 2008, vol. 11, p. 776, ISBN: 9780387333328. [Online]. Available: http://www.springer.com/engineering/robotics/book/978-0-387-33332-8.

[8] B. Van Der Sanden, M. Geilen, M. Reniers, and T. Basten, "Solving ratio games : algorithms and experimental evaluation," *Submitted*, 2016.

[9] Y.-L. Chen and F. Lin, "Modeling of discrete event systems using finite state machines with parameters," *Proceedings of the IEEE International Conference on Control Applications, CCA'00*, vol. 2, no. 805, pp. 941–946, 2000. DOI: 10.1109/CCA.2000.897591.

[10] M. Skoldstam, K. Akesson, and M. Fabian, "Modeling of discrete event systems using finite automata with variables," *Decision and Control, 2007 ...*, pp. 3387–3392, 2007. DOI: 10.1109/CDC.2007.4434894. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs%7B%5C_%7Dall.jsp?arnumber=4434894.

[11] M. Geilen, "Synchronous dataflow scenarios," *ACM Transactions on Embedded Computing Systems*, vol. 10, no. 2, pp. 1–31, 2010, ISSN: 15399087. DOI: 10.1145/1880050.1880052.

[12] R. Bloem, K. Greimel, T. A. Henzinger, and B. Jobstmann, "Synthesizing robust systems," *Formal Methods in Computer-Aided Design*, pp. 85–92, 2009.

[13] A. Voronov and K. Åkesson, "Verification of supervisory control properties of finite automata extended with variables," 2009, ISSN: 1403-266x. [Online]. Available: http://publications.lib.chalmers.se/records/fulltext/94442.pdf.

[14] L. Brim, J. Chaloupka, L. Doyen, R. Gentilini, and J. F. Raskin, "Faster algorithms for mean-payoff games," *Formal Methods in System Design*, vol. 38, no. 2, pp. 97–118, 2011, ISSN: 09259856. DOI: 10.1007/s10703-010-0105-x. arXiv: arXiv:1706.06139v1.

[15] M. Geilen and S. Stuijk, "Worst-case performance analysis of synchronous dataflow scenarios," *2010 IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, no. C, pp. 125–134, 2010. DOI: 10.1145/1878961.1878985.

[16] W. Wonham, "Supervisory control of discrete-event systems," *Notes*, p. 444, 2010, ISSN: 08957177. DOI: 10.1007/0-8176-4404-0_4. [Online]. Available: http://www.springerlink.com/content/p2763t58vn4523h3/.

[17] C. De Oliveira, J. E. R. Cury, and C. A. A. Kaestner, *Synthesis of supervisors for parameterized and infinity non-regular discrete event systems*, PART 1. IFAC, 2007, vol. 1, pp. 181–186, ISBN: 9783902661395. DOI: 10.3182/20070613-3-FR-4909.00033. [Online]. Available: http://dx.doi.org/10.3182/20070613-3-FR-4909.00033.

[18] J. Wang, *Timed Petri Nets: Theory and Application*, ser. The International Series on Discrete Event Dynamic Systems. Springer US, 1998, ISBN: 9780792382706.

[19] H. Hulgaard and S. M. Burns, "Efficient timing analysis of a class of petri nets," in *Computer Aided Verification*, P. Wolper, Ed., Berlin, Heidelberg: Springer Berlin Heidelberg, 1995, pp. 423–436, ISBN: 978-3-540-49413-3.

[20] J. O. Moody and P. J. Antsaklis, "Petri net supervisors for des with uncontrollable and unobservable transitions," *IEEE Transactions on Automatic Control*, vol. 45, no. 3, pp. 462–476, 2000, ISSN: 00189286. DOI: 10.1109/9.847725.

[21] A. Aybar and A. Iftar, "Deadlock avoidance controller design for timed petri nets using stretching," *IEEE Systems Journal*, vol. 2, no. 2, pp. 178–188, 2008, ISSN: 19379234. DOI: 10.1109/JSYST.2008.923193.

[22] D. Lefebvre, "Dynamical scheduling and robust control in uncertain environments with petri nets for dess," *Processes*, vol. 5, no. 4, p. 54, 2017, ISSN: 2227-9717. DOI: 10.3390/pr5040054. [Online]. Available: http://www.mdpi.com/2227-9717/5/4/54.

[23] F. Basile, P. Chiacchio, A. Giua, and I. Elettrica, "Optimal petri net monitor design," in *Synthesis and Control of Discrete Event Systems*, Kluwer, 2001, pp. 141–154.

[24] A. Kobetski, J. Richardsson, K. Åkesson, and M. Fabian, "Minimization of expected cycle time in manufacturing cells with uncontrollable behavior," *Proceedings of the 3rd IEEE International Conference on Automation Science and Engineering, IEEE CASE 2007*, pp. 14–19, 2007. DOI: 10.1109/COASE.2007.4341802.

[25] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *Systems Science and Cybernetics*, no. 2, pp. 100–107, 1968.

[26] J. Niño-Mora, "Stochastic scheduling," in *Encyclopedia of Optimization*, Springer US, 2008, pp. 3818–3824.

[27] J. F. Kempf, M. Bozga, and O. Maler, "As soon as probable: optimal scheduling under stochastic uncertainty," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 7795 LNCS, pp. 385–400, 2013, ISSN: 03029743. DOI: 10.1007/978-3-642-36742-7_27.

[28] G. Behrmann, A. Cougnard, A. David, E. Fleury, K. Guldstrand Larsen, and D. Lime, "Uppaal-tiga : time for playing games!," 2009. DOI: 10.1007/978-3-540-73368-3.

[29] Y. Abdeddaïm, E. Asarin, and O. Maler, "Scheduling with timed automata," *Theoretical Computer Science*, vol. 354, no. 2, pp. 272–300, 2006, ISSN: 03043975. DOI: 10.1016/j.tcs.2005.11.018.

[30] B. Van Der Sanden, M. Reniers, M. Geilen, T. Basten, J. Jacobs, J. Voeten, and R. Schiffelers, "Modular model-based supervisory controller design for wafer logistics in lithography machines," in *2015 ACM/IEEE 18th International Conference on Model Driven Engineering Languages and Systems, MODELS 2015 - Proceedings*, 2015, pp. 416–425, ISBN: 9781467369084. DOI: 10.1109/MODELS.2015.7338273.

APPENDIX A
ACTIVITIES OF THE DICE FACTORY SYSTEM

The activity structures listed here are defined using cart 1 (CR1). The activities of cart 2 (CR2) are defined equivalently.

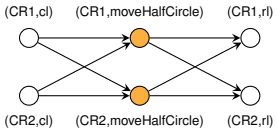


Fig. 12. Activity CartExchange.

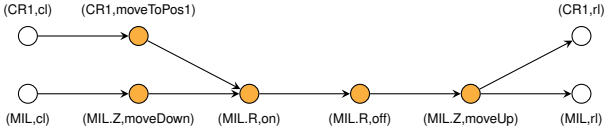


Fig. 13. Activity Mill_CR1.

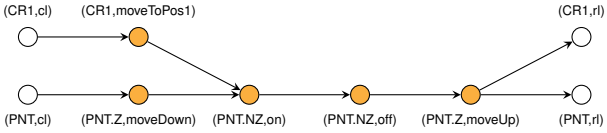


Fig. 14. Activity Paint_CR1 (nominal system).

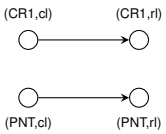


Fig. 15. Activity Paint_CR1 (refined system).

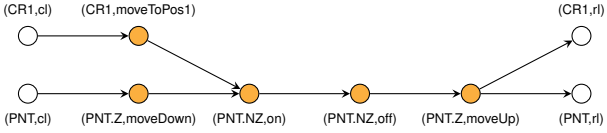


Fig. 16. Activity Paint_1_CR1 (refined system).

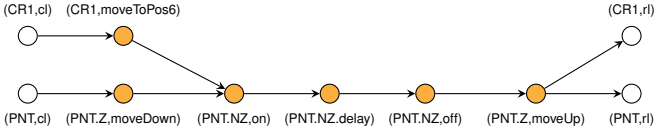


Fig. 17. Activity Paint_6_CR1 (refined system).

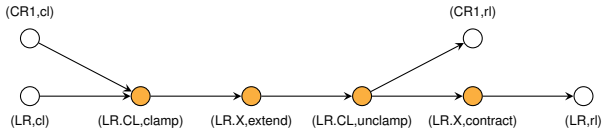


Fig. 18. Activity PickFromInput_CR1.

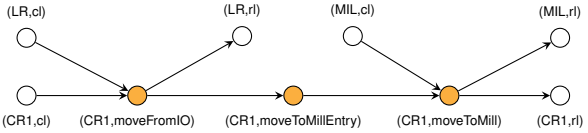


Fig. 19. Activity Move_ToMill_CR1.

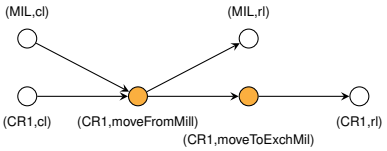


Fig. 20. Activity Move_MillToExch_CR1.

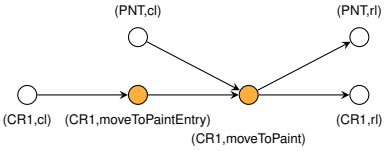


Fig. 21. Activity Move_ToPaint_CR1.

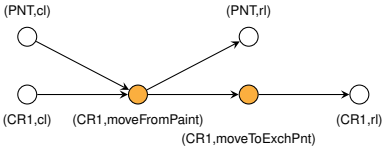


Fig. 22. Activity Move_PaintToExch_CR1.

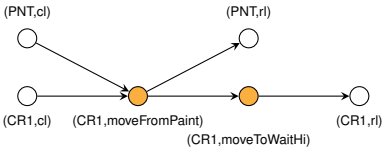


Fig. 23. Activity Move_ToWaitHi_CR1 (refined system).

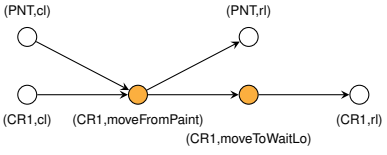


Fig. 24. Activity Move_ToWaitLo_CR1 (refined system).

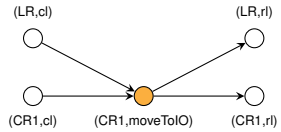


Fig. 25. Activity Move_ToInOut_CR1.

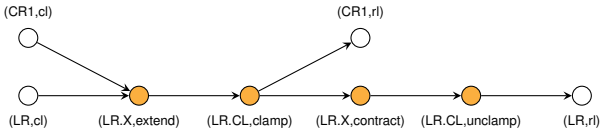


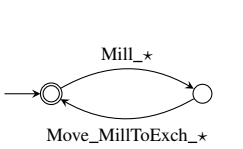
Fig. 26. Activity PutOnOutput_CR1.

APPENDIX B
PLANTS AND REQUIREMENTS OF THE REFINED DICE FACTORY SYSTEM

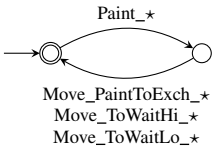
The full set of plants and requirements of the refined Dice Factory are shown in Figures 27 and 28, which include the uncontrollable transitions, guards, and updates. They are the refined versions of the plants and requirements, of Section V-C, with the refinements of Section V-D. We only show updates of ordinary variables on the edges as these are the only updates that are manually defined on the requirements.

APPENDIX C
ASML ONLY - RELATION OF PRESENTED WORK TO ASML USE CASE

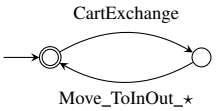
The internal version of this document for ASML will include details of how the presented work can be applied for a use case of ASML, and how the work has been implemented as prototype code in their DSL.



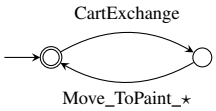
(a) Mill procedure.



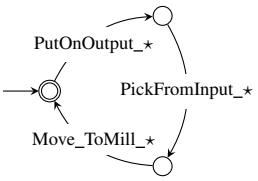
(b) Paint procedure.



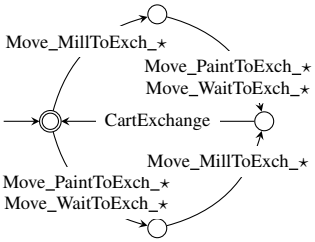
(c) Exchange finish, mill side.



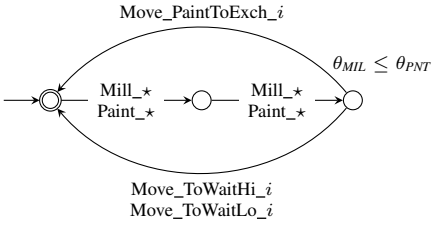
(d) Exchange finish, paint side.



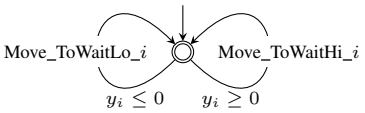
(e) Input-output sequence.



(f) Exchange entry procedure.



(g) Paint side timed choice requirement of cart i.



(h) Paint side wait location requirement of cart i.

Fig. 27. Refined requirement automata which enforce the production life cycle.

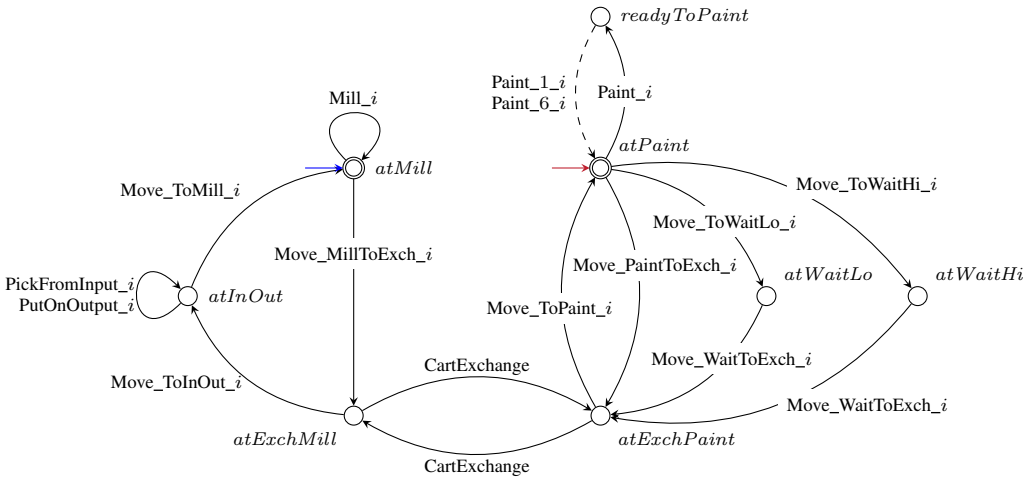


Fig. 28. Refined version of the cart plant of cart i. The initial locations of carts 1 and 2 are marked by a blue and a red arrow, respectively.