A Theory of Sequential Resource Allocation Systems for Automation

Spyros Reveliotis

Abstract— **The integrating role that is sought by automation in many contemporary applications frequently can be abstracted to the need for pertinent supervision of a function that allocates a finite set of reusable resources to a set of concurrently executing processes. This article formalizes the corresponding supervisory control problem, surveys the major results that are currently available for it, and highlights remaining open challenges for the corresponding research community. In the process, it also reveals the analytical and the computational advantages that can be attained from the reference of the considered problem to some appropriate control-theoretic frameworks, and the additional potential that is defined for such an analytic approach by the identification and exploitation of some essential problem features and the "special structure" that is defined by them.**

I. Automation as a Resource Allocation Function: A set of motivating applications

When it comes to automation and its contemporary applications, one of the major challenges facing the modern engineering community is the effective integration of a set of autonomous devices, frequently characterized by distinct, heterogeneous behaviors, into a well-behaved and properly controlled system. In the context of the research program that is presented in this article, the aforementioned integrating task can be perceived as the design and the deployment of a set of control policies that will coordinate the allocation of a finite set of reusable resources to a set of concurrently executing processes. These processes evolve in a staged manner according to some sequential logic that may be predefined, but they can also be impacted by factors that are determined in real-time, during the progression of the considered application. The execution of any single stage of the considered processes requires the exclusive allocation to them of a certain subset of the system resources. Probably the best known manifestation of such an operational environment is the popular concept of the "flexibly automated production system", where a set of numerically controlled (NC) machines is (supposed to be) integrated seamlessly with a set of buffers and material handling devices in order to support the simultaneous production of a set of part types, each produced according to its own process plan (or even a set of process plans in the particular case where the considered production system allows for some "routing flexibility") [20]. In such an environment, every processed part is an instance of a "process type" that is defined by the corresponding process plan (or set of process plans), and the "resources" requested by each stage of this process plan are some buffer space that will physically accommodate the part, the processor(s) or the material handling equipment that will act upon the part at that stage, and possibly additional equipment like fixtures, auxiliary tools, etc.

Even when focusing on the domain of material handling systems (MHS) itself, the effective operation of an automated "unit load, guidepath-based" MHS [51], like an "Automated Guided Vehicle (AGV)" system or an "overhead monorail" system, can be perceived as the control of a resource allocation function of the type described in the previous paragraph. All these MHS consist of (i) a guidepath network that interconnects a set of workstations and additional supporting service facilities, like a docking and a re-charging station, and (ii) a fleet of autonomous vehicles that transfer parts among the system workstations by using the guidepath net and eventually retiring to the docking station. Physical collisions among the vehicles are avoided by splitting the guidepath links into a number of smaller segments called "zones", and requesting that each zone is allocated exclusively to a single vehicle. Accessing a certain zone by a traveling vehicle must be negotiated with the system controller. Hence, under the considered operational regime, the execution of a vehicle trip from an "'origin"-location to some "destination"-location is essentially a "staged process" where each "stage" corresponds to the traversal of a certain zone in the followed path, while the zone itself constitutes the "resource" allocated to the vehicle-abstracting process at that particular stage. A nominal vehicle-abstracting process under the aforementioned operational regime is defined by a sequence of zones taking the corresponding vehicle first from the docking station to the workstation originating the corresponding transport request, subsequently to the destination workstation, and finally back to the docking station; however, the general model also allows for alternative routes, and the re-assignment of a retiring vehicle to a new transport task while being on its trip back to the docking station.

More recently, the resource allocation paradigm for guidepath-based MHS, that was outlined in the previous paragraph, has been extended to traffic systems involving a fleet of free-ranging mobile agents over a certain planar area [45]. In this case, each agent is represented by some canonical geometrical "footprint" – typically a disk of a certain radius – and the entire area is tesselated into a number of "cells". An agent with its footprint overlapping a given cell is considered to "occupy" this cell. Cells possess a nominal "capacity" that limits the maximal number of the agents that can simultaneously occupy the cell in a safe manner. Observance of the cell capacity is enforced by the system controller, which provides to the traveling agents the necessary permission to access a requested cell. In a simple implementation of the considered control scheme, cells possess *unit* capacity, ensuring naturally the physical separation of the traveling agents. However, the literature also avails of more complex control schemes where a cell can have a nominal capacity larger than one, allowing two or more agents to traverse it simultaneously [50]; in such a case, it is further assumed that the cell-sharing agents have sensing, communication and control capability that allows them to negotiate their local

The author is with the School of Industrial & Systems Engineering, Georgia Institute of Technology, email: spyros@isye.gatech.edu.

motion in a safe manner. In [45], the above ideas are further extended to apply to the traffic of mobile agents that takes place in three-dimensional spaces, and with the agents being in perpetual motion from their origin to their destination; hence, the considered control scheme can apply to the traffic coordination of airplanes, drones, and all other flying gadgets that are currently contemplated as future transport devices. Similarly, one can easily envision the potential extension of guidepath-based MHS to driverless monorail and railway systems serving urban or regional transportation needs over complex networks.

The resource allocation paradigm that was outlined in the previous paragraphs for flexibly automated production and material handling systems, mobile agents, and even the public transportation domain, extends to other service sectors through the notion of "automated workflow management system (WMS)" [30], [54]. WMS have been promoted as a structured solution to the automation of routine, yet complex operations that take place in business sectors like insurance claiming, banking and the backend operations of modern logistics systems. According to the WMS paradigm, the aforementioned operations are abstracted into a set of process types that execute in a staged manner and utilize critical resources like data files, various sorts of processors and communicating devices, and even the company personnel in various supporting roles. The enactment of these process types is facilitated by a "workflow management engine" that monitors the progress of the different running processes, and coordinates their overall flow and the allocation of the resources requested by them in an expedient and orderly manner.

In fact, it can be safely argued that some of the very first fully automated WMS developed by our modern technological civilization, were the operating systems of all the past computational platforms that have supported multi-threaded software. All these operating systems coordinate the allocation of various system resources like memory registers, (access to) data files, I/O devices, etc., to the concurrently running "threads" through the representation of these limited resources by some sort of "tokens", typically known as "locks" or "semaphores", that are carefully granted to the contesting processes [10]. Presently, the interest of the computing world in this resource allocation function has been revitalized thanks to the advent of "multicore" computer architectures, that have turned parallel programming into a commodity [57].

Finally, a further application of the considered resource allocation paradigm in the world of modern computing comes from the area of quantum computing [38]. In this domain, the primitive elements of information, known as "qubits", are stored in the form of ionized atoms, and various elementary operations on these qubits are performed through their physical transport to certain locations, called "ion traps", where they interact in a controlled manner. The physical transport of qubits from their initial storage locations to ion traps takes place through a network of "tunnels" that must be exclusively allocated to each traveling qubit in order to avoid "collisions" that would destroy their information content. Hence, the physical realization of an algorithmic logic unit (ALU) for a quantum computer bears very strong similarities to the operation of the guidepath-based transport systems that were discussed in the previous paragraphs, and it must address a similar set of resource allocation problems.

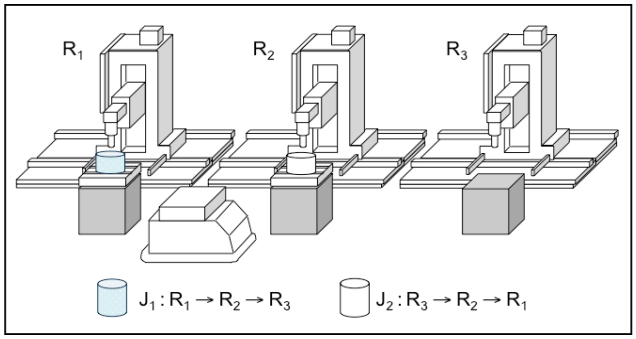

Fig. 1. This figure depicts a flexibly automated production cell consisting of three workstations R_1, R_2 and R_3, and a single AGV that transports parts among these three workstations and the I/O port of the cell. Each workstation avails of a worktable that can hold only one workpiece at a time. The cell is used for the concurrent production of the two process types J_1 and J_2 with the corresponding process plans that are annotated in the figure. It should be evident from the above description of the cell operations and the provided routing information for the two process types, that in the depicted situation, none of the currently loaded parts is able to advance to its next requested workstation, since the corresponding worktable is occupied by the other part. This is a typical situation of *deadlock* that can be encountered in such flexibly automated production systems.

II. A NEW SET OF CONTROL CHALLENGES AND THE EMERGING CONTROL PARADIGM OF SEQUENTIAL RESOURCE ALLOCATION SYSTEMS

In all the examples that were presented in the previous section, the penultimate objective for the management of the corresponding resource allocation function is the optimization of some notion of "performance" for the underlying system. Frequently, this performance is defined and measured by the production rate – or "throughput" – of the considered system, while additional considerations might involve the control of the congestion experienced by the system processes; the latter is defined in terms of the delays that are experienced by the various processes, or the process concentrations that are observed at the various processing stages. The resulting performance analysis and control problems have been studied in the past through operational models and theories pursued by the fields of operations research (OR), industrial engineering (IE) and operations management (OM). However, the need to support the considered performance objectives in the automated setting that is implied by all the previously provided examples gives rise to additional operational problems that have not been modeled and/or studied by the aforementioned disciplines.

Among these new problems, one of the most pernicious is defined by the need to establish "deadlock-free" – or "nonblocking" – operation for the underlying resource allocation function. The notion of "deadlock" is exemplified in Fig. 1, where the two depicted process instances are permanently stuck, since each of them is requesting for its further advancement the buffer space that is currently held by the other process.[1] Unless there is external interference, no further progress can be achieved by the de-

[1]A more formal definition of the notion of "deadlock" will be provided in a subsequent part of this article.

picted processes, while the effective utilization of the resources allocated to them will be equal to zero. Hence, the formation of deadlocks is utterly disruptive to the operation and the performance of the automated applications that we are considering in this work.

In the considered operational settings, circular waiting patterns among the running processes, like those depicted in Fig. 1, arise due to (a) the exclusive nature of the allocation of the finite system resources to the various process instances, (b) the ability of the running process instances to hold upon their currently allocated resources while waiting for some further resources needed for the execution of their next processing stage, and (c) the arbitrariness of the resource allocation requests that can be posed by the different processing stages. Obviously, these conditions are also present in the operation of the more conventional, non-automated versions of the aforementioned applications, like the production, material handling and service operations that are studied by the OR, IE and OM disciplines. However, in those cases, the presence of the human element in an operating or supervising role enables the resolution of any emergent deadlock through some real-time improvisation.[2] Hence, in non-automated settings, there is no (strongly) felt need to address the deadlocking problem at a formal level. On the other hand, the ubiquitous and highly disruptive nature of this problem was strongly manifested in the 1990's, where some attempts of that era to materialize the notions of the "flexibly automated manufacturing cell" and of the "lights-out factory", based on *ad hoc* integrating schemes, resulted in major fiascos for the involved companies. Furthermore, because of those past negative experiences, almost all current attempts to employ large-scale automation in the production and service sectors have sought to address the behavioral problem of deadlock formation at the design level, by adopting very simple structural designs, and by complementing these structural designs with some very conservative operational policies that seek to negate the third of the aforementioned conditions for deadlock formation. Fig. 2 exemplifies these designs and policies by presenting a typical topology – or "layout" – for the MHS guidepath network employed by the current semiconductor manufacturing industry. Similar simplifying approaches to the deadlock problem have been pursued even in presumably more sophisticated fields, like the field of multithreaded software and parallel programming. But as remarked in Fig. 2, while being robust w.r.t. the resolution of the deadlock problem itself, the currently pursued approaches also limit substantially the concurrency and the flexibility of the underlying system, and, at the end, they compromise the operational efficiencies and the enhanced performance potential that are typically sought from flexible automation.

It becomes evident from the above discussion that the deployment of automated solutions for the aforementioned applications in a way that provides a robust and efficient operation of the underlying system, can be effectively supported only through the development of a rigorous control paradigm that will enable the formal modeling of the underlying system behavior and of the imposed specifications, and will facilitate the thorough analysis and design of the necessary control policies. At the same time, in order to be practically effective, such a control paradigm must address explicitly the representational and computational complexities of the problems investigated, and eventually effect a systematic trade-off between the computational tractability of the developed methodology and the operational efficiency of the obtained solutions.

[2]This sort of improvisation is part of the notorious "firefighting" that takes place in these environments.

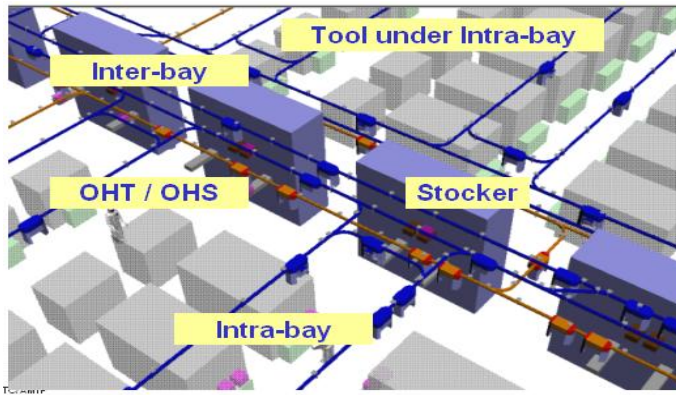

Fig. 2. This figure, taken from [53], depicts the material handling layout – usually known as the "spine" layout [58] – that is used in contemporary semiconductor fabs. This material handling system is an overhead monorail system with its guidepath network decomposed into a set of *unidirectional* loops: one loop interconnecting the processing tools of each "bay" of the fab (i.e., the blue "intra-bay" loops depicted in the figure), and a central loop that acts as the "spine" of the facility and supports wafer transfers among the fab bays (i.e., the brown "inter-bay" loop in the above figure). Intra-bay loops are interfaced to the inter-bay loop through buffering facilities that are known as "stockers". By maintaining a unidirectional vehicle motion on each loop, the considered layout eliminates the potential for deadlock formation among the traveling vehicles. But, at the same time, vehicles tend to travel much longer distances for any single requested transfer, they tend to file up behind the slowest vehicle, and the inter-bay traffic might involve considerable "double-handling" of the transported wafer cassettes at the intermediate stockers.

The rest of this article outlines such a control paradigm that is built upon the formal abstraction of a *"(sequential) resource allocation system (RAS)"* [47]. We provide a formal characterization and a taxonomy of the RAS concept, outline a control paradigm that can be defined by it while leveraging and extending existing results from various areas of modern control theory, and subsequently we focus on the particular problem of *"deadlock avoidance"* – or *"nonblocking supervision"* – for the considered RAS. For this last problem, we provide a formal characterization by means of the supervisory control theory (SCT) of discrete event systems (DES) [7], and establish a notion of optimality for the derived solutions in the form of "maximal permissiveness". On the other hand, formal complexity analysis reveals that the computation of the maximally permissive nonblocking supervisor is an NP-Hard task for most RAS instantiations. Hence, a considerable part of the article is dedicated to the endeavors of our group and the broader research community to cope with this negative result. The article concludes with some discussion of remaining open challenges w.r.t. the RAS supervisory control problem, and of all the additional issues that must be effectively addressed for the complete development of the presented RAS theory, its migration to the engineering practice, and its effective integration to the relevant engineering curricula. Collectively, the presented developments epitomize the corresponding endeavors by the author, his collaborators, as well as a broader group within the relevant research community over a timespan of more than 20 years. They also reveal how some

important challenges faced in the area of automation can benefit from, but also extend and promote, foundational disciplines like those of control engineering, operations research and theoretical computer science.

III. The RAS modeling abstraction, the corresponding control paradigm and a RAS taxonomy

A. The RAS modeling abstraction

As stated in the closing part of the previous section, the primary modeling abstraction that enables a unifying treatment of the real-time operations taking place in all the applications described in the opening section, is the sequential RAS. Following [47], a sequential RAS can be formally defined by a quintuple $\Phi = \langle \mathcal{R}, C, \mathcal{P}, \mathcal{A}, \mathcal{D} \rangle$, where: (i) $\mathcal{R} = \{R_1, \ldots, R_m\}$ is the set of the system *resource types*. (ii) $C : \mathcal{R} \rightarrow \mathbb{Z}^+$ – the set of strictly positive integers – is the system *capacity* function, characterizing the number of identical units from each resource type available in the system. Resources are assumed to be *reusable*, i.e., each allocation cycle does not affect their functional status or subsequent availability, and therefore, $C(R_i) \equiv C_i$ constitutes a system *invariant* for each i. (iii) $\mathcal{P} = \{\Pi_1, \ldots, \Pi_n\}$ denotes the set of the system *process types* supported by the considered system configuration. Each process type Π_j is a composite element itself, in particular, $\Pi_j = < \Delta_j, \mathcal{G}_j >$, where: (a) $\Delta_j = \{\Xi_{j1}, \ldots, \Xi_{j,l_j}\}$ denotes the set of *processing stages* involved in the definition of process type Π_j, and (b) $\mathcal{G}_j$ is an additional data structure that encodes the sequential logic that integrates the set of the processing stages Δ_j into a set of potential process flows. (iv) $\mathcal{A} : \Delta \rightarrow \prod_{i=1}^{m} \{0, \ldots, C_i\}$ is the *resource allocation function* associating every processing stage Ξ_{jk} with the *resource allocation vector* $\mathcal{A}(\Xi_{jk}) \neq \mathbf{0}$ required for its execution. (v) $\mathcal{D}$ is a function mapping each processing stage Ξ_{jk} in $\Delta \equiv \bigcup_{j=1}^{n} \Delta_j$ to a distribution with positive support that characterizes the *"processing time"* of the corresponding processing stage. Finally, we also set $\xi \equiv |\Delta|$, and for purposes of complexity considerations, we define the *size* $|\Phi|$ of RAS Φ by $|\Phi| \equiv |\mathcal{R}| + \xi + \sum_{i=1}^{m} C_i$.

At any point in time, the system contains a certain number of (possibly zero) instances of each process type that execute one of the corresponding processing stages; this distribution of the active process instances across the various processing stages defines a notion of *"state"* for the considered RAS. Obviously, this RAS state must respect the resource capacities; i.e., no resource type $R_i \in \mathcal{R}$ can be over-allocated w.r.t. its capacity C_i at any point in time. Furthermore, in order to model the "hold-while-waiting" effect that was discussed in the previous section, the adopted resource allocation protocol stipulates that a process instance J_j executing a non-terminal stage Ξ_{jk} and seeking to advance to some next stage $\Xi_{jk'}$, must first be allocated the resource differential $(\mathcal{A}(\Xi_{ik'}) - \mathcal{A}(\Xi_{ik}))^+$ and only then will it release the resource units $|(\mathcal{A}(\Xi_{jk'}) - \mathcal{A}(\Xi_{jk}))^-|$, that are not needed anymore.[3] Then, in the resulting operational context, the *RAS deadlock* can be formally defined as a RAS state containing a set of active process instances, DJ, such that every instance $J_j \in DJ$, in order to advance to any of its next processing stages, requests some resources currently held by some other process instance $J_k \in DJ$.

[3]This assumption is not restrictive since the release of resources that do not adhere to this protocol can be modeled by the insertion of additional processing stages in the underlying process plan.

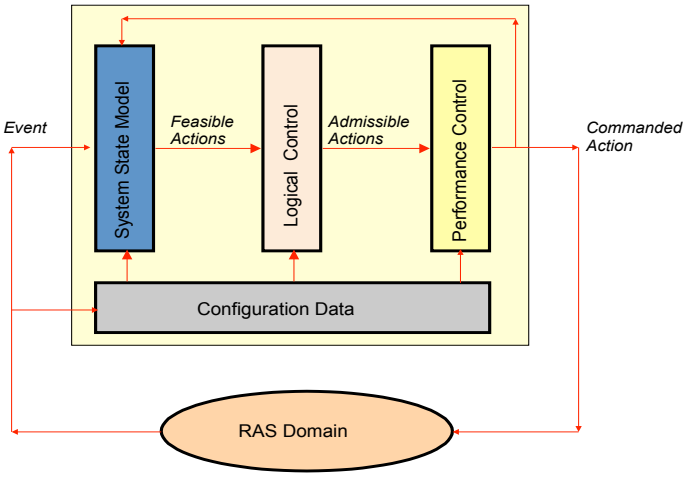

Fig. 3. An event-driven control scheme for the real-time management of the considered RAS. The controller responds to the various events taking place in the controlled RAS by updating a state model that defines the feasible behavior generated by this system. This behavior is "filtered" through the logical controller in order to obtain the admissible behavior, i.e., the behavior that is consistent with certain specifications imposed on the RAS operation. Finally, the admissible behavior is processed through the performance-oriented controller in order to select the particular action(s) among the admissible behavior that eventually will be commanded upon the RAS.

B. A real-time control framework for the considered RAS

As remarked in Section II, an effective real-time controller for the considered RAS must ensure the attainment of some set of performance objectives typically defined w.r.t. the timed RAS behavior, while keeping the RAS away from problematic behavioral patterns like the aforementioned deadlock states. This last control requirement is frequently known as the RAS *behavioral* or *logical* control problem, because the corresponding problematic behavior can be effectively avoided by controlling only the sequencing of the relevant resource allocation events and not their exact timing. Furthermore, it is generally accepted by the relevant research community that, due to the stochasticity that is generally present in the timed dynamics of the considered RAS, any robust solution to the RAS behavioral and performance control problems should rely on some feedback control scheme and not on the open-loop execution of some precomputed plan.

Such a feedback-based controller is presented in Fig. 3. The depicted control paradigm is an *event-driven* approach, where the applied control function monitors the *events* taking place in the underlying RAS and commands a certain *action sequence* in response to these events. More specifically, the proposed controller maintains a representation of the RAS *state*, which enables it to monitor the system status and to identify the entire set of *feasible actions* that can be executed by the system at any given time. Hence, this information is instrumental for enabling the controller to determine the scope of its possible responses to a certain event. However, the controller must eliminate (*"filter out"*) all those actions that can result in problematic behavior.

Such problematic behavior includes the formation of deadlock, but in the more general case, this part of the depicted control scheme will address additional specifications that might be defined, for instance, by quality concerns or some policy considerations, like those arising from a notion of "fairness" to the contesting processes. All the aforementioned concerns boil down to the systematic exclusion of certain resource allocation patterns from the RAS behavior, and, as already mentioned, the resulting problem is generally known as the *logical* or *behavioral* control problem to be addressed by the controller. The set of actions eventually accepted by the logical controller defines the space of the *"admissible"* behavior for the considered RAS. Then, the second stage of the proposed control logic must shape / bias this admissible behavior in a way that aligns best with the system *performance objectives*; this biasing is effectively achieved through the selection of the particular admissible action to be commanded upon the system, at each decision stage. The corresponding problem is known as the RAS *performance-oriented* control or *scheduling*.

The effective deployment of the RAS control scheme that is described in the previous paragraph necessitates a pertinent formal characterization of the RAS state, and the reference of the RAS logical and performance-oriented control problems to appropriate formal modeling frameworks that will enable a rigorous analysis of the corresponding RAS dynamics and the effective synthesis of the necessary policies. At a basic level, these capabilities have been conveniently provided to the developing RAS theory by the areas of *qualitative* and *quantitative analysis of Discrete Event Systems (DES)* [7]. Generally speaking, DES theory is a field of modern control theory investigating the behavior of dynamical systems that evolve their state discontinuously over time, in response to the occurrence of certain critical, instantaneous events. In this general setting, *qualitative DES theory* uses formal linguistic frameworks borrowed from theoretical computer science, augmented with control-theoretic concepts and techniques, in order to analyze and control the event sequences that are generated and observed by the underlying DES dynamics. On the other hand, *quantitative DES theory* analyzes and controls the timed DES dynamics, using models and tools that are borrowed from (stochastic) OR and simulation theory. However, as it will be further revealed in the following, while enabling a formal positioning of the RAS behavioral and scheduling problems, the practical computational capabilities of the corresponding DES frameworks are severely limited by a very high representational and computational complexity. In the subsequent parts of this article we shall demonstrate how the relevant research community has leveraged the representational and analytical capabilities that are provided by DES theory in order to develop effective *practical* solutions to the RAS behavioral control problem. Furthermore, in the closing part we shall also outline some ongoing endeavors towards the development of similar practical solutions for the RAS scheduling problem. All this discussion will also reveal that the considered developments have substantially enriched and extended the capabilities of the corresponding DES theory itself.

C. A RAS taxonomy

We close the discussion on the basic RAS model and the induced control problem, by presenting a RAS taxonomy that has been instrumental for the systematic investigation of the RAS behavioral control problem of deadlock avoidance. The main RAS classes recognized by this taxonomy are defined by (i) the structure that is supported for the process sequential logic, and (ii) the structure of the resource allocation requests that are posed by the various processing stages; the most prominent RAS classes w.r.t. these two classification attributes are presented in Table I. Furthermore, more recent developments have revealed the significance of some additional RAS attributes when it comes to the analytical characterization of the qualitative RAS dynamics and their control for deadlock avoidance. These new attributes include (iii) the absence of resources with *non-unit* capacities, (iv) the presence of *cycling* in the sequential logic of the RAS process types, and (v) the presence of RAS dynamics of an *uncontrollable* nature; this last feature can be further differentiated into (a) uncontrollability w.r.t. the exact timing of a certain resource allocation and (b) uncontrollability of the branching decisions of some underlying process that possesses alternate routings. We shall make extensive use of this taxonomy as we further detail the current theory on the RAS deadlock avoidance problem.

TABLE I
A RAS TAXONOMY [47]

Based on the structure of the Process Sequential Logic	**Based on the structure of the Requirement Vectors**
Linear: Each process is defined by a linear sequence of stages **Disjunctive:** A number of alternative process plans encoded by an acyclic digraph **Merge-Split:** Each process is a fork-join network **Complex:** A combination of the above behaviors	**Single-Unit:** Each stage requires a single unit from a single resource **Single-Type:** Each stage requires an arbitrary number of units, but all from a single resource **Conjunctive:** Stages require different resources at arbitrary levels

IV. MODELING RAS AS AN FSA, THE OPTIMAL NONBLOCKING SUPERVISOR AND ITS COMPLEXITY

A. Modeling the RAS dynamics as a Finite State Automaton

The most straightforward way to formally model the behavioral dynamics of a given RAS Φ for the purpose of deadlock avoidance is by means of a *(deterministic) finite state automaton (FSA)* [7], to be denoted by $G(\Phi)$ [56], [47]. The *state space* S of this automaton consists of all the ξ-dimensional nonnegative integer vectors $\mathbf{s}$ representing RAS states that are feasible w.r.t. the available resource capacities; i.e., each component $\mathbf{s}[l]$ of $\mathbf{s}$ corresponds to a processing stage $\Xi_{jk} \equiv \Xi_l$ of Φ and reports the number of process instances executing this processing stage, while $\mathbf{s}$ satisfies the constraints:[4]

[4]The notation $\mathcal{A}(\Xi_l)$ in Eq. 1 should be interpreted as the resource allocation vector associated with the processing stage that corresponds to the l-th component of the state vector $\mathbf{s}$. We also notice, for completeness, that the finiteness of the state set S can be deduced from Eq. 1 and the non-zero nature of each resource allocation vector $\mathcal{A}(\Xi_l)$.

$$\forall i = 1,\ldots,m, \quad \sum_{l=1}^{\xi} \mathbf{s}[l]\cdot \mathcal{A}(\Xi_l)[i] \;\leq\; C_i. \tag{1}$$

The *event set* E of the FSA $G(\Phi)$ consists of the events corresponding to (i) the initiation (or "loading") of a new process instance, (ii) the advancement of these process instances among their different stages in a manner that is consistent with the sequential logic that defines these processes, and (iii) the eventual termination (or "unloading") of a process instance by its retirement from the system and the release of all the currently held resources. The *transition function* f of the FSA $G(\Phi)$ formalizes the RAS dynamics that are generated by the aforementioned events. Furthermore, f is a partial function since the occurrence of a certain event $e \in E$ at a given state $\mathbf{s} \in S$ will be feasible only if (i) the considered state $\mathbf{s}$ avails of active process instances that can execute the contemplated event e, and (ii) the state $\mathbf{s}'$ that will result from the execution of e in $\mathbf{s}$ satisfies the constraints of Eq. 1; satisfaction of the first of these two conditions is characterized as *"process enablement"* of event e in state $\mathbf{s}$, while satisfaction of the second condition is characterized as *" resource enablement"* of e in $\mathbf{s}$. The *initial state* $\mathbf{s}_0$ of the FSA $G(\Phi)$ is the state $\mathbf{s} = \mathbf{0}$, i.e., the state where Φ is empty of any process instances, and the same state defines also the unique *marked state* of $G(\Phi)$, a fact that expresses the requirement for complete process runs. Finally, in the sequel we shall also use the notation $\hat{f}$ to denote the natural extension of the state transition function f to $S \times E^*$, where E^* denotes the set of all the finite strings of E, including the empty string ε. More specifically, for any state $\mathbf{s} \in S$ and the empty event string ε, $\hat{f}(\mathbf{s},\varepsilon) = \mathbf{s}$, while for any $\mathbf{s} \in S$, $\sigma \in E^*$ and $e \in E$, $\hat{f}(\mathbf{s},\sigma e) = f(\hat{f}(\mathbf{s},\sigma),e)$.[5]

In the context of the modeling framework that is defined by the FSA $G(\Phi)$, the *feasible* behavior of RAS Φ is modeled by the *language* $L(G)$ *generated* by $G(\Phi)$, i.e., by all strings $\sigma \in E^*$ such that $\hat{f}(\mathbf{s}_0,\sigma)$ is defined. On the other hand, the desired – or, more formally, the *admissible* – behavior of Φ is modeled by the *marked language* $L_m(G)$ of the FSA $G(\Phi)$; since the set of marked states of $G(\Phi)$ is the singleton containing the initial state $\mathbf{s}_0$, the marked language $L_m(G)$ consists of exactly those strings $\sigma \in L(G)$ that lead back to the empty state $\mathbf{s}_0$. To facilitate the subsequent discussion, we also define the *reachable subspace* S_r of $G(\Phi)$ by

$$S_r \equiv \{\mathbf{s} \in S : \exists \sigma \in L(G) \text{ s.t. } \hat{f}(\mathbf{s}_0,\sigma) = \mathbf{s}\} \tag{2}$$

and its *co-reachable subspace* S_s by

$$S_s \equiv \{\mathbf{s} \in S : \exists \sigma \in E^* \text{ s.t. } \hat{f}(\mathbf{s},\sigma) = \mathbf{s}_0\}. \tag{3}$$

Furthermore, in the following, we shall denote the respective complements of S_r and S_s w.r.t. S by $S_{\bar{r}}$ and $S_{\bar{s}}$, and we shall also use the notation S_{xy}, $x \in \{r,\bar{r}\}$, $y \in \{s,\bar{s}\}$, to denote the intersection of the corresponding sets S_x and S_y. In the context of the RAS-related literature, "state co-reachability" has been historically characterized as the property of *"state safety"*; hence, in the sequel, we shall tend to refer to the state S_s as the set of the *"safe"* RAS states, and, correspondingly, to the state set $S_{\bar{s}}$ as the set of *"unsafe"* states.

[5] In the last formula, it is implicitly assumed that $\hat{f}(\mathbf{s},\sigma e)$ is undefined if any of the one-step transitions that are involved in the right-hand-side recursion are undefined.

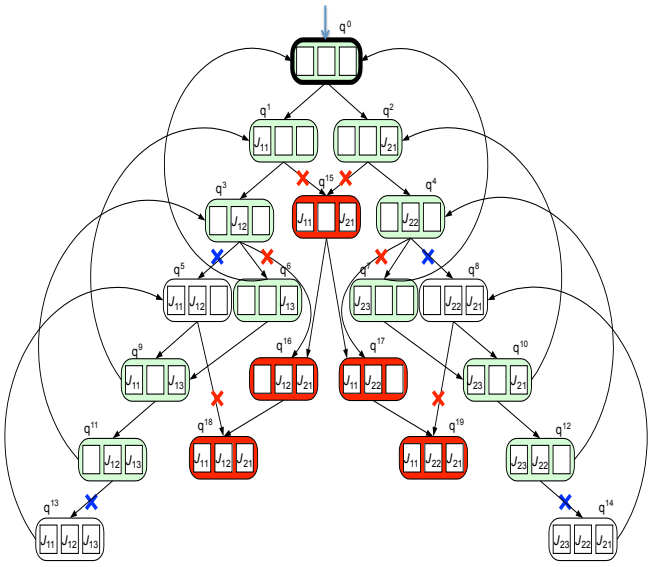

Fig. 4. The STD for the reachable state space of the FSA $G(\Phi)$ that models the buffer allocation taking place in the manufacturing cell of Fig. 1. The various RAS states are depicted graphically, with the three internal rectangles at each node representing the three worktables corresponding to resources R_1, R_2 and R_3, and with the annotation J_{jk} of these rectangles indicating the processing stage executed by the process instance that is currently loaded in the corresponding worktable. On the other hand, the considered RAS model ignores the buffering capacity of the AGV depicted in Fig. 1, since this vehicle has only a facilitating role in the part transfers taking place among the system workstations. The blue arrow pointing at the empty RAS state q^0 defines this state as the initial state for the RAS operation, while the thick borderline of the same state indicates its marked role in the RAS dynamics modeled by $G(\Phi)$. States depicted in red indicate the unsafe region $S_{r\bar{s}}$ of the considered FSA. The maximally permissive nonblocking supervisor for this FSA must confine the operation of the underlying RAS within the remaining set of (white and green) states, by recognizing and preventing the transitions that cross the boundary between the safe and unsafe subspaces; these transitions are marked by red crossings in the figure.

Fig. 4 provides the state transition diagram (STD) of the reachable subspace of the FSA $G(\Phi)$ corresponding to the RAS Φ that abstracts the qualitative dynamics of the buffer allocation taking place in Fig. 1. Furthermore, the figure also depicts the separation of the reachable space S_r into its safe and unsafe subspaces, S_{rs} and $S_{r\bar{s}}$.

B. The optimal nonblocking supervisor and its complexity

It is easy to see from all the definitions that are provided in the previous paragraphs and the example of Fig. 4, that the admissible behavior for RAS Φ, characterized by the marked language $L_m(G)$, confines the FSA $G(\Phi)$ exactly in its subspace that is defined by S_{rs}. In the relevant terminology of DES theory, the sub-automaton of $G(\Phi)$ that is induced by the state subset S_{rs} is known as the *"trim"* of $G(\Phi)$, and it can be computed by standard algorithms that are provided by qualitative DES theory. Hence, a natural way to ensure the deadlock-free operation of a given RAS Φ is by first computing the trimmed sub-automaton, $\tilde{G}(\Phi)$, of the corresponding FSA $G(\Phi)$, and subsequently implementing a logical controller that allows the occurrence of any process and resource-enabled transition in $G(\Phi)$ only if this transition appears also in $\tilde{G}(\Phi)$. In fact, such an implementation of the necessary supervision for ensuring the deadlock-free operation of RAS Φ is in line with the classical theory for the

qualitative control of DES that is known as *"Ramadge & Wonham (R&W) supervisory control theory (SCT)"* [42]. This implementation is also associated with a notion of *"optimality"* since it establishes deadlock-freedom while enforcing the minimal possible restriction to the feasible behavior of the underlying RAS.[6] Furthermore, the corresponding DES theory provides additional results that characterize the minimally restrictive deadlock avoidance policy (DAP), and support its effective computation in the form of a sub-automaton of the FSA $G(\Phi)$, even in the case that the underlying RAS Φ exhibits uncontrollable behavior [42], [7]. Finally, all the necessary algorithms for the computation of the policy-encoding sub-automaton $\tilde{G}(\Phi)$ from the original FSA $G(\Phi)$ are of polynomial complexity w.r.t. the size of $G(\Phi)$, where the latter is determined by the number of the states and the transitions of this automaton.

However, the effective deployment of the control scheme that was outlined in the previous paragraph is severely challenged by the fact that, for the most practical RAS instantiations, the size of the FSA $G(\Phi)$ grows super-polynomially (actually, very) fast w.r.t. the size of Φ.[7] An alternative implementation of the maximally permissive DAP could employ an "one-step-lookahead" control scheme that determines the admissibility of any tentative transition by assessing the safety of the resulting RAS state. Such a control scheme avoids the explicit storage of the trimmed FSA $\tilde{G}(\Phi)$, substituting the information that is contained in this FSA with the information that is provided by the on-line assessment of state safety. However, a straightforward implementation of this approach is also practically intractable, since it has been established in the relevant RAS literature that assessing the state safety of any given RAS state $\mathbf{s}$ is an NP-complete problem for all the RAS classes of the taxonomy of Table I.[8]

Next we consider how the relevant research community has sought to circumvent the above negative results regarding the super-polynomial complexity of the optimal DAP w.r.t. the size of the underlying RAS. The presented developments will reveal that, in spite of these results, currently we are able to provide very tractable implementations of the maximally permissive nonblocking supervisor for many RAS of practical size and interest. Furthermore, the development of the presented results has also extended and strengthened substantially the corresponding DES theory.

[6]From the more holistic view point of the control framework of Fig. 3, minimal restrictiveness – or, equivalently, *maximal permissiveness* – of the applied logical control policy should be interpreted as increased behavioral latitude for the controlled system that can potentially lead to an enhanced performance.

[7]Characteristically, we mention that for a *single-unit* RAS Φ where $C \equiv \max_i\{C_i\}$ and D denotes the maximal number of stages that are supported by any single resource R_i, $i = 1,\ldots,m$, the state space cardinality, $|S|$, of the corresponding FSA $G(\Phi)$ is $O(\binom{C+D}{C}^m)$. The quantity $\binom{C+D}{C}$ in this expression characterizes all the possible ways to partition the C units of capacity of any single resource type to the $D+1$ options that are defined by its D supporting stages and the pool of its idle units. On the other hand, the complete expression $\binom{C+D}{C}^m$ results from the fact that, in the considered RAS class, the entire RAS state space can be obtained by taking the cross-product of the sets that characterize the potential allocations of each single resource type.

[8]The first set of these complexity results appeared in the late 1970's and they addressed primarily RAS with conjunctive allocation [1], [19]. More recently, the works of [29], [44] have also established that the problem of state safety remains NP-complete even in the case of linear, single-unit RAS with unit resource capacities. Also, the results of [44] establish that state safety is an NP-complete problem in the RAS abstracting the traffic of the guidepath-based MHS and the free-ranging mobile agents that were discussed in the opening part of this article; in these two cases, the relevant complexity analysis must take into consideration the further constraints for the valid sequences of the resource allocation requests posed by any single process instance, that are implied by the structure of the underlying guidepath network and/or the employed tesselation.

V. Dealing with the computational challenges of the optimal nonblocking supervisor

A. "Polynomial-kernel" suboptimal nonblocking supervisors

As is the case with any other problem shown to be of an NP-complete or NP-hard nature, a first reaction of the research community dealing with the RAS deadlock avoidance problem was to seek suboptimal (i.e., non-maximally permissive) supervisors with more manageable computational requirements during their design and operational phases. A systematic way to formally characterize this endeavor is through the concept of the *"polynomial-kernel (PK)"* nonblocking supervisor. This is essentially an one-step-lookahead control scheme like the one described in the previous section for the implementation of the maximally permissive DAP, where, however, the test for safety has been substituted by the test for another state property P of polynomial complexity w.r.t. the underlying RAS size $|\Phi|$. Furthermore, in order to lead to a correct DAP, the selected property P must (i) be satisfied by the initial RAS state $\mathbf{s}_0$, and (ii) the subspace induced by the RAS reachable states satisfying P must be a strongly connected component of the reachable state space. The second requirement is important in order avoid *policy-induced* deadlocks or livelocks, i.e., situations where the considered policy takes the system to a state, or a set of states, from which there is no policy-admissible path back to the empty state $\mathbf{s}_0$. The structure of the admissible subspace of a correct PK-DAP for the RAS corresponding to the manufacturing cell of Fig. 1 is depicted in Fig. 4 by the subgraph that is induced by the green nodes of that figure.[9] The safe states q^5, q^8, q^{13} and q^{14} do not satisfy the defining property P of the considered policy and therefore they are not admissible by it.

As a more complete example of a PK-DAP, we briefly discuss a policy that has come to be known as *Banker's algorithm* in the relevant literature. The defining ideas for Banker's algorithm can be traced back to Dijkstra's work [12]. Here we discuss an implementation of this policy for the RAS class of conjunctive / disjunctive RAS in the taxonomy of Table I [27], [14]. The aforementioned property P that defines Banker's algorithm for this class of RAS is that of an *"ordered"* state. A state $\mathbf{s}$ is ordered if there exists an ordering for its active process instances such that the i-th process instance according to this ordering can advance all the way to completion utilizing only its currently allocated resources, the pool of the free resources in state $\mathbf{s}$, and the resources allocated to the first $i-1$ process instances in state $\mathbf{s}$ (which will have been released upon the earlier completion of these processes). Furthermore, assessing the admissibility of a given state $\mathbf{s}$ by Banker's algorithm boils down to the identification of an ordering for the active process instances in $\mathbf{s}$ that satisfies the aforementioned property. Such an ordering, if it exists, can be identified by a *"greedy"* search that seeks to drive to completion each of the active processes, one at a time, while releasing the resources allocated to these processes in state $\mathbf{s}$ back to the pool of free resources. Since every such process comple-

[9]More specifically, the depicted STD corresponds to an implementation for the considered RAS of the "Resource Upstream Neighborhood (RUN)" policy introduced in [25], [26].

tion increases monotonically the pool of the free resources, there is no need for backtracking in the aforementioned search. But then, the entire search process can be performed with a computational cost that is polynomial w.r.t. the size of the underlying RAS. Obviously, not every safe state is ordered. But practical experience with the implementation of Banker's algorithm in many application contexts has revealed that the algorithm can admit a pretty large part of the subspace S_{rs}.[10]

The PK-DAPs currently available can address arbitrary resource allocation (i.e., conjunctive RAS in the taxonomy of Table I), but in terms of the process-defining logic, they support primarily process types that evolve as *atomic* entities throughout their execution. Besides Banker's algorithm, some of the best known policies of this type can be traced in [2], [13], [25], [26], [28], [40]. In general, it can be argued that the identification of a pertinent property P that can lead to a correct PK-DAP is more of an art. On the other hand, in the case of RAS with no cyclic behavior in their process types, formal correctness proofs for PK-DAPs, defined by a certain property P, can be structured as an inductive argument that establishes (i) the satisfaction of property P by the initial state $\mathbf{s}_0$, and (ii) the existence of policy-admissible transitions for every RAS state $\mathbf{s}$ that satisfies property P. Furthermore, in Section V-D, we shall discuss a methodology that automates the correctness evaluation of tentative PK-DAPs for RAS with no cyclic behavior in their process types, by means of certain tests that take the form of a mathematical programming formulation.

Closing this discussion on PK-DAPs for the considered RAS, we also notice that the disjunction of a set of properties $P_1, \ldots, P_l$ defining correct PK-DAPs for a given RAS Φ, is another correct PK-DAP for Φ, as long as the number of the employed properties, l, is polynomially related to the RAS size $|\Phi|$. The subspace of S_{rs} that is admitted by this new disjunctive policy is the union of the subspaces of S_{rs} that are admitted by the constituent policies. Hence, by utilizing a set of PK-DAPs for a given RAS, one can obtain a tighter (under-)approximation of the maximally permissive DAP. The significance of this remark is further increased by the fact that some available PK-DAPs are defined through the imposition of some arbitrary ordering on the underlying resource set, with different resource orders leading to the admissibility of different parts of the underlying state space. For a comprehensive discussion on the existing set of PK-DAPs and the systematic exploitation of all the aforementioned possibilities, the reader is referred to Chapters 4 and 5 of [47].

B. RAS admitting optimal nonblocking supervision of polynomial complexity

A second typical reaction to an NP-completeness or NP-hardness result, is the quest for "special structure" of practical interest that can lead to polynomial-complexity solutions for the problem at hand. In fact, the seminal works of [1], [19] that established the first NP-completeness results for the problem of the RAS state safety, also discussed certain conditions on the sequences of the resource allocation requests posed by the RAS process types that would lead to safety assessment of polynomial complexity w.r.t. the size of the underlying RAS. Generally speaking, these conditions imply the existence of easily identifiable transition sequences leading to a monotonic increase of the pool of the free resources in the RAS behavioral space $G(\Phi)$, which, as in the case of the "ordered" RAS states, further enables a greedy search for a transition sequence that will terminate all the active process instances.

A more recent line of results leading to polynomial-complexity implementations of the maximally permissive DAP for certain RAS classes of the taxonomy of Table I is based on the essential differentiation between the notions of an unsafe state and a deadlock. We remind the reader that in the considered RAS context, a deadlock has been defined as a RAS state containing a subset of active process instances that block each other in a circular manner, since each of them holds resources requested by some other processes in the set in order to advance to their next processing stages. On the other hand, the set of unsafe states of a given RAS contains all its deadlock states, but it might also contain a subset of states that do not contain any deadlocked processes; such an unsafe state is exemplified by state q^{15} in the STD of Fig. 4. Unsafe states containing no deadlocked processes are characterized as *"deadlock-free unsafe states"*. The realization of the existence of deadlock-free unsafe states becomes essential for the complexity analysis of the considered RAS when noticing that, for most of the RAS classes of the taxonomy of Table I, the detection of a deadlock state is a task of polynomial complexity w.r.t. the underlying RAS size; this result is especially true for the class of Disjunctive / Conjunctive RAS and a relevant deadlock detection algorithm is presented in [47]. Hence, it can be inferred that, for the aforementioned RAS class, the NP-completeness of state safety is due to the presence of deadlock-free unsafe states. On the other hand, if it could be established that for certain sub-classes of this RAS class there are no deadlock-free unsafe states, then, the test for state safety could be effectively substituted by the corresponding test for deadlock, and assessing state safety would become a task of polynomial complexity w.r.t. the corresponding RAS size. Indeed, such results are available for certain sub-classes of the Disjunctive / Single-Unit RAS that are defined by easily testable conditions on the RAS structure. A more concrete example, and one of the first results of this type appearing in the literature, is stated in [49] and establishes the absence of deadlock-free unsafe states for any Disjunctive / Single-Unit RAS where every resource has at least two units of capacity and the RAS process types exhibit no internal cycling. From a more practical standpoint, this result implies that the problem of establishing deadlock-free buffer allocation in flexibly automated production cells, exemplified in Fig. 1, becomes an easy problem as long as every workstation has a buffer with at least two slots. Also, the more recent work of [50] has exploited the aforementioned result in order to develop an asynchronous, distributed coordination protocol able to ensure collision-free and nonblocking traffic for the systems of the free-ranging mobile agents that were described at the opening part of this article. Furthermore, the works of [15], [16] have further established that the aforementioned requirement for non-unit resource capacities must be satisfied only by a critical subset of the resource types of the considered Disjunctive / Single-Unit RAS, while additional extensions of all these results are developed in [29]. A comprehensive treatment of the topic of RAS admitting maxi-

[10]The reader can verify that for the example RAS defined by Figs 1 and 4, the presented version of Banker's algorithm will admit the entire reachable and safe subspace S_{rs}, i.e., the subspace admitted by the maximally permissive DAP.

mally permissive nonblocking supervision of polynomial complexity w.r.t. the underlying RAS size is provided in Chapter 3 of [47].

C. The optimal DAP as a "classifier" of the RAS states

While the two research lines that were discussed in the previous paragraphs were driven by rather typical reactions to the established NP-hardness of the maximally permissive DAP, tremendous progress w.r.t. the practical implementation of this policy to RAS instances of a very large size and practical significance has been attained in the recent years by a more aggressive approach that has treated this complexity result as a "worst-case" result and has tried to pursue the deployment of the maximally permissive DAP nevertheless, hoping for a more benign *empirical* complexity. A more detailed positioning of the rationale that underlies this new approach and drives its major developments, is as follows: The maximally permissive DAP – and, in fact, any other supervisor developed by R&W SCT – is essentially a *"classifier"* that dichotomizes the state set S of the underlying FSA $G(\Phi)$ into its admissible and inadmissible subsets. And while computing this dichotomy for any given RAS Φ is a computationally difficult task, in general, it might still be possible to contain the difficult part of this computation in an "off-line" stage, and eventually rehash / encode the obtained results in a classification mechanism that will enable a tractable "on-line" assessment of the admissibility of any given RAS state. Some important issues that must be addressed for a complete realization of this idea are (i) the specification of classification mechanisms[11] able to provide an effective representation of the corresponding state space dichotomy for any instance of the considered RAS classes, and (ii) the design of effective and computationally efficient algorithms for the computation of parsimonious implementations of the sought classifiers for any given instance from the considered RAS class.

As in the case of the research lines described in the previous sections, this new research program has targeted primarily / more explicitly the RAS class of conjunctive resource allocation and atomic process instances. In this context, the aforementioned tasks regarding the specification and the deployment of the sought classifiers are primarily defined by the vector structure of the underlying RAS state, introduced in Section IV, and they are further facilitated by additional structural and behavioral properties of the considered RAS. Furthermore, for a systematic exposition of the currently available results, it is pertinent to differentiate the employed classification schemes into *"parametric"* and *"non-parametric"* classifiers, and this is the approach that we shall adopt in the following discussion.

The specification of a *parametric* classifier able to represent the maximally permissive DAP for the aforementioned RAS class is based on some rather classical results of classification theory [39] asserting that any finite set of integer vectors of finite dimensionality can be dichotomized by a two-layered set of linear inequalities. In the considered application context, the inequalities employed by the first layer are imposed on the RAS state, while the inequalities employed by the second layer are defined w.r.t. the indicator variables that characterize the satisfiability of the first-layer inequalities. Alternatively, the second layer of the classification logic can be replaced by an appropriately defined Boolean function of the same set of indicator variables.

The practical construction of parametric classifiers expressing the maximally permissive DAP for any given RAS instance Φ is substantially facilitated by the following two facts: (i) A monotonicity property that is possessed by the state safety concept in Disjunctive / Conjunctive RAS and postulates that if state $\mathbf{s}$ is no greater, componentwise, than state $\mathbf{s}'$, and state $\mathbf{s}'$ is safe, then state $\mathbf{s}$ is also safe. This property enables the restriction of the coefficients of the inequalities employed by the sought classifier to nonnegative values, and, more importantly, it also enables the computation of a pertinent classifier that will provide effective representation of the sought dichotomy of the admissible and inadmissible subspaces by considering explicitly in this computation only the maximal safe and the minimal unsafe states. (ii) An additional simplification in the design of the sought classifiers stems from the realization that, in any given RAS Φ, process instances executing certain processing stages that will never contribute to the formation of a deadlock, can be projected away during the assessment of state safety.[12] The existing methodology for the computation of the considered classifiers identifies and removes these redundant state components in an automated manner.

The literature also avails of results that characterize the potential existence and the computation of parametric classifiers for the representation of the maximally permissive DAP that consist only of a single layer of linear inequalities on the underlying RAS state; such classifiers are characterized as *"linear"*. Obviously, linear classifiers are simpler to analyze and construct than the generic two-layered classifiers described in the previous paragraphs. But a key condition for the existence of such a classifier for any given RAS Φ is that the convex hull of the safe states of this RAS does not contain any unsafe states. On the other hand, it can also be shown that this requirement for linear separability of the safe and unsafe subspaces will always be satisfied by RAS with binary state spaces.[13]

Finally, the computation of practical, parsimonious parametric classifiers for the considered classification task has substantially benefited from the correspondence of this computation to the minimal set covering problem [55].[14] More specifically, this correspondence has provided efficient customized combinatorial optimization algorithms for computing *structurally minimal* instantiations of the sought classifiers for any given RAS instance, as well as heuristics with guaranteed performance bounds that can replace the aforementioned algorithms in the

[11] Also known as the "architectures" of the sought classifiers.

[12] As a more concrete example of such stages, the reader can consider the terminal stage(s) of any given process type.

[13] RAS with binary state spaces arise whenever each processing stage requests at least one resource type of unit capacity.

[14] In the case of linear classification, the connection of the design of a *structurally minimal* classifier to the minimal set covering problem is easily established by corresponding each candidate inequality separating some subset of the (minimal) unsafe states from the set of (maximal) safe states, to the subset of the separated (minimal) unsafe states. Then, a structurally minimal linear classifier – i.e., a classifier that uses a minimal number of linear inequalities – for the representation of the target DAP is a complete "cover" of the set of (minimal) unsafe states that utilizes a minimum possible set of the partial "covers" defined as above. In the non-linear case, the specification of the partial "covers" of the (minimal) unsafe states by the employed classification mechanism is a little more complicated, but the primary notion of developing a complete cover for the set of (minimal) unsafe states remains the same.

case of larger problem instances that would render these algorithms computationally intractable.

Non-parametric classifiers for the representation of the maximally permissive DAP in the considered RAS classes operate on the following idea: The necessary guarding against transitions from the safe to the unsafe RAS state space can be effected, in principle, by storing the entire set of unsafe states that could result from such transitions, and employing an one-step-lookahead scheme that will block any transition leading to one of these states. The corresponding set of states is known as the set of *"boundary"* unsafe states in the relevant literature, and it can be stored and searched efficiently through the employment of advanced data structures that are known as "TRIEs" [5] and are conceptually similar to the "Binary Decision Diagrams (BDDs)" [6] that have been used extensively in symbolic computation [9]. Furthermore, the aforementioned monotonicity of state safety in the considered RAS classes implies that the entire set of boundary unsafe states can be characterized only by its minimal elements, a fact that reduces dramatically the information that must be explicitly stored in the aforementioned data structures.

The literature also avails of very efficient algorithms for the identification of the minimal unsafe states and the construction of the TRIE data structures that must be employed during the "on-line" assessment of the state safety. The efficiency of these algorithms stems from the characterization of the state unsafety in the underlying RAS classes as unavoidable absorption to some RAS deadlock; hence, these algorithms manage to reconstruct the entire unsafe state space by first enumerating programmatically all the RAS deadlocks and subsequently retrieving all the deadlock-free unsafe states, including all the boundary states, through a pertinent "backtracing" from the (re-)constructed deadlocks. The tractability of such a computational scheme is further enhanced through an explicit focus upon *minimal* deadlocks and deadlock-free unsafe states,[15] and/or the employment of symbolic techniques in the aforementioned computation.

Finally, the ability to represent the entire set of the boundary unsafe states only through its minimal elements, when combined with the aforementioned capability to retrieve these minimal unsafe states through "backtracing" from the minimal deadlocks, enables also the effective implementation of the maximally permissive DAP even for RAS with infinite state spaces;[16] such a particular class investigated in the literature is the class of RAS modeling parallel programs with Reader/Writer locks, where the number of process instances that could simultaneously access a resource in the "reading" mode can be arbitrarily large. A more expansive and systematic treatment of the representation of the maximally permissive DAP as a parsimonious classifier of the underlying RAS state space, together with a comprehensive bibliography of the corresponding results, can be found in [37], [17], [43].

[15]In more technical terms, these are deadlocks or deadlock-free unsafe states such that the removal of any single process instance from them will turn them into safe states.

[16]The finiteness of the target sets of the minimal deadlocks and the minimal unsafe states is guaranteed by the, so called, Dickson's lemma [11].

D. Coping with the RAS deadlock in the Petri net modeling framework

As we have seen in Section IV, the FSA-based representation of the behavioral RAS dynamics w.r.t. deadlock can provide a succinct and very intuitive characterization of the problem of deadlock avoidance and the corresponding optimal policy, but the practical computational power of this representation is substantially limited by the very large size of the involved FSAs. An additional limitation of the FSA-based representation of the RAS behavior is that it fails to capture any information on the *"mechanisms"* that generate the dynamics that are represented by the considered FSA and its corresponding STD. Hence, in an effort to develop a more profound understanding of these mechanisms, the relevant research community has employed additional representations that formalize the underlying system structure and enable a systematic analysis of the impact of this structure on the generated behavior; this type of analysis of the RAS deadlock problem has come to be known as *"structural"* analysis.

Historically, attempts to a structural analysis of the RAS deadlock problem have preceded the formal investigation of this problem in the FSA modeling framework. These early attempts have tried to represent and trace the formation of the circular waiting patterns that correspond to a RAS deadlock through various graphical structures that express the (evolving) dependencies among the system resources determined by their (current) allocation to the running processes and the pending requests of these processes. Similar graphical structures have been instrumental in the development of the results on the absence of deadlock-free unsafe states for certain RAS classes, that were covered in Section V-B. But in the following discussion we shall focus primarily on the structural analysis of the RAS deadlock that relies on the *Petri net (PN)* modeling framework [36], since this area has been especially active and influential in the relevant literature during the last three decades. More generally, together with FSAs, PNs is the second major modeling framework used by the qualitative DES theory, and they are particularly recognized for the richness and the clarity of their semantics. These semantics enable a concise and lucid representation of the behavioral dynamics of the modeled DES while avoiding an explicit enumeration of the corresponding state space. In the sequel, we shall assume that the reader is familiar with the basic PN modeling framework; some excellent introductions to this subject are provided in [36], [7].

In the PN modeling framework, a RAS Φ is represented by a *process-resource* net $\mathcal{N}$, consisting of a set of *process subnets* modeling the various process types of the RAS, and a set of *resource places* that model the availability of the system resources. The aforementioned process subnets are interconnected through the resource places, and the corresponding connectivity models the allocation of the system resources to the running process instances. One of the first key results in the theory of process-resource nets was the connection of the concept of the RAS deadlock in Disjunctive / Single-Unit RAS to the concept of the *"empty siphon"*. This concept and its connection to the RAS deadlock are exemplified in Fig. 5, which depicts the empty siphon characterizing the RAS deadlock of Fig. 1. The seminal result of [13] established that a Disjunctive / Single-Unit RAS with no cycling in its process types will possess no RAS

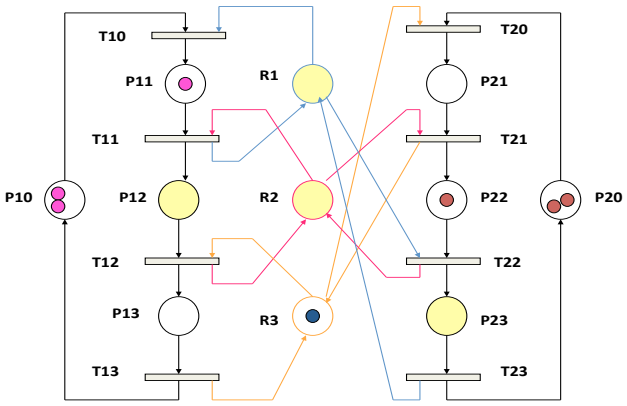

Fig. 5. The process-resource net modeling the buffer allocation that takes place in the example manufacturing cell of Fig. 1. The two process types corresponding to J_1 and J_2 are modeled by the two circuits annotated by black lining in the depicted net. In particular, the "process" places $p_{ij},\ i=1,2,\ j=1,2,3$, model the corresponding processing stages of the underlying RAS, while the "idle" places $p_{i0},\ i=1,2$, model the "external environment" for the two process types. The resource availability is traced by the marking of the "resource" places R_1, R_2, and R_3. The particular marking depicted in this figure corresponds to the RAS deadlock state depicted in Fig. 1. In the transitional dynamics of the depicted net, the occurring deadlock is manifested by the presence of the set of empty places $\mathcal{S}=\{p_{12}, p_{23}, R_1, R_2\}$, which is annotated in yellow. Letting $\bullet\mathcal{S}$ (resp., $\mathcal{S}\bullet$) denote the set of transitions that have an output (resp., input) place in $\mathcal{S}$, it can be verified that, in the considered case, $\bullet\mathcal{S}\subseteq\mathcal{S}\bullet$. This last property renders $\mathcal{S}$ a *siphon*. Furthermore, since all places of $\mathcal{S}$ are empty, $\mathcal{S}$ is an *empty siphon*. But then, all the transitions in $\mathcal{S}\bullet$ are disabled in the considered marking, since they require at least one token from some place in $\mathcal{S}$. Moreover, any transition in $\bullet\mathcal{S}$ that could bring new tokens in $\mathcal{S}$ is part of $\mathcal{S}\bullet$, and therefore, disabled. Hence, it can be concluded that the depicted empty siphon $\mathcal{S}$ will remain empty throughout the entire evolution of the dynamics of the considered process-resource net, and the transitions in $\mathcal{S}\bullet$ will be dead during this evolution.

deadlocks in its behavior[17] if and only if there are no reachable empty siphons for the corresponding process-resource net. But the presence of an empty siphon in any given marking M of some PN $\mathcal{N}$ can be easily tested by algorithms of polynomial complexity w.r.t. the size of $\mathcal{N}$, where the latter is defined by the size of the bipartite digraph that defines $\mathcal{N}$ [8]. Furthermore, the work of [8] showed that in the case of bounded PNs, these algorithms can be converted to a *Mixed Integer Programming (MIP)* formulation employing a number of variables and constraints that are polynomially related to the size of PN $\mathcal{N}$; in the resulting test, the main outcome is communicated by the optimal value of the MIP formulation, while, in the case that the tested marking M contains empty siphons, the returned optimal solution also enables the identification of the maximal empty siphon in M. The MIP formulation mentioned above becomes even more useful when the tested marking M is converted into a variable that lives in the reachability space of the corresponding net $\mathcal{N}$; in this way, the resulting MIP formulation becomes an instrument for testing the presence of an empty siphon over the entire reachability space of $\mathcal{N}$.[18] And when combined with the aforementioned results of [13], this MIP formulation eventually becomes a *verification tool* for the absence of deadlock in any instance from the corresponding subclass of Disjunctive / Single-Unit RAS.

The extension of the above results to broader RAS classes is a quite non-trivial task, since it must account for the non-uniformity of the posed resource requests w.r.t. any single resource type. This non-uniformity can give rise to deadlocking situations where the resources that are entangled in the deadlock have a non-zero slack capacity, but yet this capacity is not adequate for satisfying the requests of the deadlocked processes. Furthermore, in this more general case, the blocking resources might not be part of a deadlock but of a *livelock*, where the slack capacity of these resources can be used repetitively to satisfy the requests of other running processes that are not entangled in the deadlock. These complications can be effectively circumvented by (i) extending the notion of empty siphon to that of *deadly marked siphon*, and (ii) searching for deadly marked siphons that interpret any given RAS deadlock in a *"modified"* reachability space that is obtained from the original reachability space through a pertinent projection. A complete treatment of these developments can be found in [40], [46], [47], while some additional results of a similar nature can be traced in [52], [24], [33]. All these works also provide accompanying MIP formulations that can function as verification tools for the deadlock freedom of the corresponding RAS classes.

In certain cases, the structural characterizations of the RAS deadlock and the corresponding MIP formulations can also be used for the synthesis of correct DAPs for the considered RAS. Such an approach is especially amenable in the case of DAPs that can be expressed as a set of inequalities on the marking of the underlying process-resource net, since the seminal works of [18], [35] have established that these inequalities can be enforced upon the underlying PN by the super-position to this net of a set of additional places – known as *"monitor"* places – that play a role very similar to that of the resource places in the process-resource nets. Hence, if the PN that results from the super-position to a process-resource net of a set of monitors representing a tentative DAP falls within a class of process-resource nets whose liveness and reversibility are equivalent to the absence of deadly marked siphons, then, the assessment of the policy correctness can be performed automatically through the corresponding MIP formulation. The literature also avails of endeavors for an incremental synthesis of a correct DAP for a given RAS, through (i) the employment of the aforementioned MIPs for the detection of deadly marked siphons in the relevant reachability space (also known as *"potential deadlocks"*), (ii) the elimination of the identified potential deadlocks from the net dynamics through a set of pertinent inequalities enforced on the net marking by a set of monitor places, and (iii) the (re-)assessment of the augmented net for absence of such badly marked siphons. Clearly, if successful, such an approach can provide a correct DAP for the considered RAS while avoiding any explicit enumeration / exploration of the underlying state space. In the case of Disjunctive / Conjunctive RAS with no cycling in their process types and binary state spaces, such incremental synthesis has been shown to be capable of comput-

[17] Or, that the corresponding process-resource net will be *live* and *reversible*

[18] Since, however, the analytical characterization of the reachability space of a given PN $\mathcal{N}$ by a set of linear inequalities is a challenging task, in general, one has to resort to over-approximations of this set that are obtained by means of the state-equation of the net and / or its place-invariants. The employment of such an over-approximation raises the possibility of detecting empty siphons that do not belong to a reachable marking, and turns the overall test into a "sufficiency" test for the absence of empty siphons.

ing even the maximally permissive DAP [34].[19] Furthermore, extensive computational experimentation with a series of pretty sizable RAS has also demonstrated the scalability and the computational tractability of the method. Fig. 6 demonstrates the application of this incremental-synthesis method for the development of a set of linear inequalities – or a "linear classifier" – and the corresponding monitor places implementing the maximally permissive DAP for the example RAS of Fig. 1. On the other hand, it is also true that the application of this method to the computation of the maximally permissive DAP for RAS with non-binary state spaces is practically limited by the previously discussed potential inability to represent this DAP by a set of linear inequalities on the RAS state.

We close our discussion on the existing results regarding the structural characterizations of the RAS deadlock through the concept of the PN siphon, and the implications of these characterizations for the analysis and control of the relevant RAS behavior, by briefly mentioning an additional line of research that has sought to employ these structural characterizations in order to provide some explanation for the observed possibility of ensuring sufficient control of all the potential deadlocks of a given RAS by controlling explicitly only a limited subset of them.[20] Results of this line can be traced in [32], [48], while the work of [32] has also applied these results in a control-synthesis process through the introduction of the concept of *"elementary"* siphons.

VI. Going forward

It should be evident from the entire discussion of the previous parts of this article that the RAS concept and its accompanying theory constitute a well-established theme in the academic research community. This discussion also reveals the methodological richness and the analytical and computational strength that characterize the existing developments in this area. The presented results offer rigorous and structured solutions to some ubiquitous problems that must be addressed by the engineering community as it tries to increase the automation levels for a broad spectrum of technological applications with ever increasing operational scale and complexity. In particular, Sections IV and V, on the past developments regarding the RAS deadlock problem and its efficient resolution through the deployment of the maximally permissive DAP or some good approximation to this policy, demonstrate how automation-related research can benefit from the effective utilization of the existing formal methods for behavioral verification and synthesis. These two sections also show how the RAS-related research has contributed to the broader DES theory by capitalizing upon the special structure and the more concrete insights that are offered by the target application domains.

The previous discussion also unveils a series of directions in which the current RAS theory can be extended and strength-

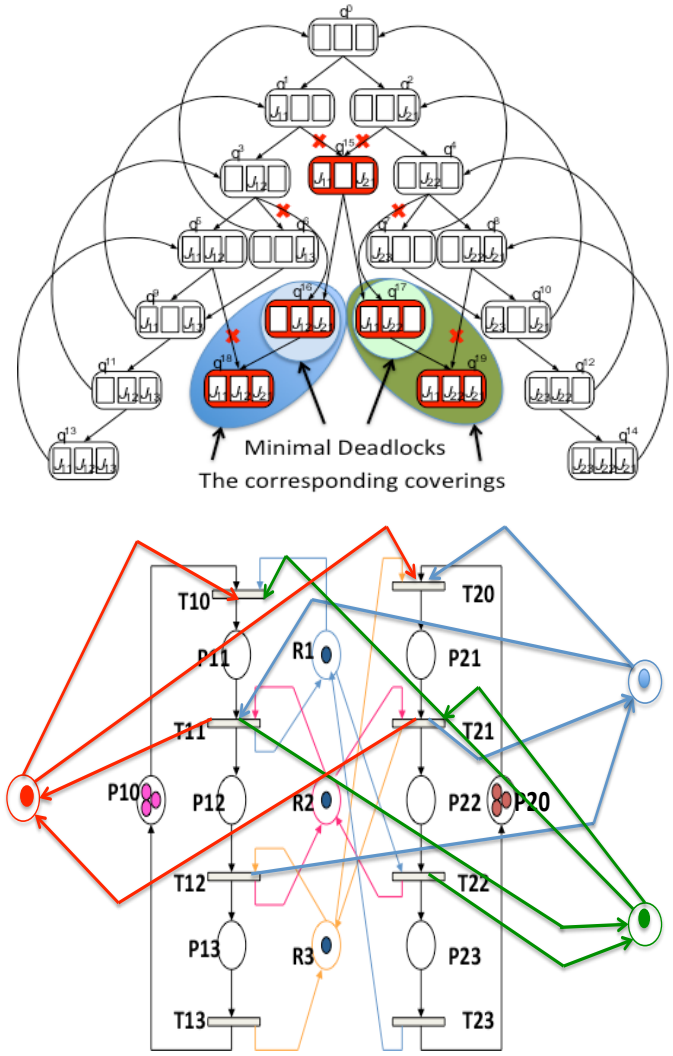

Fig. 6. Computing the maximally permissive DAP for the RAS of Fig. 1 through the incremental synthesis approach that is presented in [34]. The approach starts with the solution of a MIP formulation that assesses the presence of minimal empty siphons in the dynamics of the original process-resource net of Fig. 5. The solution of this formulation could detect either of the two minimal deadlocks corresponding to the RAS states q^{16} and q^{17}. Each of these two deadlocks can be eliminated from the dynamics of the process-resource net by enforcing upon these dynamics the respective inequalities $M(p_{12})+M(p_{21}) \leq 1$ and $M(p_{11})+M(p_{22}) \leq 1$. It is also important to notice that each of these inequalities does not eliminate only the corresponding deadlock state, but also any other state that includes the considered deadlock; in the above figure, the state subsets that are eliminated by each of these two inequalities, are respectively indicated by the blue and the green blobs in the depicted STD. The two aforementioned inequalities are enforced on the dynamics of the original process-resource net through the corresponding blue and green "monitor" places that can be constructed using the theory of [35]. The augmented net that is obtained from the addition of these monitor places remains an ordinary process-resource net, and therefore, its liveness and reversibility can still be tested through the absence of empty siphons. The solution of the relevant MIP formulation reveals an empty siphon that corresponds to the unsafe state q^{15}, which in the dynamics of the augmented process-resource net has turned into a *policy-induced* deadlock. This new empty siphon can be eliminated through the imposition of the marking inequality $M(p_{11})+M(p_{21}) \leq 1$, that is implemented by the red "monitor" place of the depicted net. The evaluation of this new process-resource net through the corresponding MIP formulation reveals the absence of any further empty siphons, and establishes its liveness and reversibility. It is also important to notice that the constructed "monitors" eliminate all the unsafe states of the net while retaining all of its safe states. Hence, the augmented net depicted in the above figure constitutes a PN-based representation of the maximally permissive DAP for the RAS of Fig. 1.

[19] In fact, the results of [34] are applicable even in the case of Disjunctive / Conjunctive RAS with binary state spaces and *cyclic* behavior for the RAS process types, as long as the routing decisions of these process types are independent from the underlying resource allocation function and the imposed DAP; this requirement can be observed by treating the process routing as uncontrollable by the designed policy.

[20] Obviously, a straightforward alternative explanation for these observed dependencies is provided by the classification theory that is discussed in Section V-C, and, in particular, the notion of "covering" of the RAS unsafe states that is effected by the classifying inequalities.

ened, and some important open challenges that must be systematically addressed by the relevant research community in order to eventually materialize the control paradigm that is epitomized by Fig. 3. Hence, when it comes to the behavioral RAS theory, one can consider the further development of the existing theory in order to address more complex classes and behaviors of the RAS taxonomy presented in Section III, than the usually studied class of Disjunctive / Conjunctive RAS. More specifically, while the currently existing results can provide liveness characterization and assessment for these broader RAS classes, there is a remaining need for methodology that will support the expedient synthesis of the maximally permissive DAP for these RAS, or some pertinent approximations of this policy. One can also consider the extension of the basic RAS behavioral control problem addressed in this article by considering application environments that provide only partial observability of the underlying RAS function, or systems that, due to their scale and / or structure, might require more distributed supervision than the centralized control scheme of Fig. 3. Reactive or proactive accommodation of random capacity losses w.r.t. certain resources due to the occurrence of unexpected events is another issue that has received only limited attention in the current RAS literature. All these extensions can be formalized through relevant results in the existing DES theory, but it is also expected that, as in the case of the past developments, the special and rich structure of the RAS concept will enable customized analyses and solutions for this set of problems, as well.

The RAS behavioral control problem can also be extended by extending the set of specifications that are addressed by it, beyond the issue of deadlock avoidance and the establishment of nonblocking behavior. As a more concrete example of such an extension, one can consider the enforcement of a "production ratio" constraint for the manufacturing cell of Fig. 1 stipulating that the two supported product types must be produced in lockstep, or that the difference of the cumulative productions for these two products should observe certain bounds, at any point in time. Then, one can seek to characterize and compute the maximally permissive policy that ensures the liveness of the considered RAS while observing this additional constraint. The resulting supervisory control problem can be addressed using the notion of "fairness" in the relevant DES theory, and a first set of results for this problem are provided in [23]. Other behavioral constraints of similar flavor can arise, for instance, by the need of observing certain formations or some patrolling procedures by a fleet of mobile agents, and by the enforcement of "aging" and other priority schemes in the resource allocation taking place among the various threads of a multi-threaded software. Recently, the work of [22], [21] has sought to extend the basic Petri net-based representation of the considered RAS in order to address time-related constraints.

But the primary open challenge for the effective complementation of the RAS control framework that has been delineated in this article is the effective and efficient resolution of the RAS performance control problem. Any pertinent solution to this new RAS control problem must integrate all the existing results of the corresponding logical control theory and remain computationally tractable. It must also account for all the stochasticities that are encountered in the underlying application domains, and remain robust to these stochasticities. Chapter 6 of [47] have shown how (some variations of) the resulting scheduling problem can be formulated, in principle, using the fundamental modeling frameworks of Markov Decision Processes (MDP) and Stochastic Dynamic Programming (DP) [3]. This analysis has also shown how the operating logic of the applied DAP can be effectively integrated in the problem formulation, and the synergies that are developed by this integration, since the resulting MDP problem belongs to an MDP sub-class with a rich theory and powerful solution algorithms. But it is also true that the enumerative nature of the optimal MDP / scheduling policy w.r.t. the underlying state space renders challenging (usually intractable) even the description of such a policy, let alone its computation. A solution to these computational challenges can be pursued in the context of the rather fledgling area of Approximate Dynamic Programming (ADP) [4], [41]. ADP has shown significant potential for providing powerful and structured approximations to the optimal policy in many DP applications, but, at the same time, the effective customization of the more generic ideas offered by this theory to a particular application context requires substantial methodological as well as contextual insights, and extensive "tuning" through empirical experimentation. Some very recent developments that seek to customize a version of the current ADP theory to the aforementioned RAS scheduling problem, and seem to hold particular promise regarding their ability to provide an effective balance between the computational tractability and the operational efficiency of the derived solutions, are presented in [31]. But definitely much more work is needed in this particular direction.

Finally, as the presented RAS theory grows and strengthens its methodology along the lines indicated in the previous paragraphs, additional endeavor must be expended towards the development of the human capital and of the technological and computational base that will enable the constructive migration of this theory to the future engineering practice. This endeavor certainly involves the eventual undertaking of some "pilot" large-scale applications that will highlight the technical strength of the theory and the competitive advantage that can be supported by it. But even more importantly, it must also seek the effective integration of the existing and the emerging results into the relevant engineering curricula, and the organization of these results in a series of computational platforms that will enable their robust and expedient utilization by the field engineers. In fact, this last activity can be part of a broader initiative concerning the further promotion of DES theory and of the emerging formal methods in the engineering curriculum and practice. It is expected that, collectively, all these endeavors will define a spectrum of fundamental developments and trends with profound and transformative repercussions for the related fields of control and automation engineering.

Aknowledgment

This article has been based on a tutorial on RAS theory that was offered by the author while visiting the Automation group at the Chalmers University of Technology. The author thanks the group for the hospitality and the constructive discussions that he enjoyed during his visit.

REFERENCES

[1] T. Araki, Y. Sugiyama, and T. Kasami. Complexity of the deadlock avoidance problem. In *2nd IBM Symp. on Mathematical Foundations of Computer Science*, pages 229–257. IBM, 1977.

[2] Z. A. Banaszak and B. H. Krogh. Deadlock avoidance in flexible manufacturing systems with concurrently competing process flows. *IEEE Trans. on Robotics and Automation*, 6:724–734, 1990.

[3] D. P. Bertsekas. *Dynamic Programming and Optimal Control, Vol. 2*. Athena Scientific, Belmont, MA, 1995.

[4] D. P. Bertsekas and J. N. Tsitsiklis. *Neuro-Dynamic Programming*. Athena Scientific, Belmont, MA, 1996.

[5] P. Brass. *Advanced Data Structures*. Cambridge University Press, NY,NY, 2008.

[6] R. E. Bryant. Symbolic Boolean manipulation with ordered binary-decision diagrams. *ACM Comput. Surv.*, 24(3):293–318, 1992.

[7] C. G. Cassandras and S. Lafortune. *Introduction to Discrete Event Systems (2nd ed.)*. Springer, NY,NY, 2008.

[8] F. Chu and X-L. Xie. Deadlock analysis of Petri nets using siphons and mathematical programming. *IEEE Trans. on R&A*, 13:793–804, 1997.

[9] E. M. Clarke Jr., O. Grumberg, and D. A. Peled. *Model Checking*. The MIT Press, Cambridge, MA, 1999.

[10] H. M. Deitel. *Operating Systems*. Addison Wesley, Reading, MA, 1990.

[11] L.E. Dickson. Finiteness of the odd perfect and primitive abundant numbers with *n* distinct prime factors. *American Journal of Mathematics*, 35(4):413–422, 1913.

[12] E. W. Dijkstra. Cooperating sequential processes. Technical report, Technological University, Eindhoven, Netherlands, 1965.

[13] J. Ezpeleta, J. M. Colom, and J. Martinez. A Petri net based deadlock prevention policy for flexible manufacturing systems. *IEEE Trans. on R&A*, 11:173–184, 1995.

[14] J. Ezpeleta, F. Tricas, F. Garcia-Valles, and J. M. Colom. A Banker's solution for deadlock avoidance in FMS with flexible routing and multi-resource states. *IEEE Trans. on R&A*, 18:621–625, 2002.

[15] M. P. Fanti, B. Maione, S. Mascolo, and B. Turchiano. Event-based feedback control for deadlock avoidance in flexible production systems. *IEEE Trans. on Robotics and Automation*, 13:347–363, 1997.

[16] M. P. Fanti, B. Maione, and B. Turchiano. Event control for deadlock avoidance in production systems with multiple capacity resources. *Studies in Informatics and Control*, 7:343–364, 1998.

[17] Z. Fei. *Symbolic Supervisory Control of Resource Allocation Systems*. PhD thesis, Chalmers University of Technology, Gothenburg, Sweden, 2014.

[18] A. Giua, F. DiCesare, and M. Silva. Generalized mutual exclusion constraints on nets with uncontrollable transitions. In *Proceedings of the 1992 IEEE Intl. Conference on Systems, Man and Cybernetics*, pages 974–979. IEEE, 1992.

[19] E. M. Gold. Deadlock prediction: Easy and difficult cases. *SIAM Journal of Computing*, 7:320–336, 1978.

[20] M. P. Groover. *Fundamentals of Modern Manufacturing: Materials, Processes and Systems*. Prentice Hall, Englewood Cliffs, N.J., 1996.

[21] H. Hu, M. Zhou, and Z. Li. Algebraic synthesis of timed supervisor for automated manufacturing system using Petri nets. *IEEE Trans. on Automation Science & Engineering*, 7:549–557, 2010.

[22] H. Hu, M. Zhou, and Z. Li. Low-cost and high-performance supervision in ratio-enforced automated manufacturing systems using timed Petri nets. *IEEE Trans. on Automation Science & Engineering*, 7:933–944, 2010.

[23] H. Hu, M. Zhou, and Z. Li. Liveness and ratio-enforcing supervision of automated manufacturing systems using Petri nets. *IEEE Trans. on SMC: Part A*, 42:392–403, 2012.

[24] M. Jeng, X. Xie, and M. Y. Peng. Process nets with resources for manufacturing modeling and their analysis. *IEEE Trans. on Robotics & Automation*, 18:875–889, 2002.

[25] M. Lawley, S. Reveliotis, and P. Ferreira. FMS Structural Control and the Neighborhood Policy, Part 1: Correctness and Scalability. *IIE Trans.*, 29:877–887, 1997.

[26] M. Lawley, S. Reveliotis, and P. Ferreira. FMS Structural Control and the Neighborhood Policy, Part 2: Generalization, Optimization and Efficiency. *IIE Trans.*, 29:889–899, 1997.

[27] M. Lawley, S. Reveliotis, and P. Ferreira. The application and evaluation of Banker's algorithm for deadlock-free buffer space allocation in flexible manufacturing systems. *Intl. Jrnl. of Flexible Manufacturing Systems*, 10:73–100, 1998.

[28] M. Lawley, S. Reveliotis, and P. Ferreira. A correct and scalable deadlock avoidance policy for flexible manufacturing systems. *IEEE Trans. on Robotics & Automation*, 14:796–809, 1998.

[29] M. A. Lawley and S. A. Reveliotis. Deadlock avoidance for sequential resource allocation systems: hard and easy cases. *Intl. Jrnl of FMS*, 13:385–404, 2001.

[30] P. Lawrence. *Workflow Handbook*. John Wiley & Sons, Inc., NY, NY, 1997.

[31] R. Li and S. Reveliotis. Performance optimization for a class of generalized stochastic Petri nets. *Discrete Event Dynamic Systems: Theory and Applications*, (to appear).

[32] Z.W. Li and M.C. Zhou. Elementary siphons of Petri nets and their application to deadlock prevention in flexible manufacturing systems. *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, 34(1):38–51, 2004.

[33] H. Liao, Y. Wang, H. K. Cho, J. Stanley, T. Kelly, S. Lafortune, S. Mahlke, and S. Reveliotis. Concurrency bugs in multithreaded software: Modeling and analysis using Petri nets. *Discrete Event Dynamic Systems: Theory and Applications*, 23:157–195, 2013.

[34] H. Liao, Y. Wang, J. Stanley, S. Lafortune, S. Reveliotis, T. Kelly, and S. Mahlke. Eliminating concurrency bugs in multithreaded software: A new approach based on discrete-event control. *IEEE Trans. on Control System Technology*, 21:2067–2082, 2013.

[35] J. O. Moody and P. J. Antsaklis. *Supervisory Control of Discrete Event Systems using Petri nets*. Kluwer Academic Pub., Boston, MA, 1998.

[36] T. Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77:541–580, 1989.

[37] A. Nazeem. *Designing parsimonious representations of the maximally permissive deadlock avoidance policy for complex resource allocation systems through classification theory*. PhD thesis, Georgia Tech, Atlanta, GA, 2012.

[38] M. A. Nielsen and I. L. Chuang. *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press, Cambridge, UK, 2010.

[39] N. J. Nilsson. *The Mathematical Foundations of Learning Machines*. Morgan Kaufmann, San Mateo, CA, 1990.

[40] J. Park and S. A. Reveliotis. Deadlock avoidance in sequential resource allocation systems with multiple resource acquisitions and flexible routings. *IEEE Trans. on Automatic Control*, 46:1572–1583, 2001.

[41] W. B. Powell. *Approximate Dynamic Programming: Solving the Curses of Dimensionality*. Wiley Interscience, Hoboken, NJ, 2007.

[42] P. J. G. Ramadge and W. M. Wonham. The control of discrete event systems. *Proceedings of the IEEE*, 77:81–98, 1989.

[43] S. Reveliotis and A. Nazeem. Deadlock avoidance policies for automated manufacturing systems using finite state automata. In J. Campos, C. Seatzu, and X. Xie, editors, *Formal Methods in Manufacturing*, pages 169–195. CRC Press / Taylor and Francis, 2014.

[44] S. Reveliotis and E. Roszkowska. On the complexity of maximally permissive deadlock avoidance in multi-vehicle traffic systems. *IEEE Trans. on Automatic Control*, 55:1646–1651, 2010.

[45] S. Reveliotis and E. Roszkowska. Conflict resolution in free-ranging multi-vehicle systems: A resource allocation paradigm. *IEEE Trans. on Robotics*, 27:283–296, 2011.

[46] S. A. Reveliotis. On the siphon-based characterization of liveness in sequential resource allocation systems. In *Applications and Theory of Perti Nets 2003*, pages 241–255, 2003.

[47] S. A. Reveliotis. *Real-time Management of Resource Allocation Systems: A Discrete Event Systems Approach*. Springer, NY, NY, 2005.

[48] S. A. Reveliotis. Implicit siphon control and its role in the liveness enforcing supervision of sequential resource allocation systems. *IEEE Trans. on SMC: Part A*, 37:319–328, 2007.

[49] S. A. Reveliotis, M. A. Lawley, and P. M. Ferreira. Polynomial complexity deadlock avoidance policies for sequential resource allocation systems. *IEEE Trans. on Automatic Control*, 42:1344–1357, 1997.

[50] E. Roszkowska and S. Reveliotis. A distributed protocol for motion coordination in free-range vehicular systems. *Automatica*, 49:1639–1653, 2013.

[51] J. A. Tompkins, J. A. White, Y. A. Bozer, and J. M. A. Tanchoco. *Facilities Planning (4th ed.)*. John Wiley & Sons, Inc., Hoboken, NJ, 2010.

[52] F. Tricas, F. Garcia-Valles, J. M. Colom, and J. Ezpeleta. A Petri net structure-based deadlock prevention solution for sequential resource allocation systems. In *Proceedings of the ICRA 2005*, pages 271–277. IEEE, 2005.

[53] J. Tung, T. Sheen, M. Kao, and C. H. Chen. Optimization of AMHS design for a semiconductor foundry fab by using simulation modeling. In *Proceedings of the 2013 Winter Simulation Conference*, pages 3829–3839. INFORMS, 2013.

[54] W. Van der Aalst and K. Van Hee. *Workflow Management: Models, Methods and Systems*. The MIT Press, Cambridge, MA, 2002.

[55] V. Vazirani. *Approximation Algorithms*. Springer, NY,NY, 2003.

[56] N. Viswanadham, Y. Narahari, and T. L. Johnson. Deadlock avoidance in flexible manufacturing systems using Petri net models. *IEEE Trans. on Robotics and Automation*, 6:713–722, 1990.

[57] Y. Wang, T. Kelly, M. Kudlur, S. Lafortune, and S. Mahlke. Gadara: Dynamic deadlock avoidance for multithreaded programs. In *OSDI'08*, 2008.

[58] T. Yang and B. A. Peters. A spine layout design method for semiconductor fabrication facilities containing automated material-handling systems. *Int. J. Oper. & Prod. Man.*, 17:490–501, 1997.