Robust Deadlock Avoidance for Sequential Resource Allocation Systems with Resource Outages

Spyros Reveliotis and Zhennan Fei

***Abstract*—While the supervisory control (SC) problem of (maximally permissive) deadlock avoidance for sequential resource allocation systems (RAS) has been extensively studied in the literature, the corresponding results that are able to address potential resource outages are quite limited, both, in terms of their volume but also in terms of their control capability. This work leverages the recently developed SC theory for switched Discrete Event Systems (s-DES) in order to provide a novel systematic treatment of this more complicated version of the RAS deadlock avoidance problem. Following the modeling paradigm of s-DES, both, the operation of the considered RAS and the corresponding maximally permissive SC policy are decomposed over a number of operational modes that are defined by the running sets of the failing resources. In particular, the target supervisor must be decomposed to a set of "localized predicates", where each predicate is associated with one of the operational modes. A significant part, and a primary contribution, of this work concerns the development of these localized predicates that will enable the formal characterization and the effective computation of the sought supervisor. With these predicates available, a distributed representation for the sought supervisor, that is appropriate for real-time implementation, is eventually obtained through an adaptation of the relevant distributed algorithm that is provided by the current s-DES SC theory.**

***Note to Practitioners* – This paper extends the existing theory of deadlock avoidance for buffer-space allocation in flexibly automated production systems so that it accounts for disruptive effects due to potential temporary outages of some of the system servers. The set of the failing servers at any time instant defines the corresponding operational mode for the underlying resource allocation system (RAS). The primary problem that is addressed by this paper is the synthesis of a resource allocation policy that will ensure the ability of all process instances that do not require the failing resources in a particular mode, to execute repetitively and complete successfully while the system remains in that mode. In line with some past literature on this problem, we call the corresponding supervisory control problem as "robust deadlock avoidance", and we leverage results from the recently emerged theory for modeling and control of switched Discrete Event Systems (s-DES) in order to characterize and compute a maximally permissive solution for it.**

***Index Terms*—Resource Allocation System, Robust Deadlock Avoidance, Switched Discrete Event Systems, Decomposition Methods in DES Supervisory Control Theory**

S. Reveliotis is with the School of Industrial & Systems Engineering, Georgia Institute of Technology, email: spyros@isye.gatech.edu. Z. Fei is with Prover Technology, Stockholm, Sweden, email: zhennan.fei@gmail.com. S. Reveliotis was partially supported by NSF grant ECCS-1405156.

I. Introduction

Research Background and Motivation: The problem of deadlock avoidance in sequential resource allocation systems (RAS) is well documented and extensively studied in the literature [1], [2]. In its basic positioning, the problem concerns the staged allocation of a limited set of reusable resources to a set of concurrently executing processes in a way that all of these processes can secure the resources necessary for the execution of their various processing stages and terminate successfully. In this setting, deadlock avoidance policies (DAPs) prevent resource allocation states that can lead to "deadlocks", i.e., circular waiting patterns where a (sub-)set of processes is permanently stalled since they are waiting for the allocation of resources that are currently held by other processes in this set. Furthermore, one can seek the deployment of the *maximally permissive* DAP, i.e., a DAP that prevents effectively the formation of deadlock and at the same time imposes the minimal possible restriction on the underlying resource allocation process.

For the RAS classes studied in the past literature, the maximally permissive DAP is well-defined and unique. On the other hand, its computation is an NP-hard problem for most RAS instances [3], [4], and therefore, extensive effort has been expended on the design of suboptimal (i.e., non-maximally permissive) DAPs that retain a significant part of the behavioral latitude supported by the maximally permissive DAP [5], [6], [7]. Nevertheless, recent developments taking place in the Ph.D. theses of [8], [9] and their derivative publications[1] have established that the deployment of the maximally permissive DAP is still a tractable proposition for RAS instances of pretty high structural and behavioral complexity, through a two-step approach that isolates the hardest part of the computation of this policy in an "off-line" computational stage, and eventually encodes the obtained results as a "classifier" that dichotomizes the underlying RAS state set into its admissible and inadmissible subsets.[2] Finally, instrumental in all the aforementioned developments, is the classification of the various RAS structures according to the taxonomy that is presented in Table I. As seen in Table I, this taxonomy is defined on the basis of the structure of (i) the

[1]Among those publications, some of the most relevant to this work are those presented in [10], [11], [12]; see also [13] for an abridged exposition of all the corresponding results.

[2]In the relevant terminology, the RAS subset that contains the admissible (resp., inadmissible) states by the maximally permissive DAP is also known as the "safe" (resp., "unsafe") subset.

TABLE I: A RAS taxonomy [6]

Based on the Process Sequential Logic	**Based on 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

sequential logic that is observed by the RAS processes, and (ii) the resource requests that are posed by the various processing stages. The availability of this taxonomy has enabled (a) a more profound understanding of the dependence of the RAS deadlock avoidance problem and its complexity to the various structural and behavioral elements of the underlying RAS, and (b) the mastering of this complexity through a progressive study of the problem from its simpler to its more complicated cases.

When viewed from the standpoint of the aforementioned taxonomy, most of the past literature on the deadlock avoidance problem has assumed that the resources involved are imperishable and always available to the contesting processes. However, in many practical cases, resources might experience temporary but lengthy outages: machines might be down due to some failing components or lack of consumables, operators might be unavailable, segments of a material-handling network might be inaccessible due to maintenance or currently blocked, etc. The work that is presented in this paper seeks to develop a new methodology for effecting maximally permissive deadlock avoidance in RAS classes that experience some of the aforementioned problems. But in order to provide a more systematic positioning of the work and its intended contributions, first we review the existing literature on RAS deadlock avoidance under resource failures.[3]

Literature Review: Generally speaking, there are two major approaches for addressing the aforementioned resource outages with respect to (w.r.t.) the problem of RAS deadlock avoidance. The first approach is of a *reactive* nature, seeking to recompute the applied DAP upon the outage or the restoration of a resource unit, so that it reflects the actual resource availability. Such an approach presumes an ability to design the necessary DAPs on the fly, and furthermore, it might need to adjust the current RAS state by temporarily unloading some running processes, in order to establish operational feasibility in the emerged operational context.

An example application of the reactive approach to the RAS deadlock avoidance problem under resource failures, as it materializes in the operational context of flexibly automated production systems, is presented in [14]. On the other hand, a more abstract treatment of the corresponding policy-reconfiguration problem can be found in [15]. More precisely, the work of [15] represents the considered RAS dynamics in the Petri net (PN) modeling framework [16], and addresses the problem of enforcing "switching" specifications that take the form of sets of linear inequalities upon the corresponding PN marking. The results provide feasibility conditions for enforcing the corresponding control policies, and also a methodology for driving the net marking to the admissible region of the currently enforced specifications (provided that this task is feasible). But the paper does not make explicit connections of the presented results to the particular problem of accommodating resource outages.

A paper that uses the PN modeling framework in order to deal more directly with the problem of DAP reconfiguration under resource failures is that of [17]. This work uses the siphon-based characterization of liveness for the considered PN class, and the corresponding liveness-enforcing supervisory control theory that was developed in the works of [18] and [19], in order to determine "monitor"-based liveness-enforcing supervisors for these nets [20], [21], for each failing mode of the underlying RAS. On the other hand, this paper does not discuss the RAS transition between two consecutive supervisors, in the case that the underlying RAS state (or PN marking) is not compatible with the new supervisor at the switching point.

The second approach for addressing resource outages in the context of RAS deadlock avoidance is of a *proactive* nature as it seeks to control the initiation and the advancement of the running process instances through the underlying RAS in a way that any occurring resource outage will effect the minimal possible disruption in the operation of the underlying RAS, and it will be handled by the system itself without the need for any further external interference. More specifically, under this approach, we seek to (i) limit the impact of an occurring resource outage only to these running processes that must utilize the unavailable resource for the execution of some of their processing stages, and (ii) ensure the capability of the processes that do not engage the failing resource to execute "smoothly" through the system.[4] For these reasons, this approach has come to be known as *"robust"* deadlock avoidance.

The research on robust deadlock avoidance for sequential RAS was initiated with the seminal paper of [22]. This work introduced the aforementioned notion of "robustness" in the context of a RAS class that models the buffer allocation in flexibly automated production systems. The resulting SC problem was formally characterized using the Ramadge & Wonham (R&W) SC framework [23], and a computationally efficient solution for it was synthesized by adapting some sub-optimal DAPs for failure-free, Linear, Single-Unit RAS. On the other hand, these prototypical developments were restricted

[3]Following rather standard practice in the corresponding literature, in the sequel, we shall use the terms of "(resource) failures" and "(resource) outages" interchangeably.

[4]Part of the mission of this paper is to provide a succinct formal characterization of this operational requirement. The reader is referred to later parts of the paper for the corresponding details.

to automated production systems with a single failure-prone processor.

The work of [22] was subsequently extended in [24], that addressed RAS modeling the operations of automated production systems with more than one failure-prone processors. The results of [24] are of similar flavor to the results of [22] in that they provide suboptimal solutions to the formulated "robust" deadlock avoidance problem by adapting ideas and techniques from classical deadlock avoidance theory to this particular context. Furthermore, all the developments of [24] are contingent upon the condition that each process type utilizes at most one of the failing processors. More recently, the works appearing in [25], [26], [27], [28] have tried to extend further the developments of [22], [24] to some RAS classes that are characterized by increased routing flexibility and / or additional special structure that impacts the underlying notion of RAS state safety. This new set of results employ a PN-based modeling framework, instead of the Finite State Automata (FSA) that were used in the original works of [22], [24], and the synthesized "robust" DAPs are employing insights and results regarding the synthesis of liveness-enforcing supervisors (LES) for RAS-modeling PNs. But, at the end, even in these new developments, the derived DAPs remain of a suboptimal nature, and their functionality is contingent upon specific structural and/or operational assumptions for the underlying RAS that enable their synthesis.

Finally, another line of work on the problem of "robust" deadlock avoidance is that presented in [29], [30]. These two works seek to establish deadlock-free resource allocation in the context of various types of automated manufacturing systems, that is also "robust" to resource failures, by enforcing some elementary sufficient conditions on the resource availability that ensure the execution of any single process type in isolation. However, the resulting policies can be very conservative, to the point that they might not even be implementable in the context of RAS with low resource availabilities on some critical resources. On the other hand, the aforementioned papers fail to acknowledge this rather restrictive feature and to provide the corresponding feasibility analysis.

Recapitulating the above discussion, we can conclude that the literature on RAS deadlock avoidance in the presence of resource failures is not as well developed as the corresponding theory for failure-free RAS. In particular, when it comes to the concept of "robust" deadlock avoidance, the relevant results are dominated by the modeling–&–analysis paradigm that was set by [22], [24], and they are generally characterized by: (i) limited applicability to failure-prone RAS that satisfy particular restrictions on their structure and / or the patterns of the experienced resource failures; and (ii) a suboptimal (i.e., non-maximally permissive) nature for the derived DAPs.

The paper content and the intended contributions: Motivated by the closing remarks in the previous paragraph, in this work we propose an alternative paradigm for addressing the basic problem of "robust" deadlock avoidance for the buffer allocation of automated production systems defined in the seminal works of [22], [24], while overcoming the limitations that are present in all the relevant past works. This new paradigm is grounded on the theory of supervisory control of switched discrete event systems (s-DES), that was developed by the paper authors in [31], [32]. Hence, according to this new paradigm, the overall DAP-synthesis process is decomposed across a number of operational "modes" for the underlying RAS, that are naturally defined by the various subsets of failing resources, and the corresponding controller is synthesized by a distributed algorithm [33] that employs one computational thread for each mode and a message-passing mechanism that ensures the correctness of the entire outcome.

From a representational standpoint, the derived DAP is also distributed across the aforementioned operational modes, and therefore, it is effectively computable and executable through the methods of [10], [11], [12] that have been employed for the deployment of the maximally permissive DAP in failure-free RAS. On the other hand, a critical issue for the effective implementation of this new computational scheme, according to the theory of [31], [32], is the ability to express the defining logic for the target policy through a set of *"localized predicates"*, one predicate for each operational mode. But the development of such a set of localized predicates that will be practically useful in the context of the aforementioned computation is a quite non-trivial task; hence, the effective resolution of this problem is at the core of the presented developments.

Once the above issue has been resolved, the sought DAP can be computed through an adaptation of the distributed computation scheme that is presented in [31], [32]. The corresponding distributed algorithm of [31], [32] needs to be extended so that it can address modal "non-blockingness" requirements, besides the modal "safety" specifications that were addressed in [31], [32].[5] We provide the necessary modifications in the corresponding part of the paper; as it will be seen in that part, these modifications are based on classical perspectives and results borrowed from DES SC theory [23], [34], and they complete the existing SC theory for s-DES in the aforementioned directions.

In view of the above positioning of the paper content and contributions, the rest of it is organized as follows: Section II introduces the considered RAS structure, its connection to the automated production systems that were mentioned in the previous paragraphs, and the corresponding robust deadlock avoidance problem. Section III represents the operation of the considered RAS as an s-DES, and subsequently Section IV characterizes the maximally permissive DAP for these RAS in this new modeling framework. Section V presents the aforementioned distributed algorithm for the computation of the maximally permissive DAP. Finally, Section VI concludes the paper and highlights some directions for further extensions of the presented work.[6]

[5]The notions of "non-blockingness" and "safety" requirements should be understood in the spirit of the R&W-SC framework [23].

[6]We also notice, for completeness, that a preliminary version of the presented results can be found in [35]; the material of that paper develops at a more conceptual level and lacks the thorough technical analysis that is presented in Sections III, IV and V of this paper.

II. The considered RAS and the corresponding deadlock avoidance problem

In this section we provide a more systematic characterization of the RAS class and the corresponding "robust" deadlock avoidance problem that we consider in this work. Both of these elements are equivalent to the corresponding elements that are defined in [24]. However, the subsequent discussion leverages concepts and results that relate to the RAS taxonomy of Table I. In this way, the presented developments will be positioned more succinctly in the broader context of the existing literature on the RAS deadlock avoidance, and there is a further potential for the re-interpretation and extension of these developments in the context of the more general abstractions that underlie this taxonomy.

The Linear Single-Unit (L-SU) RAS and its connection to automated production systems: The L-SU RAS was defined in Table I as the RAS class where (i) each process type is defined by a simple sequence of processing stages, and (ii) every processing stage requires the allocation of a single unit from a single resource type for its successful execution. A more complete definition of this RAS class, which is in line with the general definition of the sequential RAS concept that is provided in [6], is as follows:

Definition 1: A *Linear Single-Unit (L-SU)* RAS is formally defined by a 4-tuple $\Phi = \langle \mathcal{R}, C, \mathcal{P}, \mathcal{A} \rangle$ where:

1) $\mathcal{R} = \{R_1, \ldots, R_m\}$ is the set of the system *resource types*.
2) $C : \mathcal{R} \to \mathbb{Z}^+$ – where $\mathbb{Z}^+$ is 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 R_i.
3) $\mathcal{P} = \{J_1, \ldots, J_n\}$ denotes the set of the *process types* supported by the considered RAS. Each process type J_j, $j = 1, \ldots, n$, constitutes a *totally ordered* set of *processing stages*, i.e., $J_j = \langle \mathcal{S}_j \rangle = \langle \Xi_{j1}, \ldots, \Xi_{j,l(j)} \rangle$.
4) $\mathcal{A} : \bigcup_{j=1}^{n} \mathcal{S}_j \to \mathcal{R}$ is the *resource allocation function*, which associates every processing stage Ξ_{jk} with the *resource allocation request* $\mathcal{A}(j,k) \equiv \mathcal{A}_{jk}$. More specifically, each $\mathcal{A}_{jk}$ is an element of the resource set $\mathcal{R}$, the implication being that processing stage Ξ_{jk} requires the exclusive allocation of one unit from the corresponding resource type for its execution.
5) Finally, according to the applying resource allocation protocol, a process instance executing a processing stage Ξ_{jk} will be able to advance to its successor processing stage $\Xi_{j,k+1}$ only after it has been allocated the required unit of the corresponding resource type $\mathcal{A}_{j,k+1}$; and it is only upon this advancement that the process will release the currently held unit of resource type $\mathcal{A}_{jk}$.

The combination of items (3) and (4) in the above definition further implies that, in the L-SU RAS context, each process type J_j can be described by the sequence of the resource types that are supporting the execution of the corresponding sequence of processing stages $\mathcal{S}_j$; we shall refer to this resource sequence as the *"process plan"* of process type J_j. Furthermore, in the sequel, we shall characterize as the *support* $Sup(R_i)$ of resource R_i, $i = 1, \ldots, m$, the set of processing stages Ξ_{jk} that request the allocation of one unit from resource R_i for their execution.

The L-SU RAS model has been used extensively for the modeling and analysis of the deadlock-related problems that arise in the buffer allocation of many contemporary automated production cells. In this modeling paradigm, each resource type R_i, $i = 1, \ldots, m$, represents the buffering capacity of the corresponding workstation WS_i. Then, the resource allocation protocol that is defined in item (5) of Definition 1 is justified by the basic fact that each process going through the cell, being a physical entity, must always be staged at a buffer slot of some workstation; in particular, a process cannot release its currently allocated buffer slot unless it has secured a buffer slot at the next requested workstation.[7]

Introducing "resource failures": In the automated production systems that are considered in the above discussion, each workstation, besides its buffering capacity, also possesses a server that performs the processing that takes place at that workstation. And while these servers are not modeled as distinct entities in the standard L-SU RAS modeling paradigm, they are assumed to work on the visited processes sequentially, one at a time, according to a globally non-idling scheme.[8] Furthermore, processes retain their designated buffer slot while in processing; in particular, it can be assumed that either they receive service in situ, or they are processed in a separate chamber but they return to their allocated buffer slot upon the completion of their processing.

In either case, under the aforestated assumptions, the processes staged at any workstation WS_i, $i = 1, \ldots, m$, can be distinguished to (i) those waiting for processing, (ii) the one in processing, and (iii) those having completed processing and waiting for their advancement to the next workstation. Furthermore, for the purposes of deadlock analysis and control, it is also natural to assume that processes having completed their last processing stage are unloaded from the system immediately upon the completion of the corresponding service.

In the classical deadlock avoidance theory, the discrimination of the process instances that are staged at a particular workstation along the aforementioned lines is not significant since the progression of these process instances through the corresponding phases does not alter the underlying resource allocation. But as we shall see next, this discrimination is important in the context of the "robust" deadlock avoidance

[7]Under this interpretation of the resource allocation taking place in an L-SU RAS, the process transfer between two consecutive workstations, and the facilitating role of the underlying material handling system, are considered only implicitly. Alternatively, the necessary material handling steps can be considered explicitly in the employed L-SU RAS model, by modeling these steps as additional processing stages requiring a unit of buffering capacity on the corresponding material handling system, which should be modeled as an additional resource type.

[8]In the standard L-SU RAS operational context, "global non-idleness" means that when the system is not empty, at least one of the system servers must be busy, and this requirement ensures the system progress towards the completion of all the initiated process instances.

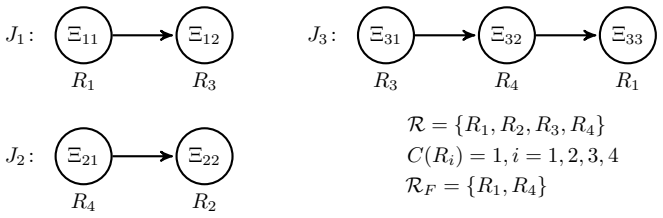

Fig. 1: The F-L-SU RAS instance considered in the provided example. Resources R_1 and R_4 might experience temporary outages, while resources R_2 and R_3 are assumed to never fail.

problem that is considered in this work.

More specifically, in this paper we assume that the servers of certain workstations may experience a non-catastrophic failure while working on some process instance, and they will stay in this failing mode for a random (and possibly extensive) time until they are eventually repaired and resume service on the interrupted process instances. We formalize this additional operational feature by extending Definition 1 to include information about these failing possibilities.

Definition 2: An *L-SU RAS with resource outages*, to be denoted by F-L-SU RAS in the sequel, is formally defined by a 5-tuple $\Phi = \langle \mathcal{R}, C, \mathcal{P}, \mathcal{A}, \mathcal{R}_F \rangle$ where:

1) The quadruple $\langle \mathcal{R}, C, \mathcal{P}, \mathcal{A} \rangle$ defines an L-SU-RAS according to the logic of Definition 1.
2) The set $\mathcal{R}_F \subseteq \mathcal{R}$ defines the set of the resource types corresponding to workstations whose servers can experience temporary outages.

In the acronym F-L-SU RAS, 'F' stands for "failing". Also, for the sake of brevity and notational simplicity, in the sequel we shall not distinguish between the resource type $R_i,\ i = 1, \ldots, m$, and its corresponding server; hence, we shall talk about "failing resources". The next definition introduces a natural, yet formal, concept for tracing the server operational status in any given F-L-SU RAS, and it has a central role in all the subsequent developments.

Definition 3: Given a F-L-SU RAS Φ, the set of failing servers at any time point t will define the current *failing mode* $x(t)$ of this RAS.

Example 1: We concretize the notions of the F-L-SU RAS and the corresponding failing modes that are defined in Definitions 2 and 3, by means of the F-L-SU RAS instance that is depicted in Figure 1. This example RAS will also be used in the rest of the manuscript for highlighting all the technical concepts and results that are presented in it. The depicted RAS consists of three process types J_1, J_2 and J_3. Among these process types, J_1 and J_2 are defined as sequences of two processing stages that are denoted by Ξ_{jk} where $j = 1, 2$ and $k = 1, 2$. On the other hand, process type J_3 is defined by a sequence of three processing stages that are denoted by Ξ_{31}, Ξ_{32} and Ξ_{33}. The system resource set is $\mathcal{R} = \{R_1, R_2, R_3, R_4\}$, and each resource R_i has a (buffering) capacity $C_i = 1$. Each processing stage requests only one unit from a single resource type; the corresponding resource allocation function is depicted in Fig. 1. Moreover, the servers that are possessed by the workstations modeled by the resource types R_1 and R_4 might experience temporary outages. If an outage occurs at a server while working on a process instance executing one of the supported stages, the server will remain at this failing status until it is restored and resumes service on the interrupted process. Hence, the F-L-SU RAS instance considered in this example has three failing modes, corresponding to the outage of either R_1 or R_4, and the simultaneous outage of both R_1 and R_4.

Maximally permissive robust deadlock avoidance for the F-L-SU RAS: Clearly, at any given failing mode $x(t)$, a process instance j_j being processed or waiting for service at a workstation with a failing server cannot proceed to its completion until the restoration of the functionality of this server (and the switching of the RAS to a different failing mode $x'(t)$). Furthermore, in a failing mode $x(t)$ with a failing resource type R_i, any process instance with its (remaining) process plan involving resource type R_i will not be able to complete. But, more generally, the inability of the aforementioned process instances to advance past a certain stage of their corresponding process plans might also lead to the blockage of additional process instances, that are executable by the non-failing resources in mode $x(t)$, but they cannot reach (some of) these resources due to the presence of the first set of process instances.[9]

Following [22], [24], we define the problem of *(maximally permissive) robust deadlock avoidance* for the considered RAS class as the corresponding SC problem that prevents the development of such second-order blockages in the operation of the considered RAS, at any of its failing modes $x(t)$. More specifically, we stipulate that *at any failing mode $x(t)$ of the considered F-L-SU-RAS, all those process types J_j with process plans executable, in isolation, by the non-failing resources in that mode, should retain their ability to be repeatedly activated and complete all their activated process instances while the system remains in the considered failing mode. Furthermore, we would like to attain the above objectives while imposing the minimal possible restriction upon the feasible operation of the underlying RAS.*

In the rest of this paper, we provide a compete formulation and a systematic solution of the aforestated RAS SC problem by adapting to this problem the notion of the s-DES and the relevant SC theory that were developed in [31], [32].

III. Modeling the F-L-SU RAS as a switched Discrete Event System

In this section we provide a formal characterization of the qualitative dynamics of the F-L-SU RAS in the s-DES modeling framework that was recently introduced in [31], [32]. We start these developments with a brief overview of the s-DES concept and its key defining ingredients.

Definition 4: According to [31], [32], a *switched Discrete Event System*, briefly annotated as s-DES, is formally defined by a 4-tuple $\mathcal{G} \equiv \langle X \times S,\ \Sigma \cup E,\ \delta,\ (x_0, s_0) \rangle$ where:

1) The finite state set S defines the set of operational *states* of $\mathcal{G}$, and the finite set X distinguishes a number of operational *modes* over S. The tuples $(x, s) \in X \times S$ are

[9]The reader can refer to Section III of [24] for some vivid examples of these blocking effects.

characterized as the *global state* of $\mathcal{G}$. On the other hand, the state (x, s) for a fixed mode $x \in X$ is perceived as a *local state* in that mode, and, in the absence of any ambiguity, it will be represented only by its second component s.

2) The finite set $\Sigma \cup E$ defines the *events* taking place in $\mathcal{G}$, and it distinguishes these events into two types: (i) The events in set Σ trigger transitions of $\mathcal{G}$ within its operational state space S, without altering its mode. (ii) The events in set E are all those events that cause a modal change in $\mathcal{G}$.
Furthermore, the set Σ is partitioned into the sets Σ^c and Σ^u defining, respectively, the sets of *controllable* and *uncontrollable* events in Σ. On the other hand, all the events in E are *uncontrollable* events.
3) The state transition function $\delta : X \times S \times (\Sigma \cup E) \rightarrow 2^{X \times S}$ describes the transition of $\mathcal{G}$ among its various global states $(x, s) \in X \times S$ upon the occurrence of the various events in $\Sigma \cup E$. Moreover, the state transition function δ is recursively extended to strings of $(\Sigma \cup E)^*$, the Kleene closure of $\Sigma \cup E$; for the sake of simplicity, we denote the extended transition function by δ, as well. Furthermore, for the needs of the subsequent developments, it is also useful to consider the restrictions of the transition function δ to its domain subsets $\{x\} \times S \times \Sigma$ and $\{x\} \times S \times E$, for any given mode $x \in X$. The first type of these restrictions encodes the transitions taking place in the local subspace that corresponds to mode x, and it will be denoted by δ_x. Furthermore, for representational economy, δ_x will be considered as reduced to a two-argument function, defined on $S \times \Sigma$.[10]
On the other hand, the restriction of the function δ to $\{x\} \times S \times E$ encodes the uncontrollable transitions in mode x that result in a mode change. This restriction will be denoted by δ_x^E, and similar to the case of the function δ_x, δ_x^E will also be considered as a two-argument function, defined on $S \times E$.
4) Finally, the pair (x_0, s_0) initializes $\mathcal{G}$ by specifying an initial mode $x_0 \in X$ and an initial state $s_0 \in S$.

Modeling the F-L-SU RAS as an s-DES: Let us consider a RAS $\Phi = \langle \mathcal{R}, C, \mathcal{P}, \mathcal{A}, \mathcal{R}_F \rangle$, belonging to the RAS class of Definition 2. The qualitative dynamics of this RAS can be modeled by a s-DES $\mathcal{G}(\Phi) = \langle X \times S,\ \Sigma \cup E,\ \delta,\ (x_0, s_0) \rangle$, as follows:

1) The (local) state set S of $\mathcal{G}(\Phi)$ characterizes the distribution of the running process instances to their corresponding processing stages, also taking into consideration the processing status of each process instance in its current processing stage. A formal encoding of this "state" concept can be obtained as follows:
Let $\xi \equiv \sum_{j=1}^{n} |\mathcal{S}_j|$, where we remind the reader that $\mathcal{S}_j,\ j = 1, \ldots, n$, denotes the set of the processing stages of process type J_j, and the application of the operator $|\cdot|$ on a given set returns its cardinality. Then, assuming that a process instance that has executed its last processing stage is unloaded immediately from the system, we take $S \subset (Z_0^+)^{3\xi - n}$. Each component of the vectors $s \in S$ corresponds to a pair of a processing stage $\Xi_{jk} \in \bigcup_{j=1}^{n} \mathcal{S}_j$ together with a particular processing status; for further reference, we shall indicate these possible statuses by I – waiting for processing, II – in processing, and III – waiting to be transferred to the next workstation. Furthermore, the proposed state representation accounts for the fact that a job that has completed its last processing stage cannot be in status III, and this explains the term '$-n$' in the definition of the dimensionality of the local state vector s.
For a complete characterization of the target set S, vectors $s \in S$ must be further qualified to ensure the feasibility of the implied resource allocation. This can be achieved by enforcing the following set of linear inequalities on s:

$$\forall i \in \{1, \ldots, m\}, \quad \sum_{q:\mathtt{res}(s[q])=R_i} s[q] \ \leq \ C_i \qquad (1)$$

$$\forall i \in \{1, \ldots, m\}, \quad \sum_{q:\mathtt{res}(s[q])=R_i \wedge \mathtt{stat}(s[q])=\mathtt{II}} s[q] \ \leq \ 1 \qquad (2)$$

The notation $\mathtt{res}(s[q])$ in the above equation implies a function returning the resource type that supports the processing stage corresponding to the state component $s[q]$. Also, the notation $\mathtt{stat}(s[q])$ is a function that returns the processing status of the process instances corresponding to the state component $s[q]$; the range of this function is the set $\{I, II, III\}$, with an interpretation of these values as discussed in the previous paragraphs. Then, it is clear that a vector s satisfies the inequalities of Eq. 1 if and only if (*iff*) it corresponds to a feasible resource allocation w.r.t. the resource capacities of the RAS Φ. Similarly, the inequalities of Eq. 2 enforce the working assumption that every workstation $WS_i,\ i = 1, \ldots, m$, of the underlying production system possesses a single server.[11]
2) The mode set X of $\mathcal{G}(\Phi)$ is defined by the power set of the set $\mathcal{R}_F$ that contains the failure-prone resources. Hence, every mode $x \in X$ corresponds to a subset of $\mathcal{R}_F$.
Furthermore, the working assumption that a workstation server can fail only while processing a process instance, implies the following additional condition for any global

[10] Formally, the reduction of δ_x from a three-argument to a two-argument function, can be obtained through the *existential quantification* of its first (constant) argument [36].

[11] The above discussion has provided a *canonical* characterization of the local state s for the considered s-DES $\mathcal{G}(\Phi)$. However, some reflection on the material that is provided in the rest of this section will reveal that the tracing of the processing status of the active process instances is necessary only for those processing stages that are supported by failure-prone resources; this realization can incur a possible reduction in the dimensionality of the employed state vector s. An additional reduction in the dimensionality of s can be incurred by the realization that failure-prone resources with a single unit of capacity cannot have active process instances waiting for processing; in these resources, process instances should be either in processing or blocked waiting for transfer to their next workstation.

state $(x, s) \in X \times S$:

$$\forall (x,s) \in X \times S,\ \forall R_i \in x, \sum_{q:\mathtt{res}(s[q])=R_i \wedge \mathtt{stat}(s[q])=\mathtt{II}} s[q] \ = \ 1 \qquad (3)$$

3) The set Σ contains the events that evolve the local state s of $\mathcal{G}(\Phi)$. More specifically, Σ contains all the process initiation, single-stage advancement, and completion events that are implied by the sequential logic that is presented in Definition 1, as well as the service initiation and completion events within a particular stage. In the considered application context, all these events are assumed to be controllable except for the events that correspond to service completion.
4) On the other hand, the set E contains all the possible events that correspond to the failure or the restoration of any single resource in $\mathcal{R}_F$. All the events in E are uncontrollable.
5) The (global) state transition function δ encodes the qualitative dynamics of the s-DES $\mathcal{G}(\Phi)$ as implied by the above definitions of the sets S, X, Σ and E. It should be clear from these definitions that, in the considered application context, δ is deterministic, i.e., it maps every element of $(x, s, q) \in X \times S \times (\Sigma \cup E)$ to a single state (x', s'). Furthermore, such a transition is feasible only if the resultant state (x', s') satisfies the constraints of Eqs 1–3.
6) Finally, the initial global state (x_0, s_0) of $\mathcal{G}(\Phi)$ is naturally defined by the state $(\emptyset, \mathbf{0})$, i.e., the state where all the system servers are functional, and the system is empty of any process instances.

Some further notions of particular significance in DES SC theory at large, and also in the theory of deadlock avoidance for sequential RAS, are those of *state reachability* and the *reachable subspace* of any given DES. In the operational context of the s-DES $\mathcal{G}(\Phi)$ that was defined in the previous paragraphs, a global state (x', s') is reachable from a global state (x, s) if there exists a feasible event sequence $\sigma \in (\Sigma \cup E)^*$ such that $\delta(x, s, \sigma) = (x', s')$. Furthermore, the reachable (global) subspace of $\mathcal{G}(\Phi)$ is defined by $Reach(\mathcal{G}(\Phi)) \equiv \{(x, s) \in X \times S : \ \exists \sigma \in (\Sigma \cup E)^* \text{ s.t. } (x, s) = \delta(x_0, s_0, \sigma)\}$. Finally, similar notions of reachability can be defined by means of the restrictions δ_x and δ_x^E of the transition function δ, that were introduced in the earlier parts of this section; the corresponding definitions are straightforward and their detailing is left to the reader.

Example 1 (cont.): We demonstrate the concepts involved in the above definition of the s-DES $\mathcal{G}(\Phi)$ and its reachable state space, $Reach(\mathcal{G}(\Phi))$, by means of the example F-L-SU RAS Φ that was introduced in Section II.

The mode set X of the s-DES $\mathcal{G}(\Phi)$ modeling the considered RAS Φ is equal to the set $\{x_0, x_1, x_2, x_3\}$ where: $x_0 = \emptyset$; $x_1 = \{R_1\}$; $x_2 = \{R_4\}$; and $x_3 = \{R_1, R_4\}$. On the other hand, the local state set S characterizes the space of all the possible allocation patterns of the capacity C_i of each resource type R_i to the processing stages in the corresponding support $Sup(R_i)$. As explained in the earlier parts of this section, all these allocation patterns can be pertinently represented by a vector s that contains one component for every combination of a processing stage with a processing status within this stage, except for the combinations that involve a terminal processing stage with a processing status of type III. In the context of the considered example, this modeling approach implies a (local) state vector s of dimensionality $3 \times 7 - 3 = 18$.

However, it is possible to reduce drastically the dimensionality of the employed state vector s by taking advantage of the observations that are provided in Footnote 11, and some additional structure that is present in the considered RAS. More specifically, since resources R_2 and R_3 are not in $\mathcal{R}_F$, the processing stages Ξ_{12}, Ξ_{22} and Ξ_{31} that are supported by these two resources can be represented by a single variable in s instead of three. In addition, the last remark in Footnote 11 further implies that the stages Ξ_{11}, Ξ_{21}, Ξ_{32} and Ξ_{33}, that are supported by the failure-prone resources R_1 and R_4, can be represented by two state variables in s instead of three. Hence, the above remarks lead to the following representation of the local state s for the considered example: $s = (v_{11,II}, v_{11,III}, v_{12}, v_{21,II}, v_{21,III}, v_{22}, v_{31}, v_{32,II}, v_{32,III}, v_{33,II})$ where the state variables v are indexed by the corresponding processing stages and statuses. Furthermore, since those process instances that have completed processing at the processing stages Ξ_{12} and Ξ_{22} can leave the system without experiencing any resource outages or posing further resource requests, the state variables that correspond to these two processing stages can be dropped completely from the employed state representation when reasoning about (robust) deadlock avoidance. And a similar remark applies to process instances that have completed processing in stage Ξ_{21} since the resource R_2 in their remaining process plan can be engaged neither in a deadlock nor in a resource outage. Hence, the local state vector s is further reduced to $s = (v_{11,II}, v_{11,III}, v_{21,II}, v_{31}, v_{32,II}, v_{32,III}, v_{33,II})$. Table II lists the state vectors for the considered example F-L-SU RAS.

The initial global state (x_0, s_0) of the considered s-DES $\mathcal{G}(\Phi)$ is the state $(\emptyset, \mathbf{0})$ where all the servers are functional and the system is idle and empty of any process instances.

The various events that evolve the state of $\mathcal{G}(\Phi)$ can be classified as follows: (i) First, there is the event set Σ consisting of all those events that activate the running process instances and advance them through the refined processing stages that are recognized by the local state vector s that was defined in the previous paragraphs. These events affect only the local state s of $\mathcal{G}(\Phi)$, and they are assumed to be controllable except for the events that correspond to the transitions $\langle \Xi_{11,II}, \Xi_{11,III} \rangle$ and $\langle \Xi_{32,II}, \Xi_{32,III} \rangle$, i.e., the service completions in the corresponding processing stages Ξ_{11} and Ξ_{32}. Furthermore, since there are no state variables tracking the number of process instances in the terminal stages Ξ_{12}, $\Xi_{21,III}$ and Ξ_{22}, the unloading events for process types J_1 and J_2 are modeled implicitly through the events $\langle \Xi_{11,III}, \Xi_{12} \rangle$ and $\langle \Xi_{21,II}, \Xi_{21,III} \rangle$. (ii) The second class of events taking place in the considered s-DES $\mathcal{G}(\Phi)$ is the event set $E = \{\mathtt{R_1\_fail}, \mathtt{R_4\_fail}, \mathtt{R_1\_repair}, \mathtt{R_4\_repair}\}$, that contains all the resource failing and restoration events. The

TABLE II: The detailed description of the RAS states appearing in Figs 2–5.

State	$(v_{11,II}, v_{11,III}, v_{21,II}, v_{31}, v_{32,II}, v_{32,III}, v_{33,II})$
$\mathbf{s_0}$	$(0,0,0,0,0,0,0)$
$\mathbf{s_1}$	$(1,0,0,0,0,0,0)$
$\mathbf{s_2}$	$(0,0,1,0,0,0,0)$
$\mathbf{s_3}$	$(0,0,0,1,0,0,0)$
$\mathbf{s_4}$	$(0,1,0,0,0,0,0)$
$\mathbf{s_5}$	$(1,0,1,0,0,0,0)$
$\mathbf{s_6}$	$(1,0,0,1,0,0,0)$
$\mathbf{s_7}$	$(0,0,1,1,0,0,0)$
$\mathbf{s_8}$	$(0,0,0,0,1,0,0)$
$\mathbf{s_9}$	$(0,1,1,0,0,0,0)$
$\mathbf{s_{10}}$	$(0,1,0,1,0,0,0)$
$\mathbf{s_{11}}$	$(1,0,1,1,0,0,0)$
$\mathbf{s_{12}}$	$(1,0,0,0,1,0,0)$
$\mathbf{s_{13}}$	$(0,0,0,1,1,0,0)$
$\mathbf{s_{14}}$	$(0,0,0,0,0,1,0)$
$\mathbf{s_{15}}$	$(0,1,1,1,0,0,0)$
$\mathbf{s_{16}}$	$(0,1,0,0,1,0,0)$
$\mathbf{s_{17}}$	$(1,0,0,1,1,0,0)$
$\mathbf{s_{18}}$	$(1,0,0,0,0,1,0)$
$\mathbf{s_{19}}$	$(0,0,0,1,0,1,0)$
$\mathbf{s_{20}}$	$(0,0,0,0,0,0,1)$
$\mathbf{s_{21}}$	$(0,1,0,1,1,0,0)$
$\mathbf{s_{22}}$	$(0,1,0,0,0,1,0)$
$\mathbf{s_{23}}$	$(1,0,0,1,0,1,0)$
$\mathbf{s_{24}}$	$(0,0,0,1,0,0,1)$
$\mathbf{s_{25}}$	$(0,0,1,0,0,0,1)$
$\mathbf{s_{26}}$	$(0,1,0,1,0,1,0)$
$\mathbf{s_{27}}$	$(0,0,1,1,0,0,1)$
$\mathbf{s_{28}}$	$(0,0,0,0,1,0,1)$
$\mathbf{s_{29}}$	$(0,0,0,1,1,0,1)$
$\mathbf{s_{30}}$	$(0,0,0,0,0,1,1)$
$\mathbf{s_{31}}$	$(0,0,0,1,0,1,1)$

occurrence of any of these events switches the mode x of the s-DES $\mathcal{G}(\Phi)$ but it does not alter its local state s, since it does not impact the processing stage and status of the running processes. All the events in the set E are uncontrollable events.

The transition function δ for the considered s-DES $\mathcal{G}(\Phi)$ is graphically represented by the finite state automata (FSA) depicted in Figs 2 – 5. These automata represent only the dynamics that take place in $Reach(\mathcal{G}(\Phi))$, the reachable subspace of the s-DES $\mathcal{G}(\Phi)$, and each of the provided figures depicts the state transition diagram (STD) that represents the local dynamics in the corresponding operational mode x_i, $i = 0,1,2,3$. The uncontrollable transitions among these automata, due to the occurrence of the events in the set E, are identified in the legends of the corresponding figures.

IV. A FORMAL CHARACTERIZATION OF THE MAXIMALLY PERMISSIVE ROBUST DAP FOR THE F-L-SU RAS

With the operational dynamics of the F-L-SU RAS Φ well-defined through the s-DES $\mathcal{G}(\Phi)$, next we turn to the formal specification of the notion of the "maximally permissive robust DAP" for this RAS class. An informal characterization of this last concept was already provided in the closing part of Section II. Following the spirit of [31], [32], here we shall seek to take advantage of the notions of "modality" and "locality" that are introduced by the s-DES concept, and distribute the control specification and the computation of the corresponding supervisor across the modes that are recognized in the operation of the underlying DES. This intention is further facilitated by the fact that the control specifications that are (verbally) stated at the end of Section II have, indeed, such a distributed character. More specifically, in the operational context of the s-DES $\mathcal{G}(\Phi)$, those earlier specifications boil down to the following two operational requirements for each mode $x \in X$ of the s-DES $\mathcal{G}(\Phi)$:

Requirement 1: In any given mode x, activated process instances that do not require any failing resource in that mode for the completion of their remaining process plans, must be able to advance to their completion.

Requirement 2: For any given mode x, let $\mathcal{P}(x) \subseteq \mathcal{P}$ denote the set of those process types that do not require any of the failing resources for the support of their corresponding process plans. Then, any process type $J_j \in \mathcal{P}(x)$ must be able to execute repetitively while the system remains in that mode.

In the FSA modeling framework that represents the modal qualitative dynamics of the s-DES $\mathcal{G}(\Phi)$, the above two requirements can be expressed by associating a corresponding set of *"marked states"* with each mode $x \in X$. Marked states is a standard method for expressing control specifications in the DES SC theory that uses an automata-based representation for the underlying dynamics [34]. Under this paradigm, the specifications are enforced by the *"co-reachability"* requirement that the specified set of marked states is reachable from any state that is reachable in the operation of the controlled DES, in spite of potential complications that might arise from the presence of uncontrollable behavior in the DES dynamics. The synthesis of the necessary supervisor that will enforce the aforestated "co-reachability" requirement in DES that are modeled by basic finite state automata, has been well studied by the DES SC theory [23], [34]. In this work, we shall capitalize upon those past results in later parts of this section and in Section V, where we shall address the computation of the maximally permissive DAP for the considered F-L-SU RAS.

Next, we detail a number of predicates that will define the local marked states that are in agreement with Requirements 1 and 2, at each mode $x \in X$ of the considered RAS. To formally define these predicates, first we need to introduce some further notation. In particular, for any given resource R_i, $i = 1,\ldots,m$, we define $ESup(R_i)$ – the *extended support* of R_i – as the set of processing stages Ξ_{jk} that are either supported by R_i, or they have remaining process plans that contain R_i. We also define, for each mode $x \in X$, the following (index) subsets $\mathcal{I}(x)$ of the components of the local

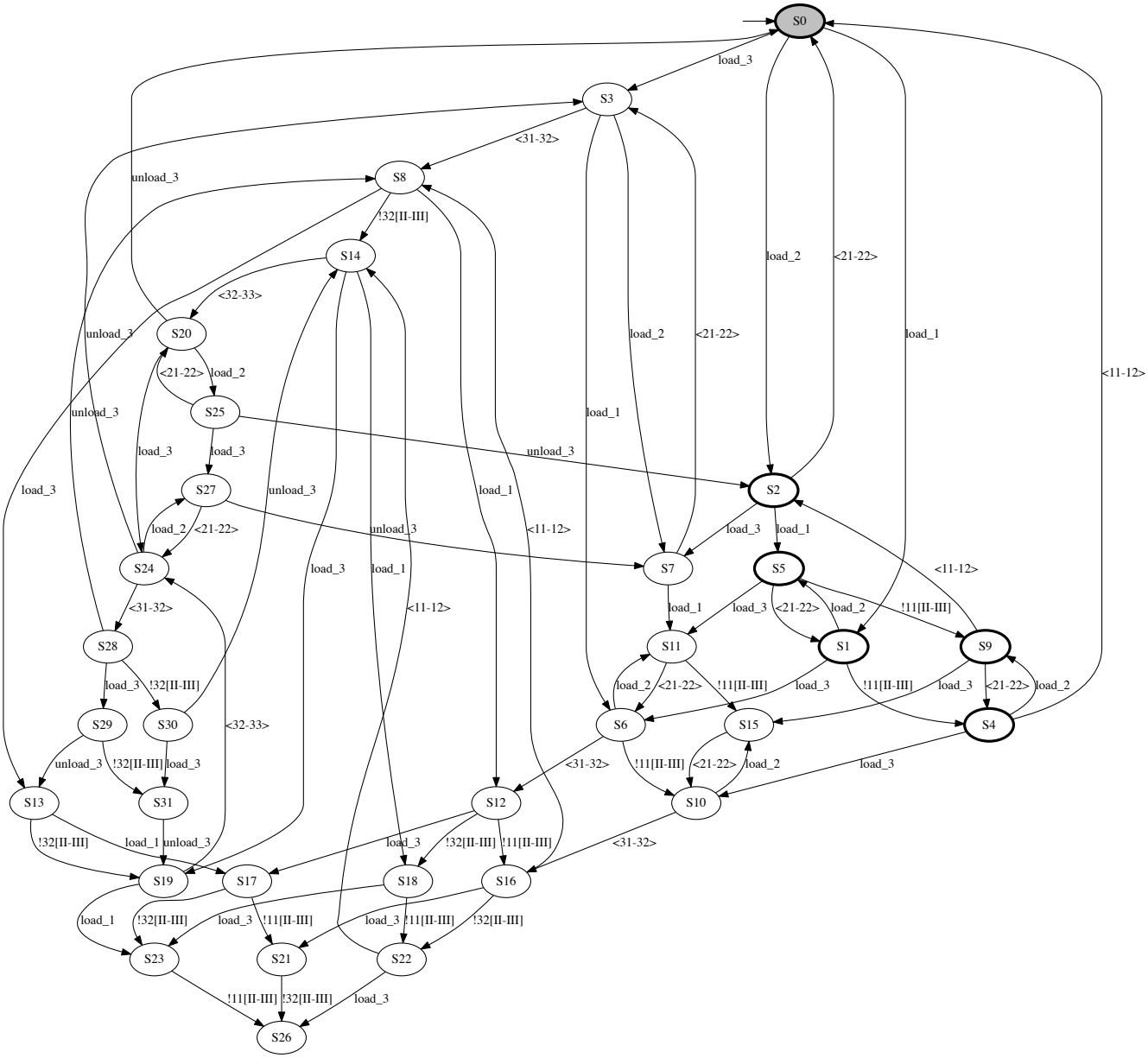

Fig. 2: The finite state automaton modeling the qualitative dynamics of the example F-L-SU RAS of Fig. 1 in mode x_0. A detailed description of each of the 32 states that are depicted in this automaton, is provided in Table II; and the same description applies also to the states that appear in Figs 3–5. Furthermore, states (x_0, s) in the depicted automation where $s \in \{\mathbf{s_1}, \mathbf{s_5}, \mathbf{s_6}, \mathbf{s_{11}}, \mathbf{s_{12}}, \mathbf{s_{17}}, \mathbf{s_{18}}, \mathbf{s_{20}}, \mathbf{s_{23}}, \mathbf{s_{24}}, \mathbf{s_{25}}, \mathbf{s_{27}}, \mathbf{s_{28}}, \mathbf{s_{29}}, \mathbf{s_{30}}, \mathbf{s_{31}}\}$ can reach states (x_1, s) in Fig. 3 upon the occurrence of the uncontrollable event $!\mathtt{R}_1\mathtt{\_fail}$. Similarly, states (x_0, s') where $s' \in \{\mathbf{s_2}, \mathbf{s_5}, \mathbf{s_7}, \mathbf{s_8}, \mathbf{s_9}, \mathbf{s_{11}}, \mathbf{s_{12}}, \mathbf{s_{13}}, \mathbf{s_{15}}, \mathbf{s_{16}}, \mathbf{s_{17}}, \mathbf{s_{21}}, \mathbf{s_{25}}, \mathbf{s_{27}}, \mathbf{s_{28}}, \mathbf{s_{29}}\}$ can reach states (x_1, s') in Fig. 4 upon the occurrence of the uncontrollable event $!\mathtt{R}_4\mathtt{\_fail}$.

state vector s:

$$\forall x \in X, \quad \mathcal{I}(x) = \Big\{ q \in \{1, \ldots, 3\xi - n\} : \Big(\mathtt{stg}(s[q]) \notin \bigcup_{R_i \in x} ESup(R_i) \Big) \vee \Big(\mathtt{res}(s[q]) \in x \wedge \mathtt{stat}(s[q]) = \mathtt{III} \wedge \mathtt{stg}(s[q+1]) \notin \bigcup_{R_i \in x} ESup(R_i) \Big) \tag{4}$$

The function $\mathtt{stg}(s[q])$ appearing in Eq. 4 returns the processing stage of the underlying RAS Φ that corresponds to its argument $s[q]$. Then, in plain terms, the index set $\mathcal{I}(x)$ characterizes those active process instances in s that do not need any of the failing servers in x for the support of their remaining process plan. This realization further implies that, for any mode $x \in X$, Requirement 1 can be formally expressed in the established semantics for the s-DES $\mathcal{G}(\Phi)$, as the requirement for co-reachability w.r.t. some local state s

that satisfies the following condition:

$$\sum_{q\in\mathcal{I}(x)} s[q] \ = 0 \tag{5}$$

Clearly, the co-reachability of any local state s' in mode x w.r.t. some local state s of x that satisfies the condition of Eq. 5 implies the ability of the underlying RAS to clear all the process instances that do not require the failing resources in x. On the other hand, the satisfaction of the additional Requirement 2 also requires the availability of free capacity in the marked states that are defined by Eq. 5, at levels that are adequate to enable the repetitive execution of any process plan that does not involve the failing resources in mode x. Considering the single-unit nature of the underlying resource allocation function $\mathcal{A}$, and setting $\mathcal{A}_j \equiv \{A_{jk} : k = 1,\ldots,l(j)\}$; i.e., $\mathcal{A}_j$ for every process type $J_j,\ j = 1,\ldots,n$,[12] this last requirement is expressed by the following additional condition for the sought marked states of mode x:

$$\forall j \in \{1,\ldots,n\} \text{ s.t. } \mathcal{A}_j \cap x = \emptyset, \quad \forall i \in \{1,\ldots,m\} \text{ s.t. } R_i \in \mathcal{A}_j : \sum_{q:\mathtt{stg}(s[q])\in Sup(R_i)} s[q] \leq C_i - 1 \tag{6}$$

The next proposition establishes formally the fact that the specification of the local marked states for any given mode $x \in X$ according to the conditions of Eqs. 5 and 6 leads to the satisfaction of Requirements 1 and 2 w.r.t. that mode in a maximally permissive manner.

Proposition 1: For any given mode $x \in X$ of the s-DES $\mathcal{G}(\Phi)$, co-reachability w.r.t. the set of states that satisfies the conditions of Eqs. 5 and 6 is necessary and sufficient for the satisfaction of Requirements 1 and 2 w.r.t. that mode.

Proof: The sufficiency part of the above proposition is obvious from the content of Eqs 5 and 6. So, next we establish the necessity part.

Hence, consider a local state s^0 of mode x. The set of the active processes in this state, to be denoted by $\mathcal{J}^0$, can be partitioned into two subsets $\mathcal{J}_1^0$ and $\mathcal{J}_2^0$, with subset $\mathcal{J}_1^0$ containing the set of those process instances that do not require any resources in x for the execution of their remaining process plan. In the semantics established by Eq. 4, the process instances in $\mathcal{J}_1^0$ are represented by those components of s that belong in the set $\mathcal{I}(x)$. Then, Requirement 1, when combined with the sequential logic of the process types in F-L-SU RAS that is established by Definitions 1 and 2, implies the existence of an event sequence $\sigma_1 \in \Sigma^*$ such that $\delta_x(s^0,\sigma_1) = s^1$, where the set of the active process instances in s^1, $\mathcal{J}^1$, satisfies $\mathcal{J}^1 = \mathcal{J}_2^0$. But then, state s^1 satisfies the condition of Eq. 5.

Regarding the satisfaction of Requirement 2, first let us notice that this requirement must be satisfied at the aforementioned state s^1. This remark further implies that (i) there must exist a further transition sequence $\sigma_2 \in \Sigma^*$ with $\delta_x(s^1,\sigma_2) = s^2$, (ii) the set of active process instances in s^2 will be $\mathcal{J}^2 = \mathcal{J}^1 = \mathcal{J}_2^0$, and (iii) s^2 will enable the execution of an instance from every process type that does not engage the failing resources in mode x. This last requirement implies that state s^2 must satisfy the condition of Eq. 6. Furthermore, since $\mathcal{J}^2 = \mathcal{J}^1 = \mathcal{J}_2^0$, state s^2 will also satisfy the condition of Eq. 5. But then, state s^2 belongs to the set of "marked states" that is defined by Eqs 5 and 6 and the sought result is established. □

[12] In plain terms, the set $\mathcal{A}_j$ is the set of all the resource types that appear in the process plan of process type $J_j,\ j = 1,\ldots,n$.

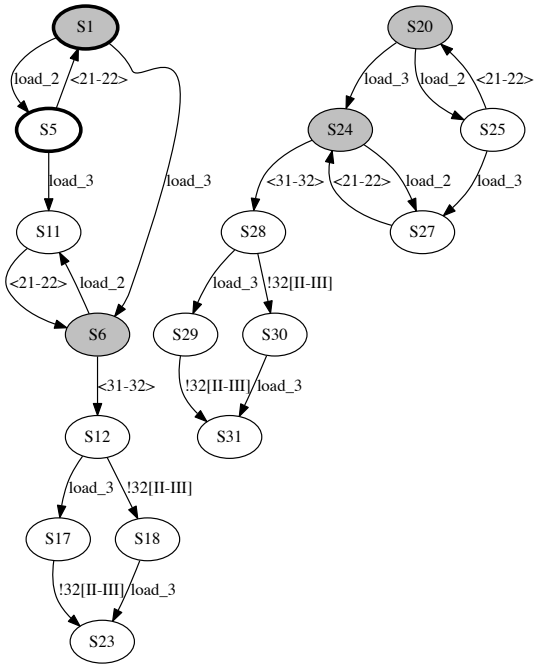

Fig. 3: The finite state automaton modeling the qualitative dynamics of the example RAS of Fig. 1 in mode x_1, where resource R_1 has failed. In this automaton, all states can reach their counterparts in mode x_0 upon the occurrence of the uncontrollable event $!\mathtt{R_1\_repair}$. On the other hand, states (x_1,s) where $s \in \{\mathbf{s_5},\mathbf{s_{11}},\mathbf{s_{12}},\mathbf{s_{17}},\mathbf{s_{25}},\mathbf{s_{27}},\mathbf{s_{28}},\mathbf{s_{29}}\}$ can reach states (x_3,s) in Fig. 5 upon the occurrence of the uncontrollable event $!\mathtt{R_4\_fail}$.

In order to turn the result Proposition 1 into a complete specification of the maximally permissive DAP for the considered RAS, we also need to account for the impact of the uncontrollability that is associated with the event sets Σ^u and E.

To facilitate the subsequent discussion, let us denote the target policy by Ω, and further define the transition function δ^Ω and its modal restrictions $\delta_x^\Omega,\ x \in X$, that provide a formal expression of the transitional dynamics of $\mathcal{G}(\Phi)$ under policy Ω. We shall also use the notation $\mathcal{G}(\Phi;\Omega)$ and $Reach(\mathcal{G}(\Phi;\Omega))$ to refer, respectively, to the controlled s-DES $\mathcal{G}(\Phi)$ under Ω, and the corresponding reachability space. Finally, for any mode $x \in X$, we define the sets $\mathcal{M}(x)$ and $S_{cor}(x)$ as follows:

$$\mathcal{M}(x) \equiv \big\{s \in S : s \text{ meets the conditions of Eqs 3, 5 and 6}\big\} \tag{7}$$

$$S_{cor}(x) \equiv \begin{cases} \big\{s \in S : s \text{ meets the condition of Eq. 3} \wedge \\ \quad \exists \sigma \in \Sigma^* \text{ s.t. } \delta_x(s,\sigma) \in \mathcal{M}(x)\big\}, & \text{if } \mathcal{M}(x) \neq \emptyset \\ \big\{s \in S : s \text{ meets the condition of Eq. 3}\big\}, & \text{o.w.} \end{cases} \tag{8}$$

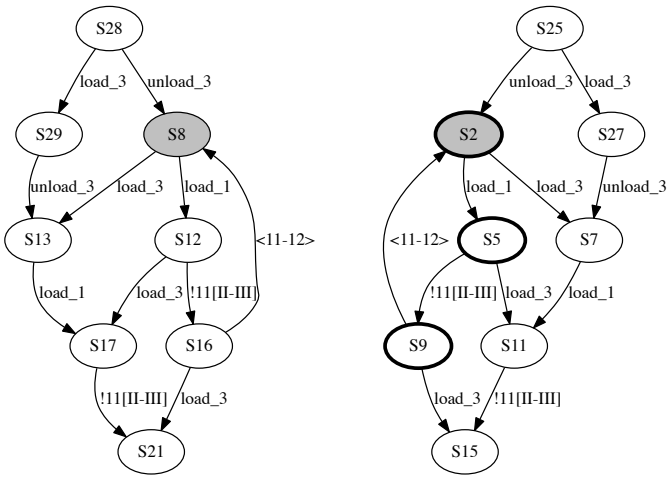

Fig. 4: The finite state state automaton modeling the qualitative dynamics of the example RAS of Fig. 1 in mode x_2, where resource R_4 has failed. In this automaton, all states can reach their counterparts in mode x_0 upon the occurrence of the uncontrollable event !$\mathtt{R_4\_repair}$. On the other hand, states (x_2, s) where $s \in \{\mathbf{s_5}, \mathbf{s_{11}}, \mathbf{s_{12}}, \mathbf{s_{17}}, \mathbf{s_{25}}, \mathbf{s_{27}}, \mathbf{s_{28}}, \mathbf{s_{29}}\}$ can reach states (x_3, s) in Fig. 5 upon the occurrence of the uncontrollable event !$\mathtt{R_1\_fail}$.

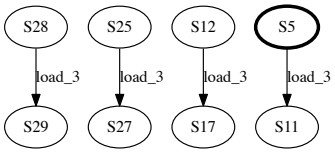

Fig. 5: The finite state automaton modeling the qualitative dynamics of the example RAS of Fig. 1 in mode x_3, where both R_1 and R_4 have failed. In this automaton, all states can uncontrollably reach either their counterparts in mode x_1 when event $!R_1\_repair$ occurs, or their counterparts in mode x_2 when event $!R_4\_repair$ occurs.

It should be obvious from Eqs. 7 and 8 that the set $\mathcal{M}(x)$ collects all the local marked states at mode $x \in X$, while the set $S_{cor}(x)$ is the local-state set satisfying the corresponding "co-reachability" predicate that is defined in Proposition 1. In particular, the reader should notice that the second branch in the right-hand-side of Eq. 8 essentially addresses the case of modes $x \in X$ where each process type $J_j,\ j = 1, \ldots, n$, requires at least one of the failing resources in that mode, and therefore, $\mathcal{M}(x) = \emptyset$. In these modes, Requirements 1 and 2 are essentially nullified, and therefore, all local states that might be experienced in these modes are admissible. The next proposition establishes that each state set $S_{cor}(x)$ is invariant w.r.t. the "local" uncontrollability that is encoded by the set Σ^u.

Proposition 2: For any mode $x \in X$ of the s-DES $\mathcal{G}(\Phi)$, $s \in S_{cor}(x) \ \wedge \ s' = \delta_x(s, q)$ for some $q \in \Sigma^u \implies s' \in S_{cor}(x)$.

Proof: Since $s \in S_{cor}(x)$, there exists an event sequence $\sigma \in \Sigma^*$ s.t. $\delta_x(s, \sigma) \in \mathcal{M}(x)$. Furthermore, since $q \in \Sigma^u$, it must be a service completion event, and therefore, the uncontrollable transition from the local state s to the local state s' does not alter the allocation of the underlying resources to the running processes. Finally, σ must contain at least one instance of the considered event q, and the removal of the first occurrence of q from this sequence provides a sequence σ' that is feasible in the state s' and $\delta_x(s', \sigma') \in \mathcal{M}(x)$. Hence, $s' \in S_{cor}(x)$. $\square$

On the other hand, the local nature of the specification of the sets $S_{cor}(x),\ x \in X$, w.r.t. the operational modes of the considered RAS Φ, together with the uncontrollable transitioning among these modes that is incurred by the resource-failing and restoration events in E, imply that the observation of Requirements 1 and 2 through the entire operation of the RAS Φ might necessitate the further restriction of the local behavior of Φ to certain subsets of the sets $S_{cor}(x),\ x \in X$. Indeed, at a first pass, it is pertinent to define these more restricted sets, for each mode $x \in X$, by:

$$\widehat{S}_{cor} \equiv \big\{s \in S :\ s \in S_{cor}(x),\ \forall x \in X \text{ that satisfies the condition of Eq. 3 w.r.t. local state } s\big\} \quad (9)$$

The above specification of the set $\widehat{S}_{cor}$ can be understood from the following two observations: (i) At any global state (x, s), a modal change due to the occurrence of an event $e \in E$ leaves the local state s unaltered. (ii) At any global state (x, s), a modal change due to the occurrence of an event $e \in E$ must lead to a RAS state (x', s) that satisfies the condition of Eq. 3. Then, the defining condition of the set $\widehat{S}_{cor}$ in Eq. 9 seeks to ensure that if the system is started at a global state (x, s) with $s \in S_{cor}(x)$, the occurrence of any event sequence $\epsilon \in E^*$ will take the system through a sequence of global states $(x^i, s),\ i = 1, 2, \ldots$, that satisfy $s \in S_{cor}(x^i),\ \forall i$. The next proposition formalizes this result and establishes some further important properties of the set $\widehat{S}_{cor}$.

Proposition 3: The local-state set $\widehat{S}_{cor}$ that is defined in Eq. 9 possesses the following properties:[13]

1) $\mathbf{0} \in \widehat{S}_{cor}$.
2) $\forall s \in S$ s.t. $|s| = 1,\ s \in \widehat{S}_{cor}$.
3) The global state set $\{(x, s) \in X \times S : s \in \widehat{S}_{cor}\}$ is invariant w.r.t. the uncontrollability of $\mathcal{G}(\Phi)$.

Proof: For the first part of Proposition 3, first it can be easily checked that $\mathcal{M}(\emptyset) = \{\mathbf{0}\}$, and therefore, $\mathbf{0} \in S_{cor}(\emptyset)$. Furthermore, since all servers are idle at local state $\mathbf{0}$, mode $\emptyset$ is the only viable mode for this local state. Hence, the result.

For the second part of Proposition 3, the reader should notice that state s can belong to at most two operational modes: (i) mode $\emptyset$, and (ii) in the case that the non-zero component corresponds to a processing status on a failure-prone server, s can also be part of the corresponding failing mode. In both cases, s is co-reachable to the corresponding set of marked states $\mathcal{M}(x)$ (in fact, in the second case it belongs in the corresponding set $\mathcal{M}(x)$).

To prove the last part of Proposition 3, consider a state (x, s) with $s \in \widehat{S}_{cor}$, and an uncontrollable event $q \in \Sigma^u \cup E$ that is feasible in (x, s). Let $(x', s') = \delta(x, s, q)$. We need to show that $s' \in \widehat{S}_{cor}$. For this, we discern the following two cases:

[13]The notation $|s|$ in the second part of this proposition implies the l_1 norm of the state vector s, i.e., the sum of all the components of s.

Case 1 – $q \in \Sigma^u$: Let $X(s)$ (resp., $X(s')$) denote the modes that satisfy the condition of Eq. 3 at state s (resp., s'). Then, the case definition implies that $X(s') \subset X(s)$. This result, together with the fact that $s \in \widehat{S}_{cor}$ and Proposition 2, imply that $s' \in S_{cor}(x),\ \forall x \in X(s')$. But then, $s' \in \widehat{S}_{cor}$.

Case 2 – $q \in E$: In this case, $s' = s$, and therefore it belongs in $\widehat{S}_{cor}$ by the working assumptions. □

Part #3 of Proposition 3 establishes formally that $\widehat{S}_{cor}$ contains, indeed, the uncontrollability of $\mathcal{G}(\Phi)$. On the other hand, an important implication of the first two parts of this proposition is that the proposed restriction of the operation of the s-DES $\mathcal{G}(\Phi)$ in the subspace $\{(x,s) \in X \times S : s \in \widehat{S}_{cor}\}$ does not impair the capability of the underlying RAS to support the execution of each process type $J_j,\ j = 1, \ldots, n$, since, in the worst case, each of these process types can be executed in isolation.

However, the "thinning" of the sets $S_{cor}(x),\ x \in X$, that is effected by their suggested substitution with the set $\widehat{S}_{cor}$ of Eq. 9, can destroy the required co-reachability of the states remaining in $\widehat{S}_{cor}$ to the corresponding modal sets of marked states $\mathcal{M}(x), x \in X$. In other words, a DAP that will seek to restrict the operation of the s-DES $\mathcal{G}(\Phi)$ so that it visits only local states in $\widehat{S}_{cor}$ at every mode $x \in X$, can experience "policy-induced deadlocks" in the resulting modal dynamics. We demonstrate this possibility through the following example.

Example 2: In this example we consider an F-L-SU RAS with four resource types, $R_i,\ i = 1, \ldots, 4$, all of capacity $C_i = 1$, and three process types $J_j,\ j = 1, 2, 3$. The corresponding process plans are $\langle R_1, R_2, R_3 \rangle$, $\langle R_3, R_2 \rangle$ and $\langle R_4, R_1 \rangle$. The set of failure-prone resources is $\mathcal{R}_F = \{R_2, R_3\}$.

Next, let us focus on the local state s that has three active process instances $j_1,\ j_2$ and j_3, with respective processing stages Ξ_{11}, Ξ_{21} and Ξ_{31}. State s belongs in $\widehat{S}_{cor}$: In particular, this state has a feasible terminating route in mode $\emptyset$ that first advances j_2 to completion, then j_1, and eventually j_3. Also, in mode $\{R_3\}$, which is the only other possible mode for this state under the restriction of Eq. 3, process instance j_1 can advance to resource R_2, enabling process instance j_3 to terminate, and the same advancement enables the repetitive execution of process type J_3 that does not require the failing resource R_3.

On the other hand, there is no single-stage advancement for any active process instance in state s that can lead to a state $s' \in \widehat{S}_{cor}$. The advancement of process instance j_1 will result in a deadlock of this process instance with process instance j_2. Also, the advancement of process instance j_2 is not admissible since the resulting state s' will not allow process instance j_3 to advance to completion in mode $\{R_2\}$. □

To cope with the policy-induced deadlocks that are demonstrated by Example 2, set $\widehat{S}_{cor}$ must be post-processed in order to identify and remove from it any local states, s, that will constitute such deadlocks in any mode $x \in X$ of the underlying RAS. The removal of these states might also necessitate the revision of the sets $\mathcal{M}(x), x \in X$, themselves, since some of the local marked states in these sets might not be able any more to initiate the execution of all the different process types that do not utilize the failing resources in the corresponding mode; these states must be identified and removed from all modal subspaces, as well. Another possibility is that, due to the effected state pruning, some local states might become unreachable in the local subspace of mode $\emptyset$. If we want to have a complete and accurate characterization of the reachable subspace of the target DAP Ω, these states must also be identified and removed from all the modal subspaces.[14] The above thinning process must also be iterative, since the removal of any set of local states from $\widehat{S}_{cor}$ might raise the need for the further thinning of this set due to some other of the three reasons mentioned above. On the other hand, the finite completion of this iterative computation with a non-empty subset of $\widehat{S}_{cor}$, $\overline{S}_{cor}$, is guaranteed by parts #1 and #2 of Proposition 3, which, as already mentioned, ensure the capability of each process type $J_j,\ j = 1, \ldots, n$, to execute in isolation under the posed operational Requirements 1 and 2.

Finally, the computation of the set $\overline{S}_{cor}$ along the aforementioned lines enables the implementation of the sought DAP Ω through the "one-step-look-ahead" rule:

$$\forall (x,s) \in X \times S,\ \forall \sigma \in \Sigma^c \text{ s.t. } \delta(x,s,\sigma) \text{ is well defined,}$$
$$\sigma \text{ is } \Omega\text{-admissible} \iff \delta_x(\sigma) \in \overline{S}_{cor} \quad (10)$$

The systematic computation of the set $\overline{S}_{cor}$, that is critical for the specification of the target policy Ω through Eq. 10, is addressed in the next section. We close the current section by highlighting and concretizing the above characterization of the target DAP Ω and its admissible subspace $Reach\mathcal{G}(\Phi;\Omega))$, by means of the example F-L-SU RAS of Fig. 1.

Example 1 (cont.): In order to derive the maximally permissive robust DAP Ω for the example F-L-SU RAS of Fig. 1 along the lines that were discussed in the previous paragraphs, first we derive the sets $\mathcal{M}(x)$ of marked states, and the corresponding sets $S_{cor}(x)$ of the co-reachable states, for each operational mode x of the underlying s-DES $\mathcal{G}(\Phi)$. With these two groups of sets available, subsequently we compute the corresponding local-state set $\widehat{S}_{cor}$. Finally, we operate on the obtained set $\widehat{S}_{cor}$ through the iterative "thinning" process that was described in the previous paragraphs, in order to obtain the final set $\overline{S}_{cor}$ that will provide a complete characterization of the maximally permissive robust DAP for this RAS through the condition of Eq. 10. Also, without any loss of generality, in the subsequent discussion, the local-state set S is restricted to the set $S_{reach}(x_0)$ that contains the states that are tabulated in Table II (i.e., the local states that are reachable in mode $\emptyset$ of the considered RAS Φ).

According to Eqs 5, 6 and 7, the marked states for the four modes $x_i,\ i \in \{0, 1, 2, 3\}$, of the s-DES $\mathcal{G}(\Phi)$ considered in this example, are defined as follows:

$$\mathcal{M}(x_0) = \{\mathbf{s_0}\} \quad (11)$$

[14] On the other hand, the identification and removal of these unreachable states from the modal subspaces is not essential for the effective implementation of the target DAP, since, due to the unreachability of these states, the deployed policy will never be called to operate on them.

$$\mathcal{M}(x_1) = \{\mathbf{s_1}, \mathbf{s_6}, \mathbf{s_{20}}, \mathbf{s_{24}}\} \tag{12}$$

$$\mathcal{M}(x_2) = \{\mathbf{s_2}, \mathbf{s_8}\} \tag{13}$$

$$\mathcal{M}(x_3) = \emptyset \tag{14}$$

In the STDs depicted in Figs 2–5, the corresponding marked states are represented by the shaded nodes. With the marked states for each mode $x_i,\ i = 0, 1, 2, 3$, well defined, the co-reachable state sets $S_{cor}(x_i),\ i \in \{0, 1, 2, 3\}$, can be obtained from Eq. 8 as shown below:

$$S_{cor}(x_0) = S_{reach}(x_0) \backslash \{\mathbf{s_{17}}, \mathbf{s_{21}}, \mathbf{s_{23}}, \mathbf{s_{26}}\} \tag{15}$$

$$S_{cor}(x_1) = \{\mathbf{s_1}, \mathbf{s_5}, \mathbf{s_6}, \mathbf{s_{11}}, \mathbf{s_{20}}, \mathbf{s_{24}}, \mathbf{s_{25}}, \mathbf{s_{27}}\} \tag{16}$$

$$S_{cor}(x_2) = \{\mathbf{s_2}, \mathbf{s_5}, \mathbf{s_8}, \mathbf{s_9}, \mathbf{s_{12}}, \mathbf{s_{16}}, \mathbf{s_{25}}, \mathbf{s_{28}}\} \tag{17}$$

$$S_{cor}(x_3) = \{\mathbf{s_5}, \mathbf{s_{11}}, \mathbf{s_{12}}, \mathbf{s_{17}}, \mathbf{s_{25}}, \mathbf{s_{27}}, \mathbf{s_{28}}, \mathbf{s_{29}}\} \tag{18}$$

The results of the above computation are tabulated in the first five columns of Table III. In this table, cells marked by '–' correspond to local states that do not satisfy the condition of Eq. 3 at the corresponding mode, and therefore, the corresponding global states are not realizable in the dynamics of the underlying RAS Φ. On the other hand, cells marked by 'X' correspond to local states that do not belong to the corresponding set $S_{cor}(x)$.

In order to compute the local-state set $\widehat{S}_{cor}$, let us denote by $S_{each}(x_i),\ i = 1, 2, 3$, the subsets of the considered local-state set $S_{reach}(x_0)$ that satisfy the condition of Eq. 3 at the corresponding mode x_i; each of these subsets consists of the local states appearing in the corresponding STDs of Figs 2–5, and they are represented by the "undashed" cells in Columns 2–5 of Table III. Then, the sought set $\widehat{S}_{cor}$ can be obtained from the following formula:[15]

$$\widehat{S}_{cor} = S_{reach}(x_0) \setminus \bigcup_{i=0}^{3} \big(S_{reach}(x_i) \setminus S_{cor}(x_i)\big) \tag{19}$$

Set $\widehat{S}_{cor}$ is reported in the sixth column of Table III. Juxtaposing the results of Table III with the state descriptions that are provided in Table II, it can be easily seen that the obtained set $\widehat{S}_{cor}$ contains the local state $\mathbf{0}$ and also every state of the set $S_{reach}(x_0)$ that has only one activated process instance; this fact is in agreement with the first two results of Proposition 3. Also, the juxtaposition of the results of Table III with the STDs of Figs 2–5 reveals that there is no transition from $S_{cor}(x_i)$ to $S_{reach}(x_i) \setminus S_{cor}(x_i)$, for any mode $x_i,\ i = 0, 1, 2, 3$, due to the local uncontrollable events $!11\{II - III\}$ and $!32\{II - III\}$; this last fact is consistent with the result of Proposition 2.

[15]More precisely, in view of the definition of the set S in the considered example as the set of local states that are reachable in mode $\emptyset$, the set that is computed by the formula of Eq. 19 is the set $\widehat{S}_{cor} \cap reach(\mathcal{G}(\Phi))$.

TABLE III: Computing the local state set $\widehat{S}_{cor}$ for Example 1. Cells marked by '–' correspond to local states that do not satisfy the condition of Eq. 3 at the corresponding mode. Cells marked by 'X' correspond to local states that do not belong to the corresponding $S_{cor}(x)$.

State	x_0	x_1	x_2	x_3	$\widehat{S}_{cor}$	$\overline{S}_{cor}$
$\mathbf{s_0}$		–	–	–	✓	✓
$\mathbf{s_1}$			–	–	✓	✓
$\mathbf{s_2}$		–		–	✓	✓
$\mathbf{s_3}$		–	–	–	✓	
$\mathbf{s_4}$		–	–	–	✓	✓
$\mathbf{s_5}$					✓	✓
$\mathbf{s_6}$			–	–	✓	
$\mathbf{s_7}$		–	X	–		
$\mathbf{s_8}$		–		–	✓	
$\mathbf{s_9}$		–		–	✓	✓
$\mathbf{s_{10}}$		–	–	–	✓	
$\mathbf{s_{11}}$			X			
$\mathbf{s_{12}}$		X				
$\mathbf{s_{13}}$		–	X	–		
$\mathbf{s_{14}}$		–	–	–	✓	
$\mathbf{s_{15}}$		–	X	–		
$\mathbf{s_{16}}$		–		–	✓	
$\mathbf{s_{17}}$	X	X	X			
$\mathbf{s_{18}}$		X	–	–		
$\mathbf{s_{19}}$		–	–	–	✓	
$\mathbf{s_{20}}$			–	–	✓	
$\mathbf{s_{21}}$	X	–	X	–		
$\mathbf{s_{22}}$		–	–	–	✓	
$\mathbf{s_{23}}$	X	X	–	–		
$\mathbf{s_{24}}$			–	–	✓	
$\mathbf{s_{25}}$					✓	
$\mathbf{s_{26}}$	X	–	–	–		
$\mathbf{s_{27}}$			X			
$\mathbf{s_{28}}$		X				
$\mathbf{s_{29}}$		X	X			
$\mathbf{s_{30}}$		X	–	–		
$\mathbf{s_{31}}$		X	–	–		

In the last part of this example, we use the results of Table III in order to compute the desired set $\overline{S}_{cor}$, and thus, characterize the maximally permissive robust DAP for the considered RAS Φ. The corresponding computation evolves through the following phases: In the first phase, this computation removes from the STDs depicted in Figs 2–5 all the corresponding local states that do not belong in $\widehat{S}_{cor}$. As a result of this removal, local states s_6 and s_{24} loose their ability to support the repetitive execution of the process type J_2 in mode x_1, and local state s_8 looses its ability to support the repetitive execution of the process types J_1 and J_3 in mode x_2. The subsequent removal of these states from all the local STDs renders in-coreachable the local state s_{16} in mode x_2, and hence, this local state has to be pruned from all STDs, as well. Also, the removal of the local states s_6, s_8 and s_{16} from

the STD of Fig. 2 renders total deadlocks the states s_3 and s_{10} in the STD of mode x_0, and therefore, they must be pruned. Due to all the above state prunings, states $s_{14},\ s_{19},\ s_{20},\ s_{22}$ and s_{25} become unreachable from the initial state $\mathbf{0}$ in the STD of mode x_0 and they must also be removed from all modal STDs, for a complete characterization of the reachable subspace of $\mathcal{G}(\Phi;\Omega)$.

The results of the state-pruning process that was described in the previous paragraph are reported in the last column of Table III. These results are also depicted graphically by the boldfaced nodes in the STDs of Figs 2–5.

Finally, at each reachable global state (x, s) of the considered RAS Φ, the maximally permissive robust DAP Ω that is defined by the above results will admit a process-loading or a controllable process-advancing event $q \in \Sigma^c$ if and only if the resulting local state s' belongs in $\overline{S}_{cor}$.

V. Computing the maximally permissive robust DAP

In this section we present an algorithm for the obtaining a distributed representation of the set $\overline{S}_{cor}$ that defines the maximally permissive DAP Ω for any given F-L-SU RAS Φ. We also provide a formal analysis for this algorithm that establishes its finite computation and the correctness of the derived DAP w.r.t. to the specifications that were posed in Sections II and IV, and we conclude with a series of remarks that can lead to more efficient implementations of the presented algorithm in terms of its execution time and / or its memory requirements.

The proposed distributed algorithm: Motivated by the corresponding results of [31], [32], the proposed algorithm distributes the computation of the target set $\overline{S}_{cor}$ over a number of threads $\Theta_x,\ x \in X$, that are in one-to-one correspondence with the distinct operational modes x of the underlying RAS Φ. These threads communicate the partial results of their computation through a message-passing mechanism [33] that regulates the progress of the overall computation and guarantees the correctness of the final outcome. Next, we detail the logic that is executed by each of the aforementioned threads, and the data structures and the communication mechanisms that are employed in this execution; the complete pseudo-code that is run by each thread Θ_x is provided in Figure 6.

We start by describing the communication mechanisms that are used by the various threads of the considered algorithm. As already mentioned, the key communication mechanism is a message-passing mechanism that is implemented by a FIFO message list, $MessageQueue_x$, maintained by each thread $\Theta_x,\ x \in X$, and the 2-dim integer table $COUNT[|X|,|X|]$ where the entry $COUNT[x,x']$ reports the number of messages sent by Θ_x to $\Theta_{x'}$ that are still unprocessed by $\Theta_{x'}$. The messages exchanged through this mechanism are essentially subsets of the local-state set S that must be pruned from some corresponding state sets that are maintained by the recipient thread. The table $COUNT[|X|,|X|]$ is a data structure that is shared among all threads $\Theta_x,\ x \in X$. Another shared data structure among these threads is the 1-dim array $DONE[|X|]$, with $DONE[x]$ being a Boolean variable indicating whether

Input: $S,\ \Sigma,\ \delta_x,\ \delta_\emptyset$
Output: $S_{cor}(x)$

/* INITIALIZATION */
$MessageQueue_x := NILL$; $DONE[x] :=$ `FALSE`;
for all $x' \in X$ **do**
  $COUNT[x,x'] := 0$;
end for
$S_{reach} := \{s \in S : \exists \sigma \in \Sigma^*,\ s = \delta_\emptyset(s_0,\sigma)\}$;
$S_{reach}(x) := S_{reach} \setminus \{s \in S_{reach} : s \not\vdash$ Eq 3$\}$;
$\mathcal{M}(x) := \{s \in S_{reach}(x) : s \vdash$ Eqs 5 and 6$\}$;
if $\mathcal{M}(x) \neq \emptyset$ **then**
  $S_{cor}(x) := \mathcal{M}(x)$;
else
  $S_{cor}(x) := S_{reach}(x)$;
end if
repeat
  $Q_{cor}(x) := S_{cor}(x)$;
  $S_{cor}(x) := S_{cor}(x) \cup \{s \in S_{reach}(x) : \exists \sigma \in \Sigma,\ \delta_x(s,\sigma) \in S_{cor}(x)\}$;
until $Q_{cor}(x) = S_{cor}(x)$;
$M_x := S_{reach}(x) \setminus S_{cor}(x)$;
if $M_x \neq \emptyset$ **then**
  for all $x' \in X \setminus \{x\}$ **do**
    $COUNT[x,x']++$; send M_x to $\Theta_{x'}$;
  end for
end if
/* PROCESS MESSAGE QUEUE */
repeat
  if $MessageQueue_x \neq NILL$ **then**
    $DONE[x] :=$ `FALSE`, $S_{rm}(x) := \emptyset$;
    while $MessageQueue_x \neq NILL$ **do**
      Pop message $M_{x'}$ from $MessageQueue_x$;
      $COUNT[x',x]--$;
      $S_{rm}(x) := S_{rm}(x) \cup \big(S_{cor}(x) \cap M_{x'}\big)$;
    end while
    if $S_{rm}(x) \neq \emptyset$ **then**
      $S_{cor}(x) := S_{cor}(x) \setminus S_{rm}(x)$;
      repeat
        $Q_{rm}(x) := S_{rm}(x)$;
        $oldS_{cor}(x) := S_{cor}(x)$;
        $S_{cor}(x) := \mathcal{M}(x) \cap oldS_{cor}(x)$;
        repeat
          $Q_{cor}(x) := S_{cor}(x)$;
          $S_{cor}(x) := S_{cor}(x) \cup \{s \in oldS_{cor}(x) : \exists \sigma \in \Sigma,\ \delta_x(s,\sigma) \in S_{cor}(x)\}$;
        until $Q_{cor}(x) = S_{cor}(x)$;
        $S_{rm}(x) := S_{rm}(x) \cup \big(oldS_{cor}(x) \setminus S_{cor}(x)\big)$;
        $Q_m(x) := \{s \in \mathcal{M}(x) \cap S_{cor}(x) : \forall J_j \in \mathcal{P}(x),\ \delta_x(s, load(J_j)) \in S_{cor}(x)\}$;
        $S_{rm}(x) := S_{rm}(x) \cup (\mathcal{M}(x) \setminus Q_m(x))$;
        $\mathcal{M}(x) := Q_m(x)$;
      until $Q_{rm}(x) = S_{rm}(x)$;
      $M_x := S_{rm}(x)$;
      for all $x' \in X \setminus \{x\}$ **do**
        $COUNT[x,x']++$; send M_x to $\Theta_{x'}$;
      end for
    end if
    $DONE[x] :=$ `TRUE`;
  end if
until $\big(\sum_{x,x' \in X} COUNT[x,x'] = 0\big) \wedge \bigwedge_{x \in X} DONE[x]$;
return $S_{cor}(x)$;

Fig. 6: The sequential logic that is executed by each thread $\Theta_x,\ x \in X$, of the presented algorithm.

thread Θ_x is in a processing or an idling mode. The terminating condition for the entire algorithm is the condition of Line 53 in Fig. 6, where all sent messages have been processed by their recipient threads, and each thread is in an idling mode.

The main phases of the sequential logic in Fig. 6 that is executed by each thread Θ_x is as follows: After initializing the aforementioned data structures and mechanisms that facilitate the thread communication (Lines 1–4), the thread computes the corresponding set $\mathcal{M}(x)$ in Lines 5–7. This is done by first computing the set $S_{reach}(x)$ containing all the local states that are reachable in mode x (Lines 5–6), and subsequently computing the set $\mathcal{M}(x)$ as the subset of $S_{reach}(x)$ that satisfies the conditions of Eqs 5 and 6. Finally, the set $S_{cor}(x)$ is computed from $\mathcal{M}(x)$ through the iterative ("fixed point") computation of Lines 8–16. Furthermore, in Lines 17–22 the algorithm synthesizes the message M_x containing all the reachable states in mode x that are not co-reachable to the set of the local marked states $\mathcal{M}(x)$, and if M_x is not empty, it broadcasts this message to every other thread $\Theta_{x'}$.

Subsequently, thread Θ_x checks its message queue for any received messages, and processes these messages by first compiling all their contents in the set $S_{rm}(x)$ (Lines 23–30). If $S_{rm}(x)$ is non-empty, then the contents of this set are removed from the maintained set $S_{cor}(x)$ (Lines 31–32), and this removal triggers a further revision of the sets $\mathcal{M}(x)$ and $S_{cor}(x)$ so that they remain consistent with their original definitions in the context of the more restricted dynamics that result from the aforementioned elimination of the local states in $S_{rm}(x)$. More specifically, thread Θ_x goes into the loop of Lines 33–45 where, at each iteration, it first updates $S_{cor}(x)$ to its subset of states that remain co-reachable to the set $\mathcal{M}(x)$ after the aforementioned removal of $S_{rm}(x)$ (Lines 34–40). Subsequently, it also updates the set $\mathcal{M}(x)$ to its subset of states that enable the initiation of every process type in $\mathcal{P}(x)$ under the restricted local dynamics that are defined by the updated value of $S_{cor}(x)$ (Lines 41–44). The loop that is defined by Lines 33-45 concludes when no further updating of $S_{cor}(x)$ or $\mathcal{M}(x)$ is necessary. Furthermore, during the execution of this loop, thread Θ_x compiles in the set $S_{rm}(x)$ all the local states that are removed from $S_{cor}(x)$ and $\mathcal{M}(x)$, and these states are subsequently broadcasted to all other threads $\Theta_{x'}$ (Lines 46–50).

After completing the processing of its current messages as described in the previous paragraph, thread Θ_x sets the Boolean variable $DONE[x]$ to TRUE (Line 51) and goes into an idling mode until (i) either some new message(s) are received, in which case the above processing cycle of Lines 23–52) is repeated on this new set of messages, (ii) or Θ_x finds out that the terminating condition of Line 53 is satisfied, in which case it exits, returning the set $S_{cor}(x)$ as the final outcome of its computation.

Algorithmic analysis: The next theorem establishes the finiteness and the correctness of the algorithm that is described in the above paragraphs.

Theorem 1: The application of the distributed algorithm defined by the pseudo-code of Fig. 6 on any given F-L-SU RAS instance Φ

1) will terminate in a finite number of steps, and
2) the SC policy Ω that admits a local controllable transition in any mode $x \in X$ *iff* the resulting state $\delta_x(s, \sigma)$ belongs in the corresponding set $S_{cor}(x)$ that is returned by the thread Θ_x, is the maximally permissive robust DAP for RAS Φ.

Proof: The finiteness of the computation of the considered algorithm can be established through the following two observations: First, we notice that the set S_{reach} that is computed in Line 5 of the pseudo-code of Fig. 6, is a finite set, due to (i) the finiteness of the resource capacities $C_i,\ i = 1, \ldots, m$, and (ii) the structure of the resource allocation function $\mathcal{A}$ of the underlying RAS Φ. Hence, all the initially obtained sets $S_{cor}(x)$ are finite, since they are subsets of S_{reach}. Furthermore, the broadcasting of a message M_x by some thread $\Theta_x,\ x \in X$, implies the strict reduction of (at least) the corresponding set $S_{cor}(x)$. Hence, the number of messages that can be exchanged among all the threads Θ_x is finite, and therefore, eventually all these threads will find themselves simultaneously in the idling mode with no further messages to be processed.

To prove that the policy Ω induced by the computed sets $S_{cor}(x),\ x \in X$, is the maximally permissive DAP for the underlying RAS Φ, first we notice that each thread $\Theta_x,\ x \in X$, computes correctly the corresponding set $S_{cor}(x)$, as defined by Eq. 8. In particular, $S_{cor}(x)$ will contain the local states s that are co-reachable to the sets in $\mathcal{M}(x)$ if the latter set is non-empty (i.e., if there are process types J_j in mode x that do not require the failing resources for their execution), or it will be equal to the set $S_{reach}(x)$ otherwise. Furthermore, any pruning of the sets $S_{cor}(x)$ and $\mathcal{M}(x),\ x \in X$, that is effected by the corresponding thread Θ_x in response to a received message $M_{x'}$, is necessary in the face of (i) the uncontrollable nature of the resource failing and restoration events in the corresponding set E, and (ii) the behavioral specifications that are posed by Requirements 1 and 2. Finally, the original construction of the sets $\mathcal{M}(x)$ and $S_{cor}(x),\ x \in X$, and the logic that drives the subsequent pruning of these sets, guarantee that, at each mode $x \in X$, the counterparts of these sets that are obtained upon the termination of the considered algorithm, still present the basic structure that was requested for the original sets $\mathcal{M}(x)$ and $S_{cor}(x)$ in order to express the corresponding Requirements 1 and 2 for that mode.

The above remarks imply that the policy Ω that is induced by the sets $S_{cor}(x),\ x \in X$, according to the logic of Theorem 1, satisfies the local Requirements 1 and 2 at each mode $x \in X$, in a maximally permissive manner. To conclude the proof of the second part of Theorem 1, next we shall also show that policy Ω will not experience any deadlocks in its global state space $X \times S$. In particular, we shall show that for each global state (x, s) that is reachable under policy Ω, there exists at least one corresponding sequence $\sigma \in (\Sigma \cup E)^*$ that leads to the state $(\emptyset, \mathbf{0})$ while visiting only Ω-adimissible global states.

To establish this last result, first we notice that, for the considered state (x, s), there will always be an event sequence $\sigma^1 \in E^*$ leading from this global state to the corresponding global state $(\emptyset, s)$ without violating the policy Ω; sequence σ^1

will consist of the restoration events for all the resources that are failing in mode x, in any order, and it is Ω-admissible since, if it passed through some global state (x', s) with $s \notin S_{cor}(x')$, then local state s would have been removed from $S_{cor}(x)$ by Θ_x in response to a message received from $\Theta_{x'}$. On the other hand, since state s belongs to the returned set $S_{cor}(\emptyset)$, state s is co-reachable to the set $\mathcal{M}(\emptyset)$ in mode $\emptyset$ through an Ω-admissible event sequence $\sigma^2 \in \Sigma^*$. But $\mathcal{M}(\emptyset)$ is the singleton $\{(\emptyset, \mathbf{0})\}$. Hence, the sought sequence σ is the sequence $\sigma^1\sigma^2$.[16] $\square$

We conclude the analysis of the presented algorithm with two additional remarks. First we remind the reader that according to the discussion that was provided in Section IV, the maximally permissive robust DAP for the considered RAS will always be able to support the execution of all the process types J_j, $j = 1, \ldots, n$, of the underlying RAS Φ, possibly running only single instances from each of these process types in isolation. Hence, the sets $S_{cor}(x)$ that are returned by each thread Θ_x, $x \in X$, of the considered algorithm will be non-empty.

Furthermore, in the context of the distributed computation of the presented algorithm and the corresponding results in Theorem 1, the sought policy Ω is determined by the sets $S_{cor}(x)$ that are returned by each thread Θ_x, $x \in X$. However, it is not hard to see that policy Ω can also be defined as the one-step-look-ahead policy that admits a controllable transition at any given RAS state (x, s) *iff* the resulting local state s' belongs in the set $\bigcup_{x \in X} S_{cor}(x)$; this last set is essentially a representation of the set $\overline{S}_{cor}$ that was used in Section IV in order to define the policy Ω through Eq. 10.

Complexity and other implementational considerations: We conclude the section on the computation of the sought policy Ω by providing some further remarks on the practical tractability of this task, and some additional possibilities that can influence that necessary computational time and the memory requirements.

For a start, the reader should notice that in the distributed computation that was discussed in the earlier parts of this section, the computation that is executed by each thread Θ_x evolves on the local state space S which is commensurate to the state spaces that are addressed in the context of failure-free RAS, and therefore, this computation is manageable through the recently emerged techniques that were discussed in the introductory section of this paper. A particularly promising approach can be based on the adoption of a symbolic representation for the underlying RAS dynamics similar to that employed in [12]. Alternatively, one can also seek to extend the methodology of [10], [11] to the considered application context; the complete realization of such an extension will also require a more systematic investigation of the monotonicity properties that are implied by the defining logic of the maximally permissive DAP Ω.

[16]More precisely, the execution of the event sequence σ^2 might be interrupted by a number of resource outages and the corresponding excursion of the RAS dynamics to some other operational modes than mode x_0. However, each of these outages eventually will be restored, returning the considered RAS to its mode x_0 for the continuation of the execution of the transition sequence σ^2.

Finally, it is also possible to consider an alternative implementation of the computation that was presented in the earlier parts of this section, where all threads Θ_x, $x \in X$, work on a single copy of the state set S_{reach}, that will constitute a shared data structure for all these threads. Under this new computational scheme, the contents of this set will be iteratively modified by the running threads Θ_x; in particular, each thread will further prune this state set on the basis of (i) the current content of the set, and (ii) the thread pruning logic that was detailed in the previous parts of this section. Such a realization can reduce extensively the memory footprint of the presented algorithm, at the expense of a reduced concurrency in the overall computation. In fact, this new scheme can even be implemented through a single computational thread that executes the pruning logic corresponding to each mode $x \in X$ as a separate subroutine that is invoked iteratively during the execution of the algorithm. The interesting feature of this scheme is that the memory footprint for the entire algorithm is no more than the size of the local state space S; i.e., the presence of many operational modes in the underlying RAS Φ does not increase the corresponding memory requirements.

VI. Conclusion

In this work, we have revisited the problem of the computation of a robust DAP for a RAS class that involves potential temporary resource outages and has been primarily motivated by the buffer allocation taking place in flexibly automated production cells. By taking advantage of the modeling and the computational capabilities that are provided by the recently emerged paradigm of switched Discrete Event Systems (s-DES) [31], [32], we were able to provide (i) a succinct characterization of the maximally permissive robust DAP Ω for the considered RAS class, and (ii) a distributed algorithm for the computation of a practically implementable representation of this policy. At the same time, the developments presented in this work extend the existing s-DES SC theory so that it can handle nonblocking supervision, in addition to the "safety" requirements that were the primary specifications addressed in the original work of [31], [32].

A possible extension of the presented work concerns the systematic investigation of the monotonicity properties that are possessed by the admissible subspace of the target SC policy Ω, and the potential efficiencies that can be incurred by these properties in the computation of Ω in line with similar developments that are presented in [13], [8]. Further extensions can try to adapt the notion of "robust deadlock avoidance" to other operational contexts that are amenable to the fundamental abstraction of the sequential RAS. This line of extensions could even try to come up with alternative notions of "robustness" that will mitigate the typical restrictiveness of the corresponding DAPs at the expense of certain operational risks that might be tolerated by these policies. Finally, it is interesting to apply the modeling and the computational methodology developed in this work to s-DES structures and dynamics that transcend the concept of the sequential RAS; certain classes of games as well as other DES with a multimodal structure for their underlying dynamics are natural candidates for such applications.

REFERENCES

[1] S. Reveliotis. Coordinating autonomy: sequential resource allocation systems for automation. *IEEE Robotics & Automation Magazine*, 22:77–94, 2015.

[2] S. Reveliotis. Real-time management of complex resource allocation systems: necessity, achievements and further challenges. *Annual Reviews in Control*, 41:147–158, 2016.

[3] 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.

[4] 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.

[5] M. Zhou and M. P. Fanti (editors). *Deadlock Resolution in Computer-Integrated Systems*. Marcel Dekker, Inc., Singapore, 2004.

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

[7] Z. Li, M. Zhou, and N. Wu. A survey and comparison of Petri net-based deadlock prevention policies for flexible manufacturing systems. *IEEE Trans. Systems, Man and Cybernetics – Part C: Applications and Reviews*, 38:173–188, 2008.

[8] 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.

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

[10] A. Nazeem and S. Reveliotis. A practical approach for maximally permissive liveness-enforcing supervision of complex resource allocation systems. *IEEE Trans. on Automation Science and Engineering*, 8:766–779, 2011.

[11] A. Nazeem and S. Reveliotis. Efficient enumeration of minimal unsafe states in complex resource allocation systems. *IEEE Trans. on Automation Science & Engineering*, 11:111–124, 2014.

[12] Z. Fei, S. Reveliotis, S. Miremadi, and K. Akesson. A BDD-based approach for designing maximally permissive deadlock avoidance policies for complex resource allocation systems. *IEEE Trans. on Automation Science and Engineering*, 12:990–1006, 2015.

[13] 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 & Francis, 2014.

[14] S. Reveliotis. Accommodating FMS operational contingencies through routing flexibility. *IEEE Trans. on R&A*, 15:3–19, 1999.

[15] J. Li, M. Zhou, T. Guo, Y. Gan, and Dai. X. Robust control reconfiguration of resource allocation systems with Petri nets and integer programming. *Automatica*, 50:915–923, 2014.

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

[17] G. Y. Liu, Z. W. Li, K. Barkaoui, and A. M. Al-Ahmari. Robustness of deadlock control for a class of petri nets with unreliable resources. *Information Sciences*, 235:259–279, 2013.

[18] 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.

[19] Z. Li, S. Zhu, and M. Zhou. A divide-and-conquer strategy to deadlock prevention in flexible manufacturing systems. *IEEE Trans. Systems, Man and Cybernetics – Part C*, 39:156–169, 2008.

[20] 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.

[21] K. Yamalidou, J. Moody, M. D. Lemmon, and P. J. Antsaklis. Feedback control of petri nets based on place invariants. *Automatica*, 32:15–28, 1996.

[22] M. Lawley and W. Sulistyono. Robust supervisory control policies for manufacturing systems with unreliable resources. *IEEE Trans. on R&A*, 18:346–359, 2002.

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

[24] S. F. Chew and M. Lawley. Robust supervisory control for production systems with multiple resource failures. *IEEE Trans. on Automation Science and Engineering*, 3:309–323, 2006.

[25] H. Yue, K. Y. Xing, and Z. Hu. Robust supervisory control policy for avoiding deadlock in automated manufacturing systems with unreliable resources. *Int. J. Prod. Res.*, 52:1573–1591, 2014.

[26] F. Wang, K-Y. Xing, M-C. Zhou, X-P. Xu, and L-B. Han. A robust deadlock prevention control for automated manufacturing systems with unreliable resources. *Information Sciences*, 345:243–256, 2016.

[27] H. Yue, K. Xing, Z. Hu, Wu W., and H. Su. Petri-net-based robust supervisory control of automated manufacturing systems. *Control Engineering Practice*, 54:176–189, 2016.

[28] Y. Wu, K. Xing, J. Luo, and X. Feng. Robust deadlock control for automated manufacturing systems with an unreliable resource. *Information Sciences*, 346–347:17–28, 2016.

[29] F. S. Hsieh. Robutness of deadlock avoidance algorithms for sequential processes. *Automatica*, 39:1695–1706, 2003.

[30] F. S. Hsieh. Robustness analysis of Petri nets for assembly/disassembly processes with unreliable resources. *Automatica*, 42:1159–1166, 2006.

[31] S. Reveliotis and Z. Fei. Invariant-based supervisory control of switched Discrete Event Systems. In *ADHS 2015*. IFAC, 2015.

[32] S. Reveliotis and Z. Fei. Invariant-based supervisory control of switched Discrete Event Systems. *IEEE Trans. on Automatic Control*, (to appear).

[33] N. A. Lynch. *Distributed Algorithms*. Morgan Kaufmann, San Francisco, CA, 1996.

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

[35] S. Reveliotis and Z. Fei. Robust deadlock avoidance for sequential resource allocation systems with resource outages. In *CASE 2016 (submitted)*. IEEE, 2016.

[36] C. Baier and J.-P. Katoen. *Principles of Model Checking*. The MIT Press, Cambridge, MA, 2008.