Calculating Restart States for Systems Modeled by Operations Using Supervisory Control Theory

Submitted to *Machines*. Pages 1 - 26.

OPEN ACCESS

Article

Calculating restart states for systems modeled by operations using supervisory control theory

Patrik Bergagård [1,*] **and Martin Fabian** [1]

[1] Department of Signals and Systems, Chalmers University of Technology, Gothenburg, Sweden

* Author to whom correspondence should be addressed; patrikm at chalmers.se, +4631-7721786

Version July 9, 2014 submitted to Machines. *Typeset by LaTeX using class file mdpi.cls*

Abstract: This paper presents a supervisory control theory based offline method for calculating restart states in a manufacturing control system. Given these precalculated restart states, an operator can be given correct instructions for how to resynchronize the control system and the manufacturing resources during the online restart process. The proposed method enables restart after unforeseen errors. It is assumed that the control system is modeled by operations and that possible operation sequences emerge through dependencies between the operations. The paper shows how reexecution requirements may be included in the calculation to obtain a correct behavior for the restarted system. In addition, it is shown how to filter out restart states, that require less effort for the operator during the online restart, and how to adapt the nominal production to always enable restart in desired restart states.

Keywords: Discrete event systems; restart; system recovery

1. Introduction

Downtime due to errors is costly in flexible manufacturing systems [1,2]. It is therefore desirable to perform a quick and correct recovery in order to resume the nominal production after an error. Among others, [3,4] see automatic error recovery as a must in todays manufacturing systems.

Error recovery in complex systems is a complicated task [5], often divided into three major activities [6]: *detection* of discrepancies between the intended behavior and the actual behavior of a system, *diagnosis* to find the original fault causing the observed error, and *recovery* of the system to continue the nominal production. Recovery is further partitioned into *error correction* and *restart*. The error correction phase concerns the process to remove underlying faults and repair the resource(s) if

required. The restart phase, which is the focus of this paper, then covers the process to resume the nominal production.

A wide variety of possible errors may cause failures in a manufacturing system. For example, a part may be missing in a resource, be badly positioned in a fixture, or be not enough processed. Resources may stop working due to faulty sensors and/or actuators, such as worn out cutting tools and broken weld guns. Typical manufacturing system errors are listed by [7–9], among others.

Some errors may be foreseen and appropriate corrective actions may then be included in the control system, see for example [6,10,11]. Tip-dressing of the electrodes used in weld applications is an example of a proactive corrective action to avoid a failure [9].

In general, however, it is impossible to foresee all errors that may occur and/or include all corrective actions in the control system [3]. Thus, restart after unforeseen errors is often not supported correctly or as efficiently as it should in relation to the potential cost of the resulting downtime.

The control system for a manufacturing system is often based on a set of *operations* that are to be executed in order to refine a product [12]. Each operation is typically modeled by three states; an *executing state* that is preceded by a state to model that the operation has not yet started, and succeeded by a state to model that the operation is completed. Possible operation sequences emerge through *dependencies* between the operations [12].

The nominal production may then be viewed as a trajectory between a composed source state where none of the operations have started, to a composed target state where a subset of the operations have completed. Each *control system state* on the trajectory will then model that some operations have not started, some operations are executing, and that the rest of the operations have completed.

For the sake of control and supervision, the resources and the product(s) in the manufacturing system can be abstracted into a set of *physical states*. When an operation is executed, the manufacturing system will, typically, change between many physical states. Thus, several physical states *correspond* to each control system state. Moreover, during the nominal production, the control system state evolves in synchrony with the corresponding physical states.

An unforeseen error is then a physical state that does not correspond to the current active control system state. Moreover, the actions required when correcting the error may force an operator to further corrupt the physical state during the error correction phase, such as moving a robot to a home-state. Thus, it is reasonable to assume that the control system and the resources are unsynchronized after the correction phase [13].

The aim of the restart phase is then to *resynchronize* the control system and the resources [9]. This may necessitate to update both the active state of the control system and the physical state of the manufacturing system.

As a consequence, the objective in most error recovery methods presented in the literature, is to restart the system such that the nominal production may continue from either an earlier, a later, or the current control system state with respect to the active control system state at the time of the error [8,14]. This state from where the control system is continued is often called a *restart state* [9]. Recovery in an earlier or a later state is often referred to as *backward* and *forward error recovery*, respectively. Flexible manufacturing systems enable, however, production according to multiple operation sequences [12], so

backward and forward error recovery are seldom well defined. Thus, the recovery concept must be generalized in order to handle flexible manufacturing systems with multiple operation sequences.

When recovering the control system from an earlier state, it may be necessary to reexecute some of the operations [9,13,15]. However, the existence of a physical product will constrain the reexecution [7, 9,14,16], certain *reexecution requirements* must be satisfied. For instance, an operation to fixate a part may be reexecuted as long as the succeeding refinement operation has not been started. This is in contrast to a glue applying operation that typically cannot be reexecuted.

Many of the restart methods presented in the literature are tailor made for specific types of manufacturing systems. Body-in-white manufacturing systems in the automotive industry is the main application for the methods presented in [5,9,15]. The method in [17] focuses on error recovery connected to loading, processing, and unloading CNC machines. Error recovery for systems where a set of resources are linked with material handling devices and intermediate buffers are described in [7]. A method that is suitable for, but not limited to, error recovery in batch systems is presented in [18]. To increase the transparency, it would be beneficial with more general methods less biased towards any specific type of system.

Few of the restart methods presented in the literature give a clear insight for how to systematically implement the theoretical ideas into a general control system for an industrial manufacturing system; among the exceptions are [4,5,19]. Some methods require a specific control system architecture and are hence not generally applicable, see for example [13,15,20].

Overviews of different techniques used in restart methods are given in [6,9,18]. In [9], some restart methods are also classified according to if the main work load is *online* when an error has been diagnosed, or *offline* before start of production.

Online methods, such as [4,7,8,16–21], typically gather a majority of the relevant restart information at the time of the error and then perform backward or forward error recovery [8]. Some methods, [16,19], reschedule the operations in the control system to find a new operation sequence after the error. A method that dynamically disables events in the control system when an error is detected is presented in [4]. Most industrial control systems are, however, not powerful enough for methods that require heavy calculation online, so such approaches contradict the industrial desire to keep the online control system simple [9].

Methods where the main work is done offline, such as [1,9–11,13,22–25], have an advantage compared to online methods, not only due to the need of less powerful hardware online. Beforehand calculation enables different restart alternatives to be analyzed already when the production in the manufacturing system is planned, such that undesirable situations can be resolved if possible. This beforehand analysis is a big advantage for the offline methods.

Motivated by the existing methods and their limitations, this paper presents an offline method for calculating restart states. The proposed method is neither tailor made for, nor limited to a specific type of manufacturing system or control system. The overall idea of the proposed method is related to the method presented in [5,9], but with some major generalizations, that will be clearly pointed out in the remainder of this section.

As for all offline methods, it is assumed that the restart consists of an *offline phase*, where the restart states are calculated, and an *online phase*, where these states are used when the manufacturing system is to be restarted after an error. As in [13] it is assumed that an error can only occur when one or more

of the resources are used. In order to relate control system states on different operation sequences, the proposed method introduces the concept of *upstream states* which generalizes the concept of backward error recovery.

To benefit from existing advances using formal methods, the proposed method is based on the supervisory control theory [26]. During an initial formalization part a user-given set of operations, with dependencies and reexecution requirements, are automatically translated into automata. The automata are automatically extended with transitions to model restart in all upstream states for each control system state where it is assumed that an error can occur. The proposed method enables alternative operation sequences and restart of multiple resources, and is not limited to straight sequences nor to restart of a single resource as in [5].

Not all upstream states are, however, valid as restart states due to the dependencies and the reexecution requirements. Therefore, a supervisor [26] is synthesized for the automata and the valid restart states are derived from this supervisor. Any supervisor synthesis algorithm can be used and not just a modified monolithic synthesis algorithm as in [5]. Thus, more efficient algorithms such as compositional synthesis [27] and/or symbolic synthesis [28] can be used.

When restarting the control system from a valid restart state the nominal production can start immediately, no reduced start-up pace is required as in [5]. Moreover, the restart states are connected to the control system states and not to the specific errors that have been detected. Thus, the method can handle restart after unforeseen errors.

With the restart states precalculated, the online restart phase is reduced to four straightforward steps. First, the operator selects a restart state from the precalculated ones, which can for example be stored in a database connected to the control system. Second, the active state of the control system is updated to the selected restart state. Thus, it is assumed that a mechanism for state transition is available in the control system. Third, the operator places the manufacturing system in a physical state corresponding to the selected restart state; the operator is beneficially guided by instructions for how to reach this physical state. Finally, the nominal production can be (re)started by the operator.

To simplify for an operator during the third step, when placing the manufacturing system in a physical state, the calculated restart states may be *filtered*. This paper shows filtering of restart states based on physical states in the manufacturing system that are easily accessible, and the restart states that minimize the number of resources to be placed during the restart phase.

In addition, it is shown how to *adapt* the nominal production to always enable desirable restart states, if there is at least one operation sequence in the system where the desired state is valid. This is accomplished by the uncontrollability property of the supervisory control theory [26].

The paper is organized as follows. Preliminaries are given in Section 2. Section 3 introduces an example upon which the results are projected throughout the paper. General online error recovery with focus on the restart phase is discussed in Section 4. Section 5 will thereafter present how the calculation of restart states is formulated as a synthesis problem, without any reexecution requirements. Filtering of restart states is discussed in Section 6. The reexecution requirements are then included in the calculation in Section 7. How to adapt the nominal production to always enable desirable restart states is presented in Section 8. Section 9 gives some concluding remarks and future ideas.

2. Preliminaries

This section presents conventions and notations used in this paper. First, the modeling formalism is presented. Thereafter, this formalism is used to model the operations for a manufacturing system.

2.1. Automata and the supervisory control theory

Definition 1 ***Finite automaton*** *A finite automaton is a 5-tuple:* $A := \langle Q_A, \Sigma_A, \delta_A, q_A^0, Q_A^m \rangle$ *where* Q_A *is the non-empty finite set of states;* Σ_A *is the non-empty finite set of events (the alphabet);* $\delta_A : Q_A \times \Sigma_A \rightarrow Q_A$ *is the partial transition function;* $q_A^0 \in Q_A$ *is the initial state; and* $Q_A^m \subseteq Q_A$ *is the set of marked states.*

A transition $\langle q, e, p \rangle \in \delta_A$ is said to be *fireable* when the *active state* of the automaton A coincide with the *source state* q. When the transition is *fired* the active state of the automaton A is *updated* to the *target state* p. Let $\delta_A(q, e)!$ denote that an event e is *defined* from a state q in an automaton A. The *active event function* $\Gamma_A : Q_A \rightarrow 2^{\Sigma_A}$ returns the set of events defined from a state q in A, $\Gamma_A(q) := \{e \in \Sigma_A | \delta_A(q, e)!\}$.

The set of all finite sequences of events over an alphabet Σ_A including the empty sequence, ε, is denoted Σ_A^*. An element $s \in \Sigma_A^*$ is called a *string*. For two strings $t \in \Sigma_A^*$ and $u \in \Sigma_A^*$ the *concatenation* tu is also in Σ_A^*. The *closure* of a string s is denoted s^*, such that $s^* = \{\varepsilon, s, ss, \dots\}$. The transition function is extended to strings, such that $\delta_A(q, \varepsilon) = q$, and $\delta_A(q, es) = \delta_A(\delta_A(q, e), s)$. A state $q \in Q_A$ is then *reachable* in the automaton A if $\exists s \in \Sigma_A^*$ such that $\delta_A(q_A^0, s) = q$.

A *language*, denoted $L(A)$, is the set of strings generated from the initial state q_A^0 of the automaton A. Given an alphabet Σ_B, $L(A)\Sigma_B$ represents the concatenation of all strings in $L(A)$ with all events in Σ_B. The *marked language*, $L^m(A) \subseteq L(A)$, is the set of strings generated from the initial state reaching a marked state. The *prefix closure* of the marked language, denoted $\overline{L^m(A)}$, is the set of all prefixes $\overline{L^m(A)} := \{t \in L(A) | tu \in L^m(A), u \in \Sigma_A^*\}$.

Interaction of two automata is modeled by full synchronous composition [29].

Definition 2 ***Full synchronous composition*** *(FSC) The full synchronous composition of two automata* A *and* B *is defined as* $C := A||B$ *where* $Q_C := Q_A \times Q_B$; $\Sigma_C := \Sigma_A \cup \Sigma_B$; $q_C^0 := \langle q_A^0, q_B^0 \rangle$; $Q_C^m := Q_A^m \cap Q_B^m$; *and* $\delta_C(\langle q_A, q_B \rangle, e) := \begin{cases} \langle \delta_A(q_A, e), \delta_B(q_B, e) \rangle & e \in \Gamma_A(q_A) \cap \Gamma_B(q_B) \\ \langle \delta_A(q_A, e), q_B \rangle & e \in \Gamma_A(q_A) \setminus \Sigma_B \\ \langle q_A, \delta_B(q_B, e) \rangle & e \in \Gamma_B(q_B) \setminus \Sigma_A \\ \textit{undefined} & \textit{otherwise} \end{cases}$

The *supervisory control theory* (SCT) [26] is a model-based framework for automatic calculation of discrete event controllers. Given a system to be controlled, a *plant* P, and the intended behavior, a *specification* Sp, a *supervisor* S may be synthesized, such that the behavior of $P||S$ always fulfills Sp. In terms of languages $L(P||S) \subseteq L(P||Sp)$ and $L^m(P||S) \subseteq L^m(P||Sp)$. The supervisor is both *non-blocking* and *controllable* [26].

Non-blocking: The supervisor S guarantees that at least one marked state may be reached from every state in the system $P||S$. This liveness property may formally be expressed as: $\overline{L^m(P||S)} = L(P||S)$

Controllable: In SCT, a subset of the events $\Sigma_P^u \subseteq \Sigma_P$ is said to be uncontrollable. The supervisor S is never allowed to disable an uncontrollable event that might be generated by the plant P. With the assumption that $\Sigma_S \subseteq \Sigma_P$, this safety property may formally be expressed as: $L\big(P||S\big)\Sigma_P^u \cap L\big(P\big) \subseteq L\big(P||S\big)$

Moreover, the supervisor is *minimally restrictive*, meaning that the plant is given the greatest amount of freedom to generate events without violating the specification. To facilitate the modeling, both the plant and the specification are often given as a set of automata that communicate through FSC. In the following, it is thus assumed that the system is modeled by several plants and specifications.

The focus of this paper is on calculating how a control system can be restarted. This problem is solved through synthesis of a supervisor and succeeding interpretation of the generated supervisor. Thus, any synthesis algorithm can be used to calculate the supervisor. In the following, the supervisor for $P||Sp$ is assumed to be given as $S = \mathcal{CNB}\big(P||Sp\big)$, where $\mathcal{CNB}$ represents any synthesis algorithm. However, to facilitate the interpretation it is assumed that the supervisor is characterized through *guard extraction* [28].

In the guard extraction all controllable events are appended with a guard. Each guard is a boolean function that maps a state in $P||Sp$ to either *true* or *false*. A transition is said to be *enabled by the supervisor* if the guard for the labeling event is true in the source state of the transition.

Note that, the algorithm for guard extraction proposed in [28] confronts the state-space explosion problem by using a symbolic representation of the full synchronous composition of the plants and the specifications. A user enters a set of automata and the supervisor is returned as a set of guards for the controllable events. Thus, guard extraction gives a concurrent model of modular automata.

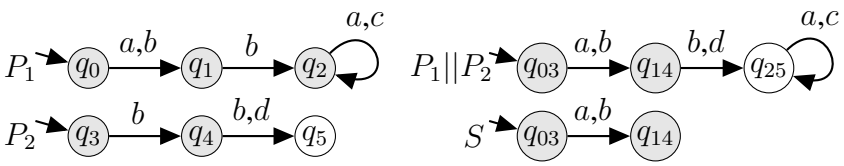
A controllable event is termed an *always enabled event* if all transitions that are labeled by the event and having source states that are reachable in the supervised system are enabled by the supervisor. In contrast, the event is termed a *sometimes enabled event* if some but not all these transitions are enabled by the supervisor.

Guard extraction and the event terminology are exemplified on the supervisor for the two automata P_1 and P_2 in Figure 1. Marked states are shaded in gray and all events are controllable. The automaton $P_1||P_2$ is the FSC of P_1 and P_2. Since the state q_{25} is blocking it is removed in the supervisor $S = \mathcal{CNB}\big(P_1||P_2\big)$. Thus, in $P_1||P_2$ only the two transitions $\langle q_{03}, a, q_{14}\rangle$ and $\langle q_{03}, b, q_{14}\rangle$ are enabled by the supervisor.

In this example, the single always enabled event is a since one of the two transitions that are labeled by a in the FSC is enabled by the supervisor and the source state for the other transition is not reachable in the supervised system (the supervisor). The event b is sometimes enabled because only one of the two transitions having source states that are reachable in the supervised system and are labeled by b in the FSC is also enabled by the supervisor. The guard for b is not satisfied in the state q_{14}. The *disabled events* c and d are neither always nor sometimes enabled.

Forbidden state combinations will be used in Section 5 to model the (un-)desired behavior. In [30] a method based on uncontrollability is presented for how to specify states locally in a set of automata such that the combination of these states are never reached in the supervised system. In the following, it is assumed that this or a similar method is used to encode given forbidden state combinations into the SCT framework.

Figure 1. The events a and b are always and sometimes enabled events, respectively.



2.2. Model of the system

In this paper, the control system for a manufacturing system is based on a set of *operations*, denoted Ω. These operations model the processes and tasks that are to be executed in order to refine a product. The basic assumption is that all operations are executed in parallel. This parallel execution of the operations can be restricted by *dependencies*.

The manufacturing system contains a set of *resources*, denoted $\mathcal{R}$. It is the resources that (physically) *realize* the operations. The resources required to realize an operation $k \in \Omega$ is denoted $\mathcal{R}_k$, such that $\mathcal{R}_k \subseteq \mathcal{R}$.

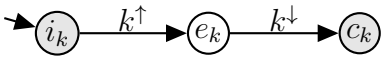
To better understand how the different dependencies affect the relations between the operations, it is beneficial to visualize subsets of operations from Ω in different projections [31]. Examples of such projections are the operations related to the main product flow or the operations realized by a specific resource. Throughout this paper, the graphical language *Sequences of Operations* introduced in [12] is used for the visualization of operations. Each visualization is referred to as a *sequence of operations* (SOP).

An operation $k \in \Omega$ may formally be modeled by an automaton, a so called *operation automaton*.

Definition 3 ***Operation automaton*** *The automaton for an operation k is denoted A_k where $Q_{A_k} := \{i_k, e_k, c_k\}$; $\Sigma_{A_k} := \{k^{\uparrow}, k^{\downarrow}\}$; $\delta_{A_k} := \{\langle i_k, k^{\uparrow}, e_k\rangle, \langle e_k, k^{\downarrow}, c_k\rangle\}$; $q^0_{A_k} := i_k$; and $Q^m_{A_k} := \{i_k, c_k\}$.*

The automaton A_k is illustrated in Figure 2. The three states denote that the operation is initial (not started), executing, and completed, respectively. The two events in Σ_{A_k} are called *operation events*.

Figure 2. An operation k modeled by an automaton A_k.



Given the automaton for a single operation, the FSC of all automata for the operations in Ω can be defined. Note that, from a practical point of view an explicit representation of the complete state-space during synthesis is to be avoided.

Definition 4 ***FSC of operation automata*** *The FSC of all automata for the operations in Ω is defined as: $A_\Omega := ||_{k\in\Omega}\, A_k$.*

The operation progress for a system may then be given through the states in A_Ω.

Definition 5 ***Operation progress*** *For each state $q \in Q_{A_\Omega}$, three disjoint sets for the operation progress, the set of operations in their respective initial, executing, and completed state, denoted Ω^i_q, Ω^e_q, and Ω^c_q,*

are defined as:

$$\begin{cases} \Omega_q^i := \{k \in \Omega | i_k \in q\} \\ \Omega_q^e := \{k \in \Omega | e_k \in q\} \\ \Omega_q^c := \{k \in \Omega | c_k \in q\} \end{cases}$$

The relation between an executing operation and the history of operation progress is captured by the definition of upstream states for an operation.

Definition 6 ***Upstream states for an operation*** *Let* $Q_{A_\Omega}^{e_k} := \{q \in Q_{A_\Omega} | k \in \Omega_q^e\}$ *be the set of states in* A_Ω *where the operation* k *executes. For a state* $p \in Q_{A_\Omega}^{e_k}$*, a state* $u \in Q_{A_\Omega}$ *is upstream of operation* k *if* $\exists s \in \Sigma_{A_\Omega}^*$ *such that* $\delta_{A_\Omega}(u, s) = p$*, and* $\Omega_u^e \cap \Omega_p^c = \emptyset$*, and* $k \in \Omega_u^i$*.*

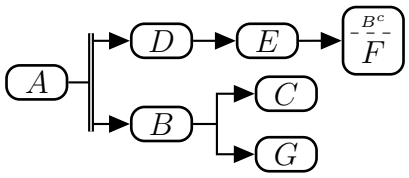
It follows from Definition 3 that each operation automaton contains a straight sequence of operation events and because the two states p and u must be connected through a string of operation events, all operations that are initial in the state p are initial in the upstream state u. With the same argument, the operations that are executing in p can either be initial or executing in u except for the operation k that is required to be initial in u. Moreover, the empty intersection in Definition 6 adds a requirement on the completed operations in p, they cannot be executing and must therefore be initial or completed in u.

3. Illustrating example

Throughout this paper, the proposed method for calculating restart states is illustrated by the example introduced below.

Example 1 *A manufacturing system comprises three resources,* $\mathcal{R} = \{R1, R2, R3\}$*, and its control system is modeled by seven operations,* $\Omega = \{A, B, C, D, E, F, G\}$*. The dependencies between the operations are visualized in different projections in the SOPs in Figures 3 and 4. The three SOPs in Figure 4 show which resources that realize the different operations.*

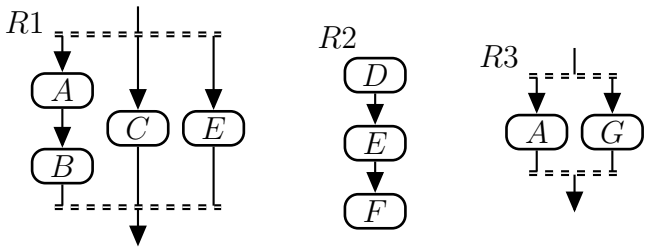
Figure 3. Dependencies between the operations in Example 1.



In the language Sequences of Operations, the dependencies between operations may be visualized both with expressions and graphical notations [12]. An arrow visualizes a precedence dependency. As an example, operation E cannot start before operation D is completed. The expression B^c is also a visualization of a precedence dependency, that operation F cannot start before operation B is completed. The double bar visualizes that the two branches, starting with operations D and B, may execute in parallel, when operation A is completed. An alternative dependency is visualized by a single bar, thus only one of the operations C and G may execute. The double dashed bars (see Figure 4) visualize arbitrary order dependencies (mutual exclusion). All operations in each branch shall execute, but no branches execute in parallel.

Arbitrary order dependency is used to model resource allocation. Each resource can realize one operation at a time. Resource allocation and deallocation takes place in the first and in the last operation, respectively, in each branch. As an example, resource $R1$ is allocated when the operations A, C, and E start and is thereafter deallocated when the operations B, C, and E complete. Thus, resource $R1$ is allocated in the state where A is completed and B is initial.

Figure 4. Arbitrary order dependencies for resource allocation in Example 1.



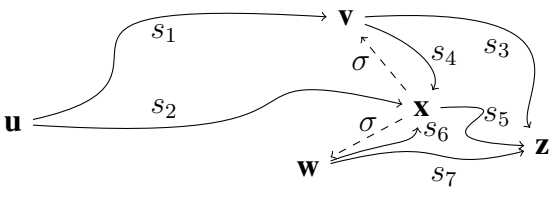
4. Error recovery in a manufacturing system

This section presents error recovery in a manufacturing system. It is shown how restart states are used *online* after an error has been detected and corrected, and the system is to be restarted. The *offline* calculation of restart states is described in Section 5.

4.1. The nominal production

Let the control system for a manufacturing system be modeled by a set of operations Ω. The *nominal production*, i.e. production according to the original production plan, can then be represented by strings of operation events, see Figure 5 where $s_i \in \Sigma^*_{A_\Omega}$. In the absence of errors, the production is given as an element in $\{s_1s_3, s_1s_4s_5, s_2s_5\}$, between an initial state, denoted **u**, where none of the operations have started to a completed state, denoted **z**, where a (user-defined) subset of the operations have completed. Let **x** denote an error state and **v** and **w** denote restart states. The event σ is a general placement event. Error states, restart states, and placement events are explained later in this section.

Figure 5. The production described by strings.



4.2. Control system states and physical states

A *control system state* is a state $q \in Q_{A_\Omega}$ and is thus a composition of operation states. Similarly to an automaton, at all times during the production, a single control system state is *active* in the control system. When the operations are executed, the active state of the control system is *updated*.

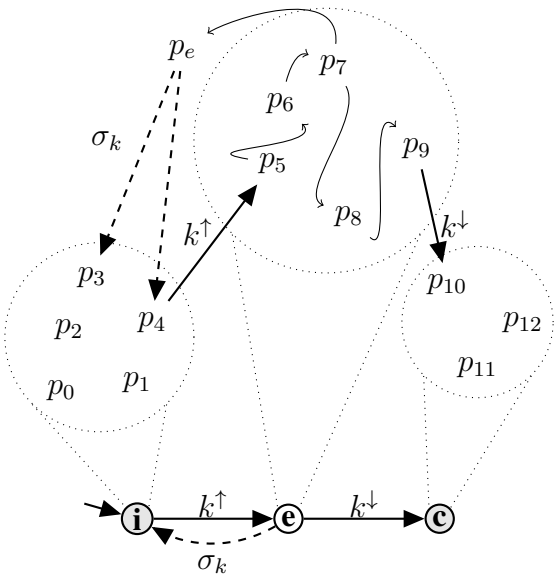
For the sake of control and supervision, the resources and the product(s) in the manufacturing system are abstracted into a set of *physical states*. A physical state is thus capturing the current position of products and which of the resources that are idling, but disregards if for example a fan in a control-cabinet is on or off and the age of the resources.

Typically, many physical states *correspond* to each control system state. During the nominal production, the control system state evolves in synchrony with the corresponding physical states. The connection between control system states and physical states (p_i) is illustrated in Figure 6 and is further discussed in the remainder of this subsection. The dashed transitions and the physical state p_e will be explained later in this section.

For clarity of presentation, the connection between control system states and physical states is first discussed with respect to the hypothetical case that the control system is modeled by a single operation and is thereafter discussed with respect to the realistic case where the control system is modeled by multiple operations.

In the hypothetical case that the control system is modeled by a single operation k, the states of the control system coincide with the states of k. Initially the control system state is **i**. No product refinement has started and all resources in the manufacturing system are idling. Thus, the manufacturing system is modeled by the single physical state. Therefore, for this hypothetical case, p_0 to p_4 denote the same physical state in Figure 6. When the operation k is started, the active state of the control system is updated to **e**. During execution of the operation, the manufacturing system will change between many physical states. The states p_5 to p_9 illustrate such physical states corresponding to the current control system state. When the operation k is complete, the active state of the control system is updated to **c**. Since the product refinement is complete, the manufacturing system is once again modeled by a single physical state. Thus, p_{10} to p_{12} denote the same physical state.

Figure 6. Mapping between states for an operation k, bottom, and physical states, top. σ_k is a placement event for k.



In the realistic case where the control system is modeled by multiple operations, the operation k in Figure 6 illustrates one of the operations in the control system. For this case, the illustrated states **i**, **e**, and **c** are operation states in k and not control system states. A subset of the operations in Ω are executed before and after the execution of operation k, respectively. Thus, the manufacturing system will change between several physical states when these other operations are executed, whilst k is initial and complete. In Figure 6, this is illustrated by the several physical states corresponding to the states **i** and **c**.

4.3. Detection, diagnosis, and correction of errors

The online error recovery starts when an error is detected through some diagnostic procedure and the system is stopped. As in [13] it is assumed in this paper that the error can only occur when one or more of the resources are realizing operations. Thus, at least one operation in Ω is in its executing state. It is therefore assumed that the error may be linked to one *error operation* that uses the faulty resource(s) for its execution.

An error may then be seen as a physical state of the manufacturing system that does not correspond to the executing state for the error operation. In Figure 6 such a non-corresponding physical state is denoted p_e. A control system state containing the executing state for the error operation is referred to as an *error state*. An error state is denoted by an **x** in Figure 5.

After the detection and the diagnosis phases, the manufacturing system is to be corrected. As pointed out in [9], errors that cannot be foreseen often require manual intervention during the correction phase. It may sometimes be advantageous to place a faulty resource in a state that facilitates correction. Thus, it is reasonable to assume that the physical state after the correction phase does not correspond to the control system state in the stopped manufacturing system. Thus, the control system and physical system are unsynchronized [9].

Mechanisms to detect, identify, and correct errors are outside the scope of this paper. In the following discussion, it is therefore assumed that such mechanisms exist in the manufacturing system. Detection and diagnosis are among others discussed by [32] and [33].

4.4. The restart phase

After the correction phase, the manufacturing system is to be restarted in order to continue the nominal production. Since neither the error nor the physical state after the error are known beforehand [9], the aim of the restart phase is to place the manufacturing system into a physical state and update the control system to a control system state from where the production may continue and eventually complete. Such a control system state is referred to as a *restart state*.

As a consequence of an error, the intended execution may not have been performed. Thus, it may be desirable to reexecute, at least, the error operation. Therefore, only restart in states upstream of the error operation is discussed in this paper.

As already mentioned, the online restart phase consists of four steps. First, the operator selects a restart state from the set of precalculated restart states for the error operation. Second, the active state of the control system is updated to the selected restart state. Third, the operator places the manufacturing

system in a physical state corresponding to the selected restart state. Thereafter, the nominal production may be (re)started by the operator.

This paper follows the terminology in [9] and terms the processes in the second and the third steps *placement*. Thus, placement implies that the active state of the control system is updated from an error state to a restart state and that the manufacturing system is placed in a physical state corresponding to this restart state, see Figure 6. Throughout this paper, placement is graphically represented by dashed transitions labeled by σ, or σ_k when the placement is connected to a specific (error) operation k. This connection between placement and error operations is thoroughly explained in the next section. In general, as will also be seen in the next section, an operation will have several restart states, where each state has a corresponding physical state. This multiplicity is reflected in the two placement transitions for the physical states in Figure 6.

For the general error state **x** in Figure 5, placement in two types of restart states may be possible; restart states that are reachable and unreachable from the initial state using strings of operation events, denoted **v** and **w**, respectively. Regardless of the type, a restart state is always upstream from the error operation and enables the nominal production to continue and eventually reach **z**.

Finally, let the letters and strings in Figure 5 constitute states and events for an automaton A where $q_A^0 = \mathbf{u}$ and $Q_A^m = \{\mathbf{z}\}$. The set of possible production sequences in the production plan may then be given as the marked language $L^m(A)$.

Note that, in the special case of a straight production sequence, without parallelism and alternatives, there exist no strings s_6 and s_7 and the strings s_2 and s_3 coincide with s_1s_4 and s_4s_5, respectively. The production plan is then given as $s_1s_4(\sigma s_4)^*s_5$, and is the single type of production plan that is possible in [5]. Thus, the proposed method is more general and is not limited to straight production sequences.

5. To calculate restart states

This section presents how to offline calculate restart states for the given set of operations Ω respecting their dependencies and reexecution requirements. As indicated in Section 4.3, it is assumed that an error can only occur if at least one operation is executing. All control states containing at least one executing state are therefore considered as *potential error states*, and analogously, the executing operations in these states are *potential error operations*. Moreover, from Section 4.4, to make up for the possible unperformed refinement due to an error only the upstream states of an error operation are to be evaluated as restart states.

Respecting these two intentions, the overall idea in the proposed method is to model restart in upstream states from potential error states by transitions in an automata model of the control system, so called *placement transitions*. However, due to the dependencies and the reexecution requirements, not all upstream states can be used as restart states. Therefore, a supervisor [26] is synthesized for the automata and the *valid restart states* for each potential error state can then be derived as the target states for the placement transitions that are enabled by the supervisor. These valid restart states can thereafter be used online as described in the preceding section.

It is fruitful to see the automata model of the control system as a composition of three submodels. First, a *nominal model* that describes the nominal production in the manufacturing system. Second,

a *placement model* that models the restart. Finally, a *reexecution model* that describes reexecution requirements on the operations. For clarity, synthesis is first discussed without the reexecution model. The reexecution model is thereafter included in the supervisor synthesis in Section 7.

It is quite common in graph based restart methods to include additional error states and/or augmentations for recovery procedures, see for example [3,8,15,25,34]. Augmentations will most likely increase the state-space of the models. As indicated in Figure 6 and explained in this section, the automata used in this method have no explicit error states. The purpose of the models for the offline analysis is only to capture how the control system states may be updated and not why, thus no additional states are necessary.

5.1. The nominal model

The nominal model consists of two automata that are synchronized, A_Ω and A_{nom}. From Definition 4, $A_\Omega = ||_{k\in\Omega} A_k$. The automaton A_{nom} models the dependencies between the operations. Since each dependency will be modeled by a single specification automaton, A_{nom} is the full synchronous composition of all these automata. The proposed method supports three types of dependencies: *precedence*, *alternative*, and *arbitrary order*. In addition, a user can also specify which operations that are *forced to complete* in order for the product refinement to be complete.

Figures 3 and 4 show the three types of dependencies graphically in the language Sequences of Operations. The operations in these SOPs will be used throughout this section to illustrate how the different types of dependencies are modeled by automata. Moreover, the automata can be generated automatically given the dependencies between the operations.

As will be seen, the dependencies are modeled by forbidden state combinations, introduced at the end of Section 2.1. Using forbidden state combinations guarantees that only the placement transitions having target states that are valid with respect to all dependencies are enabled by the supervisor. If this aspect was not respected, the supervisor could allow restart states, through the placement transitions, from where additional restart is the only possible outcome; this is of course to be avoided.

5.1.1. Precedence dependency

The precedence dependency between the two operations D and E, see Figure 3, where D is to be executed before E, may be modeled by four forbidden state combinations as: $\{(e_E, i_D), (e_E, e_D), (c_E, i_D), (c_E, e_D)\}$. When D is initial or executing, E has to be initial. E may leave its initial state, only when D has completed.

5.1.2. Alternative dependency

The alternative dependency between the two operations C and G, see Figure 3, may be modeled by four forbidden state combinations as: $\{(e_C, e_G), (e_C, c_G), (c_C, e_G), (c_C, c_G)\}$. When one of the operations starts to execute, the other must remain initial.

An alternative between a set of operations $\mathcal{O} \subseteq \Omega$, is then modeled by an alternative dependency between each pair in the set $\mathcal{O}$. In total ($|\mathcal{O}|$ binomial 2) pairs are required.

5.1.3. Arbitrary order dependency

The arbitrary order dependency between the two operation sets $\{A, B\}$ and $\{C\}$, see the leftmost SOP in Figure 4, may be modeled by seven forbidden state combinations as: $\{(i_A, e_B, e_C), (i_A, c_B, e_C), (e_A, i_B, e_C), (e_A, e_B, e_C), (e_A, c_B, e_C), (c_A, i_B, e_C), (c_A, e_B, e_C)\}$. The combinations require that both A and B have to be initial or completed when C is executing, and the opposite, that C has to be initial or completed when A and B are not both initial or completed. Note that, the combinations add no dependency between A and B.

Arbitrary order dependencies between multiple operation sets, as in the SOP for $R1$ in Figure 4, is modeled by an arbitrary order dependency between each pair of the operation sets.

5.1.4. Forced to complete

In the generic operation automaton, Definition 3, both the initial and the completed states are marked. The supervisor is non-blocking, thus by removing the marking from the initial state, the operation is forced to eventually reach its completed state in the synthesized supervisor.

To force one of the operations in an alternative to complete, this removing of marking does not work. Instead the forcing can be modeled through a specification automaton. Figure 7 illustrates such an automaton for the case when one of the operations C or G, in Example 1, is forced to complete. The effect of the automaton is that no states comprising both i_C and i_G will be marked, thus one of the operations must complete.

Figure 7. Specification for forcing one of the operations C or G to complete.

5.2. Nominal model for Example 1

The four SOPs in Figures 3 and 4 constitute the dependencies for the system in Example 1. Table 1 shows how the dependencies are modeled by forbidden state combinations. Rows 1-7, 8, and 9-12 model precedence, alternative, and arbitrary order dependencies, respectively.

To capture that the operations $\{A, B, D, E, F\}$ are forced to complete, the marking is removed from the initial state in the corresponding five automata. As indicated, to capture that one of the operations C or G are forced to complete, the specification automaton in Figure 7 is added to A_{nom}.

5.3. The placement model

The placement model is the nominal model extended with additional transitions, so called *placement transitions*. The construction of these placement transitions builds on the definition of upstream states, Definition 6. The intention with each placement transition is to *reset* to their initial states a potential error operation plus a subset of non-initial operations in the potential error state. The active state of the control system is thereby updated to an upstream state with respect to this potential error operation.

Table 1. Forbidden state combinations to model the dependencies for Example 1.

1	$(e_B, i_A), (e_B, e_A), (c_B, i_A), (c_B, e_A)$
2	$(e_D, i_A), (e_D, e_A), (c_D, i_A), (c_D, e_A)$
3	$(e_C, i_B), (e_C, e_B), (c_C, i_B), (c_C, e_B)$
4	$(e_G, i_B), (e_G, e_B), (c_G, i_B), (c_G, e_B)$
5	$(e_F, i_B), (e_F, e_B), (c_F, i_B), (c_F, e_B)$
6	$(e_E, i_D), (e_E, e_D), (c_E, i_D), (c_E, e_D)$
7	$(e_F, i_E), (e_F, e_E), (c_F, i_E), (c_F, e_E)$
8	$(e_C, e_G), (e_C, c_G), (c_C, e_G), (c_C, c_G)$
9	$(i_A, e_B, e_C), (i_A, c_B, e_C), (e_A, i_B, e_C), (e_A, e_B, e_C),$ $(e_A, c_B, e_C), (c_A, i_B, e_C), (c_A, e_B, e_C)$
10	$(i_A, e_B, e_E), (i_A, c_B, e_E), (e_A, i_B, e_E), (e_A, e_B, e_E),$ $(e_A, c_B, e_E), (c_A, i_B, e_E), (c_A, e_B, e_E)$
11	(e_C, e_E)
12	(e_A, e_G)

To calculate all restart states that are valid with respect to the dependencies and the reexecution requirements, the placement model must contain the placement transitions such that all potential error operations in all potential error states are connected with all possible upstream states. With such a model, synthesis can be performed to derive the valid restart states as the target states to the placement transitions that are enabled by the supervisor.

The set of possible upstream states for each (potential error) operation $k \in \Omega$ is correlated to the set of non-initial operations that can be reset to initial together with k. Let this set be denoted $\mathcal{O} \subseteq \Omega \setminus \{k\}$. For each pair $(k, \mathcal{O})$, a unique controllable event, a so called *placement event*, denoted $\sigma_{k:\mathcal{O}}$ is created.

The reset to initial for k and the operations in $\mathcal{O}$ is then accomplished by adding placement transitions labeled by $\sigma_{k:\mathcal{O}}$ to the corresponding operation automata. As pointed out in Section 4, it is assumed that the potential error operation is in its executing state when an error occurs. Thus, the reset to initial of k is therefore modeled by a transition $\langle e_k, \sigma_{k:\mathcal{O}}, i_k \rangle$ that is added to the transition function δ_{A_k}.

In order for the operations in $\mathcal{O}$ to be upstream after the reset to initial, they have to be non-initial in the potential error state. The reset of each operation $k' \in \mathcal{O}$ is therefore modeled by two transitions $\langle e_{k'}, \sigma_{k:\mathcal{O}}, i_{k'} \rangle$ and $\langle c_{k'}, \sigma_{k:\mathcal{O}}, i_{k'} \rangle$ that are added to the transition function $\delta_{A_{k'}}$.

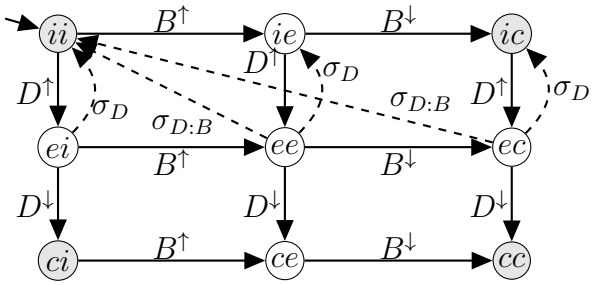
In the global automaton seen by a synthesis algorithm these locally added placement transitions will synchronize, due to the FSC, and result in a set of placement transitions. All transitions in this set will have the same target state, the upstream state that is to be evaluated as restart state for the potential error operation k.

In the following, let A_k^σ and A_Ω^σ denote the automata A_k and A_Ω extended with placement transitions. The set of placement events defined for an operation k is denoted $\Sigma_{A_k^\sigma}^\sigma$, where $\Sigma_{A_k^\sigma}^\sigma \subset \Sigma_{A_k^\sigma}$, and the set of all defined placement events is denoted $\Sigma_{A_\Omega^\sigma}^\sigma$, where $\Sigma_{A_\Omega^\sigma}^\sigma \subset \Sigma_{A_\Omega^\sigma}$, such that $\sigma_{k:\mathcal{O}} \in \Sigma_{A_k^\sigma}^\sigma \subseteq \Sigma_{A_\Omega^\sigma}^\sigma$. If all placement events are constructed, $|\Sigma_{A_\Omega^\sigma}^\sigma| = |\Omega| \times 2^{|\Omega|-1}$. Given Ω, the placement transitions can be constructed automatically and added to the operation automata.

Finally, the modeling of placement transitions is illustrated with the operations B and D from Example 1. There is no dependency between B and D. Let $A_{DB} = A_D||A_B$. For clarity, only the placement transitions for D are considered.

Since $2^{\{B,D\}\setminus\{D\}} = \{\emptyset, \{B\}\}$, the placement events for D are $\Sigma^{\sigma}_{A^{\sigma}_D} = \{\sigma_D, \sigma_{D:B}\}$. Where σ_D models reset of just D and $\sigma_{D:B}$ models reset of both D and B. Note the simplification in the indexes, if $\mathcal{O}$ is the empty set then it is removed and if it is non-empty then is written as a sequence of the elements. The automaton A^{σ}_{DB} is shown in Figure 8. The state indexes are left out.

Figure 8. Parallel execution of operations B and D. Placement transitions for D are dashed.



Operation D executes in the three states in the middle row of the automaton in Figure 8. Reset of just D is allowed in all three executing states, the transitions labeled by σ_D. Reset of both D and B is only allowed when B has started. Thus, the transitions labeled by $\sigma_{D:B}$ can be fired from the two rightmost executing states for D.

5.4. Synthesis of restart states

Given the nominal model extended with placement transitions, the restart states that are valid with respect to the dependencies are calculated through synthesis of a supervisor. Synthesis may be seen as a sieve that filters out the restart states that break at least one dependency.

Let the supervisor be denoted A_{rs}, such that $A_{rs} = \mathcal{CNB}(A^{\sigma}_{\Omega}||A_{nom})$. It is possible that the dependencies modeled by A_{nom} are too strict so that no supervisor exists [26]. In the following discussion, it is therefore assumed that A_{rs} exists.

An operation $k \in \Omega$ is coupled to its restart states by placement transitions, where each transition is labeled by a placement event $\sigma_{k:\mathcal{O}}$. Guard extraction [28] is used to find the transitions that are enabled by the supervisor. The target states of the placement transitions that are labeled by *always* and *sometimes enabled events* are the restart states that are valid with respect to the dependencies.

A transition that is labeled by an always enabled placement event can always be fired when the active state of the automaton $A^{\sigma}_{\Omega}||A_{nom}$ coincides with the source state for the transition. A transition that is labeled by a sometimes enabled placement event can, on the other hand, only be fired when the active state of $A^{\sigma}_{\Omega}||A_{nom}$ coincides with the source state for the transition and the guard for the event is satisfied in this state.

The modified synthesis algorithm presented in [5] does not preserve the state dependent behavior for the placement events. Instead, all sometimes enabled placement events are always disabled. The method

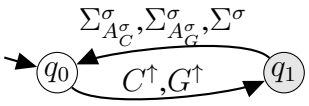
presented in this paper appends the state dependency to the placement event through the extracted guard, and thereby retains the flexibility to restart even in these cases.

5.5. Restart states for Example 1 only based on dependencies

Let $A_{ex} = \mathcal{CNB}(A_\Omega||A_{nom})$ and $A_{rs} = \mathcal{CNB}(A^\sigma_\Omega||A_{nom})$ then $|Q_{A_{ex}}| = 42$ and $|Q_{A_{rs}}| = 44$, thus the introduction of placement transitions introduces more states in the system. Inspection of A_{rs} shows that one of the two new states is used as a restart state. The state has 12 incoming placement transitions. Note that, this type of restart state is denoted by a **w** in Figure 5. Moreover, $\Sigma_{A_{rs}}$ contains 90 always and sometimes enabled placement events.

Since the specification automaton in Figure 7 contains operation events, it has to be extended with placement transitions, as shown in Figure 9, to enable that the concerned operations can be restarted. In Figure 9, $\Sigma^\sigma_{A^\sigma_C}$ and $\Sigma^\sigma_{A^\sigma_G}$ denote the placement events defined for C and G, respectively. Σ^σ denotes all placement events where C or G are among the operations to be reset, that is $\Sigma^\sigma = \{\sigma_{k:\mathcal{O}}|C \in \mathcal{O} \vee G \in \mathcal{O}\}$.

Figure 9. Specification in Figure 7 extended with placement transitions.



6. To filter out simplifying restart states

Synthesis of the nominal model extended with placement transitions will result in all restart states that are valid in the control system. Despite the fact that a control system state is a valid restart state, it can be hard, and thereby time consuming, for an operator to place the manufacturing in a corresponding physical state. Therefore, this section presents two offline approaches for how to *filter* out restart states, from this set of valid restart states, where the process to place the manufacturing in a corresponding physical state requires less effort from the operator.

In the first approach, the number of resources to be moved during the restart phase is kept at a minimum. In the second approach, all resources that are affected by the restart are placed in physical state corresponding to home-states.

6.1. To only affect resources of the error operation

To simplify for an operator online, the method in [5] aims to calculate the restart states such that only the resources used to realize the error operations are to be affected in the placement.

It is straightforward to filter out these states. The single requirement is that, all operations that are reset in a placement transition must only be realized by resources also realizing the error operation. For a general placement event $\sigma_{k:\mathcal{O}}$, it is then required that $\mathcal{O} = \{k'|\mathcal{R}_{k'} \subseteq \mathcal{R}_k\}$.

6.2. Restart from home-states

A resource is considered to be in a *home-state* when none of the various operation sequences that it can realize are executing. Due to this non-execution, it is assumed that the resource is idling in the corresponding physical states. It is also assumed that it is rather straightforward for an operator to place the resource in such an idling configuration.

To simplify for an operator, it may therefore be reasonable to filter out the restart states that restart the resources from home-states. Let the control system states corresponding to the home-states for each resource $r \in \mathcal{R}$ be given as $Q^r_{A^\sigma_\Omega} \subseteq Q_{A^\sigma_\Omega}$.

Let $q^e \in Q_{A^\sigma_\Omega}$ and $q^{rs} \in Q_{A^\sigma_\Omega}$ denote an error state and its restart state for a general placement transition $\langle q^e, \sigma_{k:\mathcal{O}}, q^{rs}\rangle \in \delta_{A^\sigma_\Omega}$. Moreover, let $\mathcal{R}_{k:\mathcal{O}} := \mathcal{R}_k \cup \left(\bigcup_{k' \in \mathcal{O}} \mathcal{R}_{k'}\right)$ denote the set of resources affected by the placement. Thus, these are the resources that are to be moved to home-states during the placement, if they are not already in a home-state.

The state q^{rs} is a home-state for the resources in $\mathcal{R}_{k:\mathcal{O}}$ if $q^{rs} \in Q^r_{A^\sigma_\Omega}\ \forall r \in \mathcal{R}_{k:\mathcal{O}}$. From Section 5.3, the operation k and the operations in $\mathcal{O}$ are reset to initial with the placement transition, thus $i_k \in q^{rs}$ and $i_{k'} \in q^{rs}\ \forall k' \in \mathcal{O}$. Then, in order to satisfy the home-state condition for the restart state q^{rs}, knowing that the operations in $\{k\} \dot\cup \mathcal{O}$ are reset to initial, the placement transition should only be fired from the states q^e where the remaining operations, $\Omega \setminus (\{k\} \dot\cup \mathcal{O})$, are in operation states such that q^{rs} is a home-state for the resources in $\mathcal{R}_{k:\mathcal{O}}$.

This requirement on the error state may be modeled by an extra home-state condition for when each placement transition can be fired. Given the connection between the operations and the home-states for the resources, it is possible to derive these home-state conditions automatically.

6.3. Home-states in Example 1

In Example 1, it is assumed that a resource is in a home-state if none of the branches in the corresponding arbitrary order SOP is active, see Figure 4. Thus, as an example, the home-states for resource $R1$ correspond to the control system states $Q^{R1}_{A^\sigma_\Omega} = \{q | \big(\big(i_A \in q \wedge i_B \in q\big) \vee \big(c_A \in q \wedge c_B \in q\big)\big) \wedge \big(i_C \in q \vee c_C \in q\big) \wedge \big(i_E \in q \vee c_E \in q\big)\}$.

Given this definition of a home-state, the home-state condition for the transition labeled by the placement event $\sigma_{E:CD}$ is discussed for demonstration. The resources to be restarted from home-states are deduced from Figure 4, this gives $\mathcal{R}_{E:CD} = \{R1, R2\}$. The operations E, C, and D will be reset to initial by the placement transition. Thus, conditions have to be added for the remaining operations that affect these two resources, that is the operations A, B, and F. Neither E, C, nor D are included in the leftmost branch of the SOP for $R1$. Thus, A and B have to be both initial or both completed. In the SOP for $R2$, F is included in the same branch as E and D, thus F has to be initial since both E and F are reset to initial.

Formally this home-state condition can now be expressed as: To place the resources $R1$ and $R2$ in home-states, the placement transitions labeled by $\sigma_{E:CD}$ can only be fired from the states q such that $\delta_{A^\sigma_\Omega}\big(q, \sigma_{E:CD}\big)!$ and $\big(\big(i_A \in q \wedge i_B \in q\big) \vee \big(c_A \in q \wedge c_B \in q\big)\big) \wedge i_F \in q$.

7. To add reexecution requirements

The reexecution requirements constrain *how many times* and *under what circumstances* an operation in Ω may be reexecuted. This section demonstrates some examples of such reexecution requirements and how each requirement can be modeled by a specification automaton in order to be included in the supervisor synthesis. By instantiating each reexecution requirement from a type library, it is possible to generate each specification automatically. Reexecution requirements other than those presented in this section can, of course, also be included, as long as the requirement can be modeled by automata.

As defined in Section 5, the full synchronous composition of all specification automata modeling reexecution requirements is called the *reexecution model* and is denoted A_{re}. The reexecution requirements do not drive the system, thus all states are marked in the reexecution model, that is $Q^m_{A_{re}} = Q_{A_{re}}$. In the following, $A_{rs} = \mathcal{CNB}\big(A^{\sigma}_{\Omega}||A_{nom}||A_{re}\big)$.

As part of the demonstration, three reexecution requirements are placed on the system in Example 1. A filtered subset of the corresponding always and sometimes enabled placement events are given in Table 2.

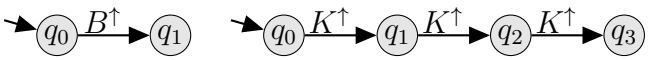
The placement events in Table 2 are filtered according to the two approaches presented in Section 6. The placement events that only affect the resources used to realize the error operation, Section 6.1, are shaded in dark gray. The events that model placement of the resources in home-states, Section 6.2, have no shade. The placement events that follow both approaches are shaded in gray. Always enabled, sometimes enabled, and disabled placement events are marked by $\forall$, $\exists$, and $-$, respectively. Column **n** shows the filtered placement events that are enabled by the supervisor when there are no reexecution requirements on the system in Example 1, Section 5.5.

7.1. Number of reexecutions

The upper limit for how many times an operation may execute is often connected to the type of process modeled by the operation. Fixation and transport of a product are examples of operations that may typically be reexecuted. Glue applying processes, on the other hand, can typically not be reexecuted.

Without any reexecution requirement, each operation may execute an arbitrary number of times. A specification for how to constrain operation B to enable zero reexecutions is shown to the left in Figure 10. B is then said to be *non-reexecutable*. A specification for how to constrain an operation K to enable at most two reexecutions is given to the right in Figure 10.

Figure 10. Specifications that operation B cannot be reexecuted and that operation K can be reexecuted at most two times.



Column **B** in Table 2 shows the always enabled placement events for the system in Example 1 when operation B is non-reexecutable. As expected, only events that do not reset operation B are enabled.

7.2. Constrained reexecution

Table 2. In white: placement in home-states, in dark gray: placement that only affect resources in error operation, and in gray: placement in home-states that only affect resources in error operation.

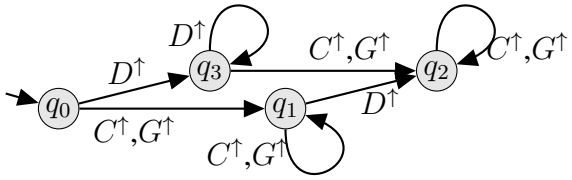
		n	**B**	**D**	$\mathcal{F}$	$\mathcal{S}$
1	σ_A	∀	∀	∀	∀	∀
2	σ_B	∀	-	∀	∀	∀
3	$\sigma_{B:A}$	∃	-	∃	∃	∃
4	$\sigma_{B:AD}$	∀	-	∃	∀	∀
5	σ_C	∀	∀	∀	∀	-
6	$\sigma_{C:AB}$	∃	-	∃	-	∃
7	$\sigma_{C:ABD}$	∀	-	-	-	∀
8	$\sigma_{C:ABDE}$	∀	-	-	-	∀
9	$\sigma_{C:ABDEF}$	∀	-	-	-	∀
10	$\sigma_{C:B}$	∃	-	∃	-	∃
11	$\sigma_{C:D}$	∀	∀	-	-	-
12	$\sigma_{C:DE}$	∀	∀	-	-	-
13	$\sigma_{C:DEF}$	∀	∀	-	-	-
14	σ_D	∀	∀	∃	∀	∀
15	$\sigma_{D:A}$	∀	∀	∃	∀	∀
16	$\sigma_{D:AB}$	∃	-	∃	∀	∃
17	$\sigma_{D:ABC}$	∀	-	-	-	∀
18	$\sigma_{D:ABG}$	∀	-	-	-	∀
19	$\sigma_{D:C}$	∀	∀	-	-	-
20	$\sigma_{D:G}$	∀	∀	-	-	-
21	σ_E	∀	∀	∀	∀	∀
22	$\sigma_{E:ABCD}$	∀	-	-	-	∀
23	$\sigma_{E:ABD}$	∃	-	∃	∀	∃
24	$\sigma_{E:ABDG}$	∀	-	-	-	∀
25	$\sigma_{E:B}$	∃	-	∃	∀	∃
26	$\sigma_{E:BC}$	∀	-	∀	-	∀
27	$\sigma_{E:BCD}$	∀	-	-	-	∀
28	$\sigma_{E:BD}$	∃	-	∃	∀	∃
29	$\sigma_{E:C}$	∀	∀	∀	-	-
30	$\sigma_{E:CD}$	∀	∀	-	-	-
31	$\sigma_{E:D}$	∀	∀	∃	∀	∀
32	$\sigma_{E:DG}$	∀	∀	-	-	-
33	σ_F	∀	∀	∀	∀	∀
34	$\sigma_{F:ABCDE}$	∀	-	-	-	∀
35	$\sigma_{F:ABDE}$	∃	-	∃	∀	∃
36	$\sigma_{F:ABDEG}$	∀	-	-	-	∀
37	$\sigma_{F:CDE}$	∀	∀	-	-	-
38	$\sigma_{F:DE}$	∀	∀	∃	∀	∀
39	$\sigma_{F:DEG}$	∀	∀	-	-	-
40	σ_G	∀	∀	∀	∀	-
41	$\sigma_{G:AB}$	∃	-	∃	-	∃
42	$\sigma_{G:ABD}$	∀	-	-	-	∀
43	$\sigma_{G:ABDE}$	∀	-	-	-	∀
44	$\sigma_{G:ABDEF}$	∀	-	-	-	∀
45	$\sigma_{G:D}$	∀	∀	-	-	-
46	$\sigma_{G:DE}$	∀	∀	-	-	-
47	$\sigma_{G:DEF}$	∀	∀	-	-	-

Constrained reexecution of an operation is often connected to the processing level of a product. For example, predecessor operations to an assembly operation can usually not be reexecuted if the assembly has started.

The specification in Figure 11 models a requirement where reexecution of operation D should be prevented when one of the operations C or G has started. If D is the first operation to execute, then the active state of the automaton is updated from q_0 to q_3. In q_3 neither C nor G has started, thus D may be reexecuted arbitrarily many times. Once C or G starts to execute, the active state of the automaton is updated from q_3 to q_2. D may not be reexecuted in q_2. If C or G is the first operation to execute, then the active state of the automaton is updated from q_0 to q_1. From the reexecution requirement, start of C or G prevents D to be reexecuted. Therefore, only a single (the nominal) start of D is enabled from q_1.

Column **D** in Table 2 shows always and sometimes enabled placement events when the specification in Figure 11 is added to the system in Example 1. Restart of the system according to the placement events

Figure 11. Specification that operation D cannot be reexecuted if operation C or G has started.



that affect the operations D, and C and/or G will reset D to initial. Once C or G has started, D may not be reexecuted. The completed state is the single marked state in D, thus in order to be non-blocking, the supervisor has to disable these placement events. Thus, all placement events that affect D, and C and/or G are marked as disabled in column **D** in Table 2.

The placement events that affect D but not C or G are marked as sometimes enabled in Column **D**. The extracted guards for these events require that neither C nor G has started. Thus, the transitions that are labeled by these events can only be fired when q_3 is the active state of the automaton in Figure 11.

7.3. Set of reexecuted operations

Another type of reexecution requirement is to demand that resetting an operation requires that a set of other operations should also be reset. For example, reseting an operation to fill a vessel could require that an operation to clean the vessel must also be reset.

A requirement that all operations in $\mathcal{O}' \subset \Omega$ are to be reset when an operation $k' \in \left(\Omega \setminus \mathcal{O}'\right)$ is reset may be modeled by a specification that disables the placement events that reset k' without resetting all of the operations in $\mathcal{O}'$.

Thus, if k' is reset by a placement transition labeled by an event $\sigma_{k:\mathcal{O}}$, that is $k' \in \left(\{k\} \dot{\cup} \mathcal{O}\right)$, then the specification should block this event if $\mathcal{O}' \not\subseteq \left(\{k\} \dot{\cup} \mathcal{O}\right)$, since otherwise the requirement is not satisfied.

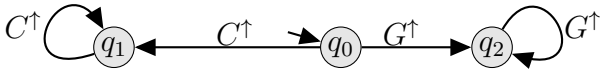
Column $\mathcal{S}$ in Table 2 shows always and sometimes enabled placement events when there is a requirement that operation B has to be reset if operations C or G are reset. As expected, Column $\mathcal{S}$ is a copy of Column **n** where all placement events that affect C or G but do not reset B, are removed.

7.4. Requirements on branches in alternatives

The proposed method supports reexecution requirements to constrain which alternative branches that are enabled in the restarted system. A constraint that the first started branch always has to be chosen during restart may be modeled by a specification as in Figure 12.

Let the two operations C and G in the alternative in Example 1 be used for demonstration. Figure 12 shows a specification that models that the first selected branch is always chosen in the restarted system. If C is the first operation to start then the active state of the automaton is updated from q_0 to q_1. Only C is allowed to reexecute in state q_1. Similarly for G in state q_2.

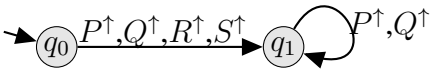
Figure 12. Specification for the alternative operations C and G.



A somewhat similar requirement is to disable a subset of alternative branches in the restarted system. This type of requirement may for example be used to disable automatic inspection in favor of manual inspection in the restarted system.

Figure 13 shows such a specification for an alternative between four operations $\{P, Q, R, S\}$, where two, R and S, are disabled in the restarted system. The active state of the automaton is updated from q_0 to q_1 when any of the four operations starts. Only P and Q may reexecute in state q_1.

Figure 13. Specification for disabling operations R and S in a restarted system.



8. To adapt a system for restart

As pointed out in Section 5.4, a transition that is labeled by a sometimes enabled placement event can only be fired when the active state of the automaton $A^{\sigma}_{\Omega}||A_{nom}||A_{re}$ coincides with the source state of the transition and the guard for the event is satisfied in this state. If a sometimes enabled placement event models a desirable restart alternative, such as restart in a home-state, it can be valuable to adapt the nominal production to always enable this event.

This section shows how the nominal production may be *adapted* such that placement events that are sometimes enabled in the supervisor become always enabled. This adaptation can be performed using the uncontrollability property of the SCT [26]. The sometimes enabled event that should be always enabled is regarded as uncontrollable and the synthesis is repeated.

The uncontrollability forces the supervisor to only allow states and transitions such that all transitions that are labeled by the selected uncontrollable placement event and having source states that are reachable in the supervised system become enabled. In this adapted system, the selected placement event is always enabled. Thus, the corresponding restart alternative is then always eligible. The system can therefore always be restarted in the desired restart state.

Note that, transitions labeled by placement events that are disabled by the synthesis cannot be enabled through this feature. These transitions can only be enabled through modifications of the dependencies and/or the reexecution requirements.

8.1. To adapt the system in Example 1

To demonstrate adaptation, assume that the placement event $\sigma_{F:DE}$ models a desirable restart alternative for operation F. The event, at row 38 in Table 2, is sometimes enabled when operation D has a constrained reexecution requirement, see Column **D**.

By regarding $\sigma_{F:DE}$ as uncontrollable and repeating the synthesis, on the original model, the system with constrained reexecution of operation D is adapted to always enable $\sigma_{F:DE}$ from the states where F executes, and D and E are non-initial. The always and sometimes enabled placement events in the adapted system are shown in Column $\mathcal{F}$ in Table 2.

As expected, transitions labeled by $\sigma_{F:DE}$ are enabled by the supervisor. In the adapted system, operations C and G may only start when operation F has completed. Thus, the adapted system has fewer states. Moreover, all placement events that affect operations C or G and at least one more operation are always disabled (rows 6, 10, 26, 29, 41). This is a consequence of the requirement to always enable reexecution of operation D as long as operation F has not completed.

The fewer states in the adapted system causes many of the sometimes enable placement events other than $\sigma_{F:DE}$ to become always enabled (rows 4, 14, 15, 16, 23, 25 28, 31, 35). This is an indirect consequence of the adaptation to make $\sigma_{F:DE}$ always enabled.

9. Conclusion

An offline method for calculating restart states for control systems modeled by operations has been presented. The derived restart states are states in the control system that can be used for restart guaranteeing that the dependencies and the reexecution requirements for the operations are followed in the restarted system. The method enables support to an operator during the online restart, which is then reduced to a process where the operator updates the active control system state to a precalculated restart state and thereafter places the manufacturing system in a physical state corresponding to the selected restart state. The restart states are calculated such that the nominal production may continue directly after the operator involvement, without any reduced start-up pace.

The method is based on the supervisory control theory. The focuses are on modeling operations, dependencies, reexecution requirements, and restart by automata and how to deduce restart states from the synthesized supervisor. The synthesis may for example be performed in Supremica[1] where it is also possible to characterize the supervisor as guards for the events. The automata generated for Example 1 are freely available[2].

Future research are concerned with practical aspects. Prototype implementations have shown that it is computationally efficient to preprocess the set of placement transitions to include in the synthesis. Therefore, it is currently investigated how the number of placement transitions to include in the model can be decreased while still guaranteeing that all restart states are eventually calculated in the synthesis. Moreover, the overall restart concept presented in this paper has been implemented and validated in a manufacturing system, containing two six-axis robots, in the Production Systems Laboratory at Chalmers University of Technology[3]. Through this proof of concept implementation, required control system

[1] A tool for formal verification and synthesis of discrete event systems. www.supremica.org

[2] http://dl.dropbox.com/u/2720019/AutomataModelsForSevenOps.wmod

[3] http://www.chalmers.se/en/areas-of-advance/production/laboratories/psl

711 augmentations as well as beneficial operator support can be studied in more detail. The generated insights
712 are valuable for future development of restart and error recovery.

713 Acknowledgements

714 This work has been carried out at the Wingquist Laboratory VINN Excellence Centre within the
715 Production Area of Advance at Chalmers. It has been supported by the European 7th FP, grant agreement
716 number 213734 (FLEXA) and Vinnova. The support is gratefully acknowledged.

717 Conflict of Interest

718 "The authors declare no conflict of interest".

719 References

720 1. Baydar, C.; Saitou, K. Off-line error prediction, diagnosis and recovery using virtual assembly
721 systems. *Journal of Intelligent Manufacturing* **2004**, *15*, 679–692.
722 2. Goh, K.; Tjahjono, B.; Baines, T.; Subramaniam, S. A Review of Research in Manufacturing
723 Prognostics. IEEE International Conference on Industrial Informatics, 2006, pp. 417–422.
724 3. Odrey, N.G. Error Recovery in Production Systems: A Petri Net Based Intelligent System
725 Approach. In *Petri Net. Theory and applications*; InTech, 2008; chapter 2.
726 4. Yalcin, A. Supervisory control of automated manufacturing cells with resource failures. *Robotics
727 and Computer-Integrated Manufacturing* **2004**, *20*, 111–119.
728 5. Andersson, K.; Lennartson, B.; Falkman, P.; Fabian, M. Generation of restart states for
729 manufacturing cell controllers. *Control Engineering Practice* **2011**, *19*, 1014–1022.
730 6. Loborg, P. Error recovery in automation - an overview. AAAI Spring Symposium on Detecting
731 and Resolving Errors in Manufacturing Systems, 1994.
732 7. Odrey, N.G.; Mejia, G. An augmented Petri Net approach for error recovery in manufacturing
733 systems control. *Robotics and Computer-Integrated Manufacturing* **2005**, *21*, 346–354.
734 8. Zhou, M.; Dicesare, F. Adaptive design of Petri net controllers for error recovery in automated
735 manufacturing systems. *IEEE Transactions on Systems, Man and Cybernetics* **1989**, *19*, 963–
736 973.
737 9. Andersson, K.; Lennartson, B.; Fabian, M. Restarting Manufacturing Systems; Restart States and
738 Restartability. *IEEE Transactions on Automation Science and Engineering* **2010**, *7*, 486–499.
739 10. Shah, S.; Endsley, E.; Lucas, M.; Tilbury, D.M. Reconfigurable logic control using modular
740 FSMs: Design, verification, implementation, and integrated error handling. American Control
741 Conference. American Automatic Control Council, 2002, Vol. 5, pp. 4153–4158.
742 11. Wen, Q.; Kumar, R.; Huang, J.; Liu, H. Fault-tolerant supervisory control of discrete event
743 systems: Formulation and existence results. Dependable Control of Discrete Event Systems,
744 2007, pp. 175–180.
745 12. Lennartson, B.; Bengtsson, K.; Yuan, C.; Andersson, K.; Fabian, M.; Falkman, P.; Åkesson, K.
746 Sequence Planning for Integrated Product, Process and Automation Design. *IEEE Transactions
747 on Automation Science and Engineering* **2010**, *7*, 791–802.

13. Loborg, P.; Törne, A. Towards error recovery in sequential control applications. International Symposium on Robotics and Manufacturing; , 1996; pp. 377–383.
14. Loborg, P.; Törne, A. Manufacturing Control System Principles supporting Error Recovery. AAAI Spring Symposium on Detecting and Resolving Errors in Manufacturing Systems; , 1994.
15. Tittus, M.; Andréasson, S.A.; Adlemo, A.; Frey, J.E. Fast restart of manufacturing cells using restart points. World Automation Congress, 2000.
16. Noreils, F.; Chatila, R. Plan execution monitoring and control architecture for mobile robots. *IEEE Transactions on Robotics and Automation* **1995**, *11*, 255–266.
17. Syan, C.; Mostefai, Y. Status monitoring and error recovery in flexible manufacturing systems. *Integrated Manufacturing Systems* **1995**, *6*, 43–48.
18. Adamides, E.; Yamalidou, E.; Bonvin, D. A systemic framework for the recovery of flexible production systems. *International Journal of Production Research* **1996**, *34*, 1875–1893.
19. Klein, I.; Jonsson, P.; Bäckström, C. Efficient planning for a miniature assembly line. *Artificial Intelligence in Engineering* **1999**, *13*, 69–81.
20. Toguyeni, A.; Berruet, P.; Craye, E. Models and algorithms for failure diagnosis and recovery in FMSs. *International Journal of Flexible Manufacturing Systems* **2003**, *15*, 57–85.
21. Bruccoleri, M.; Pasek, Z.; Koren, Y. Operation management in reconfigurable manufacturing systems: Reconfiguration for error handling. *International Journal of Production Economics* **2006**, *100*, 87–100.
22. Cox, I.J.; Gehani, N.H. Exception handling in robotics. *Computer* **1989**, *22*, 43–49.
23. Cao, T.; Sanderson, A.C. Sensor-based error recovery for task sequence plans using fuzzy Petri nets. *International Journal of Intelligent Control and Systems* **1996**, *1*, 59–82.
24. Der Jeng, M. Petri nets for modeling automated manufacturing systems with error recovery. *IEEE Transactions on Robotics and Automation* **1997**, *13*, 752–760.
25. Lee, S.; Tilbury, D.M. A modular control design method for a flexible manufacturing cell including error handling. *International Journal of Flexible Manufacturing Systems* **2008**, *19*, 308–330.
26. Ramadge, P.J.; Wonham, W.M. Supervisory control of a class of discrete event processes. *SIAM Journal of Control and Optimization* **1987**, *25*, 206–230.
27. Mohajerani, S. On Compositional Supervisor Synthesis for Discrete Event Systems. Licentiate Thesis, Chalmers University of Technology, Signals and Systems, 2012.
28. Miremadi, S.; Lennartson, B.; Åkesson, K. A BDD-based Approach for Modeling Plant and Supervisor by Extended Finite Automata. *IEEE Transactions on Control Systems Technology* **2012**, *20*, 1421–1435.
29. Hoare, C.A.R. *Communicating Sequential Processes*; Prentice-Hall International Series in Computer Science, 1985.
30. Magnusson, P.; Fabian, M.; Åkesson, K. Modular specification of forbidden states for supervisory control. Workshop on Discrete Event Systems, 2010, pp. 412–417.
31. Bengtsson, K. Flexible design of operation behavior using modeling and visualization. PhD Thesis, Chalmers University of Technology, Signals and Systems, 2012.

32. Chiang, L.; Russell, E.; Braatz, R. *Fault Detection and Diagnosis in Industrial Systems*, 1st ed.; Advanced Textbooks in Control and Signal Processing, Springer, 2001.
33. Sampath, M.; Sengupta, R.; Lafortune, S.; Sinnamohideen, K.; Teneketzis, D. Failure diagnosis using discrete-event models. *IEEE Transactions on Control Systems Technology* **1996**, *4*, 105–124.
34. Brandin, B.A. Error-Recovering Supervisory Control of Automated Manufacturing Systems. *Integrated Computer-Aided Engineering* **1996**, *3*.