Calculating restart states using reset transitions

Calculating restart states using reset transitions*

Patrik Bergagård[1], *Student member, IEEE* and Martin Fabian[1], *Member, IEEE*

***Abstract*— This paper presents a supervisory control theory based offline approach for calculating *restart states* in a manufacturing control system. Given these precalculated restart states, an operator can be given instructions for how to correctly resynchronize the control system and the manufacturing resources during the online restart phase, as part of the error recovery process. Restarting from a restart state guarantees that all requirements on the nominal and the restarted productions are fulfilled. The paper includes an empirical comparison showing that the proposed approach enables restart states calculation for systems of sizes that could not be handled using an earlier presented approach.**

I. Introduction

Downtime due to errors is costly in flexible automated manufacturing systems [1], [2]. It is therefore desirable to perform a quick and correct recovery in order to resume the production after an error.

Error recovery in manufacturing systems is, however, a complicated task [3], often divided into three major activities [4]: *detection* of discrepancies between the intended behavior of the control system and the actual behavior of the physical system, *diagnosis* to find the original fault causing the observed error leading to an unsynchronization between the control and the physical systems, and *recovery* of the systems to continue the production. Recovery is further partitioned into *error correction* to remove underlying faults and *restart* to resynchronize the control system and the physical system such that the production can be resumed [5]. The focus of this paper is on the restart phase.

The restart phase is complicated due to the existence of *reexecution requirements* specifying under what circumstances the manufacturing operations can be reexecuted in the restarted system [6], [7]. These requirements are for example related to the physical product(s). The second complication is the industrial desire to enable restart after unforeseen, non-modeled, errors [6].

To overcome these issues, different methods to manage the restart have been proposed in the literature. Extensive overviews are given in [4], [6], [8]. The main workload in a restart method is either *online* during the restart phase or *offline* before the start of production [6]. Offline methods have advantages over online methods since beforehand calculations enable the control system to be designed for restart already when the production in the manufacturing system is planned. Moreover, most industrial control systems are not powerful enough for methods that require heavy calculation online [6].

In earlier work, [9] and [10], an offline restart method has been proposed that handles reexecution requirements and unforeseen errors. The method is a generalization of the work presented in [3] and [6]. Supervisory control theory [11] is used to calculate *restart states* in the control system, that is, states from where it is *valid* to restart the system after an error such that the requirements on the nominal and the restarted productions are fulfilled. The online restart phase is then reduced to a process where the operator updates the state of the control system to a precalculated restart state and thereafter places the physical system in a corresponding physical state. The production may continue directly after the operator involvement, without any reduced start-up pace.

In [9], [10], an automata model of the control system is used during the beforehand calculation. Restart in the states that are to be evaluated as restart states are modeled by transitions from the *potential error states*, that is, the states where it is assumed that an error can be detected. These transitions modeling the restart are called *placement transitions*. Synthesis [11] is then used to derive which of the placement transitions that model restart in *valid restart states*. The major benefit with modeling restart by placement transitions is the direct connection between the valid restart states for each potential error state. The drawback, however, is that the number of transitions grows exponentially with the size of the control system model. Thus, in practice only the restart states for moderate sized systems can be calculated as the models quickly become intractable due to size.

Therefore, in this paper this static modeling of restart resulting in an overflow of placement transitions, is replaced by a modular modeling approach, still preserving the merits of the overall restart method presented in [9]. In the underlying automata model, the potential error states are connected to the states that are to be evaluated as restart states by sequences of transitions. Since the same transition can occur in many sequences, less transitions are required for the restart modeling than the corresponding number of placement transitions. Prototype implementations show that this proposed modeling approach enables restart states calculation for significantly larger control system models than what could be handled by [9], [10].

The cost for this modular approach is the absence of the direct connection between the potential error states and the restart states. The synthesized model must therefore be searched, using only the newly introduced sequences of transitions, to identify these connections.

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

[1]Department of Signals and Systems, Chalmers University of Technology {patrikm,fabian}@chalmers.se

This paper is organized as follows: Preliminaries are given in Section II. Online error recovery is discussed in Section III. Section IV presents the proposed approach for deriving the valid restart states. The approach is then compared to the approach presented in [9] in Section V. Section VI gives some concluding remarks and ideas about future work.

II. Preliminaries

First in this section, the modeling formalism is presented. Thereafter, this formalism is used to model the operations for a manufacturing system.

A. 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 a non-empty finite set of *states*; Σ_A is a non-empty finite set of *events* (the *alphabet*); $\delta_A : Q_A \times \Sigma_A \to Q_A$ 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 coincides with the *source state* q. When the transition is *fired* the active state of A is *updated* to the *target state* p. The initial active state is the initial state q_A^0. Interaction of two automata B and C is modeled by *full synchronous composition (FSC)* [12] and is denoted $B||C$. The FSC operator is associative.

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 transition function δ_A 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 A if $\exists s \in \Sigma_A^*$ such that $\delta_A(q_A^0, s) = q$.

The *supervisory control theory* [11] is a model-based framework for automatic calculation of discrete event controllers. Given a set of automata $\{P_1, \ldots, P_n\}$, a supervisor S may be synthesized, such that at least one marked state is reachable from every state in the *supervised system* $P_1||\ldots||P_n||S$.

Among others, [13] presents a method for specifying *forbidden state combinations* locally for a set of automata such that the combination of these states are never reached in the supervised system. This specification technique is exploited in Section IV.

B. Operations

The processes and tasks that are to be *executed* in order to refine a product are modeled by a set of *operations*, denoted Ω. The operations are (physically) *realized* by resources in the manufacturing system. The basic assumption is that all operations are executed in parallel. This parallel execution of the operations can be restricted by *dependencies* [14].

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

Definition 2: **Operation automaton** The automaton for an operation k is denoted A_k where $Q_{A_k} := \{i_k, e_k, c_k\}$; $\Sigma_{A_k} := \{\uparrow_k, \downarrow_k\}$; $\delta_{A_k} := \{\langle i_k, \uparrow_k, e_k \rangle, \langle e_k, \downarrow_k, c_k \rangle\}$; $q_{A_k}^0 := i_k$; and $Q_{A_k}^m := \{i_k, c_k\}$.

The automaton A_k is shown in Figure 1. The three states denote that the operation is initial (not started), executing, and completed. Marked states are shaded in gray. The two events in Σ_{A_k} are called *operation events*.

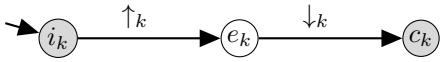

Fig. 1. Automaton model of an operation k.

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 3: **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 progress for a system may then be given through the states in A_Ω.

Definition 4: **Progress of operations** For each state $q \in Q_{A_\Omega}$, three disjoint sets for the *progress of operations*, the set of operations in their respective initial, executing, and completed state, denoted Ω_q^i, Ω_q^e, and Ω_q^c, are defined as $\Omega_q^x := \{k \in \Omega \mid x_k \in q\}$ for $x \in \{i, e, c\}$.

The relation between two states is now defined, captured by the definition of upstream states.

Definition 5: **Upstream states**[1] The set of upstream states for a state $p \in Q_{A_\Omega}$ is defined as $Q_p^{us} := \{q \in Q_{A_\Omega} \mid \exists s \in \Sigma_{A_\Omega}^* : \delta_{A_\Omega}(q, s) = p\}$.

It follows from Definition 2 that each operation automaton contains a straight sequence of operation events. The two states p and q in Definition 5 must be connected through a string of operation events. Thus, all operations that are initial in the state p are initial in the upstream state q. With the same argument, the operations that are executing in p can either be initial or executing in q, and the operations that are completed in p can be initial, executing, or remain completed in q. This upstream states definition will be useful in the restart modeling.

III. Error recovery in a manufacturing system

This section presents the *online* error recovery process using restart states according to the method presented in [9]. The *offline* calculation of these restart states, described in Section IV, differs from the approach in [9]. The online use of the calculated restart states is, however, the same.

A. 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 a string of operation events between

[1] In [9] the upstream states definition contains an extra requirement on the completed operations. In this paper, this extra requirement will be included in the static control state definition given in Section III-A.

an initial state, denoted , where none of the operations have started to a completed state, denoted , where a (user-defined) subset of the operations have completed. A *control state* corresponds to a state along such a string of operation events, and is thus a composition of operation states. Similarly to an automaton, at all times during the production, a single control 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, it is assumed that the *physical system*, i.e. the resources and the product(s) in the manufacturing system, can be abstracted into a set of *physical states*. The physical states capture aspects such as the processing level of the product(s) and the positions of the resources, but disregard aspects such as if a fan in a control-cabinet is on or off, or the aging of the resources.

Each control state *corresponds* to one or more physical states. If no errors occur during production, the active control state evolves in synchrony with the corresponding physical states, meaning that the physical system is always in a physical state corresponding to the active control state. Thus, the control system and the physical system are synchronized. In Figure 2 this synchronization is visualized by a solid line aligned with a dashed line representing the intended progress during nominal production in the control and the physical systems, respectively.

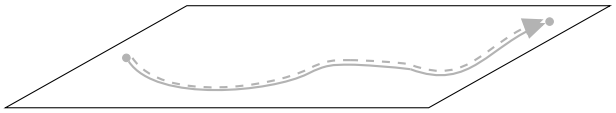

Fig. 2. The solid and the dashed lines illustrate the intended progress during the production in the control and the physical systems, respectively.

During production, resources and products are in motion only when the operations are executed. Thus, a control state containing only non-executing operations corresponds to a single physical state. Such a control state $q \in Q_{A_\Omega}$ where $\Omega_q^e = \emptyset$ is called a *static control state*.

The production will, however, not always execute as intended. A wide variety of possible faults may cause errors that result in failures in a manufacturing system [4]. For example, a part may be badly positioned in a fixture. 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 [6], [15], [16], among others. To deal with these situations, the remainder of this section presents a method for online error recovery in manufacturing systems.

B. Detection, diagnosis, and correction of errors

The online error recovery starts when an error is detected, the system is stopped, and the underlying fault causing the detected error is diagnosed. The arisen situation is illustrated in Figure 3. The thick lines visualize the progress of the two systems. As in [5] among others, it is assumed in this paper that an error may be seen as a physical state that does not correspond to the active control state . Thus, the control system and the physical system are unsynchronized, illustrated by the non-aligned lines. The active control state when an error is detected is referred to as the *error state*.

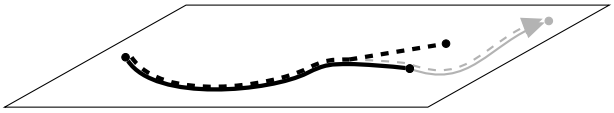

Fig. 3. An error causes an unsynchronization between the control system and the physical system. The thick lines visualize the progress.

After the detection and diagnosis phases, the manufacturing system is to be corrected. As pointed out in [6], 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. After the correction phase, the manufacturing system is to be restarted in order to continue the production.

Mechanisms to detect, diagnose, 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 [17].

C. The restart phase

Since neither the error nor the physical state after the error are known beforehand [6], the aim of the restart phase is to update the control system to a control state from where the production may continue and eventually complete, and to place the physical system into a corresponding physical state. From now on, the control state from where the control system is restarted is referred to as a *restart state* denoted . Note that, these restart states are not related to the specific errors that have been detected. Thus, restart after unforeseen errors is handled.

As a consequence of an error, the intended nominal production may not have been performed; the control and physical systems could have been unsynchronized for some time before the error was detected. Thus, it may be desirable to reexecute some of the operations. Moreover, the static control states enable unambiguous synchronization points between the control and the physical systems. Thus, restart in restart states that are static control states upstream of the error state enables reexecution of operations and placement of the physical system in definite physical states.

With the restart states precalculated, the online restart phase is reduced to four steps. Figure 4 illustrates how these steps affect the control and the physical systems. The restarted production is pictured in the uppermost plane.

In the first step, the operator selects a restart state from the precalculated ones. The restart states can for example be stored in a database connected to the control system. Second, the active control state is updated to the selected restart state, illustrated by the straight line from to . Third, the operator places the physical system in the physical state corresponding to the selected restart state , illustrated by the dashed line from to . The operator is beneficially guided by

instructions for how to reach this physical state. Finally, the production can be (re)started by the operator.

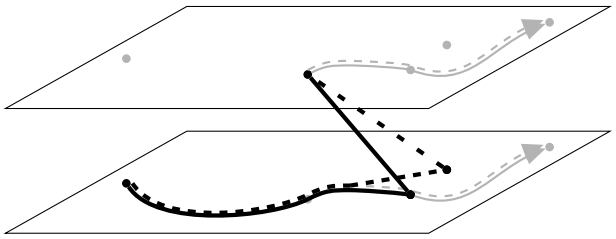

Fig. 4. The online restart phase. The control and the physical systems are resynchronized in the restart state with the corresponding physical state . The restarted production is pictured in the upper plane.

For clarity of presentation, Figure 4 shows a simplified view of the restart phase. As outlined in [9], a restart state need not be a control state passed through during the nominal production up to the error state. In some systems, some restart states are unreachable from the initial control state. Moreover, the restarted production need not follow the nominal production. To satisfy the requirements on the operations in the restarted production, it can sometimes be necessary to execute the operations in another order than the nominal.

Finally, there is the issue of *potential error states*, that is, the control states in which it is assumed that an error can be detected. In [6] all control states, except the initial control state, are potential error states. This is in contrast to [5] and [9] where it is assumed that an error can only occur when at least one resource (operation) is executing. As will be seen, the restart modeling presented in this paper supports both approaches and will thus leave to the user the decision of which approach to use.

IV. Calculating restart states using reset transitions

This section presents how to offline calculate restart states for a given set of operations Ω respecting their dependencies and reexecution requirements. As motivated in the introduction, the approach resembles the approach presented in [9] but with major differences in how the restart is modeled and how the valid restart states are derived. In the proposed approach, restart in upstream states from potential error states is modeled by sequences of transitions, so called *reset transitions*, in an automata model of the control system, the so called *control system model*. However, due to dependencies and reexecution requirements, not all upstream states are valid as restart states. Therefore, a supervisor [11] is synthesized for the control system model, and the *valid restart states* for each potential error state can be derived by searching the sequences of reset transitions that are enabled by the supervised system. These valid restart states can thereafter be used online as described in the preceding section.

The overall aim is to derive *all* valid restart states for *all* potential error states. With such an open approach, different postprocessing can be applied on the result to meet different needs. For instance, in [3] the set of valid restart states are minimized with the constraint that all potential error states have at least one valid restart state. For other systems it can be desirable to select one valid restart state for each potential error state, such that the operator action to place the physical system in a corresponding physical state becomes as simple as possible, according to some appropriate metric. Thus, by aiming at all valid restart states, the overall method is flexible and can be tailored to different types of systems and needs.

A. The control system model

The control system model is a composition of submodels, each modeled by automata. First, a *nominal model* where the operations with their dependencies model the nominal behavior of the control system. Second, a *restart model* that models the restart of the control system. And finally, a *re-execution model* that describes the reexecution requirements on the operations, that is, how many times and under what circumstances the operations can be reexecuted.

1) The nominal model: The nominal model is modeled as proposed in [9]. Thus, each operation is modeled by an operation automaton, Definition 2, and each dependency is modeled by a set of forbidden state combinations [13]. In an operation automaton both the initial and the completed states are marked. At least one marked state is reachable from every state in the supervised system [11], thus by removing the marking from the initial state, an operation is forced to eventually reach its completed state in the supervised system.

Three types of dependencies are supported by the modeling approach proposed in [9]; *precedence*, *alternative*, and *arbitrary order* dependencies. In this context of dependencies, each forbidden state combination specifies a combination of two, or more, operation states that violate a dependency. For example, two operations with an arbitrary order dependency must not be executing at the same time, thus the executing states of the two operations are specified as a forbidden state combination and simultaneous execution of the two operations are thereby avoided in the supervised system.

2) The restart model: As motivated in the preceding section, restart updates the active control state from a potential error state to a static upstream state. From the definitions of upstream states, Definition 5, and static control states, Section III-A, this state update is accomplished by resetting to their initial states operations that are executing and/or completed in the potential error state. Thus, restart in the control system model can be modeled by transitions that reset operations.

This paper proposes that this reset of operations is modeled locally in each operation automaton such that restart in the control system model is modeled by a sequence of local operation resets. Since a non-initial operation is either executing or completed, two transitions are added to each operation automaton to model the local reset. These additional transitions are called *reset transitions*. The reset transitions for an operation $k \in \Omega$ are then given as $\langle e_k, \hookleftarrow_k, i_k \rangle$ and $\langle c_k, \hookleftarrow_k, i_k \rangle$, where $\hookleftarrow_k$ is a controllable event [11] that is

unique for each operation. The event is called a *reset event*. Thus, the restart model is a set of reset transitions that are added to the operation automata in the nominal model.

3) The reexecution model: Examples of automata that model different types of reexecution requirements are given in [9]. Most of these automata contain only operation events and are thus independent of the restart model. Thus, these automata can also be used together with the restart model proposed in this paper. Any further analysis of reexecution requirements is outside the scope of this paper.

B. An example system

The control system model for a system with two operations k_1 and k_2 and the precedence dependency that k_1 must be completed before k_2 starts to execute is shown in Figure 5. The dependency is modeled by four forbidden state combinations. These combinations are crossed out in Figure 5. For clarity, the reset transitions are dotted. This example will be used throughout this section to illustrate the restart state calculation.

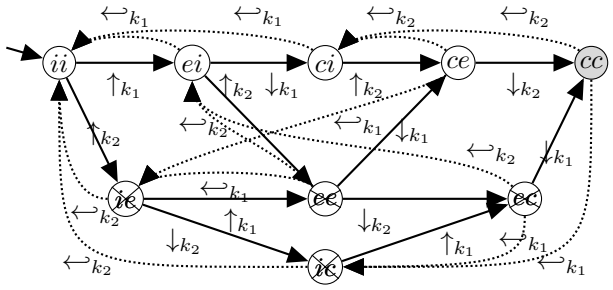

Fig. 5. Control system model for two operations k_1 and k_2 with a precedence dependency.

C. Synthesis

Given a control system model A_{csm}, a supervisor is synthesized to derive the enabled reset transitions in A_{csm} that respect all dependencies and all reexecution requirements. Let the supervised system be denoted A^{sup}_{csm}. Note, the presented modeling approach poses a general supervisory control theory problem. Thus, any synthesis algorithm can be used, such as compositional [18] and/or symbolic synthesis [19].

An automaton representation of the supervised system for the example system is given in Figure 6. The forbidden state combinations are removed.

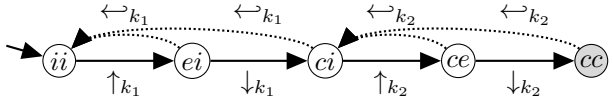

Fig. 6. The supervised system for the control system model with the two operations k_1 and k_2.

By modeling the dependencies by forbidden state combinations it is guaranteed that the supervised system will only enable restart transitions having target states that are valid with respect to all dependencies and all reexecution requirements. For all these target states, a string of operation events leads to a marked state. This is a key observation that is exploited in the last step of the restart states calculation, presented next.

D. Deriving the restart states from the supervised system

The sequences of reset transitions that are enabled by the supervised system correlates to the edges in a graph representation of the transitive closure [20] for the supervised system. Thus, the restart states are derived from the supervised system by constructing the transitive closure for A^{sup}_{csm} using only reset transitions. In the graph representation of the transitive closure, the states $Q_{A^{sup}_{csm}}$ constitute the vertices and all transitive states/vertices are connected by an edge. The set of valid restart states for a potential error state will then correlate to the set of adjacent vertices for the vertex mapping this potential error state. The directed graph in Figure 7 is the transitive closure graph for the example system, using only reset transitions.

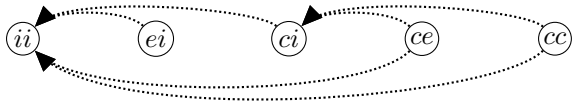

Fig. 7. The transitive closure for the supervised system for the example system using only reset transitions.

Let the subset of potential error states in the supervised system be denoted $Q^{er}_{A^{sup}_{csm}}$, where $Q^{er}_{A^{sup}_{csm}} \subset Q_{A^{sup}_{csm}}$, and let the set of valid restart states for a potential error state $\in Q^{er}_{A^{sup}_{csm}}$ be denoted Q^{rs}, where $Q^{rs} \subset Q_{A^{sup}_{csm}}$. Coming back to the discussion in the end of Section III, when all control states, except the initial state, are considered as potential error states then $Q^{er}_{A^{sup}_{csm}} = \{ei, ci, ce, cc\}$, and if only those states where at least one resource is executing are considered as potential error states then $Q^{er}_{A^{sup}_{csm}} = \{ei, ce\}$. From the adjacent vertices in Figure 7, the valid restart states for these potential error states are given as: $Q^{rs}_{ei} = Q^{rs}_{ci} = \{ii\}$ and $Q^{rs}_{ce} = Q^{rs}_{cc} = \{ci, ii\}$.

V. Empirical comparison

For comparison, the restart states for a parameterized control system model are calculated using the restart model proposed in this paper and the restart model presented in [9] with the reduction preprocess proposed in [10]. As mentioned in the introduction, in the restart model used in [9], restart is modeled by so called placement transitions from the potential error states. In this paper, on the other hand, restart is modeled by sequences of reset transitions.

The parameterized model contains a set of operations in straight parallel sequences, where the number of sequences and operations in each sequence can be varied. The example system introduced in Section IV-B contains, for example, one sequence with two operations.

Figure 8 shows the maximum number of operations $|\Omega|$ that can be equally distributed among a number of sequences $|||||$, such that the restart states can be calculated with the current implementations. As an example, when using reset transitions for the case with 3 sequences, some 54/3=18 operations can be included in each sequence, see the dotted lines in Figure 8. In the current implementation, synthesis is performed in *Supremica*[2] and the transitive closure for the supervised systems are constructed using depth-first search. The model contains no reexecution requirements and a control state is considered as a potential error state if it contains one or more executing operations.

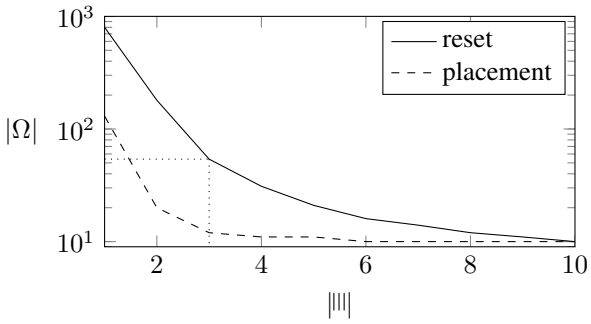

Fig. 8. Size of control system models such that the restart states can be calculated with current implementations. Solid and dashed lines represent restart modeled by reset and placement transitions, respectively.

In the calculation using placement transitions, the limiting factor is the number of such placement transitions that can be calculated according to the preprocess proposed in [10], before memory exception. When reset transitions are used, the limiting factor is the construction of the transitive closure for the supervised system, before memory exception. The trend in Figure 8 is clearly in favor of using reset transitions. Thus, the proposed restart modeling approach enables restart states calculation for models that could not be handled using placement transitions.

VI. Conclusion

An approach for restart states calculation for manufacturing control systems has been presented. These restart states are states in the control system from where it is correct to restart the manufacturing system, guaranteeing that product and process requirements for both the nominal and the restarted productions are fulfilled.

Specifically, this paper shows how to model the control system for a manufacturing system by automata, given the set of manufacturing operations describing the production, such that the restart states respecting all product and process requirements are derived. In two steps, these restart states are derived using supervisory control theory synthesis and transitive closure calculation. An included empirical comparison shows that this proposed approach enables restart states calculation for systems of sizes that could not be handled using an earlier presented approach.

For future research, the results presented in Section V builds on a monolithic supervisor representation. Instead, it is probably computationally more efficient to derive the sequences of reset transitions, and thereby the restart states, from a modular representation.

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

References

[1] C. Baydar and K. Saitou, "Off-line error prediction, diagnosis and recovery using virtual assembly systems," *Journal of Intelligent Manufacturing*, vol. 15, no. 5, pp. 679–692, 2004.

[2] K. Goh, B. Tjahjono, T. Baines, and S. Subramaniam, "A Review of Research in Manufacturing Prognostics," in *IEEE International Conference on Industrial Informatics*, pp. 417–422, 2006.

[3] K. Andersson, B. Lennartson, P. Falkman, and M. Fabian, "Generation of restart states for manufacturing cell controllers," *Control Engineering Practice*, vol. 19, no. 9, pp. 1014–1022, 2011.

[4] P. Loborg, "Error recovery in automation - an overview," in *AAAI Spring Symposium on Detecting and Resolving Errors in Manufacturing Systems*, 1994.

[5] P. Loborg and A. Törne, "Towards error recovery in sequential control applications," in *International Symposium on Robotics and Manufacturing*, (Montpellier), pp. 377–383, 1996.

[6] K. Andersson, B. Lennartson, and M. Fabian, "Restarting Manufacturing Systems; Restart States and Restartability," *IEEE Transactions on Automation Science and Engineering*, vol. 7, no. 3, pp. 486–499, 2010.

[7] P. Loborg and A. Törne, "Manufacturing Control System Principles supporting Error Recovery," in *AAAI Spring Symposium on Detecting and Resolving Errors in Manufacturing Systems*, (Stanford), 1994.

[8] E. Adamides, E. Yamalidou, and D. Bonvin, "A systemic framework for the recovery of flexible production systems," *International Journal of Production Research*, vol. 34, no. 7, pp. 1875–1893, 1996.

[9] P. Bergagård and M. Fabian, "Calculating Restart States for Systems Modeled by Operations Using Supervisory Control Theory," *Machines*, vol. 1, no. 3, pp. 116–141, 2013.

[10] P. Bergagård and M. Fabian, "Derivation of placement transitions for offline calculation of restart states," in *IEEE International Conference on Emerging Technologies and Factory Automation*, 2013.

[11] P. J. Ramadge and W. M. Wonham, "Supervisory control of a class of discrete event processes," *SIAM Journal of Control and Optimization*, vol. 25, no. 1, pp. 206–230, 1987.

[12] C. A. R. Hoare, *Communicating Sequential Processes*. Prentice-Hall International Series in Computer Science, 1985.

[13] P. Magnusson, M. Fabian, and K. Åkesson, "Modular specification of forbidden states for supervisory control," in *Workshop on Discrete Event Systems*, pp. 412–417, 2010.

[14] B. Lennartson, K. Bengtsson, C. Yuan, K. Andersson, M. Fabian, P. Falkman, and K. Åkesson, "Sequence Planning for Integrated Product, Process and Automation Design," *IEEE Transactions on Automation Science and Engineering*, vol. 7, no. 4, pp. 791–802, 2010.

[15] M. Zhou and F. Dicesare, "Adaptive design of Petri net controllers for error recovery in automated manufacturing systems," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 19, no. 5, pp. 963–973, 1989.

[16] N. G. Odrey and G. Mejia, "An augmented Petri Net approach for error recovery in manufacturing systems control," *Robotics and Computer-Integrated Manufacturing*, vol. 21, no. 4-5, pp. 346–354, 2005.

[17] L. Chiang, E. Russell, and R. Braatz, *Fault Detection and Diagnosis in Industrial Systems*. Advanced Textbooks in Control and Signal Processing, Springer, 1 ed., 2001.

[18] S. Mohajerani, *On Compositional Supervisor Synthesis for Discrete Event Systems*. Licentiate Thesis, Chalmers University of Technology, Signals and Systems, 2012.

[19] S. Miremadi, B. Lennartson, and K. Åkesson, "A BDD-based Approach for Modeling Plant and Supervisor by Extended Finite Automata," *IEEE Transactions on Control Systems Technology*, vol. 20, no. 6, pp. 1421–1435, 2012.

[20] S. Pemmaraju and S. Skiena, *Computational Discrete Mathematics: Combinatorics and Graph Theory with Mathematica*. Cambridge University Press, 2003.