Sequence Planning Using Multiple and Coordinated Sequences of Operations

Sequence Planning Using Multiple and Coordinated Sequences of Operations

Kristofer Bengtsson, Patrik Bergagård, Carl Thorstensson, Bengt Lennartson, Knut Åkesson, Chengyin Yuan, Sajed Miremadi, and Petter Falkman

***Abstract*—The sequential behavior of a manufacturing system results from several constraints introduced during the product, manufacturing, and control logic development. This paper proposes methods and algorithms for automatically representing and visualizing this behavior from various perspectives throughout the development process. A new sequence planning approach is introduced that uses self-contained operations to model the activities and execution constraints. These operations can be represented and visualized from multiple perspectives using a graphical and formal language called Sequences of Operations (SOPs).**

The operations in a manufacturing system are related to each other in various ways, due to execution constraints expressed by operation pre- and post-conditions. These operation relations include parallel, sequence, arbitrary order, alternative, and hierarchy relations. Based on the SOP language, these relations are identified and visualized in various SOPs and sequences. A software tool, Sequence Planner, has been developed, for organizing the operations into SOPs that visualize only relevant operations and relations.

***Note to Practitioners*— This paper proposes methods and algorithms for a new sequence planning approach in which sequences are automatically created based on the relations among operations instead of having to be manually constructed. Using various views, the sequences of operations related to, for example, part flow, robot operations, and operator tasks, can be visualized. The use of various views helps the user better understand the relations between cell control and mechanical design, and between product design and total system behavior.**

***Index Terms*—Automation design, control logic design, formal methods, sequence planning, sequences of operations.**

Manuscript received January 17, 2011; revised August 29, 2011; accepted November 10, 2011. Date of publication January 31, 2012; date of current version April 03, 2012. This paper was recommended for publication by Associate Editor B. Turchiano and Editor Y. Narahari upon evaluation of the reviewers' comments. This work was carried out at the Wingquist Laboratory VINN Excellence Centre within the Area of Advance Production at Chalmers, supported in part by the Swedish Governmental Agency for Innovation Systems (VINNOVA) and within the CAPE Research School, and in part by the Knowledge Foundation, and in part by General Motors and SAAB Automobile.

K. Bengtsson is with Sekvensa AB, Trollhättan 46153, Sweden (e-mail: kristofer.bengtsson@sekvensa.se).

P. Bergagård, C. Thorstensson, B. Lennartson, K. Åkesson, S. Miremadi, and P. Falkman are with the Automation Research Group, Department of Signals and Systems, Chalmers University of Technology, SE-412 96 Göteborg, Sweden (e-mail: patrik.bergagard@chalmers.se; carl.thorstensson@gmail.com; bengt.lennartson@chalmers.se; knut.akesson@chalmers.se; miremads@chalmers.se; petter.falkman@chalmers.se).

C. Yuan was with the Research and Development, General Motors, Warren, MI, USA. He is now with Pride Power System Technology Co., Ltd., Beijing, 102606 China (e-mail: chengyin.yuan@gmail.com).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TASE.2011.2178068

I. Introduction

THE famous motto "There is more than one way to do it" [27] was influential when the software language Perl was being developed. Perl's inventor, Larry Wall, a trained linguist, believed that sometimes another phrasing could make the meaning of a design clearer. The truth of this motto is even more obvious in the research field of information visualization, where Bertin stated: "A graphic is no longer *drawn* once and for all: it is *constructed* and reconstructed (manipulated) until all the relationships which lie within it have been perceived... A graphic is never an end in itself: it is a moment in the process of decision making" [23].

When developing an automation system, many engineering disciplines work together to create a functional system. Such an effort involves product designers, process planners, tool designers, robot programmers, automation engineers, logistic planners, etc. Most of these engineers have differing views of the system and the design process. It is therefore important to be able to represent design information from multiple perspectives, to make it clear to everyone in the design team.

A common method for visualizing complex information is to use multiple views to represent different projections of the information [23]. This research area, which is called coordinated and multiple views (CMVs) [17], has identified many interesting techniques and guidelines [2]. The present paper presents a tool that can be used to visualize and manage information related to sequence planning in multiple views. In particular, two CMV research areas are emphasized in this tool: data processing and preparation, and view generation.

A. Sequence Planning

The purpose of sequence planning, when developing an automation system, is to determine in what order operations could or should execute. Sequence planning integrates high-level requirements, such as cycle time and product quality issues, with low-level sequencing constraints, such as mutual exclusion, safety concerns, and actuator actions. In the present research, the operations describe the logical behavior of tasks executed by an automation system to accomplish a specific objective. The relations among operations are defined by execution conditions governing when and how an operation can execute, for example, to ensure that one particular operation always precedes another.

The challenges involved in planning operation sequences are addressed in many research areas, for example, project management [11], business process automation [6], product assembly planning [28], manufacturing task planning [18],

workflow scheduling [8], computer-aided manufacturing [14], computer-aided process planning [13], and control design [19]. Academics have focused mostly on optimization-based planning problems, but the industrial impact has been quite limited so far, probably due to the complexity of solving real problems. The industry focus has instead been on representing and visualizing sequences and tasks and on simulating them.

B. Gantt Chart

Since the operation specification of an automation system is central [3], all involved stakeholders must understand the operation specification and how various design decisions influence it. The most widespread industrial tool for specifying operations is probably the Gantt chart [29], which is easy to use and understand and intuitive to work with [9]. However, it was soon recognized that planning complex, large-scale production systems was too complicated for the Gantt chart [29]. The chart does not really represent logical relations among system elements, but rather their durations and start and stop times. For example, Gantt charts cannot handle alternatives and arbitrary order or represent interdependent and network-like relations [29].

Since sequence planning is fundamentally logical, planning using Gantt charts alone can increase the development problems. Today, it is fairly common for certain sequences to be specified early in the development process to avoid complexity and to fit them into a Gantt chart. Indirectly, this over specifies all operations. This is a problem, especially in early development phases and platform development [7], when it is important to retain as much freedom as possible in terms of parallelism, and to increase flexibility, adaptivity, and optimality. Moreover, early specification of sequences to avoid complexity can complicate the introduction of changes in the product or manufacturing process later on in the development process.

C. Method for Planning Operations and Sequences

These drawbacks tend to result in inconsistent and contradictory understandings of system behavior. Therefore, a new method for planning operation sequences has been proposed by Lennartson *et al.* [12] and Bengtsson *et al.* [4], who present a graphical and formal language, Sequences of Operations (SOPs).

The SOP language is based on self-contained operations that include only relevant conditions as to when and how every operation can execute, i.e., no information is stored dealing explicitly with a particular sequence, as in a Gantt chart. The relations within a set of operations can then be identified and visualized graphically from various perspectives, based on the constraints on each operation. Examples of these perspectives are operation sequences related to part flow through manufacturing, the execution sequence of a specific robot, and operations related to a specific safety system.

A multiplicity of perspectives helps the user better understand the relation between cell control and mechanical design, and between product design and total system behavior. The suggested method considers not only the industrial need for easy visualization and improved understanding of sequential behavior, but also the need to optimize and verify the system (e.g., [10] and [26]).

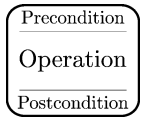

Fig. 1. An operation in the SOP language.

The SOP language focuses on how to organize, prepare, and process data related to manufacturing operations, to be able to visualize the information in multiple views. Using the SOP language and earlier research findings, this paper focuses on how to generate these multiple views, covering both necessary CMV research areas.

This paper, together with an earlier one by the same authors [5], presents algorithms for organizing operations into sequences, identifying and visualizing the relations between them automatically, and coordinating the various SOPs. These algorithms are implemented in a tool called Sequence Planner (SP) [16], which is used not only to visualize the operations in various coordinated views to help engineers plan the sequences, but also to verify, synthesize, and optimize the sequences.

The next section introduces the formal language, Sequences of Operations (SOPs). Section III demonstrates how to identify relations among operations, while Section IV presents a method for automatically visualizing sequences. The SP tool is introduced in Section V, which also describes how to manage the coordination among operations and SOPs.

II. Sequences of Operations (SOPs)

This section presents the method introduced by Lennartson *et al.* [12] and Bengtsson *et al.* [4] together with an example that will be referred to throughout the paper.

The core object of the graphical language SOPs is the operation. An operation specifies, at various levels of detail, an action or task executed by an automation system. An operation can describe, for example, the assembly of a complete product or the action of sending a signal to an actuator. This is handled by including suboperations in an operation. The level of detail of an operation specification is related to the purpose of the SOP model. Sometimes only high-level operations are relevant, but in other cases, for example, when implementing the operations in a control system, greater detail is needed.

An operation is visualized in the SOP language as in Fig. 1. Each operation includes a set of conditions, describing when and how the operation will execute. The preconditions specify when an operation can start to execute and may include guard expressions related to other operations, the state of a resource, and product or safety interlocking expressions. The preconditions also cover actions describing what should happen when the operation starts.

The operation will continue to execute until the postconditions are satisfied. Like the preconditions, the postconditions also include both guards and actions. The algorithms presented here use a formal mathematical model of the operations based on automata.

A. A Formal Operation Model

The operation model can be represented formally by extended finite automata (EFA) [21], a generalization of automata in-

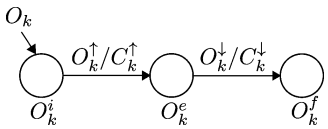

Fig. 2. EFA for operation O_k.

cluding guards and actions. The guards and actions are associated with the transitions in the automation. A transition in an EFA is enabled if and only if its corresponding guard formula is evaluated to be true; when the transition is undertaken, a set of variables may subsequently be updated. Formal tools for EFAs are available; see [15], [24], and [26]. Hence, both formal verification and synthesis can be applied directly in the suggested EFA models.

Definition 1 (Extended Finite Automaton): An extended finite automaton is a 6-tuple

$$E = \langle Q \times V, \Sigma, \mathcal{G}, \mathcal{A}, \rightarrow, (q_0, v_0) \rangle.$$

The set $Q \times V$ is the extended finite set of states, where Q is a finite set of *locations* and V is the finite domain of definition of the variables, Σ is a nonempty finite set of events (the alphabet), $\mathcal{G}$ is a set of guard predicates over V, $\mathcal{A}$ is a collection of action functions, $\rightarrow \subseteq Q \times \Sigma \times \mathcal{G} \times \mathcal{A} \times Q$ is a state transition relation, and $(q_0, v_0) \in Q \times V$ is the initial state. □

For notational purposes, the guards and actions are grouped into a set, C, of transition conditions. These transition conditions include both the current and next values of the variables after a transition. The variable values after a transition (next values) are denoted $v' \in V$. The conditions are a map $V \times V \rightarrow \mathbb{B}$, $v \in V, v' \in V$. Consider, for example, a guard, $(v_1 = 0) \wedge (v_2 = 1)$, combined with an action, $v_1 := 1; v_2 := 0$, which is expressed as a single transition condition, $(v_1 = 0) \wedge (v_1' = 1) \wedge (v_2 = 1) \wedge (v_2' = 0)$.

The following definition states how an operation can be modeled by an EFA.

Definition 2 (Operation): An operation can be described formally by an EFA where the set of locations $Q_k = \{O_k^i, O_k^e, O_k^f\}$, the event set $\Sigma_k = \{O_k^{\uparrow}, O_k^{\downarrow}\}$, the set of transition conditions $C_k = \{C_k^{\uparrow}, C_k^{\downarrow}\}$, the transition relation $\underset{k}{\rightarrow} = \{\langle O_k^i,\ O_k^{\uparrow}/C_k^{\uparrow}, O_k^e \rangle,\ \langle O_k^e, O_k^{\downarrow}/C_k^{\downarrow}, O_k^f \rangle\}$, and the initial location $q_k^i = O_k^i$ (see Fig. 2). □

First observe that the guards and actions in Definition 1 of an EFA are replaced in Definition 2 with the corresponding transition conditions. The transition for operation O_k from the initial location O_k^i to the execution location O_k^e is enabled when *precondition* $C_k^{\uparrow}$ is satisfied, after which the transition can be fired and the start event $O_k^{\uparrow}$ occurs. In the same way, completion event $O_k^{\downarrow}$ can only occur when *postcondition* $C_k^{\downarrow}$ is fulfilled. For operations that need to be repeated, the operation model in Fig. 2 is modified. A reset transition is then introduced, from O_k^e to O_k^i, such that operation O_k can be repeated from its initial location, O_k^i, when a reset event, $O_k^{\leftarrow}$, has been fired. This can happen when *reset condition* $C_k^{\leftarrow}$ is satisfied. Resettable operations are not considered in this paper.

TABLE I
NOTATIONS

SOP	Sequences of Operations	SO_k	SOP k
EFA	Extended Finite Automaton	O_k	Operation k
O_k^i	Initial location	O_k^e	Execute location
O_k^f	Finish location	$C_k^{\uparrow}$	Operation precondition
$C_k^{\downarrow}$	Operation postcondition	$O_k^{\uparrow}$	Operation start event
$O_k^{\downarrow}$	Operation stop event		

B. Condition-Based Operation Relations

The conditions are logical expressions impinging on variable set V. The operation will interact with the outside by reading and changing the values of these variables, which are defined by resources, products, and other operations. The conditions include both the current and the next values of the variables after the operation state.

To include requirements impinging on the operation locations in the pre- and postconditions, these locations are also represented as Boolean variables O_k^i, O_k^e, and O_k^f included in variable set V. Each of these variables, called operation variables, is equal to one when the corresponding location is active.

If an operation includes an operation variable in its transition condition, it is defined as *directly related* to that operation. If the precondition of O_ℓ is, for example, $C_\ell^{\uparrow} = O_k^f \vee O_j^f$, then O_ℓ is directly related to the operations O_k and O_j. Operation O_ℓ is also directly related to other operations that use O_ℓ^i, O_ℓ^e, or O_ℓ^f in their transition conditions.

Every operation is represented by this three-state model and is described as self-contained, since the relations among operations are stored in each operation. These relations are extracted from the operations when the sequences are visualized in the SOP language. This gives a flexible modeling language for complex systems, a language in which operations can be grouped into various sequences.

C. Sequences of Operations, SOPs

Each operation will start in its initial location and wait for its precondition to be fulfilled. If there are no preconditions, all operations will execute unrelated to each other. In practice, a number of relations between the operations restrict the behavior. For example, operation preconditions can be generated based on specific product assembly requirements [3]. To formally model the execution of the operations, the full synchronous composition operator $\|$, defined for EFA by Sköldstam *et al.* [22] is used. For n operations, $O_1, O_2, \ldots, O_n$, the composite operation model is $O_1 \,\|\, O_2 \,\|\, \ldots \,\|\, O_n$. A SOP is defined as follows.

Definition 3 (SOP): SOP SO_k is defined by the composite model $SO_k = O_{k1} \,\|\, O_{k2} \,\|\, \ldots \,\|\, O_{kn}$, all restrictions being represented by pre- and postconditions included in the individual operation models. The operations involved in SO_k are included in the set $\mathcal{O}_k = \{O_{k1}, \ldots, O_{kn}\}$. □

An operation O_{ki} can be included in other SOPs as well, and is not dependent on any graphical model. Thus, it can also be grouped and visualized from various perspectives. A SOP can be represented graphically by a set of sequences, $\mathcal{S}_k = \{S_{k1}, \ldots, S_{kn}\}$, a sequence being defined as follows.

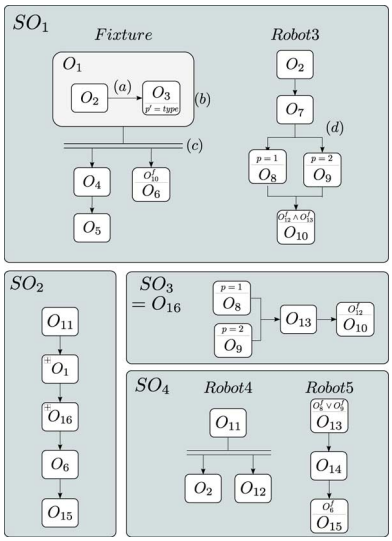

Fig. 3. Four SOPs representing the operations of the manufacturing cell shown in Fig. 4. Examples of SOP language notations: (a) sequence, (b) hierarchy, (c) parallel, and (d) alternative.

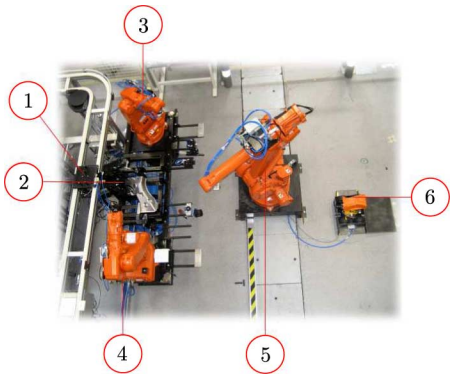

Fig. 4. A manufacturing system with six resources: a conveyor (1), a fixture (2), three robots (3,4,5), and an automated guided vehicle (6).

Definition 4 (Sequence): Sequence S_{ki} is a graph that connects a set of operations or groups of operations related to each other. The operation conditions in the sequence are represented by arrows, lines, and Boolean expressions. □

Observe that here the sequence represents not only strictly sequential operation relations, but can also include groups of operations. The SOP language can group operations into parallel, alternative, arbitrary order, and hierarchical groups.

1) Example 1 (The SOP Language): Fig. 3 shows the four SOPs, SO_1, SO_2, SO_3, and SO_4; these present operations $\mathcal{O} = \{O_1, \ldots, O_{16}\}$ (see Table II), which are executed by the manufacturing cell shown in Fig. 4. The cell assembles a product with two parts and consists of a conveyor (1), a fixture (2), three robots (3–5), and an automated guided vehicle (6). Henceforth, the activities of the fixture and the work of the three robots will be examined.

The first SOP, $SO_1 = \{O_1 || O_2 || \ldots || O_{10}\}$, includes two sequences. The left sequence describes operations realized by the fixture, and the right sequence operations performed by Robot 3. Operation O_1 fixates the product by means of two suboperations, O_2 and O_3, denoted by box (b). O_2 places the product in the fixture; as it is realized by both the robot and the fixture, it is included in both the left and the right sequences.

TABLE II
THE OPERATIONS OF $\mathcal{O}$

O_1	Fixate the product	O_9	Drill type 2
O_2	Place part B in fixture	O_{10}	Robot3 riveting
O_3	Close clamps	O_{11}	Place part A in fixture
O_4	Production logging	O_{12}	Part position measuring
O_5	Store log	O_{13}	Robot5 riveting
O_6	Release product	O_{14}	Robot5 tool change, gripper
O_7	Robot3 tool change, drill	O_{15}	Pick up part from fixture
O_8	Drill type 1	O_{16}	Drilling and riveting

Operation O_3 closes the clamps of the fixture, which can only happen after O_2 has finished, i.e., a sequential relations indicated by the arrow (a). O_3 has an extra postcondition, $p' = type$, which is an action that assigns variable p the value of the sensor $type$ when $O_3^{\downarrow}$ occurs. This sensor checks what type of product is located in the fixture and decides whether to execute operation O_8 or O_9 later on.

The parallel lines at (c) represent a parallel relation, meaning that both O_4 and O_6 are subject to O_1^f as a precondition. Since O_6 has the extra precondition O_{10}^f, the complete precondition for O_6 is $C_6^{\uparrow} = O_1^f \wedge O_{10}^f$. Operation O_6 will release the product from the fixture, which can be executed in parallel with O_4 and O_5. Operations O_4 and O_5 write production information to a logging system.

The operations realized by Robot 3 are shown in the right sequence of SO_1 in Fig. 3 starting with O_2, which places part B in the fixture. After that, O_7 changes the robot's tool from a gripper to a drill. Based on the type of product located in the fixture, either drilling operation O_8 or O_9 is executed. This alternative branch is shown at (d) and is modeled by adding an alternative booking variable to each operation. Operation O_{10} rivets the last holes shared by the two product types, but after O_{12} and O_{13}, which are realized by other robots shown in SO_4. The final precondition for O_{10} is $C_{10}^{\uparrow} = O_{12}^f \wedge O_{13}^f \wedge (O_8^f \vee O_9^f)$. □

As can be seen in Fig. 3, it is not obvious when graphic connections or Boolean expressions should be used to describe the complex operation conditions. For example, if every operation in a normal industrial manufacturing cell is included in a single sequence, it is usually very complicated to understand the operation relations. Even creating that sequence can sometimes be almost impossible. A challenge when representing relations among operations is therefore to decide which operations to include in a given sequence.

D. Operation Relations in Multiple SOPs

SOP SO_1 in Fig. 3 includes operations $O_1, \ldots, O_{10}$. These can be graphically represented in many ways; in the figure they are structured in two sequences, one for each resource. An operation can have multiple attributes defining its properties, and these attributes determine how the operation is structured. An attribute can be related to product information, executing resources, safety constraints, etc. These attributes are used to extract operations into SOPs and to graphically structure them in different sequences and SOPs.

1) Example 2 (Structure Operations): Let us go back to the cell in Fig. 4 and consider operations O_{11} to O_{16}, which according to SO_4 in Fig. 3 are related to Robots 4 and 5. The

product sequence is represented by SO_2, which includes only operations directly involved in assembling the product (see [3]). The product consists of two parts that are placed and fixated in the fixture by O_{11} followed by O_1, processed by O_{16}, released by O_6, and finally moved away by O_{15}. This sequence is based on the product requirements and design and can be described in early design phases.

The operations in SO_2 are assigned to different resources during the layout design of the cell. The process operation is complicated since it involves two robots. Therefore, the suboperations of O_{16} are shown in a separate SOP, SO_3. The SOP language allows these suboperations to be included in SO_2, but in this case are not interesting to the designer of SO_2.

Programming the various control systems calls for more detailed operations that describe every task. However, this is complicated in the case of operations structured as in SO_2. Instead a structure based on their realizing resources, shown in SO_1 and SO_4, could be used, in which the operations are structured in sequences based on each resource. The relations between the resources are shown using Boolean expressions, for example, indicating that operation O_{10} must wait for the completion of O_{12} and O_{13}. These added Boolean expressions convey a clear understanding of the operation sequence for each resource. Most of the operations shown in Table II also include suboperations, for example, O_{13}, which adds multiple rivets, but is not defined at this stage. □

It is very complicated to understand and manually organize all operations in a real development project since hundreds of operations may be involved. One aid could be to automatically create a set of sequences that represents a SOP. For example, if the control designer is interested in the interaction between the robots and the fixture, involving O_2, O_3, O_{10}, O_{12}, and O_{15}, the SOP

$$SO_5 = O_2||O_3||O_{10}||O_{12}||O_{15} \tag{1}$$

would be interesting to visualize graphically. This is done by first identifying the relations among the operations, as discussed in the next section, and then visualizing the identified relations in sequences, as presented in Section IV.

III. Relation Identification

The *direct relations* between operations stem from various design requirements and decisions, such as product assembly or manufacturing safety constraints. However, most operations will not be directly related to each other, even though they are related in some way. For example, if we look at operations O_{13}, O_{14}, and O_{15} from Fig. 3, with preconditions $C^{\uparrow}_{14} = O^f_{13}$ and $C^{\uparrow}_{15} = O^f_{14} \wedge O^f_6$, i.e., a sequence, then O_{13} and O_{15} are not directly related but O_{13} will always precede O_{15}. They are thus *indirectly related*.

To use multiple SOPs, it is crucial to identify these indirect relations among operations. Therefore, let us examine the types of relations that exist between operations and how to identify them.

A. Operation Relation Types

To analyze and reason about the relations between operations, the possible locations of an operation, related to when an event is enabled, will be studied. If we have a set of operations $\mathcal{O} = \{O_1, \ldots, O_n\}$ that defines the complete behavior of a system, the composition model $H = O_1|| \ldots ||O_n$ defines the relations among them. The finite set of locations of H is denoted Q_H.

Operation O_k will be located in one of its three locations, $q_k \in Q_k$, in every composition location $q_H \in Q_H \subseteq Q_1 \times \cdots \times Q_n$, according to the synchronization operator. The composition location q_H therefore includes the current location of each individual operation in H, i.e., $q_H = \langle q_1, \ldots, q_n \rangle$. The events that are enabled in location q_H are denoted as $\Gamma^{Q_H}(q_H)$, and two location sets related to an event σ can be defined as follows:

- $Q^{\sigma}_H : \{q_H \in Q_H | \sigma \in \Gamma^{Q_H}(q_H)\}$—the locations in Q_H, where σ is enabled;
- $O^{\sigma}_k : \{q_k \in Q_k | q_H = \langle q_1, \ldots, q_k, \ldots, q_n \rangle \in Q^{\sigma}_H\}$—the locations of operation O_k where σ is enabled.

There are seven possible location combinations of O^{σ}_k, i.e., $\{O^i_k\}$, $\{O^e_k\}$, $\{O^f_k\}$, $\{O^i_k, O^e_k\}$, $\{O^i_k, O^f_k\}$, $\{O^e_k, O^f_k\}$, $\{O^i_k, O^e_k,$ and $O^f_k\}$. For example, if $O^{O^{\uparrow}_{\ell}}_k = \{O^f_k\}$, then operation O_ℓ will only start when operation O_k is in its final location. To define the possible relations between operations O_k and O_ℓ, all four location sets, i.e., $O^{O^{\uparrow}_{\ell}}_k$, $O^{O^{\downarrow}_{\ell}}_k$, $O^{O^{\uparrow}_{k}}_\ell$, and $O^{O^{\downarrow}_{k}}_\ell$, must be identified and compared. The possible combinations of these state sets can be grouped into the following relation types.

Definition 5 (Relations Between O_k and O_ℓ):

- Always in sequence: $O_k \succ O_\ell$

$$O_k \succ O_\ell : O^{O^{\uparrow}_{\ell}}_k = \{O^f_k\} \wedge O^{O^{\downarrow}_{\ell}}_k = \{O^f_k\} \wedge O^{O^{\uparrow}_{k}}_\ell = \{O^i_\ell\} \wedge O^{O^{\downarrow}_{k}}_\ell = \{O^i_\ell\}.$$

- Sometimes in sequence: $O_k \succsim O_\ell$

$$O_k \succsim O_\ell : O^{O^{\uparrow}_{\ell}}_k = \{O^i_k, O^f_k\} \wedge O^{O^{\downarrow}_{\ell}}_k = \{O^i_k, O^f_k\} \wedge O^{O^{\uparrow}_{k}}_\ell = \{O^i_\ell\} \wedge O^{O^{\downarrow}_{k}}_\ell = \{O^i_\ell\}.$$

- Parallel: $O_k||O_\ell$

$$O_k||O_\ell : O^{O^{\uparrow}_{\ell}}_k = \{O^i_k, O^e_k, O^f_k\} \wedge O^{O^{\downarrow}_{\ell}}_k = \{O^i_k, O^e_k, O^f_k\} \wedge O^{O^{\uparrow}_{k}}_\ell = \{O^i_\ell, O^e_\ell, O^f_\ell\} \wedge O^{O^{\downarrow}_{k}}_\ell = \{O^i_\ell, O^e_\ell, O^f_\ell\}.$$

- Alternative: $O_k + O_\ell$

$$O_k + O_\ell : O^{O^{\uparrow}_{\ell}}_k = \{O^i_k\} \wedge O^{O^{\downarrow}_{\ell}}_k = \{O^i_k\} \wedge O^{O^{\uparrow}_{k}}_\ell = \{O^i_\ell\} \wedge O^{O^{\downarrow}_{k}}_\ell = \{O^i_\ell\}.$$

- Arbitrary order: $O_k \oplus O_\ell$

$$O_k \oplus O_\ell : O^{O^{\uparrow}_{\ell}}_k = \{O^i_k, O^f_k\} \wedge O^{O^{\downarrow}_{\ell}}_k = \{O^i_k, O^f_k\} \wedge O^{O^{\uparrow}_{k}}_\ell = \{O^i_\ell, O^f_\ell\} \wedge O^{O^{\downarrow}_{k}}_\ell = \{O^i_\ell, O^f_\ell\}.$$

- Hierarchy: $O_k \sqsubset O_\ell$

$$O_k \sqsubset O_\ell : O^{O^{\uparrow}_{\ell}}_k = \{O^e_k\} \wedge O^{O^{\downarrow}_{\ell}}_k = \{O^e_k\} \wedge O^{O^{\uparrow}_{k}}_\ell = \{O^i_\ell\} \wedge O^{O^{\downarrow}_{k}}_\ell = \{O^f_\ell\}.$$

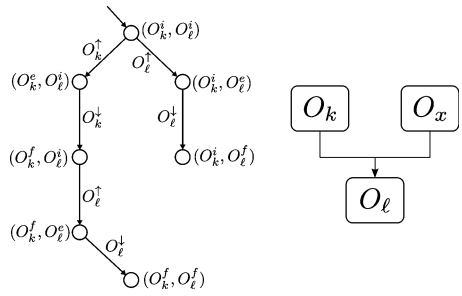

Fig. 5. An automaton and a SOP describing a sometimes in sequence relation: $O_k \succsim O_\ell$.

- Other: $O_k \curlywedge O_\ell$

$$O_k \curlywedge O_\ell : \text{All other combinations.}$$

Always in sequence, $O_k \succ O_\ell$, indicates that O_k always precedes O_ℓ, e., the start event $O_\ell^\uparrow$ is only enabled when O_k is in its final location. This is the normal sequential relation in which two operations are executed one after the other.

Sometimes in sequence, $O_k \succsim O_\ell$, indicates that O_k sometimes precedes O_ℓ. This relation is almost the same as always in sequence, in which O_k will only start when O_ℓ is in its initial location, but O_ℓ can start when O_k is in either its final or initial location. Consider Fig. 5, which shows an automaton and a SOP with a sometimes in sequence relation between O_k and O_ℓ, and $O_\ell^\uparrow = O_k^f \vee O_x^f$. If O_k is executed, then $O_k^\uparrow$ is always enabled when O_ℓ is in its initial location O_ℓ^i, and $O_\ell^\uparrow$ is enabled when O_k is located in O_k^f. However, when O_x is executed instead of O_k, $O_\ell^\uparrow$ is also enabled when O_k is in O_k^i, i.e., $O_k^{O_\ell^\uparrow} = \{O_k^i, O_k^f\}$.

Parallel, $O_k \| O_\ell$, indicates that O_k and O_ℓ can execute in parallel, i.e., any combination of events exists in the complete behavior of a system.

Alternative, $O_k + O_\ell$, indicates that if O_k or O_ℓ is not located in the initial location, the other one cannot start. This relation is usually due to an alternative branch, modeled by the booking of a mutually exclusive resource.

Arbitrary order, $O_k \oplus O_\ell$, indicates that O_k and O_ℓ will not execute at the same time, but does not specify the order in which they execute. This normally happens when two operations require the same resource, which hinders them from executing simultaneously. This means that once an operation has started, it must complete before the other operation can start, but the order in which the operations are executed is not specified.

Hierarchy, $O_k \sqsubset O_\ell$, indicates that O_k is a parent of O_ℓ. This relation implies that O_ℓ starts and completes when O_k is located in its executing location.

Other, $O_k \curlywedge O_\ell$, indicates that O_k is related to O_ℓ in some way, i.e., all other possible location combinations. Obviously, these relations are interesting, but are not currently handled by the visualization algorithms introduced in Section IV. New operation relations will be defined and managed in future research.

Observe that all four location sets, i.e., $O_k^{O_\ell^\uparrow}$, $O_k^{O_\ell^\downarrow}$, $O_\ell^{O_k^\uparrow}$, and $O_\ell^{O_k^\downarrow}$, must be considered when defining the relations between two operations. Consider, for example, the definition of arbitrary order in Definition 5. Operation O_ℓ can start when O_k is in either its initial or final location and stop when O_k is in either its initial or final location. However, O_ℓ cannot start in O_k^i and stop in O_k^f, since O_k does not start or stop in the execution location of O_ℓ, according to $O_\ell^{O_k^\uparrow} = \{O_\ell^i, O_\ell^f\}$ $O_\ell^{O_k^\downarrow} = \{O_\ell^i, O_\ell^f\}$. Hence, when one operation starts, it must be completed before the other operation starts.

B. Identifying Relations

To identify the relations among a set of operations $\mathcal{O}_{k\ell} = \{O_k, \ldots, O_\ell\}$, where $\mathcal{O}_{k\ell} \subseteq \mathcal{O} = \{O_1, \ldots, O_n\}$ and $\mathcal{O}$ includes every operation of the system, Algorithm 1 is used.

Algorithm 1: Identify operation relations

Input: $\mathcal{O}_{k\ell}$, $\mathcal{O}$
Result: RelationTable
$H = Sync(\mathcal{O})$
foreach O in $\mathcal{O}_{k\ell}$**do**
  $Q_H^{O^\uparrow} = \text{FindEnabled}(O^\uparrow, H)$;
  $Q_H^{O^\downarrow} = \text{FindEnabled}(O^\downarrow, H)$;
end
$O^\sigma = \text{ExtractOperations}(Q_H^\sigma)$;
foreach O_s in O^σ**do**
  foreach O_t in O^σ**do**
    RelationTable $[O_s, O_t] = \text{MatchRel}(O_s, O_t)$;
  end
end

Algorithm 1 takes $\mathcal{O}_{k\ell}$ and $\mathcal{O}$ as input and returns a table describing the pairwise relation between the operations. The first step of the algorithm is to synchronize the operation models with the full synchronous composition operator and store the result in H. Then, the location set Q_H^σ is calculated including the elements $Q_H^{O_s^\uparrow}$ and $Q_H^{O_s^\downarrow}$, where $s = k \ldots \ell$, by the first FOR loop. After that, the operations are extracted into O^σ and iterated through and compared pairwise according to Definition 5, i.e., location sets $O_s^{O_t^\uparrow}$, $O_s^{O_t^\downarrow}$, $O_t^{O_s^\uparrow}$, and $O_t^{O_s^\downarrow}$, where $s = k \ldots \ell$, $t = k \ldots \ell$, and $s < t$, are created and the relations are matched.

In the following example, the relations among the operations in SOP SO_5 in (1)will be identified.

1) Example 3 (Relation Identification): Consider SOP SO_5 with the operations $\mathcal{O}_5 = \{O_2, O_3, O_{10}, O_{12}, O_{15}\}$. First, the complete system $H = O_1 \| \ldots \| O_{16}$ is calculated, the location names in H being built of the location of every operation, for example, the initial location is named $\langle O_1^i, \ldots, O_{16}^i \rangle$.

After that, the location sets $Q_H^{O_s^\uparrow}$ and $Q_H^{O_s^\downarrow}$, where $s = 2, 3, 10, 12, 15$, are calculated, i.e., the locations in which the start and stop events of the operations in $\mathcal{O}_5$ are enabled, in the complete system H. This is actually done simultaneously with the synchronization to avoid iterating the locations once again. Two of the locations sets are

$$Q_H^{O_2^\uparrow} = \langle O_1^e, O_2^i, O_3^i, \ldots, O_{16}^i \rangle, \langle O_1^e, O_2^i, O_3^i, \ldots, O_{16}^i \rangle, \ldots$$
$$Q_H^{O_{12}^\uparrow} = \langle O_1^i, O_2^i, O_3^i, \ldots, O_{16}^i \rangle, \langle O_1^e, O_2^e, O_3^f, \ldots, O_{16}^i \rangle, \ldots.$$

Since we are interested in only the relations among the operations in $\mathcal{O}_5$, the above sets are reorganized based on when the event is enabled relative to the locations of the operations in $\mathcal{O}_5$.

The location sets $O_s^{O_t^{\uparrow}}$ and $O_s^{O_t^{\downarrow}}$ are the result. Below, only the sets for the start event of each operation are shown

$$
\begin{aligned}
O_2^{\uparrow} &: \{O_2^i\}, \{O_3^i\}, \{O_{10}^i\}, \{O_{12}^i, O_{12}^e, O_{12}^f\}, \{O_{15}^i\} \\
O_3^{\uparrow} &: \{O_2^f\}, \{O_3^i\}, \{O_{10}^i\}, \{O_{12}^i, O_{12}^e, O_{12}^f\}, \{O_{15}^i\} \\
O_{10}^{\uparrow} &: \{O_2^f\}, \{O_3^f\}, \{O_{10}^i\}, \{O_{12}^f\}, \{O_{15}^i\} \\
O_{12}^{\uparrow} &: \{O_2^i, O_2^e, O_2^f\}, \{O_3^i, O_3^e, O_3^f\}, \{O_{10}^i\}, \{O_{12}^i\}, \{O_{15}^i\} \\
O_{15}^{\uparrow} &: \{O_2^f\}, \{O_3^f\}, \{O_{10}^f\}, \{O_{12}^f\}, \{O_{15}^i\}.
\end{aligned}
$$

Based on this information, it is possible to identify when an event is enabled relative to the locations of the other operations. These relation patterns can now be used to match the operations pairwise against Definition 5, and the relations between all pairs of operations are defined. The result is shown below

$$
\begin{aligned}
O_2 \succ O_3 \quad & O_2 \succ O_{10} \; O_2 \succ O_{15} \; O_2 \| O_{12} \\
O_3 \succ O_{10} \quad & O_3 \succ O_{15} \; O_3 \| O_{12} \\
O_{12} \succ O_{10} \quad & O_{12} \succ O_{15} \\
O_{10} \succ O_{15}&.
\end{aligned}
$$

These relations are based on both direct and indirect relations. □

C. Computation

When the relations within only a small subset of operations in a system are identified, all the operations in the system must still be synchronized in any case. Since these computations usually involve large state set analysis, state-space explosion can be an issue in real systems. This can be handled using binary decision diagrams (BDDs) [1], as in Vahidi [25] and Miremadi *et al.* [15]. Recent results [20] indicate that a controller can be synchronized and synthesized based on EFA operation models and on BDDs for systems with 10^{11} reachable states using the software tool Supremica [24]. The implementation details are omitted here due to space constraints.

When all operations have been synchronized, pairwise identification will be done only on the subset of operations to be visualized. The maximum number of comparisons is $((n-1)^2 + (n-1))/2$, where n is the number of operations to be visualized. The proof-of-concept implementation is promising for real industrial systems, but further investigation is needed, especially concerning how to recompute the relations when only some of the operations have been changed.

IV. Relation Visualization

The relations among operations in a SOP can be visualized in many ways, since the relations can be presented both graphically and using Boolean expressions. For example, based on operation attributes such as resource names and product parts, it is possible to visualize the operations in different sequences, such as in SO_1 and SO_4 in Fig. 3. However, it is also possible to create only one sequence including all the operations in the SOP.

The algorithm presented in this section takes a set of operations and creates a sequence including all operations in the given set. If the operations in the SOP are to be organized based on an operation attribute, the operations must be structured according to the attributes before the algorithm can create the sequences.

A. Grouping

The visualization algorithm is divided into four stages: relation identification, grouping, sequencing, and drawing. The relations within a set of operations are first identified as in Section III. After that, the operations are grouped based on hierarchy, alternative, arbitrary order, or parallel relations, called *related relations*. After the operations have been grouped, they are sorted based on whether they are *predecessors* or *successors* (always and sometimes in sequence). Finally, the sequence graph is drawn. The *other* relations are not handled by the algorithm (see the end of this section).

To sort the operations into groups and sequences, each operation and group becomes a node in a data structure. Group and operation nodes are almost the same, except that a group can contain other nodes and the relation between the group and other nodes is based on all operations included in the group (and its subgroups). For example, if a group precedes an operation, this means that all nodes in the group precede the operation.

An operation node for each operation is created and added to the list $Nodes$. An operation node always represents an operation and is therefore sometimes also referred to as an operation. The list $Nodes$ is then used as input to Algorithm 2.

Algorithm 2: Grouping of operations

```
Input Nodes
Result //Create Hierachy Groups groupList := new Node List;
foreach n in Nodes do
   If n is parent then
      gH := New Hierarchy group;
      foreach k in Nodes do
         If k child of n then
            gH.add(k);
         end
      end
      groupList.add(gH);
   end
end
Nodes := ResolveGroups(groupList, Nodes);
//Create Alternative Groups
         foreach n in Nodes do
| {...}
end
Nodes := ResolveGroups(groupList, Nodes);
//Create Arbitrary Groups
foreach n in Nodes do '...
end
Nodes := ResolveGroups(groupList, Nodes);
//Create Parallel Groups
foreach n in Nodes do '...
end
Nodes := ResolveGroups(groupList, Nodes);
```

The grouping Algorithm 2 creates hierarchy, alternative, arbitrary order, and parallel groups. First, the hierarchies are iden-

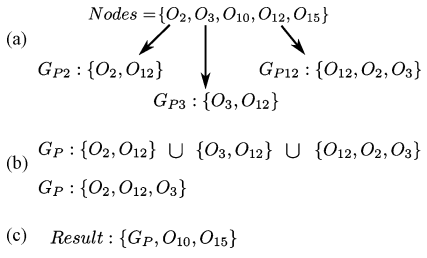

Fig. 6. Example of the grouping algorithm.

tified and created, which checks each node in $Nodes$ to see whether it is a parent (i.e., whether the operation has suboperations). When a parent is found, a new hierarchy group is created and the child nodes from $Nodes$ are added. The command *k child of n* checks the previously identified operation relation, to determine whether k is a child of n. Some of the created groups can contain the same nodes, since a parent can also be a parent to a child of a child. The groups are therefore resolved and merged if they are referencing the same nodes.

ResolveGroups is a recursive method that merges created groups if they share nodes. The set $\mathcal{G} = \{G_1, \ldots, G_n\}$ represents the groups in $groupList$, where G_i is a set of the nodes in a group. If $G_i \cap G_j \neq \emptyset$, $i \neq j$, a new group is created including the nodes of both groups, $G_{new} = G_i \cup G_j$. New intersections with G_{new} and the remaining groups in $\mathcal{G}$ are resolved recursively. After that, new groups are created in the same way based on the groups not included in G_{new}.

The result of $ResolveGroups$ is a new $Nodes$ list containing the created hierarchy groups and remaining nodes not included in a group. This list is then used when alternative groups are created and resolved. The method for creating alternative groups is the same, except that alternatives are also identified inside each hierarchy group. An alternative group includes nodes that share at least one alternative relation. Observe that not all nodes in the group must have an alternative relation with every other node in the group. Arbitrary order and parallel groups are created recursively in the same way as are the alternative groups.

1) Example 4 (Grouping): Consider the operations from the relation identification example. A $Nodes$ list is created including these operations, shown in Fig. 6(a). There are no hierarchy, alternative, or arbitrary order relations, but rather there are parallel relations. First, each operation is studied; when it has parallel relations with other operations, a parallel group G_{Pi} is created. In Fig. 6, three groups are created: G_{P2}, G_{P3}, and G_{P12}. Operations O_{10} and O_{15} do not have any parallel relations.

As can be seen in Fig. 6, the three groups are related to each other since they share nodes. First, $G_{P2} \cap G_{P3} = \{O_{12}\}$ is calculated, i.e., they share node O_{12}. The new group $G_P = G_{P2} \cup G_{P3}$ is created, including nodes O_2, O_3, and O_{12} (b). The new G_P is tested with the remaining group, $G_P \cap G_{12} = \{O_2, O_3, O_{12}\}$, which results in $G_P = G_P \cup G_{12}$. All groups are now resolved and included in G_P.

The Resolve method adds the new group to $Nodes$ and removes the nodes that were included in the group from $Nodes$. The result is $\{G_P, O_{10}, O_{15}\}$, (c). □

When all groups have been identified and created, the sequential relations need to be identified, which is handled by the sequencing algorithm.

B. Sequencing

To structure the elements in $Nodes$ sequentially, each node n is given a set of attributes. Two attributes are references to the closest preceding node, $n.Pred$, and the successor node, $n.Succ$; these are used to organize the nodes into a tree structure. Three other attributes are the lists $n.PredList$, $n.SuccList$, and $n.RelList$ used to sort a set of nodes related to n. The list from the grouping algorithm is the input to the recursive sequencing Algorithm 3.

Algorithm 3: Sequencing nodes

```
Seq(Nodes) return root Node
If Nodes.Empty then return null;
root := ExtractNode(Nodes);
If root is group then return Seq(root.nodes);
foreach n in Nodes do
    If n is Pred to root then root.PredList.add(n);
    else If n is Succ to root then root.SuccList.add(n);
    else root.RelList.add(n);
end
root.Pred := GetLast(Seq(root.PredList));
root.Succ := Seq(root.SuccList);
Seq(root.RelList);
return Structure(root);
```

The recursive Seq method takes a list of nodes and returns the root node of a tree. The first line terminates the recursive algorithm. The next line extracts one of the nodes from the set $Nodes$ that is used as $root$ by the algorithm. If $root$ is a group, the nodes inside it are also sorted by the Seq method.

Each remaining node in $Nodes$ is sorted according to whether it is a predecessor of, successor to or only related to $root$ and added to $root.PredList$, $root.SuccList$ or $root.RelList$.

The nodes in the lists of root are then sorted by Seq. This recursive method terminates when a leaf is reached, meaning that a list is empty. The returning node from the sorting of $PredList$ and $SuccList$ is first in a subsequence, but the predecessor to root is the last node in the subsequence the GetLast method finds it and assigned it to $root.Pred$. The returning node is assigned to $root.Succ$, hence, a tree is constructed with $root$ as the first node.

The first node in the sequence can be found when the $root \cdot Pred$ branch is followed. To better represent the real sequence, the tree is restructured based on the sequence order, i.e., only the $.Succ$ branch exists. For example, if A is root and $A \cdot Pred = B$ and $A \cdot Succ = C$, then B replaces A as root and $B \cdot Succ = A$ and $A \cdot Succ = C (A \cdot Pred = Null)$, i.e., the tree is restructured with the first node in the sequence at the top.

1) Example 5 (Sequencing): In Fig. 7, the list from grouping example (a) is input to the sequencing algorithm. O_{10} is picked as root and the other nodes are sorted into $O_{10}.PredList$ and $O_{10}.SuccList$, represented by the p and s arrows in (b). Since

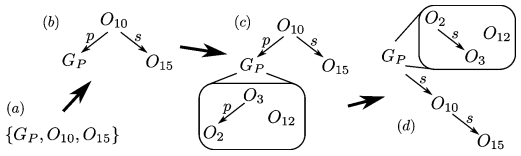

Fig. 7. Example of the sequencing algorithm.

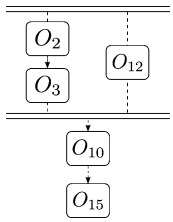

Fig. 8. SOP SO_5 and its sequence.

only O_{15} is in $O_{10}.SuccList$, that branch is resolved, i.e., $O_{10}.Succ = O_{15}$. The node in $O_{10}.PredList$ is a group, and its included nodes are sorted in the same way: O_3 is picked as root, O_2 precedes O_3 and is added to $O_3.PredList$, and O_{12} has a parallel relation with O_3 and is therefore not sorted. Finally, in (d), the nodes are restructured based on the sequence order. G_P is returned as root node. □

Once the node tree has been created, it is straightforward to generate the resulting sequence graph.

C. Drawing

When drawing the sequence, each node in the tree is drawn, one at a time: first the root node is drawn, after that the successor node, which is connected to the root node, and then the node after that until the last node has been drawn. If the node is a group, the nodes in the group are drawn inside the group type notation (cf. the parallel group G_P in Fig. 8, where the parallel lines are drawn above and below the included operations).

The lines between the operations and groups that define the sequential relations are drawn differently if the relation is direct, i.e., is defined in the operation condition, or indirect. If the relation is indirect, a dashed line is used instead.

1) Example 6 (Drawing): SOP SO_5, based on the sorting and grouping example above, is shown in Fig. 8, including one sequence. All of the relations are indirect except between O_2 and O_3, since $C_3^{\uparrow} = O_2^f$. □

One limitation of the presented algorithm is that operations that have an *other* relation with at least one operation in the sequence cannot be included. These operations are placed next to the sequence showing their conditions as Boolean expressions. Another limitation is that no loops can be included, since they are not explicitly identified in the relation identification. These limitations are an area for future research.

D. Visualizing a SOP

Methods for identifying and visualizing operations in one sequence have now been presented. However, one of the main features of the SOP language is that it lets one present the SOP operations in multiple sequences. It is usually complicated or even impossible to represent all operation relations in only one sequence. In Fig. 9, a SOP with four sequences is visualized, the relations between the sequences being presented using Boolean expressions.

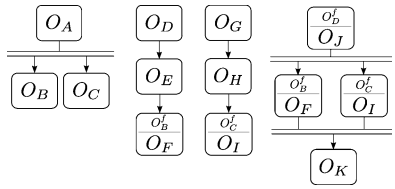

Fig. 9. A SOP with four sequences.

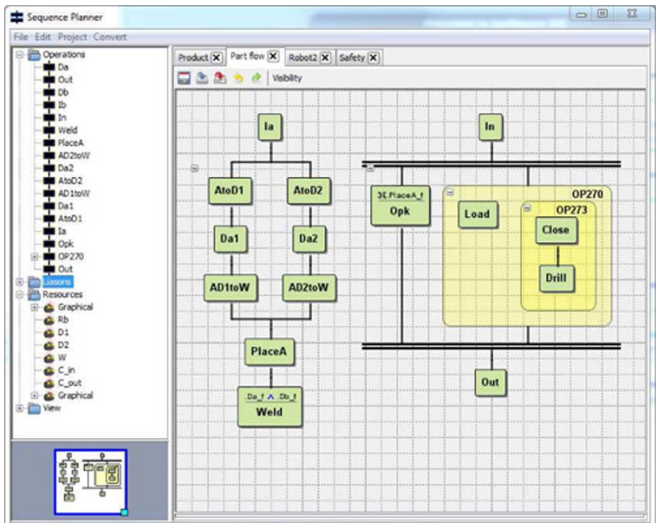

Fig. 10. Sequence planner.

When visualizing multiple sequences, the relations among the SOP operations are first identified. After that, each sequence is sorted and visualized individually. When the sequences are drawn, all operation relations that are not represented by graphical notations in any of the sequences are then described by Boolean pre- and postconditions. For example, the precondition of operation O_F in Fig. 9 is $C_F^{\uparrow} = O_E^f \wedge O_J^f \wedge O_B^f$, where O_E^f and O_J^f are represented graphically in the second and fourth sequences and O_B^f is described by the Boolean expression.

The next section presents a software tool called SP [16] that visualizes and manages operations and their relations, and coordinates changes and updates among the operations and the SOPs.

V. Sequence Planner (SP)

Sequence Planner (SP) is a prototype software tool developed to manage the SOP language and all involved operations. A screen shot is shown in Fig. 10. SP has a central repository storing all operations used in a project, shown in the upper left in the figure. In addition, a resource structure and a product structure are available. The operations are stored separately from the SOPs and the graphical representation.

SP allows the creation of any number of SOPs, one of which is shown in Fig. 10. A SOP can be created, for example, by clicking on a resource, showing all operations realized by that resource, or manually by dragging operations into the SOP window. In a SOP, new conditions on the operations can be added by either creating sequences or adding Boolean

expressions. Changes related to the operations and SOPs can also come from external applications such as PLM, simulation, verification, optimization, or synthesis tools, or can be manually changed in SP. Operation conditions may change, new operations be added, or product and/or resource data be updated. One challenge in SP is therefore to coordinate changes among the SOPs and the operations.

A. Updating Operations in Various SOPs

Every self-contained operation is stored in SP in set $\mathcal{O}$, which is unrelated to the SOPs. $\mathcal{O}$ is the main operation set for the whole model and is the interface with external applications, not only receiving changes but also sending out operation information. An operation is not fully included in the project until it has been included in $\mathcal{O}$.

It is possible to create new operations, or extract operations from $\mathcal{O}$, when creating a new SOP. However, when updating and managing changes between the SOPs and $\mathcal{O}$, it is not feasible for each SOP to directly change the operations in $\mathcal{O}$. Therefore, the SOP uses copies of the operations, and when changes are made in one SOP, they are saved to $\mathcal{O}$ and coordinated with the other SOPs.

Since the same operation can exist in multiple copies, an operation object is used to keep them apart, defined as follows.

Definition 6 (Operation Object): Operation object o_s is a data object representing operation O_k. Object o_s has two important attributes: the id of the operation, $o_s.id = k$, and the operation information-like conditions and realizing resources, stored in $o_s.value$. □

If two operation objects o_s and o_t, where $o_s \neq o_t$, have the same id, $o_s.id = o_t.id = k$, they represent the same operation, i.e., O_k. The different operation objects, however, can end up with differing values, which must be managed by SP.

Each operation in the SOPs and in $\mathcal{O}$ is represented by an operation object. For simplicity, in the rest of this section, $\mathcal{O}$ represents all operation objects and $\mathcal{O}_k$ represents operation objects for SOP SO_k.

When an operation object is changed in a SOP or in $\mathcal{O}$, other operation objects representing the same operation have differing values and need to be updated. To identify differing operation objects in set $\mathcal{O}_s$ compared with set $\mathcal{O}_t$, the set $\Delta(\mathcal{O}_s, \mathcal{O}_t)$ is introduced, which is defined as follows.

Definition 7 (Differing Operation Objects): The differing operation object set, $\Delta(\mathcal{O}_s, \mathcal{O}_t)$, is defined as the set of operation objects $o_s \in \mathcal{O}_s$, such that $\exists o_t \in \mathcal{O}_t | o_s.id = o_t.id \wedge o_s.value \neq o_t.value$. □

It is also interesting to identify operations included only in set $\mathcal{O}_s$ but not in $\mathcal{O}$. The new operation object set is defined as follows.

Definition 8 (New Operation Objects): The new operation object set, $\mathcal{N}(\mathcal{O}_s, \mathcal{O})$, is defined as the set of operation objects $o_s \in \mathcal{O}_t$, such that $\forall o_t \in \mathcal{O} | o_s.id \neq o_t.id$, $s \neq t$. □

When an operation object is changed in a SOP, it will first be saved to $\mathcal{O}$. After that, the other SOPs are notified of the change and will update the changed operation and related sequences. No new changes can be added to $\mathcal{O}$ in this period. This can be formulated in the following algorithmic steps.

1) When the changes made to SO_s are saved, $\Delta(\mathcal{O}_s, \mathcal{O})$ and $\mathcal{N}(\mathcal{O}_s, \mathcal{O})$ are calculated. The differing operation objects in $\Delta(\mathcal{O}_s, \mathcal{O})$ replace the old versions in $\mathcal{O}$. The new operation objects, $\mathcal{N}(\mathcal{O}_s, \mathcal{O})$, are added to $\mathcal{O}$.
2) Every other view SO_t where $\Delta(\mathcal{O}_t, \mathcal{O}_s) \neq \emptyset$ creates an updated version of $\mathcal{O}_t$.

When new operation relations requirements are introduced in one SOP, the impact of these changes could be better understood in other views. For example, if a product SOP is created, the work of building, for example, safety or resource requirements might result in changes to the sequences in the product SOP, leading to a need to change how the product is built. Completely understanding complex operation information and how changes affect system behavior calls for viewing the operations using multiple and coordinated SOPs.

VI. Conclusion

The relations among operations tend to be highly complex when developing an automation system. These relations are constantly changing throughout the development process, which makes it difficult to keep them up to date. Therefore, it would be more natural to consider sequence planning as an iterative process, in which sequences are viewed based on current requirements, rather than as a manual construction process. Using a graphic, formal language called Sequences of Operations (SOPs), this paper presents algorithms and methods that visualize the operation relations from multiple perspectives.

A set of sequences is used to structure the operations, for example, based on the resources used to realize them or based on how different parts are manufactured in the system. The structuring can be automated by using different user-assigned operation attributes, such as what resource was used to realize the operation. When the operations are structured, the relations among them are identified. This is accomplished by modeling the operations using EFA, which are synchronized in order to identify when an operation can execute relative to other operations. Based on the identified operation relations, a set of sequences can be created with a set of algorithms that visualizes the relations in the SOP language.

The proposed algorithms are implemented in a software tool called Sequence Planner, which uses the tool Supremica for automata calculations. Initial results and case studies indicate that the proposed approach is promising from both the computational and usability points of view. Further development is needed, focusing especially on how to recompute the relations when only some of the operations have changed.

An industrial case study is currently being conducted in which the approach is being evaluated during the design and installation of a complete manufacturing system. In addition, how to adapt Sequence Planner to other virtual process planning tools is being evaluated.

References

[1] S. B. Akers, "Binary decision diagrams," *IEEE Trans. Comput.*, vol. 27, no. 6, pp. 509–516, Jun. 1978.
[2] M. Q. W. Baldonado, A. Woodruff, and A. Kuchinsky, "Guidelines for using multiple views in information visualization," in *Proc. Working Conf. Advanced Visual Interfaces, AVI '00*, 2000, pp. 110–119.

[3] K. Bengtsson, B. Lennartson, and C. Yuan, "The origin of operations: Interactions between the product and the manufacturing automation control system," in *Proc. IFAC Symp. Inform. Control Problems in Manuf., INCOM'09*, 2009. [Online]. Available: http://www.ifac-papersonline.net/Detailed/40587.html
[4] K. Bengtsson, B. Lennartson, C. Yuan, P. Falkman, and S. Biller, "Operation-oriented specification for integrated control logic development," in *Proc. IEEE Conf. Autom. Sci. Eng., CASE*, 2009, pp. 183–190.
[5] K. Bengtsson, C. Thorstensson, B. Lennartson, K. Åkesson, C. Yuan, S. MIremadi, and P. Falkman, "Relations identification and visualization for sequence planning and automation design," in *Proc. IEEE Conf. Autom. Sci. Engineering, CASE*, Toronto, ON, Canada, 2010, pp. 841–848.
[6] A. Hofstede, W. Aalst, M. Adams, and N. Russell, *Modern Business Process Automation, YAWL and its Support Environment*. New York: Springer, 2010.
[7] W. Hsu and B. Liu, "Conceptual design: Issues and challenges," *Comput.-Aided Design*, vol. 32, no. 14, pp. 849–850, 2000.
[8] H. Hu and Z. Li, "Modeling and scheduling for manufacturing grid workflows using timed Petri nets," *Int. J. Adv. Manuf. Technol.*, vol. 42, pp. 553–568, 2009.
[9] H. Kerzner, *Project Management: A Systems Approach to Planning, Scheduling, and Controlling*, 9th ed. New York: Wiley, 2006.
[10] A. Kobetski and M. Fabian, "Time-optimal coordination of flexible manufacturing systems using deterministic finite automata and mixed integer linear programming," *J. Discrete Event Dynamic Syst.*, vol. 19, no. 3, pp. 287–315, 2009.
[11] S. H. Lee, F. Pena-Mora, and M. Park, "Dynamic planning and control methodology for strategic and operational construction project management," *Autom. Construction*, vol. 15, pp. 84–97, 2005.
[12] 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 Trans. Autom. Sci. Autom.*, vol. 7, no. 4, pp. 791–802, Oct. 2010.
[13] H. B. Marri, A. Gunasekaran, and R. J. Grieve, "Computer-aided process planning: A state of art," *Int. J. Adv. Manuf. Technol.*, vol. 14, no. 4, pp. 261–268, 1998.
[14] H. Miao, N. Sridharan, and J. Shah, "Cad-cam integration using machining features," *Int. J. Comput. Integr. Manuf.*, vol. 15, no. 4, pp. 296–318, 2002.
[15] S. Miremadi, K. Åkesson, and B. Lennartson, "Symbolic computation of reduced guards in supervisory control," *IEEE Trans. Autom. Sci. Autom.*, vol. 8, no. 4, pp. 754–765, Oct. 2011. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5764846
[16] E. Ohlson and C. Torstensson, Development, implementation and testing of sequence planner — A concept for modeling of automation systems Dept. Signals and Systems, Chalmers Univ. Technol., Göteborg, Sweden, Tech. Rep. EX/2009, 2009.
[17] J. C. Roberts, "State of the art: Coordinated & multiple views in exploratory visualization," in *Proc. 5th Int. Conf. Coordinated and Multiple Views in Exploratory Visualization*, 2007, pp. 61–71.
[18] A. Shabaka and H. Elmaraghy, "A model for generating optimal process plans in RMS," *Int. J. Comput. Integr. Manuf.*, vol. 21, no. 2, pp. 180–194, 2008.
[19] W. Shen, L. Wang, and Q. Hao, "Agent-based distributed manufacturing process planning and scheduling: A state-of-the-art survey," *IEEE Trans. Syst., Man, Cybern.*, vol. 36, no. 4, pp. 563–577, Jul. 2006.
[20] M. R. Shoaei, B. Lennartson, and S. Miremadi, "Automatic generation of controllers for collision-free flexible manufacturing systems," in *Proc. IEEE Conf. Autom. Sci. Eng., CASE*, Toronto, ON, Canada, 2010, pp. 368–373.
[21] M. Sköldstam, K. Åkesson, and M. Fabian, "Modelling of discrete event systems using finite automata with variables," in *Proc. 46th IEEE Conf. Decision and Control*, New Orleans, LA, Dec. 2007, pp. 3387–3392.
[22] M. Sköldstam, K. Åkesson, and M. Fabian, Supervisory control applied to automata extended with variables—Revised Chalmers, Göteborg, Sweden, Tech. Rep. R001/2008, Signals and Systems, 2008.
[23] R. Spence, *Information Visualization—Design for Interaction, ISBN 0132065509*, 2nd ed. Upper Saddle River, NJ: Pearson Education, 2006.
[24] Supremica. [Online]. Available: http://www.supremica.org
[25] A. Vahidi, "Efficient analysis of discrete event systems," Ph.D. dissertation, Dept. Signals and Systems, Chalmers Univ. Technol., Göteborg, Sweden, 2004.
[26] A. Voronov and K. Åkesson, "Verification of process operations using model checking," in *Proc. IEEE Conf. Autom. Sci. Eng.*, Bangalore, India, Aug. 2009, pp. 415–420.
[27] L. Wall, "Perl, the first postmodern computer language," 1999. [Online]. Available: http://www.perl.com/pub/a/1999/03/pm.html
[28] L. Wang, S. Keshavarzmanesh, H.-Y. Feng, and R. Buchal, "Assembly process planning and its future in collaborative manufacturing: A review," *Int. J. Adv. Manuf. Technol.*, vol. 41, pp. 132–144, 2009, 10.1007/s00170–008-1458–9.
[29] J. M. Wilson, "Gantt charts: A centenary appreciation," *Eur. J. Oper. Res.*, vol. 149, no. 2, pp. 430–437, 2003.

Kristofer Bengtsson received the M.S. degree in automation and mechatronics from Chalmers University of Technology, Gothenburg, Sweden, in 2007. Currently, he is working towards the Ph.D. degree supported by General Motors, Wingquist Laboratory VINN Excellence Centre at Chalmers University of Technology, and CAPE Research School, since 2007.

From 2001 to 2005, he was with Advanced Flow Control AB developing control systems, and from 2005 to 2011 with Teamster AB, an automation firm in Gothenburg. Currently, he is with Sekvensa AB, a research consulting firm. His current research interest includes design methods for control logic development for manufacturing automation.

Patrik Bergagård received the B.Sc. degree in engineering physics and the M.Sc. degree in automation from Chalmers University of Technology, Gothenburg, Sweden, in 2007 and 2009, respectively. He has been working towards the Ph.D. degree supported by the EU 7th FP FLEXA and Chalmers University of Technology, since 2009.

His research interests involve modeling and restart functionality of manufacturing systems.

Carl Thorstensson received the B.Sc. degree in business administration from Gothenburg School of Business and Law, Gothenburg, Sweden, in 2009 and the M.Sc. degree in engineering physics from Chalmers University of Technology, Gothenburg, in 2009.

Since 2010, he is employed by the automation and power technology company ABB, where he currently is working with product management of low voltage products.

Bengt Lennartson was born in 1956 in Gnosjö, Sweden. He received the Ph.D. degree in automatic control from Chalmers University of Technology, Gothenburg, Sweden, in 1986.

Since 1999, he has been a Professor of the Chair of Automation at the Department of Signals and Systems. He was Dean of Education at Chalmers University of Technology from 2004 to 2007, he is a member of the Wingquist Laboratory, a research center for virtual product and production development, and since 2005 he is also Guest Professor at University West, Trollhättan. His main areas of interest include discrete event and hybrid systems, especially for manufacturing applications, as well as robust feedback control. He has been an Associate Editor for*Automatica*, and the Chairman of the Ninth International Workshop on Discrete Event Systems, WODES'08. He is the (co-)author of two books and $\sim$ 180 peer reviewed international papers with $>$ 2200 citations (GS).

Knut Åkesson received the M.S. degree in computer science and engineering from Lund Institute of Technology, Lund, Sweden, in 1997, and the Ph.D. degree in control engineering from Chalmers University of Technology, Gothenburg, Sweden, in 2002.

Currently, he is an Associate Professor at the Department of Signals and Systems, Chalmers University of Technology, where his main research interest is to develop and applying formal methods for verification and synthesis of control logic.

Chengyin Yuan received the B.S. and M.S. degrees in mechanical engineering from Tsinghua University, Beijing, China, and the Ph.D. degree in manufacturing automation controls from the University of Illinois at Urbana–Champaign, Urbana, in 1997, 1999, and 2004, respectively.

From 2005 and 2010, he was with the General Motors Research and Development Center, Warren, MI, where he was a Senior Researcher in the Manufacturing Systems Research Laboratory. Since 2010, he has been with Pride Power System Technology Co., Ltd., Beijing, China. His current research interests include virtual manufacturing, math-based automation system design and validation, distributed systems and simulation, and reconfigurable automation controls.

Sajed Miremadi was born in 1983 in Linköping, Sweden. He received the B.Sc. degree in computer engineering from Sharif University of Technology, Tehran, Iran, in 2006 and the M.Sc. degree in computer science from Linköping University, Linköping, Sweden, in 2008. He has been working towards the Ph.D. degree in automation at Chalmers University of Technology, Gothenburg, Sweden, since 2008.

His current research interests include supervisory control and optimization of (timed) discrete event systems, using formal methods.

Petter Falkman received the M.Sc. degree in automation and the Ph.D. degree in electrical engineering both from Chalmers University of Technology, Gothenburg, Sweden, in 1999 and 2005, respectively.

He is currently a Senior Lecturer at the Department of Signals and Systems, Chalmers University of Technology, where he is also part of the Wingquist Laboratory, a research center for virtual product and production development. Since 2009, he has been Program Director for the Automation and Mechatronic Program at Chalmers University of Technology. His main research topic concerns virtual preparation, specification and optimization of discrete event systems for production systems. This research is performed in collaboration with and support from a number of companies including ABB, Dassault Systemes, Siemens, and Volvo.