Operation Behavior Modeling using Relation Identification and Visualization Algorithms

Kristofer Bengtsson, Bengt Lennartson

***Abstract*— The behavior of a system can be described by a set of operations – sometimes called activities or tasks. The process to specify these operation seems to be a real challenge in various situations, for example when designing automation systems or keeping track of the work at an emergency department. In practice, operation behavior specification is often quite inflexible because every possible execution route is explicitly defined. In this paper, a modeling method and a relation identification and visualization algorithm is introduced that does not explicitly specify operation routes, where instead the operation behavior is specified using the execution restrictions in transition conditions for each operation. This enables the possibility to create multiple projections of the operation relations to enable better understanding.**

I. Introduction

Developing new products and systems is stimulating yet challenging. Conflicting agendas and a multiplicity of perspectives among developers and other stakeholders make the development process intriguingly complex. This paper studies how to represent and specify the operation behavior of a system or product, where operations describe what the final system can do and how it is done. The process to specify these behavior is a real challenge in industrial practice.

The logical operation behavior of a system, which describe when operations start, when they finish, and how they interact with each other, results from a large number of constraints. Specifying this behavior requires integrating high-level requirements, such as cycle time, quality issues, and process limitations, with low-level constraints, such as mutual exclusion, safety concerns, and execution details. It is therefore a challenge not only to specify the behavior, but also to understand and represent it from various levels and perspectives. Throughout the design process the behavior model will evolve and transform when new design decisions are made and the understanding and knowledge are increased.

The most widespread industrial tool for specifying and modeling operations is probably the Gantt chart [1], which is easy to use and understand and intuitive to work with [2]. However, since operations are fundamentally logical, planning using Gantt charts alone can worsen development problems. Today, it is fairly common for certain operation 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, especially in early development phases 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, in order to avoid complexity, can complicate the introduction of changes in the product or manufacturing process later in the development process.

This lack of flexibility is also found in other operation planning tools, such as PERT charts [3], statecharts [4], sequential function charts (SFCs) [5], Petri nets (PNs) [6], and workflow management tools [7]. All of these attempt to describe operation behavior using operation sequences in various ways. However, that is an inflexible and limiting approach [8].

This paper presents a new algorithm for visualizing operation relations, which is an extension of the work by Bengtsson et. al [9], [10]. The algorithm requires a set of operations and their transition conditions, together with entities, resources and variables. This enables the possibility to create multiple projections of the operation relations to enable better understanding. This is accomplished by creating various Sequences of Operations, SOPs, including a multiplicity of sequences and operation relations. The key discoveries that accomplishes this, are the state-based operation relation identification and visualization methods. These methods makes it possible to use the presented research not only when designing automation systems, but also other types of situations, like keeping track of all the patients at an emergency department.

Generating various projections throughout the design process will give an understanding of the behavior, from initial high-level planning to the detailed task execution, enabling the possibility to specify operation behavior in a flexible way.

In the next section, the operation modeling is defined. In Section III the relation identification methods are introduced and in Section IV, the two examples are presented.

II. Modeling operation behavior

The presented modeling approach is an extension of the work by Bengtsson et. al [9], [10] and is described in more detail in [8]. The models can be translated into formal representations, for example an extended finite automaton (EFA) [11], a generalized automaton including variables, guards, and actions, and is more elaborated in these papers. Supremica [12] is a verification, synthesis and optimization tool that can be used with EFA models, hence, both formal verification, synthesis and optimization can be applied directly in the suggested EFA models.

The behavior of an operation model, O, can be represented by the state model depicted in Fig. 1. The initial location is denoted O^i, the executing location O^e, and the finished location O^f. The start transition between O^i and O^e has a precondition denoted by $O^{\uparrow}$ and a postcondition $O^{\downarrow}$ on the stop transition between O^e and O^f. An operation can also be restarted but is not discussed in this paper.

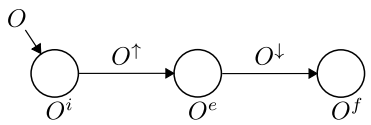

Fig. 1. A model of operation O

An operation can also be defined by a set of sub-operations describing the detailed behavior of the operation. These operations are included in a sequences of operations (SOP) structure, which

K. Bengtsson, is with Sequence AB, Göteborg, Sweden, e-mail: kristofer@sekvensa.se.

B. Lennartson is with the Automation Research Group, Department of Signals and Systems, Chalmers University of Technology, Göteborg, Sweden.

will be discussed later. Before presenting more details of the operations, let us first discuss entities and resources.

A. Entities and resources

Operations describe the behavior of a system, where the system is specified by entities and resources. Entities are objects that need to be transformed by operations to reach a complete state. Resources are defined as objects that aid in the transformation of entities. Resources that can perform operations have a set of abilities defining what type of operation they can perform, where an ability is defined as an operation model.

An entity or a resource, E, can be represented by a tuple of state variables, i.e., $E = (x_1^E, \ldots, x_n^E)$. A single state variable represents a specific aspect of E, for example, a door can have a state variable x_1^{door}, where the domain of x_1^{door} is denoted X_1^{door} with the values $X_1^{door} = \{opened, closed\}$, representing the positioning aspect of the door. A door can also have state variables representing other aspects, for example, related to its manufacturing or whether it is locked. Which state variables to use and what values they can take depend on the intended use of the system model. A state variable can also be shared among entities and resources when it represents a shared aspect or interaction. Each operation also has a state variable representing its location, where the domain of O_k is $\{O_k^i, O_k^e, O_k^f\}$.

The state variables for all entities, resources and operations are included in a state vector $x = (x_1, \ldots, x_n)$ that include the current value $x \in X$ of every entity and resource variable and operation location. The purpose of a system is to transform the state variables of the relevant entities such that a goal state is reached. A goal state can be defined by a predicate $\mathcal{X}_m : X \to \mathbb{B}$ that defines desired marked values for some of the state variables. Possible initial state(s) for the analysis is defined by a corresponding initial predicate $\mathcal{X}_i$.

B. Operation modeling using transition conditions

The pre- and postconditions, i.e., $O^\uparrow$ and $O^\downarrow$, of operation O constitute the core of the operation. Requirements and constraints related to the operation are specified using these transition conditions. A condition $C(x, \acute{x}) : X \times X \to \mathbb{B}$ is a predicate on the current value, x, and the next value, $\acute{x}$. For example $O_1^\uparrow \equiv x_1 = 1 \wedge \acute{x_1} = 0$ means that before the transition x_1 must be equal to 1, and after the transition the value must have been updated to 0. If x_1 does not include 0 in its domain or another specification hinders a change from 1 to 0, the condition will evaluate to false. A condition can also be seen as a function that maps a state vector to a set of new state vectors. If the function only returns the empty set, the condition evaluates to false and if the result includes more than one set, it represents a choice in the operation.

In the algorithm presented next, a condition is represented by a function programmed in a programming language. It can for example contain calls to external programs for evaluating the condition or include knowledge about the past transition order. However, if the operation models need to be translated to EFA models, the function is restricted to state variables.

The modeling approach presented here is used to allow the user to be as free as possible to express currently know design intentions. However, to be able to work in this way, the model needs to be visualized to be fully understood. The first step in visualizing the operations is to identify the relations among them.

III. Relation identification

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 conditions will restrict this behavior. If an operation includes the state of another operation in a transition condition, for example, $O_\ell^\uparrow \equiv O_k^f$, then the two operations are *directly related*. For example, it is obvious from studying the two operations that O_k will always execute before O_ℓ. However, most operations will not be directly related to each other, even though they are related in some way. Consider, for example, O_k, O_ℓ, and O_m, where $O_\ell^\uparrow \equiv O_k^f$ and $O_m^\uparrow \equiv O_\ell^f$. Then O_k will always precede O_m even though this is not obvious from examining only O_m and O_k; these are *indirectly related*.

To understand and use the operation model, an algorithm was presented in [9] that identifies these relations using EFA models and the tool Supremica [12]. The algorithm can utilize the built in functionality in Supremica for supervisory synthesis and guard extraction [13], but faces the challenge of state-space explosion and somewhat limited condition expressions. Therefore an alternative algorithm is presented here.

A. Operation relations identification

To analyze and reason about the relations among operations, one approach is to examine the possible locations of an operation, related to when other operation conditions are enabled. An operation O_k will be located in one of its three locations when operation O_ℓ starts. By identifying the states where $O_\ell^\uparrow$ is enabled, the possible locations of O_k can be found. A set denoted $O_k^{O_\ell^\uparrow}$ is created, which can be one of the following seven location combinations $\{O_k^e\}$, $\{O_k^f\}$, $\{O_k^i, O_k^e\}$, $\{O_k^i, O_k^f\}$, $\{O_k^e, O_k^f\}$, and $\{O_k^i, O_k^e, O_k^f\}$. For example, if $O_k^{O_\ell^\uparrow} = \{O_k^f\}$, 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., $O_k^{O_\ell^\uparrow}$, $O_k^{O_\ell^\downarrow}$, $O_\ell^{O_k^\uparrow}$, and $O_\ell^{O_k^\downarrow}$, must be identified and compared. Observe that $O_k^{O_\ell^\uparrow}$ is the set of possible locations of O_k when the start condition $O_\ell^\uparrow$ of O_ℓ is enabled, i.e. when O_ℓ has the possibility to start. The possible combinations of these state sets can be grouped into the following relation types (complete definition in [9]):

Definition 1 (Relations between O_k and O_ℓ):

- Always in sequence: $O_k \succ O_\ell$
- Sometimes in sequence: $O_k \succsim O_\ell$
- Parallel: $O_k \| O_\ell$
- Alternative: $O_k + O_\ell$
- Arbitrary order: $O_k \oplus O_\ell$
- Hierarchy: $O_k \sqsubset O_\ell$
- Other: $O_k \curlywedge O_\ell$

□

B. Identifying relations

To identify the relations among the operations, the state-space must be searched in some way. One approach trying to avoid the state-space explosion problem is to use the relation identification Algorithm 1, which is a search-based algorithm that incrementally updates the relations. The search algorithm will find a number of execution paths (defined by the noOfRounds variable) from the initial state to a goal state.

The first thing that happens during each search round is that the state is set to the initial state. After that, all operations that are enabled, i.e., whose transition conditions are satisfied in that state, are identified by evaluating all operations and adding them to the enabledOps set.

Algorithm 1: Find operation relations

```
Input: 𝒪
Result: RelationMatrix
for noOfRounds do
    state = init;
    enabledOps = getEnabledOps(𝒪, state);
    while not goalState(state) or enabledOps not empty do
        foreach O_s in enabledOps do
            foreach O_t in 𝒪 do
                append location of O_t to
                stateRelations[O_s, O_t];
            end
        end
        rOp = getRandomOp(enabledOps);
        newStates = rOp.takeTransition();
        newState = getRandomState(newStates) enabledOps =
        getEnabledOps(𝒪, newState);
    end
end
return convert stateRelations to RelationMatrix;
```

The search starts with the while loop, which terminates when either a goal state is found or no operations can be enabled in the current state (i.e., an invalid path is found). In each iteration of the while loop, the state relations are updated. The stateRelation matrix stores the state of the operations related to the start and stop events of each operation. For each enabled operation (i.e., either enabled start or enabled stop transition), the current states of all other operations are appended to the stateRelation matrix. After that, a random operation is selected from enabledOps; its enabled transition is taken and the corresponding actions update the state. Finally, all operations enabled after the state change are added to enabledOps and the loop repeats. Finally in the algorithm, the stateRelation matrix is converted into a RelationMatrix according to Definition 1.

The pseudo code of Algorithm 1 does not reveal all aspects of the algorithm. One aspect is how to handle invalid paths, i.e., if no operations are enabled and the state is not marked. If an invalid path is found, the relation matrix should not be updated during that round. Therefore, the implemented algorithm uses a temporary stateRelation matrix during each iteration, which updates the global one, if the identified path is valid.

When a single path is identified, only some of the states are added to the stateRelation matrix. Therefore, new paths need to be found to identify when an operation is enabled. The number of rounds needed to identify the relations is highly dependent on the structure of the model. The algorithm can actually not guarantee that all possible relations have been found, however, due to the randomness of the algorithm, a good distribution is possible. The performance is studied using three test cases.

Performance evaluation: This evaluation was run on a standard laptop and the code was compiled to java bytecode. No warm-up phase or other methods to find the "real" execution time was used since the absolute time is not of focus in this evaluation. Rather, the interesting comparison is the time differences between the various test cases used and to find out if the algorithm only takes seconds or minutes.

The first test case, $Case1$, is based on a case study conducted during the DARPA research program Adaptive Vehicle Make [14]. The case study was part of a larger exercise in which a number of research teams tested the interoperability of the tools developed during the research project. The exercise involved analyzing the manufacturability of a small remote-controlled car comprising approximately one hundred parts. The input to the relation identification algorithm came from a language workbench developed by Intentional Software [15], [16] and the output was generated to be used in a planning and optimization tool that identified possible manufacturing methods and resources for operation sequences. The study includes 50 operations and approximately 70 variables.

The second test case $Case2$ is a set of completely parallel operations, i.e. the transition condition always evaluate to true. The last case, $Case3$, is a specially designed case to highlight the limitations of the algorithm. That case is shown in Fig. 2, which is a SOP including two sequences, where $O_{S1}, \ldots, O_{Sn}$ have always in sequence relations and $O_{P1}, \ldots, O_{Pn}$ have parallel relations (observe also that $O_{Si} \| O_{Pj}$, $i = 1, \ldots, n$ $j = 1, \ldots, m$).

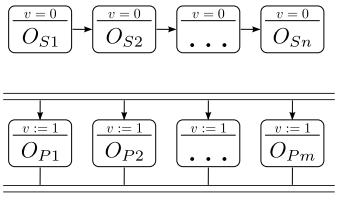

Fig. 2. A SOP with two sequences representing a hard to solve case

In the first test, 10 000 rounds were run and the execution time and the identified relations were studied. This was repeated 20 times. $Case1$ completed 10 000 rounds in 13 seconds for each of the 20 iterations. But only 50% of the cases identified all relations, although usually only one or two pairs of operation relations was not fully identified. When the number of iterations were increased to 15 000, every iteration found the correct relations in around 20 seconds.

$Case2$ was tested using 25 and 50 parallel operations. This test case represents a state space of 3^{25} and 3^{50} respectively. The 25 operations completed 10 000 rounds in 13 seconds, were each iteration found all relations. All relations was also found when running 50 parallel operations, but each iteration completed in as much as 72 seconds.

The last case, $Case3$, had seven sequential operations ($n = 7$) and 6 parallel operations ($n = 6$). The challenge is that when any of the parallel operations are executed, the variable v will be updated to 1, disabling the sequential operations. Hence, the sequential operations must execute before the parallel operations. 10 000 rounds were executed in 0.7 seconds, but no path was found that completed all operations, which means that no relations could be identified. Even when the number of rounds were increased to 300 000, completed in 20 seconds, no paths were found.

When studying the execution time for the individual steps of the algorithm, most time was spent on updating the state relations. Each operation has two location sets for every operation (i.e. $O_k^{O_\ell^\uparrow}$ and $O_k^{O_\ell^\downarrow}$) resulting in $2n^2$ location sets, where n is number of operations. When all operations are parallel, also each operation event is enabled (until it has fired), resulting in an update of all the location sets, after each transition . This leads to an exponential growth of execution time with the number of operations when they are running unrelated to each other.

Next test checks the number of necessary rounds to find all

relations. The number of iterations is unlimited during the test, but the iteration is terminated if the relations haven't been updated for the last 5000 rounds. $Case1$ completes in 10 000 to 17 000 iterations after finding all relations.

When executing the two sets of parallel operations in $Case2$, using the 3000 round limitation, the 25 operations terminates after average 3011 rounds and the 50 operations after 3019 rounds. Since the last 3000 rounds were unchanged, the correct relations was identified after just 11 and 19 rounds respectively. The algorithm actually finds all relations of the 50 parallel operations in less then 0.2 seconds.

Search heuristics and parallel computing: As can be seen, the algorithm is efficient to identify operation relations when many operations are in parallel. However, problems occur in $case3$ when the probability to reach the last operation in the sequence is low. This can be handled with a search heuristic. One implemented heuristic is applied after 1000 rounds of invalid paths, were instead of using random search, directly related operations are executed together. For $case3$, that means that the complete sequence will be executed before starting a parallel operation. With this added functionality, $case3$ is solved in less then 1 second.

Another approach when an invalid path is found, would be to start a new local search from the last state, trying to disable events not leading to a marked state. Then it may even be possible to synthesis extra guards on the parallel operations, limiting them from starting before the last operations. But this has not yet been implemented.

Another challenge is the exponential increase in time with the number of operations when they are highly parallel. Two approaches to handle this case have been implemented. The first approach is to use multiple threads that finds relations and than merge the results together. The second is to avoid updating the relation matrix if the found relation is already in the matrix.

The algorithm is highly parallelizable since the search is random. When the number of parallel algorithm instances are doubled, the execution time is almost cut in half. The current implementation runs independent algorithm instances in separate threads. After a number of rounds, the result is sent to a unification algorithm, which merges the results from the algorithm threads.

When parallel computing is combined with the check to avoid unnecessary updates, test $Case2$ with 50 operations and 10 000 iterations completes in less than 10 seconds and the other tests were all below 3 seconds using 10 threads on a two core computer.

The number of rounds required to find all relations is an important property of the search algorithm. However, it is in many cases not possible to guarantee that all relations have been found. But in practice it is not a big problem, since the identified relations will never be less restricted than the true relations. What that means is that for example if the true relation is a sequential relation, the algorithm will never find a parallel relations, but it can find a sequence relations (or more specifically a sometime in sequence relation) if the true relation is parallel. Observe that id the relation is a sequential relation between two operations, it will be enough with only one round to find the relation.

The next step is to study how the algorithm can adopt to various model structures e.g. many parallel relations or hard to find goal states. It may be necessary to run multiple iterations and study the distribution of the result, to predict the accuracy of the results, and to tune the required iterations for when the model is updated. Also the possibility to incrementally update the operation relations on a small change of a single operation condition needs to be be studied.

IV. Visualization examples

When working with complex information, it is often necessary to represent it from various perspectives to fully understand it. A quotation from information visualization research states: "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" [17]. In this section, two case studies are therefore presented to highlight the possibility to create multiple projections of operation relations to enable better understanding.

When the relations have been pairwise identified, these relations needs to be visualized in some way. Here, a graphical language called Sequences of operations, SOP, is used. The SOP, SOP is a model that represents a set of operations, $SOP^{\mathcal{O}} \subseteq \mathcal{O}$, their relations, SOP^{R}, and a set of sequences, SOP^{S}. The $SOP^{\mathcal{O}}$ operations are grouped into a SOP for various reasons, for example, that they are executed by the same resource, involved in the same product assembly, or related to a specific safety concern. A single operation can also be included in multiple SOPs. The main reason for grouping these operations is to be able to consider the relations among the included operations, which are defined by relations $SOP^{R} = \{< O_1, rel, O_2 >, < O_1, rel, O_3 >, \ldots, < O_n, rel, O_{n-1} > \{$, where rel is a relation from Definition 1, which represents the pairwise relation between operations. The final part of the SOP is the sequences. Sequence $s \in SOP^{S}$ is a graph that connects set $s^{\mathcal{O}} \subseteq SOP^{\mathcal{O}}$ of operations to visualize their relations, which is done using arrows, lines, and Boolean expressions.

Algorithms for creating sequences is presented in [9] together with definition of the graphical notation. In the following section, two examples of automatically generated SOPs in two applications are shown. Let us start with an example from a study at an emergency department.

A. Visualization of SOPs at Emergency departments

The possibility to use operation behavior identification and visualization in hospitals was studied at an emergency department (ED) in Sweden, during one week. One of the most obvious observations during the study, was the challenge for the personal to get an overview of the patient flow, especially when the ED was overcrowded with patients. During the peak time of the day, each of the three medical section in that ED could have 10-15 patients at the same time, making the work very fragmented. Getting an overview of all these patients, not only by the personal at each section but also the coordinators at the ED, is important both for current work and for improvements. It is also important for the patients to better understand what is going on and why they are waiting.

In Fig. 3 various projections are shown. The top left SOP represents the possible route for patient P_1 when arriving at 11:40. On arrival, the patient is registered and assigned an $exam$ operation and the alternative discharge or admittance. These are matched and merged with resource abilities. New operations are automatically added based on the $exam$ transition condition, for example that triage is needed due to overcrowding, and that the patient needs to be transported to one of the sections.

The $triage.*$ operation is marked red to show that the patient is waiting for that task and the $*$ define that the task can be executed by multiple resources, i.e. the task consist of the alternative between the two operation instances $triage.T1$ (Triage Team 1) and $triage.T2$ (Triage Team 2). This SOP will evolve with the patient, and operations will be added and removed based on the patient's

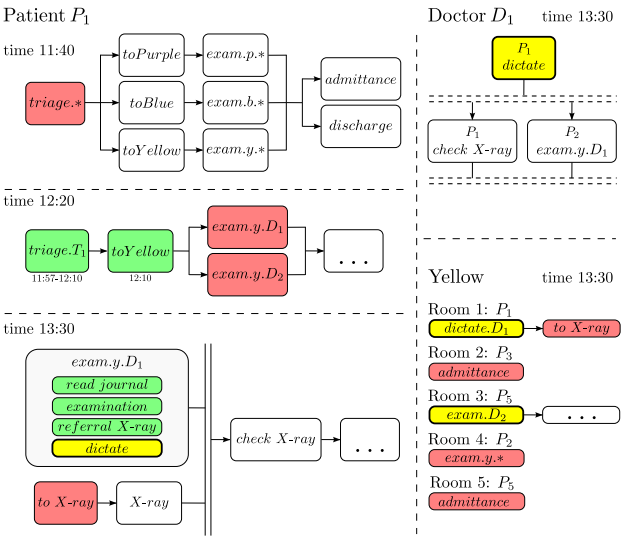

Fig. 3. Overview of Yellow section and Patient P_1

examination. The SOP below shows the status at 12:20 and at the bottom at 13:30. The green marked operations have been completed and their start and stop time is shown below. The patient was placed in the yellow section, which removes the other possible routes in the first SOP.

In the 13:30 SOP, the patient is currently being examined by doctor D_1, who has just written a referral to an x-ray examination and is currently dictating the examination. When the referral was written two new operations were added, $X\text{-}ray$ and $check\ X\text{-}ray$, and based on the $X\text{-}ray$ requirements, also $to\ X\text{-}ray$ was added. Since D_1 is not currently with the patient, it is possible to start $to\ X\text{-}ray$ directly.

The SOP at the top right shows the current and coming operations for D_1. To understand why a specific operation is not started, it is necessary to understand what the various resources are doing. The SOP will also give the personal guidance on what to do next. But maybe the most important projection to give an overview is the SOP in the lower right showing the patient in each room.

It is also possible to identify why an operation is not starting by studying its transition condition. By visualizing the relations between the studied operation and the sequences of operations that satisfies these condition, it is easy to understand what a patient is actually waiting for.

B. Visualization of SOPs during product and production development

The small toy car shown in Fig. 4 is a product assembled in the production lab at Chalmers University of Technology. The car consists of four sheet metal plates – a roof, E_1, two sides, E_2 and E_3, and a floor, E_4 – and three Lego modules, E_5, E_6, and E_7, placed on the floor. The sheet metal plates are welded together by the five weld entities, E_{W12}, E_{W13}, E_{W14}, E_{W24}, and E_{W32}.

The parts are positioned and welded together, which is specified by a set of shared state variables. These variables are updated by six Position and five Weld operations. The relations among these operations are shown in Fig. 5. The three Lego pieces E_5, E_6, and E_7 must be mounted on the floor before the roof is positioned and welded. After that the two sides are positioned and welded, which completes the car.

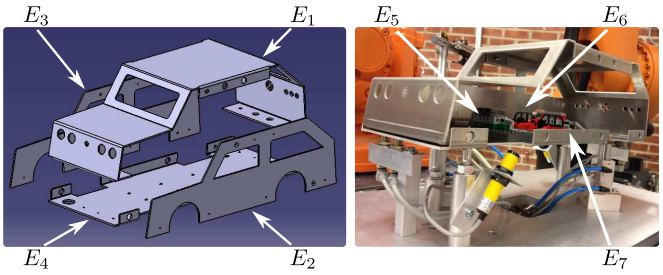

Fig. 4. A toy car developed and assembled in the production lab at Chalmers University of Technology

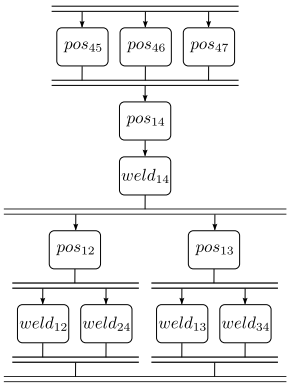

Fig. 5. Product operation sequences for the toy car example.

The car is assembled by the set of resources shown in Fig. 6. The assembly involves four robots, R_1, R_2, R_3, and R_4, a fixture, Fix, a conveyor, $Conv$, and an automatic guided vehicle, AGV. The two large robots, R_1 and R_2, can move parts into and out of the fixture. They each have two tools for handling parts, i.e., one vacuum lifter for sheet metal plates and one pneumatic gripper for small Lego objects. The two small robots, R_3 and R_4, can weld, the conveyor can transport Lego modules, and the AGV can transport the sheet metal plates and the assembled car.

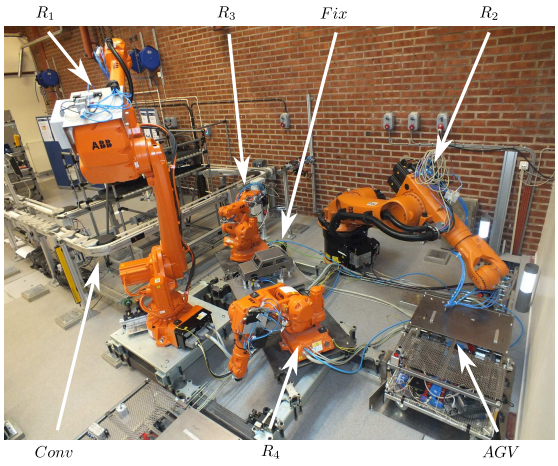

Fig. 6. The manufacturing cell in the production lab at Chalmers University of Technology

The final generated SOP shown in Fig. 7 visualizes the final operation execution. The SOP shows a sequence projections describing the sequences per resource. This view indicates what each resource

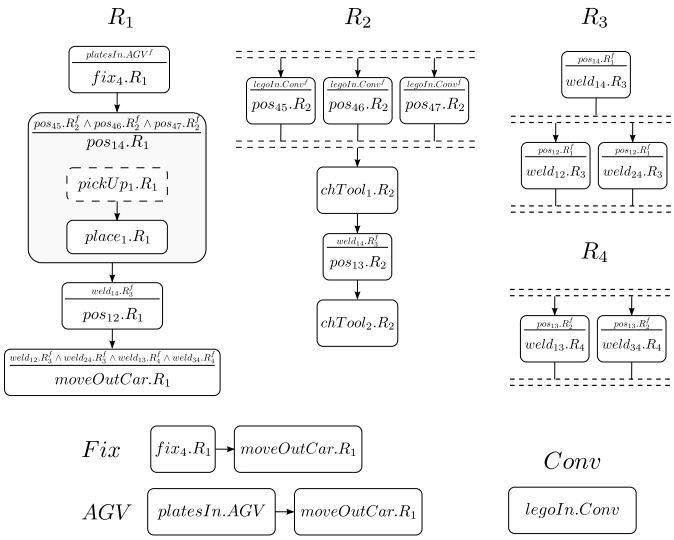

Fig. 7. The default behavior represented by sequences structured on resources

does in the manufacturing system and how the resources interact. In the sequence for R_1, operation $pos_{14}.R_1$ is expanded and shows its companion SOP. The first operation in that SOP, $pickUp_1.R_1$, has a dashed line indicating that the operation is a preoperation. The robot can pick up E_1 before the Lego pieces have been put in place by R_2. All of the operations have sub-operations and many of them also have these types of preoperations (not shown in the projection).

In this SOP, we can also see that some resources jointly execute one operation, such as $fix_4.R_1$, which is executed by Fix and R_1. This is because some of the sub-operations of $fix_4.R_1$ involve closing the clamps in Fix. Observe also that it is not necessary to repeat the same Boolean expression twice, for example, $moveOutCar.R_1$ in the bottom sequences, since every sequence is considered when representing the operation relations. For example, it is also possible to present $moveOutCar.R_1^{\uparrow} \triangleq weld_{12}.R_3^f \wedge weld_{24}.R_3^f \wedge weld_{13}.R_4^f \wedge weld_{34}.R_4^f$ in a separate sequence to avoid the long expression.

V. Conclusion

The problem with current specification practice of operation behavior arises because the routing behavior is an indirect consequence of the requirements to start executing an operation, which involve, for example, the state of a resource, product, or another operation. These operation requirements can result in many types of routing behavior, which will be almost impossible to describe in a graphical model. A better approach is therefore presented when designing operations and the routing behavior.

This paper has illustrated that operation behavior specification is a highly important activity during the design process. The The presented algorithm makes it possibility with good performance to generate multiple projections, which make it possible for a set-based design approach.

The next important step is to create a good user interface and intuitive tools. Therefore this research will continue with developing a tool called Sequence Planner, which already has implemented most of these methods.

Acknowledgement

This work was carried out at the Wingquist Laboratory VINN Excellence Centre within the Area of Advance Production at Chalmers, supported by the Swedish Governmental Agency for Innovation Systems (VINNOVA), and within the CAPE research school, supported by the Knowledge Foundation. This work was also supported by General Motors and SAAB Automobile. These supports are gratefully acknowledged.

References

[1] J. M. Wilson, "Gantt charts: A centenary appreciation," *European Journal of Operational Research*, vol. 149, no. 2, pp. 430 – 437, 2003.

[2] H. Kerzner, *Project Management: A Systems Approach to Planning, Scheduling, and Controlling, Ninth Edition*. J. Wiley & Sons, 2006. ISBN 0471741876.

[3] R. Levin and C. Kirkpatrick, *Planning and Control with PERT/CRM*. McGraw-Hill, 1966.

[4] D. Harel, "Statecharts: A visual formalism for complex systems," *Science of Computer Programming*, vol. 8, pp. 231–274, 1987.

[5] IEC 61131-3:2003, "Programmable controllers—part 3: Programming languages," tech. rep., International Electrotechnical Commission, 2003.

[6] C. Girault and R. Valk, *Petri Nets for System Engineering: A Guide to Modeling, Verification, and Applications*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2001.

[7] D. Georgakopoulos, M. Hornick, and A. Sheth, "An overview of workflow management: From process modeling to workflow automation infrastructure," *Distributed and Parallel Databases*, vol. 3, pp. 119–153, 1995.

[8] K. Bengtsson, *Flexible design of operation behavior using modeling and visualization*. PhD thesis, Department of Signals and Systems, Chalmers University of Technology, Göteborg, Sweden, 2012.

[9] K. Bengtsson, P. Bergagård, C. Thorstensson, B. Lennartson, K. Åkesson, C. Yuan, S. Miremadi, and P. Falkman, ""sequence planning using multiple and coordinated sequences of operations"," *IEEE Transactions on Automation Science and Engineering*, vol. 9, no. 2, pp. 308–319, 2012.

[10] 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.

[11] M. Sköldstam, K. Åkesson, and M. Fabian, "Modelling of discrete event systems using finite automata with variables," in *Proc. 46th IEEE Conference on Decision and Control*, (New Orleans, USA), Dec 2007.

[12] Supremica, "http://www.supremica.org,"

[13] S. Miremadi, K. Åkesson, and B. Lennartson, "Symbolic computation of reduced guards in supervisory control," *IEEE Transactions on Automation Science and Engineering*, vol. 8, pp. 754–765, 2011.

[14] DARPA AVM, "http://www.darpa.mil/our_work/tto/programs/-adaptive_vehicle_make__(avm).aspx," 2012.

[15] C. Simonyi, M. Christerson, and S. Clifford, "Intentional software," *SIGPLAN Not.*, vol. 41, pp. 451–464, Oct. 2006.

[16] Intentional software, "http://intentsoft.com/," 2012.

[17] R. Spence, *Information Visualization - Design for Interaction (2nd Edition)*. Pearson Education, 2006. ISBN 0132065509.