Event- and time-based design of operation sequences with uncertainties in execution times

Event- and time-based design of operation sequences with uncertainties in execution times

Nina Sundström
Automation Research Group
Department of Signals and Systems
Chalmers University of Technology
nina.sundstrom@chalmers.se

Bengt Lennartson
Automation Research Group
Department of Signals and Systems
Chalmers University of Technology
bengt.lennartson@chalmers.se

Abstract

In this paper, we introduce a complete framework for integrating the design of the manufacturing process and control system. We show how operation sequences can be designed in a modeling tool, Sequence Planner (SP), and how relations between operations may be expressed using logical conditions. An approach to convert the SP model into a constraint programming model for optimization is presented. The time-based solution is transformed to an event-based description. Due to uncertainties in execution times, some logical restrictions based on the optimal schedule are relaxed to avoid unnecessary delays. The control logics to achieve the desired operation sequences are added to the SP model. Hence, the process designer can revise the sequences if necessary, and the control designer retrieves a logical description of the optimized process that can be automatically converted to control code.

1. Introduction

Process planning is the practice of establishing operations and resources required to manufacture a product as well as determining the sequential order of operations based on their intermutual relations. With the increasing need for flexibility in industry, the manufacturing processes and their relations have become more complex [4]. To handle the increase in complexity, sequence planning in industry tends to be either over-specified, simplified, unclear or incorrect [2]. Scheduling entails coordinating the operation sequences and deciding the order to perform each operation based on a specific criterion [1]. Examples of previous research conducted on integrating process planning and scheduling can be found in [15],[5] and [14].

There are a wide range of tools for illustrating operation sequences. In industry, Microsoft Excel is often used as well as more graphical tools such as Gantt charts [18] and PERT charts [7]. Sequence Planner Language (SPL), a new language used for expressing sequences of operations, was presented in [6]. SPL is implemented in the tool Sequence Planner (SP) that integrates product, process and automation design. Also, the tool enables automatic generation of control code.

Throughout the development of a manufacturing system, new requirements and demands may change the sequences of operations [9]. As a consequence, the design of the manufacturing control system starts at a quite late stage in the development of the manufacturing system. Control engineers design the control system based on an optimal or at least suboptimal sequence of operations given in a schedule, often visualized using Gantt charts [13]. However, a drawback with Gantt charts is the lack of information regarding embedded conditions that express when operations can start. Hence, a large part of development time for control designers are spent on finding correct information and transforming the time-based optimal solution to an event-based formulation that is implemented in the control system [3].

A more solid approach would be to automate the whole procedure from the optimization to the control code generation. Not only to support the control engineer but also to give feedback to the process designer when modeling the sequences of operations. The process designer would then be able to iterate the procedure if the resulting schedule contains undesirable behavior for the operation sequences.

In this paper, a procedure for integrating the design of the process and the manufacturing control system is presented. The operation sequences are modeled in SP and represented as a constraint programming (CP) model in order to optimize the sequences. SP has visualization features that enables the process designer to study the optimal order of operations related to the original sequence requirements. Also, it is possible to examine operations from different views, e.g. from a product or resource view [6]. If the result is not satisfying, the process engineer could redesign the original operation sequences. The resulting schedule opens up for robustness analysis due to uncertainties in the execution times of the individual operations. If operations do not have a logical coupling, e.g. a relation in the original SP model or a common unit capacity resource, feasibility tests are performed to see if unnecessary conditions may be removed. Hence, we retrieve less restrictive sequences to avoid unnecessary delays when execution times differ from the nominal ones. The generated control logics based on the optimal solution are added to the SP model. Hence, the control engineer is handed an event-based description of the process. This can be compared to the Gantt chart, only containing timing information and operation order. As a final step, automatic generation of control code is achievable.

The paper is outlined as follows. In the following section, Section 2, an operation model is defined. How to model relations between operations is also covered as well as how to model in SP. In Section 3, an approach to parse an operation model in SP to a CP model is given. A method to generate logical constraints from an optimal schedule is explained in Section 4. Finally, a case study of an aero engine structure assembly plant is presented in Section 5 followed by conclusions in Section 6.

2. Modeling

2.1. Operation model

There are various ways to define an operation model. Since we are interested in generating event-based control policies expressing start conditions for operations in relation to other operations, we will use the definition of a logical operation model.

An operation has three different locations: initial, executing and finished. In order for an operation to move between two locations, a transition condition has to be fulfilled. This condition includes a guard expression and a transition action. When the guard expression evaluates to true and a transition action may execute successfully, the condition is satisfied. The transition condition between the initial and executing location is defined as a precondition. Similarly, a postcondition is defined as the transition condition between the executing and the finished location [2]. An automaton extended with variables for an operation is depicted in Fig. 1.

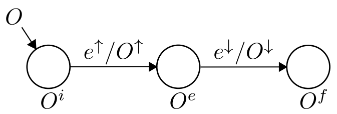

Figure 1. A model of operation O.

The initial location is denoted O^i, the executing location O^e and the final location O^f. The transition condition between O^i and O^e is given by the start event $e^{\uparrow}$ and the precondition $O^{\uparrow}$. The stop event $e^{\downarrow}$ and the postcondition $O^{\downarrow}$ are connected to the transition from O^e to O^f.

2.2. Operation relations

The core of the operation is the pre- and postconditions, i.e. $O^{\uparrow}$ and $O^{\downarrow}$ for an operation O. These transition conditions are used when specifying constraints related to the operation, e.g. to define relations between operations. A graphical representation of different operation sequences

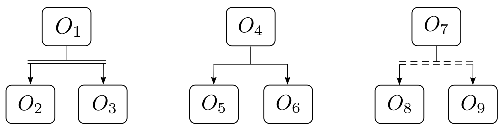

Figure 2. Graphical representations of parallel, alternative and arbitrary sequences.

is depicted in Fig. 2. We can view the operations as self-contained and formulate preconditions to express when an operation is allowed to start in the different sequences. Resource booking for a unit capacity resource R is defined as

$$R^{+} \equiv R = 0 \wedge \acute{R} = 1$$

where $\acute{R}$ is the next value of R. Similarly, the definition for unbooking a resource is

$$R^{-} \equiv R = 1 \wedge \acute{R} = 0$$

These definitions may be extended for resources with capacities greater than one.

Parallel sequence For the parallel sequence, operations O_2 and O_3 will both have operation O_1 in their precondition which can be specified as

$$O_i^{\uparrow} = O_1^f \quad i = 2, 3$$

Alternative sequence In the alternative sequence, either operation O_5 or O_6 can execute after the completion of O_4. The preconditions for operations O_5 and O_6 can be expressed as

$$O_i^{\uparrow} = O_4^f \wedge A^{+} \quad i = 5, 6$$

The condition A^{+} on the common boolean variable with initial value A = 0 models a mutual exclusion between the operations in the arbitrary sequence. The value of A will never be reset. Hence, only one of the alternative operations will execute.

Arbitrary sequence The arbitrary sequence is a combination of parallel execution and mutual exclusion. The operations in the arbitrary sequence can execute in an arbitrary order without overlap. A common resource is booked during the execution of an operation. Hence, mutual exclusion prohibits the operations to execute at the same time. The preconditions for operation O_8 and O_9 can be formulated as

$$O_i^{\uparrow} = O_7^f \wedge R^{+} \quad i = 8, 9$$

The postconditions contain the unbooking of the common resource R

$$O_i^{\downarrow} = R^{-} \quad i = 8, 9$$

2.3. Modeling using Sequence Planner

A modeling tool called Sequence Planner (SP) has been developed by the Automation Group at Chalmers University of Technology [6]. It can be used for the definition and visualization of product recipes. SP uses an approach where operations can be modeled as self-contained with only relevant information on when and how an operation can execute. Sequences of operations based on e.g. different resources or products can be displayed by using a projection of operations that somehow relates to that resource or product. Recently, it has been shown that transport operations can be automatically generated from simulation software [8]. The next step is to also generate process operations based on the product recipe from external CAD software.

In SP it is possible to model straight, parallel, alternative and arbitrary sequences as shown above. The relations between operations are specified using pre- and postconditions. Operations may also have pre- and postactions in order to formulate e.g. booking and unbooking of resources. When an operation requires one or more resources, a precondition can be formulated to express that the necessary resources have to be available before executing the operation. A preaction can state the actual resource booking. Hence, when the precondition is fulfilled, the resource is booked and the operation related to the transition condition starts executing. A processing time for each operation can also be specified.

3. Constraint programming

The origin of CP lies in the artificial intelligence and computer science communities and can be traced back to the constraint satisfaction problems (CSPs) studied in the 1970s [11]. A CSP entails finding a feasible set of decision variables subject to a number of constraints, i.e. assigning values to variables that satisfy all constraints [12]. During the last decades, CP has evolved into solving optimization problems, that is finding the solution in a feasible set that minimizes or maximizes a given objective function [10]. The constraints may be of various types; linear, nonlinear, logical, cardinal and global. This makes modeling problems using CP much more flexible compared to operations research, where only linear and integer constraints may be used.

3.1. Scheduling using constraint programming

In this paper, IBM ILOG OPL is used for posing the optimization models and IBM ILOG CP Optimizer is used for solving the models. The solver uses constraint programming targeting both constraint satisfaction and optimization problems [17].

The decision variables for CP scheduling problems using OPL are intervals. Each interval represents an operation and is characterized by a start value, end value and size. Examples of constraints that are used in OPL to describe the structure of an SOP are e.g. $endBeforeStart()$ and $alternative()$. The former states that the start of one interval has to be greater than the end of another interval. The latter models an exclusive alternative between different intervals. Another useful function in OPL is $cumulFunction$ which can be used for the resource allocation. This function is incremented as a resource is booked and decremented when the resource is released, acting as a pulse function.

If a resource has to be booked for the duration of several operation intervals a pulse function may not be used. Instead, an OPL expression $stepAtStart()$ can be used to increment the cumulative function at the start of an operation. In the same way, $stepAtEnd()$ can be used to decrement the resource at the end of an operation. An upper bound for the $cumulFunction$ corresponds to the capacity of the resource.

3.2 Mapping of operation sequences

In SP it is possible to graphically represent four types of sequences: straight, parallel, alternative and arbitrary. The three latter ones are depicted in Fig. 2. As previously mentioned, operations in CP using OPL are represented by interval decision variables. The length of the intervals equals the processing time of each operation. The mapping of the operation sequences depicted in Fig. 2 to CP will be described in the following paragraphs. Each interval I_i relates to an operation O_i. An interval variable I_1 is initiated as

```
1: interval $I_1$ size($O_1^{min}$,$O_1^{max}$)
```

where O_1^{min} is the minimum execution time for O_1 and O_1^{max} is some sufficiently large constant. Serial execution of two operations, O_1 and O_2 can be described in two ways. One alternative is

```
1: endBeforeStart($I_1$,$I_2$)
```

which allows for a segment of time between the two intervals when nothing is performed. However, if a resource is to be booked during O_1 until O_2 starts, it can be convenient to use

```
1: endAtStart($I_1$,$I_2$)
```

This constraint will ensure that I_1 ends as I_2 starts and thus guarantees that the resource is booked for the necessary period. Two operations, O_2 and O_3, executing in parallel without mutual dependency can be modeled using two sequential constraints. If O_1 is an operation preceding the two parallel operations, the system can be modeled by

```
1: endBeforeStart($I_1$,$I_2$)
2: endBeforeStart($I_1$,$I_3$)
```

If a SOP contains alternative operations, the alternative() constraint can be used. Suppose either O_5 or O_6 is to be executed after the preceding operation O_4. The following code will ensure the correct behavior.

```
1: interval $I_i$ optional size($O_i^{min}$,$O_i^{max}$), $i = 5, 6$
2: endBeforeStart($I_4, D$)
3: alternative($D, \{I_5, I_6\}$)
```

First, I_5 and I_6 are initiated as optional, i.e. neither of them have to be executed. Then, D , a dummy interval is constrained to start after I_4. The alternative() constraint then states that if D is executed, it will start and end with either I_5 or I_6. This will force one of these optional intervals to be executed.

An arbitrary sequence is when two operations are both to be executed, but during different time intervals. This behavior can also be interpreted as a parallel sequence with a dummy resource that mutually excludes the operations. The necessary constraints for mutual exclusion of two operations O_8 and O_9 are the following

```
1: endBeforeStart($I_7$,$I_8$)
2: endBeforeStart($I_7$,$I_9$)
3: cumulFunction C = pulse($I_8$)+pulse($I_9$)
4: $C \leq 1$
```

A cumulative function C is used to represent a dummy resource. On row 4, C is constrained to a maximum value of 1. A $pulse$ is a timed expression which attains value 1 as its target interval executes. As a result, with C defined as the sum of two pulses on row 3, the two operations cannot execute simultaneously without violating the upper bound of C. In other words, I_8 and I_9 mutually exclude each other. However, the order in which they execute is not specified.

From the mapping of operation sequences to constraints in CP, we see that pre- and postconditions expressing a relation to another operation can be modeled by endBeforeStart() or endAtStart(). For pre- and/or postconditions containing e.g. resource booking/releasing, cumulative functions may be used to represent this behavior in CP. For a resource R, a cumulative function is defined by

$$\text{cumulFunction } R = \sum_i \text{pulse}(I_i)$$

where i are the indices of all operation intervals that require R during execution.

4. Event generation

In the previous section we described how we can generate a constraint programming model from a set of operations and their pre- and postconditions. The resources required to perform each operation are given as well as the execution time. Information regarding relations between operations may also be given. When the operation model in SP has been parsed to an optimization model in CP, an optimal schedule may be generated. The schedule specifies the execution order for the operations in terms of starting times for each operation. Since execution time may vary from the nominal execution time, it is necessary to express the order between the operations using logics in the final control implementation. Thus, the optimal operation order will be expressed as logical conditions in the original operation model in SP. Hence the full model will contain conditions based on the original operation sequences as well as conditions from the optimization.

If the process designer finds the schedule to be unsatisfactory, e.g. due to time gaps, SP can be used for analyzing what conditions that caused this operational behavior in the schedule. Since complexity in manufacturing processes often are tackled by over-specifying the system, the process designer can in retrospect remove unnecessary conditions. If changes are made to the model, a new schedule has to be generated. Once the schedule is approved by the process designer, the control designer retrieves a logical description of the optimized process that can be automatically converted to control code.

In order to avoid unnecessary delays when execution times differ from nominal ones, we would like to perform analysis on the constraints before adding conditions to the SP model. Some operations in the schedule will have a logic coupling due to e.g. relations in the original model. However, others may be able to start before a preceding operation have finished. As a consequence, we would like to study the constraints in order to relax certain conditions and gain less restrictive operation sequences. Hence, for non-nominal execution times, the operation sequences would be less affected by delays than without these analysis. We will present an approach to generate conditions based on the optimal schedule that guarantees that the execution order of the operations is maintained. Also, we will introduce a method to analyze the operations and their conditions to see it they can be modified in order to obtain a less restrictive system for varying execution times. But first, we will present an illustrative example to demonstrate our approach.

4.1. Illustrative example

Consider two robots working in parallel. The job to be performed by Robot 1 consists of three operations in a straight sequence while Robot 2 has two subsequent operations. The second operation for Robot 1 and the last operation for Robot 2 are performed in a mutual zone. Therefore, the robots have to book the zone before entering it and unbook it when they leave in order to avoid collisions. The sequence of operations performed by Robot 1 is given by

$$J_{R1} : O_{11}(5) \rightarrow O_{12}(3) \rightarrow O_{13}(3)$$

where the processing time for each operation is given in the brackets after the operation. Robot 2 executes operations

$$J_{R2} : O_{21}(5) \rightarrow O_{22}(7)$$

The transition conditions for the operations are

$$\begin{aligned} O_{12}^{\uparrow} &= O_{11}^{f} \wedge R^{+} \\ O_{13}^{\uparrow} &= O_{12}^{f} \\ O_{22}^{\uparrow} &= O_{21}^{f} \wedge R^{+} \end{aligned}$$

where R is a common boolean variable with initial value 0 and maximum value 1. The operations requiring the shared zone R, have a postcondition where R is unbooked. In this quite trivial example it is easy to see what the resulting time-optimal schedule would look like, see Fig. 3.

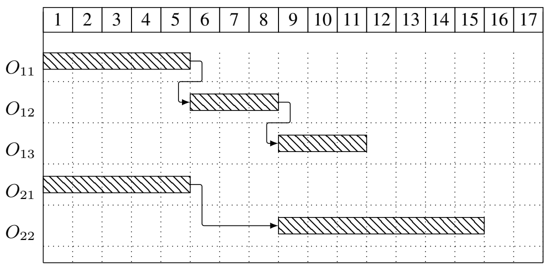

Figure 3. The time-optimal Gantt chart.

The set of conditions generated from the optimal schedule with indirect relations removed is

$$\begin{aligned} O_{12}^{\uparrow} &= O_{11}^{f} \wedge O_{21}^{f} \\ O_{13}^{\uparrow} &= O_{12}^{f} \\ O_{22}^{\uparrow} &= O_{12}^{f} \end{aligned}$$

As an example, O_{11} and O_{13}, have an indirect relation since O_{13} has a relation to O_{12}, which in turn, has a relation to O_{11}. Hence, if indirect relations were not removed, O_{11}^{f} would be a precondition for O_{13} to start. The conditions related to the operation relations in the original operation model are

$$\begin{aligned} O_{12}^{\uparrow} &= O_{11}^{f} \\ O_{13}^{\uparrow} &= O_{12}^{f} \\ O_{22}^{\uparrow} &= O_{21}^{f} \end{aligned}$$

Hence, by removing the conditions from the original operation model from the conditions generated from the optimal schedule, we end up with the following conditions.

$$\begin{aligned} O_{12}^{\uparrow} &= O_{21}^{f} \\ O_{22}^{\uparrow} &= O_{12}^{f} \end{aligned}$$

Based on these logical restrictions, the constraints endBeforeStart(I_{21}, I_{12}) and endBeforeStart(I_{12}, I_{22}) are added to the CP model to guarantee that the starting order of the operations is maintained. Since the duration for operations may vary from nominal execution times, we would like to asses these constraints in order to see if we have any unnecessary restrictions in our operation model. Hence, the constraint endBeforeStart(I_{21}, I_{12}) is modified to startBeforeStart(I_{21}, I_{12}) and a feasibility test is performed to see if this yields a feasible solution. Similarly, endBeforeStart(I_{12}, I_{22}) is modified to startBeforeStart(I_{12}, I_{22}). The feasibility test will result in an infeasible solution since operations O_{12} and O_{22} share a resource with capacity 1. Hence, we reintroduce endBeforeStart(I_{12}, I_{22}) to our model. The transition conditions for the operations in the original operation model are updated accordingly.

$$\begin{aligned} O_{12}^{\uparrow} &= O_{11}^{f} \wedge\ R^{+} \wedge (O_{21}^{e} \vee O_{21}^{f}) \\ O_{13}^{\uparrow} &= O_{12}^{f} \\ O_{22}^{\uparrow} &= O_{21}^{f} \wedge\ R^{+} \wedge O_{12}^{f} \end{aligned}$$

The Gantt chart in Fig. 3 displays a time gap between the two operations for Robot 2. This is due to the common resource for O_{21} and O_{22}. Hence, if O_{21} has a delay smaller than the time gap, i.e. the execution time for O_{12}, the optimal solution would still hold since O_{12} can start during the execution of O_{21}. Assume that O_{21} has an execution time $T_{21} = 5 + \Delta$ where Δ is the deviation from nominal execution time. When constraints are not modified, the makespan MS for the system can be expressed as

$$MS = 15 + \Delta$$

When modifying the constraints, the makespan is given as

$$MS = 15 + \max(\Delta - 3,\ 0)$$

Hence, the optimal solution would still hold if $\Delta \leq 3$.

4.2. Approach

Let $\mathcal{O}$ be the set of all operations in the operation model and $\mathcal{O}^{j}$ the set of operations that are completed before operation O_j starts. We observe that $\mathcal{O}^{j} \subset \mathcal{O}$. The following conditions guarantee that the order in the optimal schedule is maintained.

$$O_{j}^{\uparrow} = \bigwedge_{O_i \in \mathcal{O}^{j}} O_{i}^{f} \quad \forall\, O_j \in \mathcal{O} \tag{1}$$

Hence, we restrict an operation to start before all preceding operations have finished executing. The number of operations in the precondition is reduced by removing indirect relations.

As previously mentioned, we would like to analyze the conditions related to the operations before adding them to the SP model. This is due to uncertainties in execution time for operations. For non-nominal execution times, we would like to avoid unnecessary delays. Let the conditions generated from the optimal schedule be contained in a set $\mathcal{C}^{s}$ where indirect relations have been removed. The set of conditions stating operation relations in the original model in SP is denoted $\mathcal{C}^{m}$. It is of no interest to modify the original relations specified in the SP model. Therefore, to retrieve the conditions to analyze we need to study the conditions in $\mathcal{C}^{s} \setminus \mathcal{C}^{m}$. For each condition $C_j \in \mathcal{C}^{s} \setminus \mathcal{C}^{m}$, we add constraints on the related operation intervals I in the CP model

$$\text{endBeforeStart}(I_i, I_j) \quad \forall\, O_i \in \text{pre}(C_j),\ O_j \in \mathcal{O}$$

where pre$(C_j) \subseteq \mathcal{O}^{j}$ is the set of operations for which there are logical conditions in C_j. The generated endBeforeStart() constraints are included in a set $\mathcal{P}$. As previously mentioned, operations may have one or several operations in their precondition. The analysis differ between these two cases and will be described in the following paragraph.

Analyzing constraints The analysis of the constraints depends on the number of operations in a precondition. When an operation O_j only have operation O_i as a precondition, the constraint endBeforeStart$(I_i, I_j) \in \mathcal{P}$ is to be studied. In order to see if it is possible for operation O_j to not only start after but also during the execution of O_i, we replace this constraint with a constraint startBeforeStart(I_i, I_j). This constraint specifies that interval I_i has to start executing before interval I_j can start its execution. A feasibility test determines if this would result in a feasible solution. If not, we reintroduce endBeforeStart(I_i, I_j) to $\mathcal{P}$.

If a precondition contains two or more operations, they will be in parallel. To motivate this, assume two operations in a precondition could be in a straight sequence. The second operation would have the first operation in its precondition since this operation starts and ends before the second operation. Hence, the first operation would be removed from the precondition containing the two operations due to indirect relations. This implies that if an operation depends on several operations to have finished before executing, these operations will be in parallel. Consider the case when a precondition for operation O_k contains two operations O_i and O_j

$$O_k^{\uparrow} = O_i^f \wedge O_j^f \qquad O_k^{\uparrow} \in \mathcal{C}^s \setminus \mathcal{C}^m$$

It is not sufficient to study the constraints individually. Not only do we need to examine each constraint startBeforeStart(I_i, I_k) and startBeforeStart(I_j, I_k) separately, we also need to check the stronger conjunction of the constraints. Hence, if we start with evaluating if all three operations can execute in parallel and the result is infeasible, we may continue to assess the constraints individually. We start with evaluating the constraint related to the operation with the latest completion time contained in pre(C_k). Assume that the completion time for I_i is greater than the completion time for I_j. In this case, endBeforeStart(I_j, I_k) will be reintroduced to $\mathcal{P}$ and startBeforeStart(I_i, I_k) will be evaluated. If the result is infeasible, we continue with interval I_j and evaluate startBeforeStart(I_j, I_k) and reintroduce endBeforeStart(I_i, I_k) to $\mathcal{P}$. In order to avoid a large number of combinations, we have limited the number of considered parallel operations to be two. The remaining operations will have to finish executing before the succeeding operation can start.

Feasibility tests The assessments mentioned in the previous paragraph consists of feasibility tests. Two main feasibility tests are performed to see if the problem is feasible or not. The first test checks if operations share a common resource with unit capacity and hence are mutually excluded. The second test uses constraint propagation to see if it is feasible to modify a constraint. The feasibility test is formed as a constraint satisfaction problem using the optimization model excluding the objective function and the addition of the constraints in $\mathcal{P}$. A limit for the search in terms of time or fails is necessary. If a solution can not be found before this limit is reached, the test is determined to be infeasible. In this case, we will restrain an operation to start after the previous operation has finished.

Generating logical conditions When the analysis of the constraints has been completed, the result is added to the operation model in SP. We go through the constraints in $\mathcal{P}$ to see if they have been modified or not. Then, constraints are parsed to operation relations.

$$\text{startBeforeStart}(I_i, I_j) \Rightarrow O_j^{\uparrow} = O_j^{\uparrow} \wedge (O_i^e \vee O_i^f)$$
$$\text{endBeforeStart}(I_i, I_j) \Rightarrow O_j^{\uparrow} = O_j^{\uparrow} \wedge O_i^f$$

Thus, we update the precondition for the operations in the SP model with the result from the analysis. The SP model will then contain conditions based on the original sequence requirements and conditions based on the optimal schedule. Operations without logical coupling have been assessed to see if it is necessary for a succeeding operation to wait for the preceding operation to finish or if it is sufficient to wait for the preceding operation to start.

4.3. Algorithms

One approach to ensure that the order in the schedule is maintained is to sort the operation intervals after earliest start time. If we go through the sorted intervals we can specify that all intervals that are completed before a specific operation interval starts are added to the precondition for the corresponding operation. This procedure is described in Algorithm 1 as well as removing indirect relations from the preconditions by the local function hasRelation. The input to this algorithm is the sorted array

Algorithm 1: Generate conditions

```
   Input: $I$, $\mathcal{O}$
   Output:
1  for $j$ in $I$ do
2      $\mathcal{O}^j = \emptyset$;
3      for $i$ in $I$ do
4          if startOf($i$) < startOf($j$) $\wedge$
           endOf($i$) $\leq$ startOf($j$) then
5              $\mathcal{O}^j = \mathcal{O}^j \cup O_i$;
           end
       end
6      $\mathcal{C}^s = \mathcal{C}^s \cup O_j^{\uparrow}$;
7      for $O_k$ in $\mathcal{O}^j$ do
8          for $O_l$ in $\mathcal{O}^j \setminus O_k$ do
9              if hasRelation ($O_k, O_l$) then
10                 pre($C_j$) =pre($C_j$) $\setminus O_k$;
11                 continue to next $O_k$;
               end
           end
       end
   end
```

of operation intervals, I, and the set of operations, $\mathcal{O}$. The first step is to add preconditions for all operations according to (1). Next, the operations in the precondition are examined to see if indirect relations exist. If so, operations are removed from the precondition. When the conditions in $\mathcal{C}^s$ have been determined, Algorithm 2 is used to add constraints to the operation intervals. The input is the sorted array of operation intervals, I, and the conditions related to the optimal schedule $\mathcal{C}^s$ that do not exist in the condition set for the original operation model $\mathcal{C}^m$, i.e. $\mathcal{C}^s \setminus \mathcal{C}^m$. For each operation in the set of precondition operations, endBeforeStart() is added to the set of constraints, $\mathcal{P}$. The assessment of constraints is performed

Algorithm 2: Specify constraints

```
  Input: $I, \mathcal{C}^s \setminus \mathcal{C}^m$
  Output:
1 for $C_j$ in $\mathcal{C}^s \setminus \mathcal{C}^m$ do
2     for $O_i$ in pre($C_j$) do
3         $\mathcal{P} = \mathcal{P} \cup$ endBeforeStart$(I_i, I_j)$;
      end
  end
```

using the feasibility test given in Algorithm 3. The input is two operation intervals, I_i and I_j, and the output is either true or false. If the corresponding operations share a resource with capacity 1, the method returns false. Else, the constraint endBeforeStart$(I_i, I_j) \in \mathcal{P}$ is replaced with startBeforeStart(I_i, I_j). The function solve uses constraint propagation to test if the constraints from the optimization model together with the constraints in $\mathcal{P}$ are satisfied. The algorithm returns true if solve returns a feasible solution. Else, the constraint startBeforeStart$(I_i, I_j) \in \mathcal{P}$ is reset to endBeforeStart(I_i, I_j) and the algorithm returns false. When all feasibility tests have been performed, the

Algorithm 3: Feasibility test

```
  Input: $I_i, I_j$
  Output: True/False
1 if sharedResource $(I_i, I_j)$ then
2     return False;
  end
3 $\mathcal{P} = \mathcal{P} \cup$ startBeforeStart$(I_i, I_j) \setminus$
  endBeforeStart$(I_i, I_j)$;
4 if solve () then
5     return True;
  end
6 $\mathcal{P} = \mathcal{P} \cup$ endBeforeStart$(I_i, I_j) \setminus$
  startBeforeStart$(I_i, I_j)$;
7 return False;
```

final step is to generate conditions to the SP model based on the constraints in $\mathcal{P}$, see Algorithm 4.

Algorithm 4: Add conditions

```
  Input: $\mathcal{P}$
  Output:
1 for p in $\mathcal{P}$ do
2     switch p do
          case endBeforeStart$(I_i, I_j)$
3             $O_j^\uparrow = O_j^\uparrow \wedge O_i^f$;
          end
          case startBeforeStart$(I_i, I_j)$
4             $O_j^\uparrow = O_j^\uparrow \wedge (O_i^e \vee O_i^f)$;
          end
      endsw
  end
```

5. Case study

For our case study we have considered a manufacturing facility for aero engine structures, to be more specific the turbine exhaust case. Traditionally this structure has been delivered as one big piece of casting which has then been machined in several steps. The manufacturing process is very robust and several steps may be performed in the same station by using multi-purpose machines. However, a major drawback is that only a few suppliers in the world can deliver these big pieces of casting. The aero engine industries are therefore looking into the possibility of using automated manufacturing processes. They study possible methods to divide the structure into smaller parts which are automatically assembled to sub-assemblies which are further assembled to the final structure. Much in the same way as processes in the automotive industry where automated manufacturing is used to a great extent.

Process description The manufacturing steps studied in this paper, are the processes that refine smaller parts before they are assembled, as well as the assembly operations of the smaller parts to the larger parts, called segments. An extension of this manufacturing system was studied in [16]. The part of the manufacturing facility studied contains tables for fixating and unfixating the parts, a robot for transporting the parts and machines for milling, measuring and welding. The parts and segments are attached to fixtures throughout all operations. There are 2 types of segments, each consisting of either 2 or 3 smaller parts. Additionally, there are 4 different types of parts. The product recipe for all part types is given by

$$\text{Fixate (2)} \rightarrow \text{Milling (22;32;42)} \rightarrow \text{Unfixate (4)}$$

The execution time for the different operations are given in the brackets following the operation. The milling operation has three different execution times depending on the part type to be processed. The product recipes for the segment types are given by

$$\text{Fixate (8)} \rightarrow \text{Measuring (2)} \rightarrow \text{Welding (20)} \rightarrow \\ \rightarrow \text{Measuring (2)} \rightarrow \text{Unfixate (2)}$$

The manufacturing cell have one robot that performs the necessary transportation of products between operations. The characteristics of the considered system are given below.

- 14 resources, 36 parts, 13 segments, 513 operations
- Alternative transportation paths due to buffers.

- In the product recipes some operations are repeated, which results in parts returning to a workstation previously visited.
- Parts always have one ore more resources booked throughout their operation sequences.

The objective is to minimize the makespan of the production of 13 segments, i.e. a total of 36 parts.

Results The optimization was run on a Windows 7 64-bit system with a 2.66 [GHz] Intel Core2 Quad CPU and 4 [GB] of RAM. The suboptimal makespan for 13 segments was 724 minutes. More information regarding the optimization can be found in [16]. The suboptimal schedule of the manufacturing system contains 403 operations. This number differs from the number of operations in the SP model which is 513. This is due to alternative paths where the number of operations in each branch differ. The number of conditions generated from the schedule, $\mathcal{C}^s$ and the number of relation conditions in the SP model, $\mathcal{C}^m$ are given below. Also, the number of conditions to analyze, $\mathcal{C}^s \setminus \mathcal{C}^m$, is given.

$$\begin{aligned} |\mathcal{C}^s| &= 640 \\ |\mathcal{C}^m| &= 390 \\ |\mathcal{C}^s \setminus \mathcal{C}^m| &= 247 \end{aligned}$$

The number of preconditions in $\mathcal{C}^s \setminus \mathcal{C}^m$ containing more than one operation is 32. The analysis show that 135 of the conditions could be changed from O^f to $O^e \vee O^f$.

6. Conclusions

In this paper, we introduced a framework for integrating the design of a manufacturing system and the development of a control system. The modeling tool, Sequence Planner (SP), is used to model the operation sequences. The SP model is converted to a constraint programming model in order to optimize the operation sequences. The resulting time-based schedule is translated to an event-based description of the resulting operation order. Due to uncertainties in the operation execution times, some logical restrictions generated by the optimal schedule are relaxed. As a result from the less restrictive event-based operation sequences, unnecessary delays are avoided when operation execution times differ from the nominal ones. The result is fed back to the original SP model where control logics for final implementation may be automatically generated.

Acknowledgement

This work was carried out at the Wingquist Laboratory VINN Excellence Center within the Area of Advance Production at Chalmers. The work has been supported by the EU-FP7 FLEXA project, Swedish Governmental Agency for Innovation Systems (VINNOVA) and Swedish Research Council (VR). The support is gratefully acknowledged.

References

[1] In P. Beek, editor, *Principles and Practice of Constraint Programming - CP 2005*, volume 3709 of *Lecture Notes in Computer Science*. 2005.

[2] K. Bengtsson. *Flexible Design of Operation Behavior Using Modeling and Visualization*. PhD thesis.

[3] K. Bengtsson, B. Lennartson, C. Yuan, P. Falkman, and S. Biller. Operation-oriented specification for integrated control logic development. In *Automation Science and Engineering, 2009. CASE 2009. IEEE International Conference on*, pages 183–190, 2009.

[4] M. P. Fanti, B. Maione, S. Mascolo, and A. Turchiano. Event-based feedback control for deadlock avoidance in flexible production systems. *Robotics and Automation, IEEE Transactions on*, 13(3):347–363, 1997.

[5] J. Kempenaers, J. Pinte, J. Detand, and J.-P. Kruth. A collaborative process planning and scheduling system. *Advances in Engineering Software*, 25(1):3 – 8, 1996.

[6] 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*, 7:791–802, 2010.

[7] R. Levin and C. Kirkpatrick. *Planning and control with PERT/CPM*. New York, McGraw-Hill, 1966.

[8] P. Magnusson, N. Sundström, K. Bengtsson, B. Lennartson, P. Falkman, and M. Fabian. Planning transport sequences for flexible manufacturing systems. In *Preprints of 18th World Congress of the International Federation of Automatic Control*, 2011.

[9] H. Marri, A. Gunasekaran, and R. Grieve. Computer-aided process planning: A state of art. *The International Journal of Advanced Manufacturing Technology*, 14(4):261–268, 1998.

[10] P. Pardalos and M. Resende. *Handbook of applied optimization*. Oxford University Press, 2002.

[11] M. Pinedo. *Planning and scheduling in manufacturing and services*. Number v. 1 in Springer series in operations research. Springer, 2005.

[12] F. Rossi, P. van Beek, and T. Walsh. *Handbook of Constraint Programming (Foundations of Artificial Intelligence)*. Elsevier Science Inc., New York, NY, USA, 2006.

[13] G. Schmidgall, J. Kiefer, and T. Bär. Objectives of integrated digital production engineering in the automotive industry. In *Proceedings of the 16th IFAC World Congress, Prague, Czech Republic*, 2005.

[14] A. I. Shabaka and H. A. ElMaraghy. A model for generating optimal process plans in rms. *International Journal of Computer Integrated Manufacturing*, 21(2):180–194, 2008.

[15] W. Shen, L. Wang, and Q. Hao. Agent-based distributed manufacturing process planning and scheduling: a state-of-the-art survey. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 36(4):563–577, 2006.

[16] N. Sundström, O. Wigström, P. Falkman, and B. Lennartson. Optimization of operation sequences using constraint programming. In *Proceedings of 14th IFAC Symposium on Information Control Problems in Manufacturing*, 2012.

[17] P. Van Hentenryck. *The OPL optimization programming language*. MIT Press, Cambridge, MA, USA, 1999.

[18] J. M. Wilson. Gantt charts: A centenary appreciation. *European Journal of Operational Research*, 149(2):430 – 437, 2003.