Available online at www.sciencedirect.com

ScienceDirect

Procedia CIRP 63 (2017) 521 – 526

www.elsevier.com/locate/procedia

The 50th CIRP Conference on Manufacturing Systems

Toward Process Control from Formal Models of Transformable Manufacturing Systems

Otto J. Bakker[a], Jack C. Chaplin[a], Lavindra de Silva[a,*], Paolo Felli[a], David Sanderson[a], Brian Logan[b], Svetan Ratchev[a]

[a]*Institute for Advanced Manufacturing, The University of Nottingham, University Park, Nottingham, NG7 2RD, UK*
[b]*School of Computer Science, The University of Nottingham, Jubilee Campus, Nottingham, NG8 1BB, UK*

* Corresponding author. Tel.: +44-(0)115-748-6352; fax: +44-(0)115-951-3666. *E-mail address:* Lavindra.deSilva@nottingham.ac.uk

Abstract

The automation and flexibility of production systems is a key step towards improved profitability and competitiveness in high labour cost areas, when producing high-complexity, low-volume products. In the Evolvable Assembly Systems (EAS) project, the 'manufacturability' (or 'realisability') and 'control' algorithms were introduced to accommodate the batch-size-of-one production of highly customisable products. These algorithms enable checking *whether* a production line can manufacture a given product with its available set of resources, and *how* the product should be manufactured, e.g. which resources to use, and when. To this end, the authors formally define production recipes, which represent products, and manufacturing resources which make up a manufacturing facility. This paper re-defines these notions in the ISO-standard EBNF (Extended Backus-Naur Form) notation, and adapts the the manufacturability and control algorithms to accommodate the new definitions. The new algorithms and data structures reflect more closely the ones that are used in an implemented software tool. This paper also reports a method by which recipes and resources could be used to generate manufacturing process controllers in the Business to Manufacturing Markup Language (B2MML) standard. In doing so, this paper takes a step toward a complete path from the formal specification of a manufacturing facility and the products to be manufactured, to the automatic generation of executable process plans.
© 2017 The Authors. Published by Elsevier B.V. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).
Peer-review under responsibility of the scientific committee of The 50th CIRP Conference on Manufacturing Systems
Keywords: Evolvable Assembly Systems; Controller Synthesis; Standardisation.

1. Introduction and Context

Advanced manufacturing is the result of the trend towards the integration of informatics with traditional manufacturing systems. These dynamic, adaptive, decentralised systems are used to manufacture product lines with a shorter time to market, increased product diversity, greater specialisation, and shorter lifecycles. A number of government-backed initiatives have arisen in recent years to support this trend, including the German Industrie 4.0 initiative [1,2], EU EFFRA Roadmap [3], US DMDII Strategic Plan [4], and the recent Japanese "Connected Manufacturing" Industrial Value Chain Initiative [5]. Academic research in the area includes reconfigurable manufacturing systems [6,7] with 'plug and produce' technologies [8,9], holonic manufacturing [10,11], organic computing [12], and the related evolvable production systems and evolvable assembly systems [8,13–15], among others.

The Evolvable Assembly Systems project at the University of Nottingham aims to investigate advanced manufacturing systems that are self-adaptive [16] and able to address the 'batch-size-of-one' problem, where each product to be assembled may be unique [15]. In [17], a formal approach was proposed to determine both *whether* a particular product is manufacturable given a set of available manufacturing resources (the *manufacturability problem*), and *how* the product should be manufactured using those resources (the *control problem*). Their approach takes as input the manufacturing resources, and the product in the form of a *recipe* specifying the operations necessary to manufacture the product, and produces a controller specifying the detailed steps to be executed by each manufacturing resource in the production line. In this paper the abstract representations of [17] are linked to concrete ISA-95 standards [18], and their algorithms are converted into a format that manipulates data structures that are defined using the standard ISO/IEC Extended BNF language [19], which also reflect more closely the data structures that are used in the actual implementation.

This paper first describes an example system in Section 2 which will serve as a running example in later sections. Section 3 defines production recipes and manufacturing resources in EBNF notation, which are used as input to the algorithms that check manufacturability and synthesise an abstract controller. Section 4 explains how a resulting abstract controller can be translated to a more suitable format for controlling industrial processes, and Section 5 provides the detailed algorithms for

2212-8271 © 2017 The Authors. Published by Elsevier B.V. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).
Peer-review under responsibility of the scientific committee of The 50th CIRP Conference on Manufacturing Systems
doi:10.1016/j.procir.2017.03.159

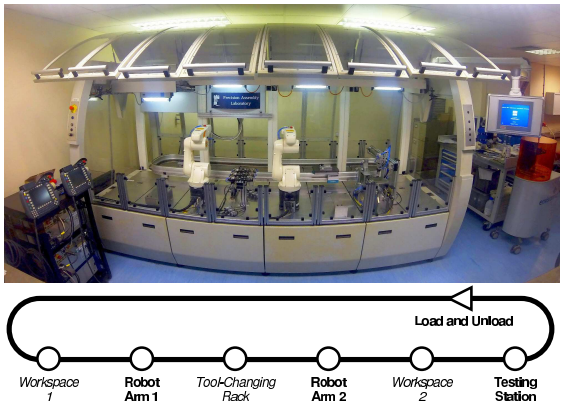

Fig. 1. The Precision Assembly Demonstrator from the front with safety screens raised (top), and its diagrammatic layout (bottom).

checking manufacturability and for synthesising an abstract controller. Section 6 summarises the paper.

2. Running Example

The Precision Assembly Demonstrator (PAD) [9] is based on the Modutec highly flexible assembly platform by Feintool. It consists of a shuttle transport system linking modular stations to a loading/unloading area; two KUKA six-axis robotic arms with associated workspaces; a shared tool changing rack; and a testing and inspection station. The modules and their layout is shown in Figure 1.

The parts to be assembled into a product are mounted on a pallet. At the loading/unloading area, the system operator can place the pallet on a shuttle that runs along a linear transfer system. In Figure 1, the six resources corresponding to the stations in the PAD are configured as follows: the pallet is first removed from the shuttle and placed at 'Workspace 1', a working area for 'Robot Arm 1', where the robot works on it before returning it to the shuttle; the 'Tool-Changing Rack' is shared between the two robot arms and holds a number of end effectors, including grippers and suction tools, that are used for product assembly; 'Robot Arm 2' and 'Workspace 2' are a mirror image of the previous robot and workspace, but otherwise identical; the 'Testing Station' allows vision and mechanical tests to be performed on the product once it has been assembled; the product is then returned to the loading/unloading area where an operator removes it from the system.

The product that is assembled with the PAD is a detent hinge for the cab interior of a truck. In the basic hinge, interior and exterior plastic leaves are fitted together and linked with a metal hinge pin. More complex hinges can be created by adding up to three metal ball–spring pairs within the interior leaf to adjust the engaging force. Additional end effectors may be used to apply glue to secure the hinge pin, or to engrave serial codes onto the leaves.

3. System Description

This section provides definitions for manufacturing resources which make up the manufacturing system, and for production recipes which represent products. These definitions are provided in the ISO/IEC Extended BNF language [19], and are based on the formal definitions in [17].

⟨Operation⟩	::=	⟨ObservableOp⟩ | ⟨ManufactOp⟩
⟨CompositeOp⟩	::=	⟨Test⟩ (";" ⟨Step⟩)*
⟨Step⟩	::=	⟨ParallelStep⟩ | ⟨ObservableOp⟩
⟨ParallelStep⟩	::=	⟨ObservableOp⟩ ("||" ⟨ObservableOp⟩)+
⟨Test⟩	::=	⟨ObsOperationName⟩ ⟨InputParts⟩
⟨ObservableOp⟩	::=	⟨ObsOperationName⟩ ⟨InputParts⟩ ⟨OutputParts⟩ | ⟨ObsOperationName⟩ "()" ⟨OutputParts⟩
⟨ManufactOp⟩	::=	⟨ObsOperationName⟩ | ⟨InternalOperation⟩
⟨InternalOperation⟩	::=	⟨IntOperationName⟩ | *nop* | ⟨Transfer⟩
⟨Transfer⟩	::=	*in* ":" *integer*+ | *out* ":" *integer*+
⟨InputParts⟩	::=	"(" ⟨Parts⟩ ")"
⟨OutputParts⟩	::=	"(" ⟨Parts⟩ ")"
⟨Parts⟩	::=	⟨Part⟩ ("," ⟨Part⟩)*
⟨Part⟩	::=	*string*
⟨ObsOperationName⟩	::=	*unique string*
⟨IntOperationName⟩	::=	*unique string*
⟨Recipe⟩	::=	⟨InitialRecipeState⟩ "{" ⟨RecipeTrans⟩ ("," ⟨RecipeTrans⟩)* "}"
⟨RecipeTrans⟩	::=	"(" ⟨RecipeState⟩ "," ⟨CompositeOp⟩ "," ⟨RecipeStateTo⟩ ")"
⟨InitialRecipeState⟩	::=	⟨RecipeState⟩
⟨RecipeState⟩	::=	*string*
⟨Resource⟩	::=	⟨InitialRecipeState⟩ "{" ⟨ResourceTransition⟩ ("," ⟨ResourceTransition⟩)* "}"
⟨ResourceTransition⟩	::=	"(" ⟨ResourceState⟩ "," ⟨ManufactOp⟩ "," ⟨ResourceStateTo⟩ ")"
⟨InitialRecipeState⟩	::=	⟨ResourceState⟩
⟨ResourceState⟩	::=	*string*
⟨Topology⟩	::=	⟨InitialTopState⟩ "{" ⟨TopTrans⟩ ("," ⟨TopTrans⟩)* "}"
⟨TopTrans⟩	::=	"(" ⟨ResStates⟩ "," ⟨ResOps⟩ "," ⟨ResStatesTo⟩ ")"
⟨InitialTopState⟩	::=	⟨ResStates⟩
⟨ResStatesTo⟩	::=	⟨ResStates⟩
⟨ResStates⟩	::=	⟨ResourceState⟩+
⟨ResOps⟩	::=	⟨ManufactOp⟩ ("||" ⟨ManufactOp⟩)*
⟨ResParts⟩	::=	⟨ResourceParts⟩+
⟨ResourceParts⟩	::=	"(" ⟨Parts⟩ ")" | "()"

Fig. 2. The EBNF syntax of operations and composite operations (above the double lines), and of recipes and resources (below the double lines).

3.1. Resources and Topologies

In a manufacturing system, a (manufacturing) *resource* is a piece of hardware with a top-level controller (and possibly sub-controllers), which is typically a Programmable Logic Controller (PLC). The PLC controls the resource, sending signals to the hardware to perform actions, and collecting readings from sensors. Manufacturing resources are either production resources, which primarily perform *observable operations* on parts, or transport resources, such as a conveyor belt or shuttle system, which perform *internal operations* that transport parts between resources. Production resources can also perform internal operations, which would then enable operations such as data logging (e.g. from sensors), or maintenance actions.

Resources are defined as state diagrams, or more specifically, as labelled transition systems (LTSs), where nodes cor-

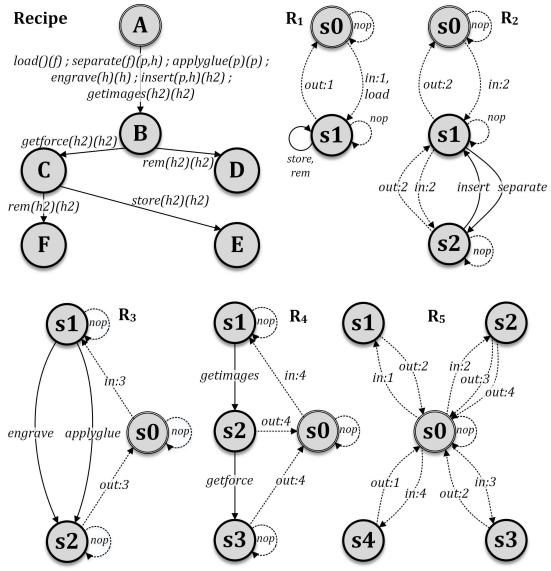

Fig. 3. Simplified labelled transition systems modelling the resources in our assembly platform, and an example of a recipe (top left). State *s0* and *A* are initial states, and the dotted transitions represent internal operations.

respond to the states of the resource, and edges correspond to operations that can be performed in the relevant states. Figure 2 defines resources and their operations as EBNF grammars.

For example, the LTSs representing the resources in the PAD are shown in Figure 3; when there are multiple transitions from one state to another they are shown on a single edge, separated by commas. Resource R_1 is able to load a new pallet, and then either remove a (potentially partially-)completed product for disposal or rework, or store it for delivery. The labels *in:1* and *out:1* represent *transfer operations*, which allow a part to be transferred between resources, from the one performing the *out* operation into the one performing the matching *in* operation. The *nop* labels denote special "no operation" tasks, which simply do nothing to the items and represent the machine idling. Resources R_2 and R_3 are the two robotic arms: R_2 can either take in a pallet on *in:2* and separate the hinge and pin, outputting each on a separate *out:2* transition, or accept each of those two parts on separate *in:2* transitions, insert the pin into the hinge barrel, and produce a single part on *out:2*. Resource R_3 can either engrave a serial number or apply glue to a given part, and R_4 is the testing station, which can perform a vision test and then possibly a force test. Note that *(i)* the vision test checks that the hinge is correctly assembled, so it must be performed before a force test, and *(ii)* it is not possible for the machine to idle between the two tests, hence the lack of a *nop* transition.

While resources R_1 to R_4 are production resources, R_5 is a transport resource which models the possible routing of parts by the shuttle system. The transitions represent the 'allowable' routes between production resources; e.g., although a newly loaded pallet can be transported from R_1 to R_2, a freshly glued part cannot be transported from R_3 to R_4 in order to prevent it becoming stuck to the testing equipment. Observe that the *out* transfer operations of production resources (e.g. *out:1* in R_1) match with *in* transfer operations of transport resources (e.g. *in:1* in R_5) and vice-versa.

These resources together form the *production topology*, formally defined as the cross product of the LTSs representing the resources, excluding transitions which have no matching in (e.g. *in:2*) and out (e.g. *out:2*) transfer operations [17].

3.2. Production Recipes

The products to be assembled on the system are defined through production *recipes*. A recipe includes the constituent parts and the operations required to process and assemble these parts into the final product, any tests that must occur to verify the product during and at the end of the manufacturing process, and how to respond to the results of tests (e.g., whether a partially completed product should be reworked or discarded following a test). The recipes specify how, but not where, these operations and checks should be performed in order to assemble a given product.

Recipes are formalised as LTSs, where the labels represent composite operations and nodes are states of parts in the assembly. Composite operations represent the observable operations to be performed by the resources on the parts, and an observable operation specifies the parts that are consumed and produced by it. Operations can be preceded by guards (tests), which test whether the product meets certain properties (e.g. whether a hole has the correct depth), and they can be combined in sequence or in parallel. Figure 2 defines recipes and composite operations as EBNF grammars. Symbol ';' denotes sequential composition, and '||' denotes parallel composition.

Figure 3 shows an example hinge recipe that is to be assembled by the PAD. The first composite operation (between states A and B) loads a new fixture pallet f, separates the pin p and hinge h, applies glue to p, engraves a serial number on h, and inserts p into h to form $h2$, which is then visually examined, generating data. At this point, a test is performed against the vision data (the guards are not shown in order to improve the readability of the diagram), and based on the result the product is either removed from the system or a force analysis is performed. The outcome of performing a test against the force data generated similarly determines whether the product is removed or sent to storage for delivery.

3.3. Manufacturability and Control

This section informally defines what it means for a production recipe to be *manufacturable* on a production topology, or in other words, *when* it is possible to manufacture a product using the supplied set of connected manufacturing resources. A recipe Recipe is said to be manufacturable on a topology Topology if it is possible to associate states in the topology to states in the recipe, by taking into account the parts currently present in the resources, such that each transition (composite operation) in the recipe can be executed by transitions (manufacturing operations) in the topology, and the same is possible for the entire recipe, irrespective of the outcome of guards and the 'path' through recipe transitions that might be followed at runtime. If a recipe is manufacturable on a topology, then there is an associated controller (or "solution") which specifies how resources should be orchestrated in order to manufacture the product.

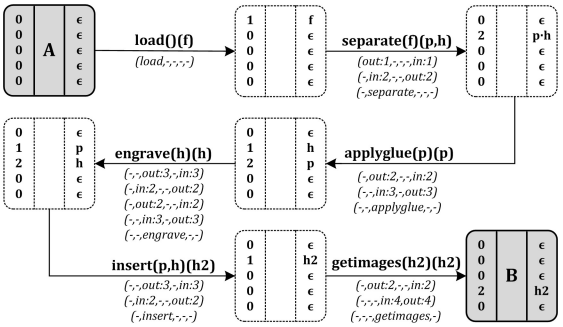

Fig. 4. A graphical illustration adapted from [17], showing a single transition in a controller, between the first two states in the recipe of Figure 3. The dotted states are "intermediate" steps. The overall transition in the controller is labelled with *(i)* the entire composite operation in the first recipe transition, and *(ii)* the complete sequence of transitions in the topology that are needed to execute the composite operation. Dashes are used in place of *nop* operations; symbol ϵ denotes the absence of parts; and the five vertical numbers (resp. strings) in states correspond to the states of (resp. parts in) the five resources.

Intuitively, a controller is a state diagram with the same structure as the recipe. A controller is obtained by annotating each recipe state with the associated resource states (i.e., topology state) and the parts being worked on in those states, and annotating each recipe transition with the sequence of topology transitions (or the "solution") that need to be carried out in order to execute the recipe transition.

Figure 4 depicts a part of a controller, corresponding to the first two recipe states in Figure 3. Each row in parenthesis on a transition represents one topology transition, and each element within a row represents the operation to be performed by the corresponding resource. For example "*(-,-,applyglue,-,-)*" indicates that the third resource should apply glue and that the rest of them should idle.

4. Controllers for Industrial Processes

This section shows how a controller defined in the previous section could be converted to the Business to Manufacturing Markup Language (B2MML) [18] standard. B2MML implements the ANSI/ISA-95 (Enterprise-Control System Integration) standards in XML, and is increasingly being adopted in the manufacturing industry [20]. B2MML defines the data and process models that form the interface to manufacturing execution systems (MESs) [21], which manage and monitor the work-in-progress on low-level shop floor control systems, via PLCs for example. Thus, B2MML provides higher-level enterprise control systems, such as multi-agent systems, with a convenient interface to hardware. This section focuses on the B2MML operations-schedule, which specifies each of the operations to be performed in sequence.

The operations-schedule contains the Bill of Process, Bill of Materials, and Bill of Tooling. To accommodate this information, an operations-schedule is divided into one or more operations-requests, each describing an operation. Each operations-request consists of one or more segment requirements, which subdivide the operation into smaller, lower-level tasks (for example, the operations-request may be a 'pick & place' operation, resulting in segment requirements for moving

```
⟨OperationsRequest⟩
  ⟨OperationsType⟩ Loading ⟨/OperationsType⟩
  ⟨ID⟩ Load ⟨/ID⟩
  ⟨Desc⟩ Load a fresh part ⟨/Desc⟩
  ⟨SegmentReqmnt⟩
    ⟨ID⟩ load()(f) ⟨/ID⟩
    ⟨Desc⟩ First instance of the load type ⟨/Desc⟩
    ⟨SegmentParam⟩
      ⟨ID⟩ Integer ⟨/ID⟩
      ⟨Value⟩ 3 ⟨/Value⟩
      ⟨Desc⟩ Load from third shelf ⟨/Desc⟩
    ⟨/SegmentParam⟩
    ⟨MaterialReqmnt⟩
      ⟨MaterialUse⟩ Produced ⟨/MaterialUse⟩
      ⟨Quantity⟩ 1 ⟨/Quantity⟩
      ⟨MaterialReqmntProperty⟩
        ⟨ID⟩ f ⟨/ID⟩
        ⟨Value⟩ Fixture ⟨/Value⟩
      ⟨/MaterialReqmntProperty⟩
    ⟨/MaterialReqmnt⟩
  ⟨/SegmentReqmnt⟩
⟨/OperationsRequest⟩
```

Fig. 5. A B2MML operations-request for the *load*()(*f*) operation in the recipe in Figure 3.

the gripper to the object, grasping it, then moving it elsewhere, and releasing it). Each segment requirement can have a number of parameters that will be passed along to the equipment controllers (for example, variables specifying which gripper or drill bit to use, or the gripper separation or depth of cut required). The segment requirements can also include materials requirements, specifying the bill of materials for each step (which may also have parameters); and equipment requirements, specifying the bill of tooling for each step (again, these may have parameters). The segment parameters differ from the materials/equipment requirements in that the former are values to be sent to lower-level controllers, whereas the latter can include requirements that must be fulfilled before the operation can proceed, which could be verified by an external RFID sensing system, for example.

B2MML operations-requests are used to encode both the operations (both observable and internal) as well as the manipulation of parts as specified in the controller. Basically, there is an operations-request for each operation in the considered controller. The specifications shown in Figures 5 and 6 are respectively the task to load a fixture/pallet of parts into the system from a shelf (the pallet is identified as a materials requirement, and the shelf location as a segment parameter), and the task to separate a fixture that was previously loaded into its constituents. Note that in Figure 6, the MaterialUse tag is used in each material requirement in two ways: to both specify the inputs to the system, which ensures that the part is present before the operation begins, and also to specify the outputs of the operation, which together with the inputs ensure the correct part/material-flow for the finished product.

Once the individual operations have been defined as B2MML operations-requests, the controller is expressed as one or more B2MML operations-schedules. In each of these, operations-requests are included in the required sequence, with added equipment requirements to tie them to the appropriate manufacturing resources. Observe from Figure 4 that the correct sequence of operations to perform and the resources that are responsible for them are both specified in the controller. One controller may amount to multiple operations-schedules because the former could have choice points like the ones shown in the recipe of Figure 3; thus, one operations-schedule represents a single execution path through the controller.

```
⟨OperationsRequest⟩
  ⟨OperationsType⟩ Production ⟨/OperationsType⟩
  ⟨ID⟩ Separate ⟨/ID⟩
  ⟨Desc⟩ Separate pin and hinge from fixture ⟨/Desc⟩
  ⟨SegmentReqmnt⟩
    ⟨ID⟩ separate(f)(p,h) ⟨/ID⟩
    ⟨Desc⟩ First instance of the separate type ⟨/Desc⟩
    ⟨SegmentParam⟩
      ⟨ID⟩ String ⟨/ID⟩
      ⟨Value⟩ Gripper3 ⟨/Value⟩
      ⟨Desc⟩ Use Gripper3 from tool rack ⟨/Desc⟩
    ⟨/SegmentParam⟩
    ⟨MaterialReqmnt⟩
      ⟨MaterialUse⟩ Consumed ⟨/MaterialUse⟩
      ⟨Quantity⟩ 1 ⟨/Quantity⟩
      ⟨MaterialReqmntProperty⟩
        ⟨ID⟩ f ⟨/ID⟩
        ⟨Value⟩ Fixture ⟨/Value⟩
      ⟨/MaterialReqmntProperty⟩
    ⟨/MaterialReqmnt⟩
    ⟨MaterialReqmnt⟩
      ⟨MaterialUse⟩ Produced ⟨/MaterialUse⟩
      ⟨Quantity⟩ 1 ⟨/Quantity⟩
      ⟨MaterialReqmntProperty⟩
        ⟨ID⟩ p ⟨/ID⟩
        ⟨Value⟩ Pin ⟨/Value⟩
      ⟨/MaterialReqmntProperty⟩
    ⟨/MaterialReqmnt⟩
    ⟨MaterialReqmnt⟩
      ⟨MaterialUse⟩ Produced ⟨/MaterialUse⟩
      ⟨Quantity⟩ 1 ⟨/Quantity⟩
      ⟨MaterialReqmntProperty⟩
        ⟨ID⟩ h ⟨/ID⟩
        ⟨Value⟩ Hinge ⟨/Value⟩
      ⟨/MaterialReqmntProperty⟩
    ⟨/MaterialReqmnt⟩
  ⟨/SegmentReqmnt⟩
⟨/OperationsRequest⟩
```

Fig. 6. A B2MML operations-request for the $separate(f)(p,h)$ operation in the recipe in Figure 3.

5. Algorithms for Manufacturability and Control

This section presents algorithms describing how the manufacturability of a given production recipe Recipe on a precomputed production topology Topology is determined. The algorithms also briefly show how in answering the question of manufacturability, the solutions found (if any) for individual composite operations in the recipe are stored as part of a controller, which details how the available manufacturing resources should be orchestrated in order to manufacture the product. The algorithms in this section are adapted from the ones in [17] to use the EBNF data structures defined in Figure 2.

The top-level algorithm, Algorithm 1 works as follows. Given a state of the recipe RecipeState (e.g. the initial state of the recipe) and the group of resource states (i.e., a state of the topology) ResStates representing the state of the entire manufacturing facility at a given point in time, the algorithm determines, for each recipe transition of the form

(RecipeState, CompositeOp, RecipeStateTo),

whether the available resources can (hypothetically) execute CompositeOp from their corresponding states as indicated in ResStates. The latter is checked via Algorithm 2, which iterates over the groups of (parallel) manufacturing operations ResOps that are executable from ResStates by the various resources, to find a group that *can* perform the first step (of parallel operations) in CompositeOp. More specifically, each topology transition of the following form is considered:

(ResStates, ResOps, ResStatesTo),

Algorithm 1 CheckManuf

Input: Topology Topology and recipe Recipe;
Current state of recipe RecipeState;
Current states of all resources ResStates.
Output: Either *success* if Recipe is manufacturable or *fail* otherwise.

```
1: Let CurrentPlan be the empty sequence
2: for each transition RecipeTrans from RecipeState do
3:     result ← ProcessCompositeOp(ResStates,
                   CompositeOp, RecipeStateTo, CurrentPlan)
       // RecipeTrans contains CompositeOp and RecipeStateTo
4:     if result = fail then
5:         return fail
6:     end if
7: end for
8: return true
```

where the last element is the group of resource states that result from executing ResOps. Checking whether the first group of parallel operations in CompositeOp is executable by ResOps, amounts to checking whether the former can be *allocated* to ResOps. This holds if two conditions are met: *(i)* there is a one-to-one mapping from the names of the parallel recipe operations to the names of the parallel resource operations, and *(ii)* the parts required as input by the recipe operations are currently present in the resources that were mapped to those operations. If the allocation is successful, then ResParts2 is not *false*, and it contains the group of parts that will be produced by the recipe operations (e.g. the input parts with some modifications).

If ResOps indicates that parts need to be transferred between pairs of resources, i.e., it has pairs of matching *in* and *out* transfer operations (e.g. *in:3* and *out:3*), one part per pair is transferred from the resource performing the *out* operation to the one performing the *in*; the new locations of the parts in the manufacturing system are then referred to as ResParts2.

Next, the group of resource states ResStatesTo, i.e., the states that would result from performing the group of resource operations ResOps, is annotated with the new locations of parts, and ResOps is appended to the current plan CurrPlan that is being built (which is later stored as part of the controller, if the composite operation being considered does turn out to be executable). The algorithm is then called recursively in order to check the executability of the remaining steps in CompositeOp above, until there are no steps left to process.

If, on the other hand, ResOps contains only internal operations, such as transfer operations (to which it will not be possible to allocate the first step (group)—of *observable* parallel operations—in CompositeOp), the algorithm checks whether the first and subsequent steps in CompositeOp could instead be allocated after the internal operations complete, i.e., to a group of (parallel) resource operations that are possible from ResStatesTo. In order to ensure, however, that the algorithm does not follow this line of reasoning indefinitely, all considered groups of resource states (annotated with the parts that were present in the corresponding resources) are remembered and checked at each recursive call to avoid reconsidering them.

6. Conclusion

This paper has presented definitions in the EBNF notation for the notions of a production recipe, production resource,

Algorithm 2 ProcessCompositeOp

Input: Topology Topology and recipe Recipe;
Current states of all (part-annotated) resources ResStates;
Composite operation currently being processed CompositeOp;
The next recipe state to explore RecipeStateTo;
The current plan being composed CurrPlan.
Output: Either *success* if Recipe is manufacturable or *fail* otherwise.

```
1: if CompositeOp has been completely processed then
2:     Store CurrPlan in the controller for the initial CompositeOp
3:     result ← CheckManuf(RecipeStateTo, ResStates)
4:     return result
5: end if
6: if ResStates was already visited then
7:     return the empty result
8: end if
9: for each transition TopTrans from ResStates do                    //
   ResOps, ResStatesTo are in TopTrans, which is in Topology
10:    Let FirstOp be the first (possibly parallel) step of CompositeOp
11:    Set result to fail
12:    ResParts2 ← Allocate(FirstOp, ResOps, ResParts)
13:    if ResParts2 ≠ false then
14:        ResParts2 ← TransferParts(ResParts2, ResOps)
15:        ResStatesTo ← AnnotateParts(ResStatesTo, ResParts2)
16:        CurrPlan ← Append(ResOps, CurrPlan)
17:        Let Rest be the remaining steps of CompositeOp
18:        result ← ProcessCompositeOp(ResStatesTo,
                        Rest, RecipeStateTo, CurrPlan)
19:    end if
20:    if ResOps has no observable operations then
21:        ResParts2 ← TransferParts(ResParts, ResOps)
22:        ResStatesTo ← AnnotateParts(ResStatesTo, ResParts2)
23:        CurrPlan ← Append(ResOps, CurrPlan)
24:        result ← ProcessCompositeOp(ResStatesTo,
                        CompositeOp, RecipeStateTo, CurrPlan)
25:    end if
26:    if result = success then
27:        return result
28:    end if
29: end for
30: return fail
```

and production topology, based on the formalism introduced in [17]. Once a controller is synthesised for manufacturing a recipe on a topology, an approach for translating the controller into the B2MML format was provided, which is being increasingly widely adopted by the manufacturing industry. Finally, detailed algorithms were provided based on [17] for manipulating the data structures defined in EBNF. These algorithms can be used to both determine whether a given recipe is manufacturable on a precomputed topology, and in doing so to also incrementally build a valid controller for orchestrating the available resources in order to manufacture the product. The algorithms and data structures presented in this paper have been implemented as part of a software tool, which can synthesise manufacturing process controllers in B2MML from initial formal specifications of production recipes and resources [22].

Acknowledgements

The authors are grateful for the support provided by the Engineering and Physical Sciences Research Council via grants EP/K018205/1 and EP/K014161/1.

References

[1] Industrie 4.0 - Innovationen für die Produktion von morgen; 2014. Bundesministerium für Bildung und Forschung.
[2] Kagermann, H., Helbig, J., Hellinger, A., Wahlster, W.. Recommendations for Implementing the Strategic Initiative INDUSTRIE 4.0: Securing the Future of German Manufacturing Industry; Final Report of the Industrie 4.0 Working Group; 2013.
[3] Factories of the Future: Multi-annual roadmap for the contractual PPP under Horizon 2020. Tech. Rep.; EFFRA; 2013. URL: http://www.effra.eu/.
[4] Strategic Investment Plan. Tech. Rep.; Digital Manufacturing and Design Innovation Institute; 2015.
[5] Connected Manufacturing. Tech. Rep.; Industrial Value Chain Initiative; 2014.
[6] Koren, Y., Heisel, U., Jovane, F., Moriwaki, T., Pritschow, G., Ulsoy, G., et al. Reconfigurable Manufacturing Systems. CIRP Annals - Manufacturing Technology 1999;48(2):527–540.
[7] ElMaraghy, H.A.. Flexible and reconfigurable manufacturing systems paradigms. International Journal of Flexible Manufacturing Systems 2005;17(4):261–276.
[8] Onori, M., Semere, D., Lindberg, B.. Evolvable systems: an approach to self-X production. International Journal of Computer Integrated Manufacturing 2011;24(5):506–516.
[9] Antzoulatos, N., Castro, E., de Silva, L., Rocha, A.D., Ratchev, S., Barata, J.. A multi-agent framework for capability-based reconfiguration of industrial assembly systems. International Journal of Production Research (IJPR) 2016;:1–11.
[10] Leitão, P., Colombo, A.W., Restivo, F.. An approach to the formal specification of holonic control systems. In: Mařík, V., McFarlane, D., Valckenaers, P., editors. Proceedings of the International Conference on Industrial Applications of Holonic and Multi-Agent Systems (HoloMAS). 2003, p. 59–70.
[11] Babiceanu, R.F., Chen, F.F.. Development and Applications of Holonic Manufacturing Systems: A Survey. Journal of Intelligent Manufacturing 2006;17(1):111–131.
[12] Müller-Schloer, C., Schmeck, H., Ungerer, T., editors. Organic Computing - A Paradigm Shift for Complex Systems. Springer Basel; 2011.
[13] Onori, M., Barata, J., Frei, R.. Evolvable Assembly Systems Basic Principles. In: Information Technology For Balanced Manufacturing Systems; vol. 220 of *IFIP International Federation for Information Processing*. 2006, p. 317–328.
[14] Neves, P., Barata, J.. Evolvable production systems. In: IEEE International Symposium on Assembly and Manufacturing. 2009, p. 189–195.
[15] Chaplin, J., Bakker, O., de Silva, L., Sanderson, D., Kelly, E., Logan, B., et al. Evolvable Assembly Systems: A Distributed Architecture for Intelligent Manufacturing. IFAC-PapersOnLine 2015;48(3):2065–2070.
[16] Sanderson, D., Antzoulatos, N., Chaplin, J.C., Busquets, D., Pitt, J., German, C., et al. Advanced Manufacturing: An Industrial Application for Collective Adaptive Systems. In: IEEE International Conference on Self-Adaptive and Self-Organizing Systems Workshops. 2015, p. 61–67.
[17] de Silva, L., Felli, P., Chaplin, J.C., Logan, B., Sanderson, D., Ratchev, S.. Realisability of Production Recipes. In: European Conference on Artificial Intelligence (ECAI). 2016, p. 1449–1457.
[18] Business To Manufacturing Markup Language (B2MML); 2015 (accessed February 11, 2017). URL: https://isa-95.com/b2mml.
[19] Information technology - Syntactic metalanguage - Extended BNF; 1996 (accessed March 10, 2017). URL: http://standards.iso.org/ittf/PubliclyAvailableStandards/.
[20] Emerson, D., Kawamura, H., Matthews, W.. Plant-to-business (P2B) interoperability using the ISA-95 standard. Yokogawa Technical Report 2007;43:17.
[21] Scholten, B.. MES Guide for Executives: Why and how to Select, Implement, and Maintain a Manufacturing Execution System. International Society of Automation; 2009.
[22] de Silva, L., Felli, P., Chaplin, J.C., Logan, B., Sanderson, D., Ratchev, S.. Synthesising industry-standard manufacturing process controllers (demonstration). In: Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS). 2017, p. to appear.