The International Journal of Advanced Manufacturing Technology (2023) 124:1155–1168
https://doi.org/10.1007/s00170-022-10553-0

ORIGINAL ARTICLE

An integrated data- and capability-driven approach to the reconfiguration of agent-based production systems

Daniele Scrimieri[1] · Omar Adalat[1] · Shukri Afazov[2] · Svetan Ratchev[3]

Received: 21 September 2022 / Accepted: 15 November 2022 / Published online: 29 November 2022

Abstract
Industry 4.0 promotes highly automated mechanisms for setting up and operating flexible manufacturing systems, using distributed control and data-driven machine intelligence. This paper presents an approach to reconfiguring distributed production systems based on complex product requirements, combining the capabilities of the available production resources. A method for both checking the "realisability" of a product by matching required operations and capabilities, and adapting resources is introduced. The reconfiguration is handled by a multi-agent system, which reflects the distributed nature of the production system and provides an intelligent interface to the user. This is all integrated with a self-adaptation technique for learning how to improve the performance of the production system as part of a reconfiguration. This technique is based on a machine learning algorithm that generalises from past experience on adjustments. The mechanisms of the proposed approach have been evaluated on a distributed robotic manufacturing system, demonstrating their efficacy. Nevertheless, the approach is general and it can be applied to other scenarios.

Keywords Reconfiguration · Capabilities · Multi-agent systems · Machine learning · Assembly

1 Introduction

Modern manufacturing is characterised by a large number of product variants, decreasing product life cycles and shorter time to market. The flexibility of mass customisation is combined with the low unit cost typical of mass production [15]. In order to be competitive, manufacturers have designed reconfigurable systems that can adapt quickly to changing market demands and maintain the desired quality of service [21, 22].

Because of their complexity, advanced manufacturing systems are difficult to maintain and update, requiring highly skilled engineers and a considerable time and cost. To address this problem, self-adaptive systems have been long investigated. These systems are able to adjust their behaviour dynamically and autonomously, in order to cope with internal or external changes (e.g. dynamic workloads, component failures) [6, 7]. In addition to the manufacturing domain, self-adaptive behaviour has been studied in many other areas, including software engineering, distributed artificial intelligence, fault-tolerant systems, biologically inspired computing and control systems, with various software solutions [8].

This paper addresses the reconfiguration problem in highly flexible manufacturing systems, where multiple product variants can be produced. While some product variants are anticipated when the production system is initially set up, others can be introduced in the future as requirements evolve. A reconfigurable production system is a dynamic system that can be adapted to accommodate physical and logical changes and deal with unanticipated product variants and market demand. We present an approach to specifying

✉ Daniele Scrimieri
d.scrimieri@bradford.ac.uk

Omar Adalat
o.j.adalat@bradford.ac.uk

Shukri Afazov
shukri.afazov@ntu.ac.uk

Svetan Ratchev
svetan.ratchev@nottingham.ac.uk

[1] Department of Computer Science, University of Bradford, Bradford, BD7 1DP, UK

[2] Department of Engineering, Nottingham Trent University, Nottingham, NG11 8NS, UK

[3] Institute for Advanced Manufacturing, University of Nottingham, Nottingham, NG8 1BB, UK

product requirements and capabilities of a distributed production system, along with a method to match them. We consider, in particular, "plug and produce" systems, where individual production resources can be added or removed dynamically, prompting a system reconfiguration managed by a multi-agent system (MAS).

The MAS enables the reconfiguration of the production system when the requirements or the available capabilities change. The MAS consists of a number of intelligent agents distributed across the production system at various levels. A resource agent is associated with each physical resource that can individually offer some basic capability. As resources are added to the system, their capabilities are combined to form more complex ones. The MAS is aware of the full set of available capabilities and is capable of recognising if certain product requirements can be safely met by the production system. The MAS can both tell if the production system *can* produce a product and, if that is the case, *how* its resources can be organised to enable production.

Together with the MAS, a machine learning technique is introduced to provide the MAS with knowledge to adapt the production system as part of the reconfiguration process. This technique consists of capturing the changes made by system integrators, evaluating them in terms of performance and generating adaptation knowledge for future reconfigurations. Such adaptation knowledge is used by the MAS to propose the most suitable reconfiguration options when, e.g. multiple resources offer the same capabilities, or further adjustments are required to fully integrate the various resources and reach the desired production objectives. In this way, the MAS learns from humans and, at the same time, generate new knowledge that is utilised by humans, harmonising human-machine interaction.

The proposed framework was evaluated on a robotic assembly system, showing that it can effectively support a reconfiguration process. Our framework offers several advantages over similar approaches. It provides not only a method for modelling requirements and capabilities, but also procedures for aggregating capabilities and matching them against requirements. These procedures are implemented in a MAS for the reconfiguration of highly dynamic production systems (plug and produce), which do not have a predefined architecture. The hierarchical and distributed organisation of the MAS offers the required level of flexibility for such systems. The reconfiguration process leverages the self-adaptive behaviour of the production system, emerging from the knowledge acquired through learning from past adaptation experience. As a result, human intervention of system integrators is minimised, reducing reconfiguration time and cost. Although our framework can be applied to other production systems with different manufacturing processes or products, a more comprehensive evaluation is required. In some cases, the application of the framework may not be feasible because of, for example prohibitive logistics costs in geographically distributed production environments or incompatible information models in large supply chains.

Adaptation experience is also useful to analyse the performance at the system level and make decisions when there are multiple solutions meeting the production requirements. Any conflicting performance goals arising from different subsystems can be resolved through the MAS hierarchy using higher-level adaptation experience. Note that our approach does not involve agent-mediated auctions, which may be difficult to define for heterogeneous resources at multiple levels in dynamic reconfigurable systems.

2 Background and related work

A number of paradigms have been introduced to enable rapid system changes and offer high levels of flexibility, reconfigurability and adaptability in manufacturing systems. While flexible manufacturing systems provide a "built-in" flexibility to handle anticipated product variants and mid-volume, mid-variety production needs, reconfigurable manufacturing systems provide more scalability by using modular equipment that be changed or integrated to meet new production requirements [13]. In holonic and agent-based manufacturing systems, autonomous and collaborative entities, called respectively holons and agents, are used for distributed problem solving and intelligent control of manufacturing systems [23, 37]. Many of these systems are based on the PROSA reference architecture [36] and are implemented using the JADE agent platform [5]. Agent-based negotiation and auction protocols have been used to design mechanisms for coordination and resource allocation in, e.g. supply chain management [42], multi-project time-cost trade-off problems [24], virtual enterprises [38].

In the "manufacturing as a service" business model, manufacturing companies seek cost-effective production by outsourcing production processes. The benefits include not only an increased capacity, shorter lead times and more competitive prices for highly customised products, but also access to the latest technological innovations without the need to purchase equipment outright, which would become obsolete in a few years' time [16, 44]. Cloud manufacturing is a service-oriented manufacturing paradigm which employs cloud computing technology to offer customisable manufacturing services. Cloud manufacturing-as-a-service (CMaaS) platforms virtualise physical resources as services in the cloud [26], supporting decision-making tasks such as service composition [1, 43], selection and scheduling [39, 45]. Despite significant research efforts, there are still implementation challenges. For example, middleware

for decentralised architectures for connecting client and service providers and reducing their technological and intellectual burden is presented by [18]. The implementation of a cloud manufacturing system using Industrial Internet of Things technologies to tackle real-time data acquisition is discussed by [25].

Various government-backed strategic initiatives that promote the digitalisation of manufacturing share the same goals and design principles underlying these paradigms. These initiatives include the German Industries 4.0 [20], the European EFFRA Roadmap [14] and the US DMDII Strategic Plan 2018 [12]. Holonic manufacturing has had a significant influence on Industry 4.0 [11], in which decentralised decision-making is an integral part. Self-adaptive behaviour was investigated, for example in assembly systems as part of the Evolvable Assembly Systems project [7]. Adaptation of assembly systems, including plug-and-produce systems, is discussed by [3, 31–33]. Learning and adaptation during the ramp-up phase of assembly systems is analysed by [34]. A Function-Behaviour-Structure methodology for self-adaptive reconfigurable assembly systems is presented by [30]. Formal approaches and algorithms for checking whether a given product can be assembled with a certain class of resources and how to organise the selected resources are described by [9, 10].

A lot of work related to capability management is based on ontologies, focusing especially on the modelling aspects, but arguably not giving sufficient attention to the computational ones. An OWL-based manufacturing resource capability ontology that supports the inference of combined capabilities is presented by [19]. Ontology-based approaches to the reconfiguration of manufacturing systems are introduced by [2, 27]. The top-down and bottom-up searches of [27] share some similarities with our presentation. Problem decomposition in cooperative distributed problem solving is the focus of [35]. Production capabilities are also called skills in the literature. For example, taxonomies of sensorial skills for assembly lines are analysed by [17]. A definition of skills as an extension of the product-process-resource concept is presented by [29]. Perhaps the work that is the closest to ours is [3], where a multi-agent system for reconfiguring a plug-and-produce assembly system is presented. Although that approach provides a representation of capabilities and product specifications, it only touches upon the matching problem and does not address learning or adaptability problems.

3 Production capabilities and product specification

A *capability* is an abstract description of what a production resource can do, expressed as the goal of an operation. By *production resource*, or simply resource, we mean a physical resource that can be "plugged in" to the system as part of a reconfiguration process, such as a robot, an end effector, a pick-and-place module or an inspection device. A capability does not define *how* the production resource carries out the associated operation, but only *what* the operation achieves. For example, typically a robot has the capability to move an object. This can be expressed as MOVE(*point*), where MOVE is the operation and *point* is a parameter specifying a 3D or 2D point where the robot can move.

To expand this example, an assembly robot with a parallel gripper as end effector also has the following two additional capabilities: GRASP(*closingWidth*), to grasp a part by closing the gripper (setting the width of the gripper fingers to *closingWidth*), and RELEASE (*openingWidth*), to release the part by opening the gripper (setting the width of the gripper fingers to *openingWidth*). These two capabilities could also be associated with the gripper itself, in particular if the gripper is an individual resource that can be replaced or used in another robot.

Each parameter has an associated range of admissible values. For example, in the case of *point*, the range of admissible values depends on the workspace of the robot, i.e. the set of points that can be reached by the end effector of the robot. Parameters can also have the form of *compound terms*, that is they can be expressed as function of other parameters, values or other compound terms. For example, the capability of a robot that can move only along X and Y at a fixed $Z = z_0$ could be expressed as MOVE(*Point*(*X, Y,* z_0)). Figure 1 shows the grammar of the capability language in BNF form. A capability is represented by the nonterminal symbol ⟨capability⟩. Product specifications will also be defined in terms of capabilities.

To use our framework, a common taxonomy with a shared terminology for all the capabilities and product specifications is necessary. Otherwise, if the same capability was expressed in different ways, the various occurrences would not match, as it will be explained, and would be considered different. Although a shared terminology may not seem feasible in a large supply chain, there are approaches to facilitate its definition and use, utilising mind maps or concept maps, visual editors and language translators, such as the one presented by [32]. This allows non-expert users to benefit from a framework like ours using visual and textual representations closer to the natural language, hiding the implementation details on the actual knowledge representation. Another important aspect of capability representation languages that requires further attention is the balance between expressiveness and computational complexity. In general, the more expressive a language, the higher the computational complexity for reasoning using the language, as investigated, for example in the context of description logics [4].

⟨capability-definition⟩ ::= *capability-name* ":=" ⟨capability-list⟩
| *capability-name* ":" *capability-name* ":=" ⟨capability-list⟩
⟨capability-list⟩ ::= ⟨capability⟩
| ⟨capability⟩ "," ⟨capability-list⟩
⟨capability⟩ ::= *capability-name*
| *capability-name* "(" ⟨term-list⟩ ")"
⟨term-list⟩ ::= ⟨term⟩
| ⟨term⟩ "," ⟨term-list⟩
⟨term⟩ ::= *parameter*
| *value*
| *functor* "(" ⟨term-list⟩ ")"

⟨product-specification⟩ ::= *product-name* ":=" ⟨operation-list⟩
⟨operation-list⟩ ::= ⟨operation⟩
| ⟨operation⟩ "," ⟨operation-list⟩
⟨operation⟩ ::= *capability-name*
| *capability-name* "(" ⟨term-list⟩ ")"

Fig. 1 BNF grammar of the capability language. The nonterminal symbols are in angle brackets, whereas the terminal symbols (lexical tokens) are in quotes or italic

3.1 Complex capabilities

The capabilities of one or more resources can be combined to form *complex* capabilities. For example, combining the previous capabilities of the assembly robot, we can define the capability PICKANDPLACE (*point1, point2, closingWidth, openingWidth*) as the sequence MOVE(*point1*), GRASP(*closingWidth*), MOVE (*point2*) and RELEASE(*openingWidth*). This means that the robot can pick a part from *point1*, by closing the gripper to width *closingWidth*, and then place it to *point2*, by opening the gripper to width *openingWidth*. A complex capability is represented by ⟨capability-definition⟩ in the grammar. The symbol ":=" is used for specifying a name for a complex capability. It is not possible to define complex capabilities recursively. Therefore, the name of a complex capability being defined cannot occur on the right side of ":=". Capabilities that are not defined as composition of other capabilities are called *atomic*. Each resource of a production system has a set of atomic capabilities.

An atomic capability can be implemented in different ways. For example, GRASP could be implemented differently by different robots and even the same robot could have different implementations if multiple types of grippers were available. A production system can have multiple resources with the same capabilities or with some capabilities in common. For example, in an assembly system there could be two identical robots performing simultaneously the same assembly steps on different parts, therefore requiring the same capabilities. Also, there could be two different robots performing different steps on the same part, still using some shared capabilities.

3.2 Hierarchical relationships

The capability PICKANDPLACE that we defined relies on the use of a parallel gripper and, as a result, it has the parameters *closingWidth* and *openingWidth*. However, other types of grippers could be used as well to carry out a pick-and-place operation, for example a magnetic or a vacuum gripper. For this reason, this definition of PICKANDPLACE may be too restrictive. The capabilities of a magnetic or vacuum gripper do not specify the parameters *closingWidth* and *openingWidth*. For example, assuming that no other parameter is required, the capabilities of a magnetic gripper could simply be expressed as MAGNETICGRIPPERGRASP() and MAGNETICGRIPPERRELEASE().

To overcome this problem on the generality of capabilities, hierarchical relationships between capabilities can be defined, similarly to classes in object-oriented programming languages. Let us rename the capabilities of a parallel gripper GRASP as PARALLELGRIPPERGRASP and RELEASE as PARALLELGRIPPERRELEASE. The capabilities PARALLELGRIPPERGRASP and MAGNETICGRIPPERGRASP can be defined as capabilities *derived* from a more general capability GRASP. This means that a resource that offers either the capability PARALLELGRIPPERGRASP or MAGNETICGRIPPERGRASP, also offers the capability GRASP. In other words, if a resource can grasp an object with a parallel or magnetic gripper, then we can say it can grasp an object and we can use it to assemble a product that requires the GRASP capability. Similarly, PARALLELGRIPPERRELEASE and MAGNETICGRIPPERRELEASE can be derived from RELEASE. The hierarchical relationship between two capabilities is indicated using the

":" symbol. The notation A : B means that A derives from B.

Based on these considerations, a more general definition of PICKANDPLACE would specify PICKANDPLACE(*point1, point2*) as the sequence of capabilities MOVE(*point1*), GRASP(), MOVE(*point2*) and RELEASE(). Such definition has the advantage that it does not impose the use of a particular type of gripper, if that is not necessary.

3.3 Product specification

A *product specification* consists of the sequence of operations required to manufacture a product, including quality inspection. These operations are defined in terms of capabilities and can specify a value for the parameters of the capabilities. For example, an operation that uses the aforementioned capability PICKANDPLACE to move a part from the point (10, 10, 5) to the point (60, 15, 5) using a closing gripper width of 3 cm and an opening gripper width of 5 cm is PICKANDPLACE((10, 10, 5), (60, 15, 5), 3, 5). If the value of a certain capability parameter is irrelevant for a product operation, or can be determined by the implementation, a variable can be used instead. A product specification is represented by ⟨product-specification⟩ in the grammar.

4 Multi-agent system

The reconfiguration of a production system is handled by a MAS. Given a product specification, the MAS is able to determine if, with the resources currently available, the production system *can* produce that product and, if that is the case, *how* the resources can be organised to enable production. The MAS includes the following three types of agents:

PA (Production Agent) - An agent of this type manages the configuration process of a (sub)-system based on the product specification and the available resources.

CA (Capability Agent) - An agent of this type manages the capabilities of a (sub)-system. This agent determines if a certain complex capability can be realised using the capabilities available in the (sub)-system.

RA (Resource Agent) - An agent of this type represents a production resource and is the interface between the MAS and the control system of the resource. When the capabilities of a resource are required for the manufacture of a product, the associated RA configures the resource accordingly by setting up the appropriate parameters or control logic.

DA (Deployment Agent) - An agent of this type deploys a RA when the associated resource is plugged into the system.

The MAS has a hierarchical structure. A PA is the root of the tree; CAs, DAs and RAs are the leaves, and other PAs are intermediate nodes. For example, Fig. 2 illustrates the structure of the MAS used in the experimental evaluation. The PA that is the root of the tree is responsible for the manufacture of the product using the entire MAS and production system. A PA that is an intermediate node is responsible for the manufacture (or inspection) of a subassembly using the agents of the MAS that are underneath it in the tree and the associated production resources. These resources represent a *subsystem* of the full production system, that is a part of the system that can accomplish one or more production tasks independently of the rest of the system, using its own capabilities.

4.1 Capability management

Each PA has an associated CA, which is aware of the capabilities of the (sub)system controlled by the PA. The CAs associated with PAs that control directly RAs (i.e. without intermediate PAs) combine the atomic capabilities provided by the RAs to create complex capabilities. The CAs above these CAs in the hierarchy combine these complex capabilities, with possibly other atomic capabilities, to create further complex capabilities, and so on up to the root. The CA associated with the PA of the root node is aware of the full capability set of the entire production system. This is the *bottom-up* approach to combining capabilities, which starts from atomic capabilities and creates more and more complex capabilities until the full set is built.

Another one is the *top-down* approach, which starts from the product specification. For each operation, starting from the root, PAs are queried recursively in a depth-first search to find out if any of them can offer a (complex) capability that matches the operation. When a PA is queried, the following cases can be distinguished:

1. The required capability can be offered directly by the PA by combining the atomic capabilities of its child RAs;
2. The required capability cannot be created by combining the atomic capabilities of the PA's child RAs.
   (a) If there is one or more child PAs, then these are queried recursively.
   (b) If there is no child PA, then the required capability cannot be offered by the PA.

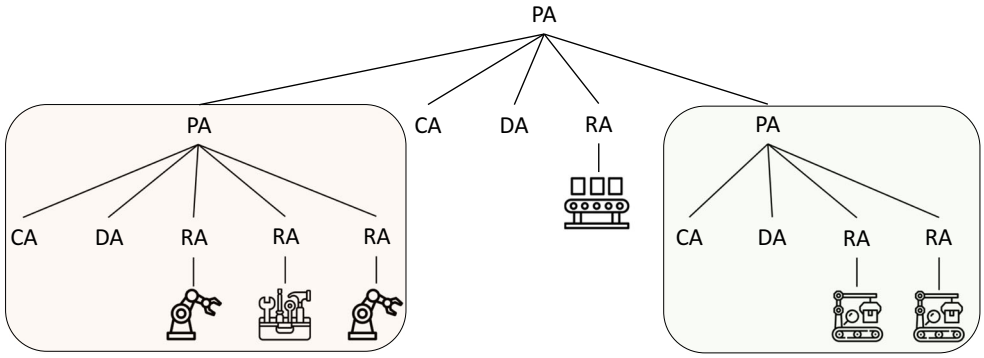

Fig. 2 The structure of the MAS used in the experimental evaluation. There are two levels of PAs: the two PAs below the root PA control the two subsystems below them (in coloured boxes). The resources of the subsystem on the left are two robots and a tool changing rack. The resources of the subsystem on the right are two inspection devices. The resource immediately below the root PA is a transport system to move parts between the two subsystems

On the one hand, the bottom-up approach has the advantage that all the complex capabilities are built from the atomic ones as soon as a physical change takes place, that is a resource is plugged in or out. However, the generated capabilities will not necessarily all be used to implement a product specification. On the other hand, the top-down approach has the advantage that only a subset of all the available capabilities is actually created, based on what is required to implement a product specification. New capabilities built this way can be stored to speed up future searches. The downside of this approach is that PAs may be queried multiple times for different operations.

If the same capability is offered by multiple PAs or RAs, we do not differentiate its multiple occurrences at this stage. Therefore, in a top-down search, once a PA that can offer a capability that matches the operation of the product specification being considered is found, the search can stop. In Section 6, we will see how to select among multiple options using performance estimates and experience-based learning.

This structured approach to capability management and production is very general. Nonetheless, note that production systems rarely need a MAS with more than 2 or 3 levels of PAs and CAs. Also, note that the MAS structure is dynamic, since resources can be added or removed and the corresponding RAs created or destroyed. Upon deployment, a RA informs its PA parent of its capabilities, which, in turn, informs the associated CA. When a resource is removed, the set of available capabilities must be updated.

The choice between the bottom-up and top-down approaches depends on the number of capabilities that can be potentially generated and the dynamics of the MAS. As a rule of thumb, if the atomic capabilities can lead to a large number of complex capabilities or if the MAS is very dynamic, then the top-down approach can be more efficient. Otherwise, if there are not many complex capabilities and the MAS structure does not change very often, then the bottom-up approach may be the best option.

4.2 User interaction

As part of a reconfiguration process, the user selects through a human-machine interface a product variant to be manufactured. The product specification of the selected product variant is sent to the root PA, who determines if the product specification can be realised with the resources currently available in the production system. The MAS is not visible to the user, who, therefore, does not need to know about its inner workings. See Fig. 3 for the operations and actors involved when making a physical change and reconfiguring a system.

The user is informed about the outcome of the realisability check performed by the MAS. If the product specification can be realised, its operations are allocated to the resources that offer the corresponding capabilities and production can begin. Otherwise, in case of failure, the user is informed about which operations of the product specification cannot be implemented and why. The user is presented with the missing capabilities, either atomic or complex. For complex capabilities, the missing constituent capabilities can be indicated. However, note that a complex capability can, in general, be created in a number of different ways, using different sets of capabilities. This is a recursive process that ends when all the missing atomic capabilities are provided.

If the realisability check is not successful, based on past experience (see Section 6), the MAS can suggest to the user which resources can be plugged in in order to provide the missing capabilities and complete the reconfiguration. The MAS keeps a history of all the resources that have been used and their capabilities. Therefore, if a missing capability was provided in the past by a certain resource, the MAS can suggest plugging in that resource.

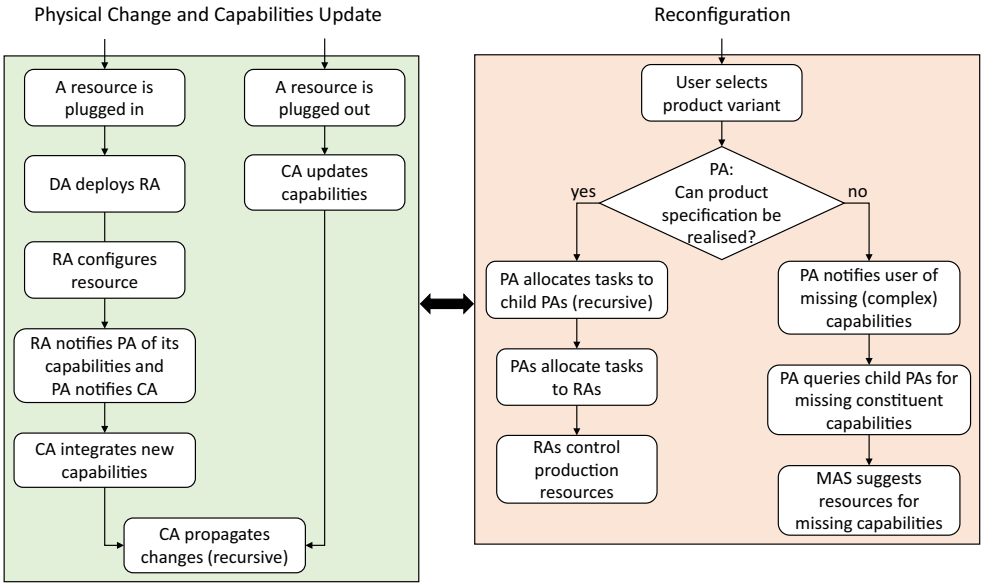

Fig. 3 Sequence of operations performed by the MAS and the user when making a physical change and reconfiguring the system. Capabilities are updated following the bottom-up approach

5 Matching product specifications and production capabilities

The root PA initiates the process of finding the capabilities required for a product specification. The root PA queries its associated CA for the capabilities available at the top level (using either the bottom-up or top-down approaches) and tries to match these against the operations of the product specification. This matching process is similar to "unification" in logic and consists of checking if the name of an operation is the same as the name of a capability, if the number and type of the parameters of the operation and capability are the same and if the parameters match. Parameters of capabilities match explicit values in the product specification if their respective types are the same and the values are within the ranges of the respective parameters. This process is repeated for all the operations part of the product specification.

The algorithm for matching an operation in a product specification and a capability is illustrated in Fig. 4. The output of the algorithm (variable S) is the "most general unifier", i.e. the most general set of variable substitutions that "unify" the operation in the product specification and the capability, if this is possible, or $\perp$ otherwise. In the algorithm, a variable is a placeholder for a term. A substitution of a variable x with a term y is denoted by x/y and means "replace x with y" where it is applied. For example, the substitution $closingWidth/3$ applied to GRASP(*closingWidth*) produces GRASP(3).

Terms to be unified are written in the form $x \doteq y$ (potential equations). The algorithm starts with the potential equation $p \doteq c$ in E, where p is a production operation and c is a capability, and the empty set of substitutions S. Equations can be added to or removed from E. The algorithm terminates when either there is no equation to be unified or there is an equation that cannot be unified. For example, MOVE(*Point*(10, 10, Z_1)) and MOVE(*Point*(*X, Y, Z*)) can be unified by $\{X = 10, Y = 10, Z = Z_1\}$, whereas MOVE(*Point*(10, 10, 5)) and MOVE(*Point*(*X, Y,* 0)) cannot be unified because $5 \neq 0$.

When the conditions "x is a variable" or "y is a variable" are true the terms x or, respectively, y must not be contained in the other term of the equation, since a variable cannot be unified with a term that contains it. This *occur check* is necessary for preventing infinite loops. For the sake of simplicity, the check on the range is not illustrated.

6 Experience-based learning and adaptation

There may be multiple ways in which a product specification can be realised in a production system if the same capability is offered by multiple resources or subsystems. The MAS is capable of identifying the best solution by learning from previous experience in similar scenarios. For example, the capability PARALLELGRIPPERGRASP(*closingWidth*) could be offered by two resources using two different parallel grippers. Regardless of which gripper is used, the capability is exactly the same. Therefore, either resource can be used to implement a product operation requiring that capability. However, the data acquired in similar scenarios may indicate that one of the two grippers is more reliable than

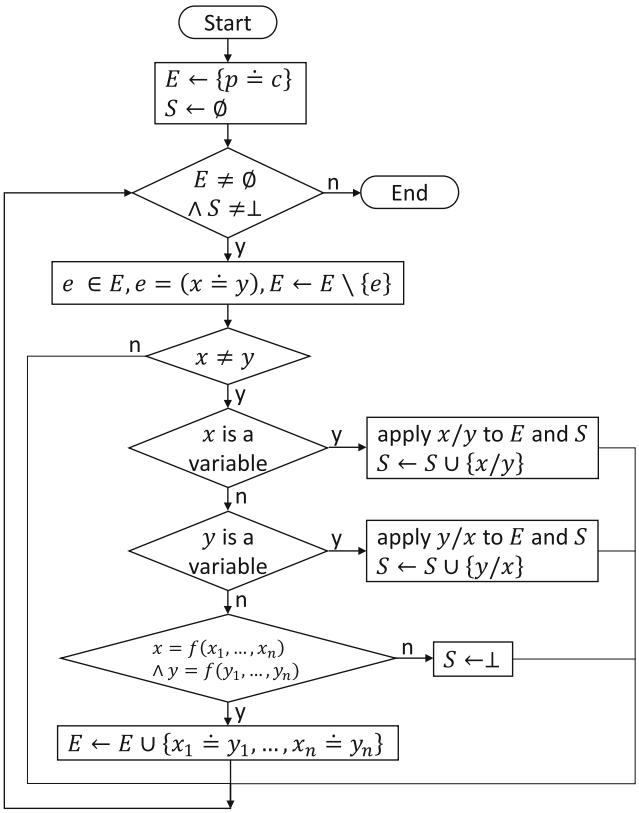

Fig. 4 Algorithm for matching an operation p in a product specification and a capability c. The output is the most general unifier of p and c if it exists, $\perp$ otherwise

the other, for example, in terms of accurate or repeatable positioning.

In addition, as part of a reconfiguration, a series of adjustments may be required to fully integrate the various resources and reach the desired production objectives in terms of quantity and quality. Plug-and-produce systems can help to solve this problem by applying logical adaptations autonomously with regard to, in particular, routing or scheduling policies. However, human intervention may still be required to perform mechanical changes that cannot be made via software or to make complex decisions.

Machine learning can be used to analyse all these changes and decisions made by the MAS or the user, and identify which ones improved the performance of the production system the most. Such knowledge on adaptations will then enable the production system to make more informed decisions autonomously or recommend changes to the system integrator. Adaptation knowledge can be generated and applied at either the system or subsystem level. The system integrator is able to query the learning system for the best changes that can be applied in a certain context.

An approach to enhancing automatic adaptations using system integrators' knowledge consists of automatically capturing and evaluating manual changes, and creating an *experience base* that can then be used to generalise the effect of individual adjustments and learn which changes are useful. In future reconfigurations, when a resource for which experience has been captured is plugged in, the system will be able to recommend to the user which changes to make, or even apply them autonomously.

This approach is in contrast to representing explicitly such adaptation knowledge in a knowledge base. The represented knowledge, for example could be in the form of rules specifying which gripper to use and for which part, when more than one option is available, or what process parameters to set, when not specified in the product specification. The advantage of this approach over a manual construction of a knowledge base is twofold. First, the adaptation knowledge is built dynamically from the experience base, whenever needed, so the user does not have to represent it explicitly or even keep it up-to-date. Second, the adaptation knowledge can be generalised and applied to new scenarios that have not been encountered before.

The following subsections describe how experience is captured and used to create adaptation knowledge. A technique to rank and propose adaptations is presented.

6.1 Experience capture

We call *experience* the representation and evaluation of changes made to the production system. Changes can be viewed at various levels of granularity, from atomic changes consisting of a single operation and affecting only one process parameter, to composite changes involving multiple parameters or operations. We refer to an atomic change as an *adjustment*. An adjustment has the form (t, v) where t is the type of adjustment (e.g. change of gripper type, change of closing angle) and v is the value of the adjustment (e.g. new gripper type, new closing angle). An *experience instance* is the representation of an adjustment and its effect.

Experience can be captured at different levels in the production system: for the whole system, for various subsystems or for a combination of system and subsystems. For each (sub)system on which experience is captured, a *performance function* is defined. A performance function is typically a function of subassembly quality or throughput and is used to evaluate the effect of changes on the (sub)system. The effect of an adjustment must be measurable by the adopted performance function. Note that, in general, subsystems and their associate agents can exhibit conflicting behaviours, in the sense that a performance increase in one subsystem may cause a performance decrease in another one. Although not investigated in this paper, it would be interesting to analyse these relationships in some typical scenarios and how they contribute to the overall system performance. While it can be useful to initially focus on

individual subsystems during an adaptation and use *local* experience with the aim of increasing both subsystem and overall system performance, performance is ultimately evaluated on the entire production system using *global* experience. From now on, by "machine" we will refer to either the entire production system or the subsystem on which experience is captured.

Machine states are represented by configuration or process parameters and state information (e.g. cycle times, inspection results) that characterise the state of the machine at a certain point in time. We call all these values *attributes*. We denote the set of attributes of the machine by $A = \{a_1, \ldots, a_n\}$ and the domain of attribute a_i by D^i, for $1 \leq i \leq n$. A machine state s is a function

$$s : \mathrm{A} \rightarrow \bigcup_{i=1}^{n} D^i \cup \{null\} \tag{1}$$

that maps attributes to their corresponding values, such that:

$$s(a_i) = \begin{cases} d \in D^i & \text{if attribute } a_i \text{ is present in state } s \\ null & \text{otherwise} \end{cases} \tag{2}$$

When a new resource is plugged in, its attributes are added to the machine state. When a resource is plugged out, its attributes are removed from the machine state, so s will be equal to $null$ for these attributes. A performance function f is a function of machine states $f : S \rightarrow [0, 1]$, where S is the set of machine states, 1 is the maximum performance and 0 is the minimum.

Sensor data generated by production resources is collected by the associated RAs to derive state information. This is embedded into *state events* that are distributed to PAs up the MAS hierarchy. Adjustments are notified through a different type of events, called *adjustment events*, which trigger the experience capture process. A "publish-subscribe" asynchronous communication mechanism is employed, whereby RAs *publish* sensor data, i.e. make it available, and PAs interested in it *subscribe* to it in order to receive it. One or more PAs up the hierarchy can subscribe to the events generated by the RAs below them, depending on the levels at which experience and knowledge are generated.

An experience instance has the form (s_1, a, s_2), where:

- s_1 is the machine state before a
- a is an adjustment
- s_2 is the machine state after a

The state events received by a PA are stored in an *event base*. When an adjustment event is received, the PA creates a representation of the machine state before and after the adjustment by querying the event base for the state events generated just before and after that adjustment. The performance function is calculated on both these states. If the value of the performance function for s_2 is higher than that for s_1, it means that a had a positive impact on the performance in machine state s_1.

6.2 Learning and adaptation

When adapting a machine, the experience acquired is used to identify the best adjustment to perform in a certain machine state. If we look at the best adjustment in a machine state as the "class" of the machine state, this problem can be expressed as a *classification problem* in machine learning. A program (*classifier*) is built to classify instances based on a given set of examples (*training set*). Each example contains an input vector and a discrete output value that indicates the class of the example. The classifier is able to predict the correct class of instances that are not in the training set. There exist a number of classification techniques, such as artificial neural networks, instance-based learning, support vector machines and decision trees (see, for example [28]).

In our problem, a machine state in which the user queries the system for the adjustment to apply represents an instance to be classified. The adjustment type corresponds to the class of the instance. The initial states and adjustment types of the experience instances of an experience base represent the training set. Throughout an adaptation process, the machine goes through various states which must be classified, that is in which the best adjustment type must be found. The associated adjustment value is also determined by our framework.

6.2.1 Experience base search

Our learning technique is based on a variant of the *K-nearest neighbour* algorithm (kNN). This algorithm searches for the k instances in the training set that are nearest to a given instance to be classified, using a certain distance measure. This measure models the similarity between two instances, in the sense that, the smaller the distance between them is, the more similar they are supposed to be. The class to be assigned to an instance is either the most common among the k neighbours or is chosen by adopting some voting scheme. In our technique, the search occurs in the experience base and, in addition to similarity between machine states, it uses performance estimates given by the performance function calculated on the experience.

6.2.2 Distance function

Machine states can be seen as points in a multidimensional space. The definition of the distance function between machine states s_1 and s_2 must handle both numerical and

categorical attributes. Therefore, we use the *heterogeneous euclidean-overlap metric* [41]:

$$d(s_1, s_2) = \sqrt{\sum_{i=1}^{n} (d_i(s_1, s_2))^2}, \tag{3}$$

where $d_i(s_1, s_2)$, for $1 \leq i \leq n$, is the distance between s_1 and s_2 on attribute a_i:

$$d_i(s_1, s_2) = \begin{cases} 1 & s_1(a_i) = null \text{ or} \\ & s_2(a_i) = null, \\ \text{overlap}(s_1(a_i), s_2(a_i)) & a_i \text{ is nominal,} \\ \text{rndiff}_i(s_1(a_i), s_2(a_i)) & \text{otherwise.} \end{cases} \tag{4}$$

The function overlap is defined as 0 if its arguments are the same, 1 otherwise. The function rndiff_i (range normalised difference) is defined as:

$$\text{rndiff}_i(x, y) = \frac{|x - y|}{\max(a_i) - \min(a_i)}, \tag{5}$$

where $\max(a_i)$ and $\min(a_i)$ are, respectively, the maximum and minimum values of a_i observed in the training set.

As defined, the distance function is calculated on all attributes of the two states. However, not all the attributes have the same relevance and some may even be redundant. Also, another potential issue is given by noisy attributes. Various weight-setting methods are available to cope with these issues [40].

6.2.3 Similarity-performance function

In order to find the best adjustment in a certain machine state, we need to search the experience base for the experience instance with the most similar initial machine state and the best adjustment, in terms of performance value of the resulting machine state. Therefore, we combine the distance function with an estimate of the resulting performance based on the acquired experience.

Let $e = (s_1, a, s_2)$ be an experience instance, s be a machine state and f be a performance function. We define the *similarity-performance function* $sp_f(e, s)$ as follows, for $f(s_2) \neq 0$ [34]:

$$sp_f(e, s) = \frac{d(s_1, s)}{f(s_2)} \tag{6}$$

If $f(s_2) = 0$, $sp_f(e, s)$ is defined to be infinitely large. The smaller the value of this function is, the better the adjustment of the experience is expected to be in the state being examined.

6.2.4 Voting and ranking

The similarity-performance function is used to rank adjustments and, when $k > 1$, to define a voting scheme. For each adjustment type, an adjustment value is also determined. The voting scheme (Fig. 5) ranks the adjustment types among the k neighbours and calculates the adjustment value associated with each of them.

Let f be a performance function and s be a machine state to be classified. There are two cases to be considered:

$k = 1$ A ranked list of adjustments for s is generated by sorting the experience instances by $sp_f(e, s)$ in ascending order. The experience instance with the smallest value of $sp_f(e, s)$ contains the the recommended adjustment.

$k > 1$ A class is selected among those of the k neighbours using the voting scheme of Fig. 5. Selecting the most common class is not a good approach because this class tends to be the most frequent in the training set. A typical approach is to assign a weight to each neighbour, given

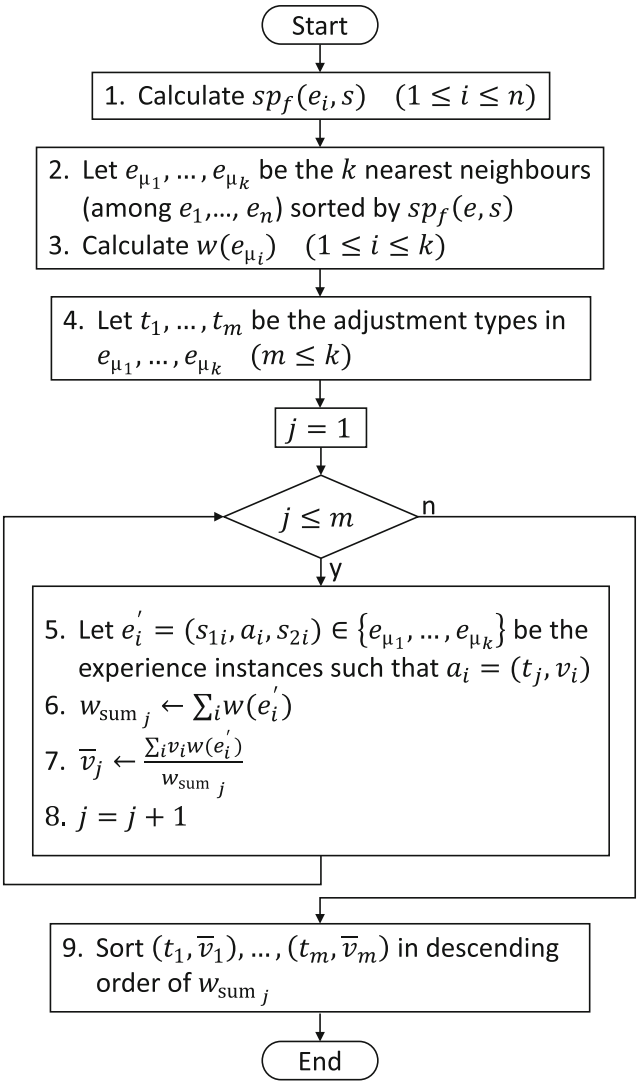

Fig. 5 Algorithm for the voting scheme for ranking adjustments when $k > 1$. The input is experience instances $e_1, \ldots, e_n$ $(k \leq n)$, a machine state s, a performance function f. The output is a ranked list of adjustments (t_i, v_i) $(i \leq m \leq k)$

by the distance of the neighbour from the machine state to classify. In our problem, we use sp_f to calculate the weights (line 1 in Fig. 5), as we want to include the effect of the corresponding adjustment on system performance. Let $e_1, \ldots, e_k$ be the k experience instances with the smallest values of sp_f (line 2). The weight $w(e_i)$ assigned to e_i is defined as:

$$w(e_i) = \frac{sp_f(e_k, s) - sp_f(e_i, s)}{sp_f(e_k, s) - sp_f(e_1, s)} \quad (7)$$

These are calculated in line 3. The instances $e_1, \ldots, e_k$ are grouped by class (line 4) and the weights of the instances of each class are summed up (lines 5–6). The adjustment types are ranked by the sum of their weights, in descending order (line 9). The recommended adjustment type t is the class with the largest sum of weights. The adjustment value for t is given by the weighted mean (using $w(e_i)$ as weights) of the adjustments values of the instances among $e_1, \ldots, e_k$ having adjustment type t (line 7).

7 Experimental evaluation

The proposed framework was evaluated on a reconfigurable manufacturing system for assembling detent hinges used in the cabin of trucks (Fig. 6). The production system includes two 6-axis robots, a tool changing rack for holding grippers, an inspection station and a shuttle system for loading/unloading the parts and taking them to the robots and inspection station. The detent hinge consists of two hinge leaves; up to three balls and springs that are placed in a cylindrical slot inside the leaves; a metal pin to lock the leaves and a retainer to keep the balls and springs inside the slot. The production system can be reconfigured for making different product variants with, e.g. one, two or three balls and springs, depending on the desired detent force. Both robots can pick tools from the rack, which is located in a shared workspace. The tools include vacuum suction and two-finger grippers of various sizes. The inspection station performs a visual check of the hinge to verify its integrity and a mechanical check of the detent force.

Fig. 6 Reconfigurable production system for assembling detent hinges

The control architecture is distributed and the multi-agent system reflects this structure. Each production resource (robot, shuttle system, inspection station) is controlled by a PLC. In case of failure of one of the two robots, the production system can be reconfigured to allocate all the assembly tasks to the other robot. Similarly, if an additional robot was added to the system, the assembly tasks could be redistributed among the three robots. The atomic capabilities of the production system include grasping and realising capabilities for the various vacuum suction and two-finger grippers, "move to" capabilities for the robots and the shuttle system, a visual check and a detent force check for the inspection station. Although the same grasping capability can be offered by multiple tools, some tools are more suitable than others for certain tasks. The experience-based learning technique can help choose the most suitable tool.

The experiment was aimed at evaluating how the proposed framework can support engineers in the reconfiguration process. The production system was reconfigured to manufacture a new type of hinge with some geometrical differences (dimensions and tolerances) and a different detent force. Note that this new hinge type was not one of the existing product variants, but a new product. The same types of assembly operations were required and, therefore, the atomic capabilities were the same. The same production resources were also used, in particular the same set of tools. However, the tools were not initially loaded on the tool changing rack, so that the realisability check of the MAS could be fully tested. The reconfiguration involved selecting the appropriate tools and control logic programs for the assembly operations, as well as setting parameters such as robot speed, gripper pressure and pick-and-place locations for the parts. Six engineers with comparable skills and knowledge of the production system participated in the experiment. The engineers were divided into two groups of three each: one group (group 1) reconfigured the production system without the support of the framework, whereas the other one (group 2) reconfigured the production system with the support of the framework.

The production system was first reconfigured by each of the engineers of group 1, restoring the initial configuration of the system for making the original product after each engineer completed the process. All the changes made by all the engineers of group 1 were recorded in an experience base. The production system was then reconfigured by each

of the engineers of group 2, using the MAS and the changes recommended by the automated learning system based on the experience base built by group 1. Similarly to group 1, the initial configuration of the production system for making the original product was restored after each engineer completed the process.

The performance of the production system was measured using a performance function representing the production quality in terms of number of good parts in the last 10 parts assembled. The target value for the performance function was 95%. The reconfiguration processes carried out by the engineers finished when this value was reached. In order to evaluate the effect of a change on the performance, after each change at least 5 parts were assembled before examining the up-to-date value of the performance function and making new changes. The same performance function was used for calculating the similarity-performance function for ranking the adjustments of the experience instances captured in the experience base built by group 1. Figure 7 shows the average value of the performance function obtained by the two groups until the target performance value was reached. Step 0 indicates the start of the process, when no changes have been made and no parts have been produced yet. Groups 1 and 2 reached the target value, on average, in 18 and 13 steps, respectively. Figure 8 shows the number of changes applied by the engineers of group 2 divided in changes recommended by the framework that were ranked first, other recommended changes (not first-ranked) and any other changes not ranked by the framework.

The evaluation indicates that the engineers of group 2 could reach the target value of the performance function in fewer steps than those of group 1. Since each of the various steps took a comparable time, this also means that group 2 could reach the target value sooner. In general,

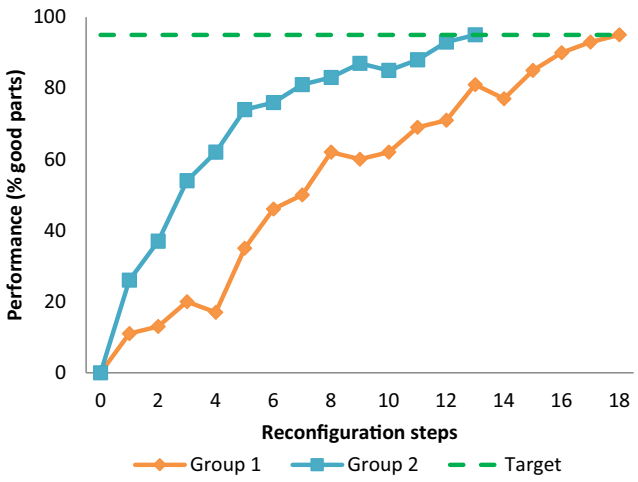

Fig. 7 Average values of the performance function obtained by the two groups of engineers during the reconfiguration experiment

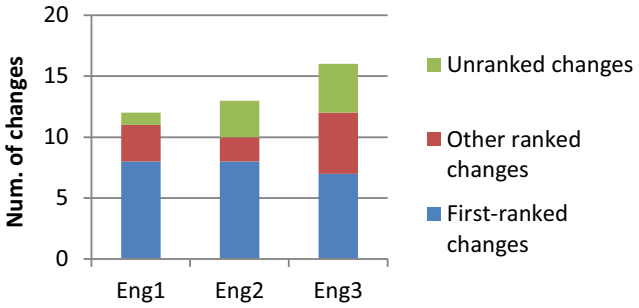

Fig. 8 Types of changes applied by the engineers that used the framework (group 2)

other reconfiguration processes or systems may require a more specific evaluation of reconfiguration time. The performance increase of group 2 is faster and seems more steady than group 1, especially at the beginning of the process. This indicates that at the beginning, when no tool was available and the realisability check of the MAS failed, the MAS could identify and recommend to the user suitable tools based on both the required capabilities and the acquired experience. Figure 8 shows that the engineers trusted the recommendations of the framework in the majority of the cases and they also applied the first-ranked change in the majority of these cases. When the engineers did not apply any recommended change, they did so because they thought that those changes were not relevant or beneficial, or they could get better results by applying a different change.

8 Conclusion

Reconfigurable systems share the design principles advocated by the current technological trends in the manufacturing industry and can offer the flexibility required to cope with global market demands. The distributed and adaptive nature of agent-based systems makes them particularly suitable for the software implementation of reconfigurable systems that have a dynamic architecture not predictable at design-time. This paper analysed reconfiguration and adaptation problems, and presented an integrated data- and capability-driven approach to address them effectively using agents. The self-adaptive behaviour of the MAS is the result of learning from past experience and is very useful for making autonomous decision or recommendations to the user when they have no prior knowledge.

Methods to represent and match product specifications and production capabilities were introduced for determining (1) whether a production system is capable of manufacturing a certain product in a particular configuration, (2) how to reconfigure the resources to enable production and (3) how to adapt the resources in the new configuration

to achieve the best performance. Although product specifications and capabilities can be quite complex, the representation language offers simple, yet powerful mechanisms to define them. This language is general and can be used for a wide variety of production systems. The procedures to aggregate capabilities and match them against product requirements are also general, since they do not depend on the specific product or production system, and can be easily implemented as part of the MAS. However, a more rigorous analysis of the balance between expressiveness and computational complexity of the language would be useful.

The experimental evaluation conducted on a reconfigurable assembly system indicates that the proposed framework can successfully support engineers in accelerating a reconfiguration process. Future research consists of applying the framework to different industrial test cases, manufacturing processes and products by fully characterising all its components (i.e. capabilities, products, adaptation experience and performance of the production system) and evaluating the effectiveness of the approach on a larger scale. In particular, this includes an analysis of the time and cost savings over manual reconfiguration and adaptation processes. In some cases, the application of the framework may not be cost-effective (e.g. geographically distributed production environments due to logistics costs, large supply chains with incompatible information models. In addition, it would be useful to investigate the impact of local experience and conflicting subsystem behaviours on overall system performance.

Author contribution D. S.: funding acquisition, investigation, methodology, implementation, experimentation, writing. O. A.: investigation, methodology, implementation, writing. S. A.: methodology, review, writing. S. R.: funding acquisition, methodology, supervision.

Funding This work was supported by the SURE Research Projects Fund of the University of Bradford and the European Commission (grant agreement no. 314762).

Data Availability No supplementary data or materials available.

Declarations

Competing interests The authors declare no competing interests.

References

1. Aghamohammadzadeh E, Valilai OF (2020) A novel cloud manufacturing service composition platform enabled by blockchain technology. Int J Prod Res 58(17):5280–5298
2. Alsafi Y, Vyatkin V (2010) Ontology-based reconfiguration agent for intelligent mechatronic systems in flexible manufacturing. Robot Comput Integr Manuf 26(4):381–391
3. Antzoulatos N, Castro E, de Silva L, Rocha AD, Ratchev S, Barata J (2017) A multi-agent framework for capability-based reconfiguration of industrial assembly systems. Int J Prod Res 55(10):2950–2960
4. Baader F, Horrocks I, Lutz C, Sattler U (2017) An introduction to description logic. Cambridge University Press, Cambridge
5. Bellifemine FL, Caire G, Greenwood D (2007) Developing multi-agent systems with JADE (Wiley Series in Agent Technology). John Wiley & Sons, Inc., Hoboken
6. Bortolini M, Accorsi R, Faccio M, Galizia FG, Pilati F (2019) Toward a real-time reconfiguration of self-adaptive smart assembly systems. Procedia Manufacturing 39:90–97. 25th International Conference on Production Research Manufacturing Innovation: Cyber Physical Manufacturing
7. Chaplin J, Bakker O, de Silva L, Sanderson D, Kelly E, Logan B, Ratchev S (2015) Evolvable assembly systems: A distributed architecture for intelligent manufacturing. IFAC-PapersOnLine 48(3):2065–2070. 15th IFAC Symposium on Information Control Problems in Manufacturing
8. Cheng BHC, de Lemos R, Giese H, Inverardi P, Magee J (2009) Software engineering for self-adaptive systems: a research roadmap. Springer, Berlin, pp –26
9. de Silva L, Felli P, Chaplin JC, Logan B, Sanderson D, Ratchev S (2016). In: Proceedings of the twenty-second european conference on artificial intelligence, ECAI'16, pp 1449–1457, NLD. IOS Press, p Realisability of production recipes
10. de Silva L, Felli P, Sanderson D, Chaplin JC, Logan B, Ratchev S (2019) Synthesising process controllers from formal models of transformable assembly systems. Robot Comput Integr Manuf 58:130–144
11. Derigent W, Cardin O, Trentesaux D (2021) Industry 4.0: Contributions of holonic manufacturing control architectures and future challenges. J Intell Manuf 32(7):1797–1818
12. Digital Manufacturing and Design Innovation Institute (2018) Strategic investment plan for 2018
13. ElMaraghy HA (2005) Flexible and reconfigurable manufacturing systems paradigms. Int J Flex Manuf Syst 17:261–276
14. European Factories of the Future Research Association (2013) Factories of the future: multi-annual roadmap for the contractual PPP under Horizon 2020. European Commission
15. Fogliatto FS, da Silveira GJ, Borenstein D (2012) The mass customization decade: an updated review of the literature. Int J Prod Econ 138(1):14–25
16. Gao J, Yao Y, Zhu VC, Sun L, Lin L (2011) Service-oriented manufacturing: a new product pattern and manufacturing paradigm. J Intell Manuf 22(3):435–446
17. Gonnermann C, Weth J, Reinhart G (2020) Skill modeling in cyber-physical production systems for process monitoring. Procedia CIRP 93:1376–1381. 53rd CIRP Conference on Manufacturing Systems 2020
18. Hasan M, Starly B (2020) Decentralized cloud manufacturing-as-a-service (CMaas) platform architecture with configurable digital assets. J Manuf Syst 56:157–174
19. Järvenpää E, Siltala N, Hylli O, Lanz M (2019) The development of an ontology for describing the capabilities of manufacturing resources. J Intell Manuf 30:959–978

20. Kagermann H, Wahlster W, Helbig J (2013) 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. Forschungsunion
21. Koren Y, Gu X, Guo W (2018) Reconfigurable manufacturing systems: principles, design, and future trends. Front Mech Eng 13:121–136
22. Koren Y, Shpitalni M (2010) Design of reconfigurable manufacturing systems. J Manuf Syst 29(4):130–141
23. Leitão P, Mařík V, Vrba P (2013) Past, present, and future of industrial agent applications. IEEE Trans Industr Inf 9(4):2360–2372
24. Li F, Xu Z, Li H (2021) A multi-agent based cooperative approach to decentralized multi-project scheduling and resource allocation. Comput Ind Eng 106961:151
25. Liu C, Su Z, Xu X, Lu Y (2022) Service-oriented industrial internet of things gateway for cloud manufacturing. Robot Comput Integr Manuf 102217:73
26. Lu Y, Xu X (2018) Resource virtualization: a core technology for developing cyber-physical production systems. J Manuf Syst 47:128–140
27. Malec J, Nilsson A, Nilsson K, Nowaczyk S (2007) Knowledge-based reconfiguration of automation systems. In: 2007 IEEE international conference on automation science and engineering, pp 170–175
28. Marsland S (2014) Machine learning: an algorithmic perspective. Chapman & Hall/Crc
29. Pfrommer J, Schleipen M, Beyerer J (2013) PPRS: Production skills and their relation to product, process, and resource. In: 2013 IEEE 18th Conference on Emerging Technologies Factory Automation (ETFA), pp 1–4
30. Sanderson D, Chaplin J, Ratchev S (2019) A function-behaviour-structure design methodology for adaptive production systems. Int J Adv Manuf Techno 105:3731–3742
31. Scrimieri D, Adalat O, Afazov S, Ratchev S (2022) Modular reconfiguration of flexible production systems using machine learning and performance estimates. IFAC-PapersOnLine 55(10):353–358. 10th IFAC Conference on Manufacturing Modelling, Management and Control MIM 2022
32. Scrimieri D, Afazov SM, Ratchev SM (2021) Design of a self-learning multi-agent framework for the adaptation of modular production systems. Int J Adv Manuf Technol 115:1745–1761
33. Scrimieri D, Antzoulatos N, Castro E, Ratchev SM (2017) Automated experience-based learning for plug and produce assembly systems. Int J Prod Res 55(13):3674–3685
34. Scrimieri D, Oates RF, Ratchev SM (2015) Learning and reuse of engineering ramp-up strategies for modular assembly systems. J Intell Manuf 26(6):1063–1076
35. Timm IJ, Woelk P (2003) Ontology-based capability management for distributed problem solving in the manufacturing domain. In: Schillo M, Klusch M, Müller JP, Tianfield H (eds) Multiagent System Technologies, First German Conference, MATES 2003, Erfurt, Germany, September 22-25, 2003, Proceedings, volume 2831 of Lecture Notes in Computer Science, pp 168–179. Springer
36. Valckenaers P (2019) ARTI reference architecture – PROSA revisited. In: Borangiu T, Trentesaux D, Thomas A, Cavalieri S (eds) Service Orientation in Holonic and Multi-Agent Manufacturing, pages 1–19, Cham. Springer International Publishing
37. Valckenaers P, Van Brussel H (2005) Holonic manufacturing execution systems. CIRP Ann 54(1):427–432
38. Wang G, Wong T, Wang X (2014) A hybrid multi-agent negotiation protocol supporting agent mobility in virtual enterprises. Inf Sci 282:1–14
39. Wang T, Zhang P, Liu J, Zhang M (2021) Many-objective cloud manufacturing service selection and scheduling with an evolutionary algorithm based on adaptive environment selection strategy. Appl Soft Comput 112:107737
40. Wettschereck D, Aha DW, Mohri T (1997) A review and empirical evaluation of feature weighting methods for a class of lazy learning algorithms. Artif Intell Rev 11:273–314
41. Wilson DR, Martinez TR (1997) Improved heterogeneous distance functions. J Artif Intell Res 6:1–34
42. Wu D (2001) Software agents for knowledge management: coordination in multi-agent supply chains and auctions. Expert Syst Appl 20(1):51–64
43. Zeng J, Yao J, Gao M, Wen J (2022) A service composition method using improved hybrid teaching learning optimization algorithm in cloud manufacturing. J Cloud Comput 11(1):1–14
44. Zhen L (2012) An analytical study on service-oriented manufacturing strategies. Int J Product Econ 139(1):220–228. Supply Chain Risk Management
45. Zhou L, Zhang L, Ren L, Wang J (2019) Real-time scheduling of cloud manufacturing services based on dynamic data-driven simulation. IEEE Trans Ind Inf 15(9):5042–5051