To appear in the *International Journal of Production Research*

A Multi-agent Framework for Capability-based Reconfiguration of Industrial Assembly Systems

Nikolas Antzoulatos[a], Elkin Castro[a*], Lavindra de Silva[a],
André Dionisio Rocha[b], Svetan Ratchev[a], José Barata[b]

[a]*Institute for Advanced Manufacturing, University Park, University of Nottingham, Nottingham, United Kingdom;* [b]*CTS-Uninova, Dep. de Eng. Electrotecnica, Faculdade de Ciencias e Tecnologia, Universidade Nova de Lisboa, Lisbon, Portugal*

(*Received 15 Nov 2015; accepted 24 Sep 2016*)

Rapidly changing market requirements and shorter product lifecycles demand assembly systems that are able to cope with frequently changing resources, resource capabilities, and product specifications. This paper presents a multi-agent framework that can adapt an assembly system in order to cope with such changes. The focus of this work is on the ability to plug resources (such as PLCs) into and out of the system, and dynamically aggregate resource capabilities to form more complex ones as resources are plugged in. In addition, an implementation of the framework on an industrial assembly system is discussed, and some insights are provided into some of the key features that product specification languages ought to have to be useful in real world assembly systems, and into the added value of using the proposed framework.

Keywords: Manufacturing, multi-agent system, capability framework, plug and produce, assembly systems, reconfiguration

1. Introduction

Since the industrial age, technological and socio-economical changes have altered customer demand. There is a shift away from mass production, where the same product is manufactured in a large number of batches, toward individualised mass production of a 'batch size of one' (Scholz-Reiter and Freitag 2007). One approach to meeting this demand is via product variants, which allow a shorter time to market by altering certain attributes in an existing product's design. This leads to lower R&D costs, since development-intensive innovation is minimised, and the need for technological improvements is reduced (Elgh 2007). In the future, individualised mass production is expected to allow clients to heavily personalise their products before purchase, as offered by the vehicle industry where cars can be personalised by choosing components and extras from a catalogue before purchase. For example, the prospective owner of a BMW 7 Series can choose from 10^{17} personalisation combinations (Kaluza and Blecker 2005).

Changes in customer demand can also have an impact on the shop floor. The reconfiguration of the shop floor that is needed to cope with this impact can be as small as hardware parameter adjustments, or as major as changes to the physical layout of the assembly system (Ribeiro et al. 2013). Since such changes may require halting assembly, it is crucial that they take place within a reasonable amount of time in order to minimise costs (Dolgui and Proth 2010).

In view of this, this paper proposes a framework to manage the adaptation of assembly systems to

*Corresponding author. Email: Elkin.Castro@Nottingham.ac.uk cope with changes in the production environment, or more specifically, in the product specification and the capabilities of production resources. Product specifications are defined "procedurally", i.e., in terms of the sequence of operations that must be carried out in order to assemble an end-product. An example of such a specification is the list of steps representing the operations "drill a hole of depth 0.5mm in the cube, and then polish it". This contrasts with a "declarative" encoding such as the one advocated by variants of the Planning Domain Definition Language (PDDL) (Fox and Long 2003), which capture the desired characteristics of the end product, e.g., that "the cube must be polished and have a hole of depth 0.5mm". While the latter kind of specification has also been successfully used in industrial applications, including print-job scheduling on reconfigurable printers (Ruml et al. 2011), the work presented here follows the approach in which the assembly of the end product is specified procedurally, as this is in congruence with what is advocated in the manufacturing industry, evidenced by languages such as the Process Specification Language (PSL) (Qiao, Kao, and Zhang 2011). Insights are provided into some of the key features that a product specification "language" should have in order to be useful in real world assembly systems.

The framework proposed in this paper uses the JADE agent platform (Bellifemine, Caire, and Greenwood 2007), and thereby its FIPA compliant communication infrastructure. This paper focuses on the ability to plug resources into and out of the system ("plug and produce"), and describes the design decisions that were made in order to achieve this, e.g., how the various agent types in the system were hierarchically organised. In particular, the proposed framework supports dynamically aggregating atomic resource capabilities to form more complex ones as resources such as PLCs are plugged in, and removing capabilities as resources are plugged out. The paper also describes an implementation of the proposed framework on an industrial assembly system used for assembling hinges for vehicle furniture, and evaluates the added value of the framework.

This paper is organised as follows.[1] Section 2 gives an overview of the research field and highlights the most important approaches. Section 3 presents the framework to adapt assembly systems based on resource capabilities. Finally, an implementation of the framework is presented in Section 4, and the paper is concluded in Section 5.

2. Related work

Broadly speaking, manufacturing systems can be classified into those that are "static" and those that are "dynamic". The traditional kind of manufacturing system can be considered static, where changes are only allowed once the system is offline. Such systems cannot be reconfigured frequently, and there is little need to be responsive to runtime changes (Landers, Ruan, and Liou 2006). Nonetheless, static systems are robust, and tailored for mass production, focusing on producing one type of product at a high production rate (Koren and Shpitalni 2010).

On the other hand, dynamic systems allow for changes to be made to the system whilst it is in operation, in order to accommodate fluctuating production volumes and varying product specifications. This suits the growing need for highly customised products, which demand manufacturing modules, such as robots, capable of autonomously adapting—at runtime—to keep up with a changing production environment (Terkaj, Tolio, and Valente 2009).

Paired with existing techniques in Artificial Intelligence (AI), a number of new manufacturing paradigms have emerged to facilitate the implementation of dynamic assembly systems. A prominent example is Cyber-Physical Production Systems (CPPS) (Monostori 2014), which may lead to the 4th Industrial Revolution or Industry 4.0. In this paradigm, the manufacturing system is transformed from a traditional hierarchical control infrastructure to a heterarchical architecture composed of single building blocks that represent machines. These blocks are equipped with their own computing power to control its hardware and to communicate in real time with surrounding

[1]This paper is an extended version of Antzoulatos et al. (2015).

blocks.

A manufacturing system composed of heterarchical modules is closely related to multi-agent architectures. In (Alsafi and Vyatkin 2010), a multi-agent system is presented that is able to adapt to changes in the manufacturing environment, such as product specifications and the topology describing how manufacturing resources are connected. The architecture is able to dynamically synthesise an alternative configuration for the changed environment, in the form of a sequence of low-level manufacturing operations. While these operations in their approach are atomic and available as soon as resources are plugged in, the work presented in this paper focuses on abstract (or "complex") operations whose availability is determined by hierarchically aggregating the atomic operations in currently available resources, such as a robot arm and a gripping tool.

Similarly, Barata et al. (Barata, Camarinha-Matos, and Cândido 2008) propose a multi-agent architecture where each manufacturing resource is represented by an agent. Like the work presented in this paper, they also allow easy resource integration and exclusion from the system ("plug and produce"). To this end, each resource agent supplies its own capabilities, which are combined to obtain the capabilities of the entire system. This set is analysed by an agent who offers the operator of the system an interface for manually aggregating the capabilities to form more complex ones (Cândido and Barata 2007). While the core of their approach is similar to the work presented here, they do not provide details regarding their mechanisms for creating complex capabilities, and how the properties of capabilities influence the configuration of the system. Similarly, Frei et al. (Frei and Serugendo 2011) propose an approach to combining simpler capabilities into more complex ones during the formation of coalitions between groups of agents. In contrast to the work presented here, their complex capabilities do not seem to be hierarchical, and they also do not provide details regarding the methods that they use for capability aggregation. Moreover, they focus on a somewhat theoretical approach, whereas the approach in this paper is fully implemented on an industrial assembly system.

In de Silva et al. (2016), the authors provide a formal approach to determining whether a given product specification can be realised on a given production topology that describes the layout of the production system. For a realisable product specification, the authors describe how a controller can be extracted in order to instruct the resources regarding how to realise the recipe, i.e., which tasks in the specification should be executed on which resources, and when. However, the authors assume that tasks appearing in product specifications are described at the same level of abstraction as the manufacturing processes that are available on the hardware resources, whereas the work presented here allows the product to be abstractly specified in a different "language", and maps this specification into the low-level processes that are offered by hardware. Moreover, while their work focuses on production layouts that do not change frequently, this paper focuses on those that do.

In Timm and Woelk (2003), the authors present a "capability management" framework that aims to bridge the gap between process planning and production control. They describe an approach where the product to be manufactured is specified declaratively, i.e., in terms of its geometric features, and this is used by a decision making algorithm in order to determine which process combinations are the most suitable ones for achieving the specification, from the processes that occur within the hierarchical capabilities in the system. The work presented in this paper, on the other hand, assumes that the process combinations needed to manufacture a product are supplied by the operator in the form of a product specification.

Perhaps the most closely related work to the approach presented in this paper is that of Järvenpää (2012), who also assigns capabilities to resources which are then matched against the product specification via certain matching rules. The main differences between the two approaches are the following. First, in the work presented in this paper, all the possible complex capabilities of the system are (re)computed as soon as a device is plugged in or out, whereas they do this only when no match is found between the existing complex capabilities and the given product specification. Second, while they do allow for certain changes in the manufacturing environment—e.g., to the product specification—they do not seem to focus on "plug and produce" systems, in particular, automatically updating the system's (basic and complex) capabilities when resources are plugged in and out; this seems to require operator intervention via a "Capability Editor" tool. Similarly, while the work presented in this paper aims at a system that reconfigures dynamically, their framework is intended for production environments in which human operators participate actively, in order to validate the computed assembly (re-)configurations, for example.

While not agent systems *per se*, HTN planners (Ghallab, Nau, and Traverso 2004) employ similar structures and semantics to how capabilities are modelled and manipulated by an agent. In contrast with the work proposed here, HTN planning is not generally concerned with building and modifying capabilities: they are supplied once by the domain expert and remain static. One HTN-based approach is the SIARAS (Malec et al. 2007) project, which applies AI reasoning to the reconfiguration of assembly systems. To this end, they develop a system called a skill server that captures the capabilities of the assembly system using OWL (Ontology Web Language), to which they match the product specification—while adhering to user-supplied criteria—in order to determine whether the product can be produced using the modelled assembly resources. While their approach also supports changing product specifications, their layout of the assembly resources seems to be static, whereas the ability to dynamically plug them in and out is a key concern in the work presented in this paper.

3. A framework for the management of resource capability-product specifications matching

In this section, an implementation of a multi-agent framework for matching the capabilities of an assembly platform against the specifications of a product is presented. This framework is designed so that after a feasible matching is performed, it automatically configures the control system to assemble the selected product. Otherwise, the HMI (Human-Machine Interface) informs the operator about the missing capabilities.

Agents in this framework have been implemented using the Java-based JADE platform, which is a tool for the development of distributed software-agent applications (Bellifemine, Caire, and Greenwood 2007). Two key features of JADE that are relevant to us are: *(i)* its management of the agent life-cycle, e.g. the automatic assignment of unique agent identifiers, and the registration and de-registration of agents from their respective sub-domains (or "groups") as they are created and destroyed; and *(ii)* the provision of an API for agent communication via the FIPA standard, which is the most popular specification for agent communication.

The proposed multi-agent framework is composed of three parts. First, in Sections 3.1 and 3.2, a methodology for managing the capabilities of an assembly platform is presented. Second, in Section 3.3, the definition of a concrete product is described. Finally, in Section 3.4 the capability matching of an assembly system against a product specification is described.

3.1 *Capability representation*

A capability is either associated with an individual resource, such as an end effector or a robot, or with a set of resources, such as a robot with an end effector attached to it. Thus, there is a distinction between *atomic* and *complex* capabilities.

An atomic capability is a functionality of a resource in isolation, which is defined as the duple (t, ref). Element t is the name of a task whose parameters are all variables and ref is a reference to the associated PLC code that executes the task. For example, a robot alone (with no end effector attached) has the atomic capabilities $Move(From, To)$ and $Press(Force)$, and the following grasp capability can be associated with a finger gripper: $Grasp(MaxWidth, MinWidth, Geom, Poc)$, where $Geom$ stands for $Geometry$, and Poc for

Pocket. A complex capability represents functionality resulting from aggregating resources that must work together in order to realise the desired capability. For example, the complex capability *pickAndPlace*(*From*, *To*, $MaxWidth$, $MinWidth$, *Geom*, *Poc*) is the aggregation of the *Move*(*From*, *To*) and *Grasp*($MaxWidth$, $MinWidth$, *Geom*, *Poc*) atomic capabilities. A complex capability is defined as the duple (t, *body*), where t is as defined above for atomic capabilities, and *body* is a set of atomic and complex capabilities that need to be performed in order to realise the complex capability t. It is assumed that the set of capabilities appearing within a body has been decided by the domain expert, possibly after trying it in practice and testing it within the given assembly environment.

3.2 *Capability aggregation*

Figure 1 is the component-and-connector view of a multi-agent system that delivers the capability aggregation functionality. The agent system in this figure is associated to the assembly platform described in Section 4.1.

The capabilities of an assembly system are managed by the following agents: Component Agents (CAs), Production Management Agents (PMAs), and Capability Management Agents (CMAs). These agents are organised as a rooted tree: CAs are the leaf nodes, and one PMA is the root. The children of a PMA are CAs and other PMAs that represent assembly resources which must work together in order to realise certain complex capabilities. The set of children of a PMA is called a subsystem. In Figure 1, the root PMA has three children: two PMAs (PMA_1 and PMA_2), and the CA of a shuttle system. The subsystem defined by PMA_1 contains the CAs associated to the following resources: tool1, tool2, tool3, tool4, tool5, tool6, robot1, and robot2. The resources of the subsystem defined by PMA_2 are a camera and a force tester.

A CA represents an individual assembly resource. This agent is created automatically after a resource is plugged in, via its associated Deployment Agent (DA). A CA is responsible for informing its associated PMA about the atomic capabilities of its corresponding assembly resource. Additionally, a CA is the interface between the multi-agent system and the control system. A CA has access to an XML file called a blueprint file, which contains information about the resource: its ID, its human-readable name and description, and the name of its 'communication library' and atomic capabilities. The communication library field specifies a particular software library used by the CA in order to interface with the control system, and reconfigure the associated resource. Any instance of a PMA has an associated CMA, which collects the capabilities at the level of abstraction of its corresponding PMA and aggregates them, according to rules defined in the configuration database, in order to provide complex capabilities. The configuration database stores all possible capabilities of the corresponding assembly system and its associated aggregation rules, which determine the sets of necessary capabilities to provide complex ones. The CMA associated to the root PMA combines the capabilities at this level, and then the root PMA informs the Prime System Agent (PSA) about the capabilities of the entire assembly system. With that information, the PSA updates the Prime Semantic Model in which the current capabilities of the assembly system are stored.

Table 1 shows the tree of Figure 1 in textual form. This table indicates that the tree has three levels of capability aggregation whose root node is the PMA at level 1. Leaf nodes are at level 2 and 3, and they must be CAs. Entries with one or more CAs are shown with their corresponding DAs. Elements in bold in the table indicate parent-child relations. For instance, the CA of Tool 2 at level 3 is a child of PMA_1 at level 2. Moreover, each cell with a PMA is shown with its corresponding CMA. As stated earlier, the set of children of a PMA is called a subsystem. Thus, the table shows that there are three subsystems as there are three PMAs: PMA_1 and PMA_2 at level 2, and PMA at level 1.

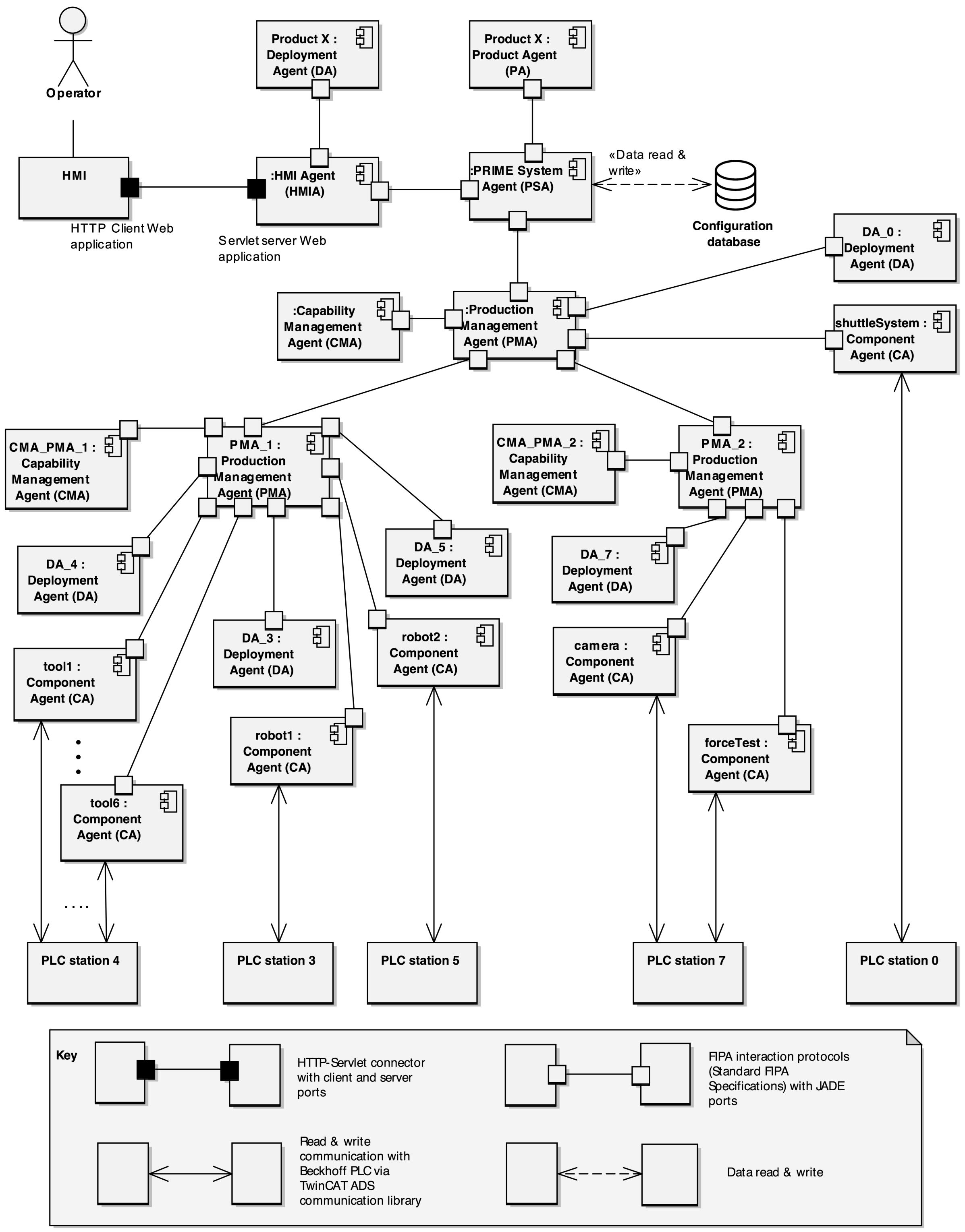

Operator
HMI
HTTP Client Web application
Servlet server Web application
Product X : Deployment Agent (DA)
:HMI Agent (HMIA)
Product X : Product Agent (PA)
:PRIME System Agent (PSA)
«Data read & write»
Configuration database
DA_0 : Deployment Agent (DA)
:Capability Management Agent (CMA)
:Production Management Agent (PMA)
shuttleSystem : Component Agent (CA)
CMA_PMA_1 : Capability Management Agent (CMA)
PMA_1 : Production Management Agent (PMA)
CMA_PMA_2 : Capability Management Agent (CMA)
PMA_2 : Production Management Agent (PMA)
DA_4 : Deployment Agent (DA)
DA_5 : Deployment Agent (DA)
DA_7 : Deployment Agent (DA)
DA_3 : Deployment Agent (DA)
robot2 : Component Agent (CA)
camera : Component Agent (CA)
tool1 : Component Agent (CA)
robot1 : Component Agent (CA)
forceTest : Component Agent (CA)
tool6 : Component Agent (CA)
PLC station 4
PLC station 3
PLC station 5
PLC station 7
PLC station 0
Key
HTTP-Servlet connector with client and server ports
FIPA interaction protocols (Standard FIPA Specifications) with JADE ports
Read & write communication with Beckhoff PLC via TwinCAT ADS communication library
Data read & write

Figure 1. A multi-agent framework for capability-product specification matching and control system reconfiguration

Table 1. Tree structure of the implemented multi-agent system framework for capability-product specification matching and control system reconfiguration

Level 1	Level 2	Level 3
	CA (Shuttle system) DA_0	–
PMA CMA	**PMA_1** CMA_PMA_1	**CA (Robot 1)** DA_3
		CA (Robot 2) DA_5
		CA (Tool 1) **CA (Tool 2)** ⋮ **CA (Tool 6)** DA_4
	PMA_2 CMA_PMA_2	**CA (Camera)** **CA (Force Test)** DA_7

The processes of resource and subsystem identification, and capability aggregation at the level of any PMA-CMA pair are executed periodically because resources can be plugged in/out at any time. This implementation allows the PSA to keep an up to date record of the capabilities of the assembly system in the Prime Semantic Model. Atomic capabilities of resources are defined in their corresponding blueprint files by the domain expert. Once these are defined, the domain expert includes aggregation rules in the configuration database in order to account for these atomic capabilities. The proposed approach contrasts with standard BDI (Belief-Desire-Intention) and HTN (Hierarchical Task Network) systems, where capabilities are not removed from the system once they are written and included by a domain expert.

3.3 *Product specification*

A product specification is a procedural encoding of the operations needed, and their sequence, in order to assemble the desired product. The product specification is defined at the same level of abstraction as the capabilities in the PSA. The approach requires that operations within a product specification are consistent with an *operation definition*, which is a name (sequence of characters) followed by a list of typed parameters, where each parameter is also a string corresponding to either a constant or a variable. For example, an operation definition in the proposed framework is *pickAndPlace*($String : From$, $String : To$, $Float : MinMaxWidth$, $Float : MaxMinWidth$, $String : Geom$, $Boolean : Poc$), which is a 'pick and place' operation that picks an item from location $From$, and moves it to location To. The third and fourth parameters specify how wide the finger gripper should open and close so that the given part can be grasped; specifically, they encode that the gripper must open to a width that is greater than $MinMaxWidth$, and close to a width smaller than $MaxMinWidth$. The $Geom$ parameter corresponds to the shape of the inner face of the fingers of the gripper, which can bind either to the constant $flat$ or to $cylinder$. Finally, the Poc parameter is a boolean that indicates whether the tool must have an indentation in its inner face or not, which provides a better grip. Thus, one example of an operation that conforms to the above operation definition is *pickAndPlace*(WS_10, WS_11, 15, 15, $flat$, $true$), where WS stands for $Workspace$.

Operations in the product specification can be partially specified by ignoring one or more of their parameters; an ignored parameter is denoted by a wildcard character. The advantage of such partial specification is twofold. First, it offers the convenience of not needing to specify bindings for parameters that are not necessary in order to achieve the overall operation. Second, it offers the underlying agent framework the flexibility to choose a suitable parameter binding, such as one that optimises resource usage. For example, in the proposed framework, if the part to be manipulated is not heavy, it does not matter whether the gripper has a pocket or not—i.e., whether the *Poc* parameter has the value *true* or *false*—provided, of course, that the remaining values are specified. Thus, one could write $pickAndPlace(WS\_10, WS\_11, 15, 15, flat, *)$. By ignoring the *Poc* parameter, the framework is also given the opportunity to use any tool to perform the task, such as the closest tool to the robot arm. Such partial specification of operations is similar to 'function overloading' in programming languages, where multiple functions have the same name but a different number of parameters.

Assembly operations such as the ones described above are combined into a sequence to form the product specification, which is sent to the PSA as follows. First, the HMI displays a range of product variants from which an operator can choose one. After a variant is picked, the HMI Agent (HMIA) uses its associated DA to create a PA according to what the operator has selected. After the PA is created, it requests the PSA to configure the assembly system according to its associated sequence of assembly operations.

3.4 *Matching capabilities to a product specification*

After a PA is created, the PSA starts matching the product specification against the capabilities of the assembly system. The PSA finds a suitable capability for each operation in a product specification such that the first element of the capability (task t) matches the operation that needs to be performed. This matching process (or "unification" as it is called in AI) involves checking whether the names of the tasks and operations are identical, that they have the same number of parameters, and whether the parameter types in the operations to be achieved correspond to those in the task definition. Finally, the matching process checks that the values of the parameters of the operation are valid with respect to the range of values of the corresponding parameters of the capability. This relates to the "relevance" and "applicability" checks performed in BDI agent platforms (Winikoff 2005). An operation may have more than one suitable capability in which case one is picked arbitrarily, unless a preference between capabilities has been specified. For example, a preference could exist for using a finger gripper over a pneumatic gripper. In this case, the preferred capability is chosen.

If there is no match (there is one or more operations in the product specification for which no resources can provide suitable capabilities) the PSA informs the HMIA about it, together with the operations in the product specification for which no match was found. On the other hand, if the product specification is matched with a set of capabilities, the PSA synthesises an assembly plan. This is a sequence of feasible capabilities, together with references to the associated PLC code of the corresponding atomic capabilities. This plan is sent to the root PMA, which decomposes it according to its hierarchy. This decomposition process is performed until the CAs are reached, at which point the CAs reconfigure the control logic.

4. Implementation

The proposed framework is implemented and robustly working on an industrial assembly system. In this section, the layout and hardware of this demonstrator is introduced, and the multi-agent system that assists with reconfiguration is described.

4.1 *The assembly platform*

The demonstrator is a highly flexible assembly platform by Feintool Automation. Fig. 2 shows the twin-robot setup and tool rack in the foreground, and the shuttle system in the background, which supports a pallet with individual parts. The main product assembled by the platform is a detent hinge, used in the cab interiors of some commercial trucks. The hinge consists of 10 parts, namely, two hinge leaves, a metal pin, three balls, three springs and a retainer. The two separate leaves are attached using the following parts: a metal pin, which "locks" the leaves together; three metal balls, which are placed into adjacent cylindrical slots in the centre of the leaves; three springs, which are placed into the same slots; and a retainer which is used to retain the springs and balls inside the leaves. The force that needs to be applied to overcome the detent of the hinge can be chosen by the customer, along with one of four possible variants of the product: three balls and three springs (i.e., the default product), two balls and two springs, one ball and one spring, or no balls nor springs.

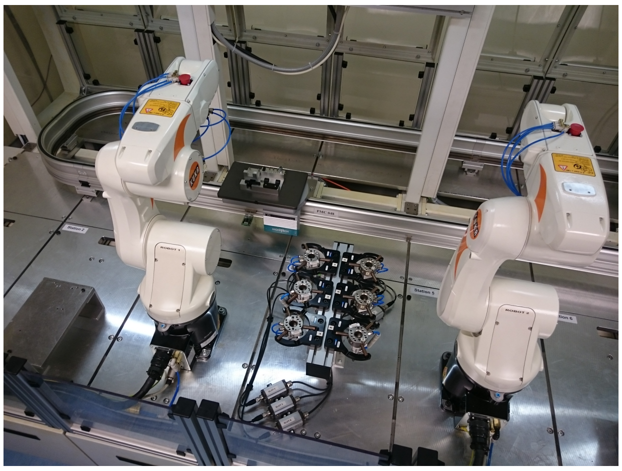
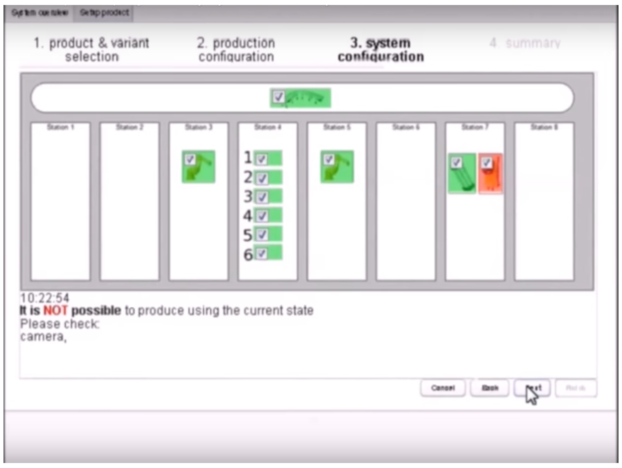


Figure 2. Twin robot setup and tool rack



10:22:54
It is NOT possible to produce using the current state
Please check:
camera,

Figure 3. Reconfiguration HMI

The assembly platform consists of eight modules, which are independent in terms of both mechanical structure and electrical wiring. All modules are served by a pallet, which is transported by a Montech linear shuttle system. The pallet carries all the raw parts, sub-assemblies and the finished product to the relevant modules.

A module is the basic structural element of the assembly platform. The demonstrator can be extended by combining additional modules side by side via a standard interface. This is possible because each module has its own process control, which allows for building a distributed control architecture. A module is controlled by a Beckhoff CX5010 PLC, and the transfer system by a Beckhoff CX2030 PLC. Six out of the eight available modules are in use, into which the following resources are installed: two robots, two workspaces, a tool changing rack, and an inspection station. Each robot is a Kuka KR5 sixx R650 six-axis robot, which has access to the shared tool changing rack as well as its own private workspace. The module between the robots is an installation of a Schunk tool changing rack, which can hold upto six RFID-identified end of arm effectors: two vacuum suction grippers, and four two-finger grippers of various shapes and sizes. Each gripper can be used for both pick-and-place and insert operations on various parts and sub-assemblies. This combination of the tool changing rack with multi-purpose, multi-part grippers creates a highly robust, flexible and reconfigurable assembly platform, which can even operate with just one robot and a (minimal) tool set with two tools. Thus, this setup is useful in the event of a resource failure, such as a broken robot or tool, or in the event of a production volume increase, when more resources need to be available. The inspection station performs two tests: a mechanical test exercises the hinge in order to check the detent force, and a vision test verifies that the assembly has completed successfully.

4.2 *Operating the assembly platform*

The assembly framework assists with the assembly and inspection of a desired product variant, encoded in the form of a product specification. Figure 4 presents a workflow describing the proposed framework, which starts with the operator choosing the product variant, and ends with a successful reconfiguration of the PLCs. Below, some of the more interesting cases depicted in the figure are described.

The operator chooses the product variant on the reconfiguration HMI, which is the interface to the agent environment and the entire assembly system. Figure 3 shows a screenshot of the HMI. The HMI provides full transparency for reconfiguration, which avoids the need for the operator to know about the workings of a multi-agent system. Like many industrial HMIs, the operator "creates" the product specification by choosing a product variant (such as a 3-ball hinge) and specifying other assembly attributes, including the inspection that must be performed on the product (i.e., either a vision or force test). The HMI verifies the attribute values and creates a PA agent, which represents the (physical) product variant within the multi-agent system. The PA then requests the PSA to reconfigure the system in order to fulfil the product specifications.

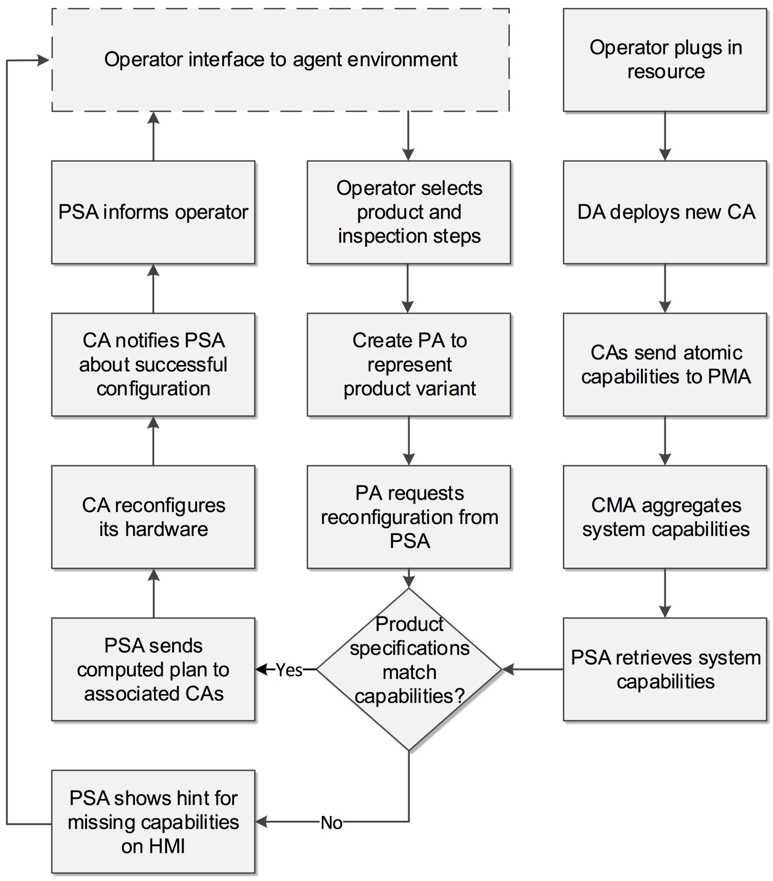

Operator interface to agent environment
Operator plugs in resource
PSA informs operator
Operator selects product and inspection steps
DA deploys new CA
CA notifies PSA about successful configuration
Create PA to represent product variant
CAs send atomic capabilities to PMA
CA reconfigures its hardware
PA requests reconfiguration from PSA
CMA aggregates system capabilities
PSA sends computed plan to associated CAs
Yes
Product specifications match capabilities?
PSA retrieves system capabilities
PSA shows hint for missing capabilities on HMI
No

Figure 4. A workflow describing the reconfiguration methodology

In some cases, matching the product specification against the recorded (complex) capabilities can fail, for example, if a vacuum gripper (tool 3) that is needed to perform a pneumatic pick-and-place operation is missing in the system, resulting in complex capabilities being unavailable for the pick-and-place operation. In such cases, the PSA informs the operator about the failure via the HMI, by highlighting which step in the product specification could not be matched, and providing a suggestion regarding a resource that might be suitable for the step, based on the ones that were recently allocated to it. The operator is also given on-screen instructions on how to extend the existing set of system resources with the new one.

Once a (possibly new) resource is plugged into the system, the following steps take place. First, its blueprint file is detected by the DA, and a new CA is deployed for the resource. Second, the CA announces itself to the higher level agents. Third, the CA sends its capabilities to its associated PMA, who then sends them to its CMA. Fourth, the CMA aggregates these capabilities with those belonging to its other associated CAs to form the complete set of capabilities in the CMA's subsystem. Finally, all such sets in the various subsystems are sent to the PSA to be used as input when determining whether a given product specification can be matched.

In the event where matching the product specification against the recorded capabilities is successful (the "Yes" case in Figure 4), the PSA generates an assembly plan, and distributes it to all the involved CAs, who then reconfigure the hardware. If the reconfiguration is successful, the operator is notified that the system is ready to start assembly. The operator then loads the shuttle system with a pallet containing all the required individual parts, which are transported and assembled by the various stations with no intervention needed from the operator or the agent system. This approach ensures that PLC code, which is often the product of years of industrial use and testing, runs unhindered. Nonetheless, the agent system does monitor the assembly process until a runtime error occurs, such as a missing tool, in which case the agent system takes control and attempts to find an alternative assembly plan.

4.3 *Evaluation of the assembly platform*

Table 2 provides a comparative evaluation between a traditional manufacturing system and one that is equipped with the proposed framework. In summary, the proposed framework minimises the skills needed to operate manufacturing hardware, and supports automated assembly, rapid reconfiguration, and adaptation to changes in products, processes and resources. This amounts to significant cost and time reductions when deploying and maintaining complex assembly systems.

Table 2. Table comparing the benefits of the proposed reconfiguration framework

Feature	Without the proposed framework	With the proposed framework
Capability Matching	The mapping from operations appearing in a (high-level) product specification to the sequence of (low-level) operations provided by hardware needs to be done manually for each product specification.	This mapping is encoded just once by a domain expert, in the form of (hierarchical) capabilities, and performed automatically whenever a new product specification is received by the system.
System Reconfiguration	The system is reconfigured by the operator. If a new product is needed, the operator must manually reconfigure the control logic as described in Section 3.4.	The operator does not need to intervene when a new product is needed, as the system will regenerate an assembly plan and reconfigure the control logic.
Plug and Produce	The addition and removal of resources to/from the system must be manually handled. This involves updating the system topology, changing the control logic, etc.	The addition and removal of resources to/from the system are detected automatically: complex capabilities are aggregated, and the control logic is reconfigured.
Multi-vendor Integration	The operator needs training to work with PLCs from different vendors (e.g. Siemens and Beckhoff), and to make them interoperate.	The operator is provided with a single, high-level interface to the potentially diverse PLCs installed, and the agents manage their interoperation.
Performance monitoring	PLC variables need to be manually monitored against things such as assembly objectives and sub-optimal behaviour.	Agents automatically monitor PLC variables against assembly objectives and bottlenecks. The continuous analysis of assembly systems allows for data logging, and early detection of system failure and sub-optimal behaviour.
Operator Support	Only skilled operators can use the system, as the system provides limited support.	Operators require only basic skills. The HMI guides the operator through the reconfiguration process and provides support to validate whether the product can be assembled, and if not, informs the operator accordingly. By minimising manual reconfiguration and programming, the HMI also minimises the introduction of errors.

5. Summary and conclusion

This paper proposes a capability-based framework to reconfigure frequently changing assembly systems, particularly those in which resources need to be plugged in and out. The requirements for assembling a product are described as product specifications, by using a language that takes a step toward catering for real world assembly systems. The framework, product specifications and manufacturing resources are managed by a multi-agent system, where agents communicate with each other via the FIPA communication infrastructure in order to aggregate resource capabilities and match them against product specifications. A fully implemented reconfiguration methodology was described, and a comparative evaluation of the proposed system was presented, by comparing it to a system that does not incorporate the presented framework.

While the proposed framework has advantages as summarised in Table 2, it also has limitations. First, the framework requires the operator to supply additional information, such as blueprint files describing resources. While such information is only supplied once, and leads to gains as described in the table, creating such information might be an overhead (in terms of needing extra effort) if the assembly system changes infrequently, as is the case with some legacy systems. Second, the framework can only be installed on assembly systems that use soft PLCs, such as Beckhoff and Siemens Open Controller.

The proposed framework is fully implemented and runs on a real-world assembly system. We believe that instantiating this framework on real industrial environments will have positive business implications for two reasons. Firstly, in the presented approach the control of manufacturing processes is the responsibility of the PLC, which means that existing control logic does not need to be changed or unveiled. Secondly, the implementation is based on off-the-shelf and industry approved control hardware, which decreases the need for new investments.

In the future, we intend to evaluate further the practicality of this approach. It would also be interesting to study how agents can be used to influence (as opposed to just monitor) resource behaviour depending on the environmental context.

Acknowledgements

The authors are grateful for the support provided by the EU Framework 7 project PRIME (Plug and Produce Intelligent Multi Agent Environment based on Standard Technology) [grant number 314762], and the EPSRC [grant number EP/K018205/1].

Disclosure statement

No potential conflict of interest was reported by the authors.

References

Alsafi, Yazen, and Valeriy Vyatkin. 2010. “Ontology-based reconfiguration agent for intelligent mechatronic systems in flexible manufacturing.” *Robotics and Computer-Integrated Manufacturing* 26 (4): 381–391.

Antzoulatos, Nikolas, André Rocha, Elkin Castro, Lavindra de Silva, Tiago Santos, Svetan Ratchev, and José Barata. 2015. “Towards a Capability-based Framework for Reconfiguring Industrial Production Systems.” *IFAC-PapersOnLine* 48 (3): 2077–2082.

Barata, José, Luís Camarinha-Matos, and Gonçalo Cândido. 2008. “A multiagent-based control system applied to an educational shop floor.” *Robotics and Computer-Integrated Manufacturing* 24 (5): 597–605.

Bellifemine, Fabio Luigi, Giovanni Caire, and Dominic Greenwood. 2007. *Developing Multi-Agent Systems with JADE (Wiley Series in Agent Technology).* Chichester, England: John Wiley & Sons.

Cândido, Gonçalo, and José Barata. 2007. “A multiagent control system for shop floor assembly.” In *Holonic and Multi-Agent Systems for Manufacturing,* edited by Vladimír Mařík, Valeriy Vyatkin, and Armando W. Colombo, 293–302. Regensburg, Germany: Springer.

de Silva, Lavindra, Paolo Felli, Jack C. Chaplin, Brian Logan, David Sanderson, and Svetan M. Ratchev. 2016. “Realisability of Production Recipes.” In *ECAI 2016 - 22nd European Conference on Artificial Intelligence, 29 August-2 September 2016, The Hague, The Netherlands - Including Prestigious Applications of Artificial Intelligence (PAIS 2016),* 1449–1457.

Dolgui, Alexandre, and Jean-Marie Proth. 2010. *Supply chain engineering: useful methods and techniques.* London, England: Springer Science & Business Media.

Elgh, Fredrik. 2007. *Computer-supported design for producibility: principles and models for system realisation and utilisation.* PhD Thesis, Chalmers University of Technology.

Fox, Maria, and Derek Long. 2003. “PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains.” *Journal of Artificial Intelligence Research* 20 (1): 61–124.

Frei, Regina, and Giovanna Di Marzo Serugendo. 2011. “Self-organizing assembly systems.” *IEEE Transactions on Systems, Man, and Cybernetics* 41 (6): 885–897.

Ghallab, Malik, Dana Nau, and Paolo Traverso. 2004. *Automated planning: theory & practice.* San Francisco, USA: Elsevier.

Järvenpää, Eeva. 2012. *Capability-based adaptation of production systems in a changing environment.* PhD Thesis, Julkaisu-Tampere University of Technology.

Kaluza, Bernd, and Thorsten Blecker. 2005. *Erfolgsfaktor Flexibilität: Strategien und Konzepte für wandlungsfähige Unternehmen.* Vol. 60. Berlin, Germany: Erich Schmidt Verlag GmbH & Co KG.

Koren, Yoram, and Moshe Shpitalni. 2010. "Design of reconfigurable manufacturing systems." *Journal of manufacturing systems* 29 (4): 130–141.

Landers, G. R., J. Ruan, and F. Liou. 2006. "Reconfigurable Manufacturing Equipment." In *Reconfigurable Manufacturing Systems and Transformable Factories,* edited by Anatoli I. Dashchenko, Berlin and Heidelberg, Germany, 79–110. Springer.

Malec, Jacek, Anders Nilsson, Klas Nilsson, and Sławomir Nowaczyk. 2007. "Knowledge-based reconfiguration of automation systems." In *International Conference on Automation Science and Engineering,* Arizona, USA, 170–175.

Monostori, László. 2014. "Cyber-physical production systems: Roots, expectations and R&D challenges." *Procedia CIRP* 17: 9–13.

Qiao, Lihong, Shuting Kao, and Yizhu Zhang. 2011. "Manufacturing process modelling using process specification language." *The International Journal of Advanced Manufacturing Technology* 55 (5): 549–563.

Ribeiro, Luis, José Barata, Dean Whittingslow, and Roland Krain. 2013. "Multiagent Mechatronic Systems with Simulation on the Loop." In *International Conference on Systems, Man, and Cybernetics,* Manchester, United Kingdom, 3842–3847. IEEE.

Ruml, Wheeler, Minh Binh Do, Rong Zhou, and Markus P. J. Fromherz. 2011. "On-line Planning and Scheduling: An Application to Controlling Modular Printers." *Journal of Artificial Intelligence Research* 40 (1): 415–468.

Scholz-Reiter, B, and M Freitag. 2007. "Autonomous processes in assembly systems." *CIRP Annals-Manufacturing Technology* 56 (2): 712–729.

Terkaj, Walter, Tullio Tolio, and Anna Valente. 2009. "A review on manufacturing flexibility." In *Design of Flexible Production Systems,* edited by Tullio Tolio, 41–61. Berlin and Heidelberg, Germany: Springer.

Timm, Ingo, and Peer-Oliver Woelk. 2003. "Ontology-based capability management for distributed problem solving in the manufacturing domain." In *Multiagent System Technologies,* edited by Michael Schillo, Matthias Klusch, Jörg Müller, and Huaglory Tianfield, 168–179. Berlin and Heidelberg, Germany: Springer-Verlag.

Winikoff, Michael. 2005. "Jack$^{\text{TM}}$ Intelligent Agents: An Industrial Strength Platform." In *Multi-Agent Programming: Languages, Platforms and Applications,* edited by Rafael H. Bordini, Mehdi Dastani, Jürgen Dix, and Amal El Fallah Seghrouchni, Boston, USA, 175–193. Springer US.