# Context-Aware Plug and Produce for Robotic Aerospace Assembly

David Sanderson[1[0000-0002-8675-4991]] and Emma Shires[1] and Jack C. Chaplin[2[0000-0003-3282-2386]] and Harvey Brookes[3] and Amer Liaqat[3] and Svetan Ratchev[2[0000-0001-9955-2806]]

[1] Centre for Aerospace Manufacturing, University of Nottingham, NG7 2PX UK
[firstname.lastname]@nottingham.ac.uk
[2] Institute for Advanced Manufacturing, University of Nottingham, NG8 1BB UK
[firstname.lastname]@nottingham.ac.uk
[3] Airbus UK
[firstname.lastname]@airbus.com

**Abstract.** Aerospace production systems face increasing requirements for flexibility and reconfiguration, along with considerations of cost, utilisation, and efficiency. This drives a need for systems with a small number of automation platforms (e.g. industrial robots) that can make use of a larger number of end effectors that are potentially flexible or multifunctional. This leads to the challenge of ensuring that the configuration and location of each end effector is tracked by the system at all times, even in the face of manual adjustments, to ensure that the correct processes are applied to the product at the right time. We present a solution based on a Data Distribution Service that provides the system with full awareness of the context of its automation platforms and end effectors. The solution is grounded with an example use case from WingLIFT, a research programme led by a large aerospace manufacturer. The WingLIFT project in which this solution was developed builds on the adaptive systems approach from the Evolvable Assembly Systems project, with focus on extending and increasing the aerospace industrial applicability of plug and produce techniques. The design of this software solution is described from multiple perspectives, and accompanied by details of a physical demonstration cell that is in the process of being commissioned.

**Keywords:** Aerospace Assembly, Context Awareness, Distributed Data Service, Flexible Manufacturing Systems, Manufacturing Service Bus, Multi-Agent Systems, Plug and Produce, Robotic Assembly.

## 1 Introduction

Global air travel has seen significant growth in recent decades, doubling every 15 years and demonstrating strong demand for new aircraft. This delivers a challenge to aircraft manufacturers to ramp up production to higher rates in order to keep pace with demand. One of the biggest challenges to the introduction of a new product is industrialising the new automation technologies that enable the required ramp up to

full production rate. The industrialisation process requires the development and validation of a wide variety of technologies in a short period of time. This can be accomplished at a reduced cost through flexible, reconfigurable, and modular production systems that enable multiple processes to be evaluated concurrently without large commissioning efforts. Such systems also enable reduction in cost by enabling adaptation in the automated system to cope with component variation and process uncertainty. This supports delivery of automated assembly processes (e.g. drilling, fastening etc.) to the correct location on the actual product. Such technology contributes to reduction in manufacturing costs, both non-recurring (e.g. production systems design and commissioning) and recurring (e.g. reduction in changeover, human intervention, and cycle time).

The WingLIFT project [1] has identified a number of specific aims in support of the industry requirements when considering the assembly of aircraft wings. One aspect of these aims is the application of innovative information management technology to optimise flow and distribution of both internal and external information across a wing sub-assembly factory. Intelligent assembly systems are required to monitor the key parameters of the entire manufacturing system, which supports quality assurance, geometric deviation awareness and control, and data feedback for real-time continuous improvement.

The work presented in this paper contributes to this technology solution specifically by monitoring the configuration of the system in real time. This technology is grounded in a specific use case that describes an automated assembly cell containing a set of process end effectors, each with a potentially large set of possible configurations, that are shared between a relatively small number of automation platforms.

This paper is organised as follows: Section 2 provides an overview of the current state of the art in flexibility in manufacturing systems, with particular reference to previous projects carried out at this institution in the area on which WingLIFT builds. Section 3 describes the motivating use case for the work, before Section 4 develops the architectural concept for WingLIFT. Section 5 specifies the demonstration scenario that will be used to validate the work, along with an outline of the solution developed. Finally, Section 6 summarises the work presented.

## 2 Flexibility in Manufacturing Systems

### 2.1 Flexible and Reconfigurable Manufacturing Systems

In a manufacturing sector characterised by market unpredictability, increased global labour costs, and growing consumer demand for highly personalised goods and services, producers are naturally pushed to remain competitive by maintaining shorter times to market, increased product diversity and specialisation, and shorter product lifecycles. This has resulted in a growing body of research into production systems that can incorporate new technologies and provide high levels of robustness, resilience, and responsiveness.

There are a number of approaches to delivering these characteristics, including flexible manufacturing systems [2, 3], reconfigurable manufacturing systems [4], automatic and adaptive control [5], and manufacturing systems modelling and simulation [6]. One specific technology that has been developed in this area is “plug and produce” [7], named by analogy to the concept of “plug and play” in computing. The EU FP7 PRIME project [8] developed a multi-agent approach to plug and produce with commercially available components for simple robotic assembly tasks in a fixed system dealing with dynamic product changes and unexpected disruptions [9–11].

### 2.2 Evolvable Assembly Systems, Context Awareness, and WingLIFT

EPSRC Evolvable Assembly Systems (EAS) [12] was a fundamental research project following on from PRIME. It aimed to deliver adaptable and cost effective manufacture by enabling a compressed product life cycle through the delivery of robust and compliant manufacturing systems that can be rapidly configured and optimised. In turn, this should enable the reduction of production ramp-up times and programme switchovers. In summary the project proposed a multi-agent system [13] to provide manufacturing control systems with the characteristics of agility, multi-functionality, adaptability, and resilience. Further information on the EAS project can be found elsewhere [14–17], but the remainder of this section focusses on how EAS viewed the concept of context-awareness in terms of intelligent production systems. One product of the EAS project was the concept of a context-aware cyber-physical production system, shown in **Fig. 1** and discussed in more detail in [18].

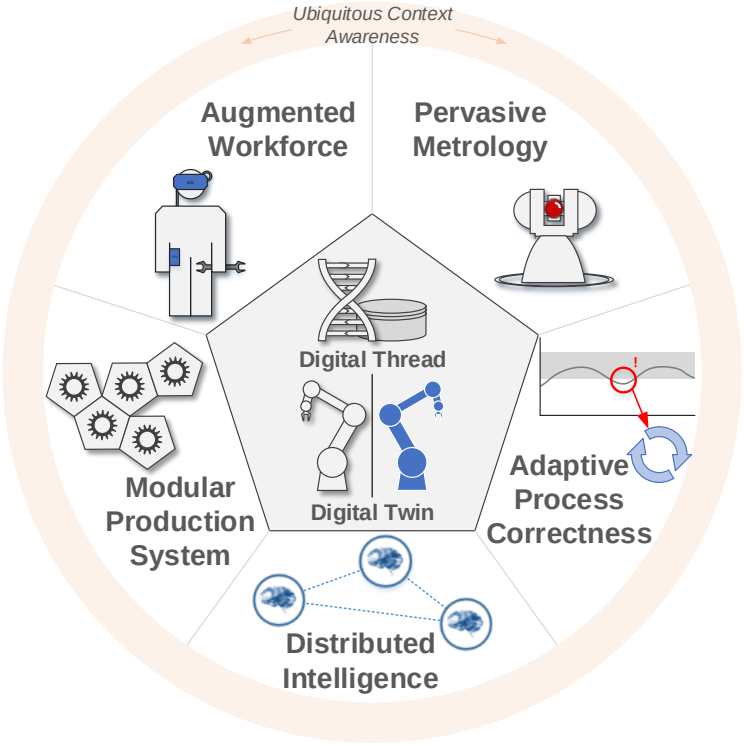


**Fig. 1.** Conceptual framework for Context-Aware Cyber-Physical Production Systems

The basis of the concept is that of a shared context for the system, where the context is knowledge about the system and its current state. This context allows all the elements of the system to share the relevant knowledge and information required to accomplish the system function. The foundation of the context is the digital information concerning the system, starting with the information created at design and commissioning ("digital twin"), and added to throughout life by the operation of the system ("digital thread"). This contextual information is gathered and handled by distributed intelligence across a set of modular system components. Pervasive metrology[1] allows for the highest quality information at all times. The information can be used to allow the system to self-adapt and maintain its correct functioning. The final aspect of the concept allows for the human workforce and the automation to work together most effectively by assigning decision-making and operations in a hybrid manner to best allocate responsibilities. In summary, the context-aware cyber-physical production systems concept enables the smart integration of all equipment in order to best leverage the available information and thereby accomplish the production aims in the most efficient manner possible with the available resources.

[1] Pervasive metrology is where metrology is used throughout the system, named by analogy with pervasive computing [31], an offshoot of ubiquitous computing [32].

WingLIFT focusses on a specific instantiation of part of this concept in order to prove its applicability in an industrial environment. The WingLIFT approach involves the aspects of data sharing that enable smart integration of dynamic modular production systems through the use of distributed intelligence. This distributed intelligence is presumed to be based on multi-agent technology, but an agent is not a "hard requirement" for every resource in the system, as will be discussed later. Other parts of the WingLIFT project include some aspects of pervasive metrology, but they will not be included in this paper.

# 3 Use Case

## 3.1 High-level Use Case Motivation

The high-level use case identified in the WingLIFT project for this work is as follows. A semi-automated aerospace assembly cell exists in which a small number of automation platforms (e.g. industrial robots) share a larger number of process end effectors (e.g. end effectors designed to either drill, fasten, seal, or position components). Each end effector may have a large number of possible configurations (e.g. for the drilling end effector, there may be a variety of cutting tool sizes; for the fastening end effector there may be a variety of fastener diameters and lengths; and so on). These configurations may be changed dynamically during operations either by the automation control system, or by the operator. Both the automation platforms and end effectors are expected to be moveable around the cell through the use of automatic tool changers and either rails or moveable platforms. The system therefore requires some method for maintaining knowledge of the current configuration state of the whole system, including the hardware configurations and positions of both automation platforms and end effectors, and the software configuration of the control system.

## 3.2 Specific Use Case Scenarios

Further developing the high-level use case described above, the project defines a set of specific use case scenarios. The scenarios are as follows:

- Resource addition and resource removal
- End effector pick-up and end effector drop-off
- Configuration change (hardware) and configuration change (software)

For the purposes of this paper, we will describe in detail the end effector pick-up use case and a generalised "configuration change" use case that combines both hardware and software changes. The project utilises a multifunctional UML approach (i.e. according to the Object Management Group's Unified Modeling Language [19]) to describe the use cases and solution; for this paper we will describe each use case with a table.

**End Effector Pick-up.** The end effector pick-up use case scenario shown in **Table 1** describes a situation where the system is required to change the currently equipped end effector on a given automation platform (in this case a robot) to a specific end effector. This requires that the system is aware of the locations of the robot and end effector, and also the current and desired configurations of the end effector.

**Table 1.** End effector pick-up use case scenario in tabular format

| **Name**: End Effector Pick-Up |
|---|
| **Actors** |
| • Existing production system<br>• Existing robot requiring end effector<br>• Existing end effector to be picked up<br>• Operator/integrator |
| **Pre-Conditions** |
| • System and all relevant resources are functioning correctly, support plug & produce, and have compatible interfaces.<br>• Robot is part of production system, is functioning correctly and has interfaces compatible with the end effector.<br>• End effector is part of production system, is functioning correctly and has interfaces compatible with the robot.<br>• End effector is not already on (any) robot, but is in a known location that is reachable by robot.<br>• End effector has (or can be given) definition of capabilities and configuration. |
| **Basic Flow** |
| 1. Start: System requires end effector to be added to robot.<br>2. System identifies current (or last known) location of end effector.<br>3. Robot and end effector are brought to the same location and connected together. Details of this physical process are out of scope, but this could happen in three ways:<br>a. Robot moves to end effector and picks it up.<br>b. End effector is brought to robot, which picks it up.<br>c. Robot and end effector move to same location, where robot picks up end effector.<br>4. Robot/system reads current configuration setting of end effector. The system becomes aware of the end effector configuration. The “Change Configuration” use case is executed – the configuration change is the joining of the robot and end effector.<br>5. Robot and end effector are now considered “joined”.<br>6. System checks that end effector configuration matches expected configuration and notifies operator of successful pick up. Either:<br>a. Configuration is as expected – success.<br>b. Configuration is not as expected – a change is required.<br>7. Completion: The end effector in the internal representation of the production system has been set as being joined to the representation of the robot: all resources in the system should be aware of the new capabilities/configuration and location/connectivity of the joint robot with end effector. |
| **Post-Conditions** |

- The new end effector is part of the production system (attached to the robot); system and all relevant resources are functioning correctly, support plug & produce, and have compatible interfaces.
- All resources in the system should be aware of the new end effector's capabilities/configuration and location/connectivity.
- The new end effector can be given commands and generate output as expected.

**Configuration Change.** The configuration change use case scenario shown in **Table 2** shows a situation where a change has been made to a resource (e.g. end effector) in the system that must be communicated to the rest of the system. Examples of this change could include an end effector being mounted to a given robot (which updates both the robot configuration and the end effector configuration), an end effector being placed in a tool rack (which updates the "location" configuration of the end effector), or an end effector setting being changed (for example the size of fastener being held).

**Table 2.** Configuration change use case scenario in tabular format

| **Name**: Configuration Change |
|---|
| **Actors** |
| • Existing production system<br>• Existing resource to be configured<br>• Operator/integrator |
| **Pre-Conditions** |
| • System and all relevant resources are functioning correctly and support being configured.<br>• Resource to be configured is part of production system and is functioning correctly, but is not in the correct configuration.<br>• Resource to be configured can be given definition of capabilities and configuration.<br>• The change to the configuration is such that the rest of the system needs to be made aware of the change (i.e. it will impact planning or production processes). |
| **Basic Flow** |
| 1. Start: A change is made to the configuration of a resource in the system that is significant enough to be communicated to the rest of the system. The details of what this configuration change is are out of scope. It may include changes to the settings on an end effector (e.g. which size bolt is loaded).<br>2. Resource reads new configuration. The details of this are out of scope, but examples include:<br>a. Some resource in the system – or the system controller – requested a configuration change be applied, so already communicated the change to the resource.<br>b. The resource automatically detects the configuration change; this could be through specific sensors or because the resource automatically determined the required configuration change so is already aware of it.<br>c. The operator adjusts hardware selector switches on the resource, which are read by the resource controller.<br>d. The operator inputs the configuration change on an HMI, which is read by the resource controller or system (which would then pass it to the resource controller). |

3. The resource updates its internal representation to reflect the new configuration.
4. The resource communicates its new configuration to the system, which notifies all relevant resources and incorporates the change into its internal representation.
5. Completion: The resource has its new configuration reflected in its internal representation. The system and all relevant resources are aware of the new configuration. This new information may be used by other resources as a trigger for other processes.

**Post-Conditions**

- System and all relevant resources functioning correctly and support configuration.
- Resource has new configuration; remainder of system is aware of new configuration as appropriate.
- Resource to be configured can be given definition of capabilities and configuration.

# 4 Reference Architecture Concept

## 4.1 Generic Process Flow

Based on the use cases developed in the project discussed in Section 3, a generic process flow has been identified and is presented diagrammatically in **Fig. 2** using the example of the "end effector pick-up" use case. This generic process flow allows a solution to be designed that will address the range of problem scenarios facing the system, where each use case is a specific example, rather than designing a solution specifically for each use case and then attempting to combine them. Each use case follows the following process:

1. **Trigger**: An event occurs to trigger the use case. In this case it is a requirement for the end effector B to be fitted to the robot A.
2. **Handle**: An entity in the system either chooses to handle the event, or is assigned to handle it. In this case the WingLIFT software will gather the required data (e.g. end effector location) from the distributed system context and assign the specific required processes to the relevant robot controller.
3. **Update**: The system configuration is updated. This happens both at the local level by each individual resource, and at the network level where the shared context is updated. The robot controller configuration will be updated to include the new end effector, and the end effector context will reflect its new location and configuration.
4. **Notify**: Once the use case has been completed, one or more entities in the system are notified of the success or failure of the process. In this case, the operator is notified through an HMI.

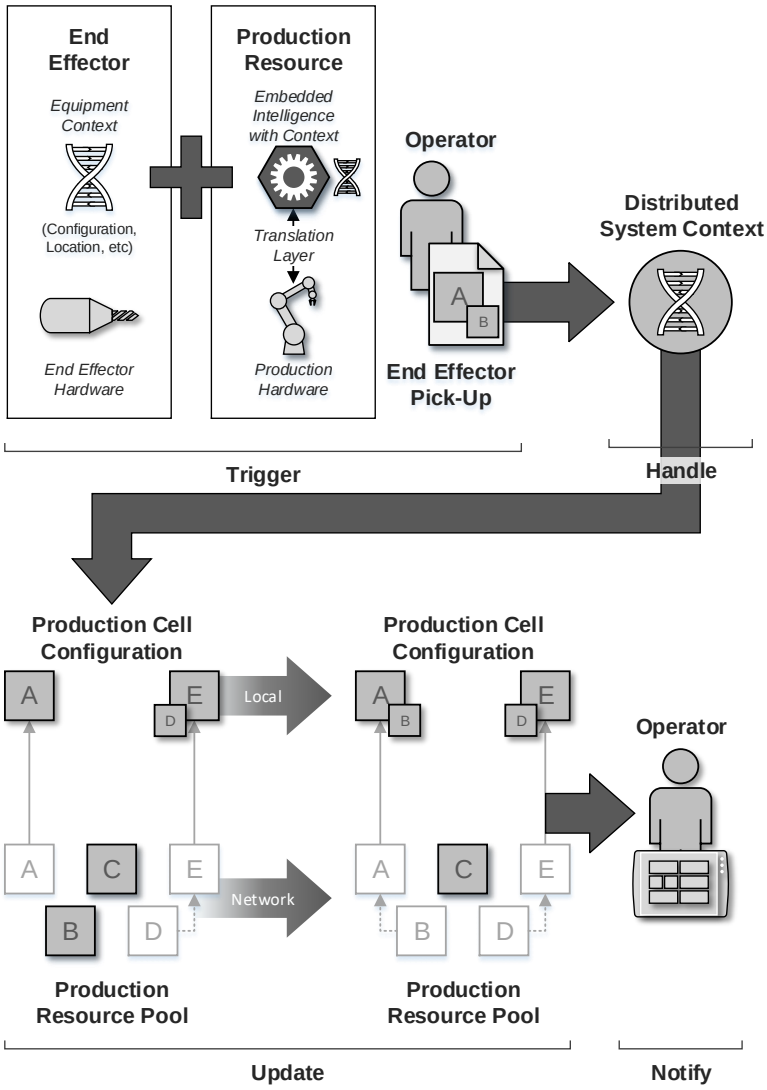


**Fig. 2.** Generic use case flow for WingLIFT, using the example of end effector pick-up

## 4.2 Architectural Concept

Placing some of the terms used in the previous section into context, the top-level agent-oriented architectural concept, based on that of the EAS project, is shown in **Fig. 3**. Each resource maintains a local internal model for low-level decision-making and control. All resources in the system are connected to a shared context which forms a "joint model" from the many local internal models. High-level decision-

making can be performed on this joint model. The shared context is also the link to the wider enterprise and any additional external data sources or storage locations.

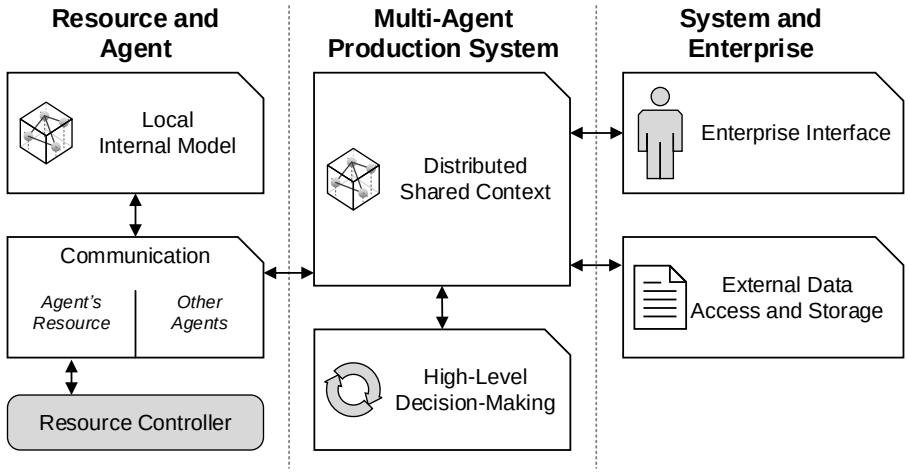


**Fig. 3.** Top-level agent-oriented architectural concept

Building on the EAS project, it is assumed that the majority of the resources in the system will be controlled by intelligent agents [13]. In Section 4.4, integration cases will be discussed where no agent is present.

### 4.3 Data Communications Concept

The "distributed shared context" in the system is transmitted over a databus that performs the functions of both a Manufacturing Service Bus and of an Enterprise Service Bus (a hybrid "manufacturing/enterprise service bus"). This "shared context" is effectively all relevant data that is collected from the system; it is a joint model of the system context that is generated by the collection of the internal models of all the system elements. Once on the bus, this data can be used as required by the actors in the system, or stored for later use.

In terms of implementation, all inter-agent (or inter-resource) communication in the system is handled by a publish-subscribe databus implemented as a Data Distribution Service (DDS, as specified by the Object Management Group (OMG) DDS standard) [20]. The publish-subscribe approach is one in which, rather than the data publisher sending data to one or more specific recipients, the data is published to a channel. The data subscribers subscribe to a channel and receive data from that channel. This frees publishers from keeping track of who is interested in their data, and subscribers from keeping track of data sources. The channel can also take care of communication implementation details, rather than requiring the agents to do so. Such details may include quality of service requirements, caching or persistence, and so on.

### 4.4 Hardware / Software Stack

Physical hardware resources in the WingLIFT architecture are usually controlled by intelligent agents deployed on embedded computers. Each agent is connected to the resource using a resource-specific translation layer and to the rest of the system using a DDS. In the EAS project this was the most common type of stack, but WingLIFT aims to address a wider range of resource types. **Fig. 4** shows this more complete view: the physical resource may provide a software interface, or may require a hardware interface; the system can also interface with humans though software running on portable devices; and the system may include software services running on other computing hardware. There should be no difference to the architecture whether the resource being managed is physical automation, a human, or a software service.

Also shown in **Fig. 4** are the data flows from the resources up to the databus, the options for where each piece of functionality is deployed in hardware terms, and the likely division between provided and developed functionality. Resource hardware is controlled by its own controller as normal. That controller then interfaces with the system databus either through an agent on an embedded computer, an agent deployed directly on the controller (in the case of a software PLC for example), or directly through its own network application programming interface (API). In most cases some translation will be required from vendor specific semantics to open common semantics for use on the databus. Some resource hardware will provide an API and open communication standards for interfacing with. Some “legacy” resource hardware may require a more complex translation layer that features hardware technical interconnectivity as well as semantic translation. Human resources communicate with resource agents via human-machine interfaces (HMIs). In the case of resources that directly connect to the databus through their own network API without an agent, they can only act as publishers and/or subscribers of data. If any decision-making takes place, it must either be handled entirely inside the resource, or a separate agent must be deployed on the databus to act on the data published and/or subscribed to by the resource.

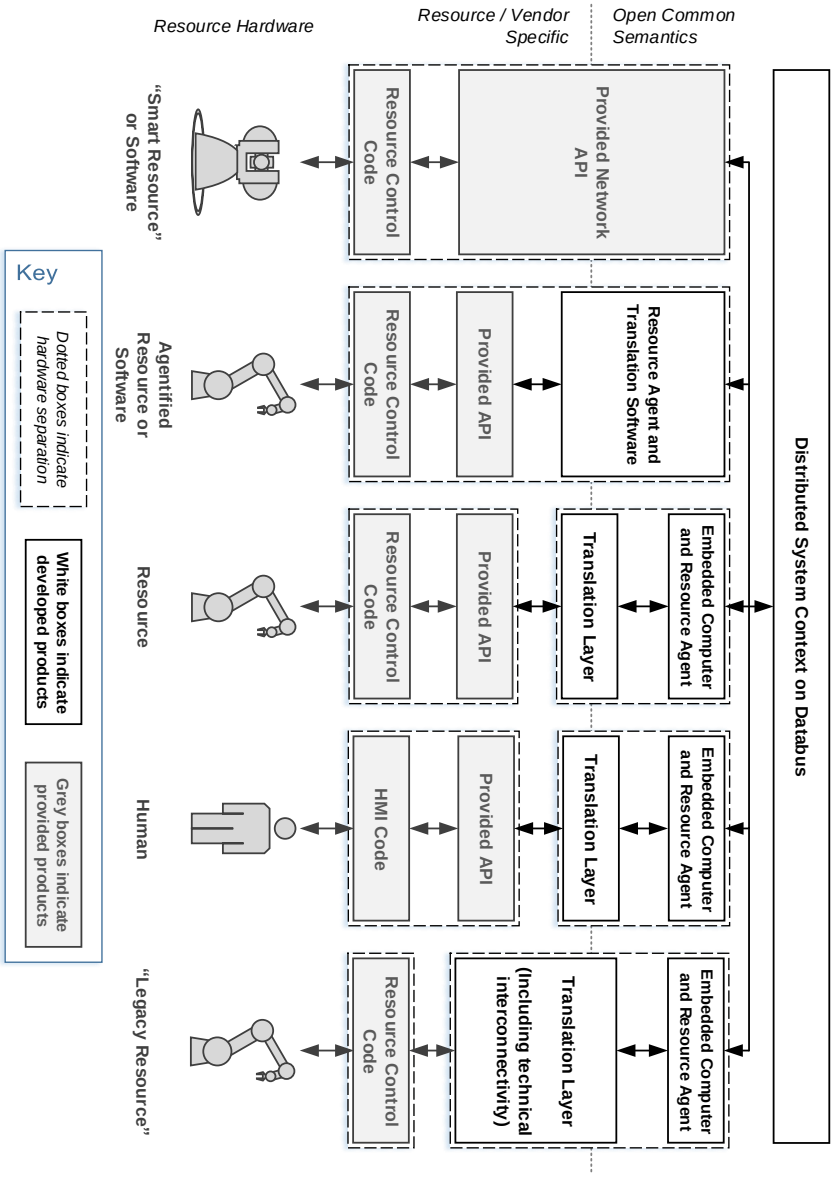


**Fig. 4.** Integration approaches for different resources types

# 5 Validation

## 5.1 Demonstration Scenario

In order to validate the proposed solution, we have developed a demonstration scenario based around the project use cases. In this scenario, two robots share a number of end effectors through the use of automatic tool changers as described in Section 3.1.

The aim of the demonstrator is to show how the two robots can share the end effectors and how the system can maintain awareness of the current configuration of both the robots and the end effectors in the face of manual changes. A visual representation of an example usage flow of the demonstrator is given in **Fig. 5** in the format of a UML activity diagram.

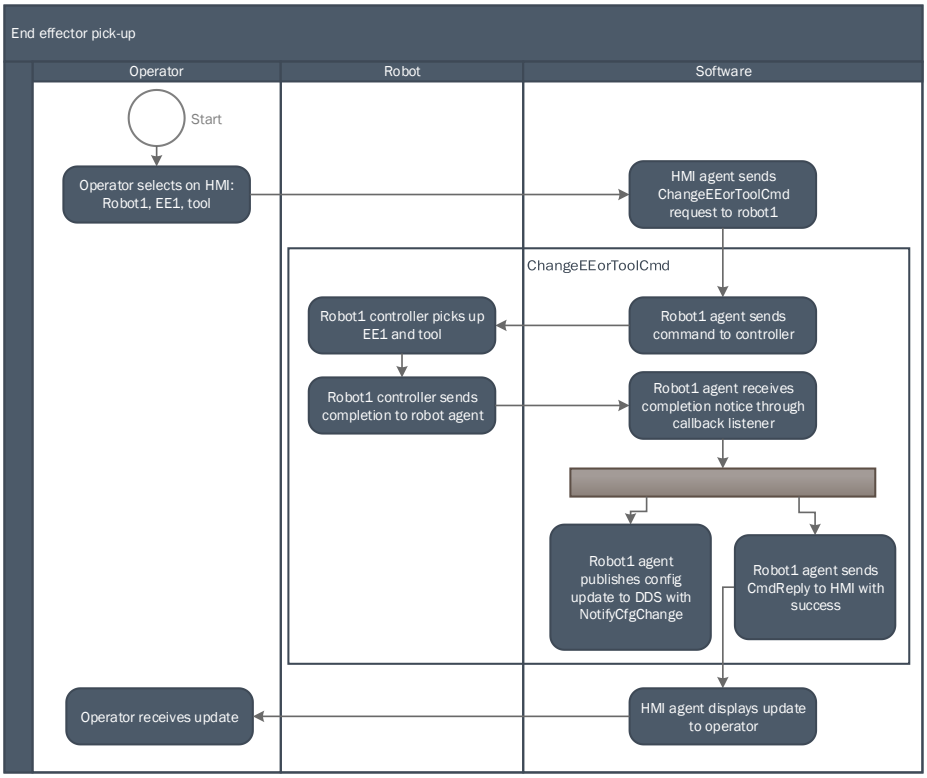


**Fig. 5.** Example flow of the demonstrator in UML activity diagram format

This shows the example of an end effector pick-up: first the use case is **triggered** by the operator making a selection on the HMI. This is **handled** by the software once it receives the ChangeEEorToolCmd request. Once the end effector has been picked up, the status is **updated** locally (by the robot once the task is complete) and at the network level (when the robot publishes the context update). Finally, the operator is **notified** once the change is complete.

### 5.2 Outline Solution

Based on the WingLIFT architecture, databus concept, and integration approaches all described in Section 4, **Fig. 6** shows the overall logical structure of the demonstration cell described in this section. Each robot is controlled by its own controller, which is in turn connected to a WingLIFT agent that is connected to the DDS databus. Also connected to the databus is an HMI for the operator, and an agent connected to each end effector. The end effector information is transmitted onto the bus through this agent. The bus carries all configuration and state information for all resources in the cell: the location of the robots and end effectors, the configuration state of all end effectors, and which end effector is connected to each robot. The remainder of this section describes the implementation of the cell in some more detail.

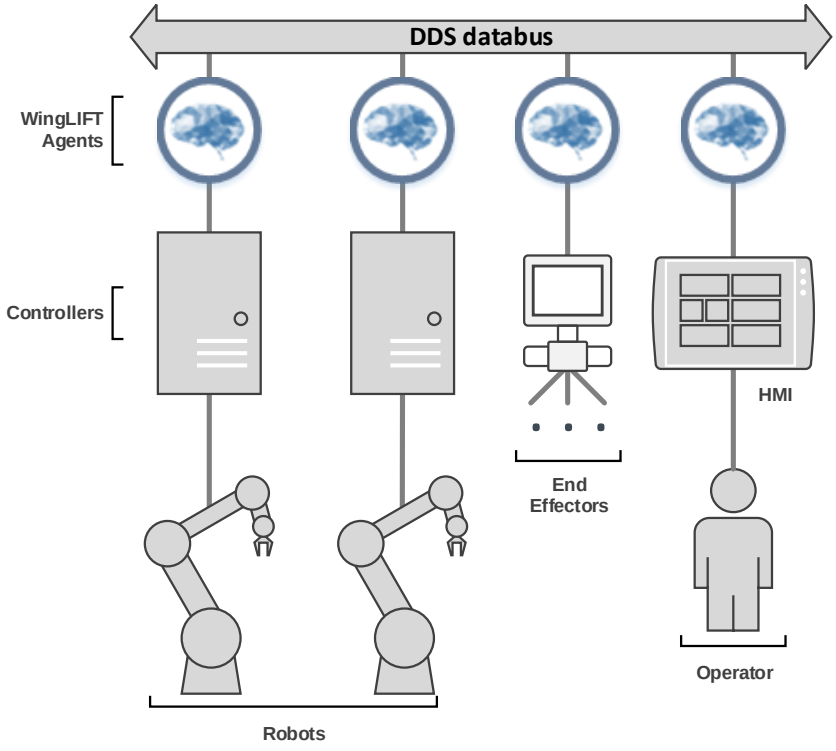


**Fig. 6.** WingLIFT demonstration cell architecture

**Resource, Controller, and Code.** Production resources must be orchestrated by the WingLIFT software. This is effectively a case of virtualising or servitising the production resource. Various approaches are possible, depending on the specific situation [21–23]: Industry 4.0 suggests an "administration shell" wrapper around the existing control approach for example [24–26]. Some modern resources may already have suitable interfaces. Others may facilitate the addition of a custom interface (e.g. a soft

PLC on which a module can be installed to communicate with the agent). WingLIFT uses an intelligent agent to virtualise each resource.

**Agent.** The WingLIFT architecture uses an agent-based approach to integrate resources that do not support intelligent integration. Each agent is based around a streamlined version of the JADE agent platform [27]. The JADE Abstract Architecture and Application frameworks have been utilized, but the Agent Communication and Agent Message Transport has been replaced by the RTI Connext DDS Pro publish-subscribe databus. Each WingLIFT agent extends the abstract JADE agent class, to provide structure for agent data management and behaviours.

**Communication.** The WingLIFT agent communicates along two main channels: via the agent-resource translation layer, and via the inter-agent communication method (e.g. DDS):

- Agent-resource translation layer: This translation layer allows the agent to communicate with the resource it is controlling. Where the resource is an HMI, it also allows the system to communicate with the human operator. While the agent side of this translation layer is common to all agents, the resource side of the layer must be customised to some degree to the specific (type/brand/make/model of) resource. This allows the agent to deal with the specific datatypes required by whatever interface is available in the resource in order to trigger operations and receive data.
- Inter-agent communication: Agents communicate with each other across a publish-subscribe RTI Connext DDS Pro databus [28]. This is based on the OMG DDS standard [20], with individual message formats defined in turn using the OMG Interface Definition Language (IDL) [29], part of Common Object Request Broker Architecture (CORBA) [30]. Connext Pro allows both the use of standard publish-subscribe communication and also request-reply messaging patterns as well. We use the request-reply pattern where appropriate, for example when sending commands to specific resources and receiving the responses.

## 6 Summary and Acknowledgements

This paper has presented the WingLIFT approach to context-aware plug and produce for flexible robotic aerospace assembly based on a Data Distribution Service. This approach allows the system to accurately and dynamically track the configuration and status of the process end effectors in a flexible and reconfigurable production cell. This is grounded in an industrial use case where the potential end effector configurations far outnumber the available automation platforms. The solution is presented as a set of design specifications along with a summary of a physical demonstration cell based at the University of Nottingham.

The authors gratefully acknowledge the support provided by Innovate UK through the Aerospace Technology Institute WingLIFT project (project reference 113162).

## References


1. UK Research and Innovation (2017) Wing Lean Innovative Future Technology (Wing LIFT) - project reference 113162
2. Sethi A, Sethi S (1990) Flexibility in manufacturing: A survey. Int J Flex Manuf Syst 2:289–328 . doi: 10.1007/BF00186471
3. Browne J, Dubois D, Rathmill K, et al (1984) Classification of flexible manufacturing systems. FMS Mag 2:114–117
4. Koren Y, Heisel U, Jovane F, et al (1999) Reconfigurable Manufacturing Systems. CIRP Ann - Manuf Technol 48:527–540 . doi: 10.1016/S0007-8506(07)63232-6
5. Monostori L, Csáji BC, Kádár B, et al (2010) Towards adaptive and digital manufacturing. Annu Rev Control 34:118–128 . doi: 10.1016/j.arcontrol.2010.02.007
6. ElMaraghy H, AlGeddawy T, Azab A (2008) Modelling evolution in manufacturing: A biological analogy. CIRP Ann - Manuf Technol 57:467–472 . doi: 10.1016/j.cirp.2008.03.136
7. Arai T, Aiyama Y, Maeda Y, et al (2000) Agile Assembly System by “Plug and Produce.” CIRP Ann - Manuf Technol 49:1–4 . doi: 10.1016/S0007-8506(07)62883-2
8. PRIME (2014) PRIME project website. http://www.prime-eu.com. Accessed 15 Nov 2014
9. Antzoulatos N, Castro E, Scrimieri D, Ratchev S (2014) A multi-agent architecture for plug and produce on an industrial assembly platform. Prod Eng 8:773–781 . doi: 10.1007/s11740-014-0571-x
10. Antzoulatos N, Rocha A, Castro E, et al (2015) Towards a Capability-based Framework for Reconfiguring Industrial Production Systems. In: Proceedings of the 15th IFAC/IEEE/IFIP/IFORS Symposium INCOM’15 on Information Control Problems in Manufacturing. pp 2145–2150
11. Antzoulatos N, Castro E, de Silva L, Ratchev S (2015) Interfacing Agents with an Industrial Assembly System for “Plug and Produce”: (Demonstration). In: Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems. International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC, pp 1957–1958
12. Ratchev S (2012) Evolvable Assembly Systems - Towards open, adaptable, and context-aware equipment and systems (grant reference EP/K018205/1)
13. Wooldridge M, Jennings NR (1995) Intelligent agents: Theory and practice. Knowl Eng Rev 10:115–152
14. Chaplin JC, Ratchev SM (2018) Deployment of a Distributed Multi-Agent Architecture for Transformable Assembly. In: Proceedings of the Eighth International Precision Assembly Seminar
15. Chaplin JC, Bakker OJ, de Silva L, et al (2015) Evolvable Assembly Systems: A Distributed Architecture for Intelligent Manufacturing. IFAC-PapersOnLine 48:2065–2070 . doi: 10.1016/j.ifacol.2015.06.393
16. Sanderson D, Chaplin JC, Ratchev S (2019) Functional modelling in evolvable assembly systems. In: IFIP Advances in Information and Communication Technology. Springer, Cham, pp 40–48
17. de Silva L, Felli P, Chaplin JC, et al (2017) Synthesising Industry-Standard

Manufacturing Process Controllers. In: Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS. International Foundation for Autonomous Agents and Multiagent Systems, pp 1811–1813

18. Sanderson D, Chaplin JC, Ratchev S (2018) Conceptual Framework for Ubiquitous Cyber-Physical Assembly Systems in Airframe Assembly. IFAC-PapersOnLine 51:417–422 . doi: 10.1016/J.IFACOL.2018.08.331
19. Object Management Group Unified Modeling Language, UML. http://www.uml.org
20. Object Management Group Data Distribution Service Specification (current). http://www.omg.org/spec/DDS/Current
21. Van Brussel H, Wyns J, Valckenaers P, et al (1998) Reference architecture for holonic manufacturing systems: PROSA. Comput Ind 37:255–274 . doi: 10.1016/S0166-3615(98)00102-X
22. Leitão P, Restivo F (2006) ADACOR: A holonic architecture for agile and adaptive manufacturing control. Comput Ind 57:121–130 . doi: 10.1016/j.compind.2005.05.005
23. Alam KM, El Saddik A (2017) C2PS: A digital twin architecture reference model for the cloud-based cyber-physical systems. IEEE Access. doi: 10.1109/ACCESS.2017.2657006
24. Bundesministerium für Bildung und Forschung (2014) Industrie 4.0 - Innovationen für die Produktion von morgen
25. Wagner C, Grothoff J, Epple U, et al (2018) The role of the Industry 4.0 asset administration shell and the digital twin during the life cycle of a plant. In: IEEE International Conference on Emerging Technologies and Factory Automation, ETFA. pp 1–8
26. Contreras JD, Garcia JI, Pastrana JD (2017) Developing of industry 4.0 applications. Int J Online Eng 13:30–47 . doi: 10.3991/ijoe.v13i10.7331
27. Bellifemine F, Poggi A, Rimassa G (1999) JADE --- A FIPA-compliant agent framework. Proc Fourth Int Conf Pract Appl Intell Agents Multi-Agent Technol
28. RTI RTI Connext DDS Professional. https://www.rti.com/products/connext-dds-professional. Accessed 16 Sep 2019
29. Object Management Group (2018) Interface Definition Language Specification v4.2. https://www.omg.org/spec/IDL/4.2/PDF. Accessed 16 Sep 2019
30. Object Management Group OMG IDL | CORBA. https://www.corba.org/omg_idl.htm. Accessed 16 Sep 2019
31. Henricksen K, Indulska J, Rakotonirainy A (2002) Modeling Context Information in Pervasive Computing Systems. Pervasive Comput Lect Notes Comput Sci 2414:167–180 . doi: 10.1007/3-540-45866-2_14
32. Weiser M (1991) The Computer for the 21st Century. Sci. Am. 265:94–104