The International Journal of Advanced Manufacturing Technology
https://doi.org/10.1007/s00170-021-07028-z

ORIGINAL ARTICLE

Design of a self-learning multi-agent framework for the adaptation of modular production systems

Daniele Scrimieri[1] · **Shukri M. Afazov**[2] · **Svetan M. Ratchev**[3]

Received: 5 November 2020 / Accepted: 29 March 2021

Abstract
This paper presents the design of a multi-agent framework that aids engineers in the adaptation of modular production systems. The framework includes general implementations of agents and other software components for self-learning and adaptation, sensor data analysis, system modelling and simulation, as well as human-computer interaction. During an adaptation process, operators make changes to the production system, in order to increase capacity or manufacture a product variant. These changes are automatically captured and evaluated by the framework, building an experience base of adjustments that is then used to infer adaptation knowledge. The architecture of the framework consists of agents divided in two layers: the agents in the lower layer are associated with individual production modules, whereas the agents in the higher layer are associated with the entire production line. Modelling, learning, and adaptations can be performed at both levels, using a semantic model to specify the structure and capabilities of the production system. An evaluation of a prototype implementation has been conducted on an industrial assembly system. The results indicate that the use of the framework in a typical adaptation process provides a significant reduction in time and resources required.

Keywords Self-learning · Adaptation · Agents · Assembly · Architecture

1 Introduction

To stay competitive, today's manufacturing enterprises must respond quickly to ever-changing requirements of volatile global and local markets. These requirements are dictated by more demanding customers and the development of advanced technology. As the number of product variants increases due to mass customisation and product life cycles become shorter, manufacturing systems must be designed to be highly reconfigurable and adaptive in order to handle product evolution and variety. Reusability of equipment can help to exploit unused potential and plays a major role in the provision of sustainable solutions. However, adapting existing equipment for new uses may require a considerable amount of knowledge and skills, integration efforts and appropriate planning. Due to the uncertainty regarding these elements and the lack of documentation about the capabilities and life cycle of existing systems, adaptation and reuse are not always feasible. For these reasons and because of general intricacies of adaptation processes themselves, manufacturing firms often take the decision to design new production systems, rather than adapting existing ones. This is the case, in particular, with regard to assembly systems [1, 2].

This paper presents the design of FRAME (Fast Reconfiguration in an Adaptive Manufacturing Environment), a self-learning framework for capturing adaptation knowledge and supporting engineers throughout an adaptation process. The framework consists of a multi-agent system whose agents can operate either at a local or global level, that is, by gathering sensor data or learning how to adapt, respectively, a single module or station, or the entire production line. The framework is not tailored to a particular

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

Shukri M. Afazov
shukri.afazov@ntu.ac.uk

Svetan M. 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

Published online: 17 May 2021

production system, but rather it can be customised for specific systems and it provides the tools necessary for doing that. The specialisation for one production system involves the definition of a semantic model describing the system in terms of structure and capabilities. One important objective of our work is minimising the software development required to deploy the multi-agent system on a new production system. The effort is, instead, mostly put on the software configuration and modelling side. This research is in line with the goals of the fourth industrial revolution (Industry 4.0) and the vision of *smart factories* [3], which promote higher levels of automation, intelligence and decentralised control in manufacturing systems.

The automatic generation of knowledge in FRAME offers significant advantages over manual approaches to the construction of a knowledge base for adaptive manufacturing systems. Firstly, since the generated knowledge is built automatically from machine data, there is no need to (1) elicit it from the domain experts (i.e. mechanical and manufacturing engineers, system integrators, shop floor operators) and (2) find a suitable representation. Both the elicitation and representation tasks are problematic because they produce results that depend on the ability of the knowledge engineers in capturing and representing useful knowledge. Secondly, although the generated knowledge is built from specific cases, its applicability is not limited only to those cases. Indeed, it can be used in future contexts and scenarios similar to those encountered in the past. FRAME uses the ObjectLogic language [4] for defining and querying production system models, whereas XML is used for agent communication and for serialising events, machine states and experience instances. The learning technique is based on a k-nearest neighbour classification algorithm, which represents adaptation contexts as points in a multidimensional space.

The rest of the paper is organised as follows. Section 2 analyses related research. Section 3 presents the design of the FRAME architecture. Section 4 introduces the self-learning technique implemented in FRAME. Section 5 describes the experiment and evaluation conducted. Section 6 draws some conclusions.

2 Background and related work

Despite the development of self-organising intelligent systems that are able to perform individual logical adaptations autonomously, there is not yet a full solution to support an entire adaptation process from start to finish. This process is often largely human-driven, primarily based on the experience of system integrators. While there are methods and tools for solving specific problems, these methods and tools are not integrated into a general framework that can be used in a wide range of scenarios. An adaptation framework comprising information models and automated learning mechanisms to capture adaptation knowledge is required in order to facilitate knowledge sharing and support decision-making.

Related work in this direction includes a capability-based methodology for adaptation planning, with the objective of developing tools for the rapid reconfiguration of production systems [1]. However, this methodology has only been applied in an academic research environment and not yet in an industrial context. Also, the computational processes associated with this ontological approach are not investigated. Research on the computational aspects includes experience-based learning techniques, using classification algorithms, for the adaptation of assembly systems [5]. Adaptation of plug and produce systems is discussed by [6]. In plug and produce systems, identification and configuration of new devices is performed with minimal human intervention. One particular objective of automated learning in intelligent manufacturing systems is accelerating the production ramp-up phase, as investigated by [7]. These works do not describe the design of the whole software system, though, or how to create a framework of general application, which is one of the aims of this paper. A reinforcement learning approach, guided by human experts, for the production ramp-up problem is presented by [8]. The results of this last work indicate, as anticipated, that an exploration strategy guided by a human operator is more efficient than one that is purely algorithmic.

2.1 Manufacturing paradigms enabling system changes

Different manufacturing paradigms, together with physical and logical enablers, have been introduced to facilitate system changes [9]. The paradigms include flexible and reconfigurable manufacturing [10, 11], holonic and agent-based manufacturing (see, for example, the ADACOR architecture by [12]) and evolvable assembly systems [13]. A recent review of reconfigurable manufacturing systems is conducted by [14]. Reviews of agent-based manufacturing are presented by [15] and [16]. The contributions of holonic manufacturing to Industry 4.0 are examined by [17]. Although agent-based and holonic manufacturing provide clear benefits, in particular in terms of flexibility and robustness, there are still barriers to their widespread industrial adoption. The fact that there is no guarantee on the operational performance of agent-based solutions prevents them from being applied to real-time control problems and is an obstacle to their acceptance by the management of companies.

System adaptations can be made to increase production, improve a production process, or produce a product variant.

They can be either physical or logical, each having different objectives. Physical adaptations are aimed at increasing production capacity or producing a product variant, and include operations such as the addition or modification of modules or machines. Logical adaptations are aimed at improving the utilisation of available resources, do not require hardware alterations and involve changing parameters or reprogramming to implement, for instance, a different scheduling or routing policy.

In particular, logical adaptations can be determined and applied autonomously by intelligent agents. Multi-agent systems exhibit many self-* capabilities and self-organising multi-agent mechatronic systems have been developed in successful academic and industrial projects [18], with typical applications including manufacturing execution systems [19, 20], routing and scheduling [21]. Self-organising behaviour is also characteristic of some natural systems, which influenced the development of new computational techniques. A recent application of multi-agent systems is, for example, distributed diagnosis with bio-inspired algorithms [22]. A holonic manufacturing execution system with control mechanisms inspired by natural systems is presented by [23]. The integration of a centralised scheduling system based on constraint programming with a holonic manufacturing execution system is described by [24]. Hybrid solutions like this one are worthwhile investigating because they benefit from the strengths of both approaches. The holonic systems of [23] and [24], along with many others, implement the popular reference architecture PROSA (see [25] for a historical perspective). It would be interesting to compare different architectures or even different implementations of the same reference architecture, defining criteria for their evaluation. Also, there is a need for agent-based design patterns, similarly to object-oriented design patterns.

Another concept related to distributed intelligence is product-driven automation. Two heterarchical architectures for intelligent products are presented by [26]. Product intelligence seeks to give customers greater control over the processing of orders [27].

2.2 Agents and cyber-physical systems

Agent-based manufacturing systems have several distinctive characteristics in common with cyber-physical systems (CPS), as they are both intelligent and distributed systems offering high levels of adaptability through the cooperation of interconnected entities. As a matter of fact, a multi-agent system can be used to implement the cyber part of a CPS. A survey of the adoption of agents in industrial CPS is conducted by [18].

According to the US National Science Foundation, CPS are "engineered systems that are built from, and depend upon, the seamless integration of computation and physical components". Although systems combining physical and computational elements have long been in existence, the design of one type of elements has normally met only some minimum interface requirements and not taken full advantage of the capabilities of the other type. To integrate cyber and physical systems, the approaches of *cyberizing* the physical and *physicalizing* the cyber are identified by [28]. Some research work in this direction leverages the multi-agent paradigm. The integrated development of multi-agent PLC-based control systems using IEC 61131-3 is discussed by [29]. The combination of mechatronic production systems and multi-agent systems is investigated by [30]. What is missing in many works is a general methodology for designing agent-based control systems, which would facilitate new developments and applications. A model-based methodology (DACS) to enable an ordinary engineer to design an agent-based control system is described by [31]. More research in this direction is required to promote the adoption of multi-agent systems in CPS.

3 Architectural design

FRAME consists of a number of distributed agents and auxiliary software components running on PCs, PLCs or industrial PCs. The functionalities implemented include self-learning and adaptation, sensor data analysis, system modelling and simulation, as well as human-computer interaction. The framework provides general implementations of the agents, in the sense that they are not specialised for one industrial application but, instead, offer functions that are useful to address typical automation problems. The use of the agents for a particular application requires the specification of application-dependent details, including communication protocols, system architecture and capabilities, production process, sensor data and configuration parameters, performance requirements. FRAME agents can be deployed on individual modules or stations, or on the entire system.

Sensor data are used to evaluate the performance of the production system and how it responds to changes. These data are used to calculate a KPI (Key Performance Indicator), which can be defined in the framework itself as a function of finished product quality or throughput. For example, the performance could be measured by the cycle time. In this case, sensor data should include the time when a part is first acted upon and the time when the finished product is complete. The physical sensors that produce these data could be photoelectric sensors using a beam of light to detect the presence of parts on a conveyor belt at the beginning and end of the production process. Another example is vision systems with cameras and inspection

software for feature detection. There are no requirements for the physical sensors that can be used with FRAME, provided that the associated sensor data are defined in a semantic model and their semantics is shared by the agents.

From a software development perspective, FRAME is designed in an object-oriented fashion, with agents implemented by objects of classes defined by the framework. The specialisation of the agents for one application involves the derivation of new classes from those offered by the framework. In addition to the main agent classes, the framework contains other classes representing entities such hardware resources, agent capabilities, production objectives, sensor data, communication messages, experience and adaptation knowledge. All these classes can be reused, either by inheritance or composition, in different specialised agents.

The communication infrastructure of the framework provides two communication mechanisms between agents: synchronous and asynchronous. The former is used to ask agents for information or to request the execution of operations. This mechanism is implemented using web services. The latter is used to notify agents when events occur. This mechanism is implemented using a publish-subscribe pattern.

3.1 Agent types and roles

Figure 1 gives an overview of the framework architecture. The various components and their relationships will be explained in the next subsections. The following types of agents are identified:

- ER agent: the Experience Recognition agent captures the adjustments being made and stores them in an *experience base*, which contains all the changes and their impact on performance (Section 3.2).
- LEARN agent: the self-learning agent offers the self-learning function. This is applied to the machine state received from the ADAPT agent and the experience base (Section 3.3).
- ADAPT agent: the adaptation agent captures the current machine state and queries the LEARN agent for applicable adjustments (Section 3.3).
- HMI agent: the human-machine interface agent provides an intelligent interface between the user (the engineer or system integrator making changes) and the agent framework. It presents the user with a ranked list of adjustments produced by the ADAPT agent and guides the user through an adaptation (Sections 3.3 and 3.4).

Multiple instances of an agent type can be created and deployed on different physical components or layers, as discussed in Section 3.5.

3.2 Experience recognition

Experience recognition (ER) is the process of recording the adjustments being made to the machine and evaluating their impact. An adaptation usually involves a number of adjustments. By adjustment we mean an atomic change that cannot be broken down into separate changes. Examples of adjustments are changing pick-and-place points, cam angles, speed, pressure, grippers or pallet geometry, or even using different part styles. Some adjustments can be performed through software (e.g. pick-and-place points), while others require physical operations. In the former case, the adjustments are captured by the framework automatically. In the latter case, they must be entered manually. The effect of an adjustment must be measurable by the KPI, in the sense that a difference in performance should be reflected in the KPI value. If this is not the case, the self-learning technique cannot be applied.

The ER process is triggered by adjustment events. Events are structured data defined in the semantic model (Section 3.6) and generated by the machine or by agents, and transmitted in XML format using a publish-subscribe pattern. The ER agent subscribes to adjustments events, i.e. it listens to this type of events. Figure 2 shows an example of adjustment event serialised into XML. The various `FrameKeyModel` elements uniquely identify the parameter being modified by referring to entities defined in the semantic model. Experience can be created in two different ways: an experience instance can be created for each individual adjustment or for an adjustment session. In the former case, when the ER agent receives an adjustment event it queries the *event base*, a deductive and object oriented data store of all the generated events, for the state events generated by the machine before and after the adjustment. These events are used to construct a representation of the state of the machine before and after the adjustment. A machine state is represented by a list of attribute-value pairs in the experience base. Figure 3 shows an example of state event serialised into XML. The ER agent then calculates the KPI on both states and creates an experience instance with this information in the experience base. In the latter case (Fig. 4), a "start adjustment session" event signals that a series of adjustments is going to take place. When the ER agent receives this event, it captures the state of the machine at that point and calculates the KPI, then it records and groups all the adjustment events generated until it receives an "end adjustment session" event. At that point, the ER agent captures the state of the machine again, calculates the KPI and creates an experience instance with these two KPI values. In both cases, the representation of a machine state at a certain point in time is constructed by querying the event base for the

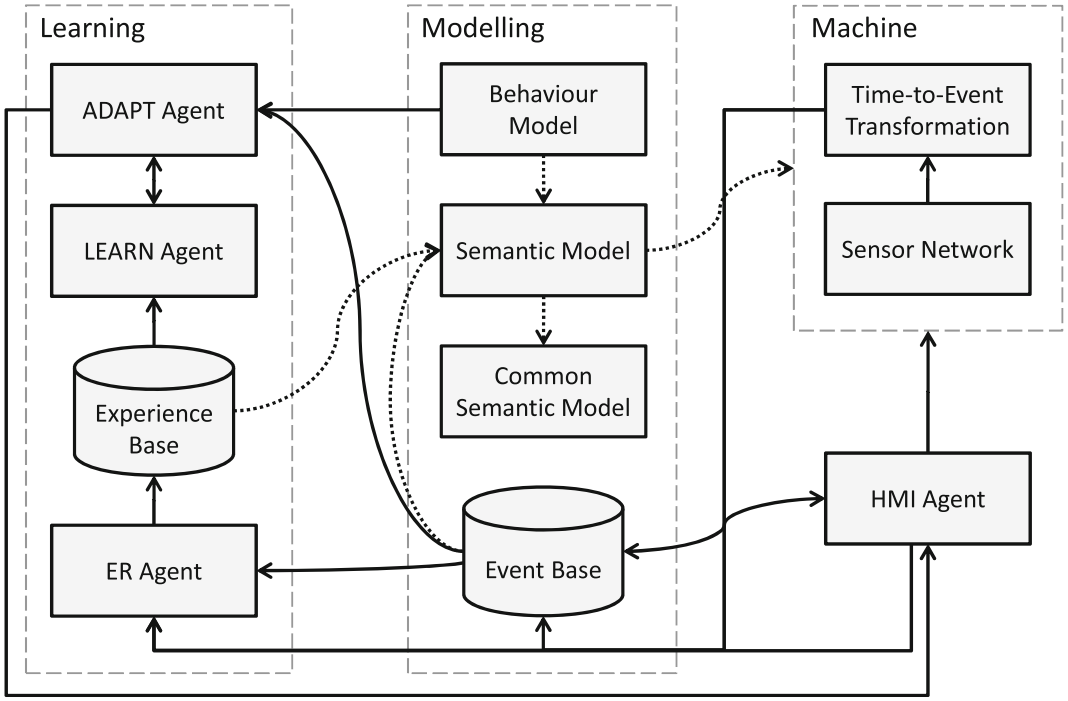

Fig. 1 Overview of FRAME architecture (solid lines indicate flow of data, dotted lines indicate dependencies)

latest state events generated up to that point. The KPI is calculated on such representation. After an adjustment occurs, it is normally necessary to wait for some parts to be manufactured before capturing the state of the machine again, calculating the KPI on the new state and thus measuring the impact of the adjustment. Figure 5 outlines the XML structure of an experience instance. `MachineStateAttributeList` in `MachineStateBeforeChange` and `MachineStateAfterChange` is a list of attribute-value pairs containing all the attributes of the associated machine state. `FrameAdjustmentEventList` in `AdjustmentLog` contains all the adjustments of the session if experience is created per adjustment session, and only one adjustment otherwise.

The choice between creating experience per adjustment or per adjustment session depends on the level of refinement that one wants to have in the experience base and therefore when adapting, as well as on performance considerations. If experience is created per adjustment, more specific knowledge on the impact of single adjustments will be available. For each occurrence of a certain type of adjustment, a separate evaluation with KPI calculations will take place, thus providing a series of related experience instances in different contexts. If experience is created per session, less specific knowledge on single adjustments will be available, but it will be possible to suggest multiple adjustments at each step of an adaptation process, as described in Section 4. In addition, if experience is created per adjustment, more queries to the event base will be executed (two for each adjustment), compared with the case of adjustment sessions (two for each session).

The experience base is used by the LEARN agent, described in the next subsection.

3.3 Self-learning and adaptation

The LEARN agent implements a learning technique for generating adaptation knowledge. This technique generalises the examples contained in the experience base created by the ER agent. This knowledge is used by the ADAPT agent to recommend adjustments to the user (the engineer or system integrator making changes). The LEARN agent is invoked by the ADAPT agent during an adaptation.

The following is a step-by-step description of a typical adaptation scenario employing the LEARN and ADAPT agents, together with the other agents, in FRAME:

1. The HMI agent warns the operator that the system performance is sub-optimal. This happens when the KPI being calculated is out of tolerance.
2. The operator queries the HMI agent for a feasible adjustment that would improve the performance. The HMI agent relays the request to the ADAPT agent.

```
<FrameAdjustmentEvent>
  <sourceStation>STAT</sourceStation>
  <sourceComponent>MOOS</sourceComponent>
  <mainType>AdjustmentEvent</mainType>
  <subType>ParameterAdjustmentEvent</subType>
  <eventT>1610365552.227</eventT>
  <cycleID>49211</cycleID>
  <cycleT0>1610365546.132</cycleT0>
  <primaryValue>280</primaryValue>
  <primaryValStr />
  <primaryUnit>Deg</primaryUnit>
  <ModelKeys>
    <FrameKeyModel>
      <concept>Product</concept>
      <instance>MedicalPen</instance>
    </FrameKeyModel>
    <FrameKeyModel>
      <concept>Process</concept>
      <instance>FRAME_C1_UG1_Reversal1</instance>
    </FrameKeyModel>
    <FrameKeyModel>
      <concept>ProcessUnit</concept>
      <instance>FRAME_C1_UG1_ST006</instance>
    </FrameKeyModel>
    <FrameKeyModel>
      <concept>Station</concept>
      <instance>FRAME_C1</instance>
    </FrameKeyModel>
    <FrameKeyModel>
      <concept>Parameter</concept>
      <instance>P_FRAME_C1_UG1_Reversal1_Device
_Grippers_Cam_FallingBegin</instance>
    </FrameKeyModel>
  </ModelKeys>
</FrameAdjustmentEvent>
```

Fig. 2 Example of adjustment event serialised into XML, indicating that the value of the parameter `P_FRAME_C1_UG1_Reversal1-_Device_Grippers_Cam_FallingBegin` (a cam angle used in the production system of Section 5) is set to 280°

```
<FrameStateEvent>
  <sourceStation>STAT</sourceStation>
  <sourceComponent>MOOS</sourceComponent>
  <mainType>StatusEvent</mainType>
  <subType>ParameterEvent</subType>
  <eventT>1610377346.150</eventT>
  <cycleID>0</cycleID>
  <cycleT0>1610377338.542</cycleT0>
  <primaryValue>41</primaryValue>
  <primaryValStr />
  <primaryUnit>INT</primaryUnit>
  <ModelKeys>
    <FrameKeyModel>
      <concept>Product</concept>
      <instance>MedicalPen</instance>
    </FrameKeyModel>
    <FrameKeyModel>
      <concept>Process</concept>
      <instance>FRAME_C1_UG1_CheckFrontCap</instance>
    </FrameKeyModel>
    <FrameKeyModel>
      <concept>ProcessUnit</concept>
      <instance>FRAME_C1_UG1_ST005</instance>
    </FrameKeyModel>
    <FrameKeyModel>
      <concept>Station</concept>
      <instance>FRAME_C1</instance>
    </FrameKeyModel>
    <FrameKeyModel>
      <concept>DataItem</concept>
      <instance>DI_FRAME_C1_UG1_CheckFrontCap_Device
_CheckCap2_successfulChecks</instance>
    </FrameKeyModel>
  </ModelKeys>
</FrameStateEvent>
```

Fig. 3 Example of state event serialised into XML, indicating that the value of the attribute (DataItem) `DI_FRAME_C1_UG1-_CheckFrontCap_Device_CheckCap2_successfulChecks` (number of successful checks in a process of the production system of Section 5) is 41

3. The ADAPT agent queries the event base for the latest state events and builds a representation of the current state of the machine.
4. The ADAPT agent queries the LEARN agent for a list of ranked adjustments that are applicable to the current machine state. The LEARN agent generates and returns this list.
5. The ADAPT agent can optionally run the received adjustments through a behaviour model to validate them and adjusts the rankings accordingly.
6. The ADAPT agent sends the ranked list of adjustments to the HMI agent, who presents it to the operator.
7. The operator selects one adjustment from the list and applies it.

The ranked list of adjustments is generated as described in the Section 4. The operator's chosen action results in a new adjustment that triggers the creation of new experience on that adjustment. Of course, the operator does not necessarily have to choose one of the presented adjustments and they are free to apply any. The new experience will refine future rankings in similar contexts.

3.4 Proactive HMI

The HMI agent provides an intelligent interface between the user and the agent framework, since it can guide the user and not simply receive input or present output. This agent plays these roles:

– Proactive system: the HMI agent guides the user through the steps involved in the execution of an adaptation and gives feedback on the changes being made by showing the resulting KPI value. For each change, the HMI agent can show pictures or videos illustrating the operations to perform step by step. By looking at the KPI value, the user can see whether the changes have produced the desired effect.

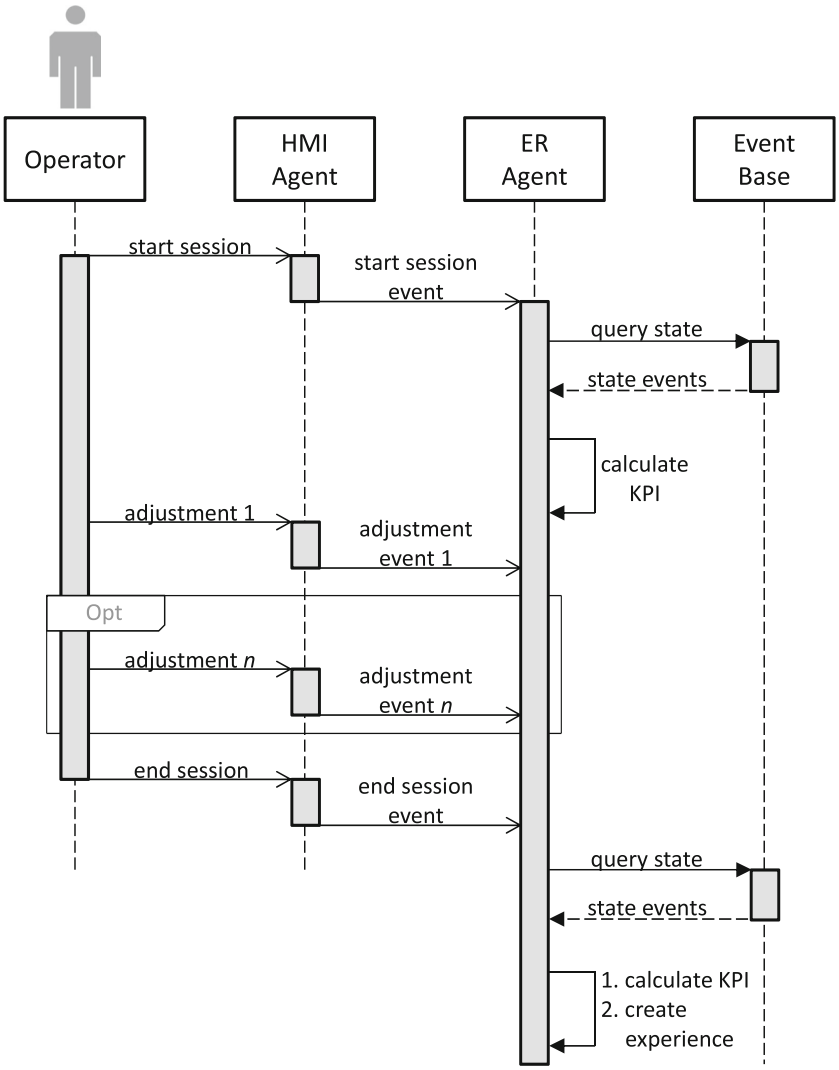

Fig. 4 Experience capture with adjustment sessions. The start and end of a session are signalled by events. All the adjustments made in a session are recorded as part of the experience instance being created

- Service provider: the HMI agent can be requested by other components to visualise messages (e.g. errors, warnings, notices) or ask the user for specific input, in particular information on manual adjustments (e.g. type of adjustment, components affected, values of parameters).
- Source of events: the HMI agent generates events about the operations carried out by the user through the HMI.
- Diagnostic tool: the HMI agent can provide the user with diagnostic information. For example, an *event browser* is available to analyse the content of events and their flow between components.

3.5 Framework layers

The framework architecture has two layers: a lower layer, called *station level*, and an upper layer, called *system level* (Fig. 6). The station level interfaces directly the machine, whereas the system level interfaces both the machine and the station level. FRAME agents are deployed on both layers. The agents of the station level are used for adapting individual modules or assembly stations and are deployed on each station. The agents of the system level are used for adapting the production system as a whole. Agents of the same type can, therefore, run simultaneously both at the station level in different stations and at the system level. Furthermore, other components (event base, experience base, semantic model and behaviour model) are also deployed on both layers.

In general, agents of the same type have the same role and behaviour in the two layers, but there are some important differences. The station level processes all the events generated by each individual station separately, whereas the system level receives all the events generated by all the stations. The ER agent of the system level captures the adjustments that have an effect on multiple stations, as well as those that are applied at the station level, and adds them to a system-level experience base. The system-level experience base represents all these changes, including an

```
<FrameExperienceInstance>
  TriggeringEvent
  <MachineStateBeforeChange>
    TimeStampStateStart
    TimeStampStateEnd
    FrameStateEventList
    MachineStateAttributeList
    KPI
  </MachineStateBeforeChange>
  <MachineStateAfterChange>
    TimeStampStateStart
    TimeStampStateEnd
    FrameStateEventList
    MachineStateAttributeList
    KPI
  </MachineStateAfterChange>
  AdjustmentParameterList
  <AdjustmentLog>
    TotalAdjustmentTime
    FrameAdjustmentEventList
  </AdjustmentLog>
</FrameExperienceInstance>
```

Fig. 5 Outline of the XML structure of an experience instance

evaluation of their effect on the whole system. In addition, the system-level experience base contains the experience received from all the stations. A LEARN agent of the station level applies the learning algorithm only to the station-level experience base of the station it is deployed on. In contrast, the LEARN agent of the system level applies the learning algorithm to the system-level experience base. The ADAPT agent of the system level creates a representation of the system-level machine state as the union of all station-level machine states (where a machine state is represented by a set of attribute-value pairs). Note that the station level can operate independently of the system level and does not require its presence. Similarly, agents of the station level deployed on different stations do not interoperate.

Since learning can occur at both station and system level, there can be changes having an opposite effect at different levels, in the sense that they are positive for a station but negative for the whole system, or the other way around. In general, the rankings of station-level adjustments may not be the same at the system-level. Also, a system-level change can have different effects on different stations. Clearly, the objective of an adaptation is to reach a target KPI at the system level. Although we do not have mechanisms to relate precisely this global objective to local ones, in order to reach the global objective, it is typically useful to focus on local objectives in the first instance. This is especially the case if there is more experience available at the station level than at the system level, indicating that the changes recommended at the station level are more reliable. This happens, in particular, when an adaptation involves using a module, possibly from another system, for which an experience base has already been built, whereas there is not experience available yet for the other modules, stations or for the full system.

3.6 Semantic modelling

An important part of the framework is composed of two models: a *common semantic model* (CSM) and a (system) *semantic model* (SM). The CSM is a meta-model of a production system, that is, a general model that describes what a typical automated production system consists of (resources, capabilities, devices, operations, products, parts, etc.), but does not define a particular one. The production engineer uses the CSM to create a SM for a specific production system, in which all the application-dependent declarative knowledge is represented. One particular application of the CSM is structuring operator knowledge and relating it to machine data during the ramp-up of an assembly system [32]. Although the main use case of FRAME is assembly systems, the default CSM

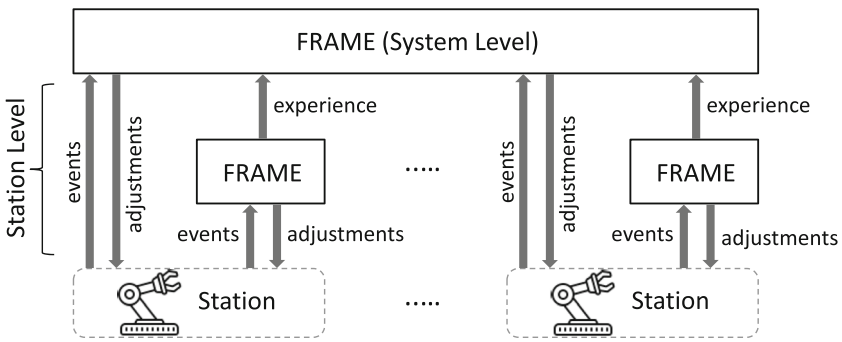

Fig. 6 2-layer FRAME architecture: system and station levels

provided by FRAME can be extended to cater for different manufacturing processes.

In particular, a SM for an assembly system specifies the properties of products being assembled, assembly process and equipment, as well as the relationships between them. Specifically, the SM contains parts and products, assembly operations and associated modules, process parameters, sensor data, performance measures, agents and their capabilities. The combination of CSM and SM provides a knowledge representation framework for modelling sensor data, experience and adaptation knowledge, that is also used for agent communication. Agents can produce or consume data whose structure is defined in the CSM/SM or data that reference entities defined in the CSM/SM.

Both CSM and SM are created using the ObjectLogic language [4] and the OntoBroker semantic middleware [33]. ObjectLogic is a deductive, object-oriented database language which combines a declarative semantics and an object-oriented data model. OntoBroker enables FRAME to store a large number of events in an event base and execute very expressive queries written in ObjectLogic. Events specify properties or instantiate elements defined in the SM, whereas queries are used to retrieve events. For example, Fig. 7, shows a query that retrieves all the latest events related to the same SM element generated in a specified time interval.

Both events and queries can reference elements defined in the CSM/SM. An editor is provided to create a SM using a mind map approach, so that no knowledge of ObjectLogic is required. The mind map is translated into ObjectLogic automatically, and the generated SM can be edited manually if necessary. An extract of the mind map of the SM used in the experiment (Section 5) is shown in Fig. 8. A screen capture of the SM editor is shown in Fig. 13 in the Appendix.

3.7 Behaviour model

Based on the SM, a behaviour model using discrete event simulation can be defined to analyse the behaviour of the production line. A behaviour model allows to simulate changes that are proposed by the ADAPT agent or that the operator intends to try out. Such a model is useful to detect poor potential changes without actually applying them on the physical machine. Key performance indicators that can be calculated include cycle time, machine and buffer usage, and failure rate. Simulations results produce knowledge on the effect of changes. This form of knowledge can be used when there is no information about a particular change in the experience base.

The creation of a behaviour model from a semantic model can be automated and the model can be dynamically updated to reflect the current state of the modelled system when changes have been applied, as presented by [34]. A behaviour model can be used by the ADAPT agent or by the user. Behaviour models are implemented in FRAME using the simulation modelling software Lanner Witness.

4 Self-learning implementation

When invoked, the LEARN agent generates a ranked list of adjustments that are applicable to the given machine state (also called adaptation context in the following, when referring to the machine state before a change). These adjustments are found by searching the experience base.

```
forall EVENT, UID, EVENTTIME, SOURCESTATION, SOURCECOMP, MAINT, SUBT, PRIMVAL, PRIMVALSTR, PRIMUNIT, CYCLEID,
    CYCLETO, CYCLETLEN, DATAITEM <- EVENT:FrameEvent[hasEventT -> EVENTTIME; hasUid -> UID;
    hasSourceStation -> SOURCESTATION; hasSourceComponent -> SOURCECOMP; hasMainType -> MAINT;
    hasSubType -> SUBT; hasPrimaryValue -> PRIMVAL; hasPrimaryValStr -> PRIMVALSTR;
    hasPrimaryUnit -> PRIMUNIT; hasCycleID -> CYCLEID; hasCycleTO -> CYCLETO; hasCycleTlen -> CYCLETLEN]@M
and inlist(MAINT,[%MAINTYPE%])
and inlist(SUBT,[%SUBTYPE%])
and between(%STARTTIME%, EVENTTIME, %ENDTIME%)
and EVENT[refersToDataItem -> DATAITEM]@M
and not (EXISTS OTHEREVENT, OTHERTIME OTHEREVENT:FrameEvent[hasSourceStation -> SOURCESTATION;
    hasSourceComponent -> SOURCECOMP; hasMainType -> MAINT; hasSubType -> SUBT; hasEventT -> OTHERTIME;
    refersToDataItem -> DATAITEM]@M
and between(%STARTTIME%, OTHERTIME,  %ENDTIME%)
and greater (OTHERTIME,  EVENTTIME)).
```

Fig. 7 ObjectLogic query that retrieves all the latest events of type %MAINTYPE% and subtype %SUBTYPE% related to the same SM element (DATAITEM) generated in a specified time interval %STARTTIME% - %ENDTIME%

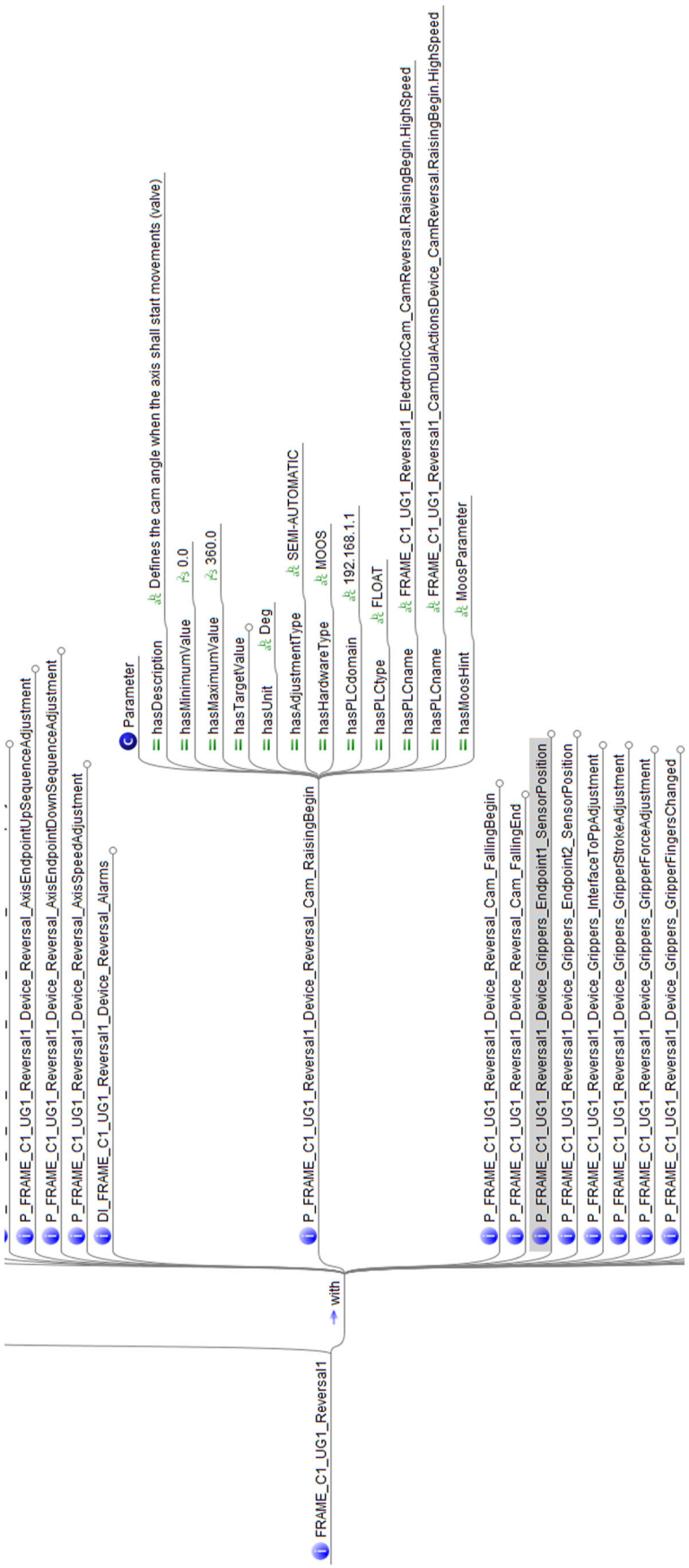

Fig. 8 Extract of the SM (mind map form) of the production system used in the experiment. FRAME_C1_UG1_Reversal1 is an operation consisting of reversing a pen. It is specified by parameters indicating, for instance, cam raising and falling angles, sensor positions and gripper force

The rankings are calculated based on the similarity of the adaptation contexts and the effectiveness of the associated adjustments in the experience base. Similarity of adaptation contexts is measured using a distance function calculated on multidimensional points representing machine states, constructed from sensor data and configuration parameters. The effectiveness of a past adjustment is measured by the KPI calculated on the machine state after the change.

The adjustments that were applied in the most similar contexts and that produced the best results are placed at the top of the list. The rationale is that an effective adjustment in a similar context is likely to produce a positive result. The first adjustment of the list is therefore the one recommended by the LEARN agent. The list may contain a large number of adjustments, depending on the accumulated experience and the number of possible adjustments. The adjustments with the lowest rankings could be filtered out. However, it may be useful to provide even the adjustments with the lowest rankings, so that the user can avoid them.

The process carried out to determine which adjustment to perform in one adaptation context can be framed as a *classification* problem. The objective of this problem is to learn how to identify the class of an instance based on a set of examples. The learning system receives in input some examples (training set) and produces a program (classifier) that is able to infer the class of instances that are not in the training set. In our adaptation problem, the instances to classify are the adaptation contexts in which the user queries the self-learning system, the classes are the possible adjustments and the training set is given by the experience instances in the experience base.

The adaptation problem is, however, more complex than a typical classification. The set of possible adjustments depends on the specific context. In general, there are preconditions associated with an adjustment, indicating if the adjustment is applicable or not. For example, adjustments related to devices not currently connected are clearly not applicable.[1] In addition, we do not want only to identify what type of adjustment is the best, but also how to make it, that is, the values of associated parameters. For example, if the recommended adjustment is "increase the pressure of cylinders", we need to know what the new pressure should be. In the rest of this section, we will show how to determine both adjustment types and values.

4.1 Experience base search

The search in the experience base is based on a variant of the k-nearest neighbour classification algorithm (kNN). Adaptation contexts are represented by points in a multidimensional space and their similarity is captured by a distance function. In kNN, given a point $\mathbf{x}$ to classify, the k nearest points to $\mathbf{x}$ are located and the class to assign to $\mathbf{x}$ is chosen among the neighbours' classes by applying a voting scheme. A simple voting scheme consists of assigning the most common class among the neighbours (see Fig. 9). The class of an adaptation context is the adjustment being applied in it.

Adaptation contexts can be defined by numerical or categorical attributes. The value of an attribute can be undefined if, for example, the module or sensor that produces it is not present or is faulty. Hence, we use a *heterogeneous Euclidean-overlap metric* (HEOM) [35]:

$$d(\mathbf{x}, \mathbf{x}') = \sqrt{\sum_{i=1}^{n} w_i (d_i(\mathbf{x}, \mathbf{x}'))^2},$$

where:

- $\mathbf{x}$ and $\mathbf{x}'$ are two n-dimensional points,
- $w_i \in [0, 1]$ is the weight assigned to attribute i, and
- $d_i(\mathbf{x}, \mathbf{x}') \in [0, 1]$ is the distance between $\mathbf{x}$ and $\mathbf{x}'$ on attribute i, defined as:

$$d_i(\mathbf{x}, \mathbf{x}') = \begin{cases} 1 & \mathbf{x}_i \text{ or } \mathbf{x}'_i \text{is unknown,} \\ \text{overlap}(\mathbf{x}_i, \mathbf{x}'_i) & \text{attribute } i \text{ is nominal,} \\ \text{rndiff}_i(\mathbf{x}_i, \mathbf{x}'_i) & \text{otherwise.} \end{cases}$$

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

$$\text{rndiff}_i(x, y) = \frac{|x - y|}{\max_i - \min_i},$$

where $\max_i$ and $\min_i$ are, respectively, the maximum and minimum values observed in the training set for attribute i.

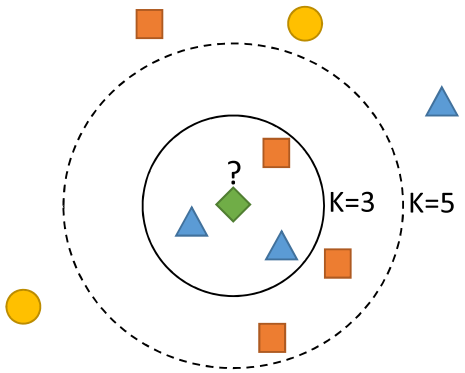

Fig. 9 kNN algorithm: The point to classify is indicated by "?". The possible classes are: "blue triangles", "orange squares" and "yellow circles". If $k = 3$ (solid line circle) the most common class is "blue triangles". If $k = 5$ (dashed line circle) the most common class is "orange squares"

[1] Preconditions are currently hard-coded in FRAME. Ideally, they should be specified in the SM.

In the generation of the rankings, we consider not only the similarity between adaptation contexts, but also the system performance obtained by the adjustments in the experience base. The performance is measured by the KPI of the machine state after the adjustment. An experience instance has the form $(\mathbf{x}, \mathsf{adj}, \mathbf{x}')$, where:

- $\mathsf{adj} = (t, v)$ is an adjustment of type t and value v
- $\mathbf{x}$ is the machine state before adj
- $\mathbf{x}'$ is the machine state after adj

Let

- $E = (\mathbf{x}, \mathsf{adj}, \mathbf{x}')$ be an experience instance,
- $\mathbf{y}$ be an adaptation context,
- d be a HEOM, and
- f be a KPI.

The *similarity-performance function* e_f^d [7] is defined as:

$$e_f^d(E, \mathbf{y}) = \begin{cases} \dfrac{d(\mathbf{x}, \mathbf{y})}{f(\mathbf{x}')} & \text{if } f(\mathbf{x}') \neq 0 \\ \infty & \text{otherwise.} \end{cases}$$

The smaller the value of $e_f^d(E, \mathbf{y})$, the better adj is anticipated to be in $\mathbf{y}$.

4.2 Voting and ranking

Let $\mathbf{y}$ be an adaptation context to be classified, d a distance function and f a KPI. There are two cases to consider:

$k = 1$ The experience instances are sorted by $e_f^d(E, \mathbf{y})$, in ascending order, to produce a ranked list of adjustments for $\mathbf{y}$. The first experience instance in the list (the one with the smallest value of $e_f^d(E, \mathbf{y})$) contains the recommended adjustment type and value.

$k > 1$ This is described in Procedure 1. Let $E_1, \ldots, E_k$ be the k experience instances having the smallest values of $e_f^d(E, \mathbf{y})$, in ascending order. A class must be selected among those of these k nearest neighbours. To this end, a voting scheme is applied. Note that selecting the most common class among the neighbours may not be a good scheme, because this class is likely to be the most frequent in the training set. One method is to assign different weights to the neighbours, based on their respective distances from the point to be classified. In our solution, we use e_f^d to calculate the weights. This way, both the distance of a neighbour and the impact of its associated adjustment on the performance of the system are considered. We define the weight $w(E_i)$ of E_i as follows:

$$w(E_i) = \frac{e_f^d(E_k, \mathbf{y}) - e_f^d(E_i, \mathbf{y})}{e_f^d(E_k, \mathbf{y}) - e_f^d(E_1, \mathbf{y})}$$

The instances $E_1, \ldots, E_k$ are grouped by class. For each class, the weights of its instances are summed up. The recommended adjustment type is represented by the class having the largest sum of weights. For an adjustment type t, the instances among $E_1, \ldots, E_k$ having class t are selected. If numerical, the adjustment value is calculated as the weighted mean of all the adjustments values of these selected instances. The values $w(E_i)$ are used as weights. Further adjustments can be proposed similarly by considering the other classes in descending order of sum of weights.

Procedure 1 rank-adjustments

Input: experience instances $E_1, \ldots, E_n$ $(k \leq n)$, adaptation context $\mathbf{y}$, HEOM d, KPI f
Output: ranked list of adjustments
  for all $1 \leq i \leq n$ **do**
    calculate $e_f^d(E, \mathbf{y})$
  end for
  sort $E_1, \ldots, E_n$ by $e_f^d(E, \mathbf{y})$
  $\{E_{\sigma_1}, \ldots, E_{\sigma_k}$ are now the k nearest neighbours in sorted order$\}$
  for all $1 \leq i \leq k$ **do**
    calculate $w(E_{\sigma_i})$
  end for
  let $t_1, \ldots, t_m$ be the adjustment types in $E_{\sigma_1}, \ldots, E_{\sigma_k}$ $(m \leq k)$
  for all $1 \leq j \leq m$ **do**
    let $E_i' = (\mathbf{x}_i, \mathsf{adj}_i, \mathbf{x}_i') \in \{E_{\sigma_1}, \ldots, E_{\sigma_k}\}$ be the experience instances such that $\mathsf{adj}_i = (t_j, v_i)$
    $\mathsf{w\_sum}_j \leftarrow \sum_i w(E_i')$
    $\bar{v}_j \leftarrow \dfrac{\sum_i v_i w(E_i')}{\mathsf{w\_sum}_j}$
  end for
  sort $(t_1, \bar{v}_1), \ldots, (t_m, \bar{v}_m)$ in descending order of $\mathsf{w\_sum}_j$

5 Experimental evaluation

FRAME was evaluated on a production system for the assembly of injection pens. This system is made up of various pick-and-place and inspection modules. The pick-and-place modules are used for rotating and moving parts, and putting them together. The inspection modules are used for checking the presence and correct position of parts on pallets, and the quality of the finished product in terms of correct dimensions and assembly. Figure 10 shows a picture of the machine. If a finished product has passed all the checks, then it is accepted, otherwise it is rejected. Bad parts are not reworked because the process would not be cost-effective. Therefore, maximising the number of good parts is indispensable for this production system.

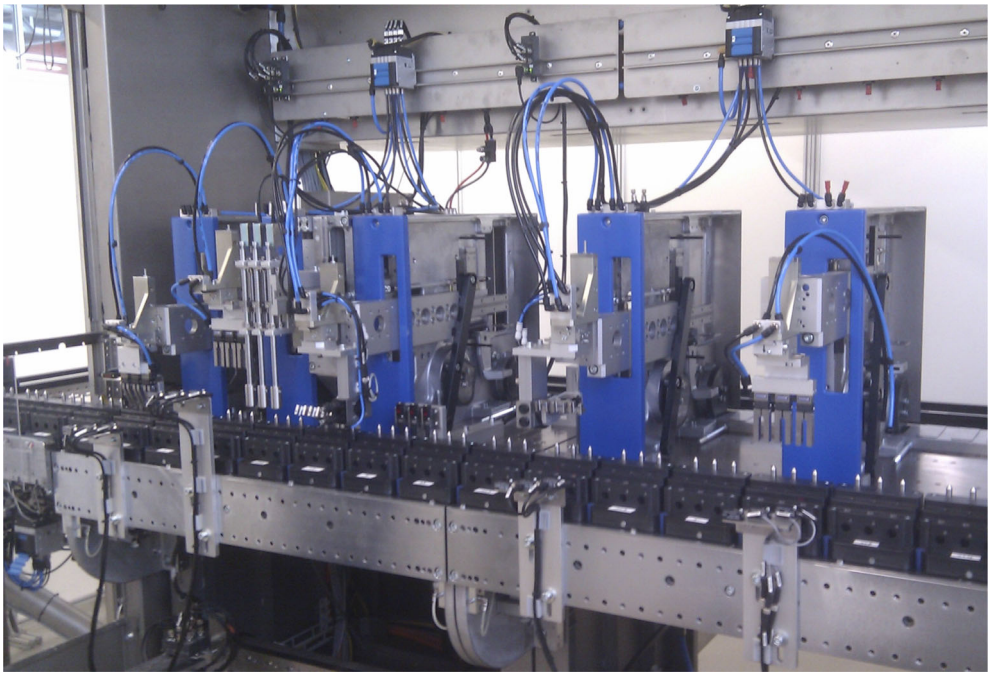

Fig. 10 Production system for the assembly of injection pens

A conveyor system is used to transport pallets holding the parts from station to station. An injection pen consists of three parts: a body, a tank and a cap. When fed into the line, each pallet holds the parts for assembling 3 injection pens and, at the end of the process, holds the finished products. Each pick-and-place module operates 3 grippers, one for each pen to assemble. The assembly machine is cam-driven and consists of 7 modules performing the following operations:

1. Check the presence of the parts on the pallet;
2. Pick-and-place tanks into bodies;
3. Check that tanks have been inserted correctly into bodies;
4. Pick-and-place caps onto bodies;
5. Check that caps have been inserted correctly onto bodies;
6. Reverse pens;
7. Check reversal.

The sequence of operations is linear.

The aim of the experiment was to evaluate how the self-learning framework can aid operators in system adaptations. This was done by adapting the production system for the manufacture of a product variant with some minor differences compared to the original product, in terms of physical and geometrical characteristics. The adaptation involved making some mechanical changes, in particular changing some of the grippers. All the affected parameters in grippers and inspection devices had to be adjusted (e.g. position and dimensions of the parts, opening and closing angles of the grippers, pressure of the grippers). Although the product variant was not much different to the original one, such adaptation process could be laborious as assembly operations must be very precise and require repeatable positioning.

The experiment was organised as follows. First, the system was adapted by 4 operators individually. During this phase FRAME was used to capture the changes being made by the operators and to build experience, but not to recommend changes. An experience base of adjustments was built by the ER agent at the system level. The original state of the machine was restored after each operator completed the process. Then, the system was adapted by 4 other operators individually using FRAME with the experience base built in the previous phase. The operators of the two groups had comparable experience and skills, and did not communicate during the experiment. The number of adjustments performed by the two groups for completing the process and the average levels of system performance reached at the end were compared.

The performance of the system was characterised in terms of quality of the finished product. The number of good parts out of the total number of parts assembled in a batch was used as KPI. The value of this KPI was updated every time the quality of an assembled part was checked. The number of parts produced by each operator between two consecutive adjustments was fixed and it was equal to 6. The current KPI value was visible to the operators

to allow them to evaluate the result of their actions. The operators were expected to obtain a target KPI value of 0.98 in order for the adaptation process to be considered complete. The average KPI values obtained by the two groups of operators in the first 25 steps of the adaptation process are shown in Fig. 11. After 25 steps, the first group (that did not use FRAME) reached an average KPI of 0.8, while the second group (that used FRAME) reached an average KPI of 0.95. Further adjustments were required for all operators except one to complete the process. To reach the target KPI, the 4 operators of the group that did not use FRAME made respectively 30, 33, 36 and 37 adjustments, whereas the 4 operators of the group that used FRAME made respectively 25, 29, 29 and 31 adjustments.

Figure 12 shows the number of times that the operators of the second group applied the following types of changes: (1) the adjustment recommended by the ADAPT agent (i.e. the first-ranked adjustment), (2) a different adjustment ranked by the ADAPT agent and (3) any other adjustment not ranked by the ADAPT agent. The operators followed the recommendations of FRAME most of the times. Only in a few cases did they choose another adjustment among those suggested, and in very few cases a different one. They decided not to apply any of the suggested adjustments when they thought that those changes were not useful or safe in that context, that they had already applied them (successfully or not) or that they could get better results with different changes.

The evaluation suggests that the use of FRAME to rank and recommend changes allows operators to perform system adaptations in fewer steps and thus in a shorter period of time. Figures 13 and 14 in the Appendix show screen captures of the SM editor used in the experiment, some KPI values being produced and new experience being created.

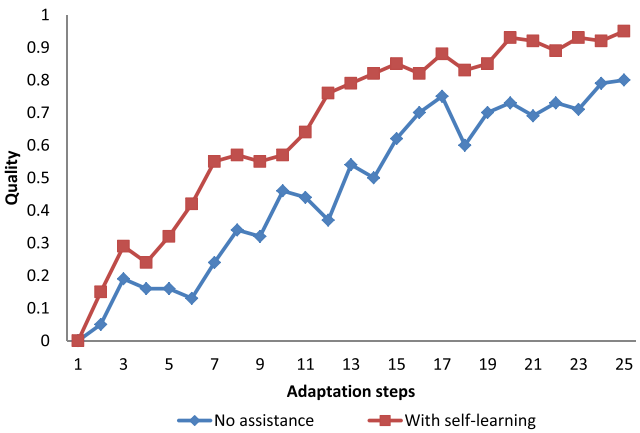

Fig. 11 Average quality values of the two groups of operators during the adaptation, the group that used FRAME (with self-learning) and the one that did not (no assistance)

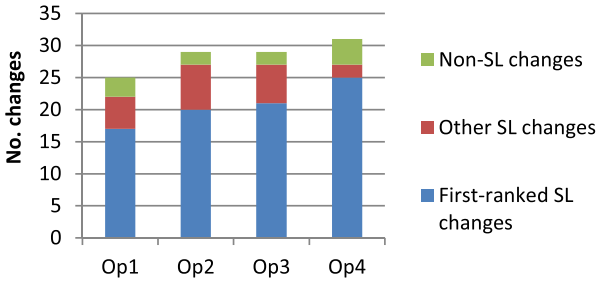

Fig. 12 Types of changes applied by the operators of the group that used FRAME in the adaptation: **1** changes recommended by the self-learning (SL) framework (i.e. first-ranked changes), **2** other changes ranked (lower than first) by the SL framework, **3** any other changes not ranked by the SL framework

6 Conclusion

Multi-agent systems can be employed to offer advanced levels of interconnectivity and intelligence in production systems, from data acquisition and analysis to distributed control, self-learning and adaptation. In this paper, we have presented FRAME, a multi-agent framework implementing a self-learning technique that generalises from the adaptation experience captured on a machine. FRAME has been evaluated on an automated assembly system, showing that it can effectively help engineers throughout an adaptation process. The domain of application of the framework is, however, wider and includes, in general, any production system. Further experimentation on a number of more complex production systems is required to confirm the generality of the results of this paper. This is planned as future work.

The deployment of FRAME on a specific production system requires the definition of a semantic model, which describes the structure and capabilities of the system, and the implementation of the low-level interface with the hardware. Minimal development work is required to customise agent behaviour for a specific production system. Adaptation knowledge is built directly from the experience base, hence there is no need to construct manually a knowledge base.

The accuracy of the learning method depends on the distance and KPI functions used to define the similarity-performance function. The similarity-performance function is based on the assumption that one adjustment is likely to have the same effect on similar states. To improve this function, weights of attributes of adaptation context could be determined dynamically based on the relevance of the attributes to different contexts and adjustments. Another interesting development of the similarity-performance function is the integration of a method to evaluate adaptations in contexts defined in terms of production

capabilities. These capabilities would be expressed using more sophisticated semantics, which adaptation contexts cannot represent at the moment as lists of attributes.

Research on cyber-physical systems in manufacturing should provide methodologies and tools to integrate seamlessly the physical and cyber aspects, despite the heterogeneity of their designs. Integration should allow the physical elements to make the best use of the available computational resources, and not simply ensure that the minimum interface requirements are met on either side. A self-learning framework like FRAME can help to achieve this goal.

Appendix:

Figures 13 and 14 contain screen captures of, respectively, the SM editor of FRAME and new experience being logged.

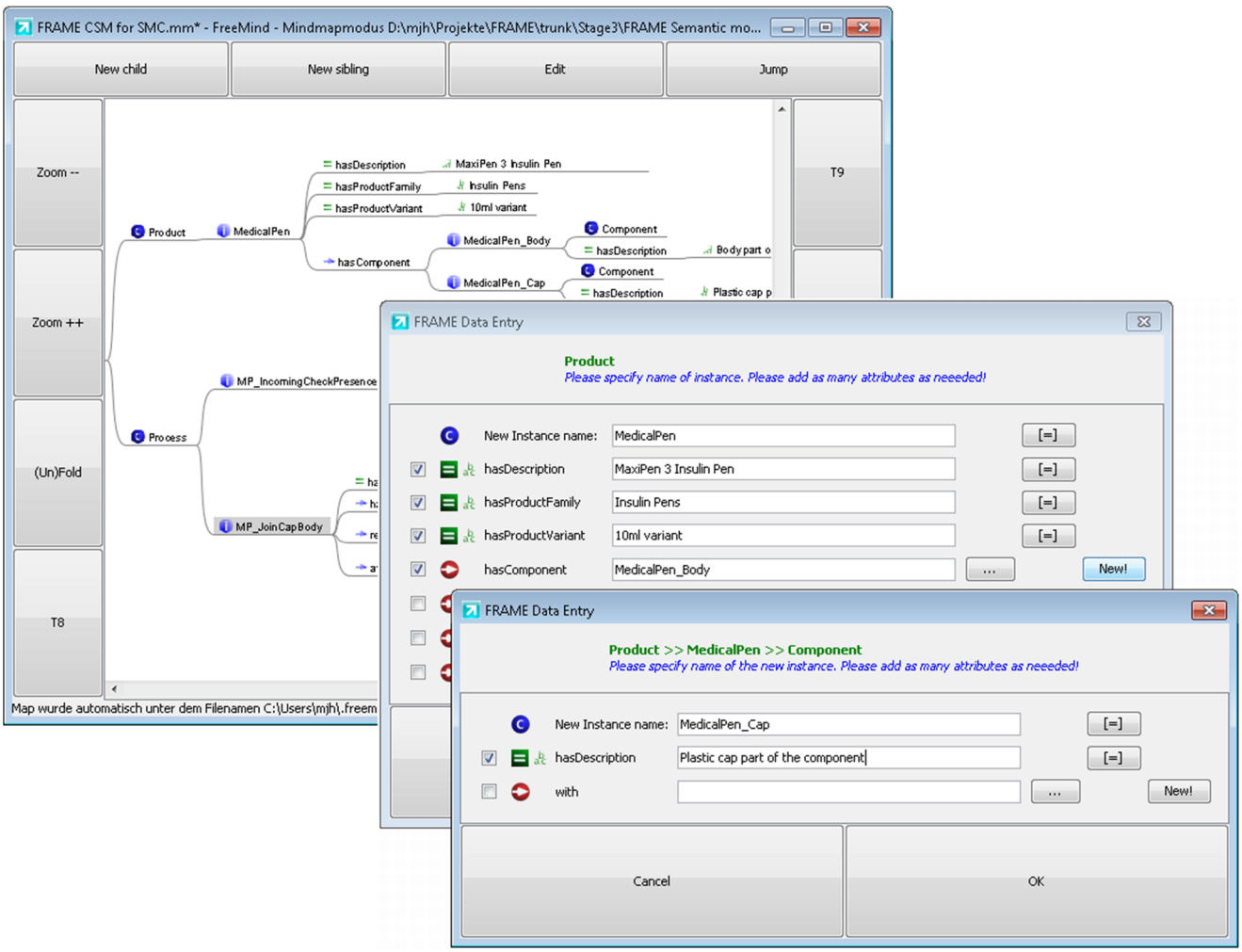

Fig. 13 A screen capture of the SM editor taken while entering product details

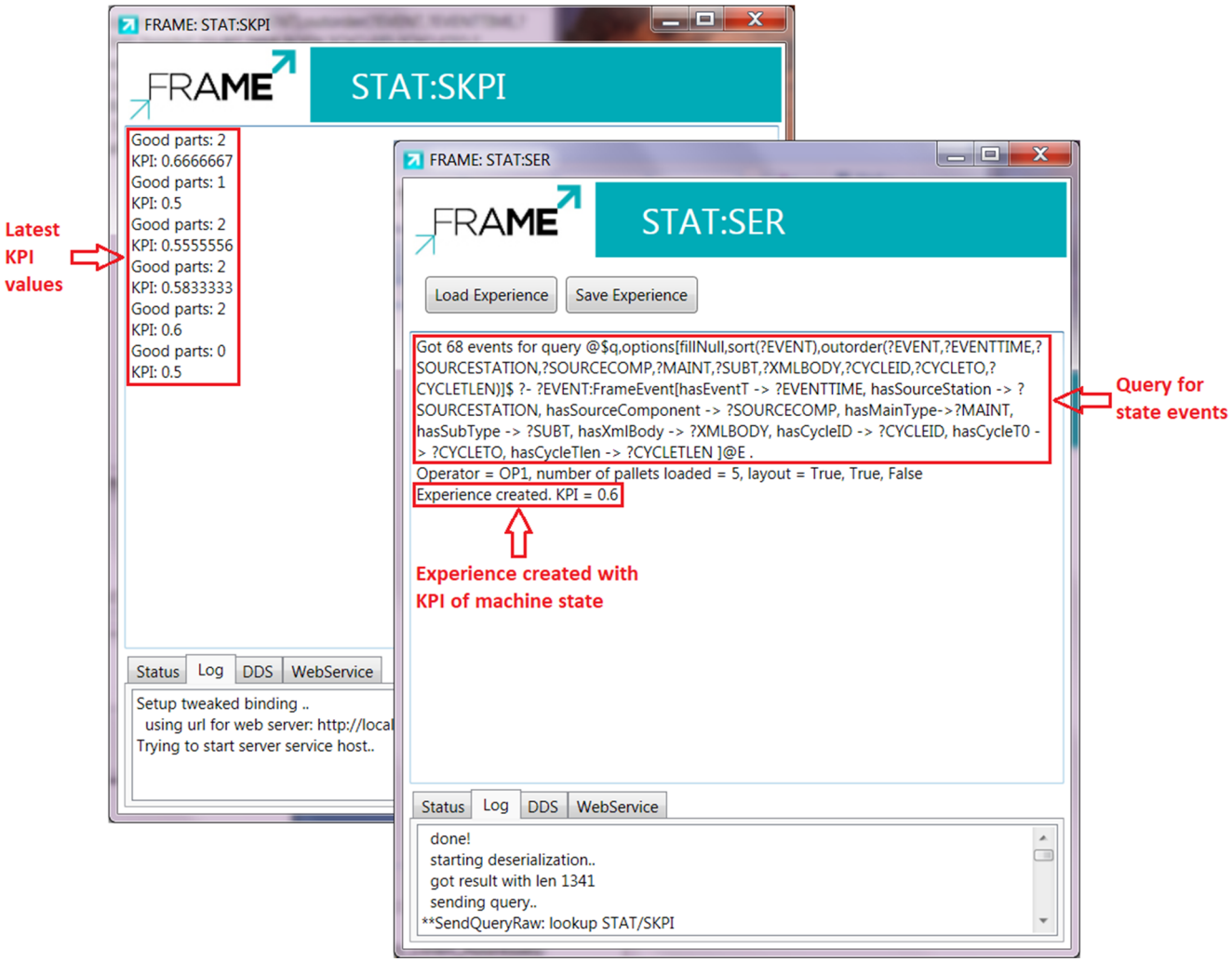

Fig. 14 A screen capture of the system taken while logging the creation of new experience, showing the most recent KPI values and a query to the event base being made to capture the machine state after a change

Author contribution D. S.: investigation, conceptualisation, methodology, software, experimentation, writing. S. M. A.: methodology, review, writing. S. M. R.: methodology, funding acquisition, supervision.

Funding This work was supported in part by the European Commission [grant agreement n. 314762].

Availability of data and materials No supplementary data or materials available.

Declarations

Conflict of interest The authors declare no competing interests.

References

1. Järvenpää E, Siltala N, Lanz M (2016) Formal resource and capability descriptions supporting rapid reconfiguration of assembly systems. In: 2016 IEEE International symposium on assembly and manufacturing (ISAM), pp 120–125
2. Harms R, Fleschutz T, Seliger G (2008) Knowledge based approach to assembly system reuse. In: Proceedings of the ASME 2008 9th Biennial conference on engineering systems design and analysis. Volume 1: Advanced energy systems; advanced and digital manufacturing; advanced materials; aerospace, pp 295–302
3. Lee J (2015) Smart factory systems. Informatik-Spektrum 38(3):230–235

4. Angele J, Kifer M, Lausen G (2009) Ontologies in F-Logic. In: Staab S, Studer R (eds) Handbook on ontologies, pp 45–70. Springer
5. Scrimieri D, Ratchev SM (2014) A k-nearest neighbour technique for experience-based adaptation of assembly stations. J Cont Auto Elect Sys 25:679–688. https://doi.org/10.1007/s40313-014-0142-6
6. 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. https://doi.org/10.1080/00207543.2016.1207817
7. Scrimieri D, Oates RF, Ratchev SM (2015) Learning and reuse of engineering ramp-up strategies for modular assembly systems. J Intell Manuf 26:1063–1076. https://doi.org/10.1007/s10845-013-0839-6
8. Doltsinis S, Ferreira P, Lohse N (2018) A symbiotic human-machine learning approach for production ramp-up. IEEE Trans Human-Machine Sys 48(3):229–240
9. ElMaraghy HA (2009) Changing and evolving products and systems – models and enablers. In: ElMaraghy HA (ed) Changeable and reconfigurable manufacturing systems, pp 25–45, London. https://doi.org/10.1007/978-1-84882-067-8_2
10. ElMaraghy HA (2006) Flexible and reconfigurable manufacturing systems paradigms. Int J Flex Manuf Syst 17:261–276
11. Koren Y, Shpitalni M (2010) Design of reconfigurable manufacturing systems. J Manuf Syst 29(4):130–141. https://doi.org/10.1016/j.jmsy.2011.01.001. http://www.sciencedirect.com/science/article/pii/S0278612511000021
12. Barbosa J, Leitão P, Adam E, Trentesaux D (2015) Dynamic self-organization in holonic multi-agent manufacturing systems: The ADACOR evolution. Comput Ind 66:99–111. https://doi.org/10.1016/j.compind.2014.10.011. http://www.sciencedirect.com/science/article/pii/S0166361514001894
13. Chaplin JC, Bakker OJ, de Silva L, Sanderson D, Kelly E, Logan B, Ratchev SM (2015) Evolvable assembly systems: a distributed architecture for intelligent manufacturing. IFAC-PapersOnLine 48(3):2065–2070. https://doi.org/https://doi.org/10.1016/j.ifacol.2015.06.393. http://www.sciencedirect.com/science/article/pii/S2405896315006321, 15th IFAC Symposium onInformation Control Problems in Manufacturing
14. Bortolini M, Galizia FG, Mora C (2018) Reconfigurable manufacturing systems: Literature review and research trend. J Manuf Syst 49:93–106. https://doi.org/10.1016/j.jmsy.2018.09.005. http://www.sciencedirect.com/science/article/pii/S0278612518303650
15. Shen W, Hao Q, Yoon HJ, Norrie DH (2006) Applications of agent-based systems in intelligent manufacturing: an updated review. Adv Eng Inform 20(4):415–431
16. Leitão P, Mařík V, Vrba P (2013) Past, present, and future of industrial agent applications. IEEE Trans Industrial Info 9(4):2360–2372
17. Derigent W, Cardin O, Trentesaux D (2020) Industry 4.0: contributions of holonic manufacturing control architectures and future challenges. J Intell Manuf. https://doi.org/10.1007/s10845-020-01532-x
18. Leitão P, Karnouskos S, Ribeiro L, Lee J, Strasser T, Colombo AW (2016) Smart agents in industrial cyber–physical systems. Proc IEEE 104(5):1086–1101
19. Borangiu T, Raileanu S, Berger T, Trentesaux D (2015) Switching mode control strategy in manufacturing execution systems. Int J Prod Res 53(7):1950–1963. https://doi.org/10.1080/00207543.2014.935825
20. Cupek R, Ziebinski A, Huczala L, Erdogan H (2016) Agent-based manufacturing execution systems for short-series production scheduling. Comput Ind 82:245–258. https://doi.org/10.1016/j.compind.2016.07.009. http://www.sciencedirect.com/science/article/pii/S0166361516301233
21. Ribeiro L, Rocha A, Veiga A, Barata J (2015) Collaborative routing of products using a self-organizing mechatronic agent framework—a simulation study. Comput Ind 68:27–39
22. Rocha AD, Lima-Monteiro P, Parreira-Rocha M, Barata J (2019) Artificial immune systems based multi-agent architecture to perform distributed diagnosis. International Journal of Intelligent Manufacturing 30:2025–2037. https://doi.org/10.1007/s10845-017-1370-y
23. Valckenaers P, Van Brussel H (2005) Holonic manufacturing execution systems. CIRP Ann 54(1):427–432. https://doi.org/10.1016/S0007-8506(07)60137-1. http://www.sciencedirect.com/science/article/pii/S0007850607601371
24. Novas JM, Van Belle J, Germain BS, Valckenaers P (2013) A collaborative framework between a scheduling system and a holonic manufacturing execution system. In: Borangiu T, Thomas A, Trentesaux D (eds) Service orientation in Holonic and multi agent manufacturing and robotics, pp 3–17. Springer Berlin Heidelberg, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-35852-4_1
25. 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, pp. 1–19. Springer International Publishing, Cham
26. Borangiu T, Raileanu S, Trentesaux D, Berger T, Iacob I (2014) Distributed manufacturing control with extended CNP interaction of intelligent products. Inter J Intel Manufacturing 25:1065–1075. https://doi.org/10.1007/s10845-013-0740-3
27. McFarlane D, Giannikas V, Wong ACY, Harrison M (2013) Product intelligence in industrial control: Theory and practice. Annu Rev Control 37(1):69–88. https://doi.org/10.1016/j.arcontrol.2013.03.003. http://www.sciencedirect.com/science/article/pii/S1367578813000059
28. Lee EA (2010) CPS foundations. In: Proceedings of the 47th design automation conference, pp 737–742
29. Schütz D, Wannagat A, Legat C, Vogel-Heuser B (2013) Development of PLC-based software for increasing the dependability of production automation systems. IEEE Trans Industrial Info 9(4):2397–2406
30. Foehr M, Leitão P, Wagner T, Jäger T, Lüder A (2012) Integrating mechatronic thinking and multi-agent approaches. In: Proceedings of 2012 IEEE 17th international conference on emerging technologies factory automation (ETFA 2012), pp 1–8
31. Bussmann S, Jennings NR, Wooldridge MJ (2004) Multiagent systems for manufacturing control: a design methodology. Springer, Berlin
32. Konrad K, Hoffmeister M, Zapp M, Verl A, Busse J (2012) Enabling fast ramp-up of assembly lines through context-mapping of implicit operator knowledge and machine-derived data. In: Proceedings of the 6th IFIP WG 5.5 International precision assembly seminar, pp 163–174. Springer
33. Angele J (July 2014) OntoBroker: Mature and approved semantic middleware. Semant. Web 5(3):221–235
34. Hirst A, Yates P (2011) Mindmap to Witness conversion program. In: WITNESS user conference
35. Wilson DR, Martinez TR (1997) Improved heterogeneous distance functions. J Artif Intell Res 6:1–34