An executable capability concept in formal resource descriptions

An Executable Capability Concept in Formal Resource Descriptions ⋆

Niko Siltala * **Eeva Järvenpää** * **Minna Lanz** *

* *Laboratory of Mechanical Engineering and Industrial Systems, Tampere University of Technology, Tampere, Finland, (e-mail: niko.siltala@tut.fi, eeva.jarvenpaa@tut.fi, minna.lanz@tut.fi)*

Abstract: Neutral description of production resources' capabilities, interfaces, and other characteristics is required for efficient design or reconfiguration of production systems. In our previous work, we have developed a Resource Description and Capability concepts. An integral part of the Resource Description is an Executable Capability concept, which describes resource's control interface in a vendor neutral form. In this paper, we show the underlying data model of Executable Capabilities, and how these capabilities can be utilised in the task programming and execution of modular production systems. Finally, a few case examples are shown.

Keywords: data models, resource description, capability, system design and programming, production execution, reconfigurable manufacturing

1. INTRODUCTION

The requirements for production systems are continuously shifting towards higher flexibility and adaptivity. Adaptivity to new requirements means that the existing production system needs to be reconfigured, either physically, parametrically, or logically. There is a need for new solutions that would drastically reduce the time and effort put into planning and implementing the alterations in a factory, or to allow autonomous adaptation during runtime, based on machine-to-machine communication and self-organization (Leitão et al., 2016). System creation or reconfiguration implies that the system and/or its resources need to be (re-)programmed and (re-)orchestrated. Such programming requires expert knowledge. Due to the requirements for fast adaptation, and shortage of talented workforce, new methods and tools are needed to ease or automatize this programming task.

The fourth industrial revolution, "Industry 4.0" or "Smart Manufacturing", aims to tackle the production system adaptation issues by introducing Cyber-Physical Systems (CPSs), which are physical systems, e.g. machines or workpieces, which combine with the digital world, i.e. sensors and intelligence, and are thus able to communicate, act, and control themselves and each other (Baheti and Gill, 2011). Important concept CPS presents is that it provides a self-contained resource module, which includes its controls and wraps them around interfaces. In Thoben et al.'s (2017) review of the current Industry 4.0 and Smart Manufacturing initiatives they identified the standardization of both interfaces and information models as one of the most relevant research issues for smart manufacturing. We propose in this paper a solution for the control interface part and related data model.

⋆ This research has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 680759 ReCaM. http://www.recam-project.eu

Many projects have studied the automation of the execution program generation for production tasks based on skill descriptions. Bøgh et al. (2012) developed skills in robotic application, with the aim to support task-oriented programming and re-use of robotic programs. Task-oriented programming specifies what the resource should do in terms of actions on the objects involved in the task and not how the task should be achieved. The SkillPro-project aimed to develop a holistic service-oriented framework for modelling and orchestration of adaptable manufacturing systems. Skills were used to represent the abilities of the available assets (asset skills) and the requirements of different production steps (production skills). AutomationML-based format was used to store and communicate the skill descriptions to facilitate autonomous setup and execution of production tasks (Pfrommer et al., 2014; Puerto et al., 2015). Backhaus and Reinhart (2017) presented a similar concept aiming to simplify the task-oriented programming of assembly systems by vendor-independent skill descriptions. They represented a concept on modelling and connecting together the product, processes, and resources, and a six-phase method for identification of applicative skills in assembly system.

The aim of these approaches has been to decrease the programming time of the resources, and to allow even non-experts to create the robotic programs. These approaches rely on specifically defined skill models and skill information, but they do not consider from where that skill information can be retrieved in large scale applications. They are also domain specific, concentrating only on narrow domain, such as robotic skills. Only a few examples of skills have been published in detail, and those have been simple, with limited amount of skill parameters. It is not clear, if these approaches provide any solution for the system design and reconfiguration planning phase, in terms of capability matchmaking between product requirements and resource capabilities. Furthermore, the existing approaches

do not make difference between the design level capabilities and the programming level capabilities. By design level capabilities we mean the functionalities that are required by the product, i.e. "pick and place". By programming level capabilities we refer to the executable capabilities that are actually executed and controlled during the production, such as "open fingers", "move to position", ...

We have developed an XML-based Resource Description concept that targets to inclusive representation of resource's capabilities, interfaces, and other characteristics (Siltala, 2016; Siltala et al., 2016). The Resource Description (RD) acts as a comprehensive digital container of all the information related to a single model or type of resource. An integral part of the RD concept is recently developed Executable Capability concept, which is presented in this paper. Executable Capability captures resource's abstract control interface information in a formalised and vendor neutral format. This information can be utilised when a) resource vendor makes the control implementation for their resource, or when b) system integrator configures (i.e. program and commission) the system for production, even (semi-)automatically.

The paper is organised as follows. The second chapter represents the proposed Executable Capability concept and its relation to other related concepts. Next chapter discusses the exploitation and utilisation of the concept, followed by a few example use case scenarios. Finally, the paper ends with conclusion.

2. EXECUTABLE CAPABILITY CONCEPT

In the next sections, the Executable Capability Concept is first connected to the context of our other works and then its data model is defined in the details.

2.1 Overview of associated concepts

In our previous works, we have presented: an OWL-based Capability Model, which can be used to describe the capabilities of manufacturing resources and combined capabilities of two or more connected resources (Järvenpää et al., 2016); capability matchmaking procedure, which aims to make match between product requirements and resource capabilities (Järvenpää et al., 2017); and interface matchmaking approach, making sure that the suggested resources can be combined from the hardware interface perspective (Siltala et al., in press, 2018). These works support the system design and reconfiguration until the point in which the suitable resources fulfilling the product requirements have been found. Executable Capability will help us to make a step further, to support also for the system's execution phase including support for the auto-programming and orchestration of resources.

Table 1 defines the main terms used in the context of this paper. Fig. 1 positions Executable Capability in the context of production system design, and shows it in association with other related concepts. The figure is divided into two layers - design and execution - to differentiate the timely nature and different requirements related to these two phases. The elements on the design layer represent concepts for which we have developed formal representation. The right side presents the Resource Description,

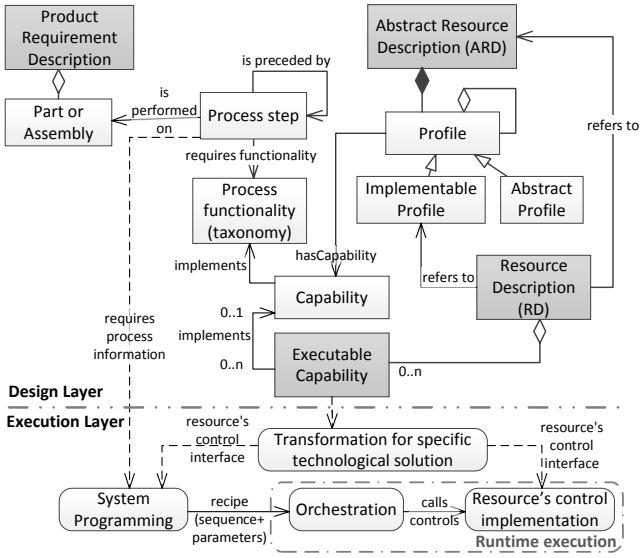

Fig. 1. Overview of associated concepts

where the resource capabilities and other characteristics are being described. The left side concentrate on describing the product requirements.

The Abstract Resource Description (ARD) concept and its Profiles define the standardisation over characteristics of different resources. The standardisation applies to Capabilities, HW interfaces, properties, and characteristics, but also to Executable Capabilities (ECs), which are assigned to a resource. The resource provider chooses one of the ARDs and one of its Profiles as basis for their resource implementation, and for the associated RD. (Siltala et al., 2016) Resource provider can then extend and add new ECs

Table 1. Definition of the key terms

Term	**Definition**
Resource	Production resource, a physical HW.
Resource Description	Formal digital representation of a resource type from a specific vendor.
Capability	Capability is process functionality that the resource or resource combination provides to the system. Capabilities have name, such as "moving", "drilling", or "fixturing", and parameters which characterize that functionality, such as "speed", "acceleration", and "force".
Executable Capability	Defines (abstract) control interface for a resource with its parameters. After applying a technological solution, it can be called at resource's implementation.
System	Implementation of a production system following selected technological solution. It includes resources and system Software (SW).
Technological solution	System design and implementation following a specific system architecture, control interface technology and/or communication protocol, such as RESTful Web Service (WS), OPC Unified Architecture (OPC UA), Message Queue Telemetry Transport (MQTT)
Orchestration	Runtime sequencer used to synchronise production operations. It executes the product recipe and calls resources' control interface(s) at right time, with right parameters. Follows a specific technological solution.

to the RD according to their resource implementation. The RD makes these publicly known for the resource users.

The Product Requirement Description represents the requirements related to the product's manufacturing. It contains the parts and assemblies of the product, and the process steps required to manufacture the product. It also represents the precedence constraints between the process steps. The process steps are linked to the required functionalities in the Process Taxonomy, where also the parametric requirements related to the functionalities are added.

Capabilities are used during the system design phase to identify, which resources fulfil the functional requirements of the product's manufacturing, while the ECs are used to program the system to perform those required functionalities in a correct sequence and with correct parameters. Resources have Capabilities and some of the Capabilities implements process function from the Process Taxonomy. (Järvenpää et al., 2016) Resource provider can connect an EC to a Capability, which means that by calling this EC, it provides an implementation for this Capability and consequently the production functionality. In such case, the execution related parameters defined for the Capability must be present also at the EC.

2.2 Structure of the Executable Capability Description

The aim of the Executable Capability (EC) concept is to define an abstract control interface of a resource in a comprehensive, formal, and vendor neutral format. It describes the actions or operations, with their parameters, which can be executed on the production resource. Examples of ECs are such as "moveToAbsolutePosition", "stopMovement", "moveToJointPosition", and "getSystemState" for a manipulator resource; "graspExternal", "release", and "closeFingers" for a gripper resource; or "screwWithTorque", and "openScrew" for a screw driver resource. Each EC contains individual set of parameters, which characterise and parametrise the requested action, and which are finally consumed by the orchestration during the execution of an action on a resource.

Fig. 2 illustrates the structure of the EC description, which is modelled in XML. For readability, the Fig. 2 shows only the main information related to EC, omitting many other elements and attributes. The complete data model of RD [1] is provided in XML Schema Definition (XSD) format. The following styles are used in the text to denote different components in the model: `Elements`, `attributes`, and *values of attributes*.

The element `ResourceDescription` is the root element for a RD. The RD has a dedicated section capturing the Executable Capability information, where the `ExecutableCapabilities` and `ExecCapas` work as container elements. The main element in the context of this paper is the `ExecCapa`, storing information about an Executable Capability. It has mandatory attributes `id` and `gid` for uniquely identifying the EC, in a local and global context correspondingly. `name` and `description` define human readable content for the EC element.

[1] http://urn.fi/urn:nbn:fi:csc-kata20180327180333533227

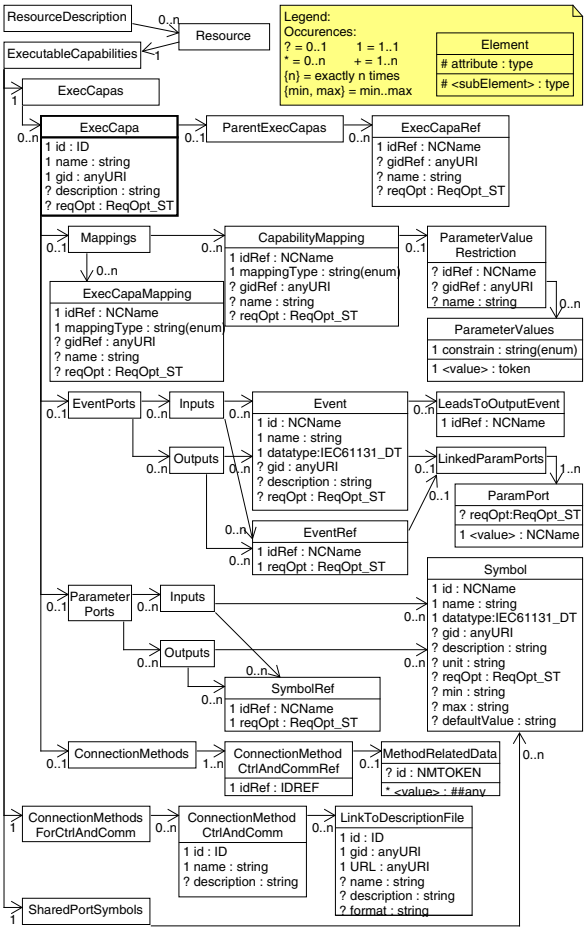

Fig. 2. Structure of Executable Capabilities in XML

`ExecCapa/ParentExecCapas/ExecCapaRef` allows to create parent-child hierarchies between ECs.

`ExecCapa/Mappings/CapabilityMapping` defines which Capability is implemented by the EC. Further restrictions can be defined with `ParameterValueRestriction` and `ParameterValues` limiting the EC implementation to only those Capabilities having named parameter set with given values.

The description of the control interface has similarities with the concept of function block interface defined in IEC-61499 (2012), having two information flows - the event flow and data flow. `ExecCapa/EventPorts/Inputs` and `/Outputs` are used to describe event information either triggering, i.e. calling the EC, or providing an output event when the execution of EC ends. This associates to the IEC-61499's event flow. `Event` itself has `id`, `gid`, `name`, and `description` identifying and describing each event. `datatype` is usually boolean, as events are usually triggers, but other data types defined in IEC-61131-3 (2013) are also available. Following the model of IEC-61499, each event is linked with parameters. This is defined by `Event/LinkedParamPorts/ParamPort` Furthermore, in case of input `Events`, `Event/LeadsToOutputEvent` can be defined, telling which output events are possible outcomes when this event is triggered. Events can be reused across

the ECs by using `EventRef` as reference to an existing `Event`.

`ExecCapa/ParameterPorts/Inputs` and `/Outputs` are used to describe the parameters associated to the events like the IEC-61499's data flow. A `Symbol` defines the parameter content. In addition to the basic information, it has `datatype` defined as enumerated string according IEC-61131-3 data types and `unit` used to define the magnitude of the parameter. SI units should be preferred. The parameter does not define the value, because it will be defined at the runtime when an event is triggered. However, the `Symbol` can define minimum and maximum boundaries for the value, and a default value giving a hint for the user from manufacturer's expected optimum. The latter is expected to match with the factory reset value. Like in case of events, parameters can also be reused by reference. This is done with `SymbolRef`. In order to assist the reuse of parameters, a higher level element `ExecutableCapabilities/SharedPortSymbols` collects together reusable `Symbol`s.

Finally, `ExecCapa` has `ConnectionMethods/Connection MethodCtrlAndCommRef` for creating a link from the EC to real implementation at the resource with some technological solution. The technological solution related implementation data is defined one level higher under `ExecutableCapabilities/ ConnectionMethodsForCtrl AndComm/ ConnectionMethodCtrlAndComm`. This refers finally to a technological solution specific descriptions, which are intended to provide enough information for the use of the resource, within the solution specific development environment.

3. EXPLOITING THE CONCEPT

The RD is intended to comprehensively capture and distribute the information about a resource, and it is provided by the resource manufacturer. The various phases of production system design and commission, and associated information systems, can make use of this information as an input source.

As the EC provides the abstract control interface(s) of a resource, a transformation (or mapping) from EC to a technological solution is needed to apply the control interface in practice. Our assumption is that by defining one transformation for each technological solution, the solution specific control interface access point can be generated for any resource from its RD, and on need basis. Both resource provider and system integrator can benefit from this. Resource provider may use it when implementing the control interfaces for their resource by following a specific technological solution. System integrators will get the access to available functions of resources through ECs and a transformation, while programming the tasks for the production system.

Fig. 3 represents how the resource and its interfaces can be utilised when the production system is integrated and commissioned. The left side of figure represents the implementation of a production resource (R1), illustrated with a green dashed border, and the right side in blue, two different system implementations (S1 and S2) each utilising a different technological solution. Following sections associated with Fig. 3 explain how the resource provider and system integrator can apply the ECs.

3.1 Resource provider view

First, the resource provider needs to describe the control interface of their resource. They can derive the basis set of ECs from ARD and its Profile while generating an initial RD. Then they can append the RD with further ECs to complete the resource's control interface. The second step is to take the resource's RD and its control interface description and provide the resource's control implementation accordingly. The implementation basis on some technological solution selected by the provider. The resource provider can utilise the transformation to generate a valid control interface template for the selected technological solution from an EC. Third, the resource provider programs the required actions starting from the generated interface template ending to the real implementation. I.e. the resource is delivered fully functioning with its logics and behaviour implemented, and it offers a compatible interface with EC. The three steps can also be reversed, but more care is needed in order to guarantee the integrity of ECs in such process.

The resource R1 (in Fig. 3) contains two Executable Capabilities (EC1 and EC2), which are defined in the RD of the resource. Each of these Executable Capabilities implements a Capability (EC1→C1 and EC2→C2). Furthermore, the RD shows that the resource provider has implemented these ECs according to three technological solutions (TS1, TS2, and TS3). In case of the first technological solution (TS1), the vendor implements only the first Executable Capability (EC1), denoted as EC1.TS1. Next, the vendor decides to support another technological solution with TS2. In this case, they implement both Executable Capabilities denoted correspondingly, EC1.TS2 and EC2.TS2.

The resource controller executes internally only one control implementation per EC, but it can offer multiple access points to utilise the EC's implementation. In other words, there is a resource internal HW-related implementation with its internal interface, which is then linked to the resource's outer interfaces of different technological solutions. For example, both the EC1.TS1 and the EC1.TS2 operate with the same implementation of the EC1 at the resource's controller (orange arrows in Fig. 3). Each resource comes with its internal implementation of control algorithms and logic, which is linked to the Executable Capabilities, and through them also to Capabilities. The internal implementation is completely hidden from the RD and resource users, thus securing the Intellectual Property (IP) of the resource provider. This part is represented with a grey ribbon at the centre and at the left rim of the R1.

3.2 System integrator view

When coming to the execution layer in Fig. 1, the production system needs to be commissioned. The programming of the system, whether (semi-)automatic or manual, needs inputs from the process steps and system layout, but also from the resources. At this stage, the EC concept provides assistance. The resource's ECs can be used to

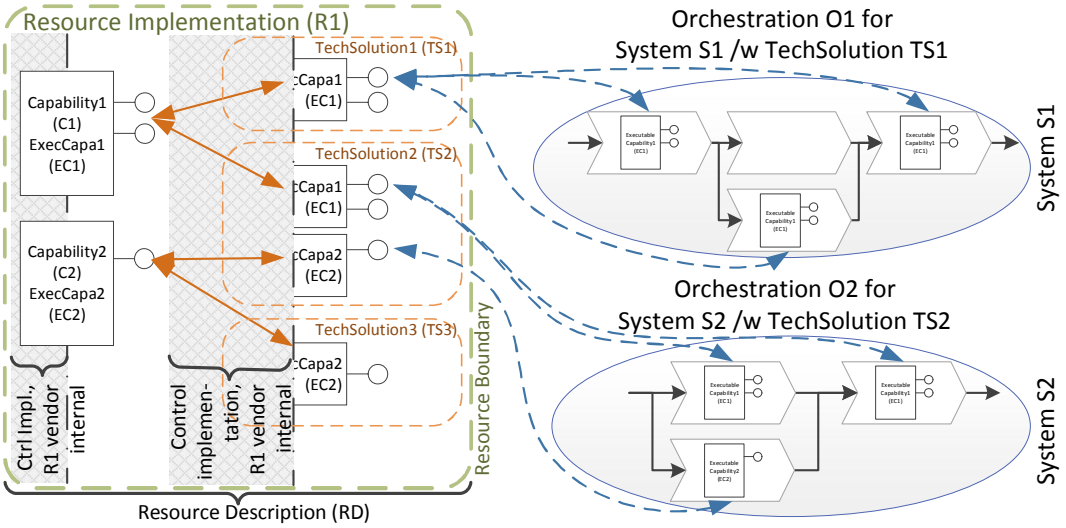

Fig. 3. Implementation and utilisation of Executable Capabilities. Modified from Siltala (2016)

create the control application for the production system, as illustrated in the right side of Fig. 3. When the system integrator implements the system, they first choose a technological solution to follow. This must reflect the selection made by the resource provider, so that there is a match between these two technological solutions – the provider (at left) meets the needs of the end user (at right). In this example, the resource (R1) is used in two different systems (S1 and S2), both of which use different technological solution.

The interface implementations of the Executable Capabilities, following selected technological solutions (e.g. EC1.TS1, EC1.TS2, and EC2.TS2), are published by the resource provider, and the resource users can access and use them through the RD. In other words, the system integrators or end users, in their system implementations based on a specific technological solution, cannot utilise directly the Capabilities (C1 and C2) and corresponding Executable Capabilities (EC1 and EC2), but only the provided implementations of technological solution.

System integrator can use the technological solution specific transformation to translate all ECs from all associated resources to control interface definitions understandable by the system level controls, and start building the control recipe for the system. The system is configured out from the modular resources and in the case of control modularity, this means orchestration of the resources and, especially, arranging and calling their ECs in a desired sequence, and with right parameters. Finally, the orchestration utilises the implemented EC interfaces (e.g. EC1.TS1) to operate the resource. For example, in the case of system S1 the application follows technological solution TS1. Thus, the resource R1 can be connected and used only through the interface EC1.TS1, making only the EC1 available for the S1.

Finally, during the system execution, orchestration engine uses the recipe information for calling and synchronising operations at system resources. The system orchestration implements, executes, and monitors the execution of the production process. The dashed blue arrows in the figure show how the S1 makes three calls to EC1.TS1 in different process phases, according to the process sequencer's commands.

The benefit of the proposed EC concept is that the recipe programming and preparation of orchestration could be performed without any access or communication with the resource and its control implementation, but to have available only the resource's RD and the transformation for the specific technological solution. The detailed system implementation and process orchestration are beyond the scope of this paper.

4. USE CASE SCENARIOS

The use case scenarios in this paper offer only a glance to the implementations. First, the ECs a gripper resource are discussed more in details and then use of transformations are discussed.

4.1 Example resource and its Executable Capabilities

We have modelled a few resources with our RD model, out of which a pneumatic gripper [2] is used as an example resource for Executable Capabilities. Especially lines 289–556 relate to the definition of ECs. The resource has seven ECs: Grasping, Grasping-External, Grasping-Internal, Release, Close fingers, Open fingers, and Get resource state. Grasping is a parent EC for external and internal grasping. Grasping-External grasps the object from outside, while the Grasping-Internal does it from inside. Both specific graspings are linked to the associated Capability called "FingerGrasping". The Release knows the internal state of the gripper and thus it is able to perform counter action for last called grasping action. Close and Open fingers do the corresponding actions, but those are not connected to any Capability. Get resource state EC is administrative in nature, and e.g. orchestration can utilise this control interface for querying the current state of the resource.

Next, the content of Grasping-External EC is looked more in detail. It uses `ParentExecCapas/ExecCapaRef` to make a reference to its parent i.e. Grasping. `Mappings/Capabili-`

[2] http://urn.fi/urn:nbn:fi:csc-kata20180327181410669395

tyMapping with value *FingerGrasping* define that this EC implements "FingerGrasping" Capability. Furthermore, ParameterValueRestriction defines a constraint. It names a Capability property called "graspingType" to have value "external", in order to meet the system design criteria. This means that this EC of Grasping-External is offered as a solution only when product requires a Capability "FingerGrasping" with property "graspingType" = "external".

Grasping-External EC for this resource has one input event (EventPorts/Inputs/Event) called *trigger*, which can lead to one of the two output events (Event/LeadsTo-OutputEvent) *done* or *error*. The trigger has no input parameters. The output events themselves are defined in EventPorts/Outputs/Event, having linked some output parameters. The done is linked with parameter *time.closing* and error with *errorCode*. Finally, the previous two parameters are defined in ParameterPorts/Outputs/Symbol.

4.2 Transformation from EC to a specific technological solution

Currently we have only one transformation available, used as a proof of concept. An XSLT is used to transform any given RD and its EC information into format used by NXT Control's engineering tool. The transformation processes EC's information and produces a set of valid files, containing the control interface information in the format of the engineering tool. These files can be opened in the engineering environment, and the resource provider can start implementing this control architecture specific implementation for the resource, which follows exactly the control interface definitions defined by the ECs.

The future work includes developing and testing more transformations (or mappers) from EC concept to other technological solutions. Proposed ones as the targets are such as OPC UA, Vorto, and AutomationML.

5. CONCLUSION

We introduced the developed Executable Capability (EC) concept and associated data model, which aim to support and speed up the development of system design, implementation, and commissioning, especially in case of system controls. The main contribution of our work is to provide continuous information chain from the product requirements to system design and finally to the execution of the processes in a production system. This is achieved by offering a method to describe the production resource's control interface in a vendor neutral form and by showing how to exploit this information in system design and commission. The standardised description of control interfaces will ease and accelerate the programming, deployment, and commissioning of the production system. The detailed system implementation and orchestration of controls were beyond the scope of this paper. Only a limited view for transformations from the ECs to control interface implementation was offered. The practical implementation and further tests remain as the future work, as well as creation of more transformations for different technological solutions.

REFERENCES

Backhaus, J. and Reinhart, G. (2017). Digital description of products, processes and resources for task-oriented programming of assembly systems. *Journal of Intelligent Manufacturing*, 28(8), 1787–1800. doi: 10.1007/s10845-015-1063-3.

Baheti, R. and Gill, H. (2011). *Cyber-physical Systems*, 161–166. IEEE Control Systems Society. URL http://ieeecss.org/general/impact-control-technology.

Bøgh, S., Nielsen, O.S., Pedersen, M.R., Krüger, V., and Madsen, O. (2012). Does your robot have skills? *Proceedings of the 43rd International Symposium on Robotics*, 6. URL http://forskningsbasen.deff.dk/Share.external?sp=S9f184d42-459a-4d30-871c-bab8082cfbba&sp=Saau.

IEC-61131-3 (2013). *IEC 61131-3:2013 Programmable controllers - Part 3: Programming languages.*

IEC-61499 (2012). *IEC 61499-1:2012 Function blocks - Part 1: Architecture.*

Järvenpää, E., Siltala, N., Hylli, O., and Lanz, M. (2017). Capability matchmaking procedure to support rapid configuration and re-configuration of production systems. *Procedia Manufacturing*, 11, 1053–1060. doi: 10.1016/j.promfg.2017.07.216.

Järvenpää, E., Siltala, N., and Lanz, M. (2016). *Formal resource and capability descriptions supporting rapid reconfiguration of assembly systems*, 120–125. IEEE. doi:10.1109/ISAM.2016.7750724.

Leitão, P., Colombo, A.W., and Karnouskos, S. (2016). Industrial automation based on cyber-physical systems technologies: Prototype implementations and challenges. *Computers in Industry*, 81, 11–25. doi: 10.1016/j.compind.2015.08.004.

Pfrommer, J., Stogl, D., Aleksandrov, K., Schubert, V., and Hein, B. (2014). *Modelling and orchestration of service-based manufacturing systems via skills*, 1–4. IEEE. doi:10.1109/ETFA.2014.7005285.

Puerto, M.J., Salle, D., Outon, J.L., Herrero, H., and Lizuain, Z. (2015). *Towards a flexible production system Environment Server implementation*, 16. IEEE. doi: 10.1109/EUROCON.2015.7313717.

Siltala, N. (2016). *Formal Digital Description of Production Equipment Modules for supporting System Design and Deployment.* Ph.D. thesis, Tampere University of Technology. URL http://urn.fi/URN:ISBN:978-952-15-3783-7.

Siltala, N., Järvenpää, E., and Lanz, M. (2016). Formal information model for representing production resources. *Advances in Production Management Systems. Initiatives for a Sustainable World. APMS 2016. IFIP Advances in Information and Communication Technology*, 488, 53–60. doi:10.1007/978-3-319-51133-7_7.

Siltala, N., Järvenpää, E., and Lanz, M. (in press, 2018). *Creating Resource Combinations Based on Formally Described Hardware Interfaces*, 11. Ifip International Federation For Information Processing.

Thoben, K.D., Wiesner, S., and Wuest, T. (2017). "industrie 4.0" and smart manufacturing - a review of research issues and application examples. *International Journal of Automation Technology*, 11(1), 4–16. doi: 10.20965/ijat.2017.p0004.