An Ontology-based Metamodel for Capability Descriptions

Michael Weser, Jürgen Bock
Corporate Research
KUKA Deutschland GmbH
Augsburg, Germany
Michael.Weser@kuka.com, Juergen.Bock@kuka.com

Siwara Schmitt
Embedded Systems Engineering
Fraunhofer IESE
Kaiserslautern, Germany
siwara.schmitt@iese.fraunhofer.de

Alexander Perzylo
Robotics
fortiss - An-Institut TU München
Munich, Germany
perzylo@fortiss.org

Kathrin Evers
Digital Business
Festo SE & Co. KG
Esslingen, Germany
kathrin.evers@festo.com

***Abstract*—This paper presents an approach to describe abilities of manufacturing resources by a formal description of capabilities using Semantic Web technologies. A hierarchical ontology architecture is proposed to represent, publish, and extend knowledge on capabilities for different application domains and use cases. Furthermore, the paper describes patterns of how the underlying formal logic can be used in taxonomy modeling and the inference of implicit capability facts. The usability and performance of the approach was validated by formalizing capability knowledge of related work and evaluated in benchmarking a prototypical implemented tool for managing and querying catalogs of resources and their capabilities. The proposed concept is intended to be used as a foundation for a future multi-layered feasibility checking, which evaluates the compatibility of resources and their offered skills with the requirements of manufacturing tasks at symbolic and subsymbolic levels. Extended evaluations might be based on parameters, analytics, simulation, and other means.**

***Index Terms*—capabilities, skills, ontology, semantic web, manufacturing**

I. Introduction

The landscape of industrial production is in constant change. For decades, costs and production rates were the most important criteria in production. Mass production systems ensured that economies of scale were achieved. However, as living standards improve, it becomes increasingly clear that the era of mass production is being replaced by an era of market niches [1]. In Europe in particular, the trend towards growing individualization with equally rising prosperity is reflected in production characterized by small, fluctuating batch sizes and quantities, high product variance [2], short product life cycles [3], and shortened lead times [4].

Future production systems must be able to deal with these challenges and have to become more flexible [5] [6]. Flexibility in production accompanies the frequent (re-)configuration of production systems. This (re-)configuration includes the assessment, whether a resource is able to execute a specific production task or not. Nowadays, this check is done by a human resource, e.g., an experienced production planner. Facing the trend toward small batch sizes, the production planning process is getting more time-consuming and cost-intensive. Depending on the availability of production planners, bottlenecks could occur that lead to delays in production and stress its financial viability. As a consequence, tools for production planning are needed that contain a formal and structured machine-interpretable description of device data and knowledge.

The research leading to these results has been funded by the German Federal Ministry of Education and Research (BMBF) under grant agreement no. 01IS16022 (BaSys 4.0) and 01IS19022 (BaSys 4.2).

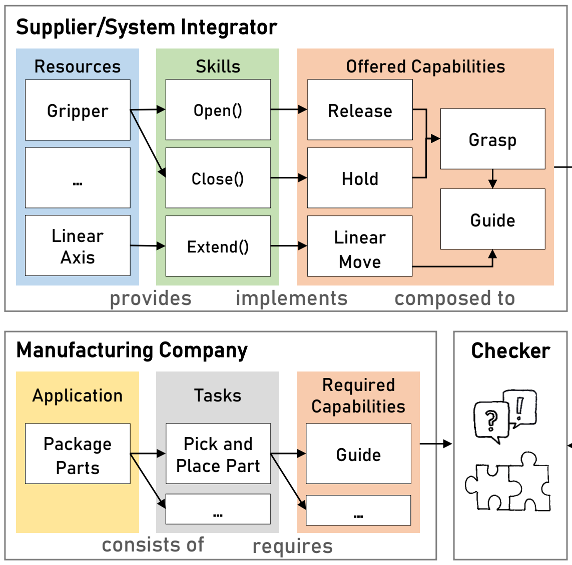


Fig. 1. Overview of capability-based production system planning

It is expected that this envisaged flexibility and changeability in production can be facilitated by formally encoding what a production process is expecting from manufacturing resources and what each resource can contribute to the production system. Based on the work of [7], this paper proposes a formal encoding of capabilities, which can realize semi-automated and in the long run even fully automated production process planning.

Fig. 1 visualizes how the concept of capabilities enables the hardware-agnostic description of manufacturing applications and associated tasks. Derived required capabilities can then be matched with the capabilities offered through skill implementations provided by manufacturing resources, such as devices, or even human workers[1].

Main use cases for capability-based production planning include:

1) A manufacturer needs to configure a new production system or reconfigure an existing production system, e.g., due to new product requirements or a broken machine.
2) A component supplier wants to automatically generate reliable solution proposals for a product catalog and to optimally match customer requirements.

Capabilities, as defined in this paper, semantically describe the effects that manufacturing resources can cause in a specific context. For instance, a drilling machine offers the capability to create a hole in a suitable material like wood or metal. Capabilities can be composed of sub-capabilities, e.g., the capability *pick-and-place* consists of the sub-capabilities *grasp*, *release*, and *move*. As discussed in [7], capabilities abstract manufacturing resource functionalities to enable the highest possible degree of changeability and flexibility in production systems. The matchmaking between required and offered capabilities shall automate the assessment, whether a device is able to execute a specific production task or not, as has been done implicitly or explicitly by human production planners so far.

In this paper, we propose an ontology-based capability description architecture using the Web Ontology Language (OWL) [8]. OWL is a widely accepted and well-defined W3C recommendation to encode and manage knowledge. It is suitable for modelling capabilities for several reasons: OWL is a formal, hence machine-interpretable knowledge representation language with a logical underpinning that ensures decidability and explainable deductions using expressing reasoning systems. Thus it is possible to automatically derive implicit facts from explicitly modeled knowledge. With this features, capabilities can be automatically composed of other capabilities or arranged in a hierarchy. Further, the automatic inference of implicit information minimizes the effort of describing capabilities by humans. Ontology models are very applicable in practice for capability descriptions, because they can be persistently hosted on servers and share capability descriptions by being accessible and referable via globally unique Internationalized Resource Identifiers (IRI). This way, ontologies can be mapped with other existing ontologies and share capability descriptions among various domains. Moreover, OWL ontologies are extendable by being importable in other ontologies. This goes along with the monotonicity of the underlying logic, meaning that knowledge added to an imported ontology can only create additional inferences and cannot invalidate previous inferences. This makes any extension of capability descriptions safe regarding consistency of inferences.

The following two sections provide a brief overview on the formal foundations of OWL ontologies and the state of the art in capability descriptions. Subsequently, this paper introduces its ontology-based capability model *Capability For Industry (C4I)* and provides an evaluation thereof. The last section concludes the paper and gives an outlook on future work.

II. Foundations

Following the definition of Studer et al. an ontology is a "formal, explicit specification of a shared conceptualisation" [9]. In this definition, *conceptualisation* refers to a model of the relevant concepts of a particular domain of interest. It is *formal* in terms of being machine readable thus excluding natural language, and *explicit* meaning that there are no hidden assumptions regarding relationships and description of concepts. The fact that it is *shared* states that the ontology represents consensual knowledge.

The Web Ontology Language (OWL) [8] is a W3C recommendation and a well-known and accepted standard for modelling ontologies. On the one hand it is based on Web standards by using IRIs as identifiers and by providing an RDF/XML serialization. On the other hand it is based on Description Logics [10] as a logical underpinning allowing for deductive reasoning.

An OWL ontology[2] is a set of axioms. An ontology can import other ontologies, which results in the set union of axioms from the imported ontologies and the importing ontology. Axioms represent statements about relationships of entities in the domain of interest. Entities are described using symbols from the ontology's vocabulary. The vocabulary consists of a set of classes, a set of instances, a set of object properties, and a set of data properties[3]. Thereby, an instance represents an object in the domain of interest. A class represents a set of instances, an object property represents a binary relation between two instances, and a data property represents a binary relation between an instance and a literal, i.e., a data value. Named classes are classes that are identified by an IRI. A (complex) class description is an anonymous characterization of a set of instances. The following class descriptions and axioms are relevant in the context of this work[4].

Let C and D be classes. The class $C \sqcap D$ denotes the class of instances that are in both C and D. The class $C \sqcup D$ denotes the class of instances that are in C or in D (or both). Let C be a class and p be a property. The class $\exists p.C$ describes the class of instances that have at least one property p to an instance of class C. The class $\forall p.C$ describes the class of instances, such that for each instance, if it has a property p, this property p relates to an instance of class C. Let $n \in \mathbb{N} \setminus \{0\}$. The class $\geq n\, p$ describes the class of instances that have at least n properties p. The class $\geq n\, p.C$ describes the class of instances that have at least n properties p to instances of class C. The property expression p^{-} states the inverse of property p, i.e.,

[1]Please note that this paper focuses on device-based resources.

[2]In this paper, *ontology* refers to an OWL ontology unless stated otherwise

[3]In this paper, *property* refers to object property, unless stated otherwise

[4]For the sake of brevity, the ontology notations follow the Description Logics notation. Refer to Baader et al. [10] for a detailed explanation of this notation and formal semantics.

iff instance a is related to instance b via p then b is related to a via p^{-}.

Let C and D be classes. The subclass (or subsumption) axiom $C \sqsubseteq D$ states, informally[5], that an instance of class C is also an instance of class D. The equivalent classes axiom $C \equiv D$ states that C and D are equivalent, i.e., $C \sqsubseteq D$ and $C \sqsupseteq D$. Axioms related to the description of classes constitute the *T-Box* of the knowledge base. Let a and b be instances. Let C be a class and r be a property. The assertion axiom $C(a)$ states that a is a member of class C. The axiom $r(a, b)$ states that a and b are connected via r, thus asserting the tuple (a, b) to r. These assertional axioms constitute the *A-Box* of the knowledge base. Let p and q be properties. The subproperty axiom $p \sqsubseteq q$ states that if two instances are related via p they are also related via q.

The special class $\top$, in OWL denoted as `owl:Thing` describes the most generic class, that is a superclass of all other classes. All instances are members of $\top$ implicitly. The class $\bot$, in OWL denoted as `owl:Nothing`, describes the most specific class, which is a subclass of all other classes. It is the empty class and contains no instances by definition.

III. Related Work

Tackling the challenges of quick changeover in lean production, the skill-based systems engineering paradigm increasingly gains interest in the research community and manufacturing companies alike. Confusion with respect to terminology seems to arise from the fact that terms are used synonymously by some, which are attributed with different meanings by others. In this paper, we distinguish the terms *skill* and *capability*. *Skills* represent the parametrizable and executable functionality of hardware or software components, and *capabilities* provide a metalevel description of their effects.

Research mainly focuses on how new skills can be implemented and how a manufacturing task can be programmed through sequences of skill invocations [11]–[14] or how production systems can be changed in order to gain or adjust skills, or to compensate the failure of parts of the system [15]. These approaches have in common that they are tool-centric. This means that a manufacturing task is described based on available hardware and software components.

By contrast, a product-centric paradigm aims at describing the product and associated production steps independent from specific production resources [16], [17], [18]. In order to identify a production resource that complies with the requirements of a hardware-agnostic task specification, its capabilities [19], [18] need to be interpreted.

[17] additionally aims at describing capabilities (called skills in this paper) in a hardware-agnostic way so that their possible effect in the world is described without naming or categorizing the solution method. This concept allows an algorithm-based planning of automated production lines, which generates a list of suitable resources for each required production step. However, a prerequisite of this approach is the description of the effect(s) that each resource may achieve – a kind of description that resource manufacturers do not provide as of now and requires a lot of initial modeling work. Work that can be considerably reduced for individual resource manufacturers when relying on already existing standards as shall be done in this paper.

There has been prior work in the German ReApp project on the composition of production systems based on a graphical workbench that combines Robot Operating System (ROS) components and OWL and Ecore models [20]. OWL ontologies are used to identify suitable ROS components via the classification of components based on offered capabilities [21]. The knowledge on how higher-level capabilities can be logically inferred from the combination of individual components was not considered.

As part of the efforts of the EU project ReCaM, an ontology-based approach to capability modeling was developed [22]. It features SPARQL Inference Notation (SPIN) rules for the orchestration of higher-level capabilities in a hierarchical fashion and conducts matchmaking queries based on modeled capabilities. This paper uses a similar approach, but tries to embed the capability concept in an open ontology architecture that is based on already available upper level ontologies and is extensible by third parties. In this approach, knowledge about the composition of capabilities and their relations is modeled directly in the ontology and not evaluated in SPARQL queries.

A similar concept of task planning by using capabilities (here skills) as an abstraction for resource functionalities is applied in the EU project MOOD2Be. As part of the EU project RobMoSys, MOOD2Be developed an approach for robot task planning by describing their capabilities and coordinating the tasks by behavior tree models [23]. The concept is role-based, so first the capabilities are getting defined by domain experts on an abstract level without any association with components that realize them. At the next step, the component developer provides a digital data sheet that is generated as a JSON file for specifying the capability realization. The robotics behavior developer imports the digital data sheet to the graphical editor Groot and plans the robot tasks by modeling a behavior tree. A behavior tree contains the logical sequence of tasks and each task has an assigned capability with specific parameter values. The translation between the abstract level of the planned tasks and the technical level of functionality execution is done by the executor BehaviorTree.CPP, a software component based on C++ [24]. MOOS2Be's and this paper's approach have in common that for the aim of more flexibility in task coordination, an abstraction of functionalities is done by defining realization-neutral capabilities [25].

IV. The Capability for Industry Ontology (C4I)

This section introduces a metamodel to define how capabilities of components can be formally described. It follows an ontology-based approach using OWL as its modelling language. Furthermore, it will be shown how the OWL-based approach allows for using this metamodel as an upper-level ontology module. To this end, modeling patterns describe best

[5]For the formal semantics of these axioms, refer to Baader et al. [10].

practices to define custom capabilities. This extension can be done in a modular way in the form of domain ontologies to describe concrete industry-specific or manufacturer-specific capabilities. The C4I meta ontology is available under its IRI: https://www.w3id.org/basyx/c4i.

A. Capability Metamodel

There are many possibilities to model a domain of interest using the description language provided by OWL. The presented metamodel follows a T-Box-based approach for the following two reasons:

1) Reasoning: The possibilities of T-Box reasoning suit the demands and expectations of capability inference (cf. Sec. I).
2) Modeling granularity: Capabilities are typically associated with hardware and software component types. If required in a later stage, single component instances can be associated with instances of the capability class, allowing for additional property assertions.

Following the T-Box modeling approach, a capability is represented as an OWL class.

$$Capability \sqsubseteq \top \tag{1}$$

In order to relate any component with capability concepts, an object property is defined as follows:

$$associatedWithCapability \sqsubseteq owl{:}topObjectProperty \tag{2}$$

$$\top \sqsubseteq \forall associatedWithCapability.Capability \tag{3}$$

This relation can be indirect, e.g., a component is associated with the capability of one of its subcomponents, although this capability might not be provided directly by the component itself. Equation 3 defines the range of $associatedWithCapability$ to be class $Capability$.

A stronger relation between components and capabilities is given by the object property

$$hasCapability \sqsubseteq associatedWithCapability \tag{4}$$

This property describes a direct relation between a component and a capability.

B. Using the Metamodel

Any specific capability model based on the metamodel defined in Sect. IV-A can define its own concrete capabilities. The modeling approach allows for building capability hierarchies originating from the $Capability$ class. To this end, capabilities can be modeled in a specialization hierarchy using the subclass axiom.

Pattern 1 (Capability Specialization): Let D be a known capability that is (directly or indirectly) a specialization of $Capability$. Let C be a new specialized capability. The axiom $C \sqsubseteq D$ states that C is a more specific capability than D. For components relating to this capability, this means that if a component is relating to C it is (implicitly) relating to D.

Pattern 2 (Capability Generalization): Let C be a known specific capability. Let D be a new capability that is more generic than C. The axiom $C \sqsubseteq D$ states that D is a more generic capability than C. If this generalization is done, it is *not* inferred that D is in fact also a capability, so an additional axiom $D \sqsubseteq Capability$ must be added at the same time.

Note, that the subclass relation and hence modeling patterns 1 and 2 do not necessarily constitute a tree, but more generically a partially ordered set of capabilities. As an example, consider a robot's capability relations $MoveOnPath \sqsubseteq Move$, $MoveToPos \sqsubseteq Move$, $MoveLin \sqsubseteq MoveOnPath$, and $MoveLin \sqsubseteq MoveToPos$. While $MoveOnPath$ and $MoveToPos$ are both specializations of $Move$, $MoveLin$ is a specialization of both $MoveOnPath$ and $MoveToPos$.

In addition to capability hierarchies, capability composition can be modeled using the $associatedWithCapability$ and $hasCapability$ properties.

Pattern 3 (Specific Capability Composition): Let $C_1, \ldots, C_n$ be capabilities that are provided by a component. Let D be a capability said component also provides due to the fact that it provides $C_i \in \{C_1, \ldots, C_n\}$. This relation can be expressed using the axiom

$$\begin{aligned} (\exists hasCapability.C_1) &\sqcap \\ (\exists hasCapability.C_2) &\sqcap \\ \ldots &\sqcap \\ (\exists hasCapability.C_n) & \\ \sqsubseteq \exists hasCapability.D) & \end{aligned} \tag{5}$$

Modeling pattern 3 can be used to describe higher-level capabilities that are based on more fine-grained capabilities provided by the same component. For example, a gripper that provides the capabilities $Hold$ and $Release$ also provides the capability $Grasp$.

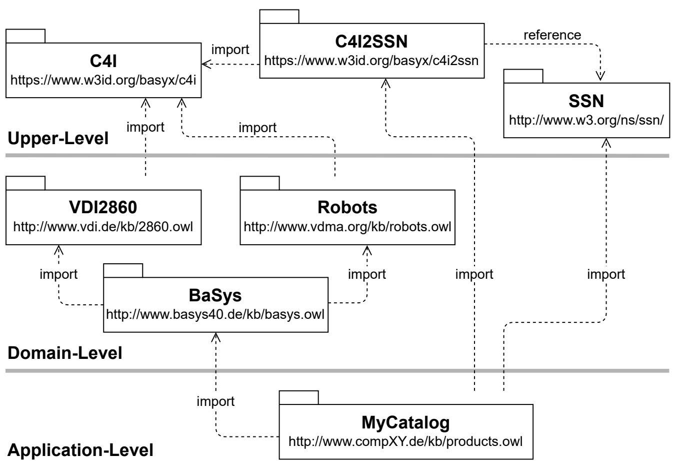
Pattern 4 (General Capability Composition): Let $C_1, \ldots, C_n$ be capabilities that are associated with a component, e.g., they are capabilities provided by subsystems of that component. Let D be a capability said component provides due to the association with each $C_i \in \{C_1, \ldots, C_n\}$. This relation can be expressed using the axiom

$$\begin{aligned} (\exists associatedWithCapability.C_1) &\sqcap \\ (\exists associatedWithCapability.C_2) &\sqcap \\ \ldots &\sqcap \\ (\exists associatedWithCapability.C_n) & \\ \sqsubseteq \exists hasCapability.D) & \end{aligned} \tag{6}$$

Modelling pattern 4 has a typical use case when describing composite components. For example, a robot system composed of a manipulator and a gripper is associated with the capabilities $MoveToPos$ and $Grasp$, resp., and thus provides the capability $PickAndPlace$.

C. Modularization Concept

A component-independent and overarching capability model can bring added value best, if it is widely accepted, easy to share and reuse, easy to extend and maintain, as well as easy to integrate with further existing domain models. The proposed



C4I
https://www.w3id.org/basyx/c4i
C4I2SSN
https://www.w3id.org/basyx/c4i2ssn
SSN
http://www.w3.org/ns/ssn/
import
reference
Upper-Level
VDI2860
http://www.vdi.de/kb/2860.owl
Robots
http://www.vdma.org/kb/robots.owl
BaSys
http://www.basys40.de/kb/basys.owl
Domain-Level
MyCatalog
http://www.compXY.de/kb/products.owl
Application-Level

Fig. 2. Modular ontology architecture

modeling approach fulfils all these requirements by facilitating a modular ontology model.

A common practice in the Semantic Web is the division into upper-level, domain-level, and application-level knowledge. Upper-level ontologies capture general relationships and axioms that are necessary to describe domain knowledge. The ontology containing the metamodel defined in Sect. IV-A is such an upper-level ontology in the context of capability modeling.

Ontologies that subsequently model knowledge from a certain area of a domain use the specifications from the upper-level ontology, typically through OWL's import axiom. Knowledge from several ontologies, which adhere to the definitions of an upper-level ontology, can thus be combined and reused more easily. In the context of capability modeling, this adherence is achieved by following the modeling patterns as described in Sect. IV-B. The different ontology modules facilitate the separation of concerns and responsibilities, as the modules can be authored and maintained by different organisations, associations, component providers, etc.

Fig. 2 shows an example of such a modularized ontology model using the ontology import mechanism. The upper-level ontology C4I contains the metamodel as described in Sect. IV-A. As an example, domain ontologies for the assembly and handling domain (based on the VDI 2860 guideline [26]), and the robotics domain are shown, as well as a specific domain context (here, the BaSys project), which reuses knowledge from both the VDI 2860 and the robotics domain. On demand, the model can also be extended on an application-specific level.

D. Mapping to Existing Ontologies

Motivated by the paradigm of the Semantic Web, existing ontologies should be used or extended where possible, resulting in a higher degree of maturity and promoting greater acceptance.

The Semantic Sensor Network Ontology (SSN) [27] has been identified to cover a domain of interest related to the domain addressed in this work. SSN itself includes the concept of capabilities of systems, but has a focus on systems and sensors. This makes it an ideal candidate to be reused and extended by the capability model presented in this approach. However, instead of importing SSN, a different mapping approach was taken, similar to the mapping from SSN to the upper-level ontology Dolce UltraLight (DUL) [28].

In this way of mapping, the SSN ontology is not directly imported by the C4I ontology, but a separate mapping ontology is provided instead. This mapping ontology imports both C4I and SSN and adds specific mapping axioms to describe the relationships between concepts from C4I and SSN.

The advantage of this approach is to avoid a large overhead of only remotely related SSN terms in applications in which these are not required, as is the case in most capability matching scenarios. However, if there is a demand to include SSN concepts, an application ontology can additionally import the mapping ontology and the SSN ontology, as depicted in Fig. 2. In this case, the relation to SSN concepts will be implicitly given for all capabilities provided by any domain ontology in the application ontology's import closure.

V. Validation

The patterns presented in Sect. IV are a result of a iterative process of design and validation. This section focuses on the validation of the C4I ontology.

To validate a novel approach it is reasonable to take existing state of the art into account. For the definition of capabilities in industry, the VDI standard 2860 [26] was already used (see [22]). The VDI 2860 standard defines common handling tasks in industry on an abstract symbolic level. Furthermore, tasks are also separated into elementary tasks, tasks that can be derived from elementary ones, or combinations of tasks. In the following, it is shown how VDI 2860 statements can be formalized following the C4I metamodel.

First, a domain ontology to hold the VDI 2860 definitions is created by following the modularization concept of Sect. IV-C. Therefore, the C4I upper ontology is imported. Next, a capability hierarchy as proposed in Sect. IV-A is set up for every task defined in VDI 2860. Finally, following the patterns from Sect. IV-B, dependencies between tasks are formalized.

For example, the dependency between ”Hold” and ”Tension” can be modeled as part of creating a capability hierarchy using pattern 1, stating that ”Tension” is a specialization of ”Hold”, which is itself (indirectly) a specialization of $Capability$:

$$Tension \sqsubseteq Hold \tag{7}$$

given that

$$Hold \sqsubseteq \ldots \sqsubseteq Capability \tag{8}$$

This modeling can be reapplied in a top-down or bottom-up manner to build a capability hierarchy for a specific domain. However, it must be ensured that the most general capability (or capabilities) must directly or indirectly be a subclass (or subclasses, resp.) of $Capability$.

Pattern 4 can be used to state dependencies of combinations of atomic capabilities. The task ”Slew” can be defined as overlay of ”Shift” and ”Turn”, which was formalized in C4I by the following axiom:

$$\begin{aligned}(\exists associatedWithCapability.Shift) \sqcap \\ (\exists associatedWithCapability.Turn) \\ \sqsubseteq \exists hasCapability.Slew) \qquad (9)\end{aligned}$$

For validation purposes, besides the VDI 2860 ontology, further domain ontologies were created, as shown in Fig. 2. The domain ontology ”BaSys” reuses definitions from both domains ”VDI2860” and ”Robots”. By using pattern 3 the capability to ”Grasp” is defined as[6]

$$\begin{aligned}(\exists hasCapability.vdi2860{:}Hold) \sqcap \\ (\exists hasCapability.vdi2860{:}Release) \\ \sqsubseteq \exists hasCapability.basys{:}Grasp \qquad (10)\end{aligned}$$

Pattern 3 was used in this case, as the capability to grasp can not be fulfilled by two different components – one providing ”Hold” while the other provides ”Release”. Both capabilities have to be provided by a single component.

Also the mapping ontology C4I2SSN shown in Fig. 2 and described by Sect. IV-D was validated by creating a catalog ontology (Fig. 2 ”MyCatalog”) holding components with their capabilities. Via the SSN ”subSystem” property also sub component dependencies could be modeled. While first application ontologies were created manually, a software tool was implemented for managing component catalogs modeled with C4I and SSN.

In order to validate the usage of the capability mondel, a capability checker service was implemented using the Java™ Spring™ microservice framework. The service, accessible via a REST API shown in Fig. 3, holds multiple catalogs of components which are organized and persisted via the Spring session mechanisms. The single catalogs or sessions can be identified and accessed by an X-Auth-Token.

The checker utilizes ontology reasoning to check whether capabilities are compatible, infer implicit capabilities of components, or find components offering a specific capability. For example, a request to get all components from a catalog that have a certain capability, implicit knowledge is needed. Therefore, inference is done via the OWL API [29] at request time.

VI. Evaluation

Besides validating the concepts of the C4I modeling approach, a first run time evaluation was conducted to assess the performance under real application conditions. The already introduced checker service presented in Sect.V was used to perform the tests as a representative service that consumes the capability model.

[6]For clarification, the namespaces *vdi2860*: and *basys*: were added in equation 10

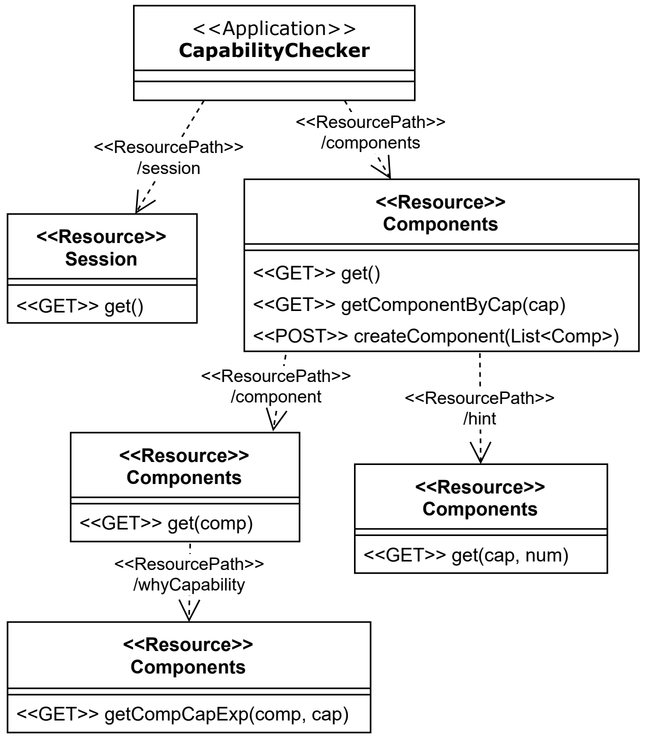


Fig. 3. capability checker service REST API

The evaluation comprises run time measurements of typical functionalities of the checker service. Of particular interest is the effect of capability complexity per component and number of components per catalog on the run time performance.

From a theoretical point of view, reasoning on the proposed ontologies can be done in polynomial time complexity, as the C4I patterns and the C4I2SSN mapping axioms can be expressed in the tractable OWL 2 EL profile. Hence, efficient reasoners, such as ELK [30] can be used.

On the practical side, a first performance study was conducted as follows. To reduce disturbances due to the Java™ execution environment, an OpenJDK™ `jmh` benchmark application was implemented to set up the test catalogs, execute warm up and measurement iterations. As an evaluation result, the average execution time of a service function was recorded.

The diagram in Fig. 4 shows the average execution time for a request to get all components of a catalog (see Fig. 3 ”⟨⟨Get⟩⟩`get()`” function for Components). This function returns explicit and implicit information about the components of a catalog, hence execution includes reasoning.

The retrieval of all components of a catalog is equivalent to a data export and thus does not provide the intended function of the checker to answer individual capability requests. Nevertheless, this benchmark shows which loads can be expected and have to be handled in the worst case.

The number of components of a catalog was increased from 10 to 1000 using a step size of 200. This was repeated for multiple capability assignments per component with varying complexity. The first data series (”No Cap.“) shows the results

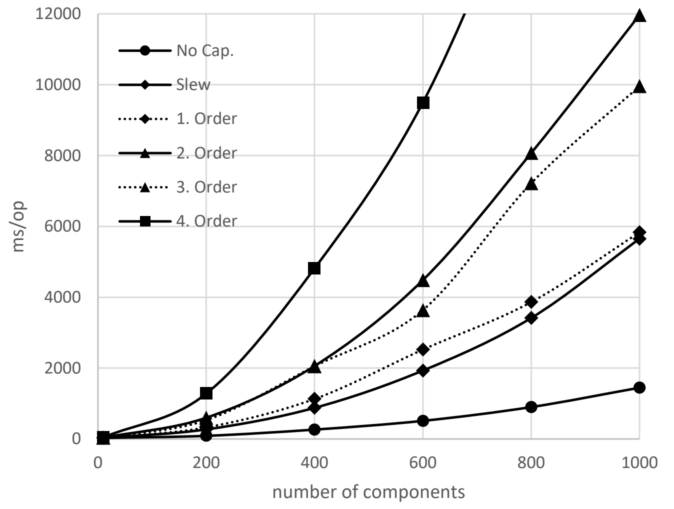


Fig. 4. Average execution time to get all components with inferred capabilities of a catalog

without any assigned capabilities and therefore was taken as a base line reference.

For the next series ("Slew"), two capabilities "Shift" and "Turn" leading to "Slew" were assigned[7]

$$Shift \land Turn \rightarrow Slew$$

The capability to order can be achieved by different combinations of elementary capabilities. The first possibility ("1. Order") has the same complexity as "Slew", here, "Position" and "Orient" leading to "Order.

$$Position \land Orient \rightarrow Order$$

The series "2. Order" adds more complexity by first inferring "Position" and "Orient" from the following 4 capabilities as shown below.

$$\begin{aligned} Shift \land CheckPosition &\rightarrow Position \\ Turn \land CheckOrientation &\rightarrow Orient \end{aligned}$$

Series "3. Order" also infers "Order" from 4 assigned capabilities but without intermediate capabilities.

$$CheckOrientation \land Branch \land Guide \land Convey \rightarrow Order$$

The last series ("4. Order") increases the complexity by adding inference for "Branch" and "Guide" from series "3. Order" and combines it with approach of series "2. Order". In the end 7 capabilities are assigned to one component giving 8 inferred implicit capabilities.

$$\begin{aligned} Turn \land Shift &\rightarrow Pass \\ Pass \land Divide &\rightarrow Branch \\ Pass \land Hold &\rightarrow Guide \end{aligned}$$

[7]For the sake of brevity, ontological axioms to express relationships between capabilities, such as in Eq. (9), are presented in simplified notation.

By investigating the resulting data it could be seen, that the number of components had a non linear, approximately quadratic, influence on the execution time. This behaviour could be explained by the implementation of the service, as for every component a query to the reasoner, to get inferred capabilities, was executed. Therefore, the impact of increasing the number of components on the reasoning time is multiplied by the number of components.

To overcome this behaviour, it is advisable to materialize the implicit capabilities of the components at creation time. A later retrieval is then equivalent to a normal database lookup.

Comparing the curves of the data series, it could be clearly observed, that the execution time increases for capability reasoning. This influence was mainly attributed to the assigned number of capabilities per component. On the other hand, it could not yet be confirmed that different modelling approaches for the allocation of capabilities have a main influence.

This becomes noticeable when comparing series "2. Order" and "3. Order", which assign the same number of capabilities but for series "2. Order", "Position" and "Orient" have to be inferred themselves. It can be observed, that there is a slightly different behaviour after 400 components. However, it cannot be ensured that this is clearly affected by the capability modelling choices and motivates further performance studies as part of future research.

Overall the results show that performance for some functions can be further optimized by materialization at creation time, but also imply that reasoning at request time still is feasible and required. For example the request to get a hint, which components of a catalog to combine to achieve a certain capability (see Fig. 3 "⟨⟨Get⟩⟩`get(cap, num)`" for path "`/components/hint`"), would need the materialization of every possible combination, following the C4I patterns, of all components of a catalog. Therefore, future focus will be on reasoning performance for such use cases.

VII. Conclusion

This paper introduces the concept of ontology-based capability matching for production resources. It shows that the formal representation of ontologies is a suitable technology to encode knowledge that previously was often described in an unstructured way or even not at all. Through a distinctive separation from programming language-specific implementations, ontology models can be more easily maintained and reused. The specification of capabilities based on a standardized vocabulary enables the seamless substitution and interplay of resources from different manufacturers.

As a purely symbolic matching of a task's requirements with a production resource's capabilities cannot consider complex real-world properties, e.g., friction coefficients or temperature influences, adding additional subsymbolic layers of compatibility checks enables the knowledge-based production system to increase the level of confidence for its compatibility predictions. In order to limit the extra effort caused by the additional checks, the symbolic evaluation is used to filter possible candidates, i.e., to reduce the search space of possible solutions. Based on a risk assessment of particular production steps, a certain confidence level of assumed compatibility could be demanded.

Composing capabilities allows to build up complex systems from scratch or based on preexisting resources. Similarly, decomposing capabilities could be used to perform tasks with resources that can do more than the task at hand. Therefore those mechanisms are of high value in plug-and-produce or interchangeability scenarios. There are two ways to build up such compositions: either rule-based or learned. In both cases, one has to rely on a publicly known taxonomy of capabilities. A prerequisite of composing and decomposing resources is to have a mechanism to formally represent resources including their composition or decomposition structure. This can be done in various ways. By expressing it in an ontology as well, it can be automatically reasoned upon. A potential mechanism to integrate this approach into production systems is to use the Asset Administration Shell in the Industry 4.0 context and introduce a new reference that carries a semantic ID, which links to concepts of the ontology presented in this paper.

Acknowledgment

We would like to thank our colleague Pierre Venet for providing his insights during fruitful discussions and expertise in software implementation.

References

[1] G. Chryssolouris, *Manufacturing Systems: Theory and Practice*. Springer Science & Business Media, Mar. 2013.

[2] J. Spingler and S. Thiemermann, "Direkte Mensch-Roboter-Kooperation am Beispiel einer flexiblen Montagezelle," *ZWF - Zeitschrift für wirtschaftlichen Fabrikbetrieb*, vol. 96, no. 11-12, pp. 616–620, 2001.

[3] K. Abd, K. Abhary, and R. Marian, "A scheduling framework for robotic flexible assembly cells," *KMUTNB: International Journal of Applied Science and Technology*, vol. 4, no. 1, pp. 31–38, 2013.

[4] J. e. A. W. M. Vos, "Module and System Design in Flexible Automated Assembly," Ph.D. dissertation, TU Delft, Delft University of Technology, Jun. 2001. [Online]. Available: "http://resolver.tudelft.nl/uuid:f5aa8b86-0bec-4e3a-9c0c-245361136ad6"

[5] K. K. Abd, *Intelligent Scheduling of Robotic Flexible Assembly Cells*, ser. Springer Theses. Cham: Springer International Publishing, 2016. [Online]. Available: http://link.springer.com/10.1007/978-3-319-26296-3

[6] A. Perzylo, M. Rickert, B. Kahl, N. Somani, C. Lehmann, A. Kuss, S. Profanter, A. B. Beck, M. Haage, M. R. Hansen, M. T. Nibe, M. A. Roa, O. Sörnmo, S. G. Robertz, U. Thomas, G. Veiga, E. A. Topp, I. Kessler, and M. Danzer, "SMErobotics: Smart robots for flexible manufacturing," *IEEE Robotics & Automation Magazine*, vol. 26, no. 1, pp. 78–90, Mar. 2019.

[7] S. Malakuti, J. Bock, M. Weser, P. Venet, P. Zimmermann, M. Wiegand, J. Grothoff, C. Wagner, and A. Bayha, "Challenges in skill-based engineering of industrial automation systems*," in *2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA)*, vol. 1, 2018, pp. 67–74.

[8] B. Parsia, P. Patel-Schneider, and B. Motik, "OWL 2 Web Ontology Language Structural Specification and Functional-Style Syntax (Second Edition)," W3C, W3C Recommendation, Dec. 2012, http://www.w3.org/TR/2012/REC-owl2-syntax-20121211/. [Online]. Available: http://www.w3.org/TR/2012/REC-owl2-syntax-20121211/

[9] R. Studer, V. R. Benjamins, and D. Fensel, "Knowledge Engineering: Principles and Methods," *Data & Knowledge Engineering*, vol. 25, no. 1-2, pp. 161–197, Mar. 1998.

[10] F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, and P. F. Patel-Schneider, Eds., *The Description Logic Handbook: Theory, Implementation and Applications*. New York, NY, United States: Cambridge University Press, May 2010.

[11] U. Thomas, G. Hirzinger, B. Rumpe, C. Schulze, and A. Wortmann, "A new skill based robot programming language using UML/P statecharts," in *IEEE International Conference on Robotics and Automation (ICRA)*, May 2013, pp. 461–466.

[12] M. Stenmark, M. Haage, and E. A. Topp, "Simplified programming of re-usable skills on a safe industrial robot: Prototype and evaluation," in *ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, New York, NY, USA, 2017, pp. 463–472. [Online]. Available: http://doi.acm.org/10.1145/2909824.3020227

[13] H. Herrero, A. A. Moughlbay, J. L. Outón, D. Sallé, and K. L. de Ipiña, "Skill based robot programming: Assembly, vision and workspace monitoring skill interaction," *Neurocomputing*, vol. 255, pp. 61 – 70, 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0925231217305441

[14] C. Schou, R. S. Andersen, D. Chrysostomou, S. Bøgh, and O. Madsen, "Skill-based instruction of collaborative robots in industrial settings," *Robotics and Computer-Integrated Manufacturing*, vol. 53, pp. 72 – 80, 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0736584516301910

[15] X. Hoang and A. Fay, "A capability model for the adaptation of manufacturing systems," in *2019 24th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, 2019, pp. 1053–1060.

[16] A. Perzylo, N. Somani, S. Profanter, I. Kessler, M. Rickert, and A. Knoll, "Intuitive instruction of industrial robots: Semantic process descriptions for small lot production," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Daejeon, Republic of Korea, Oct. 2016, pp. 2293–2300.

[17] K. Evers, J. R. Seyler, V. Aravantinos, L. Lucio, and A. Mehdi, "Roadmap to skill based systems engineering," *2019 24th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, pp. 1093–1100, 2019.

[18] D. Smale and S. Ratchev, "A capability model and taxonomy for multiple assembly system reconfigurations," *IFAC Proceedings Volumes*, vol. 42, no. 4, pp. 1923–1928, 2009.

[19] A. Perzylo, J. Grothoff, L. Lucio, M. Weser, S. Malakuti, P. Venet, V. Aravantinos, and T. Deppe, "Capability-based semantic interoperability of manufacturing resources: A BaSys 4.0 perspective," in *IFAC Conference on Manufacturing Modelling, Management, and Control (MIM)*, Berlin, Germany, Aug. 2019.

[20] M. Wenger, W. Eisenmenger, G. Neugschwandtner, B. Schneider, and A. Zoitl, "A model based engineering tool for ros component compositioning, configuration and generation of deployment information," in *IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, 2016.

[21] S. Zander, G. Heppner, G. Neugschwandtner, R. Awad, M. Essinger, and N. Ahmed, "A model-driven engineering approach for ROS using ontological semantics," *arXiv preprint arXiv:1601.03998*, 2016.

[22] E. Järvenpää, N. Siltala, O. Hylli, and M. Lanz, "The development of an ontology for describing the capabilities of manufacturing resources," *Journal of Intelligent Manufacturing*, 6 2018.

[23] MOOD2Be. [Online]. Available: https://robmosys.eu/mood2be/

[24] RobMoSys. (2018, dec) Robotic Behavior in RobMoSys using Behavior Trees and the SmartMDSD Toolchain (MOOD2be ITP). [Online]. Available: https://www.youtube.com/watch?v=_EwNZG5Xo1k

[25] A. Lotz, M. Lutz, and D. Stamper. (2017 - 2018) Introduction to System Composition in an Ecosystem. [Online]. Available: https://robmosys.eu/wiki/composition:introduction

[26] Verein deutscher Ingenieure, *VDI 2860*, ser. VDI-Richtlinien. Düsseldorf: Beuth, 1990.

[27] S. Cox, K. Taylor, M. Lefrançois, K. Janowicz, D. L. Phuoc, and A. Haller, "Semantic Sensor Network Ontology," W3C, W3C Recommendation, Oct. 2017. [Online]. Available: https://www.w3.org/TR/2017/REC-vocab-ssn-20171019/

[28] A. Gangemi, M. Uschold, and E. Daga, "Ontology:DOLCE+DnS Ultralite," OntologyDesignPatterns.org (ODP), Mar. 2010. [Online]. Available: http://ontologydesignpatterns.org/wiki/Ontology:DOLCE+DnS_Ultralite

[29] M. Horridge and S. Bechhofer, "The owl api: A java api for owl ontologies," *Semant. Web*, vol. 2, no. 1, p. 11–21, Jan. 2011.

[30] Y. Kazakov, M. Krötzsch, and F. Simančík, "The incredible ELK: From polynomial procedures to efficient reasoning with $\mathcal{EL}$ ontologies," *Journal of Automated Reasoning*, vol. 53, pp. 1–61, 2013.