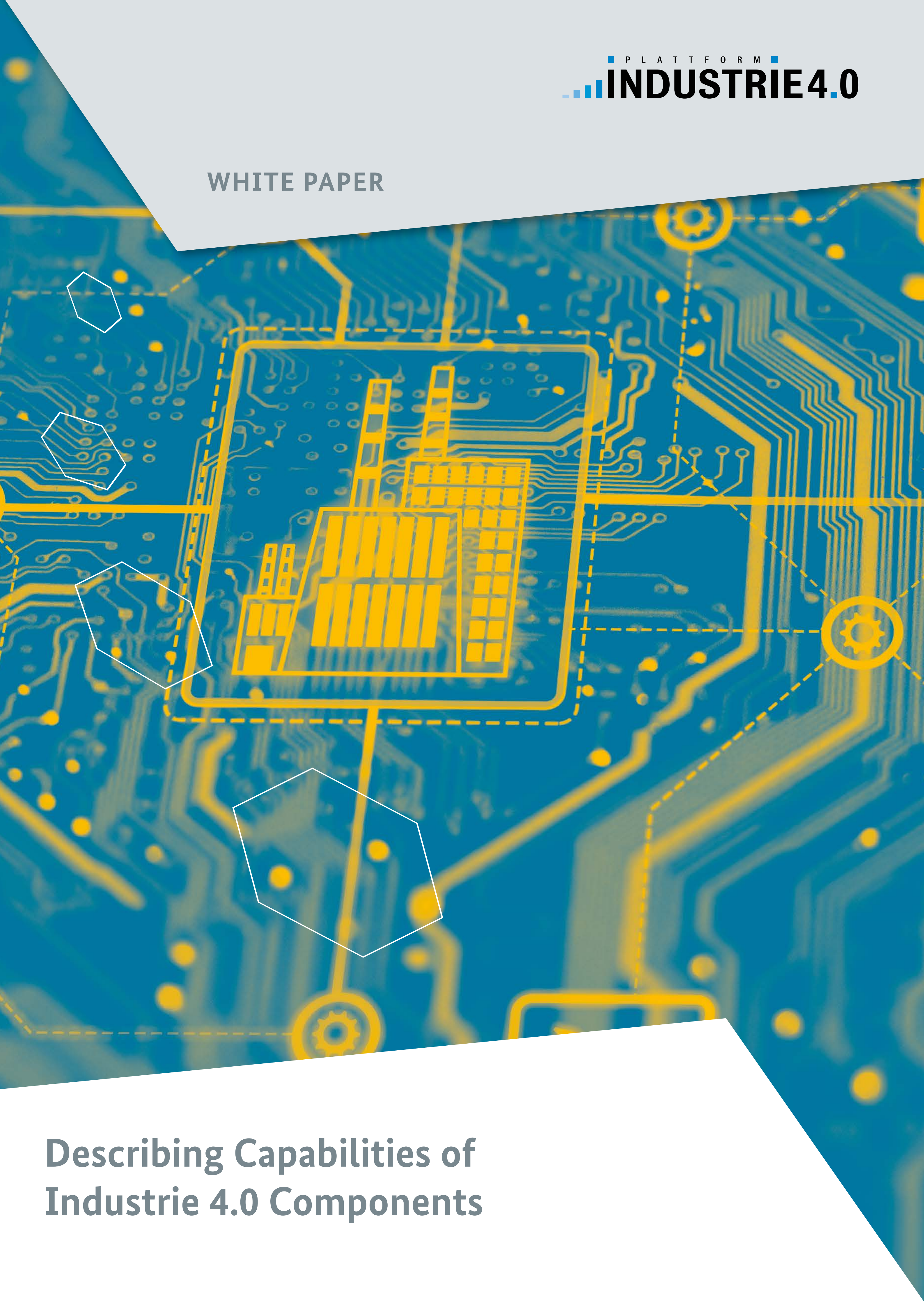
PLATTFORM INDUSTRIE 4.0

WHITE PAPER



Describing Capabilities of Industrie 4.0 Components

Imprint

Publisher
Federal Ministry for
Economic Affairs and Energy (BMWi)
Public Relations
10119 Berlin
www.bmwi.de

Editorial responsibility
Geschäftsstelle Plattform Industrie 4.0
Bülowstraße 78
10783 Berlin

Design
PRpetuum GmbH, Munich

Status
November 2020

Image credits
Michael Traitov – AdobeStock (title)
danchooalex – iStock (p. 5)
Alexander Limbach – AdobeStock (p. 8)
WrightStudio – AdobeStock (p. 10)
Blue Planet Studio – AdobeStock (p. 16)

You can obtain this and other brochures from:
Federal Ministry for Economic Affairs
and Energy (BMWi)
Public Relations
Email: publikationen@bundesregierung.de
www.bmwi.de

Central ordering service:
Tel.: +49 30 182722721
Fax: +49 30 18102722721

Contents

1 Introduction

Digitalization leads to increasing amounts of data. This huge amount of data can only be processed and used in a productive way, if machine interpretable information with a sufficient degree of formalism is available. This is why the huge amount of data can only be processed and used in a productive way if machine-interpretable information is available. This is not only true for dynamic online data but also for descriptions of functions, structure and features of the technical system that are used for the industrial production.

A promising goal is the assistance or even the automatic processing of engineering tasks. I4.0 Components are consisting out of assets and their digital twin, which is in the concept of Industrie 4.0 the Asset Administration Shell. The assets are the building blocks of the technical system which can be devices, machines or plants. During engineering technical descriptions of the asset are used to understand and identify those assets which offer the necessary functions and features for the intended task and position in the devices, machines or plants. The Asset Administration Shell is a standardized means for these descriptions.

The Asset Administration Shell can support assistance and engineering tasks by not only providing standardized and machine interpretable data but additionally providing information about its offered functionality. This whitepaper describes the means of capability and skill to achieve this added value. One prerequisite is the semi-formal description of these capabilities and skills. Therefore, this paper suggests combining property descriptions with means of ontologies.

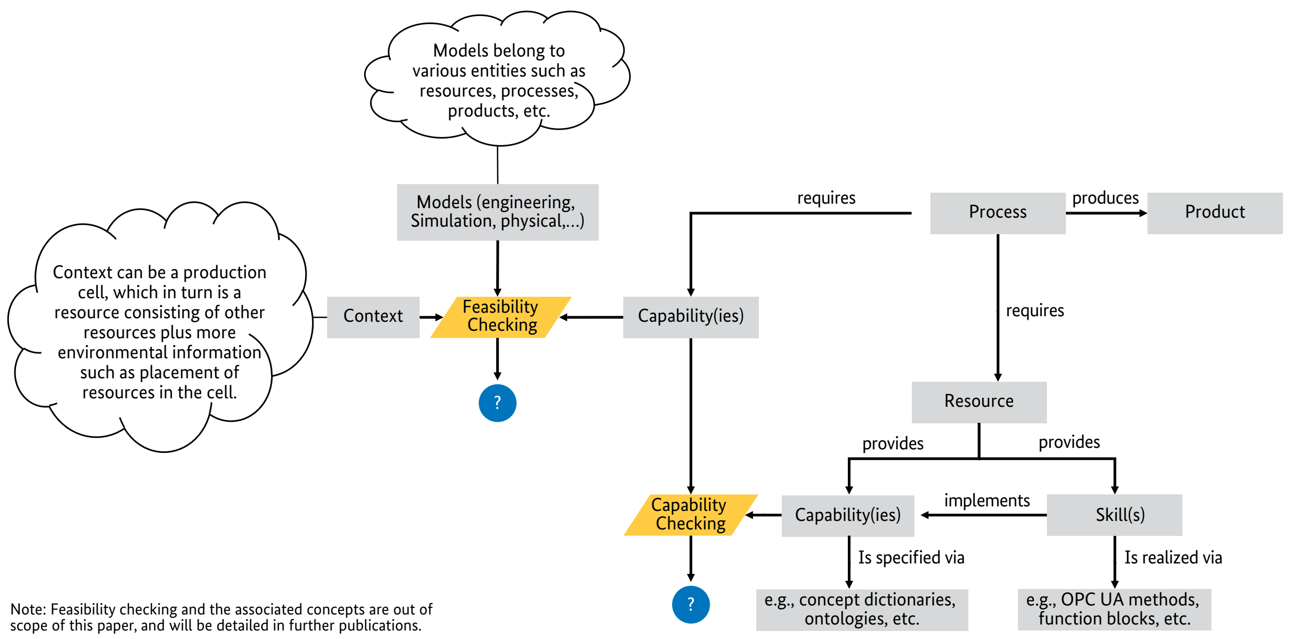
In order to facilitate communication about capabilities in a machine-interpretable manner it is crucial that assets sharing information about capabilities speak a common language. To this end, this whitepaper proposes an approach that decouples the description of capabilities from the individual assets (and Asset Administration Shells). Asset Administration Shells from arbitrary assets can refer to this shared model and express a request or offer of capabilities in a way that is understood by other assets.

2 Terminology

The following terms are used in this document, whose relations to each other are depicted in Figure 1.

Asset:	Entity which is owned by or under the custodial duties of an organization, having either a perceived or actual value to the organization [Plattform]
Capability:	The implementation-independent description of the function of a resource to achieve a certain effect in the physical or virtual world.
Capability Checking:	A formal procedure to assess the fulfillment of a required capability against the provided capabilities of a resource
Concept Dictionary:	The collection of semantic description of processes, resources and products
Context:	Additional information from a relationship or an environment that can be taken into consideration [Plattform]
Feasibility Checking:	A formal procedure to assess the possibility to achieve the desired effect of a skill execution in a concrete context
Function Block:	A stateful functional element in a system that interacts with other elements via input and output variables and performs a computation or an action and thus one way of realizing a skill[1]
Process:	Entirety of procedures in a system by means of which the material, energy or information is transformed, transported or stored [Plattform]
Product:	The intermediate or final merchandise that is created as a result of a process step
Model:	Coherent, sufficiently detailed abstraction of aspects within a field of application [Plattform]
Ontology:	A formal, explicit specification of a shared conceptualization (i.e., an abstract model of some phenomenon in the world by having identified the relevant concepts of that phenomenon) [Studer]
	Resource: An asset that is used in a process to perform the procedures
Skill:	The asset-dependent implementation of the function of a resource to achieve a certain effect in the physical or virtual world

Figure 1: Relationships among Various Terms


Models belong to various entities such as resources, processes, products, etc.
Models (engineering, Simulation, physical,...)
requires
Process
produces
Product
Context can be a production cell, which in turn is a resource consisting of other resources plus more environmental information such as placement of resources in the cell.
Context
Feasibility Checking
Capability(ies)
requires
?
Resource
provides
provides
Capability Checking
Capability(ies)
implements
Skill(s)
Is specified via
Is realized via
?
e.g., concept dictionaries, ontologies, etc.
e.g., OPC UA methods, function blocks, etc.

Note: Feasibility checking and the associated concepts are out of scope of this paper, and will be detailed in further publications.

Source: Plattform Industrie 4.0

1 Please refer also to IEC 61804-2 and IEC 61499

3 Capability-based (Continuous) Engineering and Operation

3.1 Engineering and Operation of Industrie 3.0 Systems

The rough practice in designing Industrie 3.0 production systems is to start by defining the basic production concept. For example, a robot must move an object from point A to B, or a liquid is mixed, heated, stirred and filled afterwards. Afterwards, process engineers create and document a process for the realization of the production concept, by defining the required capabilities and functionalities such as the capability of grasping objects, moving, drilling, etc. This phase still abstracts from the concrete hardware. Typically, the objects and connections during the planning are placeholder for future technical realizations, they are the requirements of them. In the next step, the appropriate resources are selected from the vendors' catalogs and data sheets based on the requirements defined in the previous step.

The outcome will be a list of physical equipment such as assembly systems, handling systems and transport systems including the automation and IT related equipment such as actuators, sensors, controllers, IT-infrastructure as well as software components such as engineering tools, firmware, libraries, SCADA, MES and ERP. The detailed planning is the next step, which consists of developing and engineering the source code for the controllers, planning the electrical, pneumatic and mechanical components and the IT-configuration. If necessary, simulations are conducted to ensure the feasibility of the system operations, construction plans are finalized and order lists are created. After having real resources in place, the system is built up, e. g., the equipment, the IT-infrastructure, the electronic connections and the automation solutions. During this process, the automation source code is loaded, parameterized and the internal functions are tested. Finally, the system goes through acceptance test and the commissioning. During the operation, errors may occur, which need to be detected and healed by the operators.

3.2 Capability-based (Continuous) Engineering and Operation of Industrie 4.0 Systems

Industrie 4.0 systems are to enable new use cases or improve the efficiency of existing use cases, e.g., having lot-size-one systems, which can produce increased variety of products in a flexible and timely manner, or simplifying and making commissioning and maintenance more flexible via the plug and produce concept, etc. Naturally, the traditional way of engineering and operating systems falls short of enabling these.

We consider capability-based engineering and operation of systems as the key enabler for various Industrie 4.0 systems. Here, the main goal is to design and operate systems based on the required capabilities for each step of a production process, instead of explicitly specifying actual production resources. There are different scenarios:

The actual production process sequence is decided during production and not in the design phase. The capability check evaluates candidate options and the feasibility check decides which options could actually be used. Additionally, a permanent check of free capacities is necessary.

If there is a new or unknown variant of a product with a known production process description it can be evaluated which assets are able to offer the necessary capability.

For example, in a simple "Device Replacement" use case, capability descriptions simplify the task of maintenance operators to find devices that offer similar capabilities to a defected device, even if the devices are not the same. In the "Lot-Size-One" use cases, one must identify whether it is possible to manufacture a new product by flexibly using the current installed resources of a plant; if not, they must identify which resources within the manufacturing system have to be added or replaced to manufacture the new product. In the "Plug and Produce" use case, one must be able to plug a new Industrie 4.0 component that offers certain capabilities within a plant, the component must be discovered, parameterized and start interacting with the rest of components within the plant.

Capability-based engineering and operation of systems can be continuous, meaning that production systems can be changed during their operation without (or with low) interruption of the production process.

Three elements play a major role to achieve capability-based (continuous) engineering and operation: Process, Product, and Resource, or the so-called PPR model. Here, resources are aware of their own capabilities to make certain effects, without knowing in which processes and for which products they will be utilized. Each process specifies the required capabilities. The properties of processes and products, as well as the properties of the resource itself determine whether a resource is feasible of making the desired effects in a process.

As depicted in Figure 2, a production system that supports capability-based (continuous) engineering and operation has three major steps.

The following paragraphs summarize general descriptions and define the characteristic of capability and feasibility checking, and skill execution.

Capability Checking: In this step, the capabilities offered by resources are matched against the requirements of the process in which the resource is to be utilized for producing a certain product. To this end, the capabilities (e.g. gripping, moving, releasing) can be used as plain symbols that represent their names and relations between each other. These definitions can be complemented by adding more detailed properties. For example, checking whether a screwdriver can fasten a screw torque or angle controlled, a robot arm can perform a certain motion primitive, etc. In Industrie 3.0 systems, this step is to a large extent performed manually by the engineers based on the data sheets of the resources, and their understanding of the process. Nevertheless, this can be automated if the description of the capabilities that are offered by resources, as well as the description of the capabilities that are required by processes are available in a machine-readable/interpretable format, e.g. using ontologies. If machine-readable capability descriptions exist for Industrie 4.0 components, advanced techniques such as auto-discovery via the mDNS protocol [mDNSProtocol] can be adopted to automatically discover resources in plants based on their provided capabilities. These are the key enablers for advanced Industrie 4.0 use cases such as plug and produce.

Feasibility Checking: The objective of this step is to ensure that the necessary conditions hold so that it is feasible for the selected resource to perform its task. These conditions are determined not only by the resource but also by the process and related product as well as the context. To perform the feasibility checking, the first step is to parametrize the resource according to the requirements of the process. Afterwards, pre-conditions to perform a task must be checked. As an example consider a robot system moving a metal object from point A to point B. The pre-condition is that the metal object must already be at point A so that the robot arm can reach it. Feasibility checking provides an assurance of some confidence that the task is performed according to the requirements; these assurances can be checked as post-conditions. In our example, the post-condition is that the metal object is located at the place B. Since a process step always takes place in a duration of time, it may be needed to ensure that certain conditions hold over the entire duration of the process. These conditions are named as invariants. In our example, the invariant is that the robot must not drop the object.

To perform feasibility checking various models as well as contextual information may be needed as input. For example, the simulation models or environmental models depicting the location of objects in a production cell. Since the feasibility check is performed based on models and before real execution of skills, there is always the likelihood that unforeseen runtime conditions contradict the results of the feasibility check.

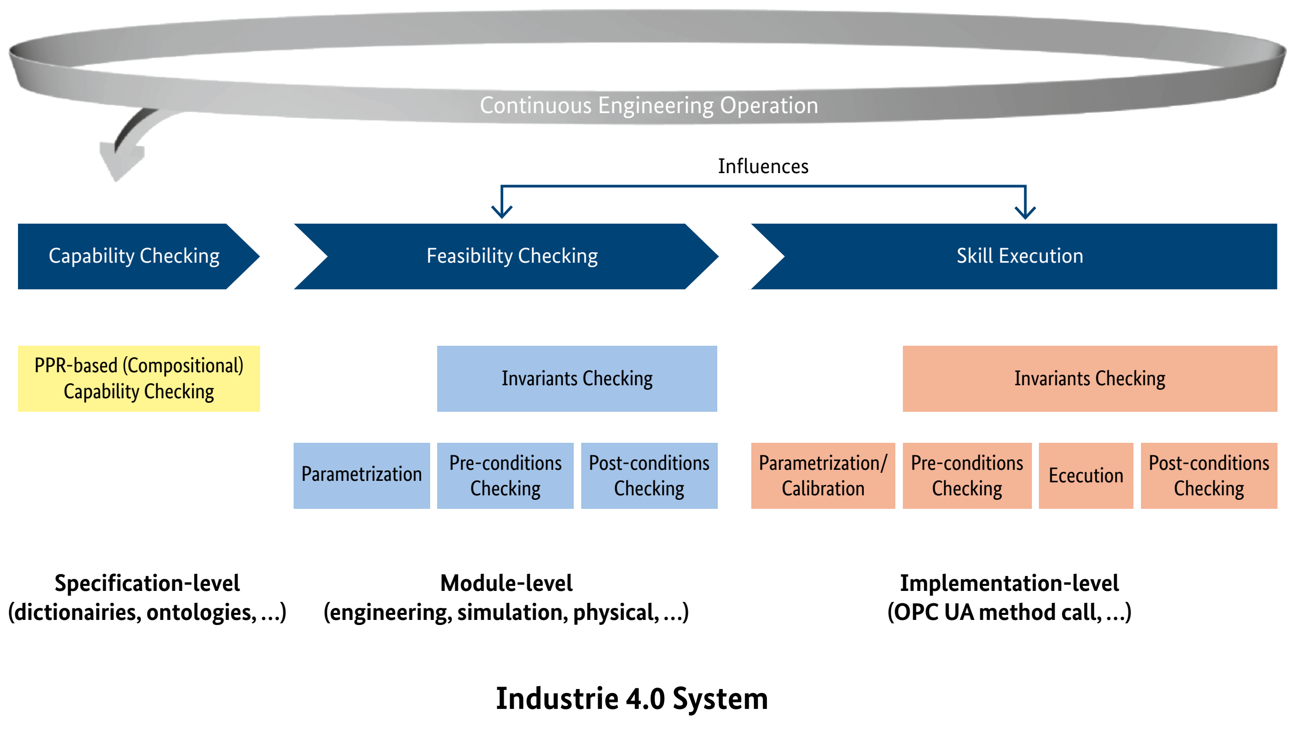
In Industrie 3.0 systems, feasibility checking is an implicit step in engineering. Engineers design a process in a deterministic way using various tools, such as production simulation tools, and ensure that the process' goal can be achieved. Nevertheless, the automation level of such checks can be improved if machine-readable descriptions of required models, processes, and resources exist. For example, formal methods, simulation and machine learning techniques can be used to assess the feasibility of designed processes.

Skill Execution: This step refers to the operational phase where resources are commissioned and put into operation. For this matter, they must be parameterized or calibrated based on the parameters specified in the Feasibility Checking step, and the operations must be invoked on the resources. At this step also pre-conditions, invariants and post-conditions can be checked to detect operational errors. For example, due to an unhandled exception the robot cannot perform its task; consequently, violating the required post-conditions. This means that the results of the Skill Execution phase also influence the Feasibility Checking results. If there is a feedback loop from the Skill Execution phase to the Feasibility Checking phase, such cases can be learnt and be considered for future feasibility checks.

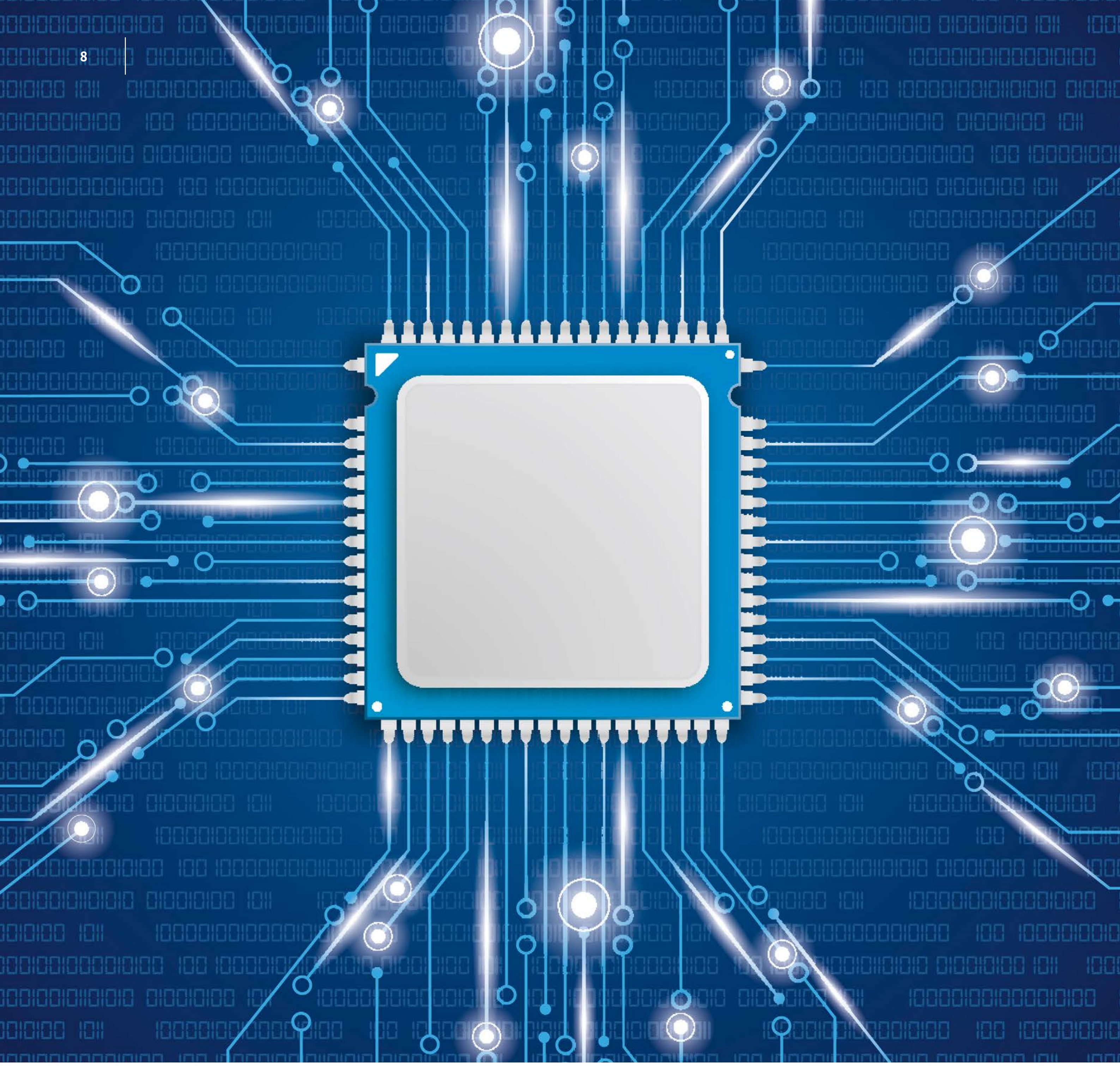
Extensive research has been performed in the field of runtime verification and recovery, where dedicated domain-specific languages and components are introduced to existing systems to perform such extra checks [RuntimeVerification]. Offering vendor-independent runtime verification and recovery techniques for Industrie 4.0 systems could be considered as a topic of further research.

In systems with continuous engineering and operation, various changes (e.g., production receipt) may be applied to the system during its operation, which also requires adjusting artifacts of prior phases such as models of the planning phase of the engineering. Such changes trigger the three steps of capability checking, feasibility checking, and skill execution.

Figure 2: Steps in Capability-based Continuous Engineering and Operation


Continuous Engineering Operation
Influences
Capability Checking
Feasibility Checking
Skill Execution
PPR-based (Compositional) Capability Checking
Invariants Checking
Parametrization
Pre-conditions Checking
Post-conditions Checking
Invariants Checking
Parametrization/ Calibration
Pre-conditions Checking
Ececution
Post-conditions Checking
Specification-level (dictionairies, ontologies, …)
Module-level (engineering, simulation, physical, …)
Implementation-level (OPC UA method call, …)
Industrie 4.0 System

Source: BaSys 4.2



4 Requirements for Capability-based (Continuous) Engineering and Operation

Different stakeholders in an Industrie 4.0 system have different requirements for capability-based engineering and operation of systems; for example:

REQ (1): As a	Plant Engineer
I would like to	engineer my plant based on capabilities of resources instead of focusing on concrete resources,
in order to	reduce the cost of adjusting the engineering projects if the concrete resource changes.
Comments	Interoperability and flexibility during the engineering phase are non-functional requirements, which can be achieved if concrete resources can be abstracted away.

REQ (2): As a	Resource Supplier
I would like to	offer the capability description of my resources, in a standardized/agreed format,
in order to	ensure that they are compatible to the engineering tools of plant engineers, and my resources are considered by plant engineers whenever my resources are suitable.
Comments	Interoperability and flexibility during the engineering phase are non-functional requirements, which can be achieved if concrete resources can be abstracted away.

REQ (3): As a	Resource Supplier
I would like to	offer the most specific capability description of my resources
in order to	ensure that the descriptions match the queries of plant engineers for resources, which can be described in a very specific or a very generic way.
Comments	Flexibility in production line is a non-functional requirement, which can be achieved by describing the capabilities of resources at higher levels of abstractions; for example, “material removal” and “making a hole” are two abstract definitions of the concrete capability „drilling“

REQ (4): As a	(re-) Planner
I would like to	find Industrie 4.0 components in the plant, which fulfill my requested capabilities
in order to	adjust/plan my production process based on the existence of certain resources.
Comments	Flexibility in production line is a non-functional requirement, which can be achieved by finding relevant resources based on their capabilities instead of specifying concrete resources.

REQ (5): As a	Maintenance Operator
I would like to	know whether two resources offer the same capabilities
In order to	be able to replace a defected resource.
Comments	Reducing maintenance and repair time to ensure continuous production is a non-functional requirement for various use cases.

REQ (6): As a	Control Component
I would like to	find resources in the plant, which fulfill my requested capabilities
in order to	adjust/plan my production process based on the existence of certain resources.
Comments	Achieving some degree of autonomy in adjusting production process is a requirement in various use cases.

REQ(7): As a	Resource
I would like to	have a machine-readable description of the capabilities and configuration parameters of other resources
In order to	be able to automatically configure them.
Comments	Achieving some degree of autonomy in adjusting production process is a requirement in various use cases.

5 Capability Description

The key to enable capability-based (continuous) engineering and operation is to have capability description in machine-readable format, and in the right level of abstraction. Ideally, the capability descriptions must be standardized and globally accessible, so that interoperability across vendors can be achieved.

5.1 The Abstraction Levels of Capability Descriptions

Capabilities can be described at various levels of abstractions, from four different dimensions [CapAbstraction]: a) "atomic" to "composed", b) "process-independent" to "process-specific", c) "product-independent" to "product-specific", and d) "resource-independent" to "resource-specific".

Atomic to Composed: It is possible to consistently separate or decompose a capability into subsidiary capabilities until they cannot be decomposed any further (in the scope of the capability model) and are presented as an "atomic" capability [CapAbstraction]. In terms of composition, it is possible to define composed capabilities out of a set of (sub-) capabilities. An example of a composed capability, which is achieved by a composition of other capabilities, would be "Pick & Place". This capability is realized by a combination of grip and move capabilities.

Process-independent to Process-specific: The process-related capability description can range from a very generic to a very specific form. For example, "Material Removal" or "Handling" are very generic descriptions of process capabilities, whereas "Making a Hole", "Gripping", "Drilling" and "Magnetic Gripping" become more concrete and determine the specific process to be used.

Resource-independent to Resource-specific: When specifying process-related capabilities, there is often also an indirect specification in terms of possible resources. This can be elaborated with the common example of the capability "Drilling", which indicates the need for a drilling machine as a resource. The correlation of the process and the resources mostly occurs if a certain level of process specification is reached. If we take the example of "Making a Hole", neither the resource nor the process is detailed, but if we go to the level of specific processes like "Drilling", the possible resources are getting more and more restricted. It is possible to conclude that the number of possible resources, which can execute a capability, correlates inversely proportional to the abstraction level of capability descriptions. While most of very specific capabilities correlate with certain resources, there are also specific processes that can be executed by a broad variety of resources. An example are capabilities, which are moving the product. This can be value-adding capabilities like assembling different product parts, as well as non-value-adding capabilities like material flow processes. It is possible in both cases to describe the movement in a very specific way, but it can be executed by a very broad variety of resources.

Product-independent to Product-specific: It would also be possible to describe capabilities based on the product they are producing. This, however, seems meaningful only on a higher level of composed capabilities because an atomic capability like "Rotating" most likely will not lead to the production of a complete product or an intermediate product. It is possible to consider a production line as a very large composed capability, which technically describes the complete production of a certain product. Again, it is possible to describe very generic capabilities, which are composed out of the most likely capabilities to manufacture a certain product category or we can specify the production for a very specific product variant. To carry it to the extreme, one could specify a generic "Manufacturing a Car" capability or the specific capability of producing a certain model of car. This heavily depends on how similar different products of a product category are manufactured. Therefore, there is no complete product-independent capability, but capabilities that are either related to a concrete product or a category of products. Likewise, this influences the process-level capabilities, which are either defined for the process of manufacturing a concrete product or a category of products.

5.2 Formalisms for Describing Capabilities

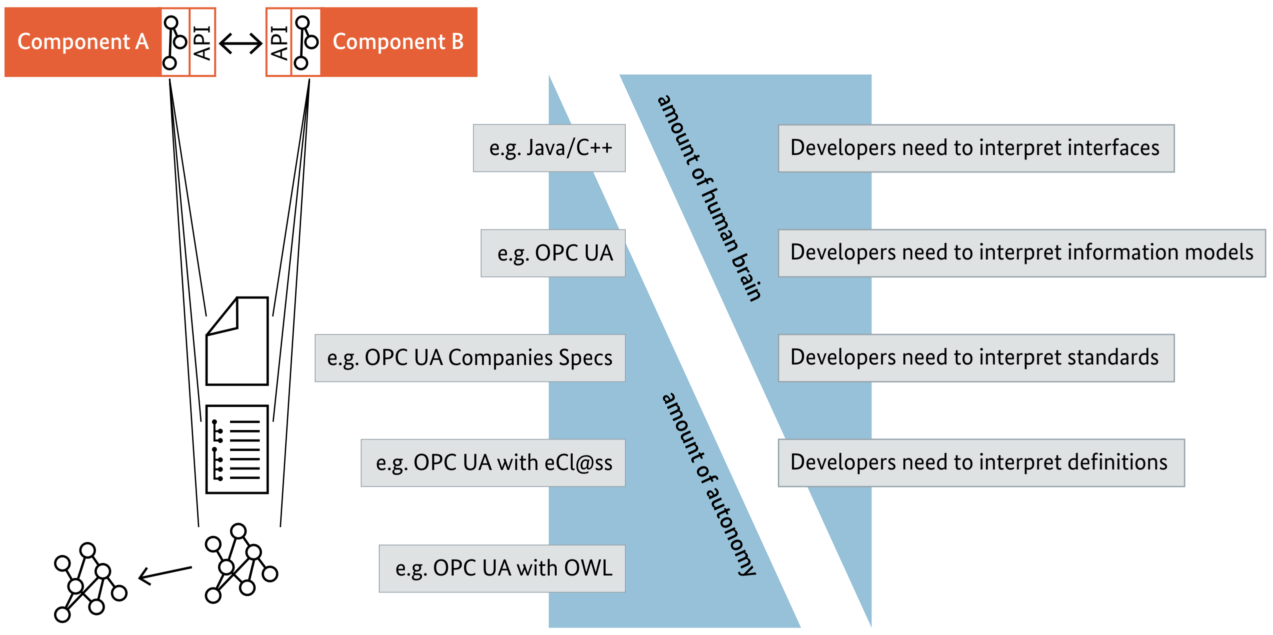
Informally, describing the semantics of a concept is describing its meaning. There can be different degrees of formality regarding this description itself, ranging from an informal (possibly colloquial) natural-language based description to a highly formal, mathematical or logical description.

Figure 3 illustrates different levels of formality for a communication between two Industry 4.0 components A and B. In order to make any communication technically possible, a component must provide an interface for data requests, operation invocations, means to send and receive messages, etc. The interface (including the data structures it accepts or provides) has a semantics that must be understood by the user of the interface in order to make meaningful requests/calls/etc.

The most informal semantic description is when the developer of component A talks to the developer of component B and explains in natural (possibly colloquial) language the semantics of the interface. In this case it is up to the developer of A to interpret the explanation of developer of B in order to use B's interface. This interpretation is manifested by the source code e.g. written in C++ or Java. A higher degree of formality is given if a commonly known and widely accepted interface is used, such as OPC UA with its powerful address space model, i.e., information model. In this case a developer no longer needs to interpret the interface, but only the information model that is provided by the interface. OPC UA defines the address space model, so a client can read and browse the address space, but the information models themselves can be arbitrary. Logical formal inference machines cannot be applied. So, there is still a large scope for interpretation, and coordination between the developers is necessary.

Using standardized information models, such as OPC UA Companion Specifications reduces this amount of coordination. Standardization defines the vocabulary terms and structures in the information models. However, developers of client components that communicate with other components that comply to such standards must still interpret the standard itself in order to make use of it properly. Interpreting the standard here means again reading a natural language document, although it is structured and written using controlled language to be as unambiguous as possible. Instead of referring to textually described stand-

Figure 3: Different Levels of Formality for Communication between two Industry 4.0 Components


Component A
API
API
Component B
e.g. Java/C++
e.g. OPC UA
e.g. OPC UA Companies Specs
e.g. OPC UA with eCl@ss
e.g. OPC UA with OWL
amount of human brain
amount of autonomy
Developers need to interpret interfaces
Developers need to interpret information models
Developers need to interpret standards
Developers need to interpret definitions

Source: Plattform Industrie 4.0

ards, the information model could define its semantics by referring to a standardized vocabulary, such as eCl@ss [eCl@ss] (via "HasDictionaryEntry" reference in OPC UA).

There, a formal concept hierarchy, as well as property lists are provided, and the vocabulary is standardized. However, interrelations between concepts are not formally described, descriptions of concepts are still described in natural language, and modelling flexibility and expressiveness is limited. Instead of references to catalogue properties (such as IEC CDD or eCl@ss) with nearly no relationship between each other and with limited formalisms, expressive formal ontologies such as the Web Ontology Language (OWL) [OWL 2] could be used to define the semantics of any concepts. The semantics is then well-defined, such that algorithms exist which allow for autonomous interpretation and decision-making.

An OWL ontology is a set of axioms, where each axiom denotes a logical relationship between some entities that represent the domain of interest. The ontology thus defines a formal semantics for the entities. In OWL, entities can be classes, properties, and individuals. The logical formalism used in OWL is a Description Logic, which defines the entailment regime that is used for inferring implicit knowledge from explicitly stated axioms. The knowledge modeled in an ontology is defined, formal and explicit, and thus inferences are deterministic and provable, which is an advantage over vagueness of natural language and uncertainty as in statistical models (for instance as provided by machine learning approaches).

In addition to its logical underpinning, OWL is based on the Resource Description Framework (RDF) in the sense that all OWL entities are valid RDF Resources and identifiers are Internationalized Resource Identifiers (IRI). This allows for reusing and incorporating existing ontologies, vocabularies, or other resources from the Linked Open Data Web.

Ontologies can be provided in a modular way, where different modules provide different parts of the knowledge about a domain of interest. If such modules are merged, the resulting ontology consists of the union of the sets of axioms from the various modules. If axioms from different modules refer to the same entities, the axiom sets complement each other.

To enable capability-based (continuous) engineering and operation, the formal semantic description must be used to describe capabilities, so that a client can query a component for its capabilities via a standardized meta-model (e.g. Asset Administration Shell [AASiD Part 1]). The capability description itself is semantically defined by pointing to an ontology that has formal definitions of the capability, its relation to other capabilities (subsumption, composition, etc.) as well as possibly other connected information.

In the context of capability description, an OWL-based model provides the following possibilities:

1. Define or describe existing vocabulary for capabilities in terms of entities with unique IDs (IRIs), but also define synonyms in terms of different labels for the same entity or equivalence axioms

2. Define hierarchies of capabilities

3. Express via ontology axioms how capabilities might be composed of different other capabilities

4. Extend the knowledge about capabilities by multiple ontology modules that cover different aspects of capabilities, for instance, a core module defining the capability entities and hierarchy, and a second module defining properties of capabilities

5.3 Capability and Ontologies

Capabilities are generic and abstract concepts that should be defined independent from any specific asset and hence independent from the Asset Administration Shell. Such an asset independent definition motivates the provision of a shared knowledge base of capabilities that can be referred to any individual Industrie 4.0 component, most notable from within the Asset Administration Shell, or from any Industrie 4.0 infrastructure element, such as registries, search services, capability checkers, etc.

An ontology, as per definition a formal, explicit specification of a shared conceptualization, constitutes a suitable way of representing such a generic capability knowledge base. Depending on the ontology language, a semantically expressive formalism for describing capabilities and relations between them can be utilized. For instance, the Web Ontology Language (OWL) provides an axiomatic description of ontology entities (here: capabilities) based on Description Logics. This formal underpinning allows for powerful reasoning on capabilities, such as capability matching based on capability hierarchies or capability composition.

As for hierarchical capability models, let C' be a sub-capability of C. A capability requester asking for C can be matched with a capability provider offering capability C' since the semantics of "sub-capability" ensures that C' also entails C. In an example, let "MoveFlangeTo6DPositionOnLinearTrajectory" be a sub-capability of "MoveFlangeTo6DPositionOnPath", which itself is a sub-capability of "MoveFlangeTo6DPosition". A requester asking for a component providing "MoveFlangeTo6DPosition" can be offered a component providing "MoveFlangeTo6DPositionOnLinearTrajectory" since this component implicitly provides the requested capability.

As for capability composition, let C, D and E be capabilities, where C is composed of D and E. Semantically this means that a component providing both D and E, also provides C. A capability requester asking for C can be matched with a capability provider offering both D and E. This component could, e.g., be a compound component built from two other components that provide capabilities D and E, respectively. For the compound component it can be inferred that it provides capability C, given that the two subcomponents are assembled to the compound component in a way that the capabilities of the subcomponents directly transfer to the compound component. In an example, let "Order" be a capability that can be composed of the capabilities "Orient" and "Position". A requester asking for a component providing "Order" can be offered a component providing both "Orient" and "Position", since this component implicitly provides the requested capability based on the general and shared knowledge about those capabilities.

Hierarchy and composition are just two examples of formal and shared knowledge about capabilities likely to be useful in practical scenarios. However, an ontology model could be extended by additional knowledge about capabilities depending on the expressiveness of the ontology language.

It is an important requirement for the ontology language to provide globally unique identifiers for the entities it declares and defines. This is necessary for defined capabilities such that they can be uniquely referred to from within the Asset Administration Shell, etc.

The proposed separation of a shared capability ontology model from the Industry 4.0 elements that refer to these models bears the expectation that the capability ontology model is universally valid, comprehensive (in terms of covering all possible domains) and complete (in terms of defining capabilities hierarchies and composition in the most fine-grained way possible). It is obvious that such an expectation is unrealistic to meet for a single universal capability ontology. It is hence required that the shared capability ontology follows a modular design. This can be realized by the following design directives (for more details please refer to [C4I Ontology] and [ETFA 2020]):

1. The capability ontology is composed of various ontology modules

2. Any ontology module can extend one or more other ontology modules in the way that it adds knowledge to the module(s) it extends

3. There must be a universal capability meta-model describing how capabilities are described, which must be followed by all ontology modules

These modularization directives allow for a distributed landscape of capability ontology modules (directive 1) that are compatible with each other (directive 3) and that facilitate domain and/or vendor specific extensions of the generic capability model (directive 2). Note that modules (e.g. vendor specific ones) extending other modules do not necessarily have to be shared and thus be publicly available in case they would unveil corporate secrets. However, it is highly discouraged to make use of this option since effective communication about capabilities in capability-based use cases (see Requirements for Capability-based (Continuous) Engineering and Operation 0) is only possible if terms and definitions are openly accessible.

The Web Ontology Language (OWL) has several features that make it a suitable candidate to represent capability ontologies:

1. The ontology language itself is a widely accepted and well-defined W3C standard

2. A wide range of tools exists, both open-source and commercial, particularly OWL reasoners

3. IRIs are used as globally unique identifiers, which is the established ID standard in the Web

4. IRI namespaces can be used to distinguish between generic, domain or vendor specific ontology modules

5. There is a formal logical underpinning (Description Logics) to ensure decidability (in terms of computational theory) and explainable deductions

6. Monotonicity of the underlying logic ensures that later extensions (as typically done by extending modules) will not invalidate previous inferences

7. Modularity can natively be realized by the language construct of ontology imports

5.4 Skill – a Capability Implementation

While capabilities are abstract implementation-independent descriptions of the asset application functions, skills offer the detailed implementation dependent description. The capability and the skill differ in the level of details. While the Capability describes the application function abstract without relations to a concrete asset the skill describes the application function related to the asset that provides it. Different skill realizations can be mapped to one capability.

The following Table 1 shows a comparison of different alternatives for skill implementation and Figure 4 accompanies the table by some examples:

Figure 4: Realization variants of skills

State Variable	Trigger Variale	Operation	Function Block (FB)	Semantic Protocol

If (State Variable == xzy)
Then (Var_o = …., …)
Supervision of application parameter
	Trigger Variable starts a function
	$Y = f(Var_i)$
		Onetime invocation of the function
		OutVar_1, …OutVar_j = Operation (InVar_1, InVar_2 …, InternPar_k)
			FB FB_NAME {
			Input: InVar_1, InVar_2, …, InVar_i
			Output: OutVar_1, OutVar_2, …, OutVar_j
			OutVar_j = f (InVar_1, InVar_2 … InternPar_k)
			}
				Decision algorithms e.g. for offer definition or offer selection

Examples : Scaling Unit conversion	**Examples :** Factory_Reset Operation Mode change	**Examples :** Calibration	**Examples :** PLC FB for Robot control	**Examples :** Bidding

Source: Plattform Industrie 4.0

Table 1: Advantages and Disadvantages of Different Types of Skill Implementation

Skill Implementation	Advantages	Disadvantages
State Variable	Lightweight solution for simple (sensor-like) resources	No candidate for a standardized way of realizing skills, since more complex assets require different solutions
Trigger Variable	Analogous to legacy architectures (e.g. fieldbus)	Unclear semantics (trigger by rising/falling edge, etc.) Additional variables for input/output parameters are required (where their association to the trigger variable is unspecified) Legacy design where more elegant ways are available (operations)
Operation	Lightweight solution for simple resources with reaction times less then the real time requirement of the calling entity (stateless, synchronous call)	Unsuited for longer running skills since no directly associated state monitoring is available
Function Block	More generic and powerful representation of any skill Long-running skills supported including state-machine to control and inform about its execution state, and the ability to stop/interrupt (compare OPC UA Programs [OPC UA Programs]) Potentially a standard way of representing skills (independent from what the skill is doing, it can be parameterized and executed in the same way as long as the structure of the FUB including its operations is standardized)	Complex representation alternative for simple skills, such as a simple sensor
Semantic protocol	High degree of autonomous of application Consideration of different application states and third party interactions	Stateful interface Management of the interaction state machines

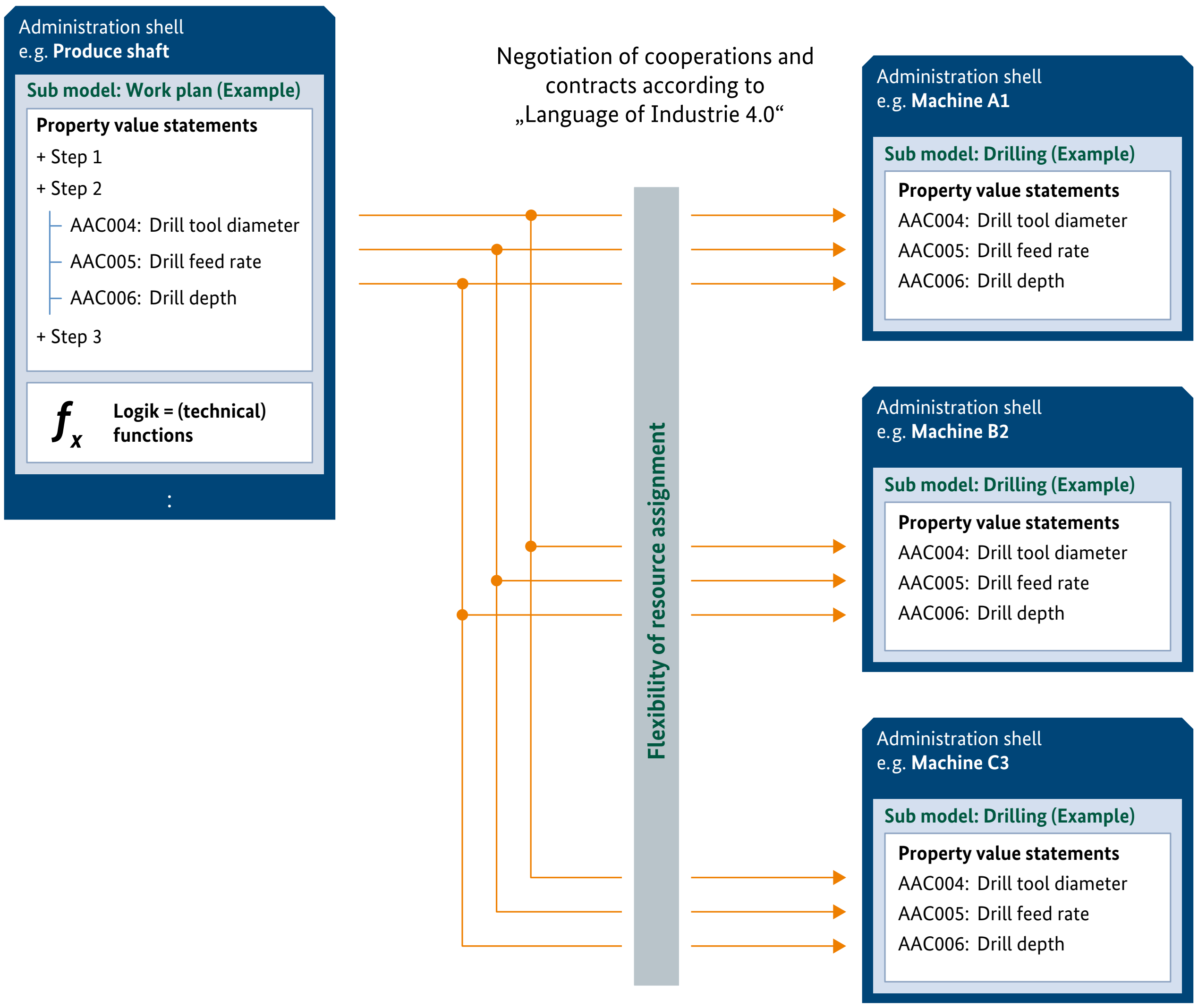
6 Capability Description and Checking for Industrie 4.0 Components

6.1 Introduction

In the world of Industrie 4.0, each asset is given an Asset Administration Shell (AAS). The asset and the AAS together built the I4.0 component. The AAS consists of a number of submodels in which all the information and functionalities of a given asset needed to realize a dedicated set of use cases– including its features, characteristics, properties, status, parameters, measurement data and capabilities – are described. It allows for the use of different communication channels and applications and serves as the link between I4.0 components and the connected, digital and distributed world.

- The Asset Administration Shell
- can be used for non-intelligent and intelligent assets
- covers the complete lifecycle of products, devices, machines and facilities
- allows for integrated value chains
- serves as the digital basis for the development of autonomous systems and AI

Figure 5: Resource Flexibility through Use of Common Properties [Composites]


Administration shell
e.g. Produce shaft
Sub model: Work plan (Example)
Property value statements
+ Step 1
+ Step 2
AAC004: Drill tool diameter
AAC005: Drill feed rate
AAC006: Drill depth
+ Step 3
Logik = (technical) functions
Negotiation of cooperations and contracts according to „Language of Industrie 4.0“
Flexibility of resource assignment
Administration shell
e.g. Machine A1
Sub model: Drilling (Example)
Property value statements
AAC004: Drill tool diameter
AAC005: Drill feed rate
AAC006: Drill depth
Administration shell
e.g. Machine B2
Sub model: Drilling (Example)
Property value statements
AAC004: Drill tool diameter
AAC005: Drill feed rate
AAC006: Drill depth
Administration shell
e.g. Machine C3
Sub model: Drilling (Example)
Property value statements
AAC004: Drill tool diameter
AAC005: Drill feed rate
AAC006: Drill depth

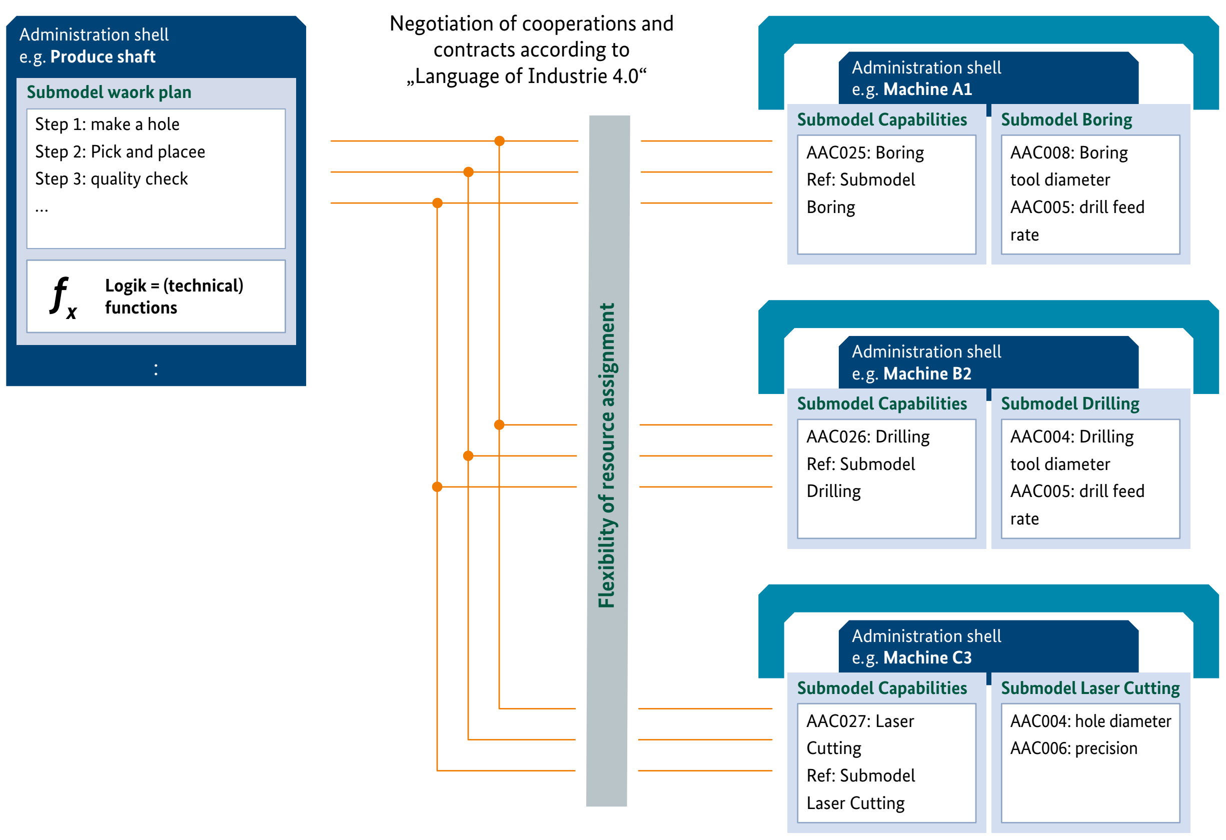
Source: Plattform Industrie 4.0

In [Composites] first use cases and requirements with respect to negotiations of co-operations and contracts were described, see for example Figure 5.

However, the notion of capability was not yet introduced: all concepts were described through property mapping only. This document now offers a systematic generalized and extended approach for handling the required capabilities and the skills the I4.0 components can offer (Figure 6).

Therefore in [AASiD Part 1], Version 2.0, Capabilities were introduced for the first time. The next chapters describe how to use these capabilities and how they are related to other elements of the Asset Administration Shell.

Figure 6: Assignment of a Work Plan on Production Process [Composites]



Source: Plattform Industrie 4.0

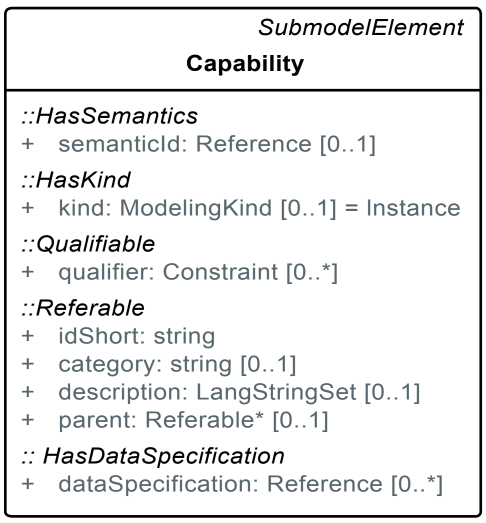
6.2 Capability Element as Description in Asset Administration Shell

For capability handling the submodel element “Capability” is foreseen (see Figure 7). Like any other submodel element in the Asset Administration Shell it is referable, i.e. it has an idShort that is unique within its namespace (e.g. a submodel or a collection) and it has a reference to its semantic definition (semanticId).

In Figure 8 the submodel elements being the most relevant when modeling capability-skill relationships are shown:

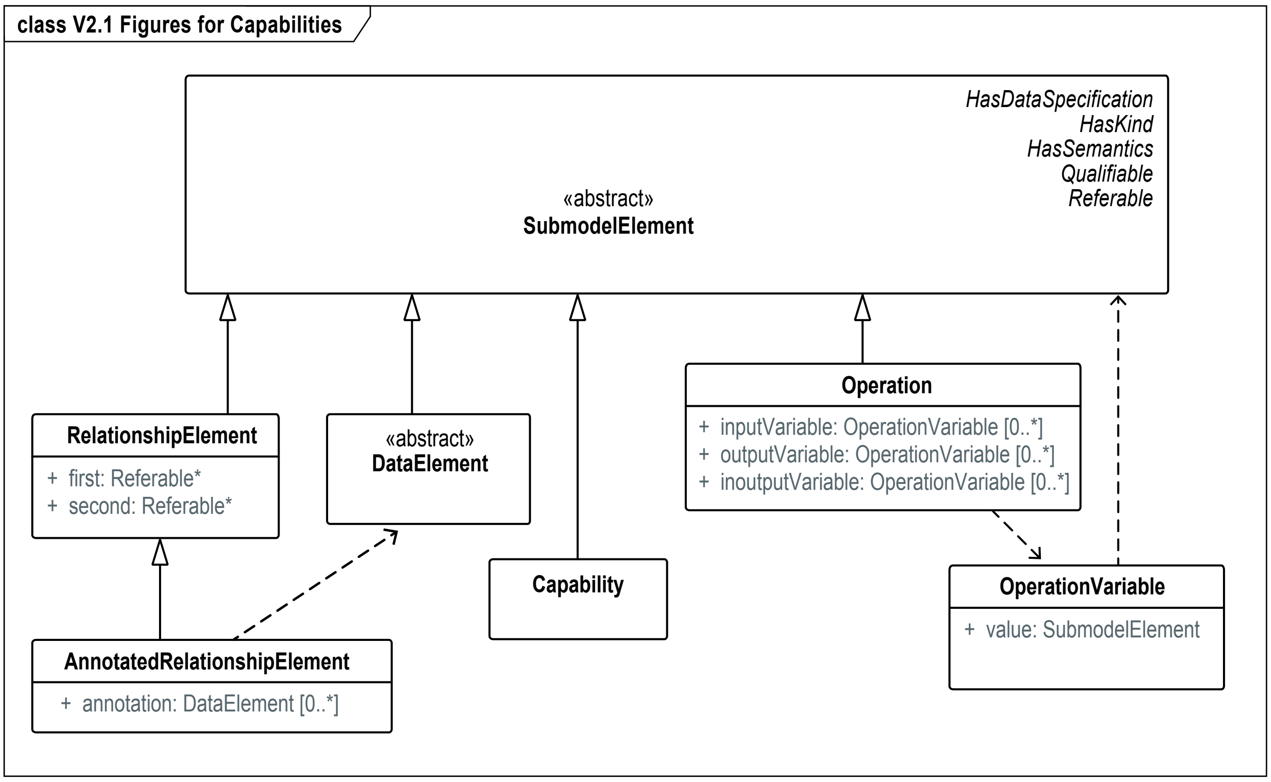
- The capability itself,
- the relationship element used for describing the relationship “CapabilityRealizedBy” between capability and skill,
- data elements for skills realized as properties, and
- operations for skills realized as executable methods

Figure 7: Capability with Inherited Attributes


SubmodelElement
Capability
::HasSemantics
+ semanticId: Reference [0..1]
::HasKind
+ kind: ModelingKind [0..1] = Instance
::Qualifiable
+ qualifier: Constraint [0..*]
::Referable
+ idShort: string
+ category: string [0..1]
+ description: LangStringSet [0..1]
+ parent: Referable* [0..1]
:: HasDataSpecification
+ dataSpecification: Reference [0..*]

Source: Plattform Industrie 4.0

Figure 8: Submodel Elements in the Asset Administration Shell including Capability



Source: Plattform Industrie 4.0

The semanticId can be an IRI, an IRDI or a custom global identifier. In the context of capabilities the semanticId has a reference to an ontology that does not only define the semantics of this single capability but also gives additional semantic information, for example inheritance relationships.

Another important feature of a submodel element and the submodel is the feature to be able to annotate it with qualifiers. A standardized qualifier for example is the life cycle qualifier as defined in IEC 62569-1 and introduced to IEC CDD (see Figure 9).

Figure 9: Qualifier Life Cycle

Value List	0112/2///61360_ 4#AAF573 - as built -CON	BUILT
	0112/2///61360_ 4#AAF576 - as inquired	INQ
	0112/2///61360_ 4#AAF577 - as offered	OFF
	0112/2///61360_ 4#AAF578 - as operated	OP
	0112/2///61360_ 4#AAF579 - as specified	SPEC
	0112/2///61360_ 4#AAF580 - as supplied	SUP
	0112/2///61360_ 4#AAF682 - as decommissioned	DECOM

Source: eCl@ss

Table 2: Usage of Property Values Statements for capabilities and skills

Abbreviation	Qualifier Value	Definition in DIN SPEC 92000:2019	Mapping to Capabilities-Skills
R	requirement	Requirement that the property value must be set in the named predicate relation to the predicate value.	Required Capability Requirement that the capability must be offered by the asset.
A	assurance	Assurance that the property value is set in the named predicate relation to the predicate value. The choice of the requested value does not affect the permissible value ranges of other properties.	Assured Capability Capability that is assured to be available by offering corresponding skills. Note: Prerequisite is that the Feasability Check passed. Note: An assured capability is also an offered capability.
O	offer	Assurance that the property value is set in the named predicate relation to the predicate value. However, the choice of the requested value affects the permissible value ranges of other properties.	Offered Capability Capability that is offered but not yet assured. Assurance depends on other required capabilities or execution context. Note: Feasability Check not yet done, i.e. the capability cannot yet be assured.

A new type of qualifier, the property value statement, is specified in DIN SPEC 92000. Examples can be found in [DIN SPEC 92000]. In Table 2 we show how to use the qualifiers in the context of capability and skill mapping[2].

6.3 Predefined Semantics for Relationship "Capability Realized By"

For being able to map capabilities to skills there needs to be a relationship with this clearly defined semantics. Since no such relationship is yet standardized in eCl@ss or IEC CDD a corresponding Industrie 4.0 relationship for Asset Administration Shells is defined.

The concept description for this predefined relationship for capability handling in the Asset Administration Shell is defined in Table 3[3]. For details of a concept description and the mandatory fields for concept descriptions for relationships see [AASiD Part 1]:

Table 3: Predefined Concept Description for Relationship "Capability Realized By"

Concept	Description
Attribute	Value
identification.idType	IRI
identification.id	https://admin-shell.io/aas/conceptDescriptions/CapabilityRealizedBy/1/0
category	RELATIONSHIP
shortName	CapabilityRealizedBy
preferred Name:	(en) Capability realized by (de) Fähigkeit realisiert durch
definition	(en) This is a directed relationship between a capability and the skill that is realizing the capability by providing a corresponding implementation. (de) Dies ist eine gerichtete Beziehung zwischen einer Fähigkeit und der Fertigkeit, die diese Fähigkeit realisiert, indem sie eine entsprechende Implementierung bereitstellt.

2 The additional qualifier values „actual Value" and „statement" are ignored in this document.

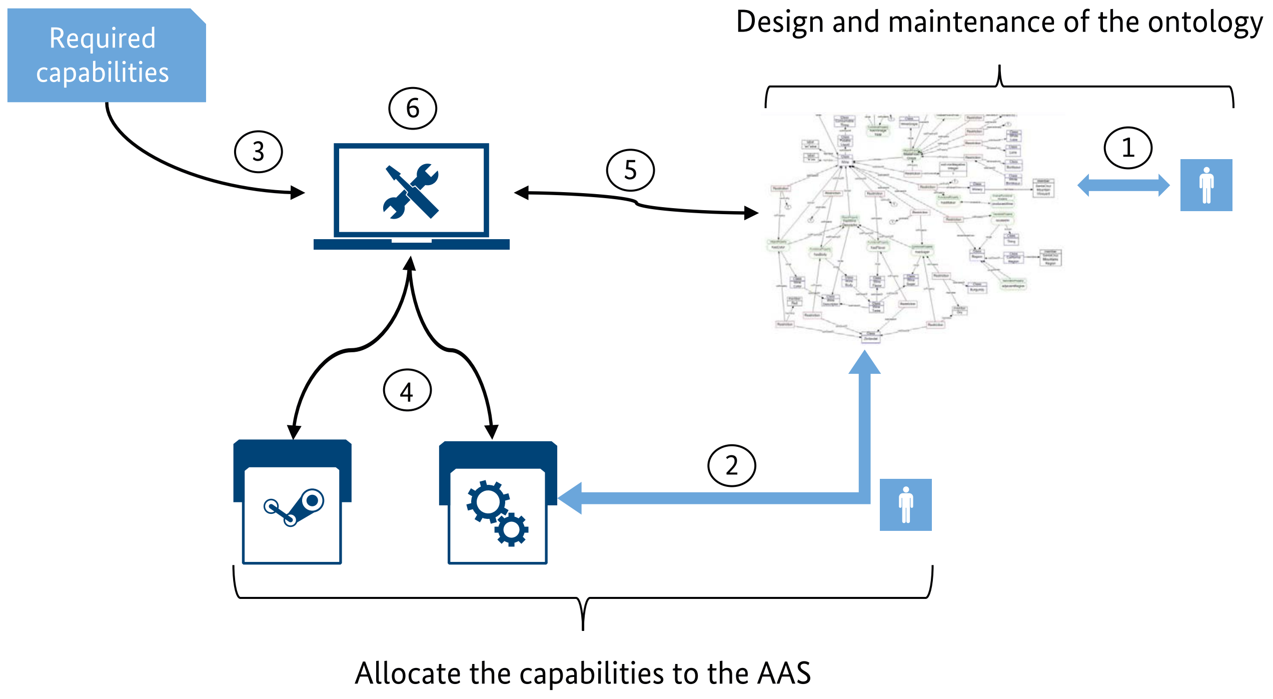
3 The identification.id is subject to change since the domain admin-shell.io is just set up and the approval of the coordination board is pending.

6.4 Capability maintenance and checking based on Asset Administration Shell

The capability check can be divided in three different aspects (see Figure 10). The development and maintenance of one or multiple connected capability ontologies (see 1 in Figure 10), the configuration of the AAS capability elements via adopting semanticIds to refer to ontologies (see 2 in Figure 10) and the process of checking asset capabilities (see 3 to 6 in Figure 10). To develop and maintain capability ontologies, an authorized organization or a standard must provide a capability description, for example as it is provided by metal processing [Heh11]. The classification must then be transformed into an (OWL) ontology, and if there are multiple ontologies, they must be combined.

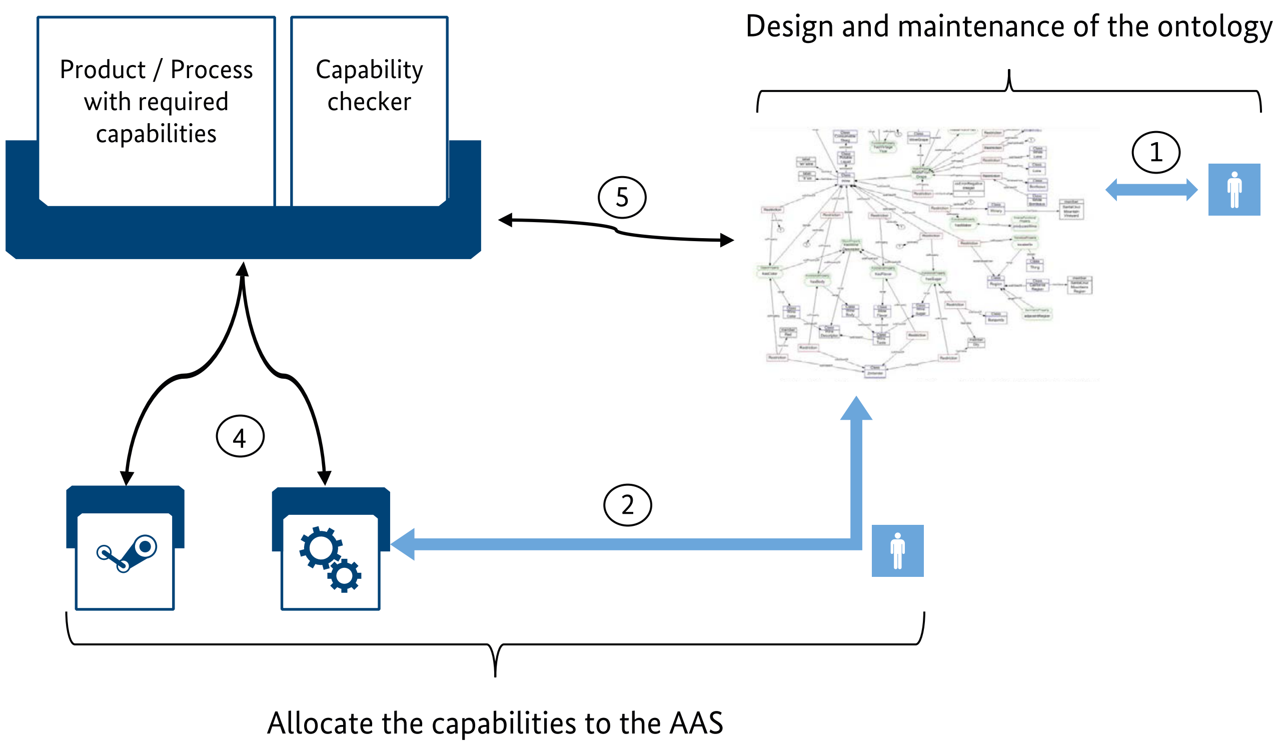
The capability ontologies can then be referenced in the capability element of the relevant Asset Administration Shells to be used in the capability checking process. Various alternatives can be adopted to implement the capability checking functionality. For example, as depicted in Figure 10 Capability Checking by an Engineering Tool, the checking can be performed by an engineering tool (see 6), which receives (see 3) the specification of required capabilities by a process step, and the specification of capabilities of available resources from their AAS (see 4). The tool searches in the capability ontologies (see 5) to match required capabilities against the offered capabilities and decides on which resource should be adopted, if any. It is also possible, that the engineering tool uses an external capability checker and initiates the checking process and takes the result only.

Figure 10: Capability Checking by an Engineering Tool


Required capabilities
Design and maintenance of the ontology
6
3
5
1
4
2
Allocate the capabilities to the AAS

Source: Plattform Industrie 4.0

Figure 11: Capability Check within an Asset Administration Shell (AAS)


Product / Process with required capabilities
Capability checker
Design and maintenance of the ontology
1
5
4
2
Allocate the capabilities to the AAS

Source: Plattform Industrie 4.0

Another example alternative is that the capability checking is provided by AAS of resources or processes (see 3 in Figure 11). In this cases a resource's AAS can be asked (see 4) whether it fulfills certain capability required by a process (see 3). This interaction can take place between the AAS of resource and process, as depicted in Figure 11.

6.5 Skill Modelling for Industrie 4.0 Components

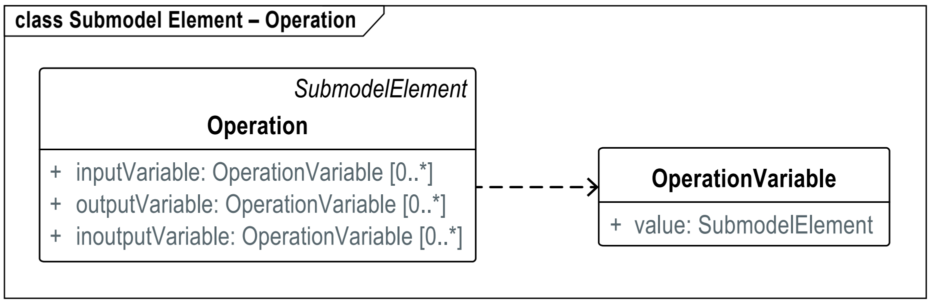
In Chapter 1.5 different alternatives of how capabilities can be realized by skills were described. In the following Table 4 it is shown how the corresponding skills are realized within the AAS:

Table 4: Skill Implementation in the Asset Administration Shell

Skill implementation	AAS Skill Mapping	Comment	Example
State Variable	Property, typically with category VARIABLE	Properties with category VARIABLE represent calculated or measured data (pull)	Simple resources, such as simple sensor devices might provide their sensed data in a data variable. For instance, a temperature sensor might provide its measurement result in a variable “Temperature”. The capability “measureTemperature” could be implemented by the data variable “Temperature”.
Trigger Variable	Event	Events that observe a data variable/property may trigger (push) other events or the execution of an operation or the change of values of properties etc.	The capability might be to “Detect and notify if temperature higher than a given threshold”. This Skill might be realized by a submodel element of type “Event” that overserves the property “Temperature”.
Trigger Variable	Property, typically with category VARIABLE	Variable with a certain value triggering a function invocation.	An Example would be to have a property “StartProgram” and depending on its value the execution of a function is triggered. This kind of mapping of a trigger variable is suitable in systems that do not support events or in which the usage of event is not recommended for other reasons: the event is mimicked by a data variable[4].
Operation	Operation	Operations are used for client/server communication	Submodel elements of type “operation” are a natural way of representing skills that can be invoked by a simple method call. Operations provide input and output variables and variables serving as input and output variables. An operation “Open” could be the skill implementing the capability “OpenPinchGripper” with an input parameter “width”.
Function Block	Submodel	A submodel may realize a complex function block. The application(s) accessing properties or operations etc. of a submodel need deep know-how of the function. Typically, a function block depends on other functions block, i.e. a function block needs to access properties of other functions that serve as input for its own functionality. Note: Submodels in AAS do not distinguish between input and output data of a function. A Function model (or a subfunction model) is typically modelled using another tool and format, for example it may be specified as a MATLAB function block. The AAS is just declaring what can be directly used by other applications but is not specifying the functionality itself. The model specification file can be added to the submodel by a File/Blob submodel element.	The AAS meta model does not provide a submodel element “function block” because a function block can be seen as an own submodel. Depending on the structure of the function block, the submodel contains data elements for input and output parameters, operations for start, stop, interrupt, etc., as well as optionally data elements representing internal states and state transitions of complex skills. For instance, a capability “Linear6DMotion” could refer to a submodel “MoveLin” which represents a function block to execute a linear motion of an articulated robot.

4 This is a typical realization in traditional fieldbus devices: an operation of a resource is triggered by a variable (e.g. a boolean set to true). Additionally, other variables could be set to provide input parameters, and others could be used to retrieve output parameters. Even though this way of modelling is deprecated, the AAS meta model allows for mimicking field device communication in this way.

Figure 12: Operations in the Asset Administration Shell


class Submodel Element – Operation
SubmodelElement
Operation
+ inputVariable: OperationVariable [0..*]
+ outputVariable: OperationVariable [0..*]
+ inoutputVariable: OperationVariable [0..*]
OperationVariable
+ value: SubmodelElement

Source: Plattform Industrie 4.0

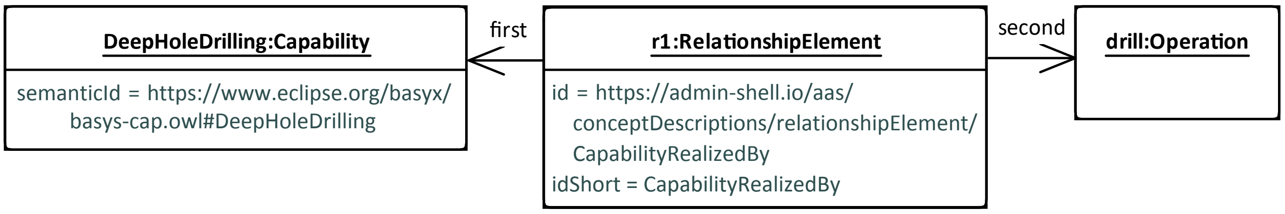
To be able to specify the implementation of a capability a predefined relationship element is introduced (see chapter "Predefined Semantics for Realtionship "CapabilityRealizedBy"). Its short name is "CapabilityRealizedBy".

The AAS is using the relationship element of specifying how a capability is implemented. There can be more than one capability implementation, i.e. if there is more than one relationship element with semanticId "CapabilityRealizedBy" for the same capability then the semantics of this set of relationships is "alternative implementations of a capability". Of course, the same operations or properties can also be used to implement different capabilities (see Figure 13 Skill '"drill" for Capability "DeepHoleDrilling").

Figure 14 shows the example of Figure 13 modelled with the AASX Package Explorer[5].

For common applicability of a capability submodel in Industrie 4.0 it is recommended to not only predefine the relationship but also to give best practices or even standardize a capability submodel with a clear semantics.

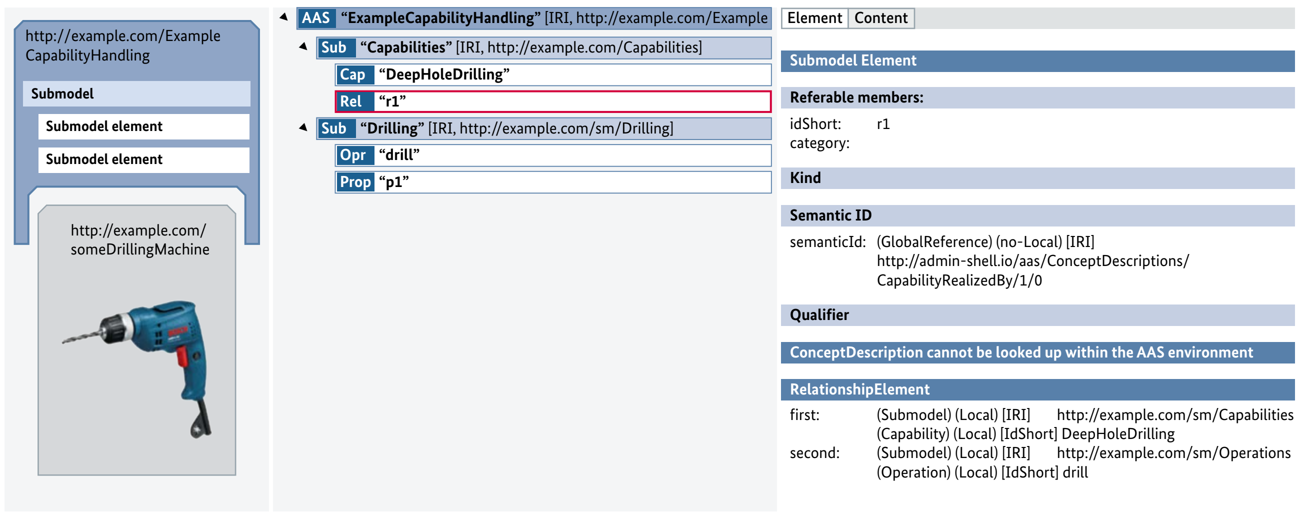
Figure 13 :Skill '"drill" for Capability "DeepHoleDrilling"


DeepHoleDrilling:Capability
semanticId = https://www.eclipse.org/basyx/basys-cap.owl#DeepHoleDrilling
first
r1:RelationshipElement
id = https://admin-shell.io/aas/conceptDescriptions/relationshipElement/CapabilityRealizedBy
idShort = CapabilityRealizedBy
second
drill:Operation

Source: Plattform Industrie 4.0

5 The AASX Package Explorer is an open source viewer and editor for Asset Administration Shells. https://github.com/admin-shell/aasx-package-explorer

Figure 14: Example for Capability Modeling with the AASX Package Explorer


http://example.com/Example CapabilityHandling
Submodel
Submodel element
Submodel element
http://example.com/ someDrillingMachine

AAS "ExampleCapabilityHandling" [IRI, http://example.com/Example
Sub "Capabilities" [IRI, http://example.com/Capabilities]
Cap "DeepHoleDrilling"
Rel "r1"
Sub "Drilling" [IRI, http://example.com/sm/Drilling]
Opr "drill"
Prop "p1"

Element | Content

Submodel Element

Referable members:

idShort:	r1
category:	

Kind

Semantic ID

semanticId:	(GlobalReference) (no-Local) [IRI] http://admin-shell.io/aas/ConceptDescriptions/CapabilityRealizedBy/1/0

Qualifier

ConceptDescription cannot be looked up within the AAS environment

RelationshipElement

first:	(Submodel) (Local) [IRI] http://example.com/sm/Capabilities (Capability) (Local) [IdShort] DeepHoleDrilling
second:	(Submodel) (Local) [IRI] http://example.com/sm/Operations (Operation) (Local) [IdShort] drill

Source: Plattform Industrie 4.0

Usage Examples of Capabilities in Industrie 4.0 Systems

In this section, we provide several examples of Industrie 4.0 systems, in which capability description and checking is used to enable (continuous) engineering and operation. In addition, multiple example capability descriptions in AAS for various resources are elaborated.

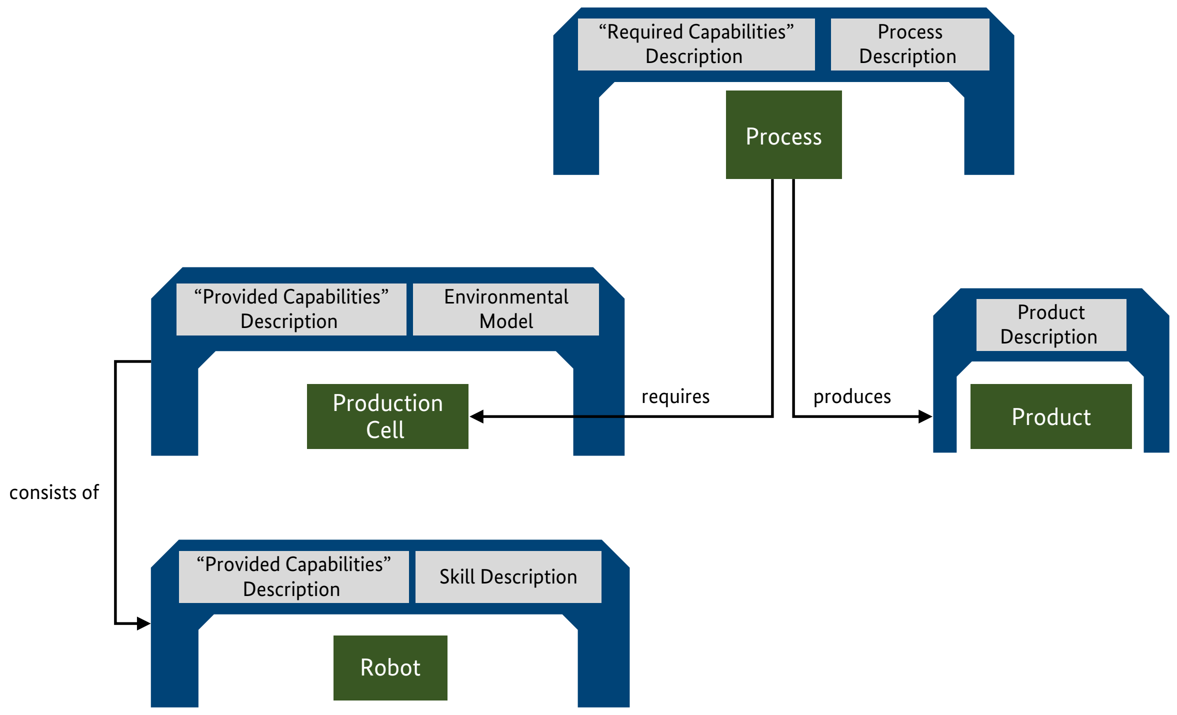
Capability and Feasibility Check for a Pick and Place Production Cell

Consider for example a process step where a robot must be utilized to move a metal object with a certain weight from position A to position B. Figure 15 illustrates the set of AAS that exist for this example. Here, we have two resources: a robot system and a production cell. The robot system can be decomposed further to smaller resources, particularly a robot and a gripper; the robot itself could be decomposed to joints, motors, etc. This decision depends on the use case and the level of granularity that we want to achieve. The production cell is a composite resource, which among others contains the robot system. The AAS of the robot system describes the capabilities that are offered by the robot system and the relevant properties. The AAS of the production cell also describes the capabilities that are offered by the production cell. In this case, we can think of describing capabilities at different levels of abstractions for the robot system and the production cell. For example, the robot system is capable of "grasping", "moving" and "releasing" objects, whereas the capability of the production cell is described at a higher level as "pick and place" capability. The AAS of the production cell also contains other submodels, for example, the environmental model that describes where the robot system and product is placed.

The AAS of a product provides the description of the product; dedicated submodels can be adopted for this matter. The AAS of the process contains the description of the process and its required capabilities.

The actual capability and feasibility checks can be performed by external applications, e.g. engineering tools (see Figure 10). For this matter, the application needs to access the available AAS's, and check the offered capabilities against the required capabilities, for example, via OWL inference. Alternatively, the checks can be performed by the relevant AAS's (see Figure 11: Capability Check within an Asset Administration Shell (AAS)). For example, the AAS of a resource has sufficient information to check whether the resource offers a certain capability, and can respond to the queries issued by the AAS of the process. The feasibility check can be performed by an external software such as MES using the available models within AAS's or external models.

Figure 15: Asset Administration Shells for a Pick and Place Production Cell



Source: Plattform Industrie 4.0

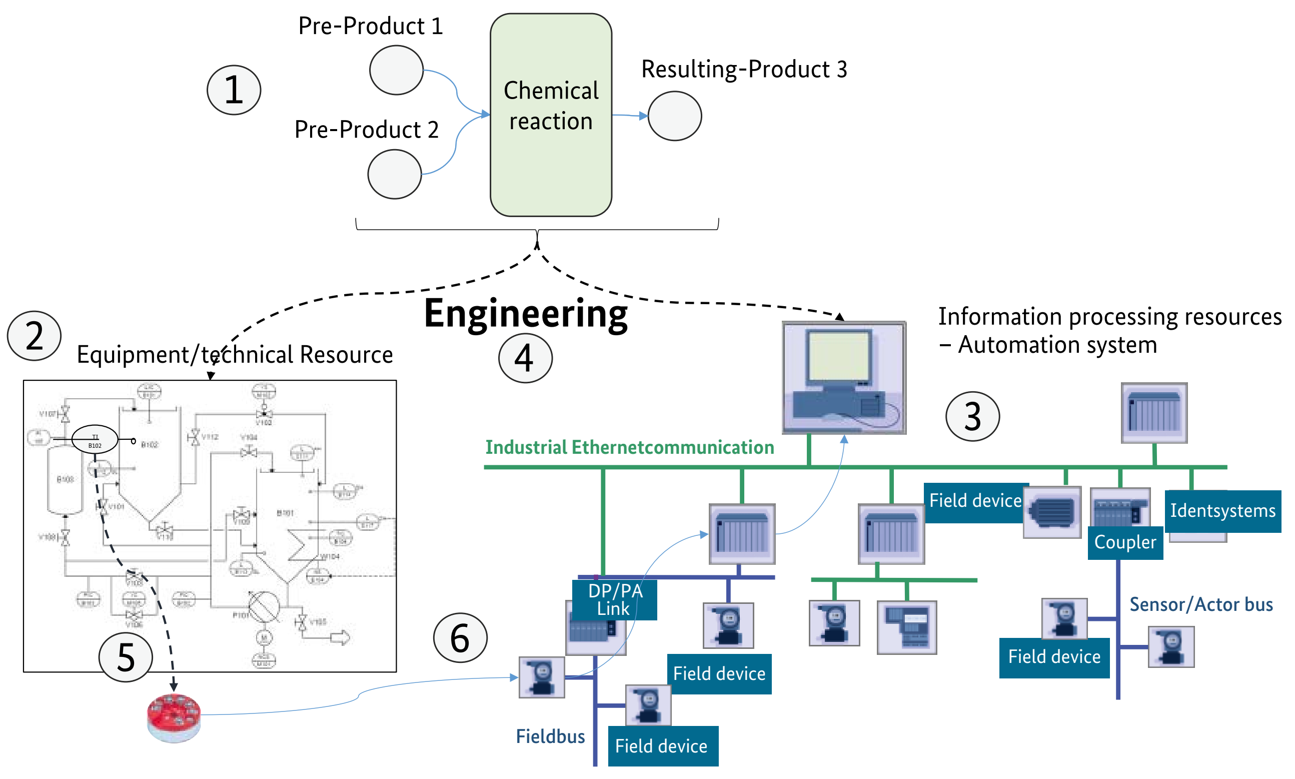
Capability and Feasibility Check during System Engineering of a Chemical Plant

The engineering of a chemical plant is one example described in this chapter. The abstract chemical reaction is described at 1 in Figure 16, and in which the Engineering team must coordinate the plant design, perhaps captured in a diagram (such as at 2), and the control system, and ultimately the plant construction, commissioning, and operation, described at 3. The engineering process starts by defining a chemical reaction which combines (in our example) two pre-products (pre-product 1 and 2) and results in resulting-product 3 (at 1 in Figure 16). This chemical process is performed by a plant (at 2) which details a reactor, represented by a so-called P&ID (Pipe & Instrumentation Diagram) as defined in IEC standard 62424. The P&ID describes all technical resources of a plant i.e. pipes, valves, vessels, pumps, heat exchanger, and many more, as well as the requirements for the automation system in terms of measurement and actuation points. These measurement and actuation points have to be implemented by suitable automation devices (at 5) which become part (at 6) of a control system (at 3). The chemical process, i.e. the products and the reaction are the basic for the design of the plant and the control system. The engineering staff or engineering tool (at 4) performs the design, the operation and maintenance of the plant and the control system.

The reactor is part of the overall plant structure. Let's take a reactor temperature measurement device 5 and 6 in Figure 1 as an example because it is required for proper reactor function. It is desirable that consistency checks or even generation of possible solutions fulfilling all constraints of chemical reaction, mechanical, DCS, electrical, control logic, and supervision by possible using capability/skill models in the future. Today, however, many steps are performed manually because the common knowledge necessary to come to the right decisions is not available in machine interpretable information models. The following describes the engineering issues based on capabilities and skills more in details.

To initiate the chemical reaction the reactor must contain the reactants (pre-product) and reach a specific temperature. This means that a measurement device is needed, which offers the capability to provide temperature measurement (1a in Figure 17). The design engineer or the engineering tool must search for devices which offer the capa-

Figure 16: Engineering of a Chemical Process in Industrie 3.0 Systems



Source: IEC

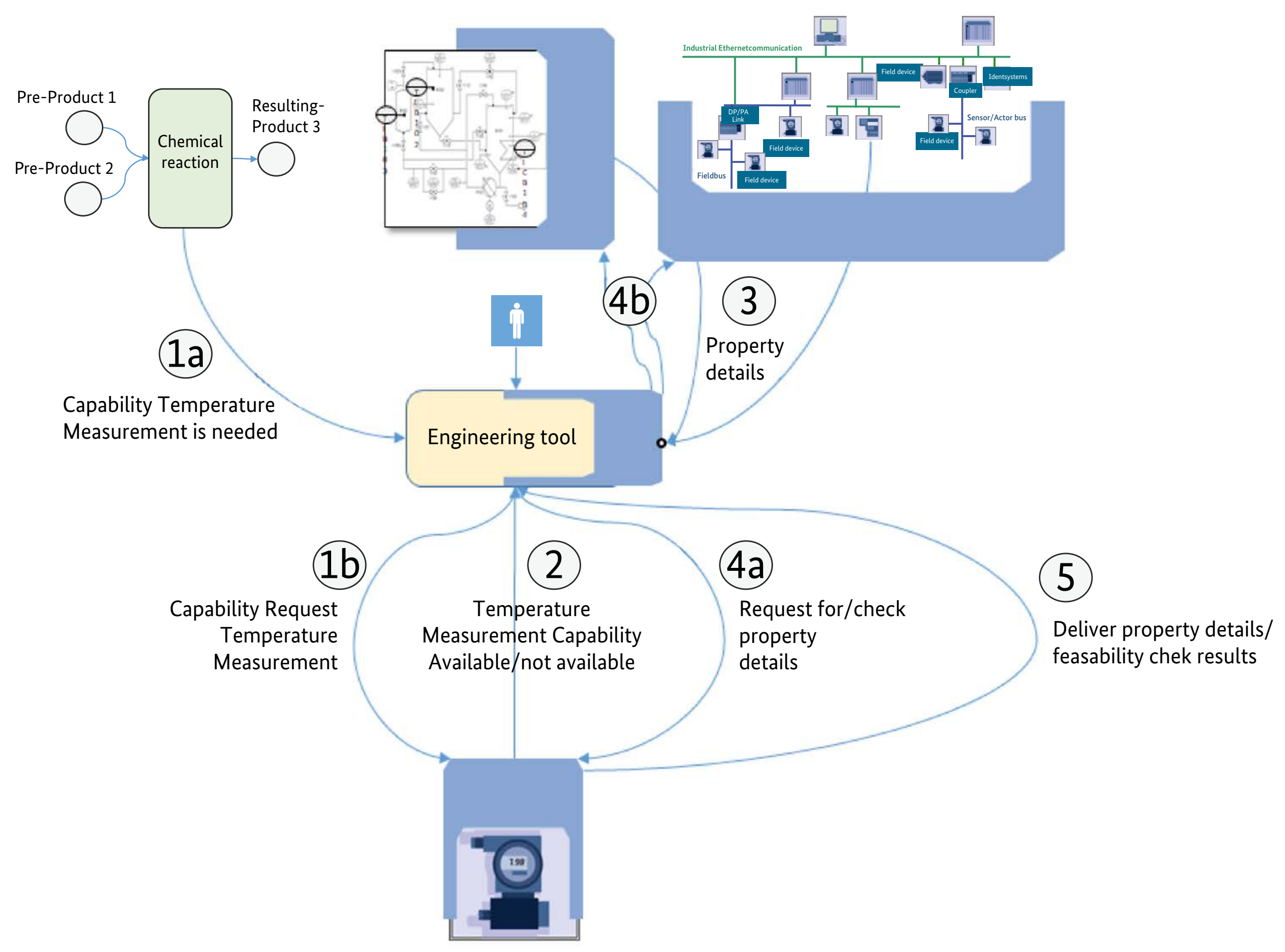
bility "temperature measurement" or "temperature value" (1b in Figure 17). Main properties such as the required temperature range must be offered by the measurement device and therefore checked during capability check. The AAS of such kind of devices must have the capability in our example "temperature measurement".

If there are AAS with this capability (2 in Figure 17) a feasibility check must be started. The temperature measurement device is normally mounted to the reactor vessel (measurement point in the P&ID[6] (IEC 62424)). For this connection, mechanical and geometric models are necessary to define the right mounting location and the mounting style (e.g., bolt the sensor to the vessel flange – not visualized in Figure 17). Additionally, the sensor needs a housing designed to withstand the temperature range the reactor will experience (not visualized in Figure 17). The design engineer or the engineering tool must check and compare many properties, like metal-liquid compatibility, sensor housing and flange mount screw diameter/pitch, and reactor/sensor temperature ranges. This is also true for the electrical, communication or digital value aspects of the measurement device. These required properties are results of design steps during the planning phase and available for the engineering tool (3 in Figure 17).

All these properties must be described in one or more than one submodels of the measurement device AAS and the AAS of the other planning resources. Each property has its description in terms of its attributes such as the range of the temperature measurement value, the offered bolts of the sensor or the allowed metal-liquid combinations of the sensor housing. The feasibility check starts with the request of the necessary property details (4a and 4b in Figure 17) which is delivered from the AAS (3 and 5 in Figure 17). Then the engineering tool or the design engineer must compare the requested and provided property details. More advanced AAS can process the check by itself and deliver the feasibility check result to the engineering tool (alternative 5 in Figure 17).

6 P&ID – Pipe & Instrumentation Diagram

Figure 17 General Capability and Feasibility Check Example for Measurement Device Selection


Pre-Product 1
Pre-Product 2
Chemical reaction
Resulting-Product 3
Industrial Ethernetcommunication
Field device
Identsystems
Coupler
DP/PA Link
Sensor/Actor bus
Field device
Field device
Fieldbus
Field device
1a
Capability Temperature Measurement is needed
4b
3
Property details
Engineering tool
1b
Capability Request Temperature Measurement
2
Temperature Measurement Capability Available/not available
4a
Request for/check property details
5
Deliver property details/ feasability chek results

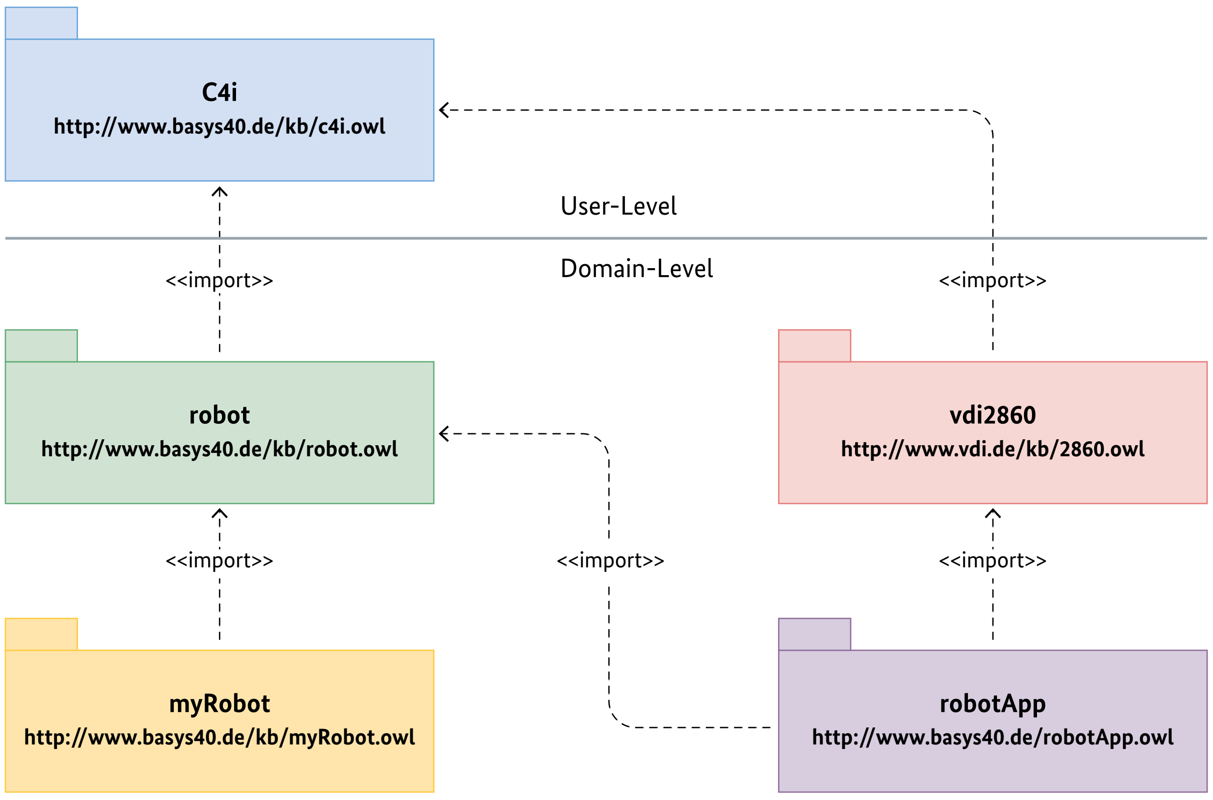
Source: Plattform Industrie 4.0

6.6 Capability of a Robot Arm

This example illustrates the advantages of using semantically expressive capability models in terms of ontologies. The focus of this example is on the ontology modelling side. For an example regarding the reference of capabilities from within the Asset Administration Shell, see section 6.7.

The capability ontology can be realized as a set of ontology modules where an upper level module defines the meta model for capabilities, and more specific domain modules define concrete capabilities. Figure 18 illustrates a possible import hierarchy for ontology modules describing robot and robot application related capabilities. An upper level ontology **c4i** specifies basic ontology classes and properties in order to guarantee compatibility of capability descriptions. On the domain level an ontology **robot** defines generic robot capabilities, and another ontology **myRobot** specializes this further to describe capabilities, e.g., provided by a particular robot manufacturer. On the other hand, a module **vdi2860** could describe capabilities as defined in the VDI 2860 [VDI 2860] guideline from a handling process point of view. Importing and specializing ontologies allows for reusing multiple existing modules, e.g., to describe specific robot-based handling operations, such as in module **robotApp**.

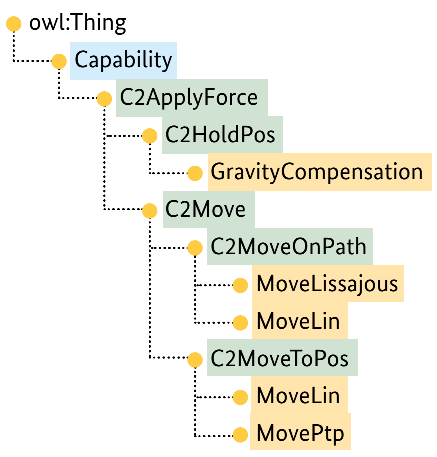
Figure 18: Import Hierarchy of Ontologies Realizing a Modular Capability Model


C4i
http://www.basys40.de/kb/c4i.owl
User-Level
Domain-Level
<<import>>
<<import>>
robot
http://www.basys40.de/kb/robot.owl
vdi2860
http://www.vdi.de/kb/2860.owl
<<import>>
<<import>>
<<import>>
myRobot
http://www.basys40.de/kb/myRobot.owl
robotApp
http://www.basys40.de/robotApp.owl

Source: BaSys 4.2

Considering the whole import closure, the set of axioms constitutes a subsumption hierarchy as depicted in Figure 19. The colors indicate the ontology module in which the capability definition is defined. In this example, the class **Capability** is defined in the **c4i** upper-level ontology module specifying that capabilities must be modeled as OWL classes in order to be compliant to this model. Generic robot capabilities such as **C2MoveToPos** or **C2MoveOnPath** as two specializations of **C2Move** are defined in the **robot** ontology module. Depending on the feature portfolio of a specific robot manufacturer, more specialized capabilities such as **MoveLin**, **MovePtp** or **MoveLissajous** can be defined in a vendor-specific and vendor-authored ontology module **myRobot**. Note, that it is possible to state more than one super-class in OWL, e.g., **MoveLin** can be a specialization of **C2MoveOnPat**h as well as **C2MoveToPos**.

Figure 19: Capability Specialization as Ontology Class Hierarchy



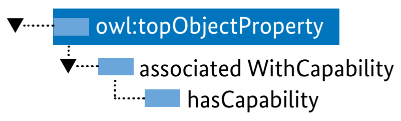
Source: BaSys 4.2

Note that the figures here show only an excerpt of the identifiers for the sake for brevity. In fact, each ontology entity can be identified by the ontology IRI and an IRI fragment. For instance, the ontology class representing the capability C2MoveToPos has identified by the full IRI http://www.basys40.de/kb/robot.owl#C2MoveToPos following the modularization example from Figure 18.

From the Description Logics based underpinning of OWL, a class subsumption relation implies that an individual/instance asserted to a class is implicitly asserted to all its super-classes. For the capability model this means that a capability of class **MoveLin** is also a capability of class **C2MoveOnPath**, **C2MoveToPos**, **C2Move**, **C2ApplyForce**, and that it is a **Capability** in general. This inference can be utilized by the capability references from Processes and Resources. Let, for instance, the Asset Administration Shell of a specific Process asset refer to **C2MoveToPos** as a required capability. A capability checker would identify every resource as a match, whose Asset Administration Shell refers to any capability as offer (or assurance) that is modeled as a subclass of **C2MoveToPos**, so as for example **MoveLin** or **MovePtp**.

Apart from capability hierarchies, ontology models allow for describing the composition of capabilities. This is particularly useful in compound components that are composed of several subcomponents working together, such as a robot system comprising a robot and a gripper. Modeling composition in ontologies can be realized using OWL properties. These properties would be part of the capability meta-model and thus described in the upper-level ontology **c4i** as depicted in Figure 20.

Figure 20: Property Hierarchy of the Ontology-based Capability Meta-Model



Source: BaSys 4.2

According to this an object can be associated with a capability or can have a capability. The property hierarchy states that if an object has a capability, it is implicitly also associated with this capability. **hasCapability** hence is a stronger relationship than **associatedWithCapability** and it should be used in a way that **hasCapability** refers to all capabilities an object offers (or assures) itself, whereas **associatedWithCapability** could refer to capabilities that a subcomponent of this object is offering (or assuring). In combination with the capability classes, this way of modeling allows for defining general class descriptions, such as

$$\exists \text{hasCapability.C2Hold}$$

as the class of objects that have the capability to hold something (e.g. according to VDI2860). A logical axiom could then state

$$\exists \text{hasCapability.C2Hold} \sqcap \exists \text{hasCapability.C2Release} \sqsubseteq \exists \text{hasCapability.C2Grasp}$$

which means that if an object has the capability to hold and it has the capability to release, then it also has the capability to grasp.

In the case of compound components, an axiom such as

$$\exists \text{associatedWithCapability.C2MoveToPos} \sqcap \exists \text{associatedWithCapability.C2Grasp} \sqsubseteq \exists \text{hasCapability.C2PickAndPlace}$$

could state that a component (e.g. a robot system) that has a subcomponent which has the capability **C2MoveToPos** (e.g. a robot), is then also associated with this capability **C2MoveToPos**. Analogously, the component is associated with the capability **C2Grasp** if it has a subcomponent (e.g. a gripper) which has this capability C2Grasp. The axiom states that if both conditions hold it can be deduced that the component has the capability **C2PickAndPlace**.

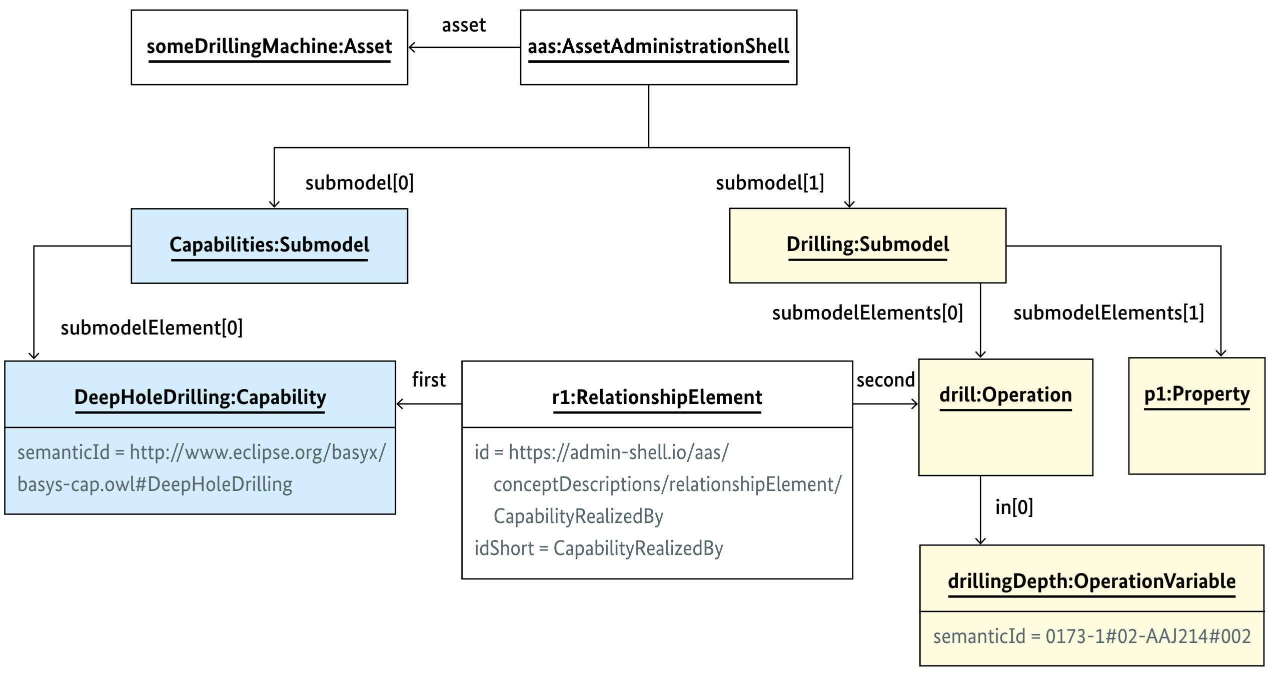
Important note: The above notion of ontology axioms in terms of Description Logics notation is for brevity reasons. This internal modelling and the formal representation are not required to be understood by any provider of capability descriptions. In fact, the modelling of capability hierarchies and composition will most likely follow a manageable set of modelling patterns, which can be used to provide templates and tool support for easy, form-based authoring of capability models. This way, existing capability ontologies can be reused and extended by domain experts, associations, companies, etc.

6.7 Capability of a Drilling Machine

Figure 21 gives an example of a drilling machine. In this example, there is a submodel Capabilities that is intended to collect all capabilities this asset may provide. In this case, there is one capability DeepHoleDrilling. The semanticId of this Capability element points to the respective class in an ontology, that specifies the semantics of DeepHoleDrilling.

A second submodel Drilling provides access to the skill drill via an operation. The operation drill is linked to the Capability element by RelationshipElement r1 which uses the above introduced semantics CapabilityRealizedBy.

Figure 21: Asset Administration Shell Structure of a Drilling Machine


someDrillingMachine:Asset
asset
aas:AssetAdministrationShell
submodel[0]
submodel[1]
Capabilities:Submodel
Drilling:Submodel
submodelElement[0]
submodelElements[0]
submodelElements[1]
DeepHoleDrilling:Capability
semanticId = http://www.eclipse.org/basyx/basys-cap.owl#DeepHoleDrilling
first
r1:RelationshipElement
id = https://admin-shell.io/aas/conceptDescriptions/relationshipElement/CapabilityRealizedBy
idShort = CapabilityRealizedBy
second
drill:Operation
p1:Property
in[0]
drillingDepth:OperationVariable
semanticId = 0173-1#02-AAJ214#002

Source: Plattform Industrie 4.0

7 Conclusion and Outlook

Various Industrie 4.0 use cases require an increase in the flexibility in production systems as well as interoperability across vendors. A key solution to achieve this is to increase the abstraction level of system design by focusing on capabilities required to perform tasks instead of concrete resources that are offered by specific vendors. To this aim, machine-readable capability description and matching, feasibility checking, the ability to match capabilities to concrete skill implementations and finally executing skills become important topics to explore.

As Asset Administration Shell is the key means to describe information about assets and is regarded as an interaction façade among the assets, capability descriptions should be incorporated as part of Administration Shell of assets. The proposal provided in this paper is the initial step towards capability modeling in Asset Administration Shell and its binding to skill implementation. Within the context of the BaSys 4.2 project, this proposal will further be assessed using various examples.

To ensure interoperability in production systems, it is essential to have standardized capability ontologies with a high degree of semantic expressiveness. Although various taxonomies exist, they are still in a preliminary stage and low semantic expressiveness. This topic must be further developed by relevant standardization communities.

To offer a high degree of flexibility in production systems, we should be able to specify the required capabilities as general as possible and provide means to match these specifications to the specification of provided capabilities of resources. Achieving the right level of abstraction in describing capabilities to be able to perform the matching and also to achieve the desired flexibility in production systems requires deeper study.

8 References

[AASiD Part 1] Specification Details of the Administration Shell – Part 1: The exchange of information between partners in the value chain of Industrie 4.0; Version 2.0. Federal Ministry for Economic Affairs and Energy (BMWi). Plattform Industrie 4.0, Berlin 2019. https://www.plattform-i40.de/PI40/Redaktion/DE/Downloads/Publikation/Details_of_the_Asset_Administration_Shell_Part1_V2.html

[Composites] Working Paper. Relationships between I4.0 Components – Composite Components and Smart Production. Continuation of the Development of the Reference Model for the I4.0 SG Models and Standards. June 2017. Federal Ministry for Economic Affairs and Energy (BMWi). Plattform Industrie 4.0. https://www.plattform-i40.de/PI40/Redaktion/DE/Downloads/Publikation/hm-2018-relationship.html

[DIN SPEC 92000] DIN SPEC 92000. Data Exchange on the Base of Property Value Statements (dt. Datenaustausch auf der Grundlage von Eigenschaftsausprägungsaussagen). ICS 25.040.01; 35.240.50. September 2019.

[Plattform] https://www.plattform-i40.de/PI40/Navigation/DE/Industrie40/Glossar/glossar.html

[CapAbstraction] Malakuti, S.; Bock, J.; Weser, M.; Venet, P.; Zimmermann, P.; Wiegand, M.; Grothoff, J.; Wagner, C. & Bayha, A.: Challenges in Skill-based Engineering of Industrial Automation Systems. In: IEEE. : 23rd IEEE International Conference on Emerging Technologies and Factory Automation, ETFA 2018, Torino, Italy, September 4–7, 2018., 2018, S. 67–74

[eCl@ss] https://www.eclass.eu

[mDNSProtocol] http://www.multicastdns.org/

[RuntimeVerification] Sánchez, C., Schneider, G., Ahrendt, W. et al. A survey of challenges for runtime verification from advanced application domains (beyond software). Form Methods Syst Des 54, 279–335 (2019). https://doi.org/10.1007/s10703-019-00337-w

[OWL 2] PARSIA, Bijan, PATEL-SCHNEIDER, Peter, MOTIK, Boris. OWL 2 Web Ontology Language Structural Specification and Functional-Style Syntax (Second Edition). 2012.

[Studer] STUDER, Rudi, BENJAMINS, V. Richard, FENSEL, Dieter. “Knowledge Engineering: Principles and Methods”. Data & Knowledge Engineering. 1998, vol 25, no. 1–2, p. 161–197.

[Heh11] Hehenberger, P.: Computerunterstützte Fertigung. Eine kompakte Einführung. Springer Berlin, Berlin[u.a], 2011.

[OPC UA Programs] OPC Foundation. OPC Unified Architecture, Part 10: Programs. 2015.

[VDI 2860] VDI. (1990). VDI 2860:1990-05 Montage- und Handhabungstechnik; Handhabungsfunktionen, Handhabungseinrichtungen; Begriffe, Definitionen, Symbole. Beuth.

[C4I Ontology] https://wiki.eclipse.org/File:2020-02-28_BaSys42_D-2.1.C4I_Ontology_Model.pdf

[ETFA2020] Michael Weser, Jürgen Bock, Siwara Schmitt, Alexander Perzylo, Kathrin Evers. An Ontology-based Metamodel for Capability Descriptions. ETFA 2020.

AUTHORS

Andreas Bayha, fortiss GmbH | Jürgen Bock, KUKA Deutschland GmbH | Birgit Boss, Robert Bosch GmbH | Christian Diedrich, ifak e.V. | Somayeh Malakuti, ABB AG

This paper is based in working groups of the BaSys 4.2 projekt, the working group VDI GMA 7.20 and working groups of the Plattform Industrie 4.0.

www.plattform-i40.de