Towards a formal ontology of engineering functions, behaviours, and capabilities

Francesco Compagno [a,*] and Stefano Borgo [a]
[a] *Labotory for Applied Ontology (LOA), Institute for Cognition Science and Technology (ISTC), Trento, Italy*
E-mail: francesco.compagno@loa.istc.cnr.it
E-mail: stefano.borgo@cnr.it

Abstract.
In both applied ontology and engineering, functionality is a well-researched topic, since it is through teleological causal reasoning that domain experts build mental models of engineering systems, giving birth to functions. These mental models are important throughout the whole lifecycle of any product, being used from the design phase up to any diagnosis activity. Though a vast amount of work to represent functions has already been carried out, the literature has not settled on a shared and well-defined methodology yet. This work develops preliminary steps towards an ontological description of functions and related concepts, such as behaviour, capability, and capacity. A conceptual analysis of such notions is carried out using the top-level ontology DOLCE as a framework, and the ensuing logical theory is formally described in first order logic and OWL, showing how ontological concepts can model major aspects of engineering products in applications. In particular, it is shown how functions can be distinguished from implementation methods, and how functions can differentiate between capabilities and capacities of a product.

Keywords: Ontology, Function, Behaviour, Capability

1. Introduction

Functionality is a concept that has interested engineers as well as philosophers and applied ontologists. It is referenced whenever engineers and scientists discuss the goals of systems, both natural and artificial. For example, engineers commonly use functions in order to design [1] and diagnose devices [2], while philosophers discuss the nature of functions themselves [3].

Despite the high amount of attention given to this topic, some problems have yet to be settled. For instance, the terminology used is quite ambiguous and words such as *capability*, *capacity*, *behavior*, or *function* itself have been used with many different meanings [4, 5], and are characterized differently, if at all. Moreover, the study of *functional decomposition*, that is, the division of functions into sub-functions, is often addressed in the literature as a means to guide the conceptualisation, design and maintenance of products, but is rarely formalized. By and large, functional decomposition consists in associating the product function with a combination of sub-functions whose execution (in a certain order and with suitable coordination) is equivalent to the execution of the main function. This decomposition simplifies both the design process and the implementation of the functional requirements into a concrete physical system. Of course, functional decomposition is not limited to the design of engineering systems. In fact, during the teleological analysis of a system, artificial or natural, domain experts speak about functions of both

*Corresponding author. E-mail: francesco.compagno@loa.istc.cnr.it.

the system and its parts. Therefore, one is always confronted with the problem of how simpler functions contribute to coarser granularity functions, so that functional decomposition is ubiquitous. Since in engineering there exist standard ways to execute functions[1], we can speak of *engineering methods*[2], or solutions [1], or 'ways' of functional achievement [6].

This paper builds on previous research, especially [4] and [7], to discuss what functionality and related concepts are, in a formal way. The formal languages used in the paper are first order logic [8], adopted for the general discussion of the concepts and their relationships, and OWL [9] for the use case; additionally, to give a clear ontological grounding for our formalisation, we use the top-level ontology DOLCE[3] as reference. Moreover, we will discuss the relation between functions as entities independent of any implementation, which we call *ontological functions*, functions that depend on the teleological analysis of a given system, that is, functions contextualised to a system, that we call *systemic functions*, and functions as entities related to an execution method, which we call *engineering functions*. Finally, ontological functions will be leveraged to propose a general distinction between capabilities and capacities of engineering artifacts.

The structure of the paper is as follows. A review of the literature about functionality that is particularly relevant for this work is presented in Section 2. Section 3 describes key concepts of DOLCE. These are exploited in Section 4 for the discussion of capabilities, capacities, behaviors, and functions. This section proposes also a formal interpretation of ontological and engineering functions in first order logic, and concludes with a preliminary characterisation of capabilities and capacities. We showcase a preliminary ontology in the OWL language in Section 6, before drawing our conclusions in Section 7.

2. Literature review

Due to strong practical interests, a vast literature about functionality has been developed within engineering. We limit ourselves to a brief review. The reader interested in a more in depth analysis can refer to other works like [5] in engineering and [12] in applied ontology.

Fundamental works about functionality are, for example, De Kleer's [13, 14], which outline foundational principles such as the locality of functions (functions of a component should refer only to neighboring entities) and the 'no function in structure' principle (structural models should not contain teleological information); and Pahl and Beitz's [1], which describes functions as black-box transformations applied to input flows, divides functions based on the property of the flow acted upon (e.g., quantity, type, position, etc.), and discusses function decomposition (that is, how to achieve a given function through a structure of adequate sub-functions).

In [1] Pahl and Beitz explain also how designers can start from a generic function and implement it with solutions based on different physical principles. Then, they reference many design catalogues, for example Roth's [15], which tabulate solutions and classify them along different criteria. In the context of design catalogues, functions and solutions exist along the same abstract-concrete axis, and differ by the 'degree of embodiment'. Thereby, the most abstract functions, called 'generally valid functions', are the most useful criteria to classify solutions in a product-independent way.

The definition of functions as (selected) actions on flows is the most common type encountered in the engineering literature, for example Chandrasekaran et al. [16] speak of 'intended input-output relations', and many vocabularies were proposed in order to standardize the terminology of such actions. The proposed vocabularies range from Keuneke's short four terms list (toMake, toMantain, toPrevent, and toControl) [17], to larger vocabularies built using complex algorithms, as Kitamura et al.'s [18]. A common and often referred to example is Stone and Wood's Functional Basis [19, 20], which gives terms for actions and flows on a three-levels taxonomy described in natural

[1] For example, the use of gearboxes made in a certain way in order to increase or reduce angular velocity. Another example: a brushless electric motor may be understood as the class subsuming all models of brushless electric motors, but it can also be recognized as a method by which electric energy can be converted to torque.

[2] If the method is not standard, say if it has just been introduced, we still call it an engineering method. In fact, in such a case, the term can refer to the principles and theories, which are shared among engineers, that the implementation follows.

[3] The interested reader can find a complete and in-depth presentation of DOLCE in [10] and its application in use cases in [11].

language. The literature about the use of such vocabularies has not settled yet and, currently, different lines of research are being pursued, such as behavioural simulation of a system from the functional model [21] or formal description and automatic construction of the functional model itself [22, 23].

It is generally recognized that functionality depends on the intention of an agent, a typical example being the designer's intent [24], also called 'design rationale' [16]. Thus, functions are not objective, at least not completely. Then, it is natural to wonder what in a function is fully objective. Behaviour, intended as what a device does, as determined by physical laws, is usually the answer.

Even the term behaviour is used ambiguously and without any universally shared definition. The ambiguity has such consequences so that some authors classified the multiple uses of behavior in engineering literature: Kitamura et al. [24] determine four types of device behaviour[4], depending on whether the device changes the state of another device, its own state, or the state of one of its operands. Within the second behavior type two additional cases are distinguished: the operand can either stay in the same position or 'move' from the input to the output ports (the latter case is called *B1 behavior*, for example, 'a motor converts electrical energy to torque' and 'the signal intensity is increased by the amplifier' are examples of B1 behavior, while 'the temperature in the room is increased by the oven' is not a B1 behavior, neither is 'the turbine is rotating'). Instead, in [25], Chandrasekaran and Josephson divide behaviours mainly depending on their time duration (instant, interval, or unspecified).

The link between function and behaviour generally reflects the duality between objectivity and intentionality: some authors state that behaviour is what a device does, whilst function is what a device is for [14], others focus on describing behaviour as a sequence of state changes and functions as abstractions of behaviours with a goal in mind [26]. In [27, 28], Sasajima et al. state that function is made from behaviour plus additional teleological information called 'functional topping'. Specifically, the functional topping allows, among other things, distinguishing the different functions that a device can execute (e.g. an electrical resistor could be devised for either heating the environment or for dropping the input voltage, and the functional topping in these two cases is different[5]

In [25] functions are the sum of intentions and sets of behavioural constraints, defined, essentially, as causal relations between devices states.

In any case, to date numerous modelling frameworks have been developed in order to deal with functional and behavioural representation of an engineering systems (e.g., Umeda et al.'s FBS [26], Sasajima et al.'s FBRL [27], Sembugamoorthy and Chandrasekaran's FR [29], Goel et al.'s SBF [30], Qian and Gero's FBS [31]). It would be impossible to explain briefly the characteristics of and the differences between these frameworks. Still, oversimplifying, one can isolate commonalities: all of them tend to see the structure of a device as a set of parts (components) together with their attributes and the relations between them. Then, typically, they say that a behaviour is a sequence of states of the components. The definition of function is more complex, but, *de facto*, a function is often used as a label for a subset of behaviours. Finally, they often recognize a dependence of functions on behaviors, and of behaviors on the structure. There are, of course, key differences. For example, behavior in Chandrasekaran's FR scheme refers to a causal process while behavior in Gero's FBS representation pertains to the properties of a structural component. Or the fact that Gero and Chandrasekaran explicitly allow for decomposition of functions into behaviors, while in Sasajima's approach a decomposition of functions in behaviors is forbidden and functions decompose only into other functions.

Among the previous frameworks, we find particularly interesting the development of FBRL. This is because, while for engineers functions usually are either already-assigned functional requirements (things that a product must be able to do) or are not distinguished by implementation methods, the authors of FBRL started, in our opinion, to study functions as entities by themselves. For example, at least from [32], they studied relations between functions themselves (called 'meta-functions'), and analyzed the properties of functions, building complex taxonomies of functional concepts. This is important for our work, since it is one clear example of what we call ontological functions, while the engineering functions are the ones that, as in Pahl and Beitz's work, exist only to be implemented by some method in an application context.

[4]They explicitly exclude static behaviours such as supporting, though.

[5]More precisely, when the resistor is used for heating, the functional topping tags as 'Focus' the heat flow exiting the output port, when the resistor is used for dropping the voltage, it is the electric energy output of the resistor that is tagged 'Focus'.

Most of the frameworks used for modelling functions come from the engineering community and tend not to be grounded in, nor aligned with, ontological theories. The presence of explicit connections between these framework and ontological theories would be quite beneficial, especially because of the large number of such engineering systems and of the importance of clarifying the use of the terminology. Moreover, engineers often do not make use of formal languages such as first order logic or OWL[6], instead preferring informal descriptions in natural language or pseudo-code[7]. In addition, some topics such as the link between function and malfunction, the difference between function and behaviour, or the methodology with which to carry out the decomposition of a function have been tackled differently by these systems but, overall, it does not seem that a reliable and shared view is emerging.

Despite the utility of an ontological analysis of these topics, the attempts carried until now are limited. A few research works have formalized part of the FR system [34], started a formalisation of the Functional Basis vocabulary (and the conceptualisation it is based on) with the upper ontology DOLCE [35, 36], compared the Functional Basis either with the FR system [37] or with both FR and FBRL [38], and the function decomposition relation has been analyzed with ontological techniques [39, 40]. Additionally, applied ontology has influenced engineering literature, as shown by the description of functions as roles played by behaviours [6, 25, 41], or the classification of behaviours as occurrents [6].

Still, no shared reference ontology of functions exists, and top-level ontologies do not explore theories of functions as understood in engineering. For example, DOLCE makes no mention of function within its specification [10], and only later work was proposed to see function execution as a particular type of events, namely achievements [4]. The last version of YAMATO classifies functions as roles and behaviours as processes[8] caused by an unintentional actor, and delegates the description of a functional ontology to a series of additional papers [6, 42, 43]. BFO defines functions as dispositions that exist in virtue of their bearer's physical make-up, and such that the physical make-up came into being through intentional design (in the case of engineering). Other than that, BFO does not commit to an axiomatisation, at least not within the axiomatisation of BFO currently present in GitHub[9]. GFO too states that functions can be realized by other entities, but makes no mention of dispositions and, instead, calls functions 'intentional entities' [44]. The last development of ISO 15926 [45] adopts BFO view of functions as dispositions realized in particular processes.

As we have seen, the meaning of function is ambiguous even within the ontology community. Moreover, it is generally not considered a top-level concept, and thus it is marginally covered by top-level ontologies. Unfortunately, to our knowledge, the ontology community has not produced a shared middle or domain level ontology dealing with functions[10].

Given the current situation, an ontological analysis clarifying the domain of functionality would be quite useful. This paper aims to provide an initial step in the direction of developing a systemic ontological treatment of functionality, focusing in particular on the engineering domain. It might be true that the ambiguity of function terminology is both necessary and rational for engineers [47]. Still, we maintain that formalisation is useful in order to show differences between the possible approaches. Moreover, it is useful, if not necessary, to develop applications that rely on functional reasoning.

In conclusion of this section, we mention very briefly some facts about capabilities and capacities. These two terms, especially capability, are used in resource modelling [48–51]. In addition, terminological problems are present also for these two concepts, which are rarely formalized [4, 52]. When distinguished, capabilities and capacities are separated along a qualitative-quantitative axis, for example ISO 15531-31[50] states that 'Capacity is strictly a quantitative concept' while 'Capability is essentially a functional and qualitative concept', and exemplify capacity with product throughput and define capability as 'the quality of being able to perform a given activity'. The same

[6]There is, of course, also the fact that OWL was first published in 2004, while many works of functional modelling are older.

[7]There are exceptions, e.g. Kitamura et al. [6] and Yang et al. [33] showcase an implementation of an industrially deployed version of FBRL, called SOFAST, and an OWL and SWRL formalisation of the Functional Basis, respectively.

[8]Note that even if processes exist as a category also in DOLCE, in that TLO behaviours are still more similar to events and not processes. This is because the process category of YAMATO and the one of DOLCE are different: in YAMATO processes wholly exist at each point in time, while in DOLCE they are a special kind of events. YAMATO and DOLCE are instead aligned on the notion of events.

[9]BFO 2020: https://github.com/BFO-ontology/BFO-2020.

[10]The taxonomy of an ontology of functions, which is still under development, was proposed by Borgo et al. in [4] and [46].

standard advises against reducing capacities as characteristics of capabilities and forbids the opposite (in contradiction to [51], where a 'Capacity is a Capability expressed in terms of amount of production'). In any case, in this paper we will try to formalize, in a preliminary way, some intuitions that transpire from the literature on resource modelling, such as the asymmetry between capacities and capabilities, the close link (but not identity) between capabilities and functionality, qualitative vs quantitative aspects, and the idea of capabilities as qualities of being able to do something.

3. An (enriched) subset of the DOLCE ontology

Here we introduce the fragment of the DOLCE ontology [10, 11] that is needed in this paper. In particular, we cover the following classes which will be needed to model functions and related concepts: *qualities*, *perdurants*, and *roles*.

DOLCE qualities are similar to tropes [53]. Qualities, for example the weight of a car and the color of a flower, are used for representing properties (e.g., weight) of individual objects (e.g., car) in which they inhere. Ontologically, qualities are individuals, that is, the colors of two different flowers are different, even though they could be both the same shade of red. Differently than tropes, qualities can change in time, for example the color quality of a flower can change its value as time passes: red in the summer and brown in the fall. This is because values of qualities are separated from qualities themselves and are called quales. In this way, the assignment of values to qualities is more flexible and follows the schema object(carrier)-quality(individual property)-quale(value). In DOLCE, qualities form the class $\mathtt{Q}(\cdot)$ and their inherence relation is written $\mathtt{qt}(\cdot,\cdot)$, 'quality-of'. A temporal quale relation associates a quality with some value which, as said, may be different at different times. (The temporal quale relation will not be used in this paper.)

We add a distinction among qualities, which is not in DOLCE, to separate *intrinsic* qualities, such as mass, length, or shape, and *relational*, or extrinsic, qualities, such as weight (which is a comparative property), the personal record for a marathon (which is relative to the definition of marathon), or the distance of an object from another object. Relational qualities are not qualities of an object *per se*, but depend on other things like the context or something in the environment. To clarify our intuition we illustrate some of the previous examples: the distance of the Moon from the Earth, seen as a property of the Moon, cannot be thought nor measured without considering the Earth, so we say that it is a relational quality of the Moon. In contrast, the color of a material object, as a property of the material itself, does not depend on other entities (not even light), so that is an example of intrinsic quality. Another example is the difference between weight and mass of a body: the weight also depends on the position of the body, say if the body is on Earth or on the Moon, so it is relational. A more technical example is the voltage at a point of a component, which, since it is a potential, requires a second point used as reference (the ground of the electrical circuit) in order to be measured. Informally speaking, capacities can be understood as having a relational nature, which explains our interest in the intrinsic/relational distinction across qualities. For, if we speak about the capacity of a device, say the capacity to process a certain number of items in a given time, then such capacity always refers to another entity, in this case the items. Analogously, in [31], Qian and Gero stated that there are different types of 'behavioral variables': structural, as the area of a room or the diameter of a water tap, and exogenous, as the water flow through a water tap. In the latter example, the water flow is an exogenous variable because 'water is not part of the water tap design, it is only related to the design', so that we could argue water is an additional entity required by the water flow quality of the tap, suggesting that this form of exogeneity can be captured via our notion of relational quality.

In the previous examples, it seems that relational qualities are those qualities that depend, in some way, on an entity different (we say *external*, see (d2)) from their bearer. Unfortunately, the exact meaning of this dependence relation changes between the different examples. In particular, in the case of capacities the dependence is 'potential', for a device can have the capacity to process a product, even when the product is not actually present. In contrast, in the case of relative distance the dependence is 'actual', for a physical object is at any time at a certain distance from another.[11] It follows that the characterisation of relational qualities is a complex matter that goes beyond the scope

[11] Note that relative distance makes sense only at times in which both objects exist.

of this paper.[12] Therefore, we just introduce relational and intrinsic qualities as complementary primitive subclasses of DOLCE-qualities:

a1 $\texttt{relationalQt}(x) \implies \texttt{Q}(x)$
d1 $\texttt{intrinsicQt}(x) \iff (\texttt{Q}(x) \wedge \neg\texttt{relationalQt}(x))$

Turning now to perdurants ($\texttt{PD}(\cdot)$), which we will also refer to as occurents in this paper, in DOLCE they are entities that are only partially present at any time they are present.[13] For example, a chemical process, the lifting of a load, and a sitting action are only partially present at each instant at which they happen. Indeed, the initial part of a chemical process is not present when the process reached the midway point, and vice versa. DOLCE uses three mereological properties to distinguish perdurants: cumulativity, homeomericity, and atomicity. Cumulativity holds if the sum of two instances of a type has the same type, that is, if the type is closed under mereological sum. Consider, for example, 'walking': if we consider two walking activities, then the activity that comprises both is still an activity of type 'walking'. Cumulative perdurants[14] are called *stative* ($\texttt{STV}(\cdot)$), while the ones that are never cumulative are called *eventive*. Homeomericity holds for a perdurant type if any parts of its instances are instances themselves. This is the case of, e.g, 'sitting', since portions of a sitting action are still sitting actions. Stative homeomeric perdurants are called *states* ($\texttt{ST}(\cdot)$), while the ones that are stative but have parts of different type are called *processes*($\texttt{PRO}(\cdot)$). Walking itself is an example of process in DOLCE: walking requires at least to complete a certain leg movement, below such granularity is not a walking movement. Another example is the buzzing of a clapper: the clapper alternates between two states when clapping (opened/closed circuit), and neither state is per se of buzzing type. In DOLCE the temporal relationship between a perdurant and the object participating in it is called *participation*, written $\texttt{PC}(\cdot,\cdot,\cdot)$: in the previous example a participant of the clapping is the buzzer.

Coming to roles, they were not covered in DOLCE originally. They have been introduced later as an extension [55], and are now part of the expanded taxonomy [11]. Roles are antirigid and dependent, or founded, classes [55–57]. A class is antirigid whenever its instances are not necessarily so, and is dependent if all of its instances (existentially) depend on some external entity, often called context. For example, a certain person can be a student, but no person is necessarily a student. Actually, we expect that a student ceases to be such after some time. In contrast, any person is necessarily a person, and is so independently of other external entities. An ontological class (existentially) depends on, or is founded on, another if, whenever an instance of the first class is present, a corresponding instance of the second is present too. For example, for every citizen there is a country, so that someone's citizenship depends on the country. Actually, in this paper, we will be more precise and distinguish between different kinds of dependence and founding, but the basic idea is still captured by the citizen-country example. Additionally, some authors divide roles depending on the type of their context, for example Loebe distinguishes relational, processual, and social roles in [58]. The classification depends on whether the context is a relation, a process, or a social object. In DOLCE roles ($\texttt{RL}(\cdot)$) are reified and considered as concepts ($\texttt{CN}(\cdot)$), which themselves are a class subsumed by the class of non-agentive social objects ($\texttt{NASO}(\cdot)$):

a2 $\texttt{RL}(x) \implies \texttt{CN}(x)$
"a role is a concept"
a3 $\texttt{CN}(x) \implies \texttt{NASO}(x)$
"a concept is a non-agentive social object"

In this paper, the fact that roles are founded is particularly important, thus, we give the following formalisation, where $\texttt{CL}(\cdot,\cdot,\cdot)$ ('classified-by', also called 'play-as', if the first argument is a role) is the classification relation between a concept and its instances at a certain time, cf. [55]. In the formalisation we use some other relations taken from DOLCE, namely: constitution, $\texttt{K}(\cdot,\cdot,\cdot)$, which is the relation holding between some amount of matter and an object when the latter is made of the first; $\texttt{O}(\cdot,\cdot,\cdot)$, which is the standard overlap relation; and $\texttt{PRE}(\cdot,\cdot)$, which is the relation "being present (exist) at time".

[12] The interested reader can find a proposal on relational and intrinsic qualities in [54].
[13] Ordinary physical objects such as cars, trees, rocks, etc. are considered fully present at every time in which they exist.
[14] More precisely, instances of a cumulative type perdurant.

d2 $\texttt{externalTo}(x,y) \iff \neg(\texttt{qt}(x,y) \vee \exists t(\texttt{K}(x,y,t)) \vee \exists t(\texttt{O}(x,y,t)))$
"x is external to y if and only if x is neither a quality of y, nor one of y's constituents[15](at any time), nor x and y have parts in common[16] (at any time)"

d3 $\texttt{dependsOn}(x,y) \iff (\exists t(\texttt{PRE}(x,t)) \wedge \forall t(\texttt{PRE}(x,t) \implies \texttt{PRE}(y,t)))$
"x existentially depends on y if and only if x exists at some time and at any time when x exists so does y"[17]

d4 $\texttt{founded}(x,y) \iff (\texttt{dependsOn}(x,y) \wedge \texttt{externalTo}(x,y))$
"x is founded on y if and only if x existentially depends on y and y is external to x"

Further, we specialize the founding relationship to concepts and their instances as follows:

d5 $\texttt{founded}_{\texttt{Inst}}(x,y) \iff (\texttt{CN}(x) \wedge \exists z,t(\texttt{CL}(z,x,t)) \wedge \forall z,t(\texttt{CL}(z,x,t) \implies \exists w(\texttt{founded}(z,w) \wedge \texttt{CL}(w,y,t))))$
"the concept x is instantiation-founded on the concept y if and only if, given any z that plays x, then z is founded on some instance of y"

a4 $\texttt{RL}(x) \implies \exists y \texttt{founded}_{\texttt{Inst}}(x,y)$
"if x is a role, then there is a y on which it is instantiation-founded"

For example, the role of 'husband' is instantiation-founded on the concept of 'marriage', since every time a person is a husband there is an individual marriage between that person and another person.

We also need to capture a different kind of founding relation, that we call *definition-founding* ($\texttt{founded}_{\texttt{Def}}(\cdot,\cdot)$). We do not formalize this relation as it requires discussing how to formally define roles, a topic beyond our concerns in this paper. We will use it with the following informal interpretation: "x is definition-founded on some entity y if and only if y is used to define x". For example, if a doctor is defined as a person who treats patients, then the doctor-role is definition-founded on the patient-role. Note that instantiation-founding and definition-founding need not coincide. A person is a doctor even though, at a certain time, she has no patient, so that the doctor-role is not instantiation-founded on the patient-role.

Finally, since roles, as well as concepts, can be seen as reified classes, there exists a specialisation relation between roles, which we write as $\texttt{SP}(\cdot,\cdot)$:

d6 $\texttt{SP}(x,y) \iff (\texttt{CN}(x) \wedge \texttt{CN}(y) \wedge \exists z,t(\texttt{CL}(z,x,t)) \wedge \forall z,t(\texttt{CL}(z,x,t) \implies \texttt{CL}(z,y,t)))$
"A concept x specializes a concept y if and only if all instances of x are also instances of y"

For example, the role of the Italian Prime Minister specializes the role of Prime Minister. This notion of specialisation is admittedly weak. One would like to add a modal characterisation: definition (d6) should hold in all possible worlds. This problem applies to other definitions we introduce in this paper and is not ontological but related to the limitations of first-order logic. Without discussing logical technicalities, in this paper we will make use of these characterisations assuming that, in suitable systems, e.g. first-order modal logic, a suitable modal formula is substituted.

4. Modeling behaviors and functions in DOLCE

In this section we propose a framework to characterize how behaviour and function of engineering artifacts can be understood to make sense of the distinctions used by engineers in different areas, from engineering design to manufacturing, from process planning and product planning to early system design planning. Within the following section, we will expand this view to capability and capacity. In this work we use DOLCE as our top-level ontology.

In the literature, many terms are used to refer to engineering systems and devices such as part, component, device, tool, machine, system, (technical) artifact, functional object etc. We use *technical artifact*[18] to mean any physical object (see (a5)) that comes into being through some intentional technical process, such as, e.g., cars, planes, tooling

[15]Constituent, or substratum, as in 'this fork is constituted by stainless steel.'

[16]Substrata, parts, and qualities may not cover all possibilities especially if a different top-level ontology were used.

[17]The existential quantifier is a technicality: without it an entity that is never present would depend on all entities. Similarly, existential quantifiers are introduced in axioms having a similar structure to this one.

[18]See [59] for a more in-depth discussion of technical artifacts.

machines, and, more generally, devices designed to perform tasks. Also, we will use the terms 'device' and 'technical artifact' as synonyms. Additionally, we will use the term *system* in order to highlight the mereological structure of a complex artifact. The notion of system is complex, cannot be reduced to that of technical artifact, and its precise characterisation is an open problem (see e.g. [60]). In the paper, we take this notion as given introducing a primitive class, $\texttt{System}(\cdot)$. In particular, technical artifacts belong to this class. The mereology mentioned above is built as usual from the temporalized parthood relation of DOLCE, $\texttt{P}(\cdot,\cdot,\cdot)$. DOLCE defines also a non-temporalized parthood relation, which we report in (d7). Axiom (a6) and the class $\texttt{POB}(\cdot)$, the category of physical objects, are also taken from DOLCE.

a5 $\texttt{TechArt}(x) \implies \texttt{POB}(x)$
"a tecnical artifact is a physical object"

d7 $\texttt{CP}(x,y) \iff \exists t(\texttt{PRE}(y,t)) \wedge \forall t(\texttt{PRE}(y,t) \implies \texttt{P}(x,y,t))$
"x is constantly part of y if and only if whenever x exists, it is part of y"

a6 $\texttt{P}(x,y,t) \implies (\texttt{PRE}(x,t) \wedge \texttt{PRE}(y,t))$
"if x is part of y at time t, then x and y are both present at time t"

4.1. Behaviors

Engineers create an artifact to realize a certain interaction between the artifact itself and elements of the environment. The behavior of the artifact is the way in which it participates in the interaction (e.g. '[the car] rattled when I hit the curve', [25]), In DOLCE one models the happening of the interaction as a perdurant. How to ontologically understand behavior is more tricky. In the literature, the term behavior is used with different meanings, e.g., as simplified part or description of processes like in these excerpts: 'The causal rules that describe the values of the variables under various conditions' [25], 'the behavior represents objective conceptualisation of its input-output relation as a black-box' [6], 'situation-independent conceptualisation of the change between input and output of the device' [43]. In YAMATO [61] behaviours are modelled as processes with an 'agent or an agent-like object as a doer'. Instead, in [34] Borgo et al. model them as relational qualities which characterize the specific way of participation of an object to individual events.

Here we discuss a few key ontological properties to distinguish possible definitions of behavior. There are, in fact, at least four axes along which different meanings of behavior can vary in the literature. First, there is what we call the occurrence-property dichotomy:

- Behavior can be something that happens (occurs) and in which the behaving entity participates, in this case it is typically referred to as a transition between states or just as (staying in) a state. Examples are Goel's [62], Chandrasekaran's [25], Umeda's [26], and YAMATO authors' [43] approaches.
- Behavior can also be a property, that is, something inhering into the behaving entity. In this case some authors speak of qualities [34], others of attributes or dispositions [63], [64].

Then, there is the token-type distinction: behavior can be a class or a concept, as for Mizoguchi in [43]. Alternatively, it can be an instance of something, or an entity relative to a specific event, as for Chandrasekaran and Josephson in [25], and for Borgo et al. in [34], respectively.

Additionally, there is the external-internal axis, that is, the behavior of an artifact may refer only to characteristics of the artifact itself, or it may need to refer to some external entities. For example, 'the electric switch can alternate between open and closed states' is an internal behavioral description, as it refers only to transitions between states of the artifact. In contrast, 'the current passing through an open switch is zero, if the applied voltage stays within operating conditions' is an external description, since, the current and the voltage are not intrinsic elements of the artifact. Some authors explicitly use behavior with the internal meaning, e.g. Zhao et al. in [65], others with the external one, as Kitamura et al. in [66].

Finally, there is the modal axis, since behaviours can be either expected (i.e. as envisioned by engineers) or actual (i.e. what actually happens), as implied by Gero in [67] with respect to design activity (though one could use the same duality when talking of, e.g., malfunctioning). This axis includes, arguably, talks of causal laws or relations, since those could be conceptualized as changes of a system under some kind of epistemic modality, that is, changes

that necessarily happen when some condition is met. We do not argue in favor of conceptualizing causal laws in this way, but, in any case, one must also take this use of behaviour into account, since it is encountered often. Other differences exist in the literature, such as device-centered vs process-centered [25], but we do not discuss them.

Following common practice in the literature, in this paper we take behaviors to form a subclass of DOLCE perdurants. This class is partially characterized by the fact that one can identify two 'processual'[19] roles in the sense of [58]: an active one, called *doer*, and a passive one, called *operand* or *flow* [1]. For instance, a cutting action entails that there is something that is the subject of the action, say a saw (or the system formed by a person or machine using the saw), and the object of the action, say a beam of wood. The same holds for pumping, joining, and other behaviors that can be described by transitive verbs. In fact, all device behaviors that can be described as operations on flows are of this kind. Such behaviors, whose class we characterize with predicate $\texttt{Behaviour}(\cdot)$, are external behaviors, since the flow is an entity external to the behaving artifact (the doer). We conceptualize the two processual roles through two specialisations of the participation relation: $\texttt{participatesAsDoer}(\cdot,\cdot,\cdot)$, to indicate participation in a process with the role of an agent-like doer, and $\texttt{participatesAsFlow}(\cdot,\cdot,\cdot)$, for the role of flow:

a7 $\texttt{participatesAsDoer}(x,y,t) \implies \texttt{TechArt}(x) \wedge \texttt{Behaviour}(y) \wedge \texttt{PC}(x,y,t)$
"if x is a doer in y at time t then x is a technical artifact, y is a perdurant, and x participates in y (in the DOLCE sense) during that time"

a8 $\texttt{Behaviour}(x) \implies \exists y,z,t(\texttt{participatesAsDoer}(y,x,t) \wedge \texttt{participatesAsFlow}(z,x,t) \wedge y \neq z)$
"x is a behavior if and only if there are at least a doer and a flow that participate in x"

Additionally, we assume that behaviors can be combined to give causal explanations of complex occurences (e.g. the beam was cut, therefore it fell to the floor), and formalize this with a binary relation called *causal contribution*[20], which we assume holds more in general between perdurants:

a9 $\texttt{causalContr}(x,y) \implies \texttt{PD}(x) \wedge \texttt{PD}(y)$

We do not axiomatize this relation further since it is outside the scope of this paper. Some initial proposal already exists, e.g., by Borgo and Mizoguchi [69][21].

In any case, whatever the definition of behavior one makes use of, one must take into account the fact that behaviors are perdurants seen from the point of view of the device. This is the main reason behind the approaches that define behaviors as relational qualities of devices [34]. In this paper we try to model the participating device point of view stating that any behavior must give raise to corresponding processual roles (a8). A comparison of the advantages and drawbacks of these two modeling choices has not been carried out yet. Nonetheless, we highlight that the two approaches are both compatible with DOLCE and could coexist (at the cost of rendundancy).

Having conceptualized behaviors as perdurants, we spend a few words about the notion of state of an engineering system. In DOLCE states are, as mentioned in Section 1, cumulative and homeomeric perdurants. The same properties should hold for engineering system states. Indeed, if we understand states, as many engineers do, as conditions determined by constraints over state variables, then such conditions are homeomeric (if a device satisfies a constraint over a time period, then it also satisfies it during a fragment of that period) and cumulative (if a device satisfies a constraint over some time periods, then it satisfies it during the union of those periods). Unfortunately, engineers commonly use the term 'state' also for oscillating phenomena and the like (e.g. the buzzing action of a clapper, which alternates between two different DOLCE-states while buzzing, namely open-circuit and closed-circuit). Hence, the right DOLCE category to conceptualize system states is the one of stative perdurants. In the following we will keep using the term 'state' following the engineering terminology, but it is to be understood as 'stative condition' in DOLCE.

[19] Note that the term 'processual role' refers to general perdurants, it is not limited to DOLCE processes. The terminology is taken from Loebe which introduces it relatively to the GFO approach [58].

[20] This relation is inspired by the one introduced in YAMATO [68]. The latter, however, is limited to YAMATO processes.

[21] Note that Borgo and Mizoguchi constrain the relation of causal contribution so that its domain and range are limited to processes. But, in [7], the authors argue that the same relation can also hold between a process and a state, with the convention that, whenever that happens, it holds between the first process and the process of achieving the state. We do not enter into the discussion of such conceptualisation. Here we take the domain and range to cover the category of perdurants.

Finally, engineers typically know what state a system should be in, therefore we assume that, given a technical artifact, some 'types' of states are selected as desired, called *goals*. Note that, typically, one selects the conditions that a state has to satisfy, that is, selects a concept and not a state-instance, since the latter would have a specific time extension. Hence, we have that

a10 $\texttt{Goal}(x) \implies \texttt{CN}(x) \wedge \forall y, t(\texttt{CL}(y, x, t) \rightarrow \texttt{STV}(y))$
"a goal is a concept that classifies stative perdurants only"

Thus, goals may correspond to expressions 'the temperature at the port B of the heat exchanger is between 80 and 110 Celsius degrees' and 'the buzzer is clapping with a frequency of at least 10kHz', which are expressions for state classifiers.

4.2. Ontological and systemic functions

In this paragraph we exploit the concepts introduced in the previous sections and propose a preliminary formalisation of ontological functions as roles. Precisely, we start by defining *systemic functions*. In doing that, we are mainly inspired by the definition presented in [7] (cfr. also Cummings' definition in [3]). Such a definition is based on the so-called *systemic view* of devices, that is, on the idea that devices are complex aggregates of components, called systems, whose behaviors combine in order to generate the behavior of the whole system. In this context, a function is seen as the contribution of the behavior of an individual component to the behavior of the system as a whole, and teleological aspects are introduced through goals imposed to the system.

d8 $\texttt{FunctionOf}_{\texttt{Sys}}(x, y) \iff (\texttt{RL}(x) \wedge \exists z, g(\texttt{System}(z) \wedge \texttt{goalOf}(g, z) \wedge \exists b, t(\texttt{CL}(b, x, t) \wedge \texttt{P}(y, z, t))$
$\forall b, t((\texttt{CL}(b, x, t) \wedge \texttt{P}(y, z, t)) \implies (\texttt{Behaviour}(b) \wedge \texttt{participatesAsDoer}(y, b, t)$
$\wedge \texttt{causalContr}(b, g)))))$
"x is a systemic function of y if and only if x is a role and there exist a system z and a goal g for z such that x is satisfied only by behaviors which have y as doer and causally contribute to achieve g, whenever y is part of z"

d9 $\texttt{Function}_{\texttt{Sys}}(x) \iff \exists y \texttt{FunctionOf}_{\texttt{Sys}}(x, y)$
"x is a systemic function if and only if it is the systemic function of some object y"

Of course, there are many different function conceptualisations in the literature and each is relevant from one engineering perspective or another. We propose to start from Definition (d8) because, differently from the others, it is ontologically clear and helps to clarify the assumptions on which other meanings rely. Note that, by assuming that a function is present whenever is played by a behaviour, definition (d9) entails that systemic functions are specifically existentially dependent on systems. If we also assume that systemic functions are external to systems (indeed, systems are neither qualities, parts, nor constituents of systemic function roles), one obtains the following:

t1 $\texttt{Function}_{\texttt{Sys}}(x) \implies \exists y(\texttt{System}(y) \wedge \texttt{founded}(x, y))$
"each systemic function is founded on some system"

In engineering practice, one often speaks about functions independently of any system, for instance in an early system design phase. For example, if one wants to address the problem of finding the set of devices that have the capability of realizing a needed transformation, the devices cannot be *a priori* associated with a certain system, since there is no system yet. Indeed, the system is a variable input of the problem. Therefore, to solve this problem, a more general concept of function is needed, as the one introduced in [46]. We call such functions *ontological functions*: they address general transformation needs perhaps without even addressing how a transformation occurs or to what entities it should apply. The informal intuition of ontological functions is that they are classified according to the ontological change they realize between the input and the output states. In this sense, they are independent of systemic functions. Yet, every systemic function is associated to one or more ontological functions, depending on the changes it classifies. It follows that the following relation between ontological functions and systemic functions holds:

a11 $\texttt{Function}_{\texttt{Ont}}(x) \implies (\exists y(\texttt{SP}(y, x) \wedge \texttt{Function}_{\texttt{Sys}}(y)) \wedge \forall y(\texttt{SP}(y, x) \wedge \neg\texttt{Function}_{\texttt{Ont}}(y) \implies \texttt{Function}_{\texttt{Sys}}(y)))$

> "any ontological function *x* is specialized by some systemic function *y* and, beyond ontological functions, it classifies systemic functions only"

For example, take a given tooling machine, say a lathe, as a system, and assume that the lathe makes use of two electrical motors, e.g., one for rotating the spindle and the other for moving the spindle horizontally. Both electrical motors perform the same ontological function of 'converting' (electrical energy into mechanical energy), but they also perform two different systemic functions specializing the ontological function in the context of the lathe: 'converting (electrical energy into mechanical energy) to rotate the spindle', and 'converting (electrical energy into mechanical energy) to translate the spindle', respectively. The intuition is that we can group systemic functions together through common characteristics, abstracting from specific systems or from their occurrences in different parts of the same system.

Such ontologically functions reflect a perdurant taxonomy based on functional grounds, meaning that a given taxonomy of perdurants corresponds to a taxonomy of functions as given in [46]. As said, we obtain the types of ontological functions using the variation of some condition before and after the realisation of the function. For example, it is common to speak of 'connect' or 'change' functions [1], that is, of functions whose underlying perdurant is such that the number of operands or, respectively, the operand type is modified during the event. Similarly, other ontological functions-types could be obtained considering variation of other properties, as done in [4] and [46], or specializing existing ontological functions to special cases, e.g. specialize 'change' to 'change electrical energy into torque'. In this paper, we do not characterize further the taxonomy of ontological functions, nor discuss the organisation of their taxonomy beyond what presented in [46].

Finally, notice that the two types of functions introduced in this paragraph correspond to (at least) two different ideas of functions used by engineers. First, there are 'general' functions that engineer use when need to, say, describe information about or collect information from different systems. For example, Collins [70], when collecting and analyzing failure experience data, speaks of 'elemental mechanical functions', which are application-independent characterisations of 'basic' functions. Analogously, Pahl et Beitz [1] speak of 'generally valid functions', and propose them as references for cataloguing design knowledge about function implementation. These types of functions could be approximated with ontological functions. In contrast, the second meaning used by engineers is system dependent. In fact, in general, when engineers are focused on a single system, they use different concepts when speaking about the system components. The terminology is varied, but typical terms are 'serial number', that is the identifier of a component-instance, 'component code', i.e. the component or assembly-type identifier, and 'functional location' or 'tag', which are identifiers that consider also the position of a component within a system. For example, Figure 1 schematises a hydraulic system containing four solenoid valves, whose tags are EPF1 to EPF4. These tags cannot refer to the valve-type, since the four valve could have the same type, neither they can refer to the valve-instances, since schema are often used to represent different system-instances. In our terminology, we could say that tags identify roles that components play in a system. Necessarily, these roles are of functional nature, for each component in an engineering system has a role in the system function, thus, tags, or, at least, the teleological content they carry, could be formalized by systemic functions.

4.3. Functional decomposition

An important feature of functions in engineering is the possibility to decompose them into sub-functions. This also allows to refine the granularity of the system description. In this paragraph, we show how such decomposition relation can be used in order to formalize the difference between ontological functions and engineering functions that we anticipated earlier.

First, observe that such decomposition cannot be reduced to a partial order relation between functions. That is, if we represent the functional decomposition of a function, say $\mathtt{f}$, into sub-functions, say $\mathtt{f}_1, \mathtt{f}_2, \ldots, \mathtt{f}_n$, as $\mathtt{decomp}(\mathtt{f}; \mathtt{f}_1, \mathtt{f}_2, \ldots, \mathtt{f}_n)$, then it does not seem possible to find a parthood relation such that $\mathtt{decomp}$ reduces the mereological sum. This is caused by, at least, the following reasons:

- Functions exists at a teleological level, therefore, any decomposition of functions must take into account the decomposition of the underlying objective substrata, that is, of the underlying behaviors and objects.

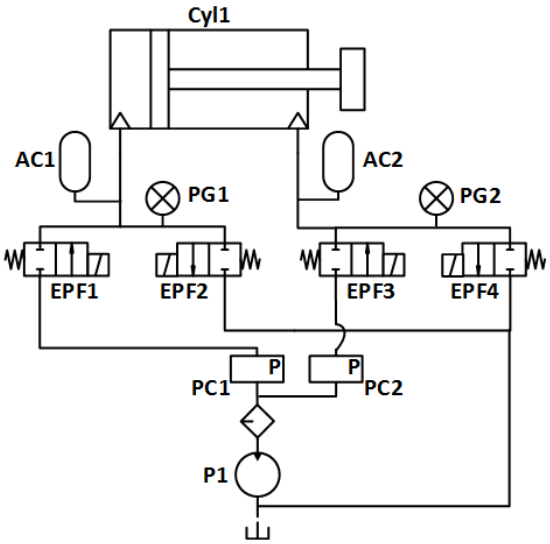

Fig. 1. The scheme of some hydraulic system, taken from [71].

- The sub-functions of a function must 'organize'[22] in order to realize the decomposed function. In particular, the composition of a mere set of functions is not unique.
- The same function can be decomposed in more ways, therefore the decomposition relation is of type many-to-many.
- Not all combinations of sub-functions are possible, due to physical and technical constraints. Moreover, among all possible combinations, engineers recognize typical ones and use them systematically.

Additionally, Vermaas has proven that attempting to model a Functional Basis style of functional decomposition[23] entails contradictions [40][24], so that there are also formal obstacles preventing the application of classical mereology to functional decompositions. We do not attempt a solution to these problems here, instead we assume that the relation `decomp` is given, and focus on engineering methods.

Inspired by the work of Kitamura et al. on 'ways of functional achievement' [6, 72], we consider engineering methods as non-agentive social objects representing the knowledge that engineers share about ways of implementing functions through functional decomposition:

a12 $\texttt{Method}_{\texttt{Eng}}(x) \implies \texttt{NASO}(x)$

Formally, such non-agentive social objects can be understood as reifications of functional decomposition relations. Precisely, we introduce roles $\texttt{mainFunction}(\cdot)$ and $\texttt{subFunction}(\cdot)$, contextualized by a decomposition, such that:

a13 $(\texttt{mainFunction}(x) \vee \texttt{subFunction}(x)) \implies (\texttt{RL}(x) \wedge \exists \texttt{m}\,(\texttt{Method}_{\texttt{Eng}}(x) \wedge \texttt{founded}(x, \texttt{m})))$
"main-functions and sub-functions are roles founded on some engineering method"

Additionally, we assume that methods are always contexts for a main-function and for a certain number of sub-functions:

[22] We borrow the term from Vermaas and Garbacz [39], in there the interested reader can find a discussion about mereology in functional decomposition, especially with respect to the Functional Basis methodology.

[23] That is, a style where functional models can be graphically represented as directed graphs with flows as edges and function as nodes. The composition, then, can be in series, between nodes that share an edge (cancelling that edge), or in parallel, between nodes that do not share edges.

[24] The counterexamples shown are based on some additional assumptions. Precisely that, first, if a function-token is part of another function-token, then the same holds for the corresponding types; and, second, that flow loops are possible.

a14 $\texttt{Method}_{\texttt{Eng}}(\texttt{m}) \iff \exists! n\, \texttt{Method}_{\texttt{Eng},\texttt{Sub}}(\texttt{m}, n)$, and n is integer
"Each method, say m, has a (unique) number, say n, of sub-functions that are contextualized by the method"

a15$_{schema}$ $\texttt{Method}_{\texttt{Eng},\texttt{Sub}}(\texttt{m}, n) \implies \exists!\texttt{main}, \texttt{sub}_1, \ldots, \texttt{sub}_\texttt{n}(\texttt{mainFunction}(\texttt{main}) \wedge \texttt{subFunction}(\texttt{sub}_1) \wedge \cdots \wedge \texttt{subFunction}(\texttt{sub}_\texttt{n}) \wedge \texttt{founded}(\texttt{main}, \texttt{m}) \wedge \texttt{founded}(\texttt{sub}_1, \texttt{m}) \wedge \cdots \wedge \texttt{founded}(\texttt{sub}_\texttt{n}, \texttt{m}))$
"for any engineering method of functional decomposition, say m, having a given number of sub-functions, say n, there exist a main-function role and n sub-function roles, which are founded on m and are uniquely determined"

Since the main-function and the n sub-functions roles of a given method are univoquely determined, we can represent the with functional symbols. In particular, we will write $\texttt{main}^\texttt{m}$, $\texttt{sub}_1^m, \ldots, \texttt{sub}_n^m$ to indicate the roles corresponding to the method m, as per Axiom (a15) (we omit the functional dependence of n on m, for ease of notation). Finally, the link between methods and decompositions is given in the following definition schema:

d10$_{schema}$ $\texttt{decomp}(\texttt{f}; \texttt{f}_1, \texttt{f}_2, \ldots, \texttt{f}_n) \iff$
$(\texttt{Function}_{\texttt{Sys}}(\texttt{f}) \wedge \texttt{Function}_{\texttt{Sys}}(\texttt{f}_1) \wedge \cdots \wedge \texttt{Function}_{\texttt{Sys}}(\texttt{f}_n) \wedge$
$\exists \texttt{m}(\texttt{Method}_{\texttt{Eng}}(\texttt{m}) \wedge \texttt{SP}(\texttt{f}, \texttt{main}^\texttt{m}) \wedge \texttt{SP}(\texttt{f}_1, \texttt{sub}_1^\texttt{m}) \wedge \ldots \wedge \texttt{SP}(\texttt{f}_\texttt{n}, \texttt{sub}_n^\texttt{m}))$
"f is decomposed in $\texttt{f}_1, \ldots, \texttt{f}_\texttt{n}$ if and only if they are all systemic functions and there is a method with corresponding main-function $\texttt{main}^\texttt{m}$ and sub-functions $\texttt{sub}_1^\texttt{m}$, ..., $\texttt{sub}_n^\texttt{m}$, which are specialized by f and $\texttt{f}_1, \ldots, \texttt{f}_\texttt{n}$, respectively"

The advantage of this approach is manyfold: it makes possible to organize the method-types within sumbsumption taxonomies, for example, 'spot welding' is a specialisation of the method 'welding', which itself is an implementation method of the function 'to join'; and to describe the properties of the methods, for instance the working principle, say Kirchhoff's law for a 'voltage divider' method. Additionally, it makes possible to clearly discuss some properties of functional decomposition, for example, if one wishes to express that all functions are decomposable:

ex1 $\texttt{Function}_{\texttt{Sys}}(x) \implies \exists y\, \texttt{mainFunction}(y) \wedge \texttt{SP}(x, y) \wedge x \neq y$

Instead, if one wishes to state that a function, say f, is not decomposable:

ex2 $\neg\exists y\, (\texttt{mainFunction}(y) \wedge \texttt{SP}(x, y) \wedge x \neq y)$

Moreover, this approach allows us to give a formal definition of engineering function and, thereby, to discuss the difference between capacities and capabilities that we anticipated in the introduction. In fact, we define engineering functions as

d11 $\texttt{Function}_{\texttt{Eng}}(x) \iff \texttt{mainFunction}(x)$
"engineering functions and main-functions coincide"

so that engineering functions are roles that systemic functions, which are defined in (d9), can play in the context of a functional decomposition. For instance, in the lathe example discussed above, it could be that the systemic functions of the two motors both are implemented through the method of, say, 'three-phase electric motor', and, therefore, play the role of engineering functions. In this case, the functional decomposition entailed by the method includes sub-functions roles for, say, 'supply electrical energy' (one per each phase), 'drive the motor', and 'output mechanical energy'.

5. Capabilities and capacities

In this section, we discuss capabilities and capacities, two notions that we briefly introduced at the end of Section 2. In particular, we will make use of the concept of ontological function to distinguish capabilities from capacities (recall that, following the approach taken in ISO 15531-31[50], the quantitative viewpoint on what a device do is characterised by capacities, while the qualitative viewpoint by capabilities).

Each technical artifact has relevant characteristics, which are part of its physical make-up and determine how the artifact interacts with certain things. For example, an individual pump could be built in such a way that it is able to pump oil with a certain flow rate. Following the quality theory of DOLCE, both the relevant characteristics of

each technical artifact and any of its faculties of being able to do something can be conceptualized as individual qualities. Consequently, we model both the flow rate (that is, a capacity), and the pumping (a capability) of the pump as individual qualities.

Whenever a capability is based on some other quality of a technical artifact, that is, when each realisation of the capability depends on some other quality, we say that it is *founded* on that quality (or qualities), and we formalize this through the relation $\texttt{founded}(\cdot,\cdot)$ defined in (d4). For example, in the case of the pump we could say that the capability of the pump to move water is founded on its flow rate capacity. Coming back to Gero and Qian's example of the water tap mentioned in Section 3, which is similar to the one of the pump, we could say that the faucet has the capability of delivering water when requested, which is founded on its flow rate capacity, which is itself founded on its diameter quality (Gero and Qian say that the flow rate is 'controlled' by the diameter [31]). Now, in this example, there is a difference between the flow rate and the diameter: the latter is intrinsic and the former is extrinsic, as we argued in Section 3. These examples suggest that we call *capacities* those relational qualities that a capability is founded on, analogously to the approach in [4]. The point being that capacities are relational qualities that provide information about the corresponding capability. Going back to the pumping example, we conclude that the flow rate capacity parametrizes, perhaps only partially, the pumping capability.

Another example: types of electronic components, such as resistors, transistors, etc., are accompanied by a datasheet that reports many technical properties of the component instances. Typical properties are, for example, the failure rate and the maximum temperature, current, or voltage that the component can operate with without problems. In our terminology, all the aforementioned properties are capacities, since they require, to be manifested, for the component to be inserted into a working electrical circuit[25]; and, additionally, they parametrize the capability of the component to, say, create a voltage drop in the case of a resistor.

Capacities themselves are founded on some intrinsic physical qualities of the bearing object. For example, the maximum operating current will depend, say, on the geometric and electrical properties of the conductor metal, such as its diameter and its resistivity, similarly the flow rate of the water tap depends on its diameter. Given these observations, we partially characterise capacities as follows:

a16 $\texttt{Capacity}(x) \implies \texttt{relationalQt}(x) \wedge \exists y(\texttt{founded}(x,y) \wedge \texttt{intrinsicQt}(y))$
"A capacity is a relational quality and is founded on some intrinsic quality"

In contrast, a characterisation of capabilities should touch upon functions as capabilities are inextricably intertwined to functional aspects. In order to do this, we first try to clarify the nature of capabilities as follows: to be able to do something is really a modal concept calling for possible worlds in which that something is actually done. But, since we want to keep our formal language as simple as possible, we reduce the modal expression to a simpler construct, analogously to the following heuristic definition:

ex3 $\exists c(\texttt{PumpingCapability}(c) \wedge \texttt{qt}(c,x)) \iff$
$\Diamond\exists e,t(\texttt{participatesAsDoer}(x,e,t) \wedge \texttt{PumpingProcess}(e))$
"An object x carries an individual capability of pumping if and only if there is a possible perdurant during which x realizes some pumping process"

Thus, we shall introduce capabilities as qualities with a modal flavour. Since the introduced capabilities are specifically dependent on their bearers, and each capability of a certain kind is unique to its bearer, we treat them as individual qualities.[26] Finally, note that Example (ex3) also seems to suggest that capabilities depend on types of functions (not just processes or behaviors, for the pumping process in (ex3) will always play a function), which are used in their definition. This suggests using the definition-founding relation introduced in Section 3 in order to capture this intuition:

d12 $\texttt{Capability}(x) \iff \texttt{relationalQt}(x) \wedge \exists y,z(\texttt{Capacity}(y) \wedge \texttt{founded}(x,y) \wedge \texttt{Function}_{\texttt{Ont}}(z) \wedge \texttt{founded}_{\texttt{Def}}(x,z))$

[25] As mentioned earlier in the paper, a clear-cut example is the voltage, which, since it is a potential, needs a fixed reference point in order to be measured.

[26] An alternative is to model them as disposition, see for instance [49], and [73] for a broader introduction to dispositions.

"x is a capability if and only if it is a relational quality founded on some capacity and is definition-founded on some ontological function"

From the definition, we have that capabilities are specifically dependent on capacities, which specify how the artifact (the bearer) can function. The intuition is that capabilities are relational qualities that associate the artifact to ontological functions, so that they must refer to those functions: in our terminology, they are definition-founded on ontological functions. Note that they are not instantiation-founded, see (d5), since an artifact could have a capability to function without actually functioning. Additionally, the function must be ontological and not systemic, because, otherwise, the bearing object would be associated *a priori* with some given system.

The characterization of capacities via (a16) and (d12) has the advantage of explaining

- the close link between functions and capabilities (capabilities are definition-founded on ontological functions).
- The asymmetry between capacities and capabilities (capabilities are founded on capacities, but not vice-versa).

The quantitative-qualitative difference between capacities and capabilities anticipated at the end of Section 2 is not yet really explained, but, notice that while a flow-rate capability takes as values positive real numbers with, say, m^3/s as unit of measure, the value space of the corresponding pumping capability is quite more complex and difficult to describe. This may be an argument against modelling capabilities as DOLCE-qualities, but it also reflects the quantitative-qualitative duality that emerges in domain experts' speech.

Finally, to recap the concepts introduced in the last paragraphs, we conclude this section with another example. Suppose that a company handling swimming pools must empty some of its pools for maintenance. This imposes a goal, say 'the pools are empty', that must be carried out through some device able to realize a 'to remove' (ontological) function, specialized to a systemic function in the context of the swimming pool system. Such a function can be realized by artifacts that have a corresponding capability. In this case, the function would probably be implemented through some fluid-emptying-method, that is, through the knowledge of the way that a recipient can be emptied, and its fluid content disposed, by pressurizing the fluid and guiding it through a path. Now, the 'pressurize' sub-function (which refers to the goal of 'having a big enough pressure gradient') could be implemented through a pumping-method, that is, referring to engineering knowledge about pumps. Then, a pump (more generally an artifact with pumping capability) could be selected for being the 'doer' of the necessary 'transform electrical energy into pressure' sub-sub-function. In this context, we could say that the pump has been selected because of its pumping-capability and that the pumping-method describes, at least, a flow rate capacity, which founds the pumping-capability, and its interaction to the other properties and entities involved in the pumping process.

6. OWL ontology

The first order logic theory developed in the previous sections can be converted to OWL language as is, with the exception of the expressions where ternary relations are used. For example, the definition of a systemic function (d8)-(d9), the instantiation-founding definition (d5), and the engineering method schema (d10) cannot be expressed in OWL. In these cases we have to weaken the axioms. For instance, in OWL we replace the definition of systemic functions with

$$\textbf{a17}_{owl} \qquad \texttt{Function}_{\texttt{Sys}} \sqsubseteq (\forall \texttt{classifies.Behaviour} \sqcap \exists \texttt{causalContr.Goal}) \sqcap \exists \texttt{founded.System})$$

where `classifies` is the inverse relation of $\texttt{CL}(\cdot,\cdot,\cdot)$. In this way the fact that systemic function are founded on systems is preserved and highlighted.

Additionally, for all the ternary temporalized relations, such as $\texttt{CL}(\cdot,\cdot,\cdot)$ and $\texttt{participateAsDoer}(\cdot,\cdot,\cdot)$, the temporal argument is removed and a homonymous binary relation is used in their place. The link between the temporalized and non-temporalized relations can be interpreted in different ways. For example, one could state that the non-temporalized relation holds if and only if the temporalized relation holds whenever one of the relation argu-

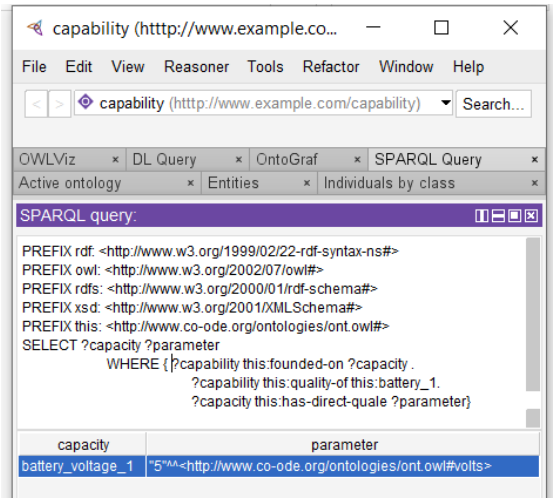

Fig. 2.

ments exists, what argument precisely depends on the relation. This is our choice for parthood (d7), classification, and participation-like relations as shown by these *meta-rules*:

d13$_{meta}$ $\mathtt{CL}(x,y) \iff ((\exists t \mathtt{PRE}(x,t)) \wedge \forall t(\mathtt{PRE}(x,t) \implies \mathtt{CL}(x,y,t)))$
"x is constantly classified by y if and only if x is classified by y whenever x exists"

d14$_{meta}$ $\mathtt{participatesAsDoer}(x,y) \iff \exists t(\mathtt{PRE}(y,t)) \wedge \forall t(\mathtt{PRE}(y,t) \implies \mathtt{participatesAsDoer}(x,y,t))$
"x constantly participates-as-doer to y if and only if x participates-as-doer to y for the whole of y duration"

One could also state that the non-temporalized relation holds if and only if the temporalized relation held true at some time adopting the following *meta-rule*:

ex4$_{meta}$ $\mathtt{participatesAsDoer}(x,y) \iff \exists t \mathtt{participatesAsDoer}(x,y,t)$

There are also other possibilities. The choice of one over the others must be planned carefully depending on the application concerns, since this kind of changes affects the intended models of the OWL ontology. For example, (d14) excludes partecipation-as-doer in, say, a chemical process only for its first part, while (ex4) allows it, but in OWL this difference is lost. Additionally, the removal of the temporal argument reduces the flexibility of the ensuing ontology. For example, one cannot track (at least not directly) dynamic aspects of roles, e.g. a component that carries out a function at a time and then it changes its function. Nor one can express that, say, there is chemical process which is driven by two different catalysts during its first and second parts.

Finally, as a further simplifying assumption, we define two binary relations that we will use as shortcuts to implement and simplify the definition schema (d10):

d15$_{meta}$ $\mathtt{mainFunctionOf}(\mathtt{f},\mathtt{m}) \iff (\mathtt{SP}(\mathtt{f},\mathtt{main}^{\mathtt{m}}) \wedge \mathtt{Function}_{\mathtt{Sys}}(\mathtt{f}))$
"f is main-function-of a method m if and only if it is a systemic function specializing the main-function role correspondig to m"

d16$_{meta}$ $\mathtt{subFunctionOf}(\mathtt{f},\mathtt{m}) \iff (\mathtt{SP}(\mathtt{f},\mathtt{sub}_i^{\mathtt{m}}) \wedge \mathtt{Function}_{\mathtt{Sys}}(\mathtt{f}))$
"f is main-function-of a method m if and only if it is a systemic function specializing any of the sub-function roles correspondig to m"

The OWL ontology, after being populated, can be used to query information about the functionality of objects. For example, one could query what method could implement a given ontological function, or what are the capacities that a given capability is founded on, and what is their value for a given object (Figure 2). Moreover, one could query what components have a capability that corresponds to a given function:

```
PREFIX : <http://www.co-ode.org/ontologies/ont.owl#>
```

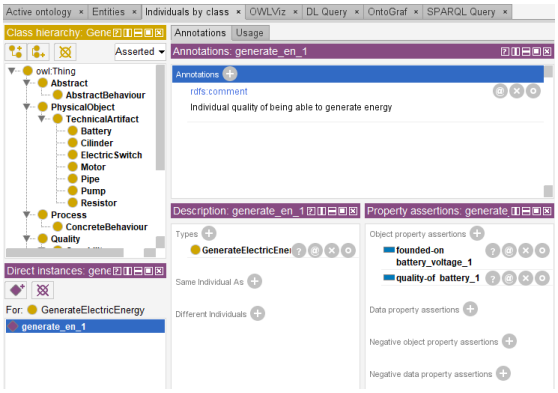

Fig. 3. A view of the ontology taxonomy in Protegé.

```
SELECT  ?component
WHERE {
    ?capability :definition-founded-on this:<the given function> .
    ?capability :quality-of ?component}
```

or what capacities of what components are involved in executing the ontological functions present in a given system:

```
PREFIX : <http://www.co-ode.org/ontologies/ont.owl#>
SELECT  ?component ?functionOntological ?capacity
WHERE {
    ?functionSystemic :function-of ?component .
    ?functionSystemic :founded-on :<the given system> .
    ?functionSystemic :specializes ?functionOntological .
    ?capability :definition-founded-on ?functionOntological .
    ?capability :quality-of ?component .
    ?capability :founded-on ?capacity .
    ?capacity :quality-of ?component}
```

The previous queries showcase the possibility of linking capabilities to functions. This is important for one could develop, based on this link, an application allowing engineers to find whether they already have available components that can satisfy a given functional requirement, e.g. in early system design.

The ontology also supports queries related to engineering methods and their relation with functions. For example, the following query takes a given system as inputs and returns all systemic functions involved in a method, with their role (main-function or sub-function) and underlying component:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX : <http://www.co-ode.org/ontologies/ont.owl#>
SELECT  ?component ?functionSystemic ?role ?method
WHERE {
    ?functionSystemic ?role ?method .
    ?method rdf:type/rdfs:subClassOf* :EngineeringMethod .
    ?functionSystemic :function-of ?component  .
    ?component :constant-part-of :<the given system>}
```

Given some ontological function, one can set a query to return the list of methods, present in the database, that can be implemented through systemic-function specialisation of the input ontological function; or explain what method-(sub)types specialize a given method, and what systemic-function-types are required by that method. This

is useful, both during design, since it gives information about how to implement a given function, and during reverse engineering, since it explains how the parts of a system cooperate to carry out a function of coarse granularity.

Even though the DOLCE ontology is primarily a first-order logic theory, OWL-adapted versions exist, such as DOLCE-lite or DOLCE-Ultralite[27]. Any of these could be used to align the ontology developed in this paper with DOLCE. In our case, we used the OWL version of DOLCE that has been recently submitted for inclusion in the ISO 21838 standard.[28] The resulting ontology, which can be found on GitHub[29], was tested using the Hermit reasoner. (For the sake of example, the ontology is populated with a few individuals).

7. Conclusion

This work contributes towards an ontological understanding of fundamental concepts used in engineering, especially functionality. In particular, we have shown how one can give ontologically-grounded definitions of capability, capacity, behavior and function using first order logic. Moreover, we have shown how one can use functional decomposition to distinguish between ontological, systemic, and engineering functions, and how ontological functions can be used to explain the difference between capabilities and capacities. Finally, we partially translated our first order theory in OWL, showcasing a preliminary serialisation of our theory in a computer-friendly formal language.

Our approach builds on a series of previous works, especially on the study of functionality carried out by engineers and researchers, in particular [1, 27, 41]; the study of resources in manufacturing, in the applied ontology literature as well as some standards [4, 50, 52]; and the top-level ontology DOLCE and its developments [10, 55]. The novelty of our work is the in-depth ontological analysis, which, exploiting a common ontological framework, clarifies the conceptualisation of functions and related concepts in a systematic way.

Acknowledgments

Francesco Compagno is funded by the company Adige Spa. This work is partially funded by the European project OntoCommons (GA 958371, www.ontocommons.eu).

References

[1] G. Pahl, K. Wallace, L. Blessing and G. Pahl (eds), *Engineering design: a systematic approach*, 3rd ed edn, Springer, London, 2007.
[2] J.E. Larsson, Diagnosis Based on Explicit Means-End Models, *Artificial Intelligence* **80**(1) (1996), 29–93.
[3] R. Cummins, Functional Analysis, *The Journal of Philosophy* **72**(20) (1975), 741–765.
[4] S. Borgo, E.M. Sanfilippo and W. Terkaj, Capabilities, Capacities, and Functionalities of Resources in Industrial Engineering, in: *CEUR Workshop Peoceedings*, Bolzano, Italy, 2021, p. 12.
[5] M.S. Erden, H. Komoto, T.J. van Beek, V. D'Amelio, E. Echavarria and T. Tomiyama, A Review of Function Modeling: Approaches and Applications, *Artificial Intelligence for Engineering Design, Analysis and Manufacturing* **22**(2) (2008), 147–169.
[6] Y. Kitamura, Y. Koji and R. Mizoguchi, An Ontological Model of Device Function: Industrial Deployment and Lessons Learned, *Applied Ontology* **1**(3–4) (2006), 237–262.
[7] R. Mizoguchi, Y. Kitamura and S. Borgo, A Unifying Definition for Artifact and Biological Functions, *Applied Ontology* **11**(2) (2016), 129–154.
[8] S. Shapiro and T. Kouri Kissel, Classical Logic, in: *The Stanford Encyclopedia of Philosophy*, Spring 2021 edn, E.N. Zalta, ed., Metaphysics Research Lab, Stanford University, 2021.
[9] W3C Recommendation, OWL 2 Web Ontology Language Quick Reference Guide, Technical Report, 2012. http://www.w3.org/TR/owl2-quick-reference/.
[10] C. Masolo, S. Borgo, A. Gangemi, N. Guarino and A. Oltramari, WonderWeb Deliverable D18, IST Project 2001-33052 WonderWeb: Ontology Infrastructure for the Semantic Web, 2003.

[27] Both openly available, e.g., at http://www.loa.istc.cnr.it/dolce/overview.html.
[28] https://www.iso.org/standard/71954.html.
[29] https://github.com/kataph/function-method-ontology.git.

[11] S. Borgo, R. Ferrario, A. Gangemi, N. Guarino, C. Masolo, D. Porello, E.M. Sanfilippo and L. Vieu, DOLCE: A Descriptive Ontology for Linguistic and Cognitive Engineering, *Applied Ontology, to appear* (2022).
[12] M. Artiga, New perspectives on artifactual and biological functions, *Appl. Ontology* **11**(2) (2016), 89–102. doi:10.3233/AO-160166.
[13] J. De Kleer, How circuits work, *Artificial Intelligence* **24**(1–3) (1984), 205–280.
[14] J.D. Kleer and J.S. Brown, A Qualitative Physics Based on Confluences (1984), 78.
[15] K. Roth, *Konstruieren mit Konstruktionskatalogen*, Springer Berlin Heidelberg, Berlin, Heidelberg, 2000.
[16] B. Chandrasekaran, A.K. Goel and Y. Iwasaki, Functional Representation as Design Rationale, *Computer* **26**(1) (1993), 48–56.
[17] A.M. Keuneke, Device representation-the significance of functional knowledge, *IEEE Expert* **6**(2) (1991), 22–25.
[18] Y. Kitamura, T. Sano, K. Namba and R. Mizoguchi, A Functional Concept Ontology and Its Application to Automatic Identification of Functional Structures, *Advanced Engineering Informatics* **16**(2) (2002), 145–163.
[19] J. Hirtz, R.B. Stone, D.A. McAdams, S. Szykman and K.L. Wood, A functional basis for engineering design: Reconciling and evolving previous efforts, *Research in Engineering Design* **13**(2) (2002), 65–82.
[20] R.B. Stone and K.L. Wood, Development of a Functional Basis for Design, *Journal of Mechanical Design* **122**(4) (2000), 359–370.
[21] T. Kurtoglu and I.Y. Tumer, A Graph-Based Fault Identification and Propagation Framework for Functional Design of Complex Systems, *Journal of Mechanical Design* **130**(5) (2008), 051401.
[22] A.S. Gill and C. Sen, Logic Rules for Automated Synthesis of Function Models Using Evolutionary Algorithms, in: *Volume 2: 41st Computers and Information in Engineering Conference (CIE)*, American Society of Mechanical Engineers, Virtual, Online, 2021.
[23] T. Kurtoglu, A. Swantner and M.I. Campbell, Automating the conceptual design process: "From black box to component selection", *Artificial Intelligence for Engineering Design, Analysis and Manufacturing* **24**(1) (2010), 49–62.
[24] Y. Kitamura and R. Mizoguchi, Ontology-Based Functional-Knowledge Modeling Methodology and Its Deployment, in: *Engineering Knowledge in the Age of the Semantic Web*, Vol. 3257, E. Motta, N.R. Shadbolt, A. Stutt and N. Gibbins, eds, Springer Berlin Heidelberg, Berlin, Heidelberg, 2004, pp. 99–115.
[25] B. Chandrasekaran and J.R. Josephson, Function in Device Representation, *Engineering with Computers* **16**(3–4) (2000), 162–177.
[26] Y. Umeda, H. Takeda, T. Tomiyama and H. Yoshikawa, Function, Behaviour, and Structure, *Applications of artificial intelligence in engineering V* **1**(Design) (1990), 177–193.
[27] M. Sasajima, Y. Kitamura, M. Ikeda and R. Mizoguchi, FBRL: A Function and Behavior Representation Language, in: *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, Vol. 2, Montreal, Quebec, Canada, 1995.
[28] M. Sasajima, Y. Kitamurat, M. Ikeda, S. Yoshikawa, A. Endou and R. Mizoguchi, An Investigation on Domain Ontology to Represent Functional Models, *Proceedings of Eighth International Workshop on Qualitative Reasoning about Physical Systems (QR 94)* (1994), 10.
[29] V. Sembugamoorthy and B. Chandrasekaran, Functional representation of devices and compilation of diagnostic problem-solving systems, *Experience, memory and Reasoning* (1986), 47–73.
[30] A.K. Goel and S.R. Bhatta, Use of Design Patterns in Analogy-Based Design, *Advanced Engineering Informatics* **18**(2) (2004), 85–94.
[31] L. Qian and J.S. Gero, Function–Behavior–Structure Paths and Their Role in Analogy-Based Design, *Artificial Intelligence for Engineering Design, Analysis and Manufacturing* **10**(4) (1996), 289–312.
[32] Y. Kitamura and R. Mizoguchi, Meta-Functions of Artifacts, in: *Proc. of The Thirteenth International Workshop on Qualitative Reasoning (QR-99)*, Loch Awe, Scotland, 1999, pp. 136–145.
[33] S.-C. Yang, L. Patil and D. Dutta, Function Semantic Representation (FSR): A Rule-Based Ontology for Product Functions, *Journal of Computing and Information Science in Engineering* **10**(3) (2010), 031001.
[34] S. Borgo, M. Carrara, P. Garbacz and P.E. Vermaas, A Formal Ontological Perspective on the Behaviors and Functions of Technical Artifacts, *Artificial Intelligence for Engineering Design, Analysis and Manufacturing* **23**(1) (2009), 3–21.
[35] S. Borgo, M. Carrara, P. Garbacz and P. Vermaas, Towards the Ontological Representation of Functional Basis in Dolce (2009), 13.
[36] S. Borgo, M. Carrara, P. Garbacz and P.E. Vermaas, A Formalization of Functions as Operations on Flows, *Journal of Computing and Information Science in Engineering* **11**(3) (2011), 031007.
[37] P. Garbacz, S. Borgo, M. Carrara and P.E. Vermaas, Two Ontology-Driven Formalisations of Functions and Their Comparison, *Journal of Engineering Design* **22**(11–12) (2011), 733–764.
[38] Y. Kitamura, S. Segawa, M. Sasajima, S. Tarumi and R. Mizoguchi, Deep Semantic Mapping between Functional Taxonomies for Interoperable Semantic Search, in: *The Semantic Web*, Vol. 5367, J. Domingue and C. Anutariya, eds, Springer Berlin Heidelberg, Berlin, Heidelberg, 2008, pp. 137–151.
[39] P. Vermaas and P. Garbacz, Functional Decomposition and Mereology in Engineering, in: *Philosophy of Technology and Engineering Sciences*, Elsevier, 2009, pp. 235–271.
[40] P.E. Vermaas, On the Formal Impossibility of Analysing Subfunctions as Parts of Functions in Design Methodology, *Research in Engineering Design* **24**(1) (2013), 19–32.
[41] R. Mizoguchi, Y. Kitamura and S. Borgo, Towards A Unified Definition of Function, *Frontiers in Artificial Intelligence and Applications* (2012), 15.
[42] Y. Kitamura and R. Mizoguchi, Characterizing Functions Based on Phase- and Evolution-Oriented Models, *Applied Ontology* **8**(2) (2013), 73–94.
[43] R. Mizoguchi, Y. Kitamura and The Hegeler Institute, A Functional Ontology of Artifacts:, *Monist* **92**(3) (2009), 387–402.
[44] H. Herre, B. Heller, P. Burek, R. Hoehndorf, F. Loebe and H. Michalek, General Formal Ontology (GFO) - A Foundational Ontology Integrating Objects and Processes [Version 1.0], Technical Report, 8, Institute of Medical Informatics, Statistics and Epidemiology (IMISE), 2006.

[45] J.W. Klüwer, F. Martin-Recuerda, A. Waaler, D. Lupp, M.M. Brandt, S. Grimm, A. Koleva, M. Khan, L. Hella and N. Sandsmark, ISO 15926-14:2020(E), Working Draft, READI, 2020.
[46] S. Borgo, A. Cesta, A. Orlandini and A. Umbrico, Knowledge-Based Adaptive Agents for Manufacturing Domains, *Engineering with Computers* **35**(3) (2019), 755–779.
[47] P. Vermaas, D. Eck and P. Kroes, The Conceptual Elusiveness of Engineering Functions:, *Philosophy & Technology* **26** (2012).
[48] 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* **30**(2) (2019), 959–978.
[49] A. Sarkar and D. Šormaz, Ontology Model for Process Level Capabilities of Manufacturing Resources, *Procedia Manufacturing* **39** (2019), 1889–1898.
[50] ISO, ISO 15531-31: Industrial Automation Systems and Integration - Industrial Manufacturing Management Data - Part 31: Resource Information Model, Industrial Manufacturing Management Data, 2004.
[51] L. Solano, P. Rosado and F. Romero, Knowledge Representation for Product and Processes Development Planning in Collaborative Environments, *International Journal of Computer Integrated Manufacturing* **27**(8) (2014), 787–801.
[52] E.M. Sanfilippo, W. Terkaj and S. Borgo, Resources in Manufacturing, in: *FOMI*, 2015, p. 12.
[53] K. Campbell, *Abstract Particulars*, Basil Blackwell, Oxford, 1990.
[54] C.M. Fonseca, D. Porello, G. Guizzardi, J.P.A. Almeida and N. Guarino, Relations in Ontology-Driven Conceptual Modeling, in: *Conceptual Modeling*, Vol. 11788, A.H.F. Laender, B. Pernici, E.-P. Lim and J.P.M. de Oliveira, eds, Springer International Publishing, Cham, 2019, pp. 28–42.
[55] C. Masolo, L. Vieu, E. Bottazzi, C. Catenacci, R. Ferrario, A. Gangemi and N. Guarino, Social Roles and Their Descriptions, in: *Principles of Knowledge Representation and Reasoning: Proceedings of the Ninth International Conference (KR2004)*, Whistler, Canada, 2004, p. 11.
[56] N. Guarino and C.A. Welty, An Overview of OntoClean, in: *Handbook on Ontologies*, S. Staab and R. Studer, eds, Springer Berlin Heidelberg, Berlin, Heidelberg, 2009, pp. 201–220.
[57] N. Guarino and C. Welty†, A Formal Ontology of Properties, in: *Knowledge Engineering and Knowledge Management Methods, Models, and Tools*, Vol. 1937, G. Goos, J. Hartmanis, J. van Leeuwen, R. Dieng and O. Corby, eds, Springer Berlin Heidelberg, Berlin, Heidelberg, 2000, pp. 97–112.
[58] F. Loebe, Abstract vs. Social Roles – Towards a General Theoretical Account of Roles, *Applied Ontology* **2** (2007), 127–158.
[59] S. Borgo, M. Franssen, P. Garbacz, Y. Kitamura, R. Mizoguchi and P.E. Vermaas, Technical Artifacts: An Integrated Perspective, *Applied Ontology* (2017), 18.
[60] R. Mizoguchi and S. Borgo, The Role of the Systemic View in Foundational Ontologies, in: *The Joint Ontology Workshops (JOWO)*, 2021, p. 11.
[61] R. Mizoguchi and F. Toyoshima, YAMATO: Yet Another More Advanced Top-Level Ontology with Analysis of Five Examples of Change, *Applied Ontology* (2017).
[62] A.K. Goel, S. Rugaber and S. Vattam, Structure, Behavior, and Function of Complex Systems: The Structure, Behavior, and Function Modeling Language, *Artificial Intelligence for Engineering Design, Analysis and Manufacturing* **23**(1) (2009), 23–35.
[63] P.E. Vermaas and K. Dorst, On the Conceptual Framework of John Gero's FBS-model and the Prescriptive Aims of Design Methodology, *Design Studies* **28**(2) (2007), 133–157.
[64] J.S. Gero, Categorising Technological Knowledge From a Design Methodological Perspective, in: *International Conference on Technological Knowledge: Philosophical Reflections*, Boxmeer, the Netherlands, 2002.
[65] M. Zhao, Y. Chen, L. Chen and Y. Xie, A State–Behavior–Function Model for Functional Modeling of Multi-State Systems, *Proceedings of the Institution of Mechanical Engineers, Part C: Journal of Mechanical Engineering Science* **233**(7) (2019), 2302–2317.
[66] Y. Kitamura and R. Mizoguchi, Ontology-Based Systematization of Functional Knowledge, *Journal of Engineering Design* **15**(4) (2004), 327–351.
[67] J.S. Gero and U. Kannengiesser, The Situated Function–Behaviour–Structure Framework, *Design Studies* **25**(4) (2004), 373–391.
[68] R. Mizoguchi, YAMATO : Yet Another More Advanced Top-level Ontology. http://www.hozo.jp/onto_library/upperOnto.htm.
[69] S. Borgo and R. Mizoguchi, A First-order Formalization of Event, Object, Process and Role in YAMATO, *Frontiers in Artificial Intelligence and Applications* (2014), 15.
[70] J.A. Collins, B.T. Hagan and H.M. Bratt, The Failure-Experience Matrix—A Useful Design Tool, *Journal of Engineering for Industry* **98**(3) (1976), 1074–1079.
[71] J.H. Lugo Calles, V. Ramadoss, M. Zoppi, G. Cannata and R. Molfino, *Modeling of a Cable-Based Revolute Joint Using Biphasic Media Variable Stiffness Actuation*, 2019. doi:10.1109/IRC.2019.00125.
[72] Y. Kitamura and R. Mizoguchi, Ontology-Based Description of Functional Design Knowledge and Its Use in a Functional Way Server, *Expert Systems with Applications* **24**(2) (2003), 153–166.
[73] W. Malzkorn, Defining disposition concepts: A brief history of the problem, *Studies in History and Philosophy of Science Part A* **32**(2) (2001), 335–353. doi:https://doi.org/10.1016/S0039-3681(00)00042-X. https://www.sciencedirect.com/science/article/pii/S003936810000042X.