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

Francesco Compagno [a,*] and Stefano Borgo [a]
[a] *Laboratory 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 approach yet. This work develops some crucial 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 the implementation methods to realize them, how one can differentiate between capabilities and capacities of a product, and how these are related to engineering functions.

Keywords: Ontology, Function, Behaviour, Capability, DOLCE

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*, *behaviour*, 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 functions with coarser granularity, 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.

The objective of our work is to answer the following research questions which are motivated by the literature review in Section 2. They all specialize the goal of clarifying the main concepts of this paper, i.e., functions, behaviours, capabilities, and capacities:

RQ1 How can the wide range of meanings carried by the term 'function' be ontologically differentiated and explained.
RQ2 How can the functional decomposition of a system be explained and formalized?
RQ3 What are capabilities and capacities and what is the difference between them (if any)?

In particular, 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 contextualized 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 Sections 4 and 5 for the discussion of capabilities, capacities, behaviours, functions, and, very briefly, *affordances*. 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 and evaluate 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.

[1] For example, the use of gearboxes made in a certain way in order to increase or reduce angular velocity, and a brushless electric motor as a way to convert electric energy 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, on which the implementation relies.

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

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, toMaintain, 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 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 that some authors classified the multiple uses of behaviour 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 behaviour 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 behaviour*, for example, 'a motor converts electrical energy to torque' and 'the signal intensity is increased by the amplifier' are examples of B1 behaviour, while 'the temperature in the room is increased by the oven' is not a B1 behaviour, neither is 'the turbine is rotating'). For another view, in [25], Chandrasekaran and Josephson divide six types of behaviours mainly depending on their time duration (instant, interval, or unspecified) and the values of the state variables during that time.

The link between behaviour and function 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 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, again roughly speaking, a function is used as a label for a subset of behaviours. Finally, they mostly recognize a dependence of functions on behaviours, and of behaviours on the structure. There are, of course, key differences. For example, (functional) behaviour in Chandrasekaran's FR scheme refers to a causal process while behaviour 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 behaviours, while in Sasajima's approach a decomposition of functions into behaviours 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

[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'.

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.

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 out 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, 7, 25], or the classification of behaviours as occurrents[8] [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]. The last version of YAMATO classifies functions as roles and behaviours as processes[9] caused by an unintentional actor. 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[10]. GFO too states that functions can be realized by other entities, but makes no mention of dispositions and, instead, calls functions 'intentional entities' [41]. The last development of ISO 15926 [42] adopts BFO view of functions as dispositions realized in particular processes.

In truth, despite the relatively small space that functions occupy in these upper-level ontologies, their authors have been discussing functionality more in length in various series of research papers. For example, DOLCE's authors suggest seeing functions as particular types of events, while behaviours are 'qualifications of the participation relation' in that they explain the specific way a device participates in a specific event [4, 36, 37]. On the other hand, YAMATO's authors deepen their idea of functional ontology in a series of additional papers, among them [6, 43, 44], in which they confirm and expand their claim that functions are roles played by behaviours, which are themselves a certain class of process (notice that YAMATO and FBRL were developed by the same research group).

The view of GFO's authors has some similarities with the one of FBRL, in that functions despite not being roles themselves, are linked to 'functional items', which are roles played by the entities realizing the function. Functions themselves are complex entities, composed of a functional item, a label to identify them, and descriptions of the initial and final states corresponding to the function execution, called 'requirements' and 'goals', respectively [45, 46]. The typical example is the function 'to transport oxygen', which is linked to the functional item 'oxygen

[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]An ontological category similar to DOLCE's perdurants, see Section 3.

[9]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.

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

transporter' which can be played by a red blood cell. Then, a requirement for the realisation of the function is the presence of oxygen in the red blood cell environment and the goal is the presence of oxygen in the body cells that need it for cellular respiration. Another similarity with the FBRL is the development of teleological relations between functions similar to meta-functions. For example, [45] mentions the relations 'support', 'enable', and 'prevent', depending on the fact that the goal of the first function fulfills partially, completely, or is incompatible with the requirement of the second function (cfr. with [32], which also uses the term 'enable' and 'prevent', among others).

Coming to BFO, the view of functions as special dispositions was expanded and defended by its authors [47, 48], as well as criticized by some philosophers, such as Jansen and Röhl [49, 50]. A full description of this debate is beyond the scope of this paper, we just mention that while BFO maintains functions as a subclass of dispositions, Jansen and Röhl argue that functions and dispositions are disjoint, though related, categories. More specifically, in BFO functions are defined as follows [48]:

> "f is a disposition & f exists in virtue of its bearer's physical make-up & this physical make-up is something that this bearer possesses because it came into being, either through evolution (in the case of natural biological entities) or through intentional design (in the case of artifacts), in order to realize processes of a certain sort."

This entails, among other things, that, if a function ceases to exist, then the physical make-up of its bearer changes as well (this is true for all dispositions in BFO). On the other hand, Jansen and Röhl argue that functions are "externally grounded". Moreover, a major point of disagreement concerns malfunctions, in that Jansen argues that malfunctioning entities are entities that have a function, but lack the means to realize it, for example because they have lost the required dispositions; while Smith and colleagues maintain that when an entity malfunctions it loses its function, partially or completely, and is recognized as an entity of its kind only because of its history, if at all (e.g., a cancerous lung is not a lung). Notice that in both of these positions functions and roles are assumed to be disjoint subclasses of realizable entities, on the ground that roles are accidental for their players, while functions are essential for their players.

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[11].

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 [52]. Still, we maintain that formalisation is needed for interoperability and 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 briefly mention other engineering concepts whose ontological status is quite complex: capabilities, capacities, and affordances. Capacities and, especially, capabilities are used in resource modelling [53–56]. In addition, terminological problems are present also for these two concepts, which are rarely formalized [4, 57]. When distinguished, capabilities and capacities are separated along a qualitative-quantitative axis, for example ISO 15531-31[55] 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 standard advises against reducing capacities as characteristics of capabilities and forbids the opposite (in contradiction to [56], 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.

[11] An initial taxonomy of an ontology of functions in this sense, which is resumed and extended in this paper, was proposed by Borgo et al. in [4] and [51].

Affordances too are a source of conceptual confusion: they were originally introduced by [58], and popularized in engineering design by Maier and Fadel in a series of works ([59, 60], among others) as a paradigm-shift from a function-based design, towards an interaction-based design, characterized by the focus on all the possible ways a user can interact with a product, included those unintended by the designer and not strictly relevant for the product function. Briefly, affordances are "what it [the environment] offers the animal, what it provides or furnishes, either for good or ill" [58]. That is, they are potential behaviours that the environment, or a part thereof, allows an agent to do. In engineering, they are often defined as the set of all the behaviours that an artifact enables a user to do [61], but this is far from being a universally shared definition. The original example of Gibson was that a flat surface affords footing to an animal, while other examples could be a pool full of water, which affords a person to swim, or a briefcase, which, affords a person to be grabbed; so that one could say that 'swimmability' and 'grabbability' are affordances of the pool and the briefcase, respectively. Unsurprisingly, an unsettled debate exists in the disciplines that make use of the concept of affordances, about their meaning [61], up to the point that some authors called for their abandonment [62]. In this paper, we do not carry out an ontological analysis of affordances. Instead, we mention them as an important engineering concept that is related to functions [61] and, at the end of Section 4, we point out a few similarities affordances share with capabilities.

3. An (enriched) subset of the DOLCE ontology

As stated in the introduction, we will use DOLCE as reference ontology. The reasons we chose DOLCE are, at least, the following:

- DOLCE is a well-known upper ontology that has been tested in many applications.
- DOLCE has been stable since its foundation, differently than many other upper-ontology, which have been reworked extensively.
- DOLCE is a descriptive ontology that "aims at capturing the ontological categories underlying natural language and human commonsense" [10]. For this reason, theories formulated using DOLCE tend to be easy(er) to use and understand.
- Finally, we agree with Jansen and Röhl [49, 50] about the external grounding of functions and the relation of functions with malfunctions, as will become clear from the following sections. Therefore, we could not use ontologies, such as BFO, which make incompatible ontological commitments. In particular, malfunctions are common and important events to be modeled in industrial use cases, and, using Jansen's words [50] "malfunction ascriptions imply, in fact, ascriptions of functions plus the denial of the matching dispositions". This is an argument in favour of rejecting the subsumption of functions within dispositions in engineering settings, at least from a descriptivist point of view.

In this section we introduce the fragment of the DOLCE ontology [10, 11] that is needed in this paper[12]. In particular, we cover the following classes which will be needed to model functions and related concepts: *qualities*, *perdurants*, and *roles*. We anticipate to the reader that we will argue in favor of subsuming capabilities and capacities into qualities, behaviours into perdurants, and (a certain kind of) functions into roles. The links among these concepts will be, roughly, as follows: role-functions classify behaviours, capabilities require functions in order to be defined, capacities and capabilities are tightly coupled concepts.

DOLCE qualities were inspired by the trope theory [63] but differ from tropes in important ways. 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-quality-quale (or carrier-individual property-value). In DOLCE, qualities form the class $\mathtt{Q}$ (whose predicate is written $\mathtt{Q}(\cdot)$) and the

[12] See Figure 1 for a complete taxonomy

inherence relation is 'quality-of' (written $\mathtt{qt}(\cdot,\cdot)$). A temporal quale relation associates a quality with some value which, as said, may be different at different times, and is written $\mathtt{ql}(\cdot,\cdot,\cdot)$.

For our purposes in this paper, we introduce a distinction between *intrinsic* and *relational* (or extrinsic[13]) qualities, which is not part of DOLCE but is well discussed in the literature [64]. Standard examples of intrinsic qualities are mass, length, and shape; of relational qualities are weight (which is a comparative property), the personal record for a marathon (which is relative to the definition of marathon), and the distance of an object from another object. Relational qualities are qualities of an object *per se* that depend on other things like another object, the context or 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 of nor measured without considering the Earth, so we say that it is a relational quality of the Moon (similarly, for the distance of the Earth from the Moon). In contrast, if a certain brick has a given tensile strength value, this fact does not depend on other entities, so that tensile strength is an example of intrinsic quality; similarly, other mechanical or chemical properties, for example ductility or the structure of the atomic lattice in crystals, can be considered intrinsic. 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.[14] It follows that the characterisation of relational qualities is a complex matter that goes beyond the scope of this paper.[15] Therefore, we just introduce relational and intrinsic qualities as complementary primitive subclasses of DOLCE-qualities:

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

Turning now to perdurants ($\mathtt{PD}(\cdot)$), in DOLCE they are entities that are only partially present at any time they are present.[16] 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[17] are called *stative* ($\mathtt{STV}(\cdot)$), 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

[13] In truth, 'extrinsic' and 'relational' can be used with different meanings. Due to the limited scope of our paper, we don't make such a distinction.

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

[15] The interested reader can find a proposal on relational and intrinsic qualities in [65].

[16] Ordinary physical objects such as cars, trees, rocks, etc. are considered fully present at every time in which they exist.

[17] More precisely, instances of a cumulative type perdurant.

of a sitting action are still sitting actions. Stative homeomeric perdurants are called *states* ($\mathtt{ST}(\cdot)$), while the ones that are stative but have parts of different type are called *processes*($\mathtt{PRO}(\cdot)$). 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 (or buzzer): 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 $\mathtt{PC}(\cdot,\cdot,\cdot)$: in the previous example a participant of the buzzing is the clapper.

Coming to roles, they were not covered in DOLCE originally. They have been introduced later as an extension [66], and are now part of the expanded taxonomy [11]. Roles are antirigid and dependent (aka founded) classes [66–68]. 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 [69]. The classification depends on whether the context is a relation, a process, or a social object. In DOLCE roles ($\mathtt{RL}(\cdot)$) are reified and considered as concepts ($\mathtt{CN}(\cdot)$), which themselves are a class subsumed by the class of non-agentive social objects ($\mathtt{NASO}(\cdot)$):

a2 $\mathtt{RL}(x) \implies \mathtt{CN}(x)$
"a role is a concept"

a3 $\mathtt{CN}(x) \implies \mathtt{NASO}(x)$
"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 $\mathtt{CL}(\cdot,\cdot,\cdot)$ ('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. [66]. In the formalisation we use some other relations taken from DOLCE, namely: constitution, $\mathtt{K}(\cdot,\cdot,\cdot)$, which is the relation holding between some amount of matter and an object when the latter is made of the first; $\mathtt{O}(\cdot,\cdot,\cdot)$, which is the standard overlap relation (over time); and $\mathtt{PRE}(\cdot,\cdot)$, which is the relation "being present (exist) at time".

d2 $\mathtt{externalTo}(x,y) \iff \neg(\mathtt{qt}(x,y) \vee \exists t(\mathtt{K}(x,y,t)) \vee \exists t(\mathtt{O}(x,y,t)))$
"x is external to y if and only if x is neither a quality of y, nor one of y's constituents[18](at any time), nor x and y have parts in common[19] (at any time)"

d3 $\mathtt{dependsOn}(x,y) \iff (\exists t(\mathtt{PRE}(x,t)) \wedge \forall t(\mathtt{PRE}(x,t) \implies \mathtt{PRE}(y,t)))$
"x existentially depends on y if and only if x exists at some time and at any time when x exists so does y"[20]

d4 $\mathtt{founded}(x,y) \iff (\mathtt{dependsOn}(x,y) \wedge \mathtt{externalTo}(x,y))$
"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 $\mathtt{founded}_{\mathtt{Inst}}(x,y) \iff (\mathtt{CN}(x) \wedge \exists z,t(\mathtt{CL}(z,x,t)) \wedge \forall z,t(\mathtt{CL}(z,x,t) \implies \exists w(\mathtt{founded}(z,w) \wedge \mathtt{CL}(w,y,t))))$
"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 $\mathtt{RL}(x) \implies \exists y\, \mathtt{founded}_{\mathtt{Inst}}(x,y)$
"if x is a role, then there is a y on which it is instantiation-founded"

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

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

[20] 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.

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 sick people, then the doctor-role is definition-founded on the sick-person-role. Note that instantiation-founding and definition-founding need not coincide. A person is a doctor even though, at a certain time, she not treating any sick person, so that the doctor-role is not instantiation-founded on someone being in a sick state.

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. Modelling behaviours 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 and maintenance, 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 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*[21] to mean any physical object (see (a5)) that comes into being through some intentional technical process, such as, e.g., cars, planes, tooling 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. [71]). 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"

[21] See [70] for a more in-depth discussion of technical artifacts.

4.1. Behaviours

Engineers create an artifact to realize a certain interaction between the artifact itself and elements of the environment. The behaviour of the artifact is the way in which it participates in the interaction (e.g. '[the car] rattled when it hit the curve' [25]). In DOLCE one models the happening of the interaction as a perdurant. How to ontologically understand behaviour is more tricky. In the literature, the term behaviour 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 behaviour 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' [44]. In YAMATO [72] 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 behaviour. There are, in fact, at least four axes along which different meanings of behaviour can vary in the literature. First, there is what we call the occurrence-property dichotomy:

- behaviour can be something that happens in time 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 provided by Goel [73], Chandrasekaran [25], Umeda [26], and YAMATO authors' [44].
- behaviour can be a property, that is, something inhering into the behaving entity. Examples are provided by Borgo et al. in terms of qualities [34], Vermaas or Gero and colleagues in terms of attributes or dispositions [74], [75].

Then, there is the token-type distinction: behaviour can be a class or a concept, as for Mizoguchi in [44]. 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 behaviour of an artifact may refer only to characteristics of the artifact itself, or it may need to refer to external entities. For example, 'the electric switch can alternate between open and closed states' is an internal behavioural 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 behaviour with the internal meaning, e.g. Zhao et al. in [76], others with the external one, as Kitamura et al. in [77].

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 [78] 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-centric vs. environment-centric [25], but we do not discuss them.

We take behaviours to form a subclass of DOLCE perdurants. This class is partially characterized by the fact that one can identify two 'processual'[22] roles in the sense of [69]: 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 behaviours that can be described by transitive verbs. In fact, all device behaviours that can be described as operations on flows are of this kind. Such behaviours, whose class we characterize with predicate `Behaviour`$(\cdot)$, are external behaviours, since the flow is an entity external to the behaving artifact (the doer). We conceptualize the two processual roles through two specialisations of the

[22] 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 [69].

participation relation: $\texttt{participatesAsDoer}(\cdot,\cdot,\cdot)$, to indicate participation in a process with the role of an agent or agent-like doer, and $\texttt{participatesAsFlow}(\cdot,\cdot,\cdot)$, for the role of flow:

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

a8 $\texttt{participatesAsFlow}(x,y,t) \implies \texttt{Behaviour}(y) \wedge \texttt{PC}(x,y,t)$
"if x is a flow in y at time t then y is a behaviour, and x participates in y during that time"[23]

a9 $\texttt{Behaviour}(x) \implies \exists y,z,t(\texttt{participatesAsDoer}(y,x,t) \wedge \texttt{participatesAsFlow}(z,x,t) \wedge y \neq z)$
"x is a behaviour only if there are at least a doer and a flow that participate in x"

From (a9) it follows that behaviors are perdurants. Additionally, we assume that behaviours can be combined to give causal explanations of complex perdurants (e.g. the beam was cut, therefore it fell to the floor), and formalize this with a binary relation called *causal contribution*[24], which we assume holds more in general between perdurants:

a10 $\texttt{causalContr}(x,y) \implies \texttt{PD}(x) \wedge \texttt{PD}(y)$

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 [80][25].

In any case, whatever the definition of behaviour one makes use of, one must take into account the fact that behaviours are perdurants seen from the point of view of the device. This is the main reason behind the approaches that define behaviours as relational qualities of devices [34]. In this paper we try to model the participating device point of view stating that any behaviour must give raise to corresponding processual roles (a9). A comparison of the advantages and drawbacks of these two modelling choices has not been carried out yet. Nonetheless, we highlight that the two approaches are both compatible with DOLCE and could coexist (perhaps at the cost of some redundancy).

Having conceptualized behaviours as perdurants, we spend a few words about the notion of state of an engineering system. In DOLCE states are, as mentioned in Section 3, 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. To avoid a terminological clash, in the following we will use the term 'state' following the usual engineering terminology, but formally it is understood as a 'stative condition' in DOLCE. Thus, we will use the stative predicate $\texttt{STV}(\cdot)$ in formulas.

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

a11 $\texttt{Goal}(x) \implies \texttt{CN}(x) \wedge \forall y,t(\texttt{CL}(y,x,t) \rightarrow \texttt{STV}(y))$
"a goal is a concept that classifies stative perdurants only"

[23] We do not commit to a classification of flows into some categories, since we want our theory to be flexible enough to consider physical objects, substances, and qualities (cfr [36]), but also other entities such as events or information.

[24] This relation is inspired by the one introduced in YAMATO [79]. The latter, however, is limited to YAMATO processes.

[25] 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.

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. Systemic functions and ontological functions

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]), from which we also take the concept name. 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 behaviours combine in order to generate the behaviour of the whole system. In this context, a function is seen as the contribution of the behaviour of an individual component to the behaviour of the system as a whole, and teleological aspects are introduced through goals imposed to the system.

d8 $\texttt{FunctionOf}_{\texttt{Sys}}(x,y) \iff (\texttt{RL}(x) \wedge \exists z,g,b,t(\texttt{System}(z) \wedge \texttt{goalOf}(g,z) \wedge \texttt{CL}(b,x,t) \wedge \texttt{P}(y,z,t)$
$\wedge \forall b',t'(\texttt{CL}(b',x,t') \implies (\texttt{Behaviour}(b') \wedge \texttt{participatesAsDoer}(y,b',t')$
$\wedge \texttt{causalContr}(b',g)))))$
"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 behaviours which have y as doer and causally contribute to achieve g, whenever y is part of z"[26]

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, observing Definitions (d8) and (d9), it is natural to assume that systemic functions are definition-founded on systems, a fact that we introduce with the following axiom:

a12 $\texttt{Function}_{\texttt{Sys}}(x) \implies \exists y(\texttt{System}(y) \wedge \texttt{founded}_{\texttt{Def}}(x,y))$
"each systemic function is definition-founded on some system"

Ontological functions. In engineering practice, one may speak 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 [51]. We call such functions *ontological functions* as they are relative only to the ontological system one is using. These functions 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 with one or more ontological functions, depending on the changes it classifies.

More precisely, our reasoning is that a complete description of perdurants (occurrences) is extremely complicated to obtain. For instance, walking may be considered a simple action, but, in order to describe such a process, one needs to "project" the walking onto some limited viewpoints, such as the variation of the distance of the feet from the floor, the plane trajectory of the body's center of mass, the changing contact forces between the feet and the floor, as well as those developing into the leg joints, and so on. In addition, one can, for sake of simplicity, disregard

[26] This means that the term 'systemic' in 'systemic function' refers to the dependence of such role-concept to a system, and does not imply that the player artifact is a system itself. For example, if a wood table is held together by screws, those screws do have systemic functions in the table-system: they connect the table-legs to the table-top, even though single screws may not be systems themselves.

dynamic behaviour and only consider these "projections" at some finite set of time instants, and think of them as transitions between states. Then, each of these "projections" is itself a (simplification of) part of the original process.

Assuming that an upper ontology is fixed (e.g., DOLCE in this paper), one can group the state transitions depending on the classes available in that upper ontology and that are relevant to the transition of interest. Let us consider again the case of walking. The relevant feature(s) that characterize a walking in the upper level of DOLCE are the change in distance of the entity from some location. Indeed, at this level the ontology commits only to space and to location qualities (among the entities relevant to walking). Other classes, like foot, or other qualities, like speed, are not characterized or committed to at this level of the ontology. The function could not be said to be a walking function since at this high level in DOLCE there is no walking concept nor the needed ontological commitments to introduce it. If the walking function at this level can be characterized only by the change of the location quality, then we cannot distinguish it from the swimming and the flying functions (of course, this depends on the choice of DOLCE about what is in the upper level, it may differ in other ontologies). Note that even at this level cutting is a different kind of function than walking: cutting is characterized by the presence of one entity at the beginning and two or more at the end. In DOLCE, like most (perhaps all) upper ontologies, entities can be distinguished in number, thus cutting turns out to be a different type of function than walking even at this coarse level of characterization. In general, we call ontological functions the transformations as characterized by the ontological commitments (thus, distinctions) present in the upper ontology. As said, this provides only a high level (coarse) characterization of types of functions. Since the resulting taxonomy of ontological function-types strictly depends on the ontology adopted, we use the qualifier "ontological" in "ontological functions" (where more than one upper level ontology is used, one could be more precise indexing the ontological functions by the ontology they depend upon, e.g., 'DOLCE-ontological functions'). Continuing the discussion with DOLCE as the upper ontology, the ontology makes available the categories *region*, *quality*, *physical object*, *amount of matter*, etc. (see Figure 1), which give rise to several ontological function-types based on *change in quality-value* (for example, *the temperature of the oven increased*), *change of physical object* (e.g., *four wooden leg were assemblied together with a table-top to form a table*), *change of amount of matter* (e.g., *the water was vaporized*), and so on, see Figure 3.

Note that an ontological category can be involved in a transition in different ways. For example, an instance of said category could be eliminated, meaning that the instance was present in the state before the transition but not after, or an instance could be created, meaning that the instance was not present before a transition but only after. Another possibility is that there is a change of a relation of the ontology, e.g., in the initial state there were two individuals that were, say, one part of the other, while in the final state they are not. More complex changes can be described, depending on the difference between the initial and final structure of the ontology. Note also that in most cases the same event may be classified by several ontological functions. For instance, an event of cutting by a device, say a knife, is also a squeezing event, i.e., a compression (a change of the shape quality of the initial object) and a moving event (a change of the device location relatively to the object's location). We have reported some of these cases in Figure 2, which is not exhaustive. Finally, the functional terminology used in this paragraph is inspired by engineering literature which, as we have seen earlier, is not homogeneous. For instance, the *reclassification* (i.e., a change such that an instance changes type) of energy or matter is in other places called 'convert' [19, 27] or 'change' [1] or 'transform' [77], while the change of value of a spatial location quality is called 'channel' [1, 19], and so on.

Functional terms that are linked to domain-level terms (e.g., 'vaporize', 'moisten', 'heat') are not captured at the lever of ontological functions and require a domain or middle-level ontology containing corresponding concepts (e.g., vapour, humidity, temperature) in order to expand the construction we have just described up to that level of ontological distinctions. The ontological nature of some concepts, like 'energy', is unclear [81] despite being a fundamental concept in engineering and physics. Nonetheless, given an ontology containing a conceptualisation of energy, it is possible to introduce a corresponding ontological function, which might be different in another ontology with a different conceptualisation of energy. For instance, one could conceptualize energy as a quality, as done in [36]. This is the case, arguably, when one speaks about the energy *of something*, e.g, of a battery. In this case, in our ontology there would be a, say, *energy content* quality-type, which gives rise to a corresponding *convert energy* ontological function. Another possibility is to conceptualize energy as an endurant, which can reside into entities, change its location, and change type (e.g., heat energy, kinetic energy, etc.). In that case, the *convert energy* ontological function would be a reclassification-type transformation.

Formalizing ontological functions might be difficult. One would benefit from the introduction in the ontology of a general relation 'change of ... in event ...'. An ontological function, as understood in this paper, is essentially an event classifier which looks at what changes (or remains stable) in the event. This suggests defining specific functions by comparing, for instance, initial and final states. For example, a *connect* function could be introduced as follows:

d10 $\texttt{Connect}(x) \iff (\texttt{CL}(y,x) \iff \texttt{E}(y) \wedge \texttt{startState}(y,s_i,t_i) \wedge \texttt{endState}(y,s_f,t_f) \wedge \texttt{Goal}(s_f) \wedge \exists a,b,c(\texttt{POB}(a) \wedge \texttt{POB}(b) \wedge \texttt{POB}(c) \wedge \texttt{PC}(a,s_i,t_i) \wedge \texttt{PC}(a,s_f,t_f) \wedge \texttt{PC}(b,s_i,t_i) \wedge \texttt{PC}(b,s_f,t_f) \wedge \texttt{PC}(c,s_f,t_f) \wedge \texttt{P}(a,c,t_f) \wedge \texttt{P}(b,c,t_f)) \wedge \neg\exists c'(\texttt{POB}(c') \wedge \texttt{PC}(c',s_i,t_i) \wedge \texttt{P}(a,c',t_i) \wedge \texttt{P}(b,c',t_i)))$
"A concept is of *connect* type if and only if it classifies precisely those events that are transitions between a starting state (s_i, with time extension t_i) and a final state (s_f, with extension t_f) and, moreover, there are two physical objects (a and b) that participate in such states and, during the final state but not in the initial one, they are both part of a physical object (c)"

In the previous formula we used the predicate $\texttt{startState}(y,s_i,t_i)$ to mean that s_i is a state with time extension t_i, which is part of the perdurant y and such that there are no parts of the time extension of y which precede t_i. Analogous meaning for $\texttt{startState}(\cdot,\cdot,\cdot)$. Notice that, if one reverses the initial and final situations of Formula (d10), then a definition of a *divide* function could be obtained.

Another example, to define a *convert* function:

d11 $\texttt{Convert}(x) \iff (\texttt{CL}(y,x) \iff \texttt{E}(y) \wedge \texttt{startState}(y,s_i,t_i) \wedge \texttt{endState}(y,s_f,t_f) \wedge \texttt{Goal}(s_f) \wedge \exists\Phi,\Psi,a(\Phi \in \mathcal{NR} \wedge \Psi \in \mathcal{NR} \wedge \Psi \cap \Phi = \emptyset \wedge \texttt{PC}(a,s_i,t_i) \wedge \texttt{PC}(a,s_f,t_f) \wedge \Phi(a,t_i) \wedge \Psi(a,t_f)))$
"A concept is of *convert* type if and only if it classifies precisely those events that are transitions between a starting state (s_i, with time extension t_i) and a desired final state (s_f, with extension t_f) and, moreover, there is an entity (a) that at time t_i is instance of a type (Φ), while at time t_f is instance of a different, disjoint, type (Ψ). Necessarily, Φ and Ψ are types belonging to the set of all non-rigid types of the underlying ontology ($\mathcal{NR}$)"

Notice that in the previous definition we have quantified over the set of all non-rigid classes of an ontology ($\mathcal{NR}$). If such a set is finite, like in DOLCE, then the quantification corresponds to a finite disjunction of predicates, therefore Formula (d11) is a first-order formula. Moreover, upper-level ontologies typically contain only rigid categories (i.e., if an individual is a physical object, it is always a physical object), therefore, such a convert function becomes useful when modelling changes in entities that go through phases [67], when modelling entities that change their processual role (as described earlier) or in combination with non-rigid categories (as provided by middle or domain-level ontologies).

The type of all ontological functions that involve some change in quality values could be defined as follows:

d12 $\texttt{ChangeQV}(x) \iff (\texttt{CL}(y,x) \iff \texttt{E}(y) \wedge \texttt{startState}(y,s_i,t_i) \wedge \texttt{endState}(y,s_f,t_f) \wedge \texttt{Goal}(s_f) \wedge \exists a,q,v_i,v_f(\texttt{qt}(q,a)t_i \wedge \texttt{qt}(q,a)t_f \wedge v_i \neq v_f \wedge \texttt{PC}(a,s_i,t_i) \wedge \texttt{PC}(a,s_f,t_f) \wedge \texttt{ql}(q,v_i,t_i) \wedge \texttt{ql}(q,v_f,t_f)))$
"A concept is of *change of quality-value* type if and only if it classifies precisely those events that are transitions between a starting state (s_i, with time extension t_i) and a desired final state (s_f, with extension t_f) and, moreover, there is a participating entity (a) which has a quality (q) with different values, v_i and v_f, at times t_i and t_f, respectively."

If the case that the quality appearing in the formula (d12) is of type spatial location, the ensuing ontological function will be of type *channel*. Instead, if the quality in (d12) is such that it takes values into an ordered space (e.g., *temperature*, *humidity*, etc.) then the ensuing ontological function will be of type *vary*, which can clearly be specialized into *increase* and *decrease* types. (we are borrowing those terms from [19] and [1], mostly). Some functions, such as *store* or *support*, hint at the absence of change, as, for instance, a battery stores energy very well if it does not lose its charge over time, while a load-bearing wall supports the roof only if the roof does not fall. Therefore, concepts like store, support, or similar terms could be defined by modifying formula (d12) imposing that the initial and final values of the quality are equal instead of different.

At this point we make three observations:

– first, some processes are impossible to categorize only by observing the differences between a single couple of states. Take, for example, a temperature-controlled oven. The controller unit of the oven makes it so that the temperature in the oven stays, up to some tolerance, at a target value. The controller could achieve this by turning off the heat when a sensor realizes that the target value is surpassed, and by turning on the heat when the current temperature is less than the target. This makes it so that the temperature value over time oscillates around the target value (because the temperature keeps overshooting or undershooting), and such a process cannot be deduced by only observing the differences between two states. Therefore, processes like this cannot be reduced to ontological function. In this particular case, one could also argue that the function of the controller is to maintain the temperature in the oven around the set temperature[27], while the oscillations of the temperature are merely a side effect of the chosen implementation method, and would be different, or even absent, if the implementation method was different. In general, though, if one actually wishes to classify a transformation that cannot be described by comparison of two states, then the method employed in the previous formulas is not sufficient and a suitable comparison of what happens during the event should be introduced.
– Second, actual events can be extremely complex and participated by many relevant entities (think, e.g., the complexity of waking from a biomechanical point of view). This, together with the fact that we admit also ontological functions defined by the absence of some type of change (e.g., *support*), makes so that many events would be classified by several (possibly even all) ontological functions at once (as observed earlier while discussing the cutting example). This is not unexpected, and, in fact, reflects the flexibility of teleological thinking. Take, for example, a load-bearing wall. One could say that its function is to *support* the weight of the roof, but it may also *transmit* the load to the floor, or even *stop* the wind from entering the house, or *divide* the space inside the house into smaller rooms, and so on. To recognize what aspect is the most relevant in a given context is a task for the engineer or technician, which, using their contextual intelligence and creativity, determine what aspects to focus on. This is the reason for using the term 'function' (in combination with 'ontological', a qualifier we already justified earlier) to characterize these classifiers. Events, by themselves, are open to different interpretations.
– Lastly, we argue in favour of the flexibility of ontological functions. In the previous paragraphs, we have mainly used functional terms taken from the Functional Basis [19] or from the generally valid functions of [1]. Such vocabularies are indeed capable of expressing a vast quantity, if not the totality, of functions used within engineering domains, but they may not do so in the most natural way. For example, suppose that one is working with tooling machines that cut holes, slots, grooves, etc. in the workpieces. Then, using the Functional Basis one is reduced to using just the term *remove* to talk about the functions realizing such features in the workpiece. Instead, if one uses a domain ontology that contains concepts such as hole, slot, groove, etc., then one can build corresponding functional terms, say *make hole*, *make slot*, *to groove*, etc., defined analogously to the previous formulas. Such terms may be more natural than just using 'removing', and may even differ significantly from 'removing', depending on their precise formalisation. Another example: in the Functional Basis vocabulary, the term 'stop' means 'to cease the transfer of a flow', for instance 'A reflective coating on a window stops the transmission of UV radiation through a window' [19]. But what if perdurants are important in our domain and we want to say that something *stops a process* (e.g., "The addition of a respiratory inhibitor stops the absorption of amino acids", not that "stops the amino acid-material-flow from moving", but that it "stops the absorption", where absorption is an important element of the domain? We could derive useful functional terms, from an appropriate domain ontology managing concepts of domain processes.

Up to this point we have defined some types of ontological functions, but not the concept of ontological function itself. A possibility is to find an exhaustive list of types of ontological functions and then define ontological function as the disjunction of all those types, for instance:

d13 $\texttt{Function}_{\texttt{Ont}}(x) \Longleftrightarrow (\texttt{Connect}(x) \vee \texttt{Divide}(x) \vee \texttt{Convert}(x) \vee \dots)$

[27] This could be expressed in the style of the previous definitions by comparing an initial state, in which the set value and the actual value are different, to a final state, in which said values are equal.

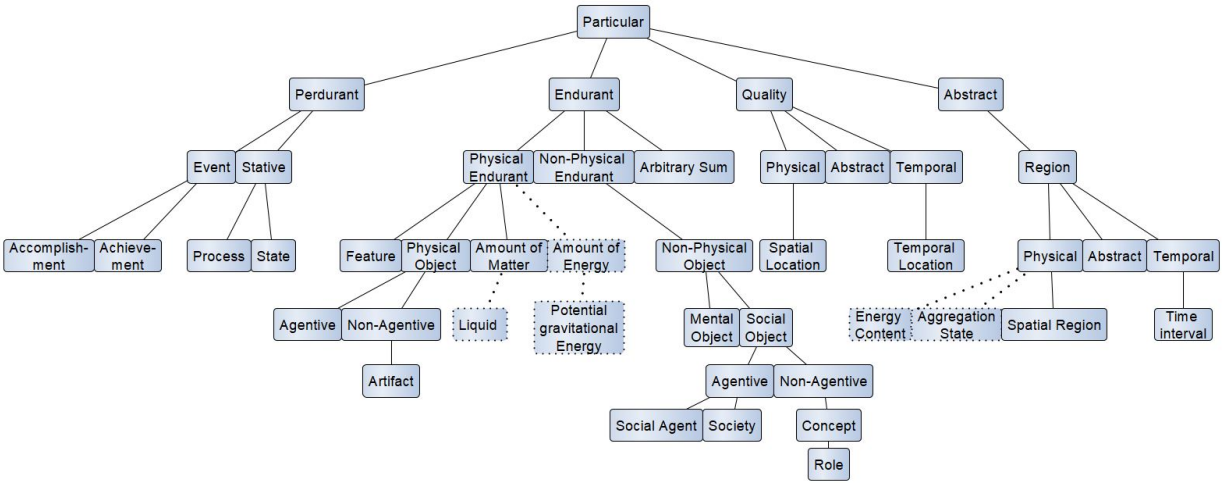

Fig. 1. DOLCE taxonomy, taken from [11]. Most divisions in subcategories are not exhaustive. The dotted classes are added to help some descriptions detailed in the paper, and are not part of DOLCE taxonomy.

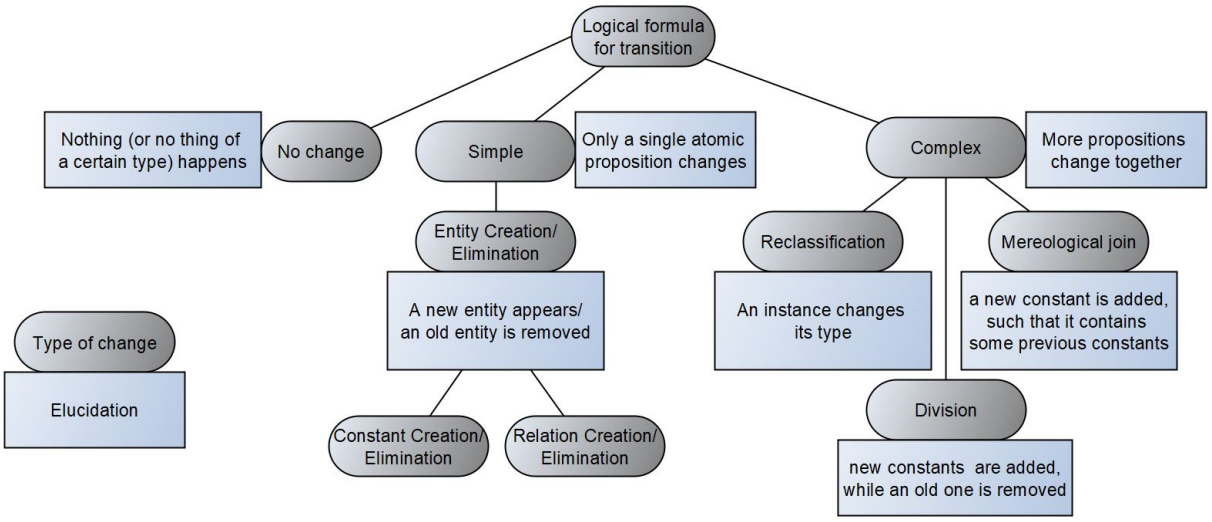

Fig. 2. A non exhaustive taxonomy of the logical formulas describing changes between states, classified depending on what predicates differ between the initial and final states.

This is, in principle, correct, but, in practice, requires an exhaustive enumeration of top-function types, which we do not want to commit to, in this paper. Instead, an intrinsic definition of ontological functions would be, informally:

d14 "A concept is a ontological function if and only if it classifies exactly those events consisting in a transition such that the changes (or the absence thereof) between the initial and final states (or across the whole transition) is characterized in terms of a formula expressed in the language of an upper or reference ontology."

(Of course, the formulas that engineers actually use to express functions are not so broad as in (d14), and typically focus on a small set of cases). Since "a formula expressed in the language of an upper or reference ontology" is not something that can be written in a natural way using first-order logic, we cannot give an intrinsic first-logical definition of ontological functions.

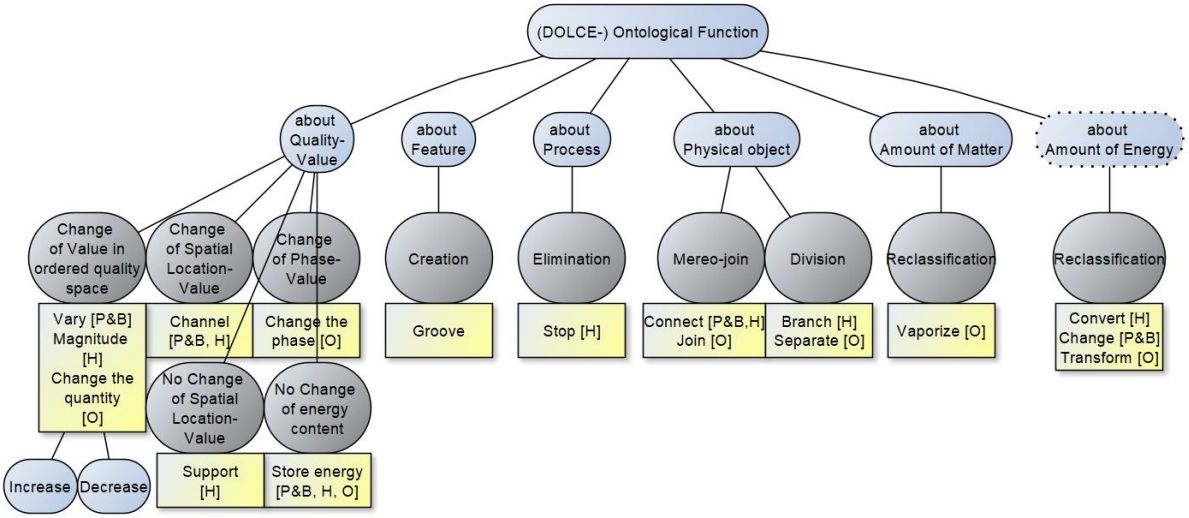

Fig. 3. A possible taxonomy of ontological functions. The light blue rectangles contain the ontological functions-categories. The gray circles contain the corresponding type of change at a logical level, and refer to Figure 2. Finally, the yellow rectangles contain terms used by other authors that could, arguably, used to label the ontological function category: H, P&B, and O refers to [19], [1], and [27] respectively. Notice that some of these terms are used by their authors with different meaning than in this figure, additionally, we do not have added terms related with information content.

Link between systemic functions and ontological functions. Returning to the relation between systemic functions and ontological functions, we observe that the former correspond to system-dependent functional descriptions, while the latter corresponds to system-independent functional descriptions. Since ontological functions are more general than systemic functions, they can be used to classify them:

a13 $\texttt{Function}_{\texttt{Ont}}(x) \implies (\exists y(\texttt{SP}(y,x) \wedge \texttt{Function}_{\texttt{Sys}}(y)) \wedge \forall y(\texttt{SP}(y,x) \wedge \neg\texttt{Function}_{\texttt{Ont}}(y) \implies \texttt{Function}_{\texttt{Sys}}(y)))$

"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, along the lines of definitions (d10) to (d12), abstracting from specific systems or from their occurrences in different parts of the same system.

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 engineers use when they need to, say, describe information about or collect information from different systems. For example, Collins [82], 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, a second meaning used by engineers is system-dependent. In general, when engineers focus on a single system, they use different concepts to speak 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

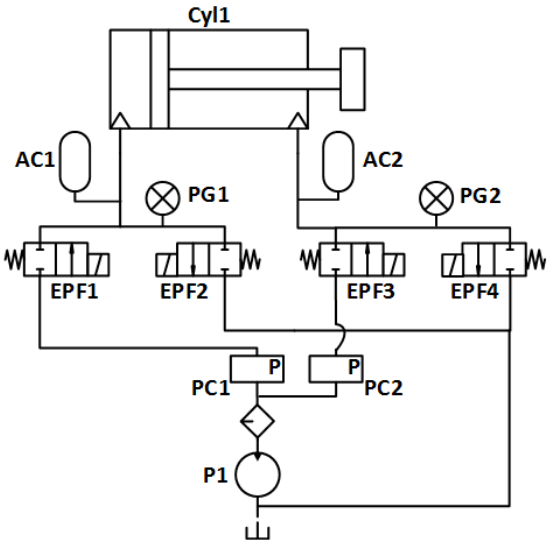

Fig. 4. The scheme of some hydraulic system, taken from [83].

example, Figure 4 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 valves could have the same type, nor 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 described 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 exist 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 behaviours and objects.
- The sub-functions of a function must 'organize'[28] 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.

[28] 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.

Additionally, Vermaas has proven that attempting to model a Functional Basis style of functional decomposition[29] entails contradictions [40][30], 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 $\texttt{decomp}$ is given, and focus on engineering methods.

Inspired by the work of Kitamura et al. on 'ways of functional achievement' [6, 84], we consider engineering methods as non-agentive social objects representing the knowledge that engineers share about ways of implementing functions through functional decomposition:

a14 $\texttt{Method}_{\texttt{Eng}}(x) \implies \texttt{NASO}(x)$

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:

a15 $(\texttt{mainFunction}(x) \vee \texttt{subFunction}(x)) \implies (\texttt{RL}(x) \wedge \exists \texttt{m}\,(\texttt{Method}_{\texttt{Eng}}(x) \wedge \texttt{founded}(x, \texttt{m})))$
"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 (axiom (a16)* is actually a meta-axiom, for this reason we mark it with a $*$-symbol, note that, given a finite set of methods, the axiom can be substituted with a finite set of axioms in FOL; the next axiom (a17) is an axiom schema):

a16* $\texttt{Method}_{\texttt{Eng}}(\texttt{m}) \iff \exists! n\, \texttt{Method}_{\texttt{Eng},\,\texttt{Sub}}(\texttt{m}, n)$, and n is integer
"Each method, say $\texttt{m}$, has a (unique) number, say n, of sub-functions that are contextualized by the method"

a17$_{schema}$ $\texttt{Method}_{\texttt{Eng},\,\texttt{Sub}}(\texttt{m}, n) \implies \exists! \texttt{main}, \texttt{sub}_1, \ldots, \texttt{sub}_\texttt{n}(\texttt{mainFunction}(\texttt{main}) \wedge \texttt{subFunction}(\texttt{sub}_1) \wedge \cdots \wedge \texttt{subFunction}(\texttt{sub}_\texttt{n}) \wedge \texttt{founded}(\texttt{main}, \texttt{m}) \wedge \texttt{founded}(\texttt{sub}_1, \texttt{m}) \wedge \cdots \wedge \texttt{founded}(\texttt{sub}_\texttt{n}, \texttt{m}))$
"for any engineering method of functional decomposition, say $\texttt{m}$, having a given number of sub-functions, say n, there exist a main-function role and n sub-function roles, which are founded on $\texttt{m}$ and are uniquely determined"

Since the main-function and the n sub-functions roles of a given method are univocally determined, we can represent them 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 $\texttt{m}$, as per Axiom (a17) (we omit the functional dependence of n on $\texttt{m}$, for ease of notation). Finally, the link between methods and decompositions is given in the following definition schema:

d15$_{schema}$ $\texttt{decomp}(\texttt{f}; \texttt{f}_1, \texttt{f}_2, \ldots, \texttt{f}_n) \iff$
$(\texttt{Function}_{\texttt{Sys}}(\texttt{f}) \wedge \texttt{Function}_{\texttt{Sys}}(\texttt{f}_1) \wedge \cdots \wedge \texttt{Function}_{\texttt{Sys}}(\texttt{f}_n) \wedge$
$\exists \texttt{m}(\texttt{Method}_{\texttt{Eng}}(\texttt{m}) \wedge \texttt{SP}(\texttt{f}, \texttt{main}^{\texttt{m}}) \wedge \texttt{SP}(\texttt{f}_1, \texttt{sub}_1^{\texttt{m}}) \wedge \ldots \wedge \texttt{SP}(\texttt{f}_\texttt{n}, \texttt{sub}_n^{\texttt{m}}))$
"$\texttt{f}$ 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 $\texttt{f}$ and $\texttt{f}_1, \ldots, \texttt{f}_\texttt{n}$, respectively"

The advantage of this approach is manyfold: it makes it 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 it possible to introduce properties of functional decomposition. For example, if one wishes to express that all functions are decomposable, she can state:

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

[29] 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 (canceling that edge), or in parallel, between nodes that do not share edges.

[30] 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.

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

d16 $\texttt{Function}_{\texttt{Eng}}(x) \Longleftrightarrow \texttt{mainFunction}(x)$
"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. That is, we ground the distinction on an ontological argument moving beyond the practical view given in ISO 15531-31[55], which associates capacities to a quantitative viewpoint and capabilities to a qualitative viewpoint.

In order to differentiate between capacities and capabilities, recall that the characteristics of a technical artifact are part of its physical make-up and determine how the artifact interacts with its environment. For example, an individual pump is 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 physical characteristics of the technical artifact and its 'being able' to do something can be conceptualized as individual qualities.[31] Consequently, we can model the capacity 'flow rate' (something that can be quantified) as a relational quality of the artifact relative to a certain kind of fluid, and the capability to pump (something that realizes a type of interaction or ability) of the same artifact as another (yet related) relational quality. But what are these capabilities? We assume that an entity has a capability if it can participate as doer in an ontological function. More precisely, an entity has a capability of a certain type (it has the capability of cutting) if and only if it can participate in the role of doer in an ontological function of the corresponding type (it participates as doer in a, perhaps not actual, event classified as cutting). Thus, we assume that ontological functions define types of capabilities, and that an entity has a capability only if it can manifest the corresponding ontological function.

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 an intrinsic quality and the former is an extrinsic quality, as we argued in Section 3. These examples suggest that *capacities* are those relational qualities that a capability is founded on, analogously to the approach in [4]. The point is that capacities are relational qualities that provide information about how the corresponding capability can be realized in practice. Going back to the pumping example, we conclude that the flow rate capacity parametrizes, perhaps only partially, the pumping capability.

[31] As seen in Section 3, individual qualities in DOLCE have associated quality spaces, which are essentially types of conceptual spaces as introduced in [85]. In the case of capabilities, the associated quality spaces should characterize the kinds of entities through which the capability can be realized.

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 in standard conditions. In our terminology, all the aforementioned properties are capacities: these properties are manifested only when the component is inserted into a working electrical circuit[32]; and they parametrize some capability of the component like, say in the case of a resistor, to create a voltage drop.

Capacities themselves are founded on some intrinsic physical qualities of the bearing object. For example, the maximum operating current will depend 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 characterize capacities as follows:

a18 $\texttt{Capacity}(x) \implies \texttt{relationalQt}(x) \wedge \exists y(\texttt{founded}(x,y) \wedge \texttt{intrinsicQt}(y) \wedge \texttt{bearer}(x) = \texttt{bearer}(y))$
"A capacity is a relational quality and is founded on some intrinsic quality, which is carried by the same bearer."
Where $\texttt{bearer}(\cdot)$ is the (unique) bearer of a quality, i.e.:

d17 $\texttt{bearer}(x) = \imath y(\texttt{qt}(x,y))$

a19 $\texttt{Capacity}(x) \implies \exists y(\texttt{Capability}(y) \wedge \texttt{founded}(y,x))$
"Each capacity founds a capability."

Axioms (a18) and (a19) do not fully define what a capacity is. They only give some constraints. Modelling the relation between capacities and capabilities, if possible, is difficult and requires further investigation on the topic. A full definition of capacity is, therefore, left as an open issue. Yet, we mention one interesting way to tackle the problem: to define a capacity as 'parameter' of the capabilities it founds, that is, as a quality such that its values are always related to the conceptual space of the corresponding capability:

ex3 $\texttt{Capacity}(x) \iff \texttt{relationalQt}(x) \wedge \exists y,z(\texttt{founded}(x,y) \wedge \texttt{intrinsicQt}(y) \wedge \texttt{founded}(z,x) \wedge \texttt{Capability}(z) \wedge \texttt{bearer}(x) = \texttt{bearer}(y) = \texttt{bearer}(z) \wedge \forall u,t(\texttt{ql}(u,z,t) \implies \exists v(\texttt{ql}(v,y,t) \wedge \texttt{P}(v,u,t))))$
"Capacities are precisely those relational qualities that are founded on some intrinsic quality (of the same bearer), and which found some capability (of the same bearer), such that every time that the capability takes a value (u), the capacity takes a corresponding value (v), which is part of u."

Briefly put, the formula above views capacities as the parameters of capabilities.

In contrast to (ex3), a characterisation of capabilities should touch upon the fact that capabilities are inextricably intertwined to functional aspects. In order to do this, we first try to clarify the nature of capabilities. To be able to do something is, arguably, a modal concept calling for possible worlds in which that something is actually done:

ex4 $\exists c(\texttt{PumpingCapability}(c) \wedge \texttt{qt}(c,x)) \iff \Diamond\exists e,t(\texttt{participatesAsDoer}(x,e,t) \wedge \texttt{PumpingProcess}(e))$
"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.[33] Finally, note that Example (ex4) also seems to suggest that capabilities depend on types of functions (not just processes or behaviours, for the pumping process in (ex4) will always play a function), which are used in their definition. This suggests to adopt the definition-founding relation already introduced in Section 3:

d18 $\texttt{Capability}(x) \iff \texttt{relationalQt}(x) \wedge \exists y(\texttt{Function}_{\texttt{Ont}}(y) \wedge \texttt{founded}_{\texttt{Def}}(x,y))$

[32] 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.

[33] An alternative is to model them as disposition, see for instance [54], and [86] for a broader introduction to dispositions.

"x is a capability if and only if it is a relational quality definition-founded on some ontological function"
Additionally, we assume that each capability is parameterized by at least a capacity (cf. (a19)):

a20 $\texttt{Capability}(x) \implies \exists y(\texttt{Capacity}(y) \wedge \texttt{founded}(x, y))$
"A capability is founded on some capacity"

From this axiom, we have that capabilities are specifically dependent on capacities, which specify how the artifact (the bearer) can function.The intuition, as anticipated at the beginning of this section, is that capabilities are relational qualities that associate their bearers with 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 the 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 (a18) and (d18) 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 fully explained by this approach. Yet, 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.

One reason for the difficulty in defining capacities and capabilities is the relational and potential nature (in our interpretation) of these concepts: they are relational, in the sense that we have attempted to capture with relational qualities, since they need the relation between their bearer and its environment to make sense; and they are potential, in the sense that they are strictly related to events that may occur, but need not to actually happen necessarily. Functions are, arguably, similar to capabilities in this aspect, and this may partially explain the terminological and conceptual complexity that these concepts bring. Of course, these concepts are not the only ones to be both puzzling and commonly used in engineering: *affordances* are another well-known example. As mentioned at the end of the literature review, engineering affordances are often defined as "interaction between artifact and user in which properties of the artifact offer a potential use to the user" [60], but this is not an universally shared definition, and conceptual confusion persists in the disciplines (not only engineering) that make use of this concept. We do not attempt to carry out an ontological analysis of affordances, but we do make two brief observations:

- first, the reasons for the confusion around the concept of affordance may be, at least partially, the same for the concepts analyzed in this paper. They strongly depend on relations between entities, and they are potential concepts, as opposed to actual.
- Second, one could attempt to reduce affordances to capabilities using, for example, the following informal equivalence:

  ex5 "An engineering artifact affords a user (or another artifact) to do something if and only if the system made by the artifact, the user (or the second artifact), and their relation has the capability to do that something"

From this point of view, our analysis of capabilities would carry at least some insight into affordances.

Finally, to recap the concepts introduced in the last two sections, 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 of, 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. Evaluation and possible applications

6.1. *Reduction to* OWL *ontology*

To facilitate the deployment of our theory in applications, we present an OWL version of the first-order logic formalization. This is standard practice, for at least the following reasons: first, OWL is decidable, while first-order logic is not, so that reasoning tasks will terminate in finite time[34]. Second, OWL is a standard endorsed by the World Wide Web Consortium (W3C) and is part of the Semantic Web technology stack [9], and is the language of choice for many ontologies in various domains [87].

Unfortunately, the increased computational properties of OWL come with a tradeoff with respect to its expressibility, therefore, one has to simplify the first-order theory. The original theory is still useful as it constrains with more detail the OWL concepts, and provides a more expressive language for conceptual modelling.

The first-order theory developed in the previous sections can be converted to OWL language as is, except for the expressions where ternary relations are used, as well as those that necessarily need at least three variables to be formulated. For example, the definition of a systemic function (d8)-(d9), the instantiation-founding definition (d5), and the engineering method schema (d15) 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

a21$_{owl}$ $\texttt{Function}_{\texttt{Sys}} \sqsubseteq (\forall \texttt{classifies.Behaviour} \sqcap \exists \texttt{causalContr.Goal}) \sqcap \exists \texttt{founded.System})$

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, all the ternary temporalized relations used in the first-order version, such as $\texttt{CL}(\cdot,\cdot,\cdot)$ and $\texttt{participateAsDoer}(\cdot,\cdot,\cdot)$, occur in the OWL version with the temporal argument removed, i.e., as binary relations. 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 arguments exists, what argument precisely depends on the relation. This is our choice for parthood (d7), classification, temporal quale, and participation-like relations as shown by these *meta-rules* aimed to show the intended interpretations:

d19$_{meta}$ $\texttt{CL}(x,y) \iff ((\exists t \texttt{PRE}(x,t)) \wedge \forall t(\texttt{PRE}(x,t) \implies \texttt{CL}(x,y,t)))$
"x is constantly classified by y if and only if x is classified by y whenever x exists"

d20$_{meta}$ $\texttt{cql}(x,y) \iff ((\exists t \texttt{PRE}(x,t)) \wedge \forall t(\texttt{PRE}(x,t) \implies \texttt{ql}(x,y,t)))$
"y is a constant temporary quale of the quality x if and only if y is a temporary quale of y whenever x exists"

d21$_{meta}$ $\texttt{participatesAsDoer}(x,y) \iff \exists t(\texttt{PRE}(y,t)) \wedge \forall t(\texttt{PRE}(y,t) \implies \texttt{participatesAsDoer}(x,y,t))$
"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 holds true at some time adopting the following *meta-rule*:

ex6$_{meta}$ $\texttt{participatesAsDoer}(x,y) \iff \exists t\, \texttt{participatesAsDoer}(x,y,t)$

[34] Though the complexity is exponential in the general.

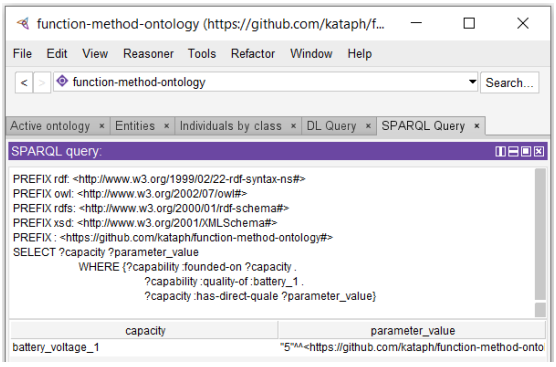

Fig. 5. An implementation of (CQ3) in Protegé.

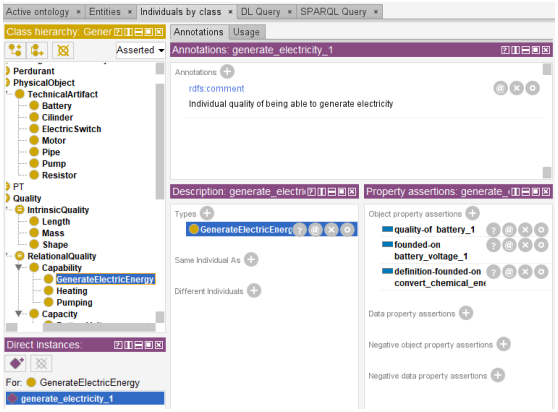

Fig. 6. A view of the ontology taxonomy in Protegé.

There are 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, (d21) excludes participation-as-doer in, say, a chemical process only for its first part, while (ex6) 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 a 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 (d15):

d22$_{meta}$ $\mathtt{mainFunctionOf(f,m)} \Longleftrightarrow (\mathtt{SP(f,main^m)} \wedge \mathtt{Function_{Sys}(f)})$
"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"

d23$_{meta}$ $\mathtt{subFunctionOf(f,m)} \Longleftrightarrow (\mathtt{SP(f,sub_i^m)} \wedge \mathtt{Function_{Sys}(f)})$
"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"

Even though the DOLCE ontology is primarily a first-order logic theory, OWL-adapted versions exist, like DOLCE-lite or DOLCE-Ultralite.[35] 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

[35] Available at http://www.loa.istc.cnr.it/index.php/dolce/.

the ISO 21838 standard.[36] The resulting ontology, which can be found on GitHub[37], was tested using the Hermit reasoner. (For the sake of example, the ontology is populated with only a few individuals).

6.2. Evaluation

The ontology developed in this paper is a hand-crafted prototype that discusses middle and high-level concepts. Therefore, some common approaches to evaluation cannot be employed, as, for example, there is no "gold standard" [88], or set of formal ontologies that one could use to compare common ontology metrics (such as those extracted by OntoMetrics[38]), nor there is a precise application against which to carry out the evaluation. Instead, we will evaluate our ontology using the OntoClean methodology [67] and the Ontology Pitfall Scanner [89]. Moreover, we will discuss how our ontology answers the research questions (RQ1)-(RQ3), as well as some competency questions that will serve to demonstrate the expressive potential of the ontology. Finally, we will also discuss some possible application scenarios, though we highlight that the ontology main goals are the research questions (RQ1)-(RQ3), while we delegate the development of derived application-level ontologies to further work.

OntoClean. OntoClean does not identify any issue in our taxonomy, that is, none of the meta-level rules that OntoClean methodology enforces are violated. In fact, this is almost trivial to check: since we are extending DOLCE categories, the only possible source of issues would be if the extensions of the role-concepts (that is, the set of all entities that, at a given time, are classified by the role-concept) were not anti-rigid: the other rules that must be checked in OntoClean methodologies are trivially true. This condition is not violated, since (systemic) functions, as we defined them, are contextual (e.g., a heat exchanger, with its behaviour, may be used either to heat a substance or to cool it, depending on the context). The ease of applying OntoClean is mainly due to our use of a (clean) upper-level ontology as reference.

Ontology Pitfall Scanner The OWL ontology was evaluated with the Ontology Pitfall Scanner (OOPS!) [89], which searches for common design errors from a list of 41 items. Among the pitfalls not related to imported ontologies, the scanner found some minor stylistic issues[39] and warned about a few important pitfalls: P11 ("Missing domain or range in properties"), P24 ("Using recursive definitions"), and P30 ("Equivalent classes not explicitly declared"). Of these three, P11 means that not every object property has domain and range axioms, but this is a conscious design choice to reduce the ontological commitment of the ontology. P24 occurs because OOPS! thinks that (a13) is a recursive definition (since the class of ontological functions appears both on the left and on the right of the material implication), while (a13) is a normal axiom. Similarly, P30 occurs because OOPS! is, incorrectly, assuming that the class of resistors (capabilities) and the class of resistances (capacities) should be equivalent, perhaps based on the similarity in their names.

Research questions and competency questions Our ontology answers the research questions listed in the introduction. In particular:

- **Answer to (RQ1).** We modeled functions as DOLCE-concepts that classify certain classes of perdurants. This modelling choice explains how functions can exist even when they are not executed. Moreover, we separated functions into the distinct categories of engineering methods and (proper) functions, and, further, we separated functions into systemic or ontological depending on their abstraction level, thus explaining the ambiguity on the why-how or abstraction-implementation axes.
- **Answer to (RQ2).** Systemic functions can be decomposed as discussed in Subsection 4.3, where each decomposition is associated with an engineering method.
- **Answer to (RQ3).** We have attempted full definitions of capabilities and capacities in Section 5. In the first case we leveraged ontological functions, in the second case we suggested considering capacities as parameters of capabilities. Moreover, capabilities and capacities are clearly separated by the use of the founding relation: capabilities are founded on capacities, while the opposite does not hold.

[36] https://www.iso.org/standard/71954.html.
[37] https://github.com/kataph/function-method-ontology.git.
[38] https://ontometrics.informatik.uni-rostock.de/ontologymetrics/index.jsp.
[39] Some inverse object-properties are not declared explicitly, and the naming convention differs between object-properties and classes.

Additionally, since the formal ontology answers these research questions, it is also able to answer related competency questions, which can be of interest in applications, such as the following:

CQ1 Given an (ontological) function, which artifacts have the capability of satisfying it?
CQ2 How can the given ontological function be implemented?
CQ3 What are the capacities that a given capability is founded on?
CQ4 Which parameters explain the performance of the some component in a given system? (If interpreted as 'what are the capacities of the capabilities of the component that are relevant in executing its function?')
CQ5 Which is the functional decomposition of a given system?
CQ6 Which is the function of the given tag and what is its purpose?

The OWL ontology, after being populated, can be used to answer the competency questions (CQ1)-(CQ6) by means of SPARQL queries. For example, these are implementations of (CQ1):

```
PREFIX : <https://github.com/kataph/function-method-ontology#>
SELECT  ?component
WHERE {
  ?capability :definition-founded-on :<the given function> .
  ?capability :quality-of ?component}
```

of (CQ4):

```
PREFIX : <https://github.com/kataph/function-method-ontology#>
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}
```

and of (CQ5), provided that the functional decomposition of a system is interpreted as the list of all systemic functions of the given system 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 : <https://github.com/kataph/function-method-ontology#>
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>}
```

The remaining competency questions can be implemented in a similar way (see also Fig.5 for (CQ3)).

The fact that our ontology can answer all these questions shows its expressivity and coverage. In the following paragraphs we explain how these characteristics could be used in application scenarios.

6.3. Motivating scenarios

Innovative Design (and other things). A good portion of the literature on functional modelling argues that functional modelling can be used to improve engineering design. For instance, the basic argument of the German school is as follows: designers should start from the customer's requirements, translate them into high-level functions, then decompose those functions into less abstract ones, until arriving at a point where finding a solution that realizes the functions is easy [1]. This methodology, is argued, facilitates engineers to quickly develop high-quality designs and enhances their creativity by decoupling the goals of the design from the ways they are achieved.

This argument is not wrong, but, in actual practice, one rarely starts from a blank slate. Designers are typically constrained by the need to reuse pre-existing solutions, for a company may have developed a legacy that is difficult to depart from, possibly because the company would not be able to (profitably) reroute its know-how and resources towards innovative lines of products, or other reasons. Therefore, designers usually have to devise marginal improvements or corrections in pre-existing solutions, and these activities seem at odds with the aforementioned methodology. Additionally, such methodology is design-oriented and does not seem relevant to other activities, such as troubleshooting or reverse engineering, despite functional reasoning being clearly relevant for those.

One important reason for carrying out ontological analysis is to bridge all such activities and cases, by producing an explicit representation (the ontology) of the shared conceptualisation of the relevant stakeholders, which can be used by all of them. We argue that our ontology is useful for the blank-slate innovation scenario (for, e.g., competency questions (CQ1) and (CQ3) showcase the possibility of linking capabilities to functions, which is important since it allows engineers to find whether they already have available components that can satisfy a given functional requirement), as well as many other scenarios, some of which we describe briefly in the following.

Incremental innovation In a company, innovation may happen mainly through incremental changes in pre-existing products. Therefore, product types will have a version history, in which each new version improves on the former. Using our ontology, one could describe the relevant improvements, allowing their digitalisation in a semantically meaningful way. For instance, a new version may introduce new capabilities, or it may have the same capabilities as the former version, but it could realize them better due to different capacities. Storing such information could be more helpful to engineers than just storing structural information (like the bill of material) of the different versions in the enterprise management system.

Knowledge transfer An engineer or technician could have been recently employed by a manufacturing company. In that case, they are (hopefully) trained to learn about the company's products and their functioning. Still, a good part of the knowledge the company possesses, especially functional knowledge, is implicit. Therefore, the trainee will learn about it only through experience or through its background knowledge of the domain. This works, but it may be a slow process, and some knowledge could be lost. Building a knowledge base modelling functional knowledge (necessarily together with other types of information, e.g. geometrical from CAD models, etc.) could facilitate training since it would relieve the trainee from having to deduce the functionality of components from their structure, as well as the (functional) role of components within their systems.

Troubleshooting When a technician troubleshoots a product he often makes use of functional reasoning (e.g., "the volume of the speakers is constantly louder than it should be. It may be that there is something wrong with the amplifier circuit, since it is that component that has the function of controlling the volume"). Sometimes, a technician may lack sufficient knowledge about the functional structure of a product to carry out such reasoning fruitfully (say, it is an external contractor with limited knowledge of the company which manufactures the product). In those cases, a clear rapresentation of the product functional structure, if accessible to the technician, may solve the issue. For example, a query answering (CQ4) may be useful in this case.

Formal Requirements. Requirements are an essential part of engineering design, which engineers have to write, share, maintain, and implement. The ensuing work and documentation can be daunting, especially in large progects. Therefore, attempts have been made to standardize [90], trace [91], and automatize [92] requirement pipelines using ontologies. Some approaches discuss how requirements should be written, so that their quality, shareability, and even automatic verification against a design model can be assured (e.g., [93, 94]). In the latter case, requirements are usually interpreted as assertions (constraints) about an engineering artifact that can be expressed (at least some

of them can) in a formal language. The fact that the requirement refers to something that *should be*, and not to something that *is*, means that requirements are modal concepts, though this can be left implicit in formal representations. For example, a requirement translated directly from a client's request, could be that the product, say a table, "must weight as little as possible", then engineers could precise this by stating

ex7 $weight(table\_top) \leqslant 5kg \wedge weight(i_{th}\_table\_leg) \leqslant 2kg$, for $i = 1, 2, 3, 4$.

Therefore, if one also designs products using the same formal language, the products can be checked against the set of constraints, to verify that those are satisfied by the products.

One difficulty of this approach is that some requirements are more difficult to express than others. Requirements such as (ex7) are easy to express, as they refer to physical properties of the products (they are sometimes called "physical requirements" [93]). On the other hand, requirements linked to product performance or functions are more difficult to express. For example, in [93], the requirements "The artifact (desk spot lamp) should be able to illuminate more than half a square meter of room", "The base (of the desk spot lamp) should provide support to the artifact", and "the short arm must have a hole [...]" are formalized as follows:

ex8 $arm(p) \implies \exists f(hole\_feature(f) \wedge feature\_of(f, p) \wedge [\dots]$.
ex9 $desk\_spot\_lamp(p) \implies \exists f(illuminating\_feature(f) \wedge feature\_of(f, p) \wedge illumination\_area(f) \geqslant 0.5$
ex10 $base(p) \implies \exists f(provide\_support\_feature(f) \wedge feature\_of(f, p)$

Therefore, predicates "*illuminating_feature*", "*provide_support_feature*", and "*hole_feature*" are introduced to classify the capability to illuminate, the function to support, and the hole of the short arm as "$feature\_of$" the corresponding artefacts. So that capabilities and functions are features in the same sense that a hole is, and we consider this an ontological error[40].

The point is that being able to express requirements that mention capabilities or functions is difficult, and an ontology describing such concepts would make things easier. For example, using our theory we would write requirements (ex9) and (ex10) as, respectively (in the following we have introduced a subclass of capabilities, *illuminating_capability*, a subclass of capacities *illumination_area*, and a subclass of (ontological) functions, *provede_support*):

ex11 $desk\_spot\_lamp(p) \implies \exists f, c, v(illuminating\_capability(f) \wedge \mathtt{qt}(f, p) \wedge illumination\_area(c) \wedge \mathtt{founded}(f, c) \wedge \mathtt{cql}(c, v) \wedge \mathtt{CP}(v, [0.5, \infty))$
"Every desk spot lamp has a capability to illuminate, founded on an illumination-area-capacity greater than 0.5"

ex12 $base(p) \implies \exists f_o, f_s(provide\_support(f_o) \wedge \mathtt{CL}(f_s, f_o) \wedge \mathtt{FunctionOf}_{\mathtt{Sys}}(f_s, p))$
"Every lamp base has a (systemic) function of (ontological-function-)type provide support"

Notice that (ex11) and (ex12) can be readily rewritten in OWL, so that they can be part of an OWL knowledge base containing all the requirements that can be expressed in a similar way. Then, if engineers build during design a model of the product, using the same language, they just need to import the model into the knowledge base with the requirements: the product model satisfies the requirements if and only if the ensuing ontology is consistent (similarly one can find if the requirements are inconsistent or if there are duplicates). In an actual application the user may never write complex formulas such as (ex11) and (ex12), unless user-friendly templates are made available as done in [94].

An analogous procedure is the one that the READI project [95, 96] aims to implement. In their case, the requirement are expressed in SCD (Scope Condition Demand) form. This means that they are assertions of the form

ex13 $\mathrm{S} \sqcap \mathrm{C} \sqsubseteq \mathrm{D}$

[40]For example, in DOLCE features form a category of endurants that are (generically constantly) dependent on physical objects that cannot be detached from their corresponding physical object without losing their identities. In this sense, a hole is a feature, while the wheel of a car is not. The fact that, in our approach, features are not capabilities or functions, entailed by our categorisation of those as qualities and non-physical endurants, respectively.

where S, C, and D are OWL classes, for example (the example is taken from [95]), "Equipment with a transport dry weight above 1000 kg shall be weighed by the manufacturer and a weight certificate shall be issued" is written in such a form, if S, C, and D are interpreted as the classes 'Equipment', 'things with transport dry weight above 1000 kg', 'things that have a weight certificate', respectively. Again, it is difficult to express functions- and performance-related requirements in form (ex13) if one lacks an appropriate reference ontology.

To summarize, using our ontology, one could enhance current work aimed at the digitalisation of written requirements, by facilitating the assertion of requirements related to product performance and functions.

7. Conclusion

The work presented in this paper contributes to the 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, behaviour, and function using first-order logic. Moreover, we have shown how one can use an ontology and the notion of 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, 7, 27]; the study of resources in manufacturing, in the applied ontology literature as well as some standards [4, 55, 57]; and the top-level ontology DOLCE and its developments [10, 11, 66]. This work relies on an in-depth ontological analysis of the domain, which, exploiting the characteristics of the DOLCE ontology, clarifies the conceptualisation of functions and related concepts in a systematic way.

We evaluated the quality of our implementation by assessing the clarity, expressive capability, and flexibility of our ontology with respect to some research questions, some competency questions, and some application scenarios. The relevance of our work is highlighted by the ability to answer a series of questions, from (RQ1)-(RQ3) to (CQ1)-(CQ6), showing that the ontology could be able to support applications in most engineering scenarios where functional reasoning is required.

Although this paper has set the core elements of our theory, many things require further work and testing to achieve a level of detail and coverage suitable for real applications. For example, linking the notions of behaviour to the modelling equations that engineers use when simulating a system is an interesting topic for additional research, as well as analyzing more in-depth how an ontological approach based on our theory can help the application scenarios that were touched on in the paper.

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.
[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* **17**(1) (2022), 45–69.
[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] 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.
[42] 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.
[43] Y. Kitamura and R. Mizoguchi, Characterizing Functions Based on Phase- and Evolution-Oriented Models, *Applied Ontology* **8**(2) (2013), 73–94.
[44] R. Mizoguchi, Y. Kitamura and The Hegeler Institute, A Functional Ontology of Artifacts:, *Monist* **92**(3) (2009), 387–402.
[45] P. Burek, R. Hoehndorf, F. Loebe, J. Visagie, H. Herre and J. Kelso, A Top-Level Ontology of Functions and Its Application in the Open Biomedical Ontologies, *Bioinformatics* **22**(14) (2006), e66–e73.
[46] P. Burek, F. Loebe and H. Herre, Overview of GFO 2.0 Functions: An Ontology Module for Representing Teleological Knowledge, *Procedia Computer Science* **192** (2021), 1021–1030.
[47] R. Arp and S. Barry, Function, Role and Disposition in Basic Formal Ontology, in: *Nature Precedings*, 2008.
[48] D.A. Spear, C. Werner and S. Barry, Functions in Basic Formal Ontology, *Applied Ontology* **11** (2016).
[49] J. Röhl and L. Jansen, Why Functions Are Not Special Dispositions: An Improved Classification of Realizables for Top-Level Ontologies, *Journal of Biomedical Semantics* **5**(1) (2014), 27.
[50] L. Jansen, Functions, Malfunctioning, and Negative Causation, in: *Philosophy of Science*, Vol. 9, A. Christian, D. Hommen, N. Retzlaff and G. Schurz, eds, Springer International Publishing, Cham, 2018, pp. 117–135.
[51] S. Borgo, A. Cesta, A. Orlandini and A. Umbrico, Knowledge-Based Adaptive Agents for Manufacturing Domains, *Engineering with Computers* **35**(3) (2019), 755–779.
[52] P. Vermaas, D. Eck and P. Kroes, The Conceptual Elusiveness of Engineering Functions:, *Philosophy & Technology* **26** (2012).
[53] 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.
[54] A. Sarkar and D. Šormaz, Ontology Model for Process Level Capabilities of Manufacturing Resources, *Procedia Manufacturing* **39** (2019), 1889–1898.
[55] ISO, ISO 15531-31: Industrial Automation Systems and Integration - Industrial Manufacturing Management Data - Part 31: Resource Information Model, Industrial Manufacturing Management Data, 2004.
[56] 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.
[57] E.M. Sanfilippo, W. Terkaj and S. Borgo, Resources in Manufacturing, in: *FOMI*, 2015, p. 12.
[58] J.J. Gibson, The Theory of Affordances, in: *The Ecological Approach to Visual Perception.*, Houghton Mifflin, Boston, 1979.
[59] J.R. Maier and G.M. Fadel, Affordance: the fundamental concept in engineering design, in: *International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, Vol. 80258, American Society of Mechanical Engineers, 2001, pp. 177–186.
[60] J.R.A. Maier and G.M. Fadel, Affordance Based Design: A Relational Theory for Design, *Research in Engineering Design* **20**(1) (2009), 13–27.
[61] D.C. Brown and L. Blessing, The Relationship Between Function and Affordance, in: *Volume 5a: 17th International Conference on Design Theory and Methodology*, ASMEDC, Long Beach, California, USA, 2005, pp. 155–160.
[62] S. Mota, Dispensing with the Theory (and Philosophy) of Affordances, *Theory & Psychology* **31**(4) (2021), 533–551.
[63] K. Campbell, *Abstract Particulars*, Basil Blackwell, Oxford, 1990.
[64] D. Marshall and B. Weatherson, Intrinsic vs. Extrinsic Properties, in: *The Stanford Encyclopedia of Philosophy*, Spring 2018 edn, E.N. Zalta, ed., Metaphysics Research Lab, Stanford University, 2018.
[65] 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.
[66] 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.
[67] 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.
[68] 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.
[69] F. Loebe, Abstract vs. Social Roles – Towards a General Theoretical Account of Roles, *Applied Ontology* **2** (2007), 127–158.
[70] S. Borgo, M. Franssen, P. Garbacz, Y. Kitamura, R. Mizoguchi and P.E. Vermaas, Technical Artifacts: An Integrated Perspective, *Applied Ontology* (2017), 18.
[71] R. Mizoguchi and S. Borgo, The Role of the Systemic View in Foundational Ontologies, in: *The Joint Ontology Workshops (JOWO)*, 2021, p. 11.
[72] R. Mizoguchi and F. Toyoshima, YAMATO: Yet Another More Advanced Top-Level Ontology with Analysis of Five Examples of Change, *Applied Ontology* (2017).
[73] 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.

[74] 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.
[75] J.S. Gero, Categorising Technological Knowledge From a Design Methodological Perspective, in: *International Conference on Technological Knowledge: Philosophical Reflections*, Boxmeer, the Netherlands, 2002.
[76] 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.
[77] Y. Kitamura and R. Mizoguchi, Ontology-Based Systematization of Functional Knowledge, *Journal of Engineering Design* **15**(4) (2004), 327–351.
[78] J.S. Gero and U. Kannengiesser, The Situated Function–Behaviour–Structure Framework, *Design Studies* **25**(4) (2004), 373–391.
[79] R. Mizoguchi, YAMATO : Yet Another More Advanced Top-level Ontology. http://www.hozo.jp/onto_library/upperOnto.htm.
[80] 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.
[81] C. McGinn, The Ontology of Energy, in: *Basic Structures of Reality*, Oxford University Press, New York, 2012.
[82] 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.
[83] 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.
[84] 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.
[85] P. Gärdenfors, *Conceptual spaces: The geometry of thought*, MIT press, 2004.
[86] 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.
[87] Ontology repositories, Technical Report. https://www.w3.org/wiki/Ontology_repositories.
[88] H. Sfar, A.H. Chaibi, A. Bouzeghoub and H.B. Ghezala, Gold Standard Based Evaluation of Ontology Learning Techniques, in: *Proceedings of the 31st Annual ACM Symposium on Applied Computing*, ACM, Pisa Italy, 2016, pp. 339–346.
[89] M. Poveda-Villalón, A. Gómez-Pérez and M.C. Suárez-Figueroa, OOPS! (OntOlogy Pitfall Scanner!): An On-line Tool for Ontology Evaluation, *International Journal on Semantic Web and Information Systems (IJSWIS)* **10**(2) (2014), 7–34.
[90] H. Alrumaih, A. Mirza and H. Alsalamah, Domain Ontology for Requirements Classification in Requirements Engineering Context, *IEEE Access* **8** (2020), 89899–89908.
[91] M.S. Murtazina and T.V. Avdeenko, An Ontology-based Approach to Support for Requirements Traceability in Agile Development, *Procedia Computer Science* **150** (2019), 628–635.
[92] O.M. Holter, Towards Scope Detection in Textual Requirements (2021), 15.
[93] Jinxin Lin, M.S. Fox and T. Bilgic, A Requirement Ontology for Engineering Design, *Concurrent Engineering* **4**(3) (1996), 279–291.
[94] R. Chen, C.-H. Chen, Y. Liu and X. Ye, Ontology-Based Requirement Verification for Complex Systems, *Advanced Engineering Informatics* **46** (2020), 101148.
[95] P.Ø. Øverli, T. Saltvedt, I. Ingebrigtsen, J. Slettebakk K., B. Ydstebø R., T. Karlsen, Klüwer K. Johan, M. Skjæveland G. and M. Knædal O., Asset Information Modelling Framework (IMF), Technical Report, Release 1, 2021.
[96] J.W. Klüwer, Ontology-Based Requirements Management - Presentation at SemWeb.Pro 2018, November 6 2018, Paris (2018), 29.