AACHENER VERFAHRENSTECHNIK

Technical Report LPT-2008-25

OntoCAPE 2.0

The Upper Level

J. Morbach, B. Bayer, A. Wiesner,

A. Yang, W. Marquardt

July 2008

Enquiries should be addressed to:

RWTH Aachen University
Aachener Verfahrenstechnik
Process Systems Engineering
52056 Aachen

Tel.: +49 (0) 241 80 - 94668
Fax: +49 (0) 241 80 – 92326
E-Mail: secretary.pt@avt.rwth-aachen.de

RWTH

Contents

List of Figures

1. Upper Level of OntoCAPE

The partial model **upper_level** is located on the Upper Layer of OntoCAPE. It establishes the fundamental organizational paradigm for the ontology and states the principles governing its design and evolution. The concepts introduced by the **upper_level** partial model are generic in the sense that they are applicable to different domains; thus, the partial model resembles the *meta_model* (Morbach et al., 2008a) in this respect. Yet unlike the Meta Model concepts, the concepts of the **upper_level** are intended for direct use and will be passed on to the domain-specific parts of OntoCAPE.

As for its function within the ontology, the **upper_level** serves two major purposes: Firstly, it gives a concise and comprehensive overview on OntoCAPE, thus helping a user to find his/her way around the ontology and to understand its major design principles. Secondly, it establishes a framework for the development (and later extension) of the ontology.

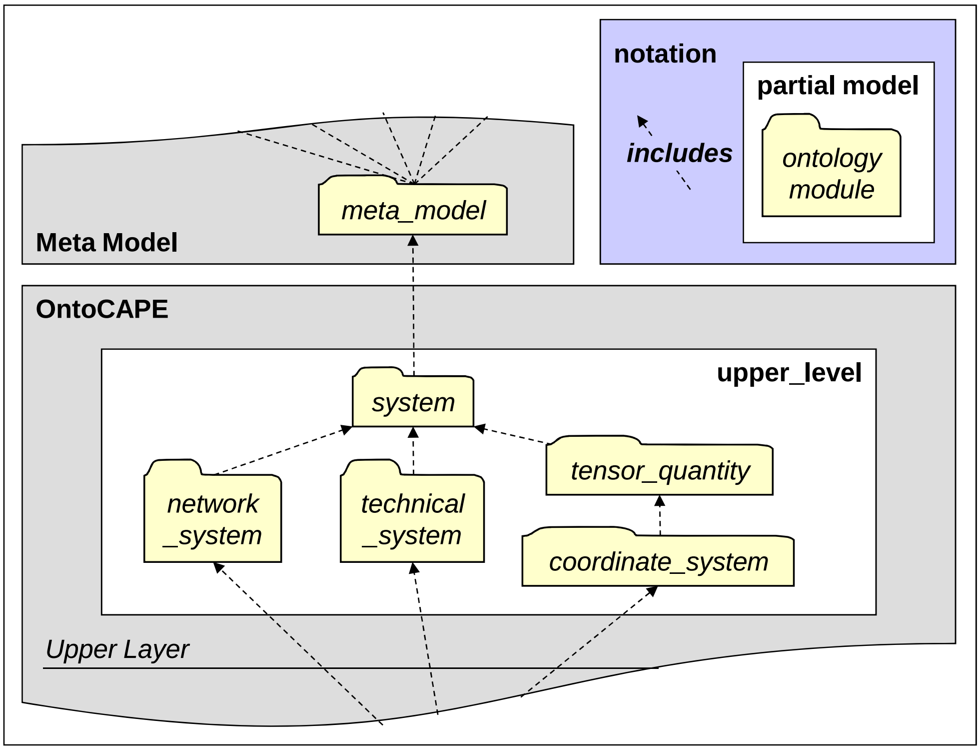

Fig. 1: The partial model *upper_level*

The **upper_level** partial model comprises five ontology modules (cf. Fig. 1). The module *system* is the most fundamental part of OntoCAPE. Consequently, it is located at the top of the "inclusion lattice" (Gruber and Olsen, 1994) that constitutes the ontology. As indicated in Fig. 1, the *system* module may import the ontology modules of the *meta_model*, provided that such an import is desired (cf. discussion in Morbach et al., 2008a).

The *system* module establishes the fundamental design paradigm according to which the ontology is organized: OntoCAPE is based on *general systems theory*[1] and *systems engineering*[2], which are considered advantageous organizing principles for building large engineering ontologies (e.g., Alberts, 1994; Borst, 1997; Bayer and Marquardt, 2004). The *system* module introduces the constitutive

[1] General systems theory is an interdisciplinary field that studies the structure and properties of systems (von Bertalanffy, 1968).

[2] Systems engineering can be viewed as the application of engineering techniques to the engineering of systems, as well as the application of a systems approach to engineering efforts (Thomé, 1993).

systems-theoretical and physicochemical primitives, such as *system*, *property*, *physical quantity*, *physical dimension*, etc., and specifies their mutual relations.

The remaining modules of the **upper_level** complement the *system* module: The modules *network_system* and *technical_system* introduce two important types of *systems* and their characteristics. The module *tensor_quantity* provides concepts for the representation of vectors and higher-order tensors, while *coordinate_system* introduces the concept of a *coordinate system*, which serves as a frame of reference for the observation of system properties.

2. System

2.1. Basic Axioms of Systems Theory

The *system* class is the central concept of the *system* module. It denotes all kinds of systems, which may be physical or abstract. The notion of a *system* is defined by the following axioms, which summarize the numerous definitions of the systems concept given in the literature (e.g., von Bertalanffy, 1968; Bunge, 1979; Patzak, 1982; Klir, 1985; Gigch, 1991):

(1) A *system* interacts with, or is related to, other *systems*.

(2) The constituents of a *system* are again *systems*[3].

(3) A *system* is separable from its environment by means of a conceptual or physical boundary.

(4) A *system* has *properties* which may take different *values*.

(5) The *properties* of a *system* can be explicitly declared or inferred from the properties of its constituent subsystems.

The above axioms constitute the basic principles of systems theory, as it is conceptualized in OntoCAPE. They will be revisited in the following sections, which discuss the concrete realization of the systems concept.

2.2. Inter-System Relations

Axiom (1) states that *systems* interact with, or are related to, other *systems*. These interactions are modeled by the relation isRelatedTo, which subsumes all kinds of binary relationships[4] between *systems* (cf. Fig. 2). The isRelatedTo relation is symmetric to account for the fact that, if *system* A is related to *system* B, then B is related to A, as well. Moreover, the relation is declared to be transitive, such that a third *system* C, which is explicitly related to B, can be inferred to be related to A, as well. Additionally, the non-transitive relation isDirectlyRelatedTo is established, which subsumes all direct relations between *systems*.

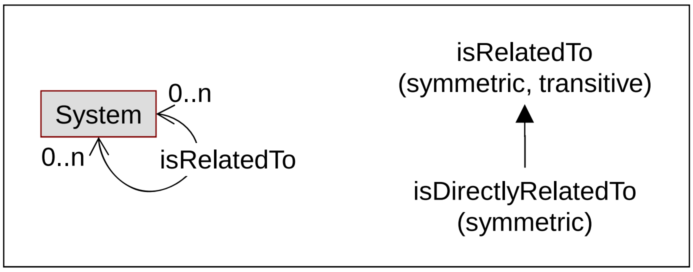

Fig. 2: Inter-system relations

2.3. Subsystems and Supersystems

For the realization of axiom (2) – the constituents of a *system* are again *systems* – the following concepts are introduced.

Firstly, the transitive relations hasSubsystem and its inverse isSubsystemOf are introduced as specializations of the isRelatedTo relation. They are derived from the aggregation relations hasPart and

---

[3] In systems theory, there are divergent views on the nature of system constituents (e.g., Bunge, 1979: "A system component may or may not be a system itself."). Sect. 2.3 addresses this issue in greater detail.

[4] A class to represent n-ary relations between *systems* is currently not implemented in OntoCAPE.

isPartOf introduced in the Meta Model (partial model **mereology**, cf. Morbach et al., 2008a; their respective definitions are identical, except that their ranges and domains are restricted to *systems*.

Next, the classes *subsystem* and *supersystem* are introduced as subclasses of *system*; they correspond to the generic *parts* and *aggregates* defined in the Meta Model. A necessary and sufficient condition that a system qualifies as a *subsystem* is that the system is linked to another *system* via an isSubsystemOf relation. Similarly, a *supersystem* is a *system* that has a hasSubsystem relation with some other *systems*. In accordance with the mereological theory defined in the partial model **mereology**, a *subsystem* can have *subsystems* of its own, and a *supersystem* may be part of another *supersystem*.

The relation hasDirectSubsystem is established as a means to indicate the direct *subsystems* of a *system*; hasDirectSubsystem is a subrelation of both hasSubsystem and isDirectlyRelatedTo, and it is defined analogously to the hasDirectPart relation introduced in the Meta Model. Similarly, its inverse isDirectSubsystemOf is declared to be a specialization of the isSubsystemOf relation.

A particular *subsystem* may be part of more than one *system*[5]. To indicate a *subsystem*'s unambiguous affiliation to a *supersystem*, the relation isExclusivelySubsystemOf and its inverse isComposedOfSubsystem are to be used. These relations are subrelations of isDirectSubsystemOf and hasDirectSubsystem, respectively; they are special types of the composition relations introduced in the partial model **mereology**. *Systems* that are involved in these relations are named *exclusive subsystem* and *composite system*.

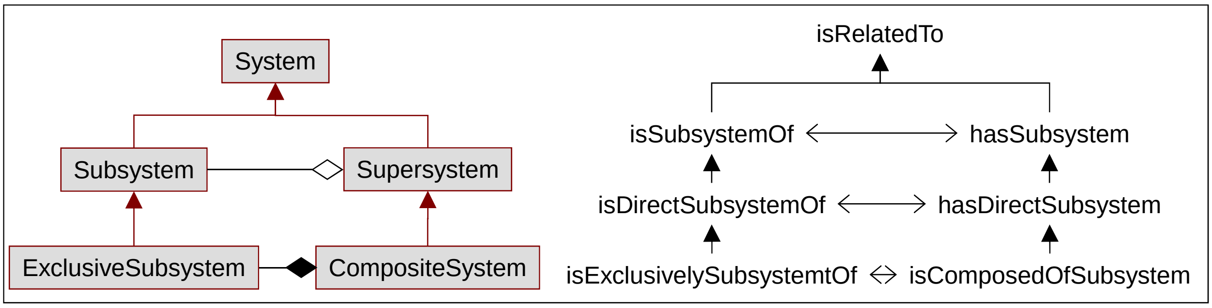

Fig. 3: Composition and decomposition of systems

Fig. 3 summarizes the classes and relations that represent the (de)composition of *systems*. In analogy to UML notation, we use a line with a white diamond-shaped arrowhead to represent the relations isSubsystemOf and isDirectSubsystemOf; a black diamond-shaped arrowhead indicates the relation isExclusivelySubsystemOf.

Unfortunately, current OWL reasoners scale badly when processing large collections of individuals connected via transitive, inverse relations (Rector and Welty, 2005). Hence, the relations hasSubsystem and isSubsystemOf can cause performance problems if applied to large data sets. A possibility to avoid these problems is to employ a single, non-inverse relation, instead. To this end, the unidirectional contains relation is introduced as a replacement for hasSubsytem. Like hasSubsystem, it is a transitive relation; unlike hasSubsystem, it has no inverse counterpart. The non-transitive relation containsDirectly is established as a specialization of contains; it is to be used analogously to the hasDirectSubsystem relation (cf. Fig. 4).

Aside from the performance considerations, there is another application case for the contains(Directly) relation: It is to be used when only one side of the aggregation relation is of interest, namely the indication of the constituting elements of a *supersystem*; by contrast, the inverse relation (i.e., the affiliation of a *subsystem* to a particular *supersystem*) is of little or no concern in this application case.

[5] A typical example for such a case are the classes *property model* (which models, e.g., the thermodynamic behavior of materials) and *process model* (which represents the mathematical model of a chemical process). These classes, introduced in the partial model *process_model*, are special types of *systems*. A particular *property model* may be a subsystem of different *process models* (Morbach et al., 2008c).

As an example, consider the relation between the concepts *mixture* and *chemical component*[6]. For the definition of a particular *mixture*, the information about its constituent *chemical components* is essential. However, for the definition of a *chemical component*, it is irrelevant to know of which *mixtures* the *chemical component* is part of. For that reason, the constituents of a *mixture* are indicated by means of the containsDirectly relation.

Note that the contained *systems* are not classified as *subsystems*, as this information is not relevant, as explained above. Only the containing *systems* are classified as *supersystems*.

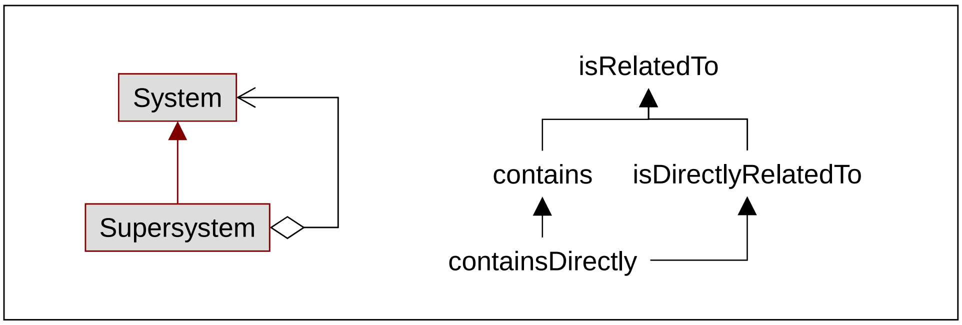

Fig. 4: The hasSubsystem relation may be replaced by the contains relation

As a graphical notation for the contains(Directly) relation, we use a line with a white diamond at the one end and an arrowhead at the other end. The diamond indicates the containing *system*, whereas the arrow points towards the contained *system*.

Closing the discussion on system (de)composition, it should be pointed out that some systems theorists (e.g., Bunge 1979) prefer an alternative formulation of axiom (2):

(2*) A system consists of multiple elements, which may or may not be systems themselves.

Thus, contrary to the original formulation of the axiom, the decomposition of a system into its constituent elements is mandatory, whereas these elements being systems is optional. This alternative version of axiom (2) will be referred to as axiom (2*) hereafter.

Fig. 5 shows the formal representation of axiom (2*). As can be seen, the representation of axiom (2) must be extended by one additional class (*element*) and two inverse relations (hasElement and isElementOf).

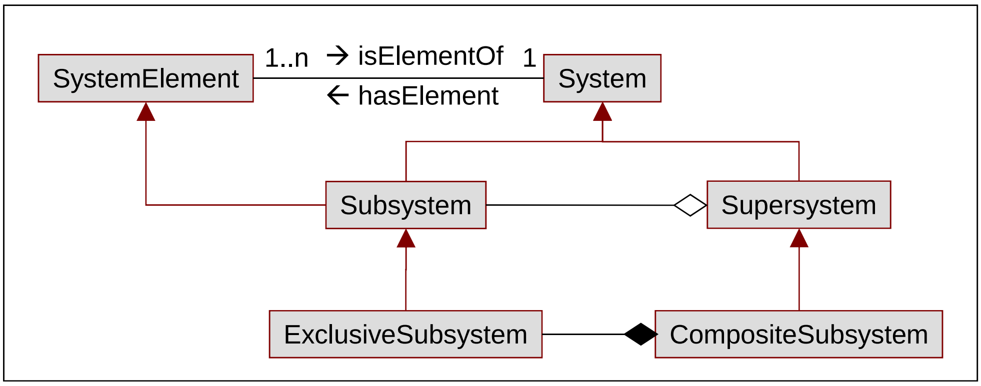

Fig. 5: Formal representation of axiom (2*)

There may be application cases where axiom (2*) is more advantageous than axiom (2). However, for the applications of OntoCAPE encountered so far, axiom (2) has proven to be adequate. Furthermore, since axiom (2) can be represented in a more compact way (cf. Fig. 3 and Fig. 5), it has been preferred

[6] *Mixture* and *chemical component* are special types of *systems*, which are introduced in the partial model *substance* (cf. Morbach et al., 2008b).

over (2*). As demonstrated, axiom (2) can be easily converted into axiom (2*) by adding the abovementioned classes and relations to the ontology, if such an extension is required by some application.

2.4. Levels of Decomposition

A (sub)system is considered *elementary* if it is not further partitioned into subsystems. However, it is often impossible to decide definitively if a system is elementary or composite. It might be elementary in one context, but in a different context a further refinement of the system's description might be needed (Bayer, 2003). Thus, being elementary is not a static classification.

In OntoCAPE, an *elementary system* is defined as a *subsystem* that (currently) has no *subsystems* of its own.

In an analogous manner, further (de)composition levels of systems can be established:

- A *top-level system* is a *supersystem* that is not a constituent of some other *system*.
- A *first level subsystem* is a *subsystem* that is a direct subsystem of a *top-level system*.
- A *second level subsystem* is a direct subsystem of a *first level subsystem*.
- Etc.

Due to the open world assumption, a DL reasoner cannot infer the membership to the classes *top-level system* and *elementary system* (cf. [Meta Model/Mereology]). Thus, membership must be declared explicitly. Once the top (or bottom) of a decomposition hierarchy has been defined that way, the membership to the intermediate decomposition levels can be inferred automatically.

2.5. Topological Connectivity of Systems

The relations isConnectedTo and isDirectlyConnectedTo are introduced to describe the topological connectedness of *systems*. They are defined and used just like the homonymic topological relations introduced in Morbach et al. (2008a) except that their ranges and domains are restricted to *systems* (cf. Fig. 6).

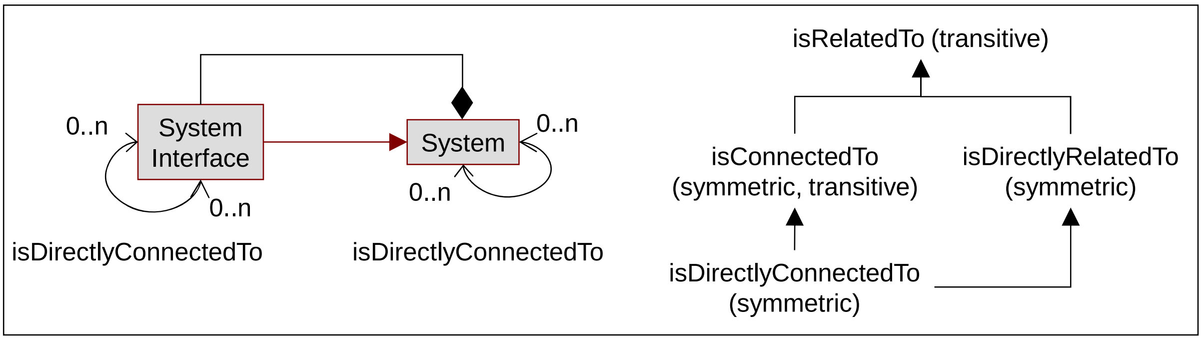

Fig. 6: Connectivity of systems

The relation isConnectedTo is symmetric and transitive; it summarizes all types of connections between *systems* (including indirect connectivity). The relation isDirectlyConnectedTo, a non-transitive specialization of isConnectedTo, represents direct connectivity between *systems*.

As explained in the Meta Model, mereological and topological relations exclude each other. Thus, isConnectedTo relations between a *subsystem* and its *supersystem* are prohibited. To enforce this restriction, the following range restrictions are imposed on the isDirectlyConnectedTo relation:

- A *first* (*second* …) *level system* can only be connected to a *first* (*second* …) *level system*.
- An *elementary system* can only be connected to an *elementary system*.

- A *top-level system* can only be connected to a *top-level system*.

Hence, connectivity is only allowed if two *systems* are on the same level of decomposition. If these restrictions are violated, the reasoner will produce an error message.

The class *system interface* represents the interfaces through which *systems* are connected to each other. The usage of this class is optional. It is derived from the meta class *connector* and should be utilized analogously.

2.6. Model

According to Wüsteneck (1963), a *model* is a system that is used, selected, or produced by a third system to enable the understanding of or the command over the original system, or to replace the original system. Model system and original system share certain characteristics that are of relevance to the task at hand.

Following this definition, the class *model* is introduced as a subclass of *system* (cf. Fig. 7). A *system* qualifies as a *model* if it models some other *system* (i.e., having a models relation to another *system* is a necessary and sufficient condition for being subsumed as a *model*). The relation isModeldBy is defined as the inverse of models.

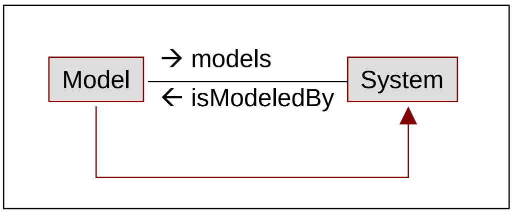

Fig. 7: Representation of models

Different types of *models* can be distinguished:

- *Iconic models* resemble the physical object they represent, but are simplified and/or employ a change of scale or materials. Typical examples would be an aircraft mockup used for wind tunnel testing, or a pilot plant that simulates the behavior of an industrial scale plant.
- *Symbolic models* represent the modeled system by means of some symbolic representation. Typical examples are mathematical models or information models.

Iconic models are *technical systems*, as defined in the ontology module *technical_system* (cf. Sect. 4). Symbolic models may be considered as *technical systems*, as well; however, this is not necessarily the case. A special class of symbolic models, *mathematical models*, is introduced in the ontology module *mathematical_model* (cf. Morbach et al., 2008c).

2.7. Representation of Viewpoints

Systems are often too complex to be understood and handled as a whole. A technique for complexity reduction that is widely used in systems engineering is the adoption of a *viewpoint*[7]. A viewpoint is an abstraction that yields a specification of the whole system restricted to a particular set of concerns (IEEE, 2000). Adopting a viewpoint makes certain aspects of the system ‘visible’ and focuses attention on them, while making other aspects ‘invisible’, such that issues in those aspects can be addressed separately (Barkmeyer et al., 2003).

In the following, the term *aspect system* (Patzak, 1982) will be used to denote those aspects about the overall system that are relevant to a particular viewpoint. An aspect system consists of a subset of the components (elements, relationships, and constraints) of the overall system. These components

[7] In the literature, the viewpoint approach is also referred to as “viewing the system from a certain *perspective*” or “considering the system under a particular *aspect*”.

constitute again a system, which is a subsystem of the overall system. Thus, an aspect system is a particular subsystem, which contains only those components of the overall system that are considered under the respective aspect.

In OntoCAPE, an *aspect system* is modeled as a subclass of an *exclusive subsystem* (cf. Fig. 8). The type of the respective *aspect system* can be explicitly labeled by an instance of the *aspect* class: To this end, the *aspect system* is linked to that *aspect* via the relation isConsideredUnderAspectOf. Like any *system*, an *aspect system* can be further decomposed – either into 'normal' *subsystems* or into further *aspect systems*. By means of the latter, an *aspect system* can be gradually refined.

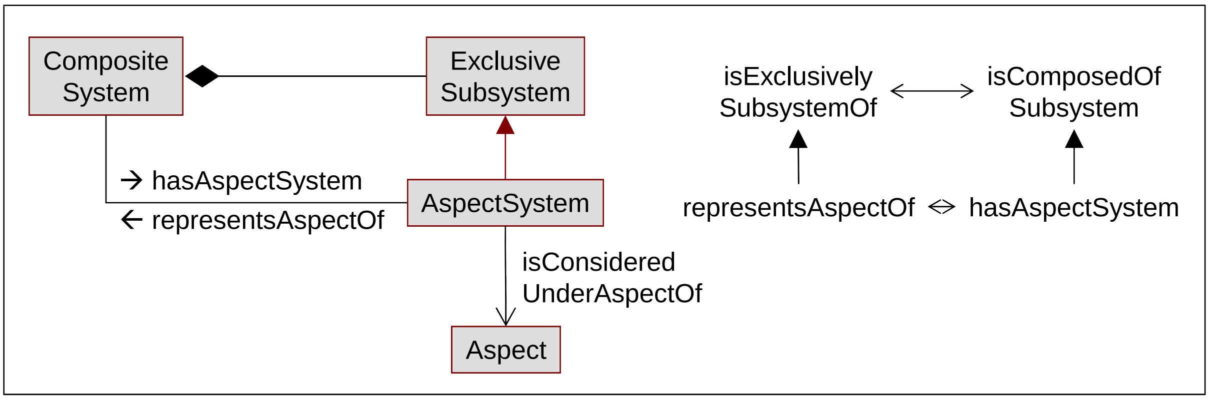

Fig. 8: Representation of aspect systems

The relationship between the *aspect system* and the overall (*composite*) *system* is given by the inverse relations representsAspectOf and hasAspectSystem, which are specializations of the composition relations isExclusivelySubsystemOf and isComposedOfSubsystem. These relations can be further refined to indicate the type of the *aspect system*: In the ontology module *technical_system*, for example, the class *system function* is introduced as a special type of an *aspect system* (cf. Sect. 2.7); a *system function* is linked to the overall *system* via the relation representsFunctionOf, which is a specialization of representsAspectOf.

Aspect systems play a key role in the organization of the OntoCAPE ontology. They are used to partition complex *systems* into manageable parts, which can be implemented in segregate ontology modules. An example is given in Fig. 9. Two *aspect systems*, *process* and *plant*, are shown, which represent a functional and a constitutional view on a *chemical process system* (cf. Wiesner et al., 2008). Each *aspect system* is represented in its own ontology module (*process* and *plant*, respectively). These modules are imported by the ontology module that holds the overall *system* (here, module *chemical_process_system* holding the *chemical process system*).

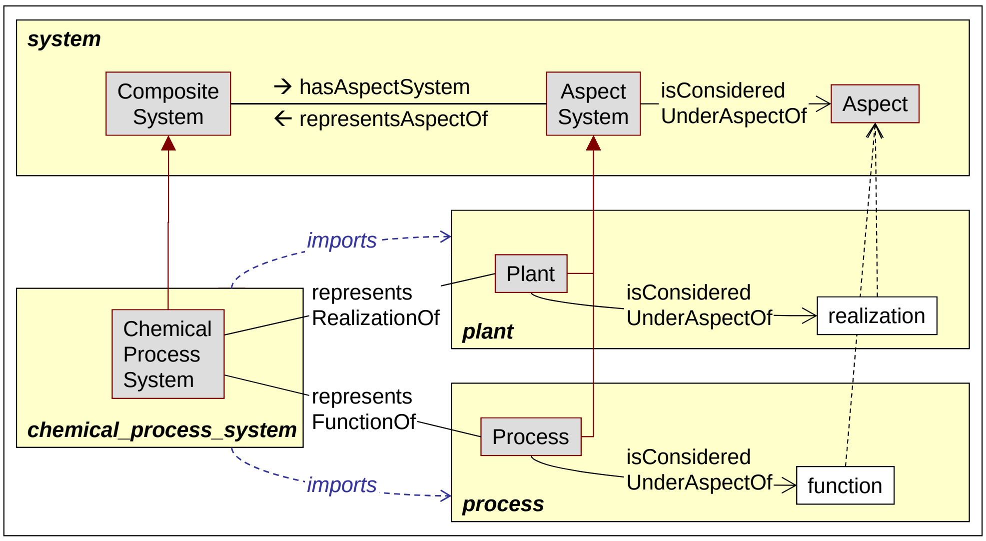

Fig. 9: Partitioning of a complex system into manageable parts

Within their respective ontology modules, *plant* and *process* are modeled as subclasses of *system*; only in the *chemical_process_system* module, they are identified as *aspect systems*. This is achieved by linking *plant* and *process* to the *chemical process system* via the relations representsRealizationOf and representsFunctionOf, respectively, which are specializations of the representsAspectOf relation. Based on this information, a reasoner can infer that *plant* and *process* are special types of *aspect systems*.

The above pattern is universally applied in OntoCAPE. The advantage of this pattern is that the *aspect systems* can be used and maintained independently of the overall *system*.

2.8. System Environment

Axiom (3) states that a system is separable from its environment by means of some conceptual boundary (which may or may not coincide with a physical system boundary). The key idea of this axiom is that the scope of a system is uniquely defined, i.e., it is clearly determinable whether a particular object forms part of the system or belongs to the system's environment. In OntoCAPE, the environment of a system can be modeled explicitly, as discussed in the following. The system boundary, on the other hand, is not represented in OntoCAPE, as it is merely an auxiliary construct to mentally demarcate the system from its environment[8].

Generally, the environment of a system includes everything that is not defined as that system (Alberts, 1994). Thus, the environment of a given *system* S can be defined as the class of all things that are not S. Note that such an environment class must be individually defined for each *system*, since the environment concept is relative.

However, the above definition is too broad for practical use. Normally, one is only interested in the *immediate* environment of a *system*, as defined by Bunge (1979):

> "Our definition of the environment of a system as the set of all things coupled with components of the system makes it clear that it is the *immediate* environment, not the *total* one – i.e., the set of all the things that are not parts of the system. […] we are interested not in the transactions of a system with the rest of the universe but only in that portion of the world that exerts a significant influence on the thing of interest."

In OntoCAPE, the immediate environment of a *system* is even further constrained to those individuals that are again *systems*[9]. Therefore, the environment of a system is defined as follows:

> The immediate environment of a particular *system* S includes all *systems* that (1) are not S, (2) are no subsystems of S, (3) are no supersystems of S, but (4) are directly related to S.

Note that the definition excludes subsystems since they form part of S and thus cannot be part of the environment of S. Supersystems are excluded since this would lead to false conclusions: It would allow a *supersystem* SupS of S to be part of the environment of S. On the other hand, S is a subsystem of SupS by definition. This would eventually imply that S is a subsystem of its environment.

In the formal specification of OntoCAPE, the class *system environment* exemplarily implements this definition for a sample *system* S[10].

---

[8] "The choice of the system boundary corresponds to a division of the universe of discourse into those parts included in the system under consideration and those belonging to the environment" (Marquardt, 1995).

[9] As opposed to *properties*, *values*, etc.

[10] *Implementation advice*: Currently, as of 2008, the reasoner RacerPro is not able to infer the environment of a system correctly. The problem is possibly caused by the allDifferent statement for individuals, which is not evaluated properly. Nevertheless, the definition is correct in principle.

2.9. Properties of Systems

Axiom (4) states that a *system* has *properties* which may take different *values*. In OntoCAPE, the *property* class represents the individual properties (traits, qualities) of a *system*, which distinguish the *system* from others. Typical examples would be *size, color*, or *weight*, which are modeled as subclasses of *property*.

The subclasses of *property* represent *general* properties, which exist autonomously (i.e., independent of a particular *system*). The *individual* property of a *system* is modeled by (1) instantiating the respective subclass of *property* and (2) linking that *property* instance to the *system*. For (2), the inverse relations hasProperty and isPropertyOf are to be used (cf. Fig. 10). As soon as the *property* instance is linked to a *system*, it represents an inherent quality of that particular *system* and thus must not be assigned to any other *system*. To ensure that a *property* instance is assigned to one *system* instance at most[11], the isPropertyOf relation is declared to be functional.

Subclasses of *property* will be introduced on the lower levels of OntoCAPE to represent properties such as *height*, *volume*, *diameter* etc. These classes can be further specialized in order to clarify the meaning of the respective *property* (e.g., refine *diameter* to *internal diameter*, *nominal diameter*, etc.). However, the refinement must not imply the affiliation to a particular *system*; for example, neither *pipe diameter* nor *vessel diameter* are valid refinements of *diameter*[12]. Instead, the affiliation to a specific *system* is modeled on the instance level by assigning a *property* instance to a *system* instance via the isPropertyOf relation.

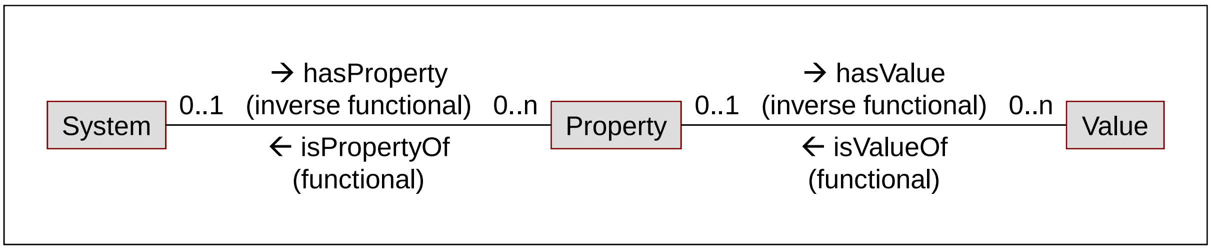

Fig. 10: A *system* has *properties* which may take different *values*

A *property* has certain values – for example the *property* '*color*' may take the values 'red', 'green', 'blue', etc. In OntoCAPE, the values of a *property* are represented through the *value* class, which is linked to a *property* via the isValueOf relation and its inverse hasValue, respectively. A *value* is either of qualitative nature (pertaining to *properties* like *color*, *taste*, etc.) or of quantitative nature (pertaining to *properties* like *weight*, *height*, or *temperature*). To avoid ambiguities, the isValueOf relation is declared to be functional; thus, an instance of *value* can be assigned to one *property* instance at most. A *property*, in contrast, may have multiple *values*: Take for example the *temperature* of a solid body – while the existence of this property itself is invariant (a solid body will always have a temperature), the temperature *values* may change over time.

2.10. Backdrop

To distinguish the different *values* of a *property*, the concept of a *backdrop* (Klir, 1985) is introduced. Adapting Klir's definition[13] to the terminology of OntoCAPE, a backdrop is some sort of background

---

[11] Some *properties* are not owned by a particular *system* at all (cf. Sect. 2.15)

[12] As an exception to this rule, one may define high-level categorizing *properties* which subsume the *properties* of a specific *system*; for instance, the class *phase system properties* subsumes the various *properties* of a *phase system*. However, these kinds of *properties* are only introduced for organizational purposes and are not to be instantiated for practical use.

[13] Klir defines a *backdrop* as "any underlying property that is actually used to distinguish different observations of the same attribute […]. The choice of this term, which may seem peculiar, is

against which the different *values* of a *property* can be observed. Thus, a backdrop provides a frame of reference for the observation of a *property*. Space and time are typical choices of backdrops.

In OntoCAPE, the *values* of any *property* can act as a backdrop to distinguish the *values* of another *property*. The relation isObservedAgainstBackdrop maps the *values* that are to be distinguished to their respective backdrop *values*. An example is presented in Fig. 11: Here, the *values* of the *property* Time are used to distinguish the different *values* of the *property* Temperature, which arises in the course of an observation[14]. In this particular example, a temperature of 285 Kelvin was observed at the beginning of the observation; after 300 seconds, the temperature had cooled down to 273 Kelvin.

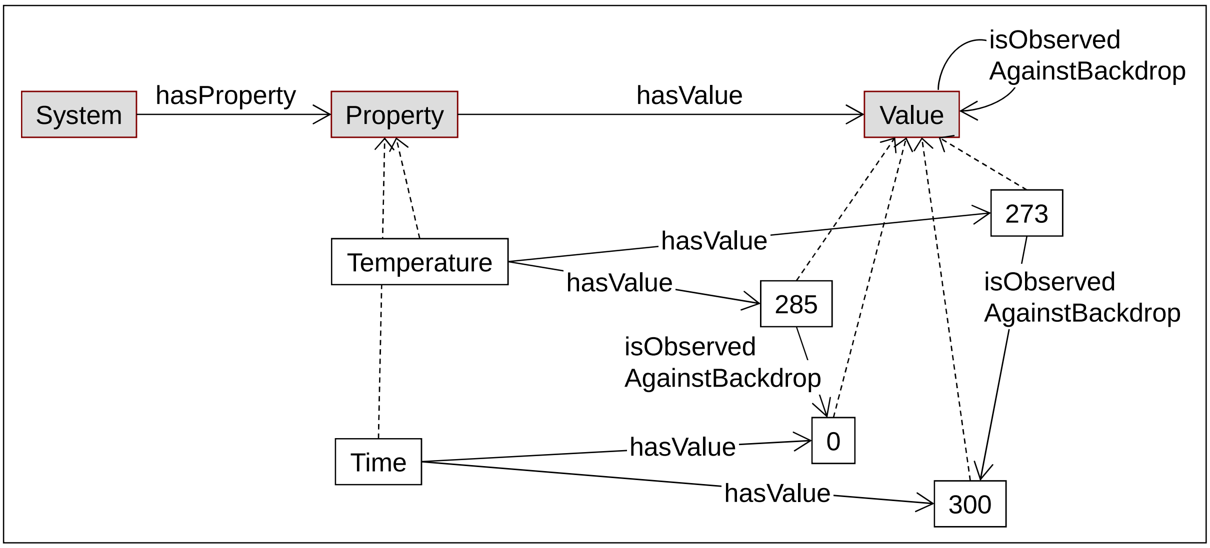

Fig. 11: Distinguishing the different *values* of a *property* by means of the backdrop relation.

The observed *property* and its backdrop *property* may both be owned by the same *system*; however, this is not mandatory. Often, the backdrop *property* is owned by a *coordinate system*, which is introduced in the ontology module *coordinate_system* (cf. Sect. 5).

Note that the backdrop concept is relative: A *physical quantity* acting as a backdrop may be observed against another backdrop quantity. Consider for instance a *physical quantity* that is observed against the space coordinate of a moving *system;* the movement of this space coordinate could in turn be measured against the space coordinate of a fixed coordinate system. Another example is given in Fig. 12. It extends the above example of temperature measurement (Fig. 11) by indicating the time and date of the observation. To this end, one defines a backdrop relation between the starting time of the observation (t = 0 sec) and the date-time, given by the time standard UTC (Coordinated Universal Time, cf. Sect. 2.10).

---

motivated by the recognition that the distinguishing property […] is in fact some sort of background against which the attribute is observed".

[14] The properties in the example are *physical quantities* (cf. Sect. 2.11). Actually, the values of *physical quantities* are represented in a slightly different manner, but the representation is simplified here for the sake of clarity. The exact representation of the example is shown in Fig. 15.

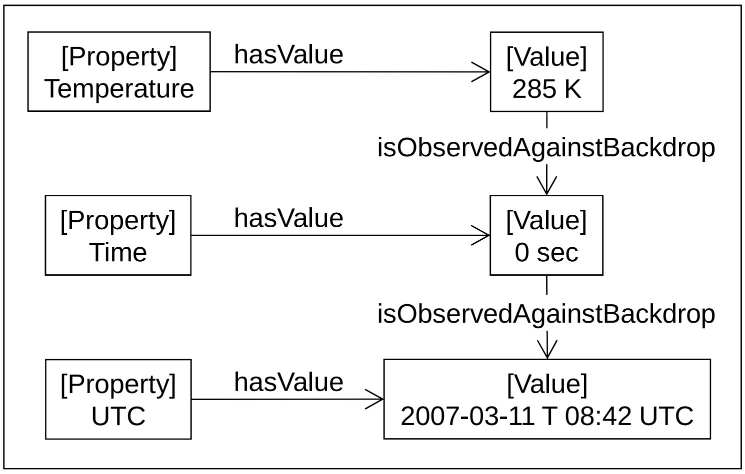

Fig. 12: UTC as a backdrop for the starting time of the observation

The indication of backdrop is not mandatory; it can be omitted if it is not important or if it can be recognized from the context. In particular, a backdrop is often superfluous if the *property* can take only a single value. In this case, the property is classified as a *constant property*.

2.11. Physical Quantity

The *International Vocabulary of Basic and General Terms in Metrology* defines a *physical quantity* (often abbreviated as a 'quantity') as a "property of a phenomenon, body, or substance, to which a magnitude can be assigned" (VIM, 1993). A more extensive definition of the term is given in the *EngMath ontology* (Gruber and Olsen, 1994):

> "Physical quantities come in several types, such as the mass of a body (a scalar quantity), the displacement of a point on the body (a vector quantity), […] and the stress at a particular point in a deformed body (a second order tensor quantity). […] Although we use the term "physical quantity" for this generalized notion of quantitative measure, the definition allows for nonphysical quantities such as amounts of money or rates of inflation. However, it excludes values associated with nominal scales, such as Boolean state and part number […]."

In OntoCAPE, a *physical quantity* is a *property* that has quantifiable values (the latter are represented through the class *quantitative value*, cf. Fig. 13). In agreement with the definition given in the EngMath ontology, the class denotes both physical and nonphysical quantities, and it comprises scalars as well as vectors and higher-order tensors. Only *scalar quantities* are considered here; the representation of *vector quantities* and higher-order *tensor quantities* is discussed in Sect. 6.

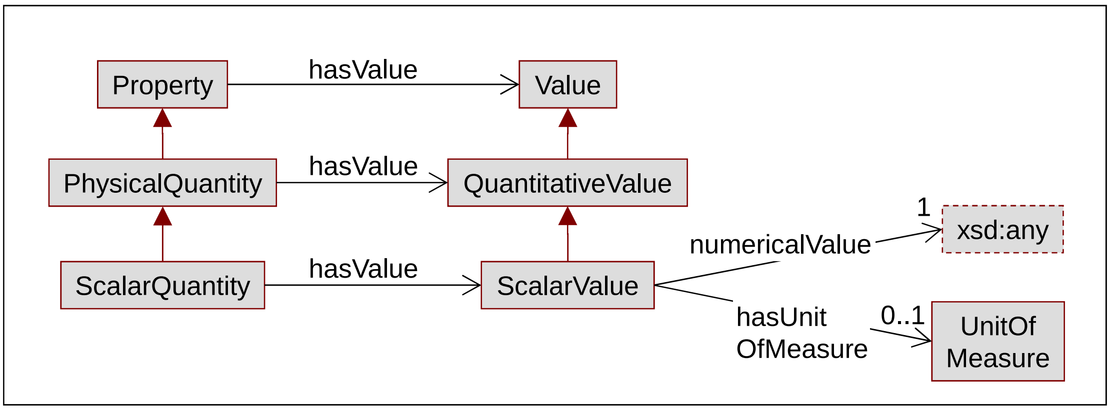

Fig. 13: Representing the values of *physical quantities*

Generally, the value of a *scalar quantity* consists of a number and (possibly) a unit of measure. The unit of measure is a particular example of the quantity concerned, which is used as a reference, and the number is the ratio of the value of the quantity to the unit of measure (BIPM, 2006). In OntoCAPE, the values of a *scalar quantity* are represented by instances of the class *scalar value*, a subclass of *quantitative value*: The number part of a *scalar value* is expressed by the attribute numericalValue[15], and the unit of measure part is represented by an instance of the *unit of measure* class, which is connected to the *scalar value* via the relation hasUnitOfMeasure (cf. Fig. 13). An application example is presented in Fig. 14, which shows the representation of a temperature value of 351.8 Kelvin.

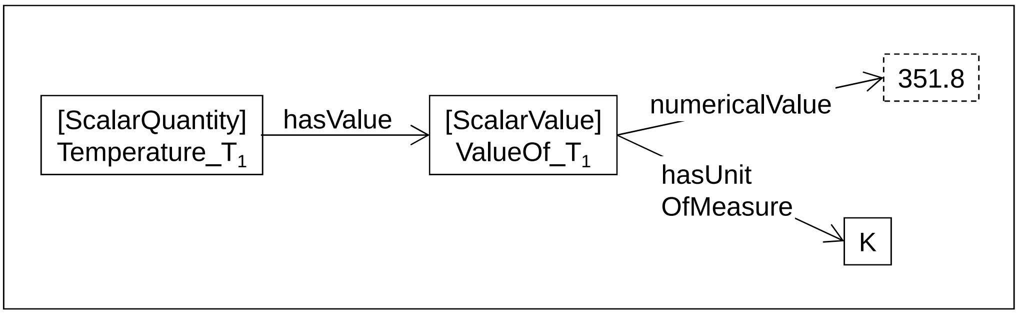

Fig. 14: Application example: The quantity Temperature T1 has a value of 351.8 K.

Fig. 15 shows a more extensive example Fig. 11; it represents the time-dependent measurement of a temperature. The *scalar quantity* Time acts as a backdrop to distinguish the different values of the *scalar quantity* Temperature.

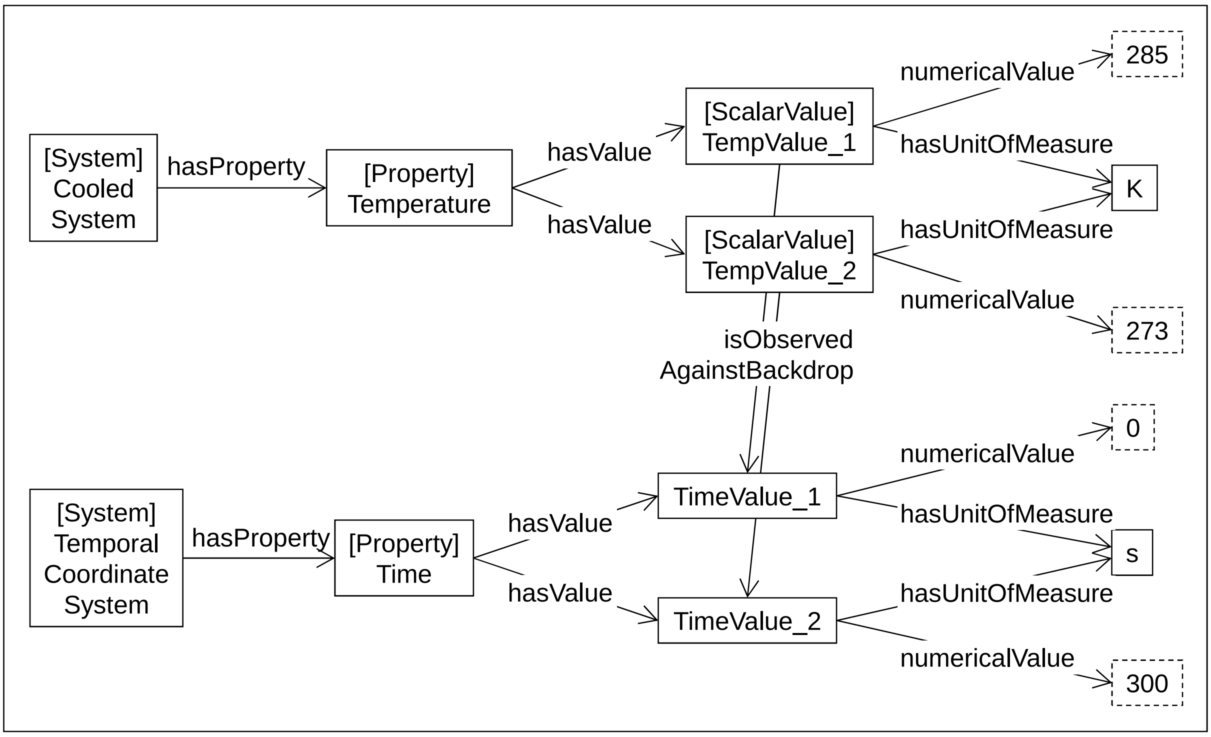

Fig. 15: Application example: Temperature measurement with multiple values

[15] Ordinarily, the values of numericalValue are of type float; however, other XML Schema datatypes are also possible, such as dateTime.

2.12. Physical Dimension

By convention, *physical quantities* are organized in a system of dimensions (BIPM, 2006). In such systems, each *physical quantity* has exactly one associated *physical dimension*. A typical example would be the dimension of length, which can be associated with such *physical quantities* as *height*, *thickness*, or *diameter*.

In OntoCAPE, dimensions are modeled by the class *physical dimension*. A particular instance of *physical dimension* can be assigned to both a *physical quantity* and a *unit of measure* via the relation hasDimension (cf. Fig. 16). For instance, both the *scalar quantity* '*radius*' and the *unit of measure* 'meter' have the dimension of length.

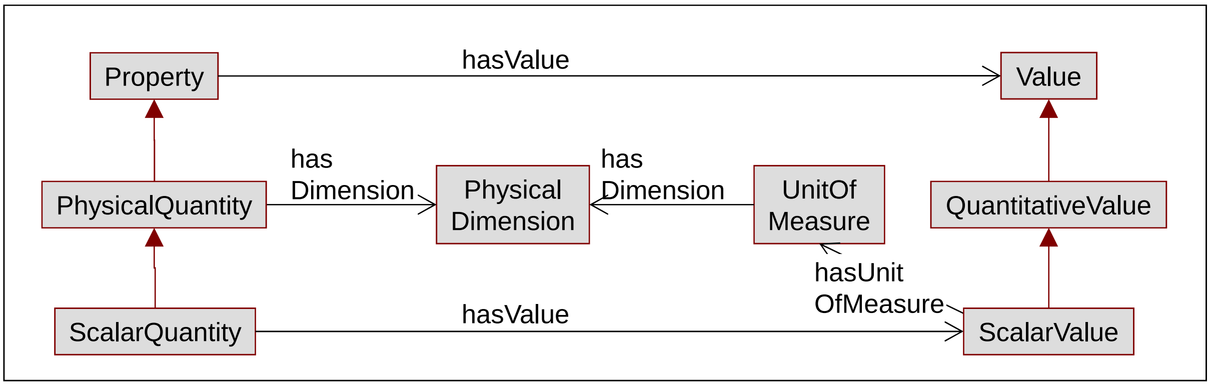

Fig. 16: Physical dimensions

Physical dimensions serve two functions in OntoCAPE:

(1) *Physical quantities* of the same *physical dimension* share certain characteristics; for instance, their *scalar values* relate to the same set of *units of measure*. Thus, the concept of *physical dimension* may be used to identify *physical quantities* of the same kind[16] and to differentiate those from other kinds of *physical quantities*.

(2) According to the conceptualizations stated so far, arbitrary *units of measure* can be assigned to the *scalar value* of a particular *scalar quantity*. Now, the *physical dimension* provides a means to constrain the set of possible *units of measure* for a given quantity. To this end, one needs to implement[17] the following constraint:

> A *unit of measure* that is assigned to the *scalar value* of a *scalar quantity* must have the same *physical dimension* as the *scalar quantity*.

On the basis of this constraint, the consistency of unit of measure assignment and conversion can be checked. For example, a meter is a valid *unit of measure* for measuring the *scalar value* of a *radius*, as

[16] The International Vocabulary of Basic and General Terms in Metrology (VIM, 1993) defines 'quantities of the same kind' as "quantities that can be placed in order of magnitude relative to one another". While it is true that quantities of the same kind must have the same physical dimension, the opposite is not true, i.e., having the same physical quantity is a necessary, but not a sufficient condition for being of the same kind. For example, moment of force and energy are, by convention, not regarded as being of the same kind, although they have the same dimension, nor are heat capacity and entropy (VIM, 1993).

[17] In principle, the constraint could be formulated in the OWL modeling language; however, such an implementation would be quite exhausting, as the constraint would have to be formulated individually for each *scalar quantity*. Alternatively, the constraint can be implemented through a single, generic rule, which applies to all quantities. Rules form not part of current OWL, but can be formulated on top of the language. The latter approach is taken in OntoCAPE.

both *radius* and meter have the dimension of length. Similarly, meters can be converted into feet, as both *units* of measure have the same dimension.

2.13. Qualitative Value

Obviously, not all *properties* are *physical quantities*. The *values* of *properties* like 'color' or 'flavor' are not (numerically) quantifiable. Instead, such *values* are represented by means of the class *qualitative value*, a subclass of *value* (cf. Fig. 17).

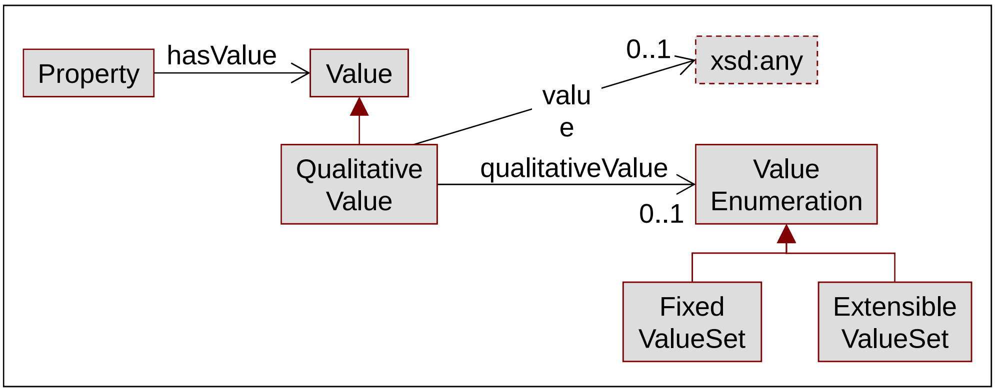

Fig. 17: Representation of qualitative values

The actual value of a *qualitative value* can be specified in two alternative ways: either by means of the attribute value, which accepts any string input, or by referring to an instance of the class *value enumeration* via the relation qualitativeValue. A *value enumeration* defines a (finite) set of possible values, which may be assigned to different *qualitative values*. The *value enumeration* class is derived from the meta class *feature space* and can be either a *fixed value set* or an *extensible value set*:

- A *fixed value set* is a specialization of the meta class *value set*. It is uniquely defined by an exhaustive enumeration of its instances. Thus, the number of possible values is fixed.
- An *extensible value set* is a specialization of the meta class *non-exhaustive value set*. Unlike a *fixed value set*, it is not defined by an (exhaustive) enumeration of its instances. Thus, the number of possible values may change at run time.

Like every other *value*, a *qualitative value* can be related to a backdrop *value*. Fig. 18 provides the example of a chameleon, whose skin color is observed against a temporal backdrop.

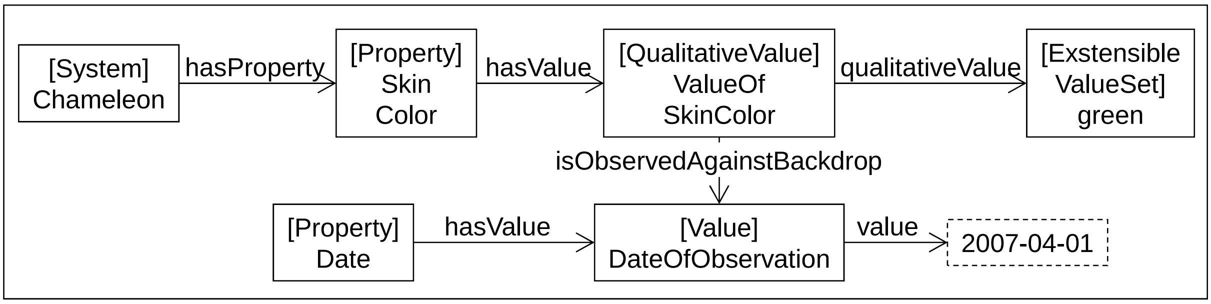

Fig. 18: Application example of a qualitative value

At first sight, the representation of a qualitative value may seem unnecessarily complicated as it requires an instantiation of both the *qualitative value* class and the *value enumeration* class. Yet both classes are required for the complete specification of the qualitative value: While the *value enumeration* class represents the actual value, the *qualitative value* class serves the function of correlating the actual value with the corresponding backdrop value. A combination of these two functions into a single class is not possible, since the instances of *value enumeration* must not be the origin of a relation (cf. the discussion on feature values in the Meta Model). However, in cases where the specification of a backdrop is not required, the value representation can be simplified, as will be explained in the following section

2.14. The hasCharacteristic Relation

Generally, the characterization of a *system* through *properties* and their *values* is fairly complex, requiring the concatenation of several concepts: First, the *property* class must be instantiated and linked to the *system* via a hasProperty relation; only then can the *value* be specified and assigned to the *property* by means of the hasValue relation. Such a ‘chain of concepts’ is indispensable for representing *properties* that take multiple *values*, as explained in the previous sections. However, in the case of a *constant property* having only a single *value*, the function of the *constant property* is reduced to that of a binary relation relating the *value* to the *system*. Hence, one may use a shorthand notation instead. To this end, the relation hasCharacteristic is introduced. Via this relation, the *values* of *constant properties* can be directly assigned to a *system*, thus substituting the *constant property*.

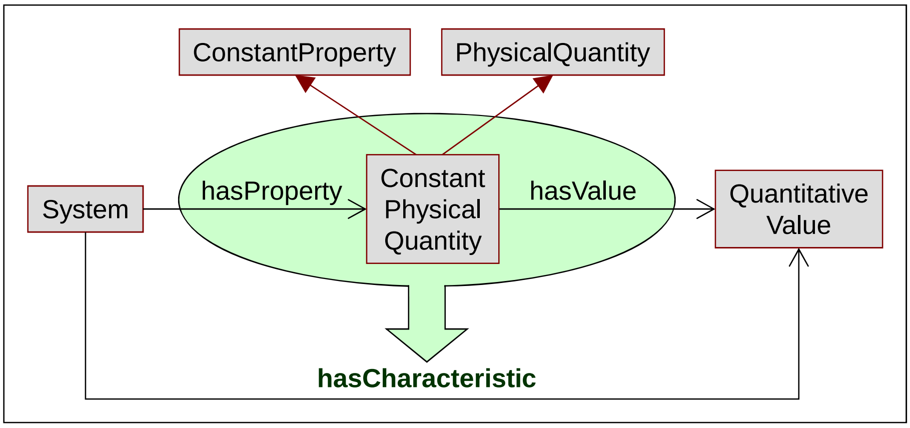

Fig. 19: Shorthand notation for constant *physical quantities*

Two cases must be distinguished:

- If the *constant property* is a *physical quantity*, hasCharacteristic replaces the concepts hasProperty, *physical quantity*, and hasValue (cf. Fig. 19).
- If the *constant property* has a *qualitative value*, the relation additionally substitutes the concepts *qualitative value* and the relation qualitativeValue, thus referring directly to the *value enumeration* (cf. Fig. 20).

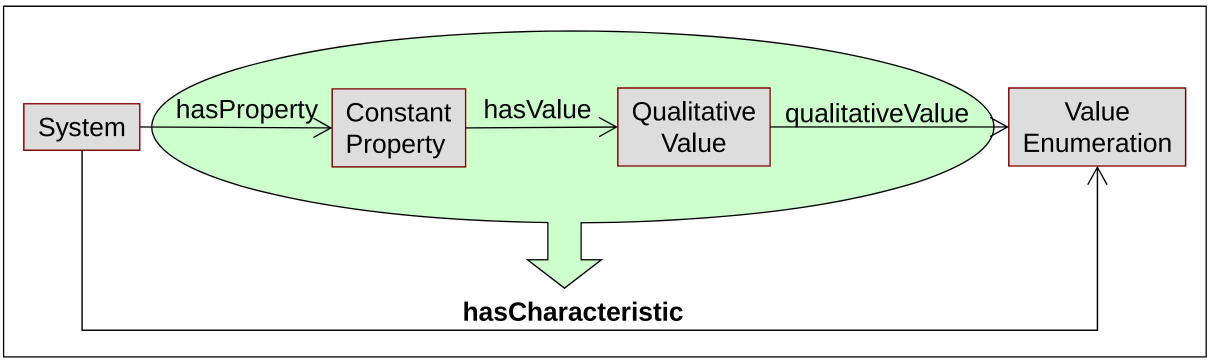

Fig. 20: Shorthand notation for *constant properties* with *qualitative values*

Finally some remarks on the usage of the introduced primatives:

- Just like the *property* class can be specialized to represent specific types of *properties*, the hasCharacteric relation needs to be specialized to substitute these *properties*. For instance, to replace the *property 'height'*, the relation hasHeight may be introduced as a specialization of hasCharacteristic.

- Specializations of hasCharacteristic may be utilized to implicitly define polyhierarchies of classes (cf. Morbach et al., 2008a). In this case, the utilized relation should be declared to be a specialization of both the relation hasCharacteristic and the meta relation isOfType.
- The hasCharacterstic relation allows linking a single *value* instance to different *system* instances. This is exploited to relate the *value* of a *physical constant* to different *systems* (cf. Sect. 2.15).

2.15. Physical Constant

A *physical constant* is a special type of a *constant property* with a fixed (*scalar*) *value*. It is defined as a *physical quantity*, the *value* of which is believed to be both universal in nature and invariant over time. Examples are the elementary charge, the gravitational constant, Planck's constant, and the speed of light in the vacuum. Such specific constants are modeled as instances of the *physical constant* class.

Due to its universal nature, a *physical constant* cannot be owned by a specific *system* and thus must not be assigned to a *system* instance via the hasProperty relation. Instead, the hasCharacteristic relation is used to relate the *value* of the *physical constant* to a *system*. That way, the *physical constant* itself remains independent.

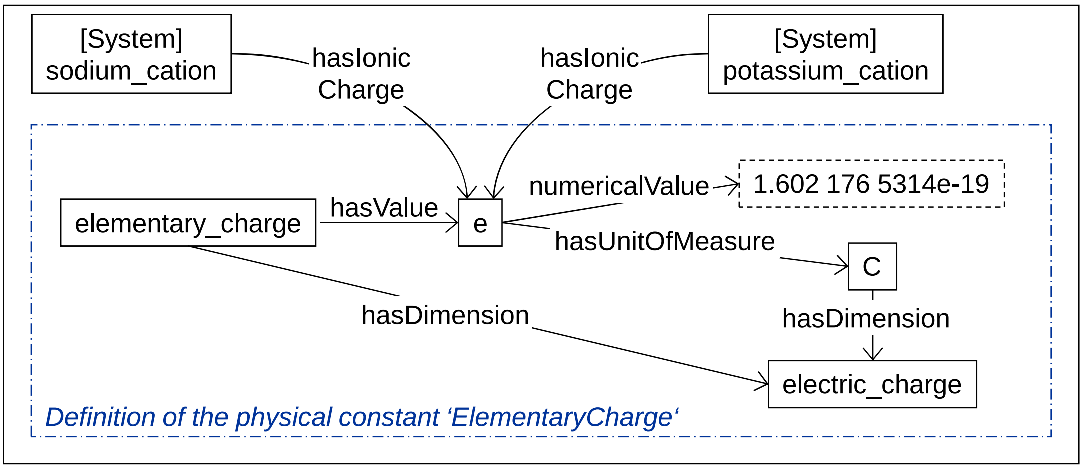

Fig. 21: Modeling of the elementary charge

Exemplarily, Fig. 21 illustrates the modeling of the elementary_charge as an instance of *physical constant*. The elementary_charge has a *physical dimension* of electric_charge; its *value* e equals 1.6021765314e-19 coulomb. By means of the relation hasIonicCharge (a specialization of hasCharacteristic), e can be assigned to different *systems*, such as the sodium_cation or the potassium_cation.

2.16. Internal and External Properties

According to axiom (5), not all the *properties* of a *system* need to be declared explicitly. Instead, they can be represented as *properties* of its constituent *subsystems*. We call those *properties* of a *system* that are explicitly assigned to the *system* the ‘external *properties*’ of the system. Accordingly, the ‘internal *properties*’ of a *system* are the external *properties* of its constituent *subsystems*.

The internal *properties* of a *system* can be inferred from the external *properties* of its *subsystems* by means of a reasoner. To this aim, one needs to define a query class, which subsumes the (external) *properties* of all *systems* that are subsystems of a given *system*. Such a query class must be individually defined for each *system* instance. An exemplary query class named ‘*internal properties*’ has been implemented in the formal specification of this ontology module. The query class retrieves the internal *properties* of a sample *system* S[18].

2.17. Property Set

A *property set* constitutes an (unordered) collection of *properties*, which may be of different types. The *properties* contained in a *property set* are identified via the relation comprisesDirectly, which is a specialization of the transitive relation comprises. These relations are defined analogously to the contains(Directly) relation between *supersystems* and *subsystems*, yet with their ranges and domains restricted to *properties*. Consequently, the comprises(Directly) relation is depicted by the same symbol as the comprises(Directly) relation: a white diamond with directed arrow (Fig. 22).

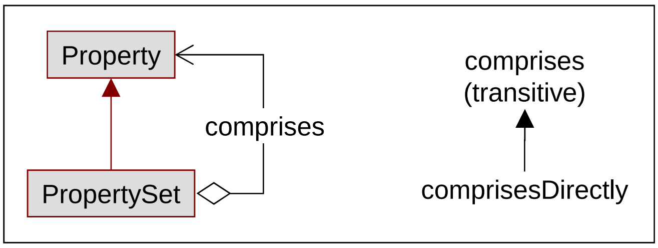

Fig. 22: Property set

A *property set* is itself a *property*; thus, a *property set* may comprise other *property sets*. However, a *property set* cannot have a *value* of its own.

Concept Descriptions

Individual concepts of the module *system* are defined below.

Classes

Aspect

Description

An *aspect* represents a particular viewpoint of a *system*. An instance of the *aspect* class explicitly denominates that viewpoint.

Relations

- *Aspect* is a specialization of the meta class *non-exhaustive value set*.

Aspect system

Description

An *aspect system* is an *exclusive subsystem* that contains those system components, relationships, and constraints that are of relevance to a particular *aspect*.

Definition

An *aspect system* is an *exclusive subsystem* that is considered under some *aspect*.

---

[18] A system can have both internal and external *properties* of the same type. For example, consider a *phase system*, which is composed of two *single phases*. Both the overall *phase system* and the two *single phases* have a *property* of type *density*. However, their meanings are different: The external *property* of the *phase system* represents the (averaged) density of overall system, whereas the internal *properties* represent the densities of the constituent liquid phase and vapor phase, respectively.

Relations

- *Aspect system* is a subclass of *exclusive subsystem*.
- An *aspect system* is an *exclusive subsystem* that is considered under exactly one *aspect*.
- An *aspect system* represents an *aspect* of exactly one *composite system*.

Composite system

Description

A *composite system* is a *system* that is composed of other *systems*.

Definition

A *composite system* is composed of some *systems*.

Relations

- *Composite system* is a subclass of *supersystem*.
- *Composite system* is derived from the meta class *composite object*.
- A *composite system* can only be composed of *systems*.
- A *composite system* may have some *aspect systems*.

Constant property

Description

A *constant property* is a *property* that has exactly one *value*.

Definition

See description.

Relations

- *Constant property* is a subclass of *property*.
- A *constant property* has exactly one *value*.

Elementary system

Description

An *elementary system* is a *subsystem* that cannot further partitioned into *subsystems*.

Definition

An *elementary system* is a *subsystem* that is not a *supersystem*.

Relations

- *Elementary system* is a subclass of *subsystem*.
- *Elementary system* is derived from the meta class *part only*.
- An *elementary system* cannot have direct subsystems.
- An *elementary system* can only be directly connected to *elementary systems*.

Exclusive subsystem

Description

An *exclusive subsystem* is a direct subsystem of a *composite system*; it cannot be a direct subsystem of any other *system*.

Definition

An *exclusive subsystem* is exclusively a subsystem of some *system*.

Relation

- *Exclusive subsystem* is a subclass of *subsystem*.
- *Exclusive subsystem* is derived from the meta class *part of composite object*.
- An *exclusive subsystem* can only be the exclusive subsystem of a *system*.

Extensible value set

Description

An *extensible value set* is a *value enumeration* which, unlike a *fixed value set*, is not defined by an (exhaustive) enumeration of its instances. Thus, the number of possible values may change at run time.

Relations

- *Extensible value set* is a subclass of *value enumeration*.
- *Extensible value set* is derived from the meta class *non-exhaustive value set*.

First-level subsystem

Description

A *subsystem* at the first level of decomposition.

Definition

A *subsystem* that is a direct subsystem of a *top-level system*.

Relations

- *First-level subsystem* is a subclass of *subsystem*.
- *First-level subsystem* is derived from the meta class *first-level part*.
- A *first-level subsystem* can only be directly connected to a *first-level subsystem*.

Fixed value set

Description

A *fixed value set* is a *value enumeration* that is defined by an exhaustive enumeration of its instances. Thus, the number of possible values is fixed.

Relations

- *Fixed value set* is a subclass of *value enumeration*.
- *Fixed value set* is derived from the meta class *value set*.

Internal properties

Description

The ‘internal *properties*’ of a *system* are the *properties* of its constituent *subsystems*. They can be specified by means of a query class and thus inferred by a reasoner. Such a query class must be defined individually for each *system* instance. The query class ‘*internal properties*’ exemplarily demonstrates this approach for a sample *system* S.

Definition

The *internal properties* of the *system* instance S are equivalent to the *properties* of the *subsystems* of S.

Model

Description

A *model* is a system that is used to enable the understanding of or the command over the original system, or to replace the original system. Model system and original system share certain characteristics that are of relevance to the task at hand (Wüsteneck 1963).

Definition

A *model* is a *system* that models some other *system*.

Relations

- *Model* is a subclass of *system*.
- A *model* models some *system*.
- A *model* models only *systems*.

Physical constant

Description

A *physical constant* is a *scalar quantity*, the *value* of which is believed to be both universal in nature and invariant over time. Examples are the elementary charge, the gravitational constant, Planck's constant, and the speed of light in the vacuum.

Relations

- *Physical constant* is a subclass of *scalar quantity*.
- A *physical constant* has exactly one *scalar value*.
- A *physical constant* cannot be a property of a particular *system*.

Usage

A *physical constant* must not be assigned to a *system* instance via the hasProperty relation. Instead, the hasCharacteristic relation is used to relate the *value* of the *physical constant* to a *system*.

Physical dimension

Description

A *physical dimension* is a characteristic associated with *physical quantities* and *units* of measure for purposes of organization or differentiation. Mass, length, and force are exemplary instances of *physical dimension*.

Relations

- *Physical dimension* is a specialization of the meta class *non-exhaustive value set*.

Physical quantity

Description

A *physical quantity* is a *property* that has quantifiable *values*. The concept includes scalars as well as vectors and higher-order tensors. Moreover, it comprises both physical quantities, such as mass or velocity, and nonphysical quantities, such as amount of money or rate of inflation.

Definition

A *physical quantity* is a *property* that has a *physical dimension*.

Relations

- *Physical quantity* is a subclass of *property*.
- A *physical quantity* has exactly one *physical dimension*.
- A *physical quantity* can only have *quantitative values*.

Property

Description

The *property* class represents the individual properties (traits, qualities) of a *system*, which distinguish the *system* from others. Typical examples are *size, color*, or *weight*, which are modeled as subclasses of *property*.

Relations

- *Property* is a specialization of the meta class *object*.
- A *property* may have some *values*.
- A *property* can only be a property of a *system*.
- A *property* cannot be a property of more than one *system*.

Usage

The subclasses of *property* represent *general* properties, which exist autonomously (i.e., independent of a particular *system*). The *individual* property of a *system* is modeled by (1) instantiating the respective subclass of *property* and (2) linking that *property* instance to the *system*. As soon as the *property* instance is linked to a *system*, it represents an inherent quality of that particular *system* and thus must not be assigned to any other *system*.

Property set

Description

A *property set* constitutes an (unordered) collection of *properties*, which may be of different types.

Definition

A *property set* is a property that directly comprises some *properties*.

Relations

- *Property set* is a subclass of *property*.
- A *property set* directly comprises only *properties*.
- A *property set* does not have a *value*.

Qualitative value

Description

A *qualitative value* is a *value* that is not (numerically) quantifiable.

Relations

- *Qualitative value* is a subclass of *value*.
- A *qualitative value* has either one entry to the value attribute or it refers to one *value enumeration* via the relation qualitativeValue.

Quantitative value

Description

A *quantitative value* is the value of a *physical quantity*.

Definition

See description.

Relations

- A *quantitative value* can only be the value of a *physical quantity*.
- *Quantitative value* is a subclass of *value*.

Scalar quantity

Description

A *scalar quantity* is a scalar-valued *physical quantity*.

Relations

- *Scalar quantity* is a subclass of *physical quantity*.
- A *scalar quantity* can only have *scalar values*.

Scalar value

Description

A *scalar value* is the value of a *scalar quantity*.

Relations

- *Scalar value* is a subclass of *quantitative value*.
- A *scalar value* can only be the value of a *scalar quantity*.
- A *scalar value* may have one *unit of measure*.
- A *scalar value* has exactly one numericalValue.

Second-level subsystem[19]

Description

A *subsystem* at the second level of decomposition.

Definition

A *subsystem* that is a direct subsystem of a *first-level subsystem*.

Relations

- *Second-level subsystem* is a subclass of *subsystem*.
- *Second-level subsystem* is derived from the meta class *second-level part*.
- A *second-level subsystem* can only be directly connected to a *second-level subsystem*.

Subsystem

Description

A *subsystem* is a *system* that is a constituent of another *system*.

Definition

A *subsystem* is a *system* that refers to another *system* via the isSubsystemOf relation.

---

[19] This concept simply demonstrates that second, third, fourth, … level subsystems can be defined in an analogues manner to the *First-level subsystem*, if required.

Relations

- *Subsystem* is a subclass of *system*.
- *Subsystem* is derived from the meta class *part*.
- *Subsystem* can only be the subsystem of a *system*.

Supersystem

Description

A *supersystem* is a *system* that has some constituent *subsystems*.

Definition

A *supersystem* is a *system* that refers to another *system* via the hasSubsystem relation.

Relations

- *Supersystem* is a subclass of *system*.
- *Supersystem* is derived from the meta class *aggregate*.
- *Supersystem* can only have subsystems of type *system*.

System

Description

The *system* class denotes all kinds of systems, which may be physical or abstract.

Relations

- *System* is a specialization of the meta class *object*.
- A *system* may be related to other *systems*.
- A *system* may have some properties of type *property*.
- A *system* can only be modeled by a *model*.
- A *system* may contain some *systems*.
- A *system* may have some characteristics of type *scalar value* or *value enumeration*.

System environment

Description

The *immediate environment* of a given *system* S consists of all *systems* that are directly related to S. It can be specified by means of a query class. As the environment concept is relative, such a query class must be defined individually for each *system* instance. The query class *system environment* exemplarily demonstrates the approach for sample *system* S.

Definition

The immediate environment of the *system* instance S includes all *systems* that (1) are not S, (2) are not subsystems of S, (3) are directly related to S.

System interface

Description

The class *system interface* represents the interface through which a *system* can be connected to another *system*.

Relations

- *System interface* is a subclass of *exclusive subsystem*.

- *System interface* is derived from the meta class *connector*.
- A *system interface* can only be directly connected to a *system interface*.
- A *system interface* can be directly connected to one *system interface*, at most.

Usage

Typically, some constraints are imposed on a *system interface* to ensure that it can only be connected to *system interfaces* with matching properties.

Top-level system

Description

A *top-level system* is a *supersystem* that is not a constituent of some other *system*.

Definition

A *top-level system* is a *supersystem* that is not a *subsystem*.

Relations

- *Top-level system* is a subclass of *supersystem*.
- *Top-level system* is derived from the meta class *aggregate only*.
- A *top-level system* cannot be a direct subsystem.
- A *top-level system* can only be directly connected to *top-level systems*.

Unit of measure

Description

A *unit of measure* is a standard measure for the *scalar value* of *physical quantity*, which has been adopted by convention.

Relations

- *Unit of measure* is a specialization of the meta class *non-exhaustive value set*.
- A *unit of measure* has exactly one *physical dimension*.

Value

Description

The *value* class denotes the different values of a *property*.

Relations

- A *value* can only be the value of a *property*.
- A *value* cannot be a value of more than one *property*.
- A *value* may be the backdrop of another *value*.
- A *value* may be observed against some backdrop *value*.

Value enumeration

Description

A *value enumeration* specifies the (finite) set of possible values of a *qualitative value*.

Definition

A *value enumeration* is either a *fixed value set* or an *extensible value set*.

Relations

- *Value enumeration* is a specialization of the meta class *feature space*.

Relations

comprises

Description

The relation comprises indicates the members of a *property set*.

Characteristics

- Derived from the meta relation hasPart
- Domain: *property set*
- Range: *property*
- Transitive

comprisesDirectly

Description

The relation comprisesDirectly indicates the direct members of a *property set*.

Characteristics

- Specialization of comprises
- Derived from the meta relation hasDirectPart
- Domain: *property set*
- Range: *property*

contains

Description

The contains relation constitutes an alternative to the hasSubsystem relation. It should be used instead of hasSubsystem

- if the hasSubsystem relation causes performance problems, or
- if only one side of the aggregation relation is of interest, namely the indication of the constituting elements of a *supersystem*.

Characteristics

- Specialization of isRelatedTo
- Derived from the meta relation hasPart
- Domain: *supersystem*
- Range: *system*
- Transitive

containsDirectly

Description

The relation containsDirectly is an alternative to the hasDirectSubsystem relation. It should be used instead of hasDirectSubsystem

- if the hasDirectSubsystem relation causes performance problems, or

- if only one side of the aggregation relation is of interest, namely the indication of the direct constituents *supersystem*.

Characteristics

- Specialization of contains and isDirectlyRelatedTo
- Derived from the meta relation hasDirectPart
- Domain: *supersystem*
- Range: *system*

hasAspectSystem

Description

The relation hasAspectSystem designates the *aspect systems* of a *system*.

Characteristics

- Specialization of isComposedOfSubsystem
- Domain: *system*
- Range: *aspect system*
- Inverse: representsAspectOf

hasCharacteristic

Description

The hasCharacteristic relation constitutes a shorthand notation for the specification of a *constant property* and its *value*.

Characteristics

- Domain: *system*
- Range: *value enumeration*, *value set*

hasDimension

Description

The relation hasDimension specifies the *physical dimension* of a *physical quantity* or a *unit of measure*.

Characteristics

- Domain: *property*, *unit of measure*
- Range: *physical dimension*
- Functional

hasDirectSubsystem

Description

The relation hasDirectSubsystem refers from a *supersystem* to its direct *subsystem*.

Characteristics

- Specialization of hasSubsystem and isDirectlyRelatedTo
- Derived from the meta relation hasDirectPart
- Domain: *supersystem*
- Range: *subsystem*

- Inverse: isDirectSubsystemOf

hasProperty

Description

The relation hasProperty indicates the *properties* of a *system*.

Characteristics

- Specialization of the meta relation inter-objectRelation
- Domain: *system*
- Range: *property*
- Inverse: isPropertyOf
- Inverse functional

hasSubsystem

Description

The relation hasSubsystem denotes the relation between a *supersystem* and its *subsystem*.

Characteristics

- Specialization of isRelatedTo
- Derived from the meta relation hasPart
- Domain: *supersystem*
- Range: *subsystem*
- Inverse: isSubsystemOf
- Transitive

hasUnitOfMeasure

Description

The relation hasUnitOfMeasure establishes the *unit of measure* of a *scalar value*.

Characteristics

- Specialization of the meta relation object-featureRelation
- Domain: *quantitative value*
- Range: *unit of measure*
- Functional

hasValue

Description

The hasValue relation designates the *values* of a *property*.

Characteristics

- Specialization of the meta relation inter-objectRelation
- Domain: *property*
- Range: *value*
- Inverse: isValueOf
- Inverse functional

isBackdropOf

Description

The isBackdropOf relation states that the *value* serves as a backdrop for the observation of some other *value*.

Characteristics

- Specialization of the meta relation inter-objectRelation
- Domain: *value*
- Range: *value*
- Inverse: isObservedAgainstBackdrop

isComposedOfSubsystem

Description

The relation isComposedOfSubsystem indicates the non-sharable, direct *subsystem* of a *supersystem*.

Characteristics

- Specialization of hasDirectSubsystem
- Derived from the meta relation isComposedOf
- Domain: *supersystem*
- Range: *subsystem*
- Inverse: isExclusivelySubsystemOf
- Inverse functional

isConsideredUnderAspectOf

Description

The relation isConsideredUnderAspectOf indicates the type of an *aspect system* by referring to an instance of the *aspect* class.

Characteristics

- Specialization of the meta relation object-featureRelation
- Domain: *aspect system*
- Range: *aspect*

isConnectedTo

Description

The relation isConnectedTo represents topological connectivity between *systems.*

Characteristics

- Specialization of isRelatedTo
- Derived from the meta relation isConnectedTo
- Domain: *system*
- Range: *system*

- Symmetric
- Transitive

isDirectlyConnectedTo

Description

The relation isDirectlyConnectedTo denotes the direct topological connectedness of two *systems.*

Characteristics

- Specialization of isConnectedTo and isDirectlyRelatedTo
- Domain: *system*
- Range: *system*
- Symmetric

isDirectlyRelatedTo

Description

The relation isDirectlyRelatedTo subsumes all kinds of direct inter-system relations.

Characteristics

- Specialization of the meta relation isRelatedTo
- Domain: *system*
- Range: *system*
- Symmetric

isDirectSubsystemOf

Description

The relation isDirectlySubsystemOf links a *subsystem* to its direct *supersystem*.

Characteristics

- Specialization of isSubsystemOf and isDiretlyRelatedTo
- Derived from meta relation isDirectPartOf
- Domain: *subsystem*
- Range: *supersystem*
- Inverse: hasDirectSubsystem

isExclusivelySubsystemOf

Description

The relation isExclusivelySubsystemOf links a non-sharable *subsystem* to its direct *supersystem*.

Characteristics

- Specialization of isDirectSubsystemOf
- Derived from meta relation isExclusivleyPartOf
- Domain: *subsystem*
- Range: *supersystem*
- Inverse: isComposedOfSubsystem

isModeledBy

Description

The relation isModeldBy points from a modeled *system* to its *model*.

Characteristics

- Specialization of isDirectlyRelatedTo
- Domain: *system*
- Range: *model*
- Inverse: models

isObservedAgainstBackdrop

Description

The isObservedAgainstBackdrop relation maps a *value* against a backdrop *value*.

Characteristics

- Specialization of the meta relation inter-objectRelation
- Domain: *value*
- Range: *value*
- Inverse: isBackdropOf

isPropertyOf

Description

The relation isPropertyOf links a *property* instance to a *system* instance.

Characteristics

- Specialization of the meta relation inter-objectRelation
- Domain: *property*
- Range: *system*
- Inverse: hasProperty
- Functional

isRelatedTo

Description

The relation isRelatedTo subsumes all kinds of inter-system relations.

Characteristics

- Specialization of the meta relation inter-objectRelation
- Domain: *system*
- Range: *system*
- Symmetric
- Transitive

isSubsystemOf

Description

The relation isSubsystemOf refers from a *subsystem* to its *supersystem*.

Characteristics

- Specialization of isRelatedTo
- Derived from the meta relation isPartOf
- Domain: *subsystem*
- Range: *supersystem*
- Inverse: hasSubsystem

isValueOf

Description

The relation isValueOf assigns a *value* to a *property*.

Characteristics

- Specialization of the meta relation inter-objectRelation
- Domain: *value*
- Range: *property*
- Inverse: hasValue
- Functional

models

Description

The relation models links a *model* to the modeled *system*.

Characteristics

- Specialization of isDirectlyRelatedTo
- Domain: *model*
- Range: *system*
- Inverse: isModeldBy

qualitativeValue

Description

The relation qualitativeValue specifies the actual value of a *qualitative value*.

Characteristics

- Specialization of the meta relation object-featureRelation
- Domain: *qualitative value*
- Range: *value enumeration*

representsAspectOf

Description

The relation representsAspectOf links an *aspect system* to its respective *system*.

Characteristics

- Specialization of isExclusivelySubsystemOf
- Domain: *aspect system*
- Range: *system*

- Inverse: hasAspectSystem

Attributes

numericalValue

<u>Description</u>

The attribute numericalValue specifies the number part of a *quantitative value*.

<u>Characteristics</u>

- Domain: *scalar value*
- Range: any built-in XML Schema Datatype. Ordinarily, the values of numericalValue are of type float; however, other XML Schema Datatypes are also possible, such as dateTime.

value

<u>Description</u>

The value attribute holds the actual value of a *qualitative value*.

<u>Characteristics</u>

- Domain: *qualitative value*
- Range: any built-in XML Schema Datatype

Instances

S

<u>Description</u>

Sample *system*; used for the construction of the query classes *system environment* and *internal properties*.

<u>Characteristics</u>

- Instance of *system*

3. Network System

The ontology module *network_system* introduces a structured representation for complex *systems*, which is applicable in such different domains as biology, sociology, and engineering. The common strategy of these disciplines is to represent the system as a *network*. In this context, a network is understood as a modular structure that "is determined on hierarchical ordered levels by coupling of components and linking elements" (Gilles, 1998). Thus, the representation of network systems calls for two different mechanisms: the mereological decomposition of systems and the topological ordering of the system components.

The concepts required for the mereological decomposition of *systems* are provided by the ontology module *system*, which allows for the structuring of *systems* into *subsystems* across multiple levels of hierarchy (cf. Sect. 2.4). Hence, what remains to be done is to introduce concepts for the topological organization of the system components. To this aim, we adopt the design pattern for the representation of graphs that was defined in the ontology module *topology* of the Meta Model (cf. Morbach et al., 2008a). Hence, *network system* is introduced as a specialization of *system* incorporating mereological as well as topological considerations. According to the design pattern, graphs are represented through *nodes* and connecting *arcs*, where an *arc* may or may not be directional. Additionally, *ports* and *connection points* may be used to further specify the connectivity between *nodes* and *arcs*.

Applying this design pattern to the representation of network systems, two special types of *systems*, *device* and *connection*, are introduced. Hence, a *network system* is composed of at least one *device* and one *connection* as shown in Fig. 23. *Device* and *connection* correspond to the meta classes *node* and *arc*, respectively, and are defined equivalently. Additionally, a *directed connection* is established as a subclass of *connection*.

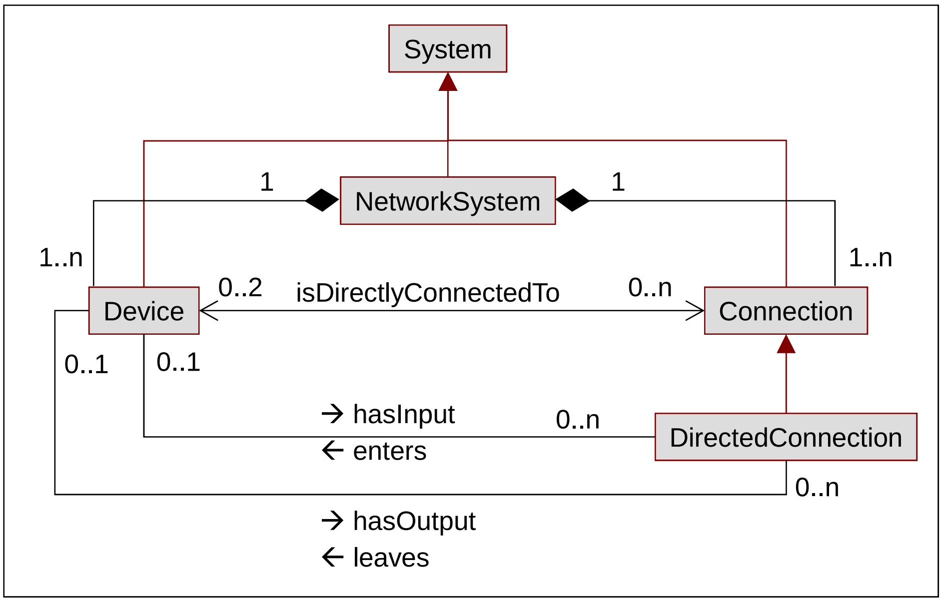

Fig. 23: Connectivity of *devices* and *connections*

The relation isDirectlyConnectedTo, previously established in the *system* module (cf. Sect. 2.5), is utilized to couple a *connection* with a *device*. For linking a *directed connection* to a *device*, the relations enters and leaves are to be used, which are defined analogously to the Meta Model (cf. Fig. 24).

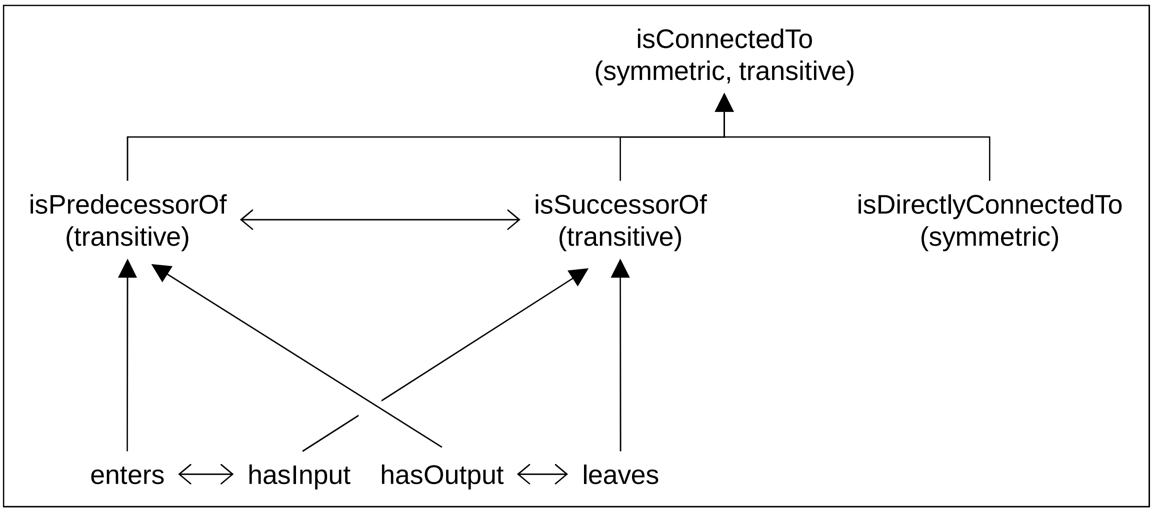

Fig. 24: Hierarchy of topological relations

So far, we have considered only such *connections* that are connected to exactly two *devices*. Another special case of *connection* is the single-edge *connection*, which is directly connected to only a single *device*. We denote such a class as *environment connection* because it represents the connectivity of a *network system* with its (not explicitly specified) *environment* (cf. Fig. 25).

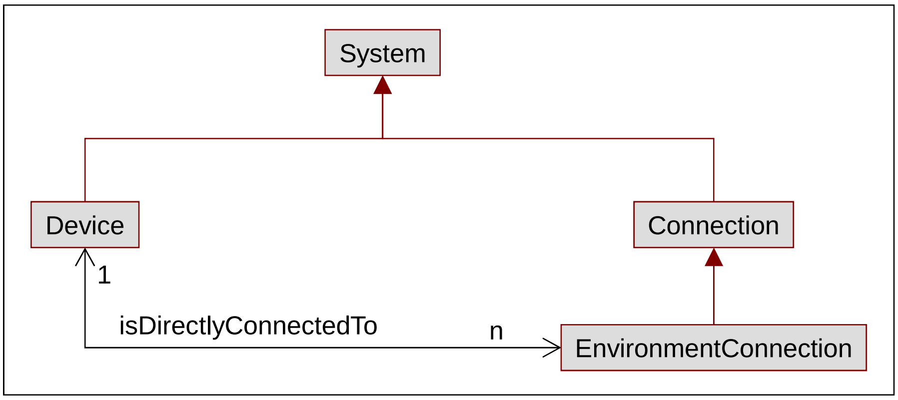

Fig. 25: Connectivity of *environment connection*

Ports and *connection points* are introduced as special types of *system interfaces* (Fig. 26). Just like in the Meta Model, *ports* and *connection points* represent the interfaces of the *devices* and *connections*. Their characteristics need to match in order to realize a valid coupling (cf. Morbach et al., 2008a).

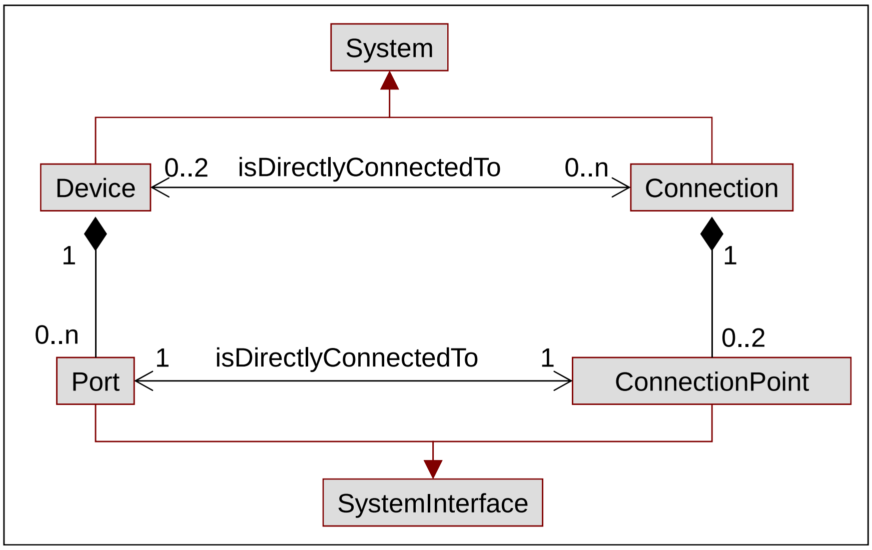

Fig. 26: *Ports* and *Connection points*

The decomposition of *devices* and *connections*, depicted in Fig. 27, is governed by the following regulations:

- *Devices* can only have direct subsystems of type *device*, *connection*, or *port*.
- *Connections* can only have direct subsystems of type *device*, *connection*, or *connection point*.
- If a *device* is decomposed into a number of sub-*devices*, then these sub-*devices* must be connected by *connections*. Thus, a *device* needs to be decomposed into two *devices* and one intermediate *connection*, at least.

Similarly, if a *connection* is decomposed into sub-*connections*, then there must be *devices* in between the sub-*connections*. Thus, a *connection* needs to be decomposed into two *connections* and one intermediate *device*, at least.

The aforementioned regulations are derived from the decomposition rules for *nodes* and *arcs* established in the Meta Model. For details on this issue, refer to Morbach et al. (2008a).

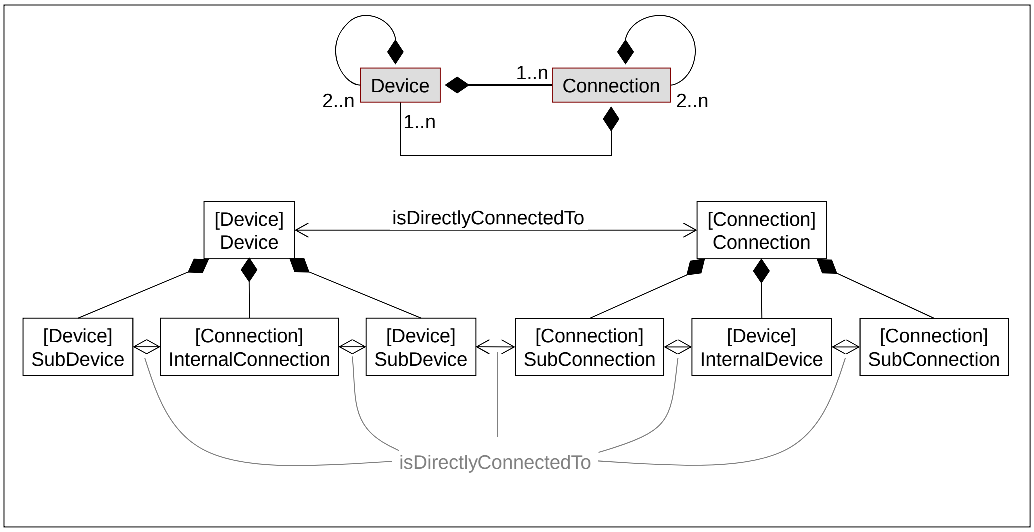

Fig. 27: Decomposition of devices and connections

Finally, we define a *network system* as a *system* that is composed of some *devices* and *connections*.

Usage

A large number of real-world systems can be modeled as *network systems*: technical systems (Alberts, 1994; Marquardt, 1996; Marquardt et al., 2000), physico-chemical systems (e.g., Marquardt, 1992; Marquardt, 1994, Marquardt, 1995; Gilles, 1998), biological systems (e.g., Mangold et al., 2005), economic systems (e.g., Andresen, 1999), social systems (e.g., Bunge, 1979), and others. Generally, the *devices* are the crucial elements of a *network system* and hold the major functionality, while the *connections* represent the linkages between the *devices*.

To enhance the understanding for the applicability of network systems, three examples of describing real-world systems as network systems are discussed subsequently:

- Marquardt (1992) and Gilles (1998) propose a framework for the development of mathematical models for physico-chemical systems, wherein *devices* and *connections* represent the individual model building blocks. Within the modeling framework, only the *devices* have the capability for the accumulation and/or change of extensive physical quantities, such as energy, mass, and momentum. The *connections*, on the other hand, describe the fluxes of quantities that are interchanged between the *devices*; different types of fluxes can be modeled this way – of matter (e.g., material flow through a pipe), energy (e.g., heat conduction through a wall), momentum (e.g., shock wave in a fluid medium).
- Network systems are particularly suitable for the representation of process flowsheets. For example, consider a Block Flow Diagram (BFD), which is used to specify the conceptual design of a chemical process: The individual process units (unit operations) can be considered as *devices*, and the material and energy streams that are exchanged between the units can be represented as *connections*. Another example is the Piping & Instrumentation Diagram (P&ID) applied in basic and detail engineering: Here, the apparatuses and machines are modeled as *devices*, while *connections* represent the pipes (for materials and utilities) and the power supply lines.
- In the area of control theory, the control components (controller, sensor, controlled system,…) can be modeled as *devices*, while the *connections* represent the signal lines that transmit information between the control components (Bayer et al., 2001).

Concept Descriptions

Individual concepts of the module *network_system* are defined below.

Classes

Connection

Description

Connections are those elements of a *network system* that represent the linkages between the *devices*.

Relations

- *Connection* is a subclass of *system*.
- *Connection* is derived from the meta class *arc*.
- A *connection* can only be directly connected to a *device*.

- A *connection* cannot be directly connected to more than two *devices*.
- A *connection* can only be a direct subsystem of a *device* or *connection*.
- A *connection* can only have *connections* or *device* or *connection points* as a direct subsystem.
- A *connection* can only have *connections* or *device* or *connection points* or *ports* as subsystems.

Usage

A *connection* can be decomposed into a required number of sub-*connection*. However, these sub-*connections* need to be connected by *devices* again. Thus, a *connection* must be decomposed into three subsystems, at least: two *connections* and one *device*. Unfortunately, this decomposition axiom cannot properly represented in OWL, as OWL does currently not support qualified cardinality restrictions.

Connection point

Description

A *connection point* represents the interface through which a *connection* can be connected to the *port* of a *device*. *Connection points* may have certain attributes that further specify the type of connection. *Connection points* are subsystems of the corresponding *connection* or *directed connection*, respectively.

Relations

- *Connection point* is a specialization of *system interface*.
- *Connection point* is derived from the meta class *connection point*.
- A *connection point* can only be directly connected to a *port*.
- A *connection point* cannot be connected to more than one *port*.
- A *connection point* is a subsystem of at least one *connection*.
- A *connection point* can only be a direct subsystem of a *connection*.

Usage

Connection points constrain the connections that a *connection* can have to a *device*. A connection between a *connection* and a *device* is feasible only if the values of the attributes describing the *connection point* matching the corresponding attributes of a *port*.

Device

Description

Devices are the crucial elements of a *network system*, holding the major functionality.

Relations

- *Device* is a subclass of *system*.
- *Device* is derived from the meta class *node*.
- A *device* may have only *directed connections* as inputs.
- A *device* may have only *directed connections* as outputs.
- A *device* can only be directly connected to *connections*.
- A *device* can only be a direct subsystem of a *device* or *connection*.
- A *device* can only have *devices* or *connections* or *ports* as direct subsystems.

- A *device* can only have *devices* or *connection* or *ports* or *connection points* as subsystems.

Usage

A *connection* can be decomposed into a required number of sub-*connection*. However, these sub-*connections* need to be connected by *devices* again.

A *device* can be decomposed into a required number of sub-*devices*. However, these sub-*devices* need to be connected by *connections* again. Thus, a *device* must be decomposed into three subsystems, at least: two *devices* and one *connection*. Unfortunately, this decomposition axiom cannot properly represented in OWL, as OWL does currently not support qualified cardinality restrictions.

Directed Connection

Description

Directed connection is a specialization of *connection* and represents likewise the connecting element between *devices*. However, the usage of *directed connection* implies the indication of a directed interconnection.

Relations

- *Directed connection* is a subclass of *connection*.
- *Directed connection* is derived from the meta class *directed arc*.
- A *directed connection* enters either zero or one *device*.
- A *directed connection* leaves either zero or one *device*.
- A *direct connection* cannot be directly connected to another *system*.
- The isDirectlyConnectedTo relation is not applicable to a *directed connection*.

Environment Connection

Description

Environment connection is a specialization of *connection* and represents a single-edge connection to exactly one *device*. Thus, special connections like system inputs or outputs may be represented for not explicitly defined environments.

Relations

- *Environment connection* is a subclass of *connection*.
- An *environment connection* is connected directly to exactly one *device*.

Network system

Description

A *network system* is a *system* that is composed of *connections* and *devices*.

Definition

A *network system* is a *system* that is composed of some *connections* and some *devices*.

Relations

- *Network system* is a specialization of *composite system*.
- A *network system* is composed of some *connections* and some *devices*.

- A *network system* is composed of only *connections* or *devices*
- A *network system* is itself either a *connection* or a *device*.

Port

Description

Ports represents the interfaces through which *devices* are connected to *connections*. A *port* may have certain attributes that characterize the type of the connection.

Relations

- *Port* is a specialization of *system interface*.
- *Port* is derived from the meta class *port*.
- A *port* can only be directly connected to a *connection point*.
- A *port* cannot be directly connected to more than one *connection point*.
- A *port* is subsystem of at least one *device*.
- A *port* can only be a direct subsystem of a *device*.

Usage

Ports constrain the number and type of connections that a *device* can have: A *device* can only be connected to as many *connections* as it has designated *ports*. Moreover, a connection between a *device* and a *connection* is feasible only if the attributes of the *port* and the corresponding *connection point* match.

Relations

enters

Description

The relation enters interconnects an ingoing *directed connection* to its target *device*.

Characteristics

- Specialization of isPredecessorOf
- Derived from the meta relation enters
- Domain: directed *connection*
- Range: *device*
- Inverse: hasInput

hasInput

Description

The relation hasInput connects a *device* to an incoming *directed connection*.

Characteristics

- Specialization of isSuccessorOf.
- Derived from the meta relation hasInput

- Domain: *device*
- Range: *directed connection*
- Inverse: enters

hasOutput

Description

The relation hasOutput connects a *device* to an outgoing *directed connection*.

Characteristics

- Specialization of isPredecessorOf
- Derived from the meta relation hasOutput
- Domain: *device*
- Range: *directed connection*
- Inverse: leaves

isSuccessorOf

Description

The relation isSuccessorOf identifies all *devices* and *directed connections* that are successors of the considered one.

Characteristics

- Specialization of isConnectedTo
- Derived from the meta relation isSuccessorOf
- Domain: *device* or *directed connection*
- Range: *device* or *directed connection*
- Inverse: isPredecessorOf
- Transitive

isPredecessorOf

Description

The relation isPredecessorOf identifies all *devices* and *directed connections* that are predecessors of the considered one.

Characteristics

- Specialization of isConnectedTo
- Derived from the meta relation isPredecessorOf
- Domain: *device* or *directed connection*
- Range: *device* or *directed connection*
- Inverse: isSuccessorOf
- Transitive

leaves

Description

The relation leaves connects an outgoing *directed connection* to its source *device*.

Characteristics

- Specialization of isSuccessorOf
- Derived from the meta relation leaves
- Domain: *directed connection*
- Range: *device*
- Inverse: hasOutput

sameAs

Description

The relation denotes a correspondence between a *connection* and its placeholder in a decomposition hierarchy.

Characteristics

- Specialization of isDirectlyRelatedTo
- Derived from the meta relation sameAs
- Domain: *connection*
- Range: a *connection* that is directly connected to a *device*
- Symmetric

4. Technical System

The ontology module *technical_system* introduces the class *technical system* as a special type of a *system* which has been developed through an (engineering) design process. The criterion to qualify as a *technical system* is "to be designed in order to fulfill some required function" (Bayer, 2003). Thus, the *technical system* concept may denote all kind of technical artifacts, such as chemical plants, cars, computer systems, or infrastructure systems like a sewage water system. But also non-technical artifacts like chemical products and even non-physical artifacts, such as software programs or mathematical models, can be considered as *technical systems*.

For a comprehensive description of a *technical system*, five designated viewpoints are of major importance (Bayer, 2003): the system *requirements*, the *function* of the system, its *realization*, the *behavior* of the system, and the *performance* of the system. These five viewpoints are explicitly modeled in this ontology module, as will be explained in the following sections: In Sects. 4.1 to 4.5, the precise meaning of the respective viewpoints will be clarified. In the subsequent Sect. 4.5, the implementation of these viewpoints as specialized *aspect systems* (cf. Sect. 2.7) will be described. Lastly, Sect. 4.6 discusses the interrelations between the different *aspect systems*.

Before going into details, it should be mentioned that the concepts provided by this module may be used to describe the 'as-is' state (i.e., the current status) of a *technical system* as well as its 'to-be' state[20] (future state, nominal state). Yet while the concepts are usable for both the 'as-is' case and the 'to-be' case, the two cases are not explicitly distinguished within the current version of OntoCAPE. Thus, it has to be deduced from context, which of the two cases prevails.

4.1. Function and Requirements

The ontological representation of *function* in design is a long-standing research issue. Various definitions of the function concept have been proposed in the literature; for a review of those, see for example Baxter et al. (1994); Chandrasekaran (1994); Bilgic and Rock (1997); Chandrasekaran and Josephson (2000); Szykman et al. (2001); and Kitamura and Mizoguchi (2003).

Here, we adopt the definition of Chandrasekaran and Josephson (2000), who define function as *desired behavior*. Thus, function is an abstraction of the *actual behavior* (cf. Sect. 4.3) insofar as only the desired effects are considered, whereas all the unwanted and/or side-effects are ignored.

According to Chandrasekaran and Josephson (2000), two interpretations of the function concept must be distinguished for a *technical system*: function seen from an *environment-centric* viewpoint and function seen from a *device-centric* viewpoint (in this context, 'device' is used synonymously with *technical system*). The former viewpoint reflects the desired effect that a *technical system* exerts on its environment, yet without considering how this effect is to be achieved; the latter viewpoint additionally incorporates the principle of function of the *technical system*.

In OntoCAPE, the class *system function* represents the device-centric viewpoint, while the environment-centric viewpoint is described through the class *system requirement*; both are subclasses of *aspect system*.

The environment-centric viewpoint (*system requirements*) is more abstract than the device-centric viewpoint (*system function*): *System requirements* can be stated without knowledge of their technical

[21] In a network system, the system requirements of a device can be indicated by specifying the change between (some of) the system inputs and (some of) the system outputs. This corresponds to the specification of generic process steps given in the CLiP model

realization; only the desired effect on the environment needs to be specified. The *system function*, on the other hand, specifies how the *technical system* fulfills the *system requirements*. Hence, the *conceptual design* of the *technical system* must be specified in terms of the underlying physicochemical or technical principles.

As an example, consider the design of a process unit. The *system requirements* can be stated by describing the effect that the process unit shall exert on the processed materials (e.g., to separate dispersed particles from a liquid"). Yet to specify the *system function*, one needs to consider the physical or technical principles based on which the desired effect is going to be achieved (e.g., decide whether the separation is realized by means of sedimentation, centrifugation, or filtration). Thus, "moving from an environment-centric functional description towards a device-centric description calls for partially solving the design problem" (Chandrasekaran and Josephson, 2000).

Clearly, the main use for the concepts of *system requirements* and *system function* is to specify the 'to-be' state of a *technical system* during its design phase. Usually, the *system requirements* are formulated first, specifying the desired effect of the *technical system* on the environment. Later, at the conceptual design stage, the *system requirements* are refined into *system functions*, particularizing the principle based on which the desired effect is to be accomplished.

In addition to that, the concepts of *system requirements* and *system function* may also be used to characterize the 'as-is' state of a *technical system*. Note, however, that the semantics differ slightly, depending on whether the 'as-is' state or the '"to-be"' state of the *technical system* is to be described:

- In the 'to-be' case, the *system requirements* and *system function* specify the planned desired behavior of the *technical system*, as, for example, envisioned in the early phases of the design process.
- In the 'as-is' case, the *system requirements* and *system function* provide an abstract (i.e., environment-centric or device-centric) description of the actual desired behavior.

In other words: the 'as-is' case describes the desired behavior that is effectively attainable under optimal conditions. Obviously, this may differ from the planned desired behavior reflected by the 'to-be' case. As an example, consider a chemical plant that has been designed for a nominal production capacity of 200,000 tons per year. After commissioning, however, it turns out that – due to some unforeseen problems – the actual production capacity is only 190,000 tons per year, at best. The nominal production capacity can be considered as the 'to-be' *system requirements*, whereas the actual production capacity can be considered as the 'as-is' *system requirements*.[21].

4.2. Realization

The realization aspect, represented through the class *system realization*, reflects the physical (or virtual) constitution of the *technical system*. In case of a physical system, the *system realization* describes the system's physical structure, including its geometrical and mechanical properties. For example, the *system realization* of a chemical process would comprise the equipment and machinery required for materials processing; the *system realization* of a chemical product would reflect its molecular structure, crystal morphology, etc. In case of a non-physical system (such as a computer program), the *system realization* reflects the logical or abstract structure of the system; also, it may describe the (physical) implementation of the non-physical system (e.g., the model equations of a mathematical model or the source code of a computer program). Generally, the *system realization* gives a static description of the *technical system*, as opposed to the *system behavior* (cf. next section), which describes its dynamic behavior. Consequently, a *system realization* has mostly *constant properties*, which are often represented in shorthand notation via the hasCharacteristic relation (cf. Sect. 2.14).

A *system realization* may describe the 'as-is' state of the *technical system* as well as its 'to-be' state. In the 'as-is' case, it is comparable to a technical documentation, which reflects the current state of the *technical system*. By contrast, the 'to-be' case is comparable to a technical specification, as it is typically created in an engineering design project to specify the *technical system* that is to be built. In this context, it is important to remember that a *system realization* holds only information pertaining to the system itself; information that specify how to realize a *technical system* (e.g., assembly instructions or production planning) do not form part of the *system realization*.

Note that a *system realization* can be specified on different levels of detail and abstraction. For example, the *system realization* of a chemical plant may be stated on the information level of a P&ID (which represents the major equipment items and their main dimensions, but no geometrical details) as well as on the more detailed information level provided by isometric drawings and 3D models.

4.3. Behavior

The class *system behavior* describes how a *technical system* operates under certain conditions. If the *technical system* is described 'as-is', the *system behavior* reflects the behavior that can be actually observed. In the 'to-be' case, the *system behavior* concept represents the predicted behavior, which may be estimated on the basis of experiments or mathematical models.

The *system behavior* can be described both quantitatively and qualitatively. A quantitative description is provided by the *values* of its *properties*, which must be distinguished by means of a suitable backdrop *property*, usually a *temporal coordinate*[22] (cf. Morbach et al., 2008d). This agrees well with the literature on dynamic systems (e.g., Föllinger, 1982), where the behavior of a system is often defined as the change of its states over time. According to Bayer et al. (2001), the *values* of one distinct *property* and their related (temporal) backdrop *values* can be considered as a *state variable* of the *technical system*. The *state* of a *technical system* is given by the totality of all state variables at one particular point in time. Thus, a state can be considered as a temporal snapshot of the *system behavior*, and the *system behavior* can be described by the sequence of its states over time.

A qualitative description of the *system behavior* can be obtained by indicating the system's characteristic *phenomena*. In this context, a *phenomenon* denotes a typical mode of behavior exhibited by the system. The specification of a phenomenon implies (1) the existence of certain *properties* associated with that particular mode of behavior, and (2) that the *values* of these *properties* follow a designated pattern. To give an example: the indication of the *physicochemical phenomenon* of laminar flow (cf. Sect. Wiesner et al., 2008) implies that (1) the *properties* '*velocity*' (or '*mass flow*'), '*viscosity*', and '*density*' are of relevance for describing the *system behavior*, and (2) that the *values* of these *properties* must comply with the laws of laminar flow. Thus, through the specification of the prevailing *phenomena*, the state of the *technical system* can be qualitatively defined[23].

4.4. Performance

The *system performance* is concerned with the evaluation and benchmarking of the *technical system*. The concept itself represents a performance measure for the evaluation. Different performance measures are possible, depending on the chosen evaluation criterion. Typical criteria would be safety, reliability, ecological performance, and economic performance; a typical performance measure for latter would be costs. The *system performance* can represent the predicted performance ('to-be' case) as well as the performance that is actually measured ('as-is' case).

---

[22] Of course, other choices of backdrop *properties* are also possible.

[23] Even for the specification of the quantitative behavior, it is advantageous to specify the *phenomena* first; afterwards, one may query the ontology for a list of relevant *properties* and physical *laws* associated with these *phenomena*.

Note that a *system performance* may evaluate only a particular aspect of the *technical system*: For example, construction costs measure the economic performance of a *system realization*, operating costs denote the economic performance of a *system behavior*, and a ranking of conceptual design alternatives corresponds to the performance evaluation of some *system function*.

4.5. Implementation of the Technical System in OntoCAPE

In OntoCAPE, the viewpoints of *system requirements*, *system function*, *system behavior*, *system realization* and *system performance* are modeled as subclasses of *aspect system*. Each *aspect system* is assigned an instance of the *aspect* class, which explicitly typifies the nature of the respective *aspect system*: For example, the *system function* is assigned the *aspect* of function (cf. Fig. 28).

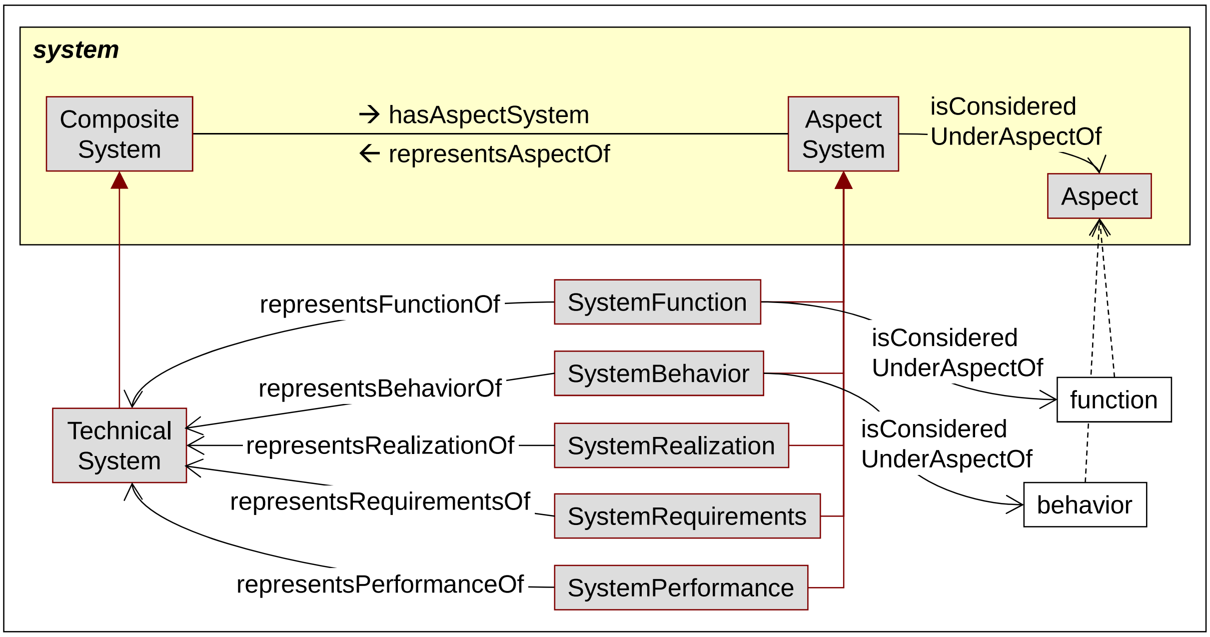

Fig. 28: The five major aspects of a technical system

The relationships between the *technical system* and its *aspect systems* are established via specializations of the relations hasAspectSystem and representsAspectOf, as indicated in Fig. 28 and Fig. 29.

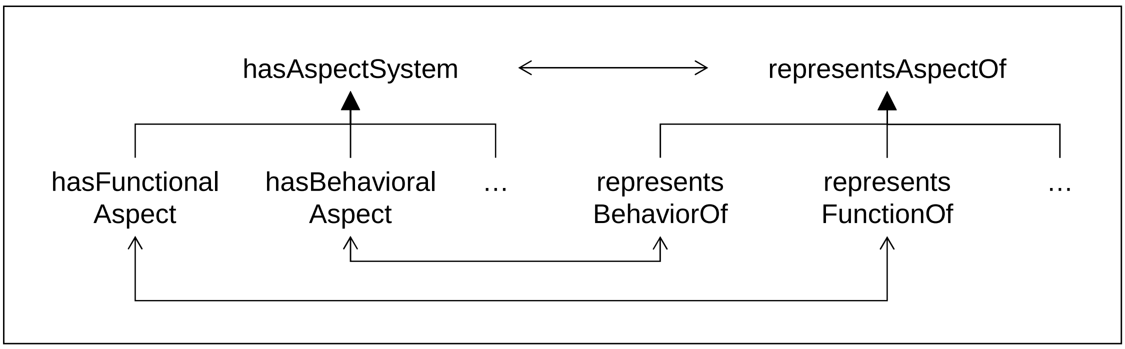

Fig. 29: Refinement of the hasAspectSystem relation

As explained above, the *system behavior* can be qualitatively described by indicating the relevant phenomena. This is modeled through the class *phenomenon*, which is assigned to a *system behavior* via the relation hasPhenomenon (Fig. 30).

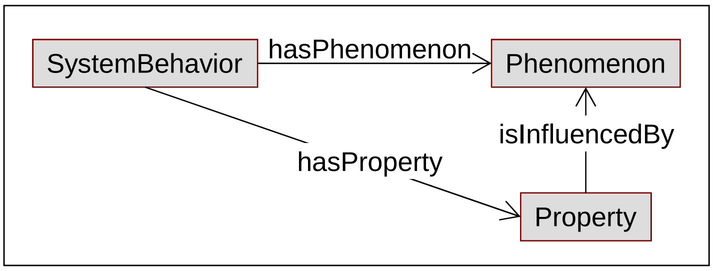

Fig. 30: Qualitative description of system behavior

The occurrence of a particular *phenomenon* exerts an influence on certain *properties*: For example, if the *phenomenon* of laminar flow is present, it will influence the *properties* '*velocity*' and/or '*mass flow*'; the *phenomenon* of chemical equilibrium has an influence on the *concentrations*, etc. These kinds of interdependencies can be modeled by means of the relation isInfluencedBy, which explicitly designates those *properties* that are influenced by a particular *phenomenon*.

4.6. Relations between Aspect Systems

Manifold relations and dependencies exist between the *aspect systems* of *technical system*. The type and the number of relations vary, depending on the respective application context. For example, the following relationships will arise in the course of a design project:

- In conceptual design, the *system requirements* are transformed into *system functions*.
- Later, the *system function* is detailed into the *system realization* at the stage of basic design.
- The *system realization* sets boundary conditions that constrain the possible *system behavior*.

Depending on the target application, an ontological model of these relations can turn very complex. For example, Kitamura and Mizoguchi (2003) present a fairly large ontology designated solely for modeling the interrelations between *system requirements* and *system functions*. According to the authors, this level of detail is required to provide adequate support for an intelligent design environment. So far, such applications have not been the focus of OntoCAPE; consequently, the inter-aspect relations are presently not modeled in detail. Fig. 31 presents some generic binary relations, which may be used to navigate between aspect systems; additional ones may be introduced if required.

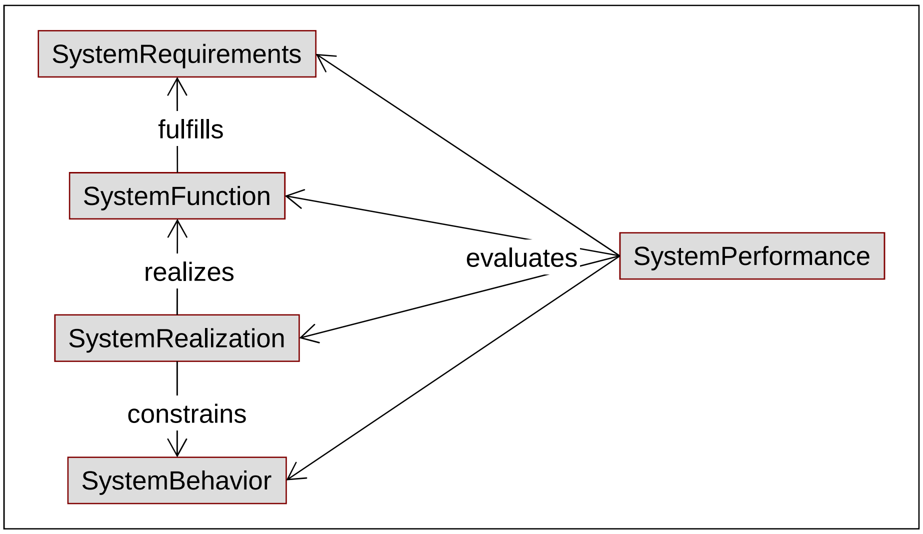

Fig. 31: Exemplary relation applied between *aspect systems*

Generally, the inter-aspect relations displayed in Fig. 31 are specializations of the isRelatedTo relation.

- *System requirements* and *system function* can be linked via the relations fulfills and its inverse isAchievedThrough, thus stating that a conceptual design solution fulfills a particular requirement.
- The relation realizes and its inverse isRealizedBy indicate that a particular *system realization* is able to implement some *system function*.
- The relations constrains and isConstrainedBy denote the restrictions on the *system behavior*, which are imposed by a *system realization*.
- Finally, the relation evaluates refers from a *system performance* to the *aspect system* the performance of which is measured; its inverse hasPerformanceMeasure points from the evaluated *aspect system* to the performance measure.

Concept Descriptions

Individual concepts of the module *technical_system* are defined below.

Classes

Phenomenon

<u>Description</u>

A *phenomenon* denotes a typical mode of behavior exhibited by a *technical system*, thus providing a qualitative description of a recurring *system behavior*.

<u>Relations</u>

- *Phenomenon* is derived from the meta *class non-exhaustive value set*.

Property (continued)

<u>Relations</u>

- A *property* may be influenced by *phenomenon*.

System behavior

Description

The *system behavior* describes how a *technical system* operates under certain conditions; this description can be of qualitative or quantitative nature.

Definition

A *system behavior* represents the behavioral aspect of a *technical system*.

Relations

- *System behavior* is a subclass of *aspect system*.
- A *system behavior* is considered under the aspect of behavior.
- A *system behavior* may have some *phenomena*.
- A *system behavior* may be constrained by a *system realization*.
- A *system behavior* may have some performance measure of type *system performance*.

System function

Description

A *system function* describes the desired behavior of a *technical system* from a device-centric perspective (cf. Chandrasekaran and Josephson 2000). To indicate the *system function* of a *technical system*, the conceptual design of the *technical system* must be specified in terms of the underlying physicochemical and/or technical principles.

Definition

A *system function* represents the functional aspect of a *technical system*.

Relations

- *System function* is a subclass of *aspect system*.
- A *system function* is considered under the aspect of function.
- A *system function* may fulfill some *system requirements*.
- A *system function* may be realized by a *system realization*.
- A *system function* may have some performance measure of type *system performance*.

System Performance

Description

The *system performance* concept constitutes a performance measure for the evaluation and benchmarking of *technical systems*. Different performance measures are possible, depending on the chosen evaluation criterion. Typical criteria would be safety, reliability, ecological performance, and economic performance.

Definition

A *system performance* represents the performance aspect of a *technical system*.

Relations

- *System performance* is a subclass of *aspect system*.

- A *system performance* is considered under the aspect of performance.
- A *system performance* may evaluate some *aspect system* which itself can not be an instance of system performance.

System realization

Description

The *system realization* represents the physical (or virtual) constitution of the *technical system*. In case of a physical system, the *system realization* describes the system's physical structure, including its geometrical and mechanical properties. In case of a non-physical system, the *system realization* reflects the logical or abstract structure of the system; moreover, it may describe the (physical) implementation of the non-physical system.

Definition

A *system realization* represents the realization aspect of a *technical system*.

Relations

- *System realization* is a subclass of *aspect system*.
- A *system realization* is considered under the aspect of realization.
- A *system realization* may constrain some *system behavior*.
- A *system realization* may realize some *system function*.
- A *system realization* may have some performance measure of type *system performance*.

System requirements

Description

The *system requirements* specify the desired behavior of a *technical system* from an environment-centric perspective (cf. Chandrasekaran and Josephson, 2000). From the perspective of *systems requirements*, the *technical system* is viewed as a black box: Its structure and the underling physical and technical principles are not considered; only the effect on the environment is specified.

Definition

The *system requirements* represent the requirements aspect of a *technical system*.

Relations

- The class *system requirements* is a subclass of *aspect system*.
- The *system requirements* are considered under the aspect of requirements.
- The *system requirements* can be achieved through a *system function*.
- The *system requirements* may have some performance measure of type *system performance*.

Technical system

Description

A *technical system* is a *system* which has been developed in an engineering design process. The criterion to qualify as a *technical system* is "to be designed in order to fulfill some required function" (Bayer, 2003). Thus, the *technical system* concept may denote all kinds of technical artifacts, such as chemical plants, cars, computer systems, or infrastructure systems like a sewage water system. But also non-technical artifacts like chemical products, and even non-physical artifacts, such as software programs or mathematical models, can be considered as *technical systems*.

Relations

- *Technical system* is a subclass of *system*.
- A *technical system* has at most one behavioral aspect represented by a *system behavior*.
- A *technical system* has a functional aspect represented by a *system function*.
- A *technical system* has a performance aspect represented by a *system performance.*
- A *technical system* has at most one realization aspect represented by a *system realization.*
- A *technical system* has at most one requirement aspect represented by *system requirements.*

Relations

constrains

Description

The constrains relation indicates that a *system realization* imposes constraints on the *system behavior*.

Characteristics

- Specialization of isDirectlyRelatedTo
- Domain: *System realization*
- Range: *System behavior*
- Inverse: isConstrainedBy

evaluates

Description

The relation evaluates refers from a performance measure to the *aspect system* the performance of which is evaluated.

Characteristics

- Specialization of isDirectlyRelatedTo
- Domain: *System performance*
- Range: *Aspect system*
- Inverse: isEvaluatedBy

fulfills

Description

The fulfills relation states that a *system function* fulfills a particular *system requirement*.

Characteristics

- Specialization of isDirectlyRelatedTo
- Domain: *System function*
- Range: *System requirements*

- Inverse: isAchievedThrough

hasBehavioralAspect

Description

The relation points to the behavioral aspect of a *technical system*.

Characteristics

- Specialization of hasAspectSystem
- Domain: *System*
- Range: *System behavior*
- Inverse: representsBehaviorOf

hasFunctionalAspect

Description

The relation points to the functional aspect of a *technical system*.

Characteristics

- Specialization of hasAspectSystem
- Domain: *System*
- Range: *System function*
- Inverse: representsFunctionOf

hasPerformanceMeasure

Description

The relation hasPerformance points from an *aspect system*, the performance of which is evaluated, to the performance measure.

Characteristics

- Specialization of isDirectlyRelatedTo
- Domain: *Aspect system*
- Range: *System performance*
- Inverse: evaluates

hasPerformanceAspect

Description

The relation points to the performance aspect of a *technical system*.

Characteristics

- Specialization of hasAspectSystem
- Domain: *System*
- Range: *System performance*
- Inverse: representsPerformanceOf

hasPhenomenon

<u>Description</u>

The relation hasPhenomenon assings a *phenomenon* to a *system behavior*.

<u>Characteristics</u>

- Specialization of object-featureRelation
- Domain: *system behavior*
- Range: *phenomenon*

hasRealizationAspect

<u>Description</u>

The relation points to the realization aspect of a *technical system*.

<u>Characteristics</u>

- Specialization of hasAspectSystem
- Domain: *System*
- Range: *System realization*
- Inverse: representsRealizationOf

hasRequirementsAspect

<u>Description</u>

The relation points to the requirements aspect of a *technical system*.

<u>Characteristics</u>

- Specialization of hasAspectSystem
- Domain: *System*
- Range: *System requirements*
- Inverse: representsRequirementsOf

isInfluencedBy

<u>Description</u>

The relation isInfluencedBy indicates which *properties* are influenced by a particular *phenomenon*.

<u>Characteristics</u>

- Derived from the meta relation object-featureRelation
- Domain: *property*
- Range: *phenomenon*

isAchievedThrough

<u>Description</u>

The relation isAchievedThrough states that a *system requirement* can be achieved by means of a some *system function*.

Characteristics

- Specialization of isDirectlyRelatedTo
- Domain: *System requirements*
- Range: *System function*
- Inverse: fulfills

isConstrainedBy

Description

The constrains relation states that the *system behavior* is limited by the constraints imposed by the *system realization*.

Characteristics

- Specialization of isDirectlyRelatedTo
- Domain: *System behavior*
- Range: *System realization*
- Inverse: constrains

isRealizedBy

Description

The relation isRealizedBy states that a *system function* is implemented by some *system realization*.

Characteristics

- Specialization of isDirectlyRelatedTo
- Domain: *System function*
- Range: *System realization*
- Inverse: realizes

realizes

Description

The relation realizes states that a *system realization* implements a particular *system function*.

Characteristics

- Specialization of isDirectlyRelatedTo
- Domain: *System realization*
- Range: *System function*
- Inverse: isRealizedBy

representsBehaviorOf

Description

The relation refers from a *system behavior* to the overall *technical system*.

Characteristics

- Specialization of representsAspectOf
- Domain: *System behavior*
- Range: *System*
- Inverse: hasBehavioralAspect

representsFunctionOf

Description

The relation refers from a *system function* to the overall *technical system*.

Characteristics

- Specialization of representsAspectOf
- Domain: *System function*
- Range: *System*
- Inverse: hasFunctionalAspect

representsPerformanceOf

Description

The relation refers from a *system performance* to the overall *technical system*.

Characteristics

- Specialization of representsAspectOf
- Domain: *System performance*
- Range: *System*
- Inverse: hasPerformanceAspect

representsRealizationOf

Description

The relation refers from a *system realization* to the overall *technical system*.

Characteristics

- Specialization of representsAspectOf
- Domain: *System realization*
- Range: *System*
- Inverse: hasRealizationAspect

representsRequirementsOf

Description

The relation refers from the *system requirements* to the overall *technical system*.

Characteristics

- Specialization of representsAspectOf

- Domain: *System requirement*
- Range: *System*
- Inverse: hasRequirementAspect

Individuals

behavior

Description

Explicitly designates a behavioral *aspect system*.

Characteristics

- Instance of *aspect*

function

Description

Explicitly designates a functional *aspect system*.

Characteristics

- Instance of *aspect*

performance

Description

Explicitly designates an *aspect system* that represents the aspect of performance.

Characteristics

- Instance of *aspect*

realization

Description

Explicitly designates an *aspect system* that represents the aspect of realization.

Characteristics

- Instance of *aspect*

requirements

Description

Explicitly designates an *aspect system* that represents the aspect of requirements.

Characteristics

- Instance of *aspect*

5. Coordinate System

The ontology module *coordinate_system* is a supplement to the *system* module. Fig. 32 gives an overview on the concepts established by *coordinate_system*. In particular, it introduces the concept of a *coordinate system*, a special type of *system* that provides a frame of reference for the observation of *properties* owned by other *systems*.

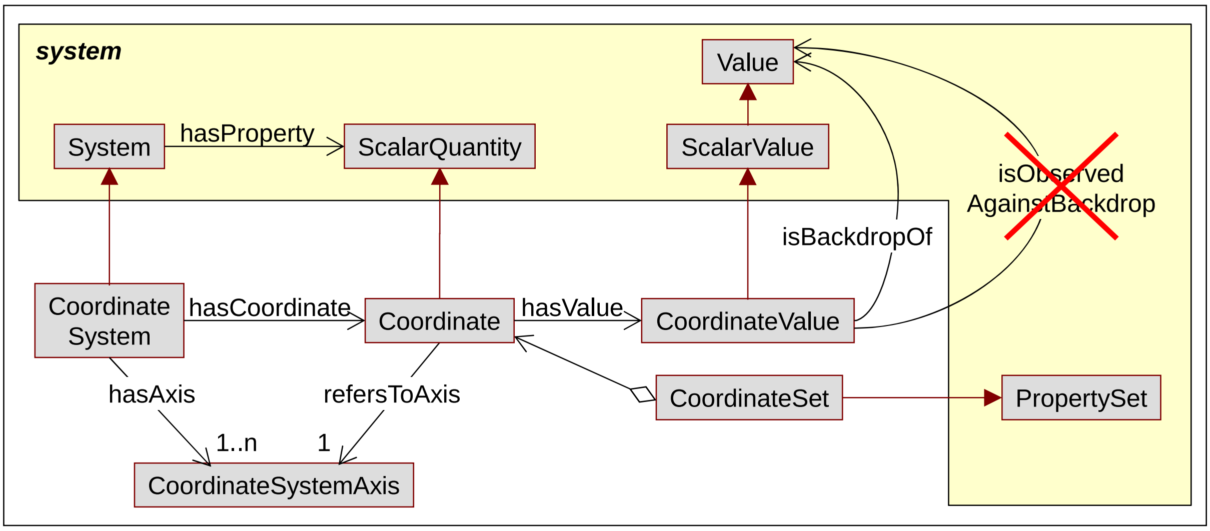

Fig. 32: Coordinate system

The *properties* of a *coordinate system* are called *coordinates*. A *coordinate* is defined as a *scalar quantity*, the values of which (i) serve as a backdrop for some *values* and (ii) cannot be observed against some further backdrop. Hence, as a *coordinate* cannot have a backdrop of its own, it constitutes an ‘absolute’ or ‘final’backdrop for the observation of *properties*; it thus breaks the loop caused by the relativity of the backdrop concept (cf. the discussion in Sect. 2.10).

Each *coordinate* refers to one *coordinate system axis*, which further qualifies the *coordinate*. For example, a spatial coordinate may refer to the x-axis of a spatial coordinate system, thus clarifying its spatial orientation. The *coordinate system axis* itself is not further specified through ontological concepts; consequently, its characteristics – e.g., its orientation relative to some spatial objects not described by OntoCAPE – must be defined outside the ontology.

Detailed concept definitions are given below. The usage of the concepts is explained in Morbach et al. (2008d) as part of the documentation of ontology module *space_and_time* .

Concept Descriptions

Individual concepts of the module *coordinate_system* are defined below.

Classes

Coordinate

Description

A *coordinate* is a property of a *coordinate system*. The values of a *coordinate* provide an ‘absolute’ or ‘final’ backdrop for the observation of some *properties*.

Relations

- *Coordinate* is a subclass of *scalar quantity*.
- A *coordinate* is a property of some *coordinate system*.
- A *coordinate* can only be the property of a *coordinate system*.
- A *coordinate* refers to exactly one *coordinate system axis*.
- A *coordinate* has only *coordinate values*.

Coordinate set

Description

A *coordinate set* groups some *coordinates* which logically belong together.

Definition

A *coordinate set* is a *property set* that comprises only *coordinates*.

Relations

- *Coordinate set* is a subclass of *property set*.
- A *coordinate set* comprises at least two *coordinates*.
- A *coordinate set* comprises only *coordinates*.
- A *coordinate set* can only be the property of a *coordinate system*.

Coordinate system

Description

A *coordinate system* constitutes a frame of reference for the observation of *properties* owned by other *systems*.

Definition

A *coordinate system* is a *system* that has some *coordinates* as properties.

Relations

- *Coordinate system* is a subclass of *system*.
- A *coordinate system* has some *coordinates* as properties.
- A *coordinate system* has one or more *coordinate system axes*.

Coordinate system axis

Description

A *coordinate system axis* represents an axis of a *coordinate system*.

Relations

- *Coordinate system axis* is a specialization of the meta class *non-exhaustive value set*.

Coordinate value

Description

A *coordinate value* serves as a backdrop for some *values*, yet it cannot have a backdrop of its own.

Definition

A *coordinate value* is a *scalar value* which is the value of a *coordinate*.

Relations

- *Coordinate value* is a subclass of *scalar value*.
- A *coordinate value* is the value of some *coordinate*.
- A *coordinate value* can only be the value of a *coordinate*.
- A *coordinate value* is the backdrop of some *value*.
- A *coordinate value* cannot be evaluated against some backdrop.

Relations

hasAxis

Description

The relation hasAxis identifies the *coordinate system axes* that belong to a particular *coordinate system*.

Characteristics

- Specialization of the meta relation object-featureRelation
- Domain: *coordinate system*
- Range: *coordinate system axis*

hasCoordinate

Description

The relation hasCoordinate indicates the *coordinates* of a *coordinate system*.

Characteristics

- Specialization of hasProperty
- Domain: *coordinate system*
- Range: *coordinate*

refersToAxis

Description

By means of the relation refersToAxis, a *coordinate* can be further specified. For example, a spatial coordinate may refer to the x-axis of a spatial coordinate system, thus clarifying its spatial orientation.

Characteristics

- Specialization of the meta relation object-featureRelation
- Domain: *coordinate*
- Range: *coordinate system axis*
- Functional

6. Tensor Quantity

As explained in Sect. 2.11, *physical quantities* include not only scalars but also vectors (e.g., velocity vector) and higher-order tensors (e.g., the dyadic stress tensor). The ontology module *tensor_quantity* provides the necessary concepts to define such *tensor quantities*.

A *tensor quantity* is a *physical quantity* that is assigned a tensor order. A *tensor quantity* of order *k* can be defined by induction:

- A *tensor quantity* of order 0 is a *scalar quantity*.
- A *tensor quantity* of rank *k* is given by an n-tuple, the elements of which are again *tensor quantities* of order (*k*-1).

Thus, a *tensor quantity* of arbitrary order can be recursively decomposed into *tensor quantities* of lower order, ultimately obtaining *scalar quantities*.

The above definition is implemented in OWL as follows. The order of the *tensor quantity* is denoted by the attribute hasTensorOrder. For the modeling of the tuple structure, we apply the design pattern for an array introduced in the Meta Model (cf.). This leads to the structure displayed on the left-hand side of Fig. 33.

A *tensor quantity* has elements of type *physical quantity*, which may again be *tensor quantities* of a lower order (note that the rank reduction of the tensor elements cannot be enforced in the OWL language, but must be accomplished manually). The order of the tensor elements is established through the *index* class: Each tensor element is assigned an *index* with unique integer value (given by the index attribute) via the determinesPositionOf relation; the *indices* refer to the *tensor quantity* via the isOrderedBy relation (cf. Sect. 6 for details).

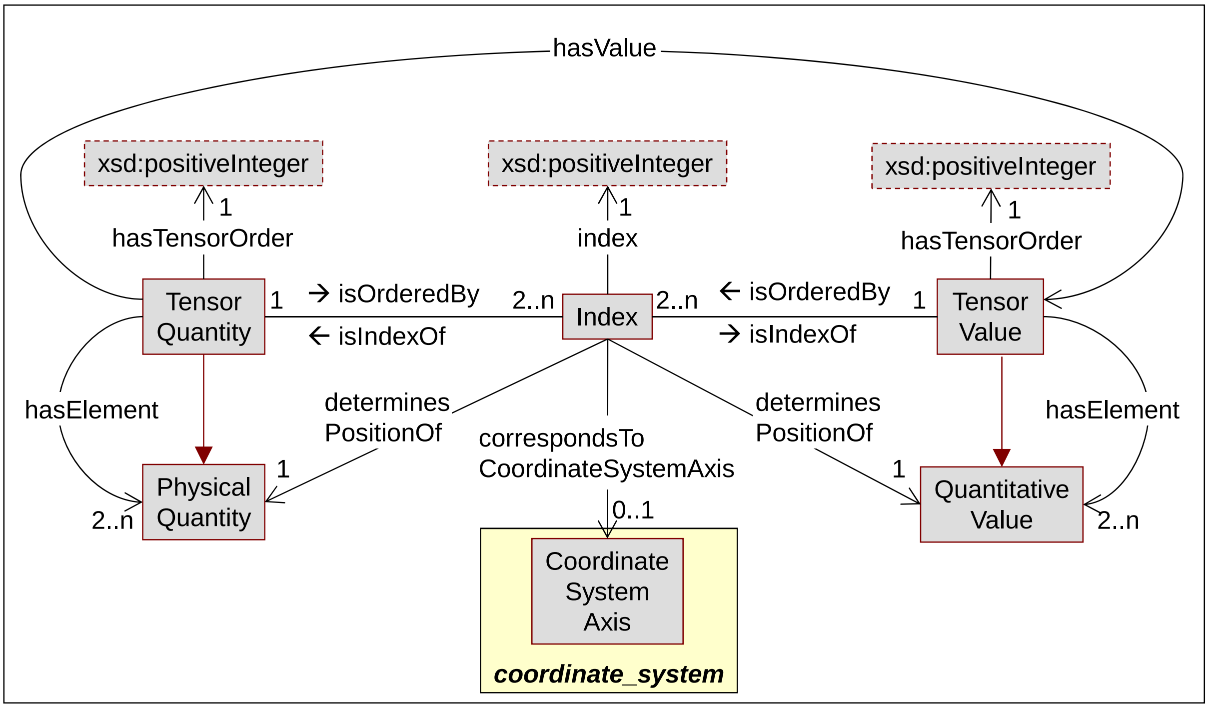

Fig. 33: Tensor quantity and tensor value

The *value* of a *tensor quantity* must again be a tensor of the same order as the *tensor quantity*. To this end, the class *tensor value* is introduced. A *tensor value* is defined analogously to a *tensor quantity*, as can be seen on the right-hand side of Fig. 33. Thus, each *tensor value* can be ultimately decomposed into *scalar values*.

Like all *physical quantities*, a *tensor quantity* is assigned a *physical dimension*, which must be the same *physical dimension* as that of its tensor elements[24]. Thus, unlike the concept of a *property set*, a *tensor quantity* comprises only *physical quantities* of the same type.

Two special types of *tensor quantities* are exemplarily introduced below: the *vector quantity* and the *matrix quantity*.

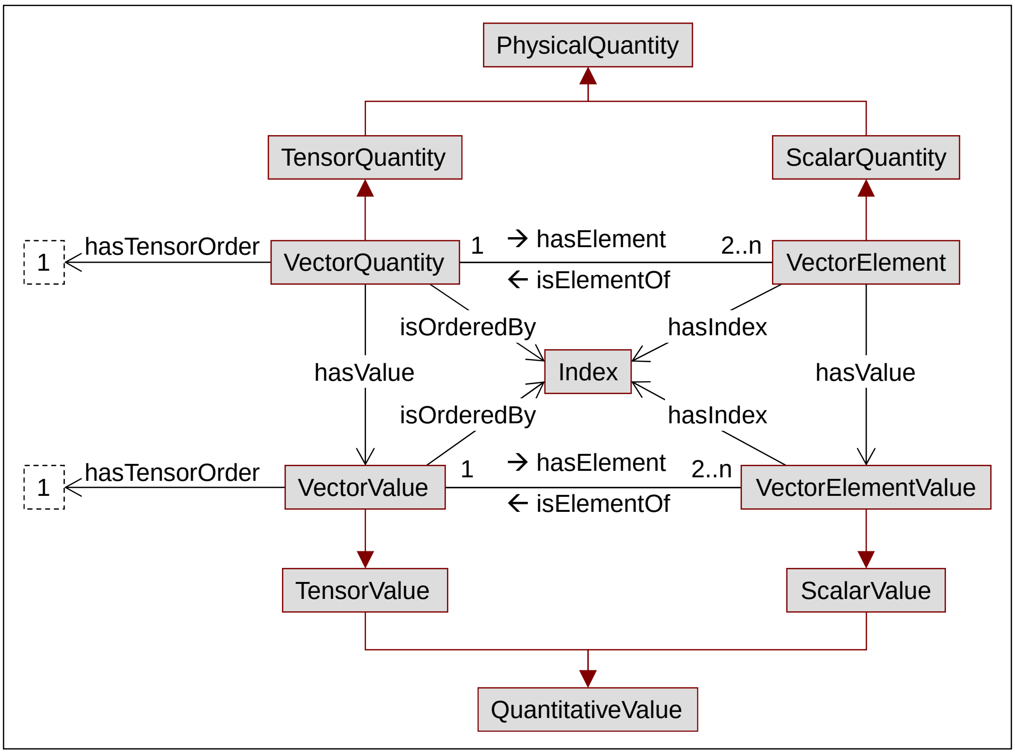

Fig. 34: Interrelations between *vector quantity*, *vector element*, *vector value*, and *vector element value*

A *vector quantity* is a *tensor quantity* that has a tensor order of 1. It is composed of *vector elements*, subclasses of *scalar quantity*, which by default refer to an *index* via the hasIndex relation. A *vector quantity* has *vector values*, which are defined analogously to *vector quantities.* A *vector value* is composed of scalar *vector element values*; these are specialized *scalar values* referring to an *index*. Fig. 34 summarizes the above concept definitions.

---

[25] CamelCase is the practice of writing compound words joined without spaces; each word is capitalized within the compound. While the UpperCamelCase notation also capitalizes the initial letter of the compound, the lowerCamelCase notation leaves the first letter in lowercase.

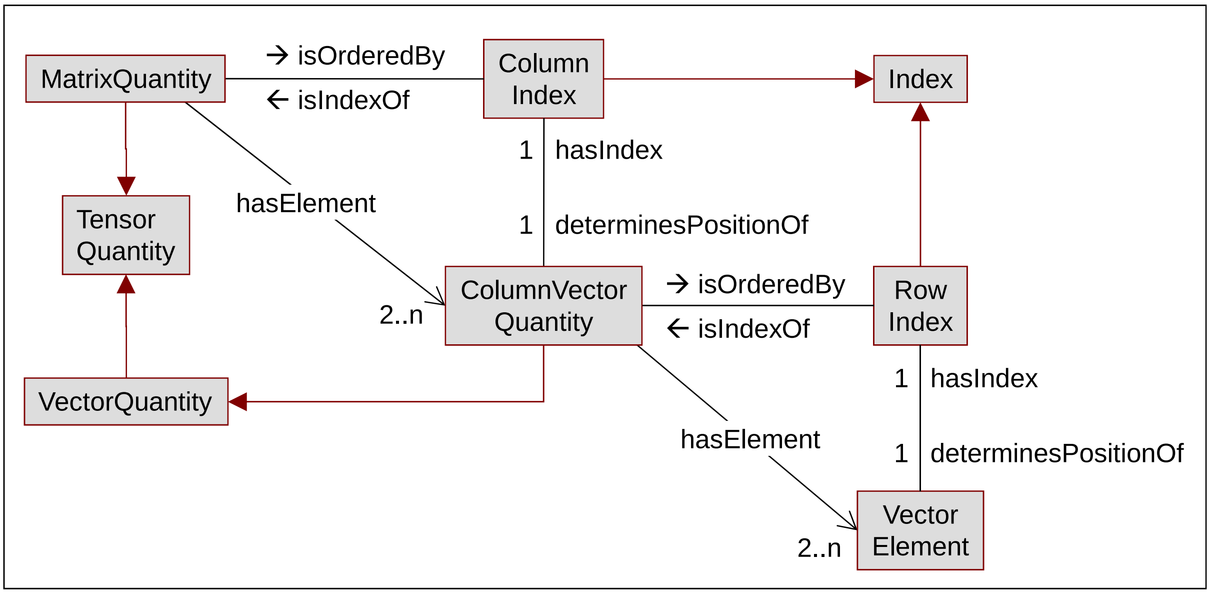

Fig. 35. Definition of the *matrix quantity* concept

A *matrix quantity* is a *tensor quantity* of rank 2, the elements of which are *vector quantities*. As these vectors constitute the columns of the *matrix quantity*, they are specifically designated as *column vector quantities*, and each *column vector quantity* is assigned a *column index*. By contrast, the *vector elements* of the *column vector quantity* are ordered by a *row index*. The definitions of these concepts are summarized by Fig. 35; Fig. 36 illustrates their usage.

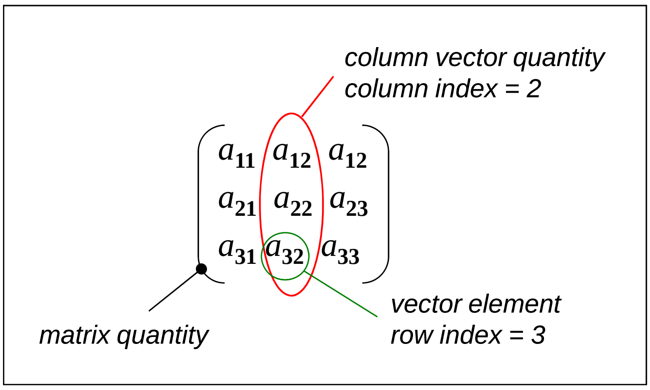

Fig. 36: Usage of the *matrix quantity* concept

The value of a *matrix quantity* is designated as a *matrix value* (not shown in Fig. 35 for the sake of clarity). Analogously to the above definitions, a *matrix value* is composed of *column vector values*, again ordered by a *column index*; the elements of the *column vector value* are *vector values*, which are ordered by a *row index*.

Concluding the above discussion, Fig. 37 gives an application example. It shows a two-dimensional stress tensor (i.e., *matrix quantity*), consisting of the *scalar quantities* σ_x, τ_{xy}, τ_{yx}, and σ_y, and its associated *matrix value*. Note that only the second columns of *matrix quantity* and *matrix value* are elaborately modeled. For the sake of clarity, the respective class names in brackets are omitted.

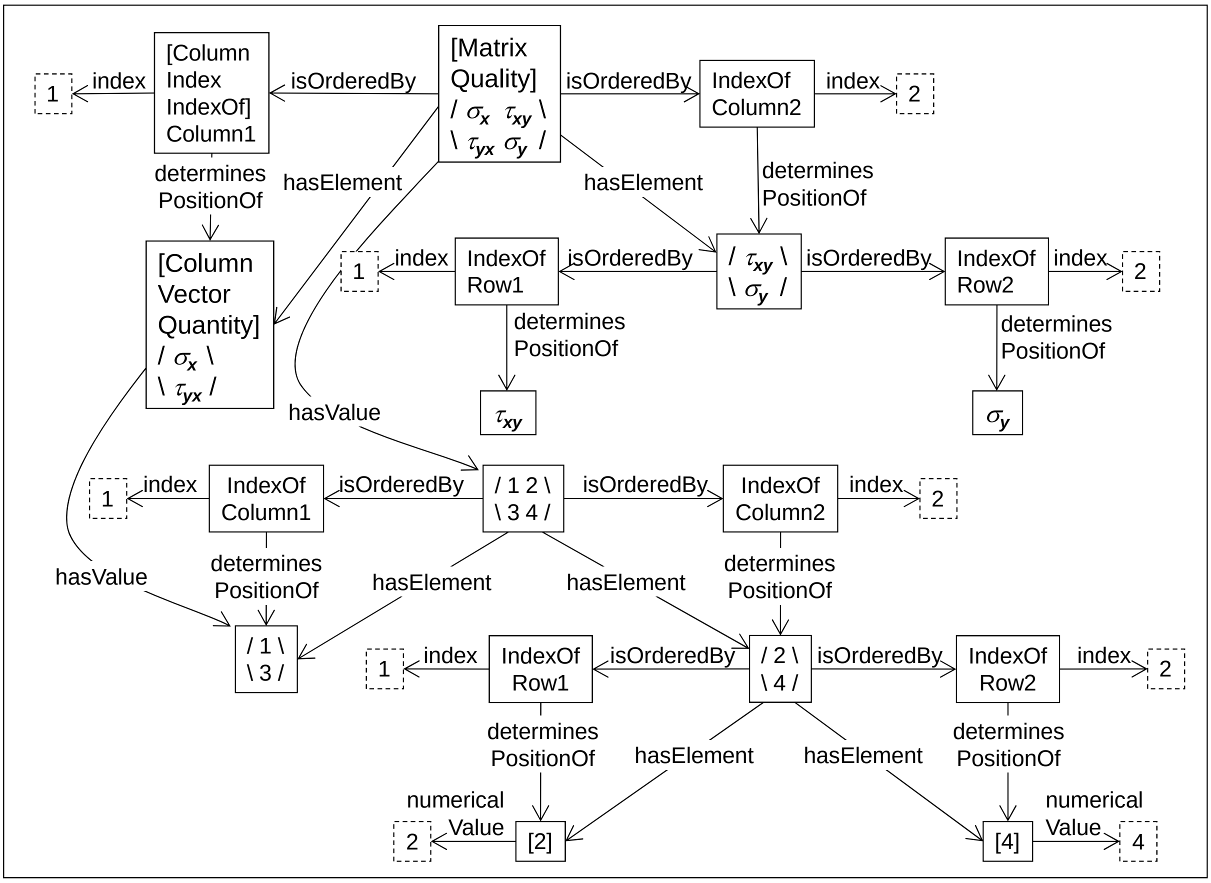

Fig. 37: Application example of a *matrix quantity* and its *matrix value*

The definitions introduced so far conceptualize a tensor as a mere data structure, thereby ignoring its geometrical properties. Yet the complete specification of a tensor requires a statement of direction or orientation (Gruber and Olsen, 1994). The tensor orientation can be indicated by assigning a spatial dimension to each element of a tensor; concretely, this is realized by referring from a *vector element* to the concept of a *coordinate system axis* (cf. Morbach et al., 2008d) via the relation hasOrientation. Note that a *vector element* may refer to a *cartesian coordinate system axis* or a *curvilinear coordinate system axis* (cf. Morbach et al., 2008d). The latter enables the definition of rotation vectors to represent *physical quantities* like torque or angular momentum.

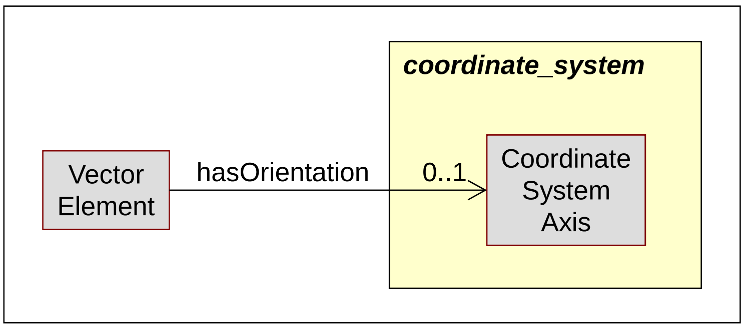

Fig. 38: Specifying the orientation of a tensor by referring to a coordinate system axis

The reference to a *coordinate system axis* (cf. Fig. 38) is of special importance, since we have defined the tensor as the recursive composition of its scalar elements. Yet while a tensor (as a whole) is independent of any chosen frame of reference, the decomposition of the tensor into its scalar elements depends on the particular choice of the reference frame. Thus, for a complete definition of a tensor in terms of its constituent elements, the respective reference *coordinate system* must be specified. If such specification is omitted, the following will be assumed by default: The tensor elements refer to a positive Cartesian coordinate system, where the *vector element* with an index value of 1 refers to the *x*-axis, and the *vector element* with an index value of 2 refers to the *y*-axis, etc.

Concept Descriptions

Individual concepts of the module *tensor_quantity* are defined below.

Class Descriptions

Classes

Column index

Description

A *column index* denotes the position of a column vector within a matrix.

Relations

- *Column index* is a subclass of *index*.
- A *column index* determines the position of either a *column vector quantity* or a *column vector value*.
- A *column index* is the index of either a *matrix quantity* or a *matrix value*.

Column vector quantity

Description

A *column vector quantity* represents a column vector of a *matrix quantity*.

Definition

A *column vector quantity* is a *vector quantity* that is an element of a *matrix quantity*.

Relations

- *Column vector quantity* is a subclass of *vector quantity*.
- A *column vector quantity* is an element of exactly one *matrix quantity*.
- A *column vector quantity* has exactly one *column index*.
- A *column vector quantity* is ordered by some *row indices*.
- A *column vector quantity* can only be ordered by *row indices*.
- A *column vector quantity* can only values of type *column vector value*.

Column vector value

Description

A *column vector value* represents a column vector of a *matrix value*.

Definition

A *column vector value* is a *vector value* that is an element of a *matrix value*.

Relations

- *Column vector value* is a subclass of *vector value*.
- A *column vector value* is an element of exactly one *matrix value*.
- A *column vector value* has exactly one *column index*.
- A *column vector value* is ordered by some *row indices*.
- A *column vector value* can only be ordered by *row indices*.
- A *column vector value* can only be the value of a *column vector quantity*.

Index

Description

An *index* represents the n-ary relation between a tensor, one of its elements, and the index attribute that denotes the position of the tensor element.

Relations

- *Index* is derived from the meta class *index*.
- An *index* determines the position of exactly one tensor element, which may be either a *scalar quantity* or *scalar value*.
- An *index* is index of exactly one *tensor quantity* or *tensor value*.
- The numerical value of the *index* is specified by the index attribute, which takes exactly one value of type positive integer.

Matrix quantity

Description

A *matrix quantity* is a second order *tensor quantity*.

Relations

- *Matrix quantity* is a subclass of *tensor quantity*.
- A *matrix quantity* has a tensor order of 2.
- A *matrix quantity* has some elements of type *column vector quantity*.
- A *matrix quantity* can only have elements of type *column vector quantity*.
- A *matrix quantity* is ordered by some *column indices*.
- A *matrix quantity* can only be ordered by *column indices*.
- A *matrix quantity* can only have values of type *matrix value*.

Matrix value

Description

A *matrix value* is a second order *tensor value*.

Relations

- *Matrix value* is a subclass of *tensor value*.
- A *matrix value* has a tensor order of 2.
- A *matrix value* has some elements of type *column vector value*.
- A *matrix value* can only have elements of type *column vector value*.
- A *matrix value* is ordered by some *column indices*.
- A *matrix value* can only be ordered by *column indices*.
- A *matrix value* can only be the value of a *matrix quantity*.

Row index

Description

A *row index* denotes the position of a scalar element within a column vector.

Relations

- *Row index* is a subclass of *index*.
- A *row index* determines the position of either a *vector element* or a *vector element value*.
- A *row index* is the index of either a *column vector quantity* or a *column vector value*.

Tensor quantity

Description

A *tensor quantity* is a non-scalar *physical quantity*, such as a velocity vector or a stress tensor.

Relations

- *Tensor quantity* is a subclass of *physical quantity*.
- A *tensor quantity* has at least two elements of type *physical quantity*.
- The elements of a *tensor quantity* must always be of type *physical quantity*.
- A *tensor quantity* is ordered by at least two *indices*.
- A *tensor quantity* can only be ordered by an *index*.
- The value of a *tensor quantity* must be of type *tensor value*.
- The order of the *tensor quantity* is specified by the attribute hasTensorOrder, which takes exactly one value of type positive integer.

Usage

The following axioms cannot be modeled in OWL and must therefore be enforced manually:

- The elements of a *tensor quantity* of rank *n* are again tensors of rank (*n*-1).
- The elements of a *tensor quantity* must have the same *physical dimension* as the *tensor quantity*.

Tensor value

Description

A *tensor value* is non-scalar *quantitative value* of a *tensor quantity*.

Relations

- *Tensor value* is a subclass of *quantitative value*.
- A *tensor value* has at least two elements of type *quantitative value*.
- The elements of a *tensor value* must always be of type *quantitative value*.
- A *tensor value* is ordered by at least two *indices*.
- A *tensor value* can only be ordered by an *index*.
- A *tensor value* can only be the value of a *tensor quantity*.

Vector element

Description

See definition.

Definition

A *vector element* is a *scalar quantity* that is the element of a *vector quantity*.

Relations

- *Vector element* is a subclass of *scalar quantity*.
- A *vector element* is an element of exactly one *vector quantity*.
- A *vector element* has exactly one *index*.
- A *vector element* can only have values of type *vector element value*.
- The orientation of a *vector element* may be indicated by referring to one *coordinate system axis*.

Vector element value

Description

A *vector element value* is a *scalar value* that is the element of a *vector value*.

Relations

- *Vector element value* is a subclass of *scalar value*.
- A *vector element value* is an element of exactly one *vector value*.
- A *vector element value* has exactly one *index*.
- A *vector element value* can only be the value of a *vector element*.

Vector quantity

Description

A *vector quantity* is a first order *tensor quantity*.

Relations

- *Vector quantity* is a subclass of *tensor quantity*.
- A *vector quantity* has a tensor order of 1.
- A *vector quantity* has some elements of type *vector element*.
- A *vector quantity* can only have elements of type *scalar quantity*.

- A *vector quantity* can only have values of type *vector value*.

Vector value

Description

A *vector value* is a first order *tensor value*.

Relations

- *Vector value* is a subclass of *tensor value*.
- A *vector quantity* has some elements of type *vector element value*.
- A *vector quantity* can only have elements of type *scalar value*.
- A *vector quantity* can only be the value of a *vector quantity*.

Relations

determinesPositionOf

Description

The relation determinesPositionOf refers from an *index* to the associated tensor element.

Characteristics

- Specialization of the meta relation determinesPositionOf
- Domain: *index*
- Range: *physical quantity*, *quantitative value*
- Inverse: hasIndex
- Functional

hasElement

Description

The relation hasElement identifies the elements of a tensor.

Characteristics

- Specialization of the meta relation isComposedOf
- Domain: *tensor quantity*, *tensor value*
- Range: *physical quantity*, *quantitative value*
- Inverse: isElementOf

hasIndex

Description

The relation hasIndex refers from a tensor element to its *index*.

Characteristics

- Specialization of the meta relation hasIndex
- Domain: *physical quantity*, *quantitative value*

- Range: *index*
- Inverse: determinesPositionOf
- Inverse functional

isElementOf

Description

The relation isElementOf denotes the affiliation of a tensor element to a tensor.

Characteristics

- Specialization of the meta relation isExclusivelyPartOf
- Domain: *physical quantity*, *quantitative value*
- Range: *tensor quantity*, *tensor value*
- Inverse: hasElement

isIndexOf

Description

The relation isIndexOf points from an *index* to the associated tensor.

Characteristics

- Specialization of the meta relation isIndexOfArray
- Domain: *index*
- Range: *tensor quantity*, *tensor value*
- Inverse: isOrderedBy
- Functional

isOrderedBy

Description

The relation isOrderedBy identifies the *index* of a tensor.

Characteristics

- Specialization of the meta relation isOrderedBy
- Domain: *tensor quantity*, *tensor value*
- Range: *index*
- Inverse: isIndexOf
- Inverse functional

hasOrientation

Description

The relation hasOrientation specifies the orientation of a tensor element by referring to the corresponding *coordinate system axis*.

Characteristics

- Specialization of the meta relation object-featureRelation
- Domain: *vector element*
- Range: *coordinate system axis*
- Functional

Attributes

hasTensorOrder

Description

The attribute denotes the order (or rank) of a tensor. Scalars are of order 0, vectors of order 1.

Characteristics

- Domain: *tensor quantity*, *tensor value*
- Datatype: positiveInteger (built-in XML Schema Datatype)
- Functional

index

Description

The attribute indicates the numerical value of an *index*.

Characteristics

- Domain: *index*
- Datatype: positiveInteger (built-in XML Schema Datatype)
- Functional

References

Alberts LK (1994) YMIR: a sharable ontology for the formal representation of engineering design knowledge. In: Gero JS, Tyugu E (eds.): *Formal Design Methods for CAD*. Elsevier, New York:3–32.

Andresen T (1999) The macroeconomy as a network of money-flow transfer functions. *Model. Ident. Contr.* **19**(4):207-223.

Barkmeyer EJ, Feeney AB, Denno P, Flater DW, Libes DE, Steves MP, Wallace EK (2003) *Concepts for Automating Systems Integration*. Technical Report (NISTIR 6928), National Institute of Standards and Technology (NIST), Gaithersburg, MD.

Baxter JE, Juster NP, de Pennington A (1994) A functional data model for assemblies used to verify product design specifications. *Proc. Inst. Mech. Eng. Part B J. Eng. Manuf.* **208** (B4):235-244.

Bayer B (2003) Conceptual Information Modeling for Computer Aided Support of Chemical Process Design. Fortschritt-Berichte VDI: Reihe 3, Nr. 787. VDI-Verlag, Düsseldorf.

Bayer B, Krobb C, Marquardt W (2001) *A Data Model for Design Data in Chemical Engineering - Information Models*, Internal Report LPT-2001-15, Lehrstuhl für Prozesstechnik, RWTH Aachen University.

Bertalanffy L (1968). General System Theory: Foundations, Development, Applications, Braziller, New York.

Bilgic T, Rock D (1997) Product data management systems: State-of-the-art and the future. In: *Proceedings of the 1997 ASME Design Engineering Technical Conferences, Sacramento, CA*.

BIPM (2006) *The International System of Units (SI)*, 8th edition. SI brochure, published by the International Committee for Weights and Measures (Bureau International des Poids et Measures, BIPM). Online available at http://www.bipm.fr/en/si/si_brochure/. Accessed 26 September 2007.

Borst WN (1997) *Construction of Engineering Ontologies for Knowledge Sharing and Reuse*, PhD Thesis, Centre of Telematics and Information Technology, University of Twente.

Bunge M (1979) Treatise on Basic Philosophy, Volume 4. Ontology II: A World of Systems. Reidel, Dordrecht.

Chandrasekaran B (1994) Functional representation and causal processes. In: Yovits MC (ed.): *Advances in Computers*. Academic Press, New York.

Chandrasekaran B, Josephson JR (2000) Function in device representation. *J. Eng. Comput*. **16**(3/4):162-177.

Föllinger O (1982) Einführung in die Zustandsbeschreibung dynamischer Systeme. Oldenbourg, München.

Gigch JP (1991) System Design Modeling and Metamodeling. Springer, New York.

Gilles ED (1998) Network theory for chemical processes. *Chem. Eng. Technol.* **21** (8):121–132.

Gruber TR, Olsen GR (1994) An Ontology for Engineering Mathematics. In: Doyle J, Torasso P, Sandewall E (eds.): *Proceedings of Fourth International Conference on Principles of Knowledge Representation and Reasoning*. Morgan Kaufmann. Online available at http://www-ksl.stan-ford.edu/knowledge-sharing/papers/engmath.html. Accessed September 2007.

IEEE (2000) IEEE Recommended Practice for Architectural Description for Software-Intensive Systems. *IEEE Standard 1471-2000*, Institute for Electrical and Electronics Engineering, New York.

Kitamura Y, Mizoguchi R (2003) Ontology-based description of functional design knowledge and its use in a functional way server. *Expert Syst. Appl.* **24** (2):153-166.

Klir GJ (1985) *Architecture of Systems Problem Solving*. Plenum Press, New York.

Mangold M, Angeles-Palacios O, Ginkel M, Kremling A, Waschler R, Kienle A, Gilles ED (2005) Computer-aided modeling of chemical and biological systems: methods, tools and applications. *Ind. Eng. Chem. Res.* **44**:2579–2591.

Marquardt W (1992). An object-oriented representation of structured process models. *Comput. Chem. Eng.* **16**:329–336.

Marquardt W (1994) Computer-aided generation of chemical engineering process models. *Int. Chem. Eng.* **34**:28–46.

Marquardt W (1995) Towards a Process Modeling Methodology. In: R. Berber: *Methods of Model-Based Control*. NATO-ASI E, Applied Sciences, **293**, Kluwer, Dordrecht:3-41.

Marquardt W (1996) Trends in computer-aided process modeling. *Comput. Chem. Eng.* **20** (6/7):591–609.

Marquardt W, von Wedel L, Bayer B (2000) Perspectives on lifecycle process modeling. In: Malone MF, Trainham JA, Carnahan B (eds.): *Foundations of Computer–Aided Process Design*, AIChE:192–214.

Morbach J, Yang A, Marquardt W (2007) OntoCAPE – A large-scale ontology for chemical process engineering. *Eng. Appl. Artif. Intell.* **20**(2):147-161.

Morbach J, Wiesner A, Marquardt W (2008a) *OntoCAPE 2.0 – The Meta Model*. Technical Report (LPT-2008-24), Lehrstuhl für Prozesstechnik, RWTH Aachen University. Online available at http://www.avt.rwth-aachen.de/ AVT/index.php?id=541&L=0&Nummer=LPT-2008-24.

Morbach J, Yang A, Marquardt W (2008b) *OntoCAPE 2.0 – Materials*. Technical Report (LPT-2008-27), Lehrstuhl für Prozesstechnik, RWTH Aachen University. Online available at http://www.avt.rwth-aachen.de/AVT/in-dex.php?id=541&L=0&Nummer=LPT-2008-27.

Morbach J, Yang A, Marquardt W (2008c) *OntoCAPE 2.0 – Mathematical Models*. Technical Report (LPT-2008-28), Lehrstuhl für Prozesstechnik, RWTH Aachen University. Online available at http://www.avt.rwth-aachen.de/AVT/index.php?id=541&L=0&Nummer=LPT-2008-28.

Morbach J, Yang A, Wiesner A, Marquardt W (2008d) *OntoCAPE 2.0 – Supporting Concepts*. Technical Report (LPT-2008-26), Lehrstuhl für Prozesstechnik, RWTH Aachen University. Online available at http://www.avt.rwth-aachen.de/AVT/index.php?id=541&L=0&Nummer= LPT-2008-26.

Patzak G (1982) Systemtechnik – Planung komplexer innovativer Systeme. Springer, Berlin.

Rector A, Welty C, eds. (2005) *Simple part-whole relations in OWL Ontologies*. W3C Editor's Draft, 11 August 2005. Online available at http:// www.w3.org/2001/sw/BestPractices/OEP /SimplePartWhole/. Accessed November 2007.

Szykman S, Sriram RD, Regli WC (2001) The role of knowledge in next-generation product development systems. *J. Comput. Inf. Sci. Eng.* **1** (1):3–11.

Thomé B, ed. (1993) Systems Engineering: Principles and Practice of Computer -based Systems Engineering. John Wiley, New York.

VIM (1993) *International Vocabulary of Basic and General Terms in Metrology (VIM), 2nd Edition*. Jointly prepared by ISO, IEC, BIPM, IFCC, IUPAC, IUPAP and OIML. Published by ISO, Geneva, as ISO Guide 99:1993.

Wiesner A, Morbach J, Bayer B, Yang A, Marquardt W (2008a) *OntoCAPE 2.0 – Chemical Process System*. Technical Report (LPT-2008-29), Lehrstuhl für Prozesstechnik, RWTH Aachen University. Online available at http://www.avt.rwth-aachen.de/AVT/index.php?id=541&L=0&Nummer =LPT-2008-29.

Appendix A Documentation Format

Classes

Classes are characterized by the following categories:

Description: A lexical description of the class, for example “A chemical reactor is an apparatus for holding substances that are undergoing a chemical reaction.” The description explains the meaning of the class to the user.

Definition: Unlike a description, a definition can be transcribed into a formal ontology language, where it establishes the set of necessary and sufficient conditions from which the membership of an ontological concept (class or individual) to the class can be inferred. Classes for which such a definition can not be indicated are called primitive classes.

Relations: The following characteristics are indicated, if existent:

- *Specialization*. A list of parent classes from which the current class is derived via specialization.
- *Disjointness*. A list of classes which are disjoint with the present class. Disjointness between classes means that an instance of the first class cannot simultaneously be an instance of the second class.
- *Restrictions*. Restrictions of binary relations (or attributes) specify the existence of a relation (or attribute) as well as its cardinality and value range with respect to the current class.

Usage: Some recommendations for the use of the class may be given if such advice is required.

Relations

Binary relations are characterized by the following categories:

Description: Similar to that of classes mentioned above.

Characteristics: The following characteristics are listed, if existent:

- *Specialization*. A listing of the relations from which the relation is derived via specialization.
- *Domain*. The domain of the relation.
- *Range*. The value range of the relation.
- *Inverse*. The inverse of a relation.
- Further characteristics, such as if the relation is *transitive*, *symmetric*, or (*inverse*) *functional*.

Usage: As above.

Attributes

Attributes are characterized by the following categories:

Description: As above.

Characteristics: The following characteristics are listed, if existent:

- *Specialization*. A listing of the attributes from which the attribute is derived via specialization.
- *Domain*. The domain of the attribute.

- *Range* or *datatype*. The value range of the attribute, which is usually indicated by referring to a built-in XML Schema Datatype (Biron et al., 2004).
- Further characteristics, such as if the attribute is *functional*.

Usage: As above.

Individuals

Predefined individuals are characterized by the following categories:

Description: As above.

Characteristics: The following characteristics are indicated, if existent:

- *Instance of*. The classes from which the individual is instantiated.
- *Different from*. A list of individual which are explicitly declared to be different from the present individual.
- *Relations*. Instances of binary relations the individual is involved in.
- *Attributes*. Attribute values of the individual.

Usage: As above.

Notation Conventions

Classes and relations of the Meta Model are named according to the CamelCase[25] naming convention: UpperCamelCase notation is used to denote identifiers of classes, while relation identifiers are represented in lowerCamelCase notation. No particular naming convention is followed for identifiers of individuals (i.e., instances of classes).

In this document, class identifiers are highlighted by *italicized sans-serif font*; for better readability, the UpperCamelCase notation is not applied in the text, but the individual words that constitute the class identifiers are written separately and in lowercase (e.g., *class identifier*). If relations are explicitly referred to in the text, they are written in lowerCamelCase notation and are additionally highlighted by sans-serif font. Individuals are accentuated by **bold sans-serif font**. Partial models are denoted **bold serif font**, *italicized serif font* refers to ontology modules.

In figures, a graphical notation in the style of UML class diagrams is used; the basic elements are depicted in Fig. 39. Grey shaded boxes represent *classes*, white boxes represent *individuals*. *Attributes* are denoted by grey shaded boxes with dashed boundary lines, *attribute values* by white boxes with dashed boundary lines. *Specialization* is depicted through a solid line with a solid arrowhead that points from the subclass to the superclass. A dashed line with an open arrowhead denotes *instantiation*. *Binary relations* are depicted though solid lines. Three basic relation types are distinguished: a line with one open arrowhead represents a *unidirectional* relation; a line with two open arrowheads represents a *symmetric* relation; a line without any arrowheads represents a bidirectional relation[26]. Finally, graphic elements for two special types of relation are introduced: an *aggregation* relation is depicted through a line with a white diamond-shaped arrowhead pointing towards the aggregate class. Similarly, a black diamond-shaped arrowhead indicates a *composition* relation.

---

[26] In OWL, a bidirectional relation is modeled through a unidirectional relation and its inverse.

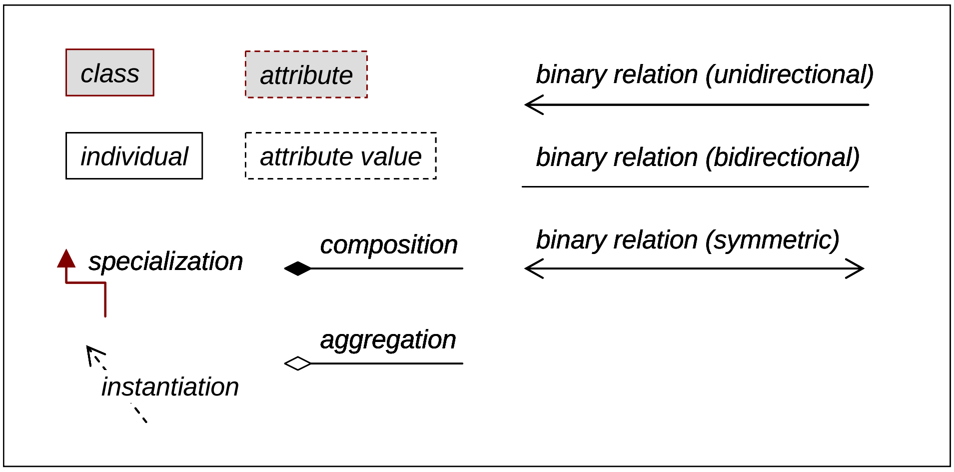

Fig. 39: Basic elements of graphical notation

Index of Concepts