3 Overview on OntoCAPE

Having established the scientific background of ontology engineering, we now present version 2.0 of the ontology OntoCAPE. Compliant with the terminology introduced in the previous chapter, OntoCAPE can be characterized as a *formal*, *heavyweight ontology*, which is represented in the OWL modeling language. It consists of several sub-ontologies, which perform different functions: According to the classification framework introduced in Sect. 2.6, the individual sub-ontologies serve the functions of a meta ontology (at the logical metalevel), a top-level ontology, a domain ontology, as well as some application ontologies.

3.1 Overview and Structure

As for any complex system, a sound architecture is critical for an ontology (1) to facilitate its efficient construction and long-term maintenance, and (2) to enable its reuse in different application contexts. In the following, we discuss how this concern is addressed by the design of OntoCAPE.
Fig. 3.1 gives an overview on OntoCAPE. As can be observed, the ontology is organized by means of two orthogonal structuring principles, which will be discussed in the two consecutive subsections: *abstraction layering* and *modularization*.

3.1.1 Abstraction Layering

To improve the usability and reusability of an ontology, numerous authors (e.g., Chandrasekaran and Johnson 1993; Russ et al. 1999; Borst 1997; Jarrar and Meersman 2002) have proposed the idea of structuring an ontology into different *levels of abstractions*. Following their recommendation, OntoCAPE has been subdivided by means of layers (cf. Fig. 3.1), which separate general knowledge from knowledge about particular domains and applications.
The design of each layer follows the principle of "minimal ontological commitment" (Gruber 1995) (cf. Sect. 10.5), meaning that a layer holds only those ontological terms and axioms that are essential for its function; terms and axioms that are not essential for the layer's purpose are sourced out to lower layers. The topmost *Meta Layer*, is the most abstract one; it holds a meta ontology (cf. Sect. 2.6), which introduces fundamental modeling concepts and states the design guidelines for the construction of the actual ontology. Next, the *Upper Layer* of OntoCAPE defines the principles of general systems theory according to which the ontology is organized. On the subjacent *Conceptual Layer*, a conceptual model of the CAPE

W. Marquardt et al., *OntoCAPE*, RWTHedition, DOI 10.1007/978-3-642-04655-1_3,

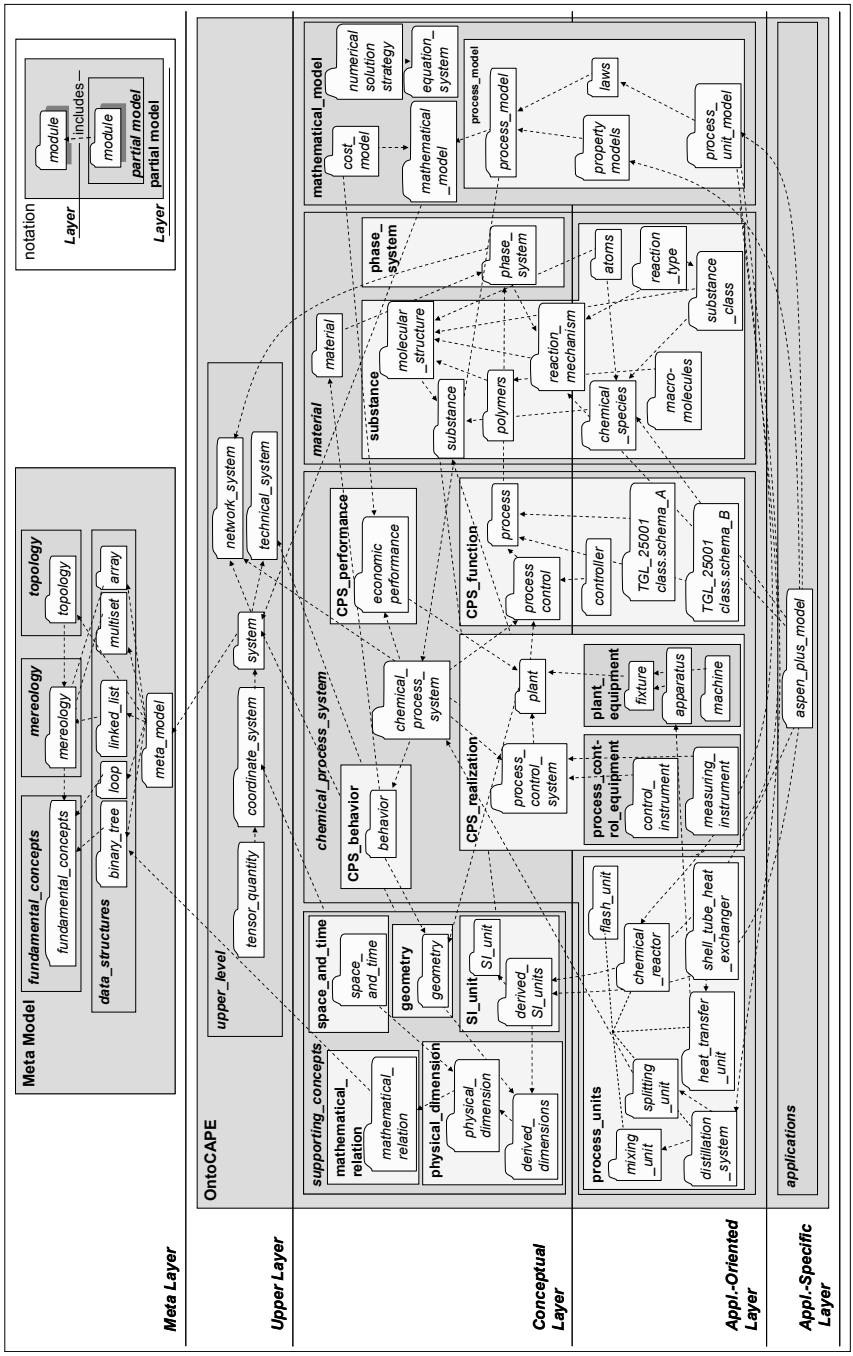

Fig. 3.1: Structure of OntoCAPE

domain is established, which covers such different areas as unit operations, equipment and machinery, materials and their thermophysical properties, chemical process behavior, modeling and simulation, and others. The two bottommost layers refine the conceptual model by adding classes and relations required for the practical application of the ontology: The *Application-Oriented Layer* generically extends the ontology towards certain application areas, whereas the *Application-Specific Layer* provides specialized classes and relations for concrete software applications.

The layered design takes the reusability-usability trade-off problem (cf. Sect. 1.3) into account: According to the trade-off problem, the general knowledge, which is located on the Upper Layer of OntoCAPE, can be reused in a variety of application contexts, but it is not immediately usable. By contrast, the knowledge located on the lower layers is ready for use, but problem-specific and thus hardly transferable to other applications. Thus, each layer contains knowledge of a specific degree of usability and reusability; traversing down the ontology, the usability of the knowledge increases, whereas its reusability decreases. Now, if (a part of) the ontology is to be reused for building some knowledge-based application, the appropriate abstraction level for knowledge reuse must be found. In practice, this means that one needs to traverse up the ontology (starting from the Application-Specific Layer) until the encountered knowledge is generic enough to fit the respective application context.

As an example, consider an intelligent CAPE tool, for the development of which a preferably large part of OntoCAPE is to be reused. The knowledge on the bottommost layer is application-specific and therefore of little value for any new tool. Yet already the above Application-Oriented Layer may contain some reusable knowledge, provided that the tool operates in an application area that is covered by OntoCAPE at all. If this is not the case, we need to move up to the Conceptual Layer; here, at the latest, some reusable knowledge can be found. Thus, for building a CAPE tool, one may reuse the ontology down to and including the Conceptual Layer at least. If, on the other hand, the tool was from a different application area than CAPE, it would still be possible to reuse knowledge from the Upper Layer and the Meta Layer.

3.1.2 Modularization

Modularization (i.e., the subdivision of the ontology into largely self-contained units), has been recommended by many authors (e.g., Gruber and Olsen 1994; Borst 1997; Bernaras et al. 1996; Visser and Cui 1998; Pinto et al. 1999; Heflin and Hendler 2000; Rector 2003; Stuckenschmidt and Klein 2003) as a means to promote the intelligibility (cf. Sect. 10.3), the adaptability (cf. Sect. 10.4), and generally the reusability of an ontology.

In OntoCAPE, modularization has been realized by partitioning the ontology into *modules* and *partial models*. Throughout the text, the identifiers of modules will be denoted in *italicized serif font*, whereas the identifiers of partial models will be denoted in **bold serif font**.

3.1.2.1 Modules

A *module* assembles a number of interrelated classes, relations, and axioms, which jointly conceptualize a particular topic (e.g., the module *plant* holds a conceptualization of chemical plants). The boundaries of a module are to be chosen such that the module can be designed, adapted, and reused to some extent independently from other parts of an ontology (Stuckenschmidt and Klein 2003). A module may include another module, meaning that if module A includes module B, the ontological definitions specified in B are valid in A and can thus be directly used (i.e., extended, refined …) in A. This allows to decompose OntoCAPE into an "inclusion lattice" (Gruber and Olsen 1994) of loosely coupled modules, as shown in Fig. 3.1.

By definition, modules have strong internal coherence but relatively loose coupling with the other parts of the ontology (Borst 1997), which facilitates their handling, thus improving the *adaptability* of the ontology (this issue is discussed in Sect. 10.4). Moreover, the modules are concise and therefore easier to comprehend than an entire ontology, hence bringing advantages with respect to *intelligibility* (cf. Sect. 10.3). Furthermore, the modular structure facilitates the selective reuse of the ontology: A user may choose to reuse only a selected part of the ontology if other parts are not relevant in the respective application context. In this case, it is relatively simple to cut the connections between the modules to be reused and the remainder of the ontology.

In the formal specification of OntoCAPE, modules are manifested through XML namespaces (Bray et al. 2006a). By convention, the concepts of a common namespace are stored in a single OWL file of the same name as the corresponding module. Inclusion is realized by means of the OWL import mechanism. Moreover, each module is (conceptually) assigned to one particular layer (cf. Fig. 3.2), thus integrating the structuring mechanisms of layering and modularization.

3.1.2.2 Partial Models

Modules that address closely related topics are grouped into a common *partial model* – for instance, the partial model **plant_equipment** clusters the thematically related modules *fixture*, *apparatus*, and *machine*.

The partial models constitute a coarse categorization of the domain. Unlike modules, partial models may be nested and may stretch across several layers. While the boundaries of the modules are chosen for practical considerations (i.e., such

that the interdependencies between the individual modules are minimized), the boundaries of the partial models reflect the 'natural' thematic boundaries of the domain. In the course of ontology evolution (cf. Sect. 11.1), the partial model structure is therefore supposed to remain relatively stable, whereas the number of modules as well as their content and mutual dependencies are likely to change over time. Thus, the partial model structure provides a stable frame of orientation for the organization of the modules.
In the formal specification of OntoCAPE, the partial models are implemented as (file) directories. That way, they establish a directory structure for managing the OWL files.

3.1.2.3 Variants

As there is no unique way of modeling an area of interest, different *variants* of an ontology module may evolve. These variants represent alternative conceptualizations of the subject covered by the module. Variants will particularly arise on the application-near layers whenever the ontology is adapted to a new application because a new application usually implies a different view on the domain and consequently a different conceptualization (Noy and Klein 2004). Consider for example the module *plant* (cf. Sect. 8.3.1), which holds a conceptualization of plant equipments and their connections: A knowledge management system, as the one described in Sect. 12.2, calls for a simple, coarse-grained description of connectivity (i.e., the mere indication that equipment A is connected to equipment B is sufficient for this application). On the other hand, an ontology-based system for the integration of engineering data, as the one sketched in Sect. 12.3, would require a more precise description: Connectivity must be addressed by indicating the number and position of flanges, specification of their type and diameter, etc. Yet for the knowledge management system, these details are irrelevant and should thus be omitted. As a solution, two variants of the module *plant* can be developed, providing different conceptualizations of connectivity specifically tailored to the information demands of the respective application.
While the variants will predominantly emerge on the lower layers of the ontology, the basis for their evolvement must be established on the higher layers already, particularly on the Meta Layer and the Upper Layer. These layers define the fundamental theories on which the variants are based. Consequently, these theories must be formulated in a flexible manner, such that they allow for alternative refinements in form of different variants. In the above example, for instance, both variants of *plant* are based on a generic theory of connectivity formulated in the Meta Model (partial model **topology**, cf. Sect. 4.4). Thus, the generic theory must tolerate both the coarse-grained and the fine-grained specification of connectivity.
In the formal specification of OntoCAPE, the variants of a module are represented as separate OWL files. These files are stored in the same directory, and they have the first part of their two-part file name in common. However, this proceeding is

only appropriate for a small number of variants; with an increasing number, the use of a variant management software (such as Pure::Variants; Pure Systems 2008) should be considered.

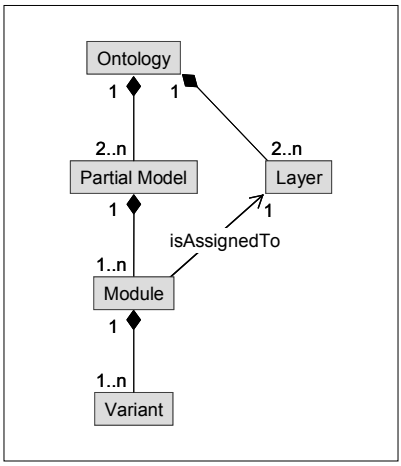

Fig. 3.2: Formal model of the structuring elements used for organizing OntoCAPE

Summarizing the above discussion, Fig. 3.2 displays a formal model of the structuring elements that have been used to organize OntoCAPE.

3.2 Representation and Dissemination

As stated in Sect. 2.6, a reusable ontology must be available both in form of an informal specification and in form of a formal specification. Accordingly, OntoCAPE 2.0 is issued in both forms of representation:

- The informal specification currently takes the form of six technical reports (Morbach et al. 2008f; 2008g; 2008h; 2008i; 2008j; Wiesner et al. 2008a), which jointly comprise about 500 pages. The important concepts introduced in these reports are summarized in the respective chapters of this book (cf. Chaps. 4-9). Within these chapters, the organization and structure of OntoCAPE are presented, and the conceptualizations of various topic areas are described in detail, often complemented by intuitive UML-like diagrams. Moreover, the major design decisions of ontology engineering are explicated, thus providing the reader with the necessary background knowledge for extending and customizing the ontology to his/her own purposes. Finally, the usage of the ontology is explained, and some sample applications are presented.

– As mentioned before, the formal specification of OntoCAPE has been realized in OWL. A short introduction to that particular modeling language can be found in Sect. 2.4. The current release of OntoCAPE consists of 62 OWL files, each of which includes one module of the ontology. In total, the implementation comprises about 500 classes, 200 relations, and 40,000 individuals[18].

According to Smith (2006), an ontology must be "open and available to be used by all potential users without any constraint, other than (1) its origin must be acknowledged and (2) it should not to be altered and subsequently redistributed except under a new name"[19]. Compliant with this demand, OntoCAPE is publicly accessible at http://www.avt.rwth-aachen.de/Ontocape. Via this webpage, both the informal and the formal specification of OntoCAPE can be accessed free of charge. OntoCAPE is distributed under the terms of the GNU General Public License (cf. GNU Project 2007)

3.3 The Meta Model

Fig. 3.3 gives an overview on the Meta Model. As can be observed, the Meta Model is partitioned into the partial models **fundamental_concepts**, **mereology**, **topology**, and **data_structures**. While both **mereology** and **topology** contain only a single module, **data_structures** comprises five: *array*, *linked_list*, *multiset*, *binary_tree*, and *loop*.
The module *meta_model* includes all these modules, thus assembling the ontological definitions of the Meta Model. The module *meta_model* is, in turn, included by the top-level module of the target ontology (shown here is the module *system*, which resides on the upper level of OntoCAPE). That way, the concepts defined in the Meta Model are available in the target ontology.
In the following, we will give a very brief overview on the structure and contents of these four partial models.

3.3.1 Fundamental Concepts

The partial model **fundamental_concepts** introduces meta root terms and their refinements. Meta root terms are the root classes and relations in the Meta Model and all other classes and relations – in the Meta Model as well as in the Onto-

[18] Virtually all of the individuals represent chemical species data.

[19] The formulation has been adopted from the distribution terms of the Open Biomedical Ontologies Foundry, as stated at http://www.obofoundry.org/crit.shtml.

CAPE ontology – can be derived from the meta root terms by specialization. Typical meta root terms are, for example, the classes object and relation class: The former subsumes all "self-standing" (Rector 2003) entities – whether physical or abstract – that exist in an application domain; the latter denotes all kinds of n-ary relations that may exist between objects.

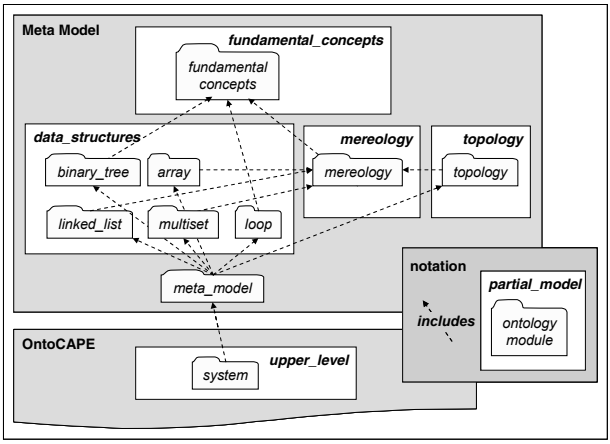

Fig. 3.3: Relations between the modules of the Meta Model and those of OntoCAPE

3.3.2 Data Structures

The partial model **data_structures** provides design patterns for the representation of the following structures, which frequently occur within ontologies.

- The module *array* establishes a pattern for representing an ordered collection of elements. The elements are ordered by an index, which specifies the position of an element within the array through a consecutive sequence of integer values. A particular element can be accessed via its respective index value. The array pattern is applied several times in OntoCAPE, e.g., for the conceptualization of vector quantities (cf. Sect. 4.5.3).
- Similar to an array, a *linked list* is a sequentially ordered collection of elements. The position of an element is defined by pointing to the next (and optionally also to the previous) element in the list. The pattern for linked lists is, for example, utilized to represent the version history of a document (cf. Sect. 4.5.4).
- A *multiset* differs from an ordinary set in that there may be multiple appearances of the same element (e.g., the multiset {a, b, b, b, c, c} has three

appearances of element b). The corresponding design pattern provides a shorthand for representing such multisets; to this end, each element is assigned a multiplicity, which indicates the number of its appearances in the multiset (e.g., in the above multiset, element b is assigned a multiplicity of 3). Amongst others, this pattern is used to conceptualize reaction stoichiometry in OntoCAPE (cf. Sect. 4.5.2).

- A *binary tree* is a tree-like structure that is formed by a set of linked nodes. A node can have zero, one, or two child nodes, which are clearly identified as either the left or the right child node. In OntoCAPE, the pattern for binary trees is particularly utilized to represent mathematical equations (cf. Sect. 4.5.1).
- The design pattern *loop*[20] allows for a compact representation of repetitive structures, the elements of which differ in a systematic manner. Instead of enumerating such structures explicitly, the design pattern models only the first (and optionally also the last) element, the systematic change from one element to the next, and the total number of repetitions. A typical application of this pattern is the representation of mathematical models, which are composed of submodels of the same type (e.g., the tray-by-tray model of a distillation column; cf. Sect. 4.5.5).

3.3.3 Mereology

The partial model **mereology** establishes a theory for describing the relations between parts and wholes.

It follows common best-practice guidelines and takes up an idea from UML to distinguish between aggregation and composition. To that end, aggregation is the binary relation that exists between an aggregate (or whole) and one of its parts. A part may be part of more than one aggregate, i.e., it may be shared by several aggregates. A part can exist independently from the aggregate. Furthermore, a composition is identified as a special type of an aggregation relation, which exists between a composite object and its parts. These parts are non-shareable, i.e., they cannot be part of more than one composite object. If the composite object ceases to exist, its parts cease to exist, as well. For a more comprehensive description refer to Sect. 4.3.

[20] The name ‘loop’ is chosen because the syntax used to represent the loop pattern is similar to that of a ‘for loop’ in a programming language.

3.3.4 Topology

The partial model **topology** establishes a theory for describing topological relations between distributed entities.
The most fundamental concept of module *topology* is the relation isConnectedTo, which denotes the connectivity between objects. The relation is declared to be both symmetric and transitive. A key aspect of our topological theory is to keep mereological and topological relations strictly apart: Only the former or the latter relation can be applied between individuals. This approach, which has been adopted from Borst et al. (1997), enables the formulation of a compact but sufficient theory of mereotopology; theories that do not make this assumption require the definition of additional concepts like overlap, boundary, interior and exterior, etc., which can be avoided here. Please refer to Sect. 4.4 for a more detailed description.

3.4 The Upper Layer

The Upper Layer of OntoCAPE contains only a single partial model, called **upper_level**, which serves the function of a top-level ontology (cf. Sect. 2.6). Thus, the **upper_level** introduces a number of key concepts, which are specialized and refined on the lower layers. Moreover, it establishes the principles of *general systems theory*[21] and *systems engineering*[22], according to which the ontology is organized. The explicit representation of these principles, on the one hand, imparts an overview on the design of OntoCAPE, which helps a user to find his/her way around the ontology; on the other hand, it provides some guidance for extending or refining the ontology.
The concepts introduced by the **upper_level** are generic in the sense that they are applicable to different domains; thus, the partial model resembles the Meta Model 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.

[21] General systems theory is an interdisciplinary field that studies the structure and properties of systems.

[22] 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).

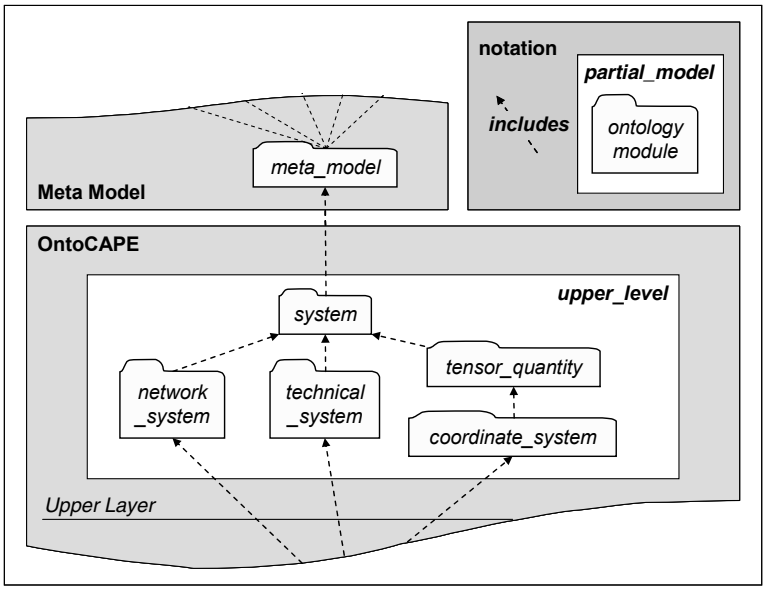

Fig. 3.4: The partial model **upper_level**

The **upper_level** partial model comprises five ontology modules (cf. Fig. 3.4). The module *system* is the most fundamental one of these. It establishes the constitutive systems-theoretical and physicochemical primitives, such as *system*, *subsystem*, *property*, *physical quantity*, *physical dimension*, etc., and specifies their interrelations. It also introduces the concept of an *aspect system* (cf. Sect. 5.1.7), which yields an abstraction of a system with respect to a particular viewpoint and thus allows partitioning a complex system into manageable parts.
As indicated in Fig. 3.4, the *system* module is located at the top of the inclusion hierarchy; it may import the ontology modules of the Meta Model, provided that such an import is desired (cf. the discussion in Chapt. 4). The remaining modules of the **upper_level** complement the *system* module:

- The ontology module *network_system* introduces a structured representation for complex systems, which is applicable to such different domains as biology, sociology, and engineering. To this end, the system is modeled as a network – that is, as a modular structure which "is determined on hierarchical ordered levels by coupling of components and linking elements" (Gilles 1998).
- The ontology module *technical_system* introduces the concept of a *technical system*, which represents a system that has been developed through an engineering design process. 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 viewpoints are explicitly modeled as *aspect systems*.
- The module *tensor_quantity* extends the concept of a *physical quantity* (which is restricted to scalars in module *system*) to vectors and higher-order tensors.
- Finally, module *coordinate_system* introduces the concept of a *coordinate system*, which serves as a frame of reference for the observation of system properties.

For a more extensive description of the **upper_level**, refer to Chap. 5.

3.5 The Conceptual Layer

The Conceptual Layer constitutes the core of OntoCAPE. It is structured into four large partial models, which jointly conceptualize the CAPE domain.

- The partial model **material** provides an abstract description of materials and material behavior.
- The **chemical_process_system** conceptualizes all those notions that are directly related to materials processing and plant operating, such as plant equipment, process flowsheets, control systems, etc.
- The partial model **mathematical_model** defines terms required for a description of mathematical models and model building.
- Finally, **supporting_concepts** supplies auxiliary concepts, such as commonly used physical dimensions, SI units, mathematical expressions, etc. These concepts do not directly belong to the CAPE domain, but support the specification of domain concepts.

In the following, we will give a very brief overview on the structure and contents of these four partial models.

3.5.1 Supporting Concepts

The partial model **supporting_concepts** defines basic notions, such as spatial and temporal coordinate systems, geometrical concepts, mathematical relations, as well as commonly used physical dimensions and SI-units. The concepts defined in this partial model do not belong to the core of the CAPE domain, but are merely utilized by the other partial models of OntoCAPE for defining domain concepts. For that reason, **supporting_concepts** is only rudimentarily developed, as it is not the objective of OntoCAPE to conceptualize areas that are beyond the scope of the CAPE domain. For example, the partial model **mathematical_relation** does not

aim at establishing a full-fledged algebraic theory, as does the EngMath ontology (Gruber and Olsen 1994; cf. Chap. 11); rather, it provides a simple but pragmatic mechanism for the representation of mathematical relations, which serves the needs of the other partial models of OntoCAPE[23].

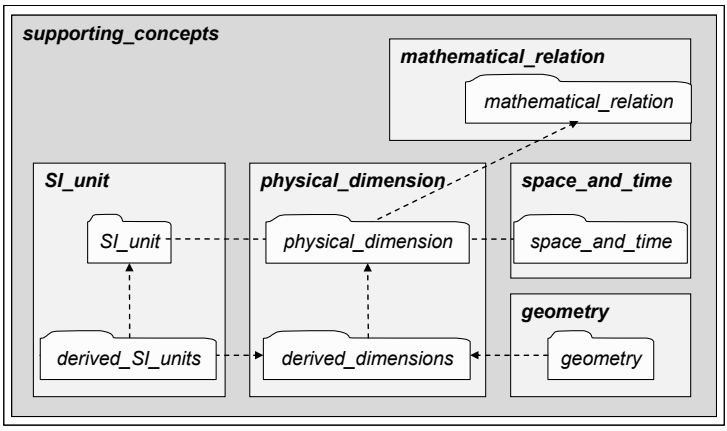

Fig. 3.5: Overview on partial model **supporting_concepts**

As depicted in Fig. 3.5, **supporting_concepts** comprises five subordinate partial models, which are **mathematical_relation**, **physical_dimension**, **SI_unit**, **space_and_time**, and **geometry**.

- Partial model **mathematical_relation** introduces concepts to represent mathematical expressions. However, it is not the objective of this module to describe mathematical models – this is the responsibility of the partial model **mathematical_model** (cf. Chap. 9). Rather, it provides auxiliary concepts, which are utilized by other ontology modules (e.g., for the definition of units).
- Partial model **physical_dimension** comprises two modules. The main module, *physical_dimension*, defines a set of base dimensions and establishes the proceedings to derive further physical dimensions from these base dimensions. It is extended by the module *derived_dimensions*, which introduces a number of frequently used derived dimensions.
- Partial model **SI_unit** comprises the modules *SI_unit* and *derived_SI_units*. The former module introduces the base units of the SI system and establishes a mechanism to derive further units from these. The latter module utilizes this mechanism to define a number of frequently used SI units.

[23] Note that this choice has been made deliberately since a full-fledged algebraic theory is not required in this context. It would only complicate matters and reduce the efficiency (cf. Sect. 10.6) of the ontology.

- Refining the concept of a *coordinate system* introduced on the Upper Layer, the partial model **space_and_time** establishes common types of spatial and temporal coordinate systems. Moreover, it provides concepts for the representation of spatial and temporal points as well as periods of time.
- Finally, the partial model **geometry** provides the concepts for describing the shapes and main dimensions of simple geometric figures.

More detailed descriptions of the above partial models can be found in Chap. 6.

3.5.2 Material

The partial model **material** provides concepts that enable an abstract description of matter. In this context, 'matter' refers to "anything that has mass and occupies space" (Gold et al 1987). The partial model considers only those characteristics of matter that are *independent of a material's concrete occurrence*[24] in time and space; complementary, the partial model **CPS_behavior** (cf. Sect. 8.6) describes the behavior of materials in the concrete setting of a chemical process.
Material comprises two partial models, called substance and phase_system (cf. Fig. 3.6).

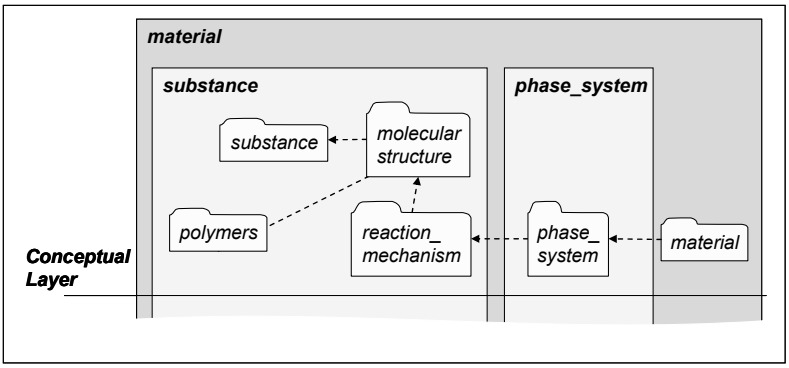

Fig. 3.6: Partial model **material** on the Conceptual Layer

On the Conceptual Layer, **substance** includes the following modules:

- *Substance* is the main module of partial model substance. It provides essential concepts for the description of pure substances and mixtures at the macroscopic scale.

[24] By 'concrete occurrence', we mean the actual spatiotemporal setting – for example, the manufacturing of some material in a chemical plant or its usage as a construction material. For details on this issue, we refer to Morbach et al. (2008j).

- The module *molecular_structure* is concerned with the characterization of pure substances at the atomic scale.
- *Polymers* completes *molecular_structure* with concepts for the description of macromolecular structures.

Finally, *reaction_mechanism* allows representing the mechanism and the stoichiometry of chemical reactions.The partial model **phase_system** comprises a single ontology module, named *phase_system*; it describes the thermodynamic behavior of *materials* subject to a certain *physical context.*
Refer to Chap. 7 for an extensive description of the partial model **material**.

3.5.3 Chemical Process System

Partial model **chemical_process_system** is concerned with the conceptualization of chemical processes and chemical plants. Its key concept is the *chemical process system*, which is a special type of a *technical system* (cf. Sect. 5.3) designed for the production of chemical compounds. The *chemical process system* is characterized from four distinct viewpoints; these viewpoints are modeled as *aspect systems* and are represented in separate partial models (cf. Fig. 3.7):

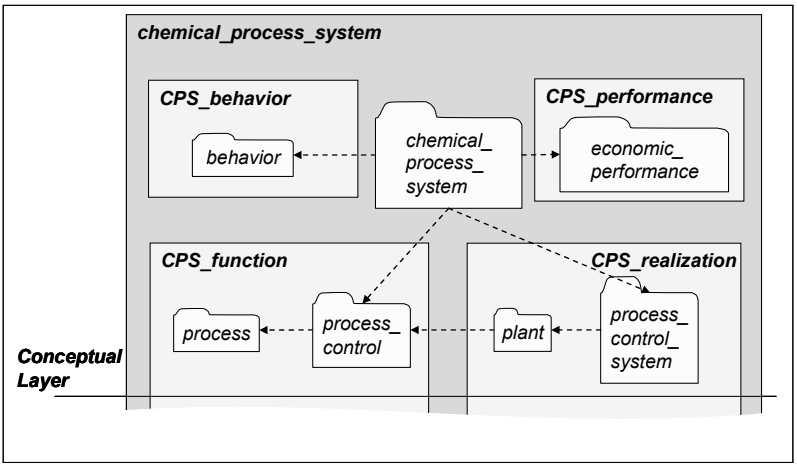

Fig. 3.7: Partial model **chemical_process_system** on the Conceptual Layer

- **CPS_function** enables a functional specification of *chemical process systems*. Its main module *process* describes chemical processes by an approach called 'the phase model of production' (Polke 1994; Bayer 2003). This formalism, which may be transformed into a process flow diagram or a state-task network, is suitable for describing both continuous processes

and batch processes. The supplementary module *process_control* allows the specification of control strategies in terms of function blocks and control loops.

- **CPS_realization** describes the technical realization of a *chemical process system*. The main module *plant* conceptualizes the major types of processing equipment and machinery as well as the connectivity of the equipments via pipes and fittings. Complementarily, the module *process_control_sys-tem* defines the basic components required for process automation, such as measuring instruments, signal lines, and controllers.
- **CPS_behavior** describes the behavior of the *chemical process system*. The module enables both a qualitative and a quantitative-empirical characterization of chemical process behavior – the former is achieved by indicating the prevailing physicochemical phenomena, the latter by indicating the (measured or projected) *values* of the system *properties*.

Finally, the **CPS_performance** evaluates the economic performance of the *chemical process system* in terms of investment and production costs.
A more extensive description of the partial model is given in Chap. 8.

3.5.4 Mathematical Model

The partial model **mathematical_model** is concerned with the description of mathematical models; CapeML (von Wedel 2002) has been used as an important source. Fig. 3.8 gives an overview on **mathematical_model** on the Conceptual Layer. The main module, *mathematical_model*, introduces the basics concepts for mathematical modeling, including model variables and items pertaining to submodels and their connections: A *mathematical model* is conceptualized as a special type of *system*, the properties of which are reflected by its *model variables*.

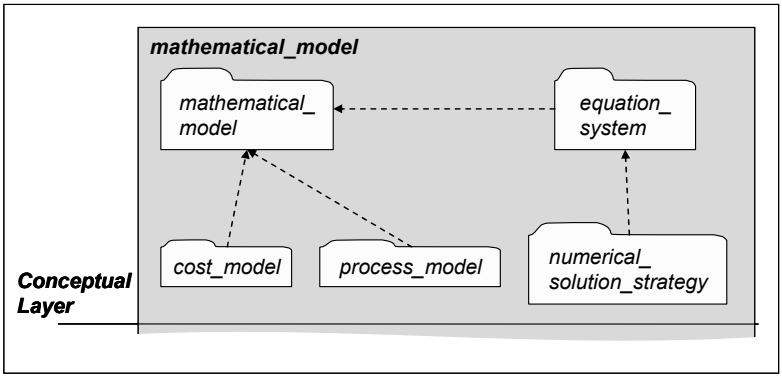

Fig. 3.8: Partial model **mathematical_model** on the Conceptual Layer

The ontology module *equation_system* further specifies the characteristics of the model equations that constitute a *mathematical model*. Based on these characteristics, an appropriate numerical solver can be selected, which is the concern of the ontology module *numerical_solution_strategy*. The modules *process_model* and *cost_model* describe two particular types of mathematical models: *process models* model the behavior of *chemical process systems* and *materials*, while *cost models* predict the costs of *chemical process systems*.
Further information about the partial model can be found in Chap. 9.

3.6 The Application Layers

The Application-Oriented Layer and the Application-Specific Layer (collectively referred to as 'application layers') extend the ontology towards concrete applications. As explained in Sect. 3.1.1, the major difference between the two application layers is that the concepts of the Application-Oriented Layer are relevant for a whole range of applications, whereas the concepts of the Application-Specific Layer are rather intended for one particular application.
Typically, the terms introduced on the application layers are instances or specializations of terms introduced on the Conceptual Layer. Accordingly, the modules located on the application layers do not open up new subject areas, but complete and refine existing partial models. These modules can thus be considered as application (domain) ontologies: As explained in Sect. 2.6, an application ontology holds application-specific, but state-independent information.
Over the lifecycle of OntoCAPE, new modules are expected to appear on the application layers with each new application encountered. For the applications realized so far (cf. Chapt. 12), the following extensions have been developed:

- Partial model **material** is extended by modules holding instance data about chemical elements, pure substances, and technical polymers. Further modules provide classification schemata for substance classes (such as alcohols or esters) and for types of chemical reactions (such as esterification or hydrohalogenation).
- Partial model **chemical process system** is refined by a number of classification schemata, which introduce special types of unit operations, of plant equipment (i.e., apparatuses, machines, and fixtures), of measuring and control instruments as well as of controller types. Furthermore, the behaviors of typical *process units*[25] are modeled in an application-oriented extension of the partial model **CPS_behavior**.

[25] The term *process unit* denotes an elementary subsystem of a *chemical process system*, such as a reactor, a heat exchanger, or a distillation column.

– Within the partial model **mathematical_model**, a number of modules are added to complement the module *process_model*. For sake of illustration, these add-ons will be discussed in more detail below.

As explained in Chap. 9, the module *process_model* enables the definition of specialized *process models*, which model the behavior of *chemical process systems* and *materials*. Typically, a *process model* is composed of *submodels*, of which *laws* and *property models* are important subtypes. A *law* constitutes the mathematical representation of a scientific law, such as the law of energy conservation. A *property model*, in turn, represents a mathematical correlation for the computation of a single physical property. Typical property models would be vapor pressure correlations or activity coefficient models.
The terms *law* and *property model* have already been defined on the Conceptual Layer in the module *process_model*. On the Application-Oriented Layer, *process_model* is extended by the ontology modules *laws* and *property_models* (cf. Fig. 3.9). The former module establishes models for a number of physical laws that are common in the context of chemical engineering. These laws represent

– the conservation of energy, mass, and momentum;
– thermal, mechanical, or phase equilibrium, as well as reaction equilibrium;
– non-equilibrium transport phenomena, such as diffusion.

Similarly, the *property_models* module defines special types of *property models*, which can be categorized into three major classes:

– *Chemical kinetics property models* specify how to calculate the rate coefficients of homogenous or heterogeneous reactions.
– *Phase interface transport property models* provide correlations for computing certain *phase interface transport properties*.
– *Thermodynamic models* indicate the correlations between certain *intensive thermodynamics state variables* and *intra-phase transport properties*.

Module *process_model* is furthermore extended by module *process_unit_models*, which defines a number of specialized *process models* for customary *process units*. Examples of such *process unit models* are a *CSTR model* or a *tray-by-tray distillation column model*. Typically, a *process unit model* includes one or several *laws* (e.g., a *tray-by-tray distillation column model* includes a *phase equilibrium law*); therefore, module *laws* is imported by *process_unit_models*.
The above described modules are all located on the Application-Oriented Layer. By contrast, the module *aspen_pus_model* constitutes a further extension of module *process_model* on the Application-Specific Layer. It has been developed for an application in knowledge management, which is presented in Sect. 12.2. Basically, the module provides concepts for the semantic annotation of simulation documents in the format of the simulation software *Aspen Plus* (AspenTech 2008). Such documents contain the specification of an *Aspen Plus model*, which is a spe-

cial type of a *process model*. The average *Aspen Plus model* includes at least one *property model* as well as a number of *process unit models*. The latter are *process unit models* of particular types, which are provided by the model library of the simulation software. To give an example, the Aspen Plus model library holds a model named *RadFrac*, which is a particular implementation of a *tray-by-tray distillation column model*. In contrast to the more general *tray-by-tray distillation column model*, the *RadFrac* model has a number of preassigned properties – such as being a closed-form model of nonlinear algebraic type.

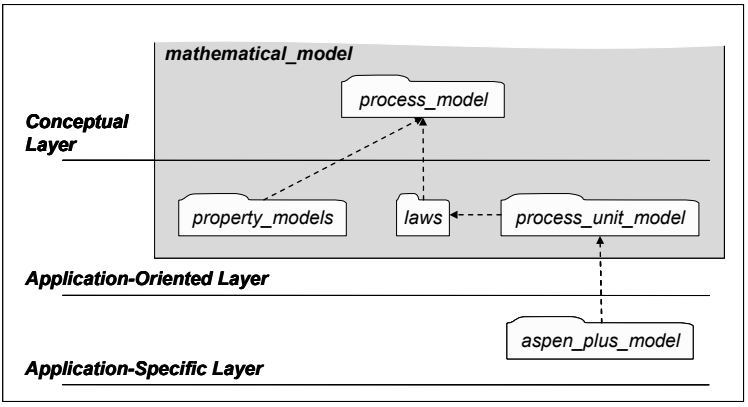

Fig. 3.9: Extension of the module *process_model* on the application layers

3.7 References

AspenTech (2008) *Aspen Plus*. Online available at http://www.aspentech.com/products/aspen-plus.cfm. Accessed June 2008.

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

Bernaras A, Laresgoiti I, Corera J (1996) Building and reusing ontologies for electrical network applications. In: Wahlster W (ed.): *ECAI 1996 – Proceedings of the 12th European Conference on Artificial Intelligence*. Wiley, Chichester:298–302.

Borst P, Akkermans JM, Top JL (1997) Engineering ontologies. *Int. J. Hum Comput Stud.* **46**:365–406.

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

Bray T, Hollander D, Layman A, Tobin R, eds. (2006a) *Namespaces in XML 1.0 (Second Edition).* W3C Recommendation, 16 August 2006. Online available at http://www.w3.org/TR/xml-names. Accessed September 2008.

Chandrasekaran B, Johnson TR (1993) Generic tasks and task structures: history, critique and new directions. In: David JM, Krivine JP, Simmons R (eds.): *Second Generation Expert Systems*. Springer, New York:232–272.

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

GNU Project (2007) The GNU General Public Licence. Online available at http://www.gnu.org/copyleft/gpl.html. Accessed December 2007.

Gold V, Loening KL, McNaught AD, Sehmi P (1987) Compendium of Chemical Terminology. Blackwell, Oxford.

Gruber TR (1995) Toward principles for the design of ontologies used for knowledge sharing. *Int. J. Hum Comput Stud.* **43** (5/6):907–928.

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.stanford.edu/knowledge-sharing/pa-pers/engmath.html. Accessed September 2007.

Heflin J, Hendler J (2000) Dynamic ontologies on the web. In: Proceedings of the Seventeenth National Conference on Artificial Intelligence (AAAI-2000). AAAI Press, Menlo Park, CA:443–449.

Jarrar M, Meersman R (2002) Scalability and knowledge reusability in ontology modeling. In: *Proceedings of the International Conference on Infrastructure for e-Business, e-Education, e-Science, and e-Medicine* SSGRR2002.

Morbach J, Wiesner A, Marquardt W (2008f) *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, Bayer B, Wiesner A, Yang A, Marquardt W (2008g) OntoCAPE 2.0 – The Upper Level. Technical Report (LPT-2008-25), Lehrstuhl für Prozesstechnik, RWTH Aachen University. Online available at http://www.avt.rwth-aachen.de/AVT/index.php?id=541&L=0&Nummer=LPT-2008-25.

Morbach J, Yang A, Wiesner A, Marquardt W (2008h) *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.

Morbach J, Yang A, Marquardt W (2008i) *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/index.php?id=541&L=0&Nummer=LPT-2008-27.

Morbach J, Yang A, Marquardt W (2008j) *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.

Noy NF, Klein M (2004) Ontology evolution: not the same as schema evolution. *Knowl. Inform. Syst.* **6**:428–440.

Pinto HS, Gomez-Perez A, Martins JP (1999) Some issues on ontology integration. In: *Proceedings of the IJCAI'99 Workshop on Ontologies and Problem Solving Methods*.

Polke M, ed. (1994) *Process Control Engineering*. VCH, Weinheim.

Pure-Systems (2008) *Pure::Variants*. Online available at http://www.pure-systems.com. Accessed September 2008.

Rector A (2003) Modularisation of domain ontologies implemented in description logics and related formalisms including OWL. In: Genari J (ed.): *Knowledge Capture 2003*. ACM Press:121–128.

Russ T, Valente A, MacGregor R, Swartout W (1999) Practical experiences in trading off ontology usability and reusability. In: *Proceedings of the 12th Banff Knowledge Acquisition for Knowledge-Based Systems Workshop*. SRDG Publications.

Smith B (2006) Against idiosyncrasy in ontology development. In: Bennett B, Fellbaum C (eds.): *Formal Ontology in Information Systems*. IOS Press:15–26.

Stuckenschmidt H, Klein M (2003) Integrity and change in modular ontologies. In: Gottlob G, Walsh T (eds.): *IJCAI-03 – Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence*. Morgan Kaufmann:900–905.

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

Visser PRS, Cui Z (1998) Heterogeneous ontology structures for distributed architectures. In: *Proceedings of the ECAI-98 Workshop on Applications of Ontologies and Problem-Solving Methods*:112–119.

von Wedel L (2002) *CapeML – A Model Exchange Language for Chemical Process Modeling*. Technical Report (LPT-2002-16), Lehrstuhl für Prozesstechnik, RWTH Aachen University.

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-2008-26.

http://www.springer.com/978-3-642-04654-4

OntoCAPE
A Re-Usable Ontology for Chemical Process Engineering
Marquardt, W.; Morbach, J.; Wiesner, A.; Yang, A.
2010, XVII, 481 p. 286 illus., Hardcover
ISBN: 978-3-642-04654-4