9 Mathematical Models

The partial model **mathematical_model** is concerned with the description of mathematical models. Fig. 9.1 gives an overview of the ontology modules of **mathematical_model** and their interrelations. The main module, *mathematical_model* (cf. Sect. 9.1), introduces the basic concepts for mathematical modeling, including model variables as well as items pertaining to sub-models and their connections. CapeML (von Wedel 2002) was taken as an important source. The ontology module *equation_system* (cf. Sect. 9.2) 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* (cf. Sect. 9.3). The modules *process_model* (cf. Sect. 9.5) and *cost_model* (cf. Sect. 9.4) describe two particular types of mathematical models: *process models* model the behavior of *process units* (cf. Sect. 8.8) and *materials* (cf. Sect. 7.1), while *cost models* predict the *costs* (cf. Sect. 8.7.1) of *chemical process systems*.

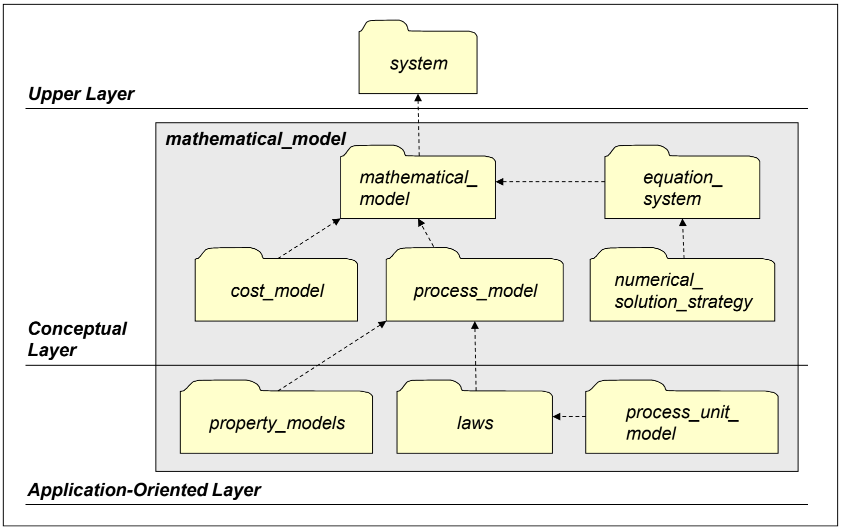

Fig. 9.1: Overview on partial model **mathematical_model**

The *process_model* module is extended on the Application-Oriented Layer: The ontology module *laws* (cf. Sect. 9.6) establishes models for a number of physical laws that are common in the context of chemical engineering (e.g., the law of energy conservation). Thus, laws may be associated with the phenomena introduced and described in *behavior* (Sect. 8.6.1). *Property_models* (cf. Sect. 9.7) provides correlations for designated *physical quantities*, such as vapor pressure correlations or activity coefficient models. Finally, the module *process_unit_model*

W. Marquardt et al., *OntoCAPE*, RWTHedition, DOI 10.1007/978-3-642-04655-1_9,

(cf. Sect. 9.8) establishes customary mathematical models for process units, such as ideal reactor models or tray-by-tray models for distillation columns.

9.1 Mathematical Model (Ontology module)

A *mathematical model* is a special type of *model* (cf. Sect. 5.1.6), which uses mathematical expressions to describe the behavior of the modeled *system,* for example by means of simulation.

9.1.1 High-Level Concepts

A *mathematical model* has a number of *properties*, the most important of which are *model quantities*. As indicated in Fig. 9.2, a *model quantity* is a subclass of *physical quantity* that is linked to the model via the relation hasVariable (a specialization of hasProperty, cf. upper right corner of Fig. 9.2). Like any *physical quantity*, a *model quantity* has a particular *physical dimension* and can be either a *scalar quantity* (cf. Sect. 5.1.11) or a *tensor quantity* (cf. Sect. 5.5). The *value* of a *model quantity* is represented by the class *model quantity specification*; each *model quantity* has exactly one *model quantity specification*.

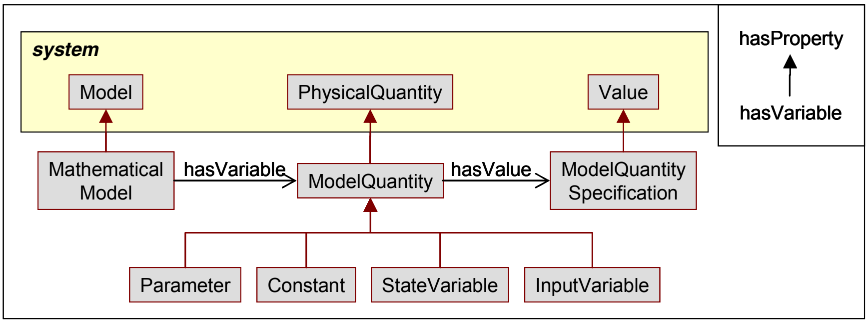

Fig. 9.2: *Mathematical model*, *model quantity*, and *model quantity specification*

A *model quantity* can be one of the following types: a *constant*, a *parameter*, a *state variable,* or an *input variable*, depending on the intended specification of its value: *Constants* and *parameters* constitute the fixed set of specified variables. *Input variables* represent time or spatially dependent inputs, which have to be specified for dynamic and/or spatially distributed systems. Finally, *state variables* constitute the fixed set of unknown variables, which have to be computed by the model. The *model quantity specification* indicates the numericalValue (cf. Sect. 5.1.11) of the *model quantities*. Unlike *constants*, *parameters* and *input variables* may have different *model quantity specifications* in different simulation runs. If the *model quantity* is of type *parameter* or *state variable*, the *model quantity specification* may indicate their *upper limit* and *lower limit* (cf. Fig. 9.3).

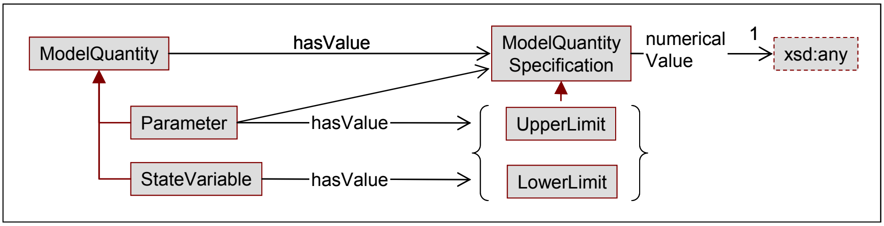

Fig. 9.3: Assignment of particular *model quantity specifications* to *model quantites*

A *system* which is the target of a models relation is classified as a *modeled object*. The correspondences between a *model quantity* and a *physical quantity* of the *modeled object* can be explicitly represented by means of the relation corresponds-ToQuantity (cf. Fig. 9.4).

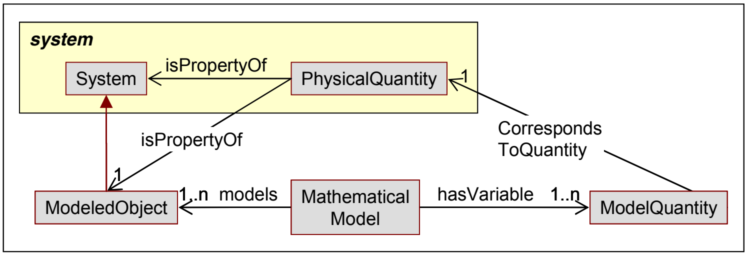

Fig. 9.4: Correspondence between a *model quantity* and a *physical quantity* of the *modeled object*

9.1.2 Modeling of Decomposition

Like any *system*, a *mathematical model* can be decomposed into *subsystems*, which are called *submodels*. The *submodel* models the same *system* as its superordinate *mathematical model*. Consequently, there is no need to specify the models relation between *submodel* and *system* explicitly. However, such a relation may be indicated if the *submodel* models a designated *subsystem* of the overall *system*.

9.1.3 Modeling of Connectivity

The different *submodels* of a *mathematical model* are coupled via their *model quantities* as explained in the following.
At first, the concept of a *model port* is introduced. A *model port* is a special type of *property set*, which comprises *model quantities* that can participate in a connection with another model. Thus, a *model port* has the function to identify and to bundle the "public" variables of a *mathematical model.*
Next, the concept of a *coupling* is established. A *coupling* is a property of the overall *mathematical model*, which defines a connection between two of its *submodels* by linking their respective *model ports*[96]. The *coupling* implicitly defines equality constraints between the *model quantities* in the two *model ports* and must be treated as such (e.g., during a degrees-of-freedom analysis of a complex model). It may be used to connect *mathematical models* both 'horizontally' (i.e., on the same level of decomposition) and 'vertically' (i.e., across levels of decomposition).
The order of the *model quantities* within a *model port* can be specified by a *port index*, as shown in Fig. 9.5. The *port index* is used to identify corresponding *model quantities* in a *coupling*: Two *model quantities* of different *model ports* are coupled if and only if their *port indices* have the same indexValues. The specification of a *port index* may be omitted if the correspondence between *model quantities* is evident from the context (e.g., if each of the coupled *model ports* comprises only a single *model quantity*, or if corresponding *model quantities* can be uniquely identified through their *physical dimension*.)

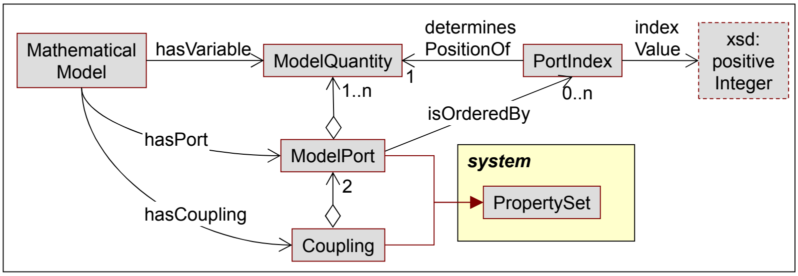

Fig. 9.5: Variables, ports, couplings

Fig. 9.6 shows exemplarily the definition of a *mathematical model* **M**, which consist of two *submodels*, **M1** and **M2**. **M1** has the *model quantities* **a**, **b**, and **c**, while **M2** has the *model quantities* **x**, **y**, and **z**. Model **M1** owns the *model port* **P1**, which comprises the quantities **b** and **c**. Similarly, the *model port* **P2** of model **M2** comprises the

[96] Please note that this way of modeling does not contradict the principles stated in *network_system* (cf. Sect. 5.2) since *model port* and *model quantity* are subclasses of *property* and are not considered as subsystems.

quantities **y** and **z**. **P1** and **P2** are coupled via the *coupling* **C**, which is a property of the overall model **M**. The corresponding quantities of the *coupling* are identified via their *port indices*: **b** and **y** have the same indexValue and are thus linked by an equality constraint. The same holds true for quantities **c** and **z**.

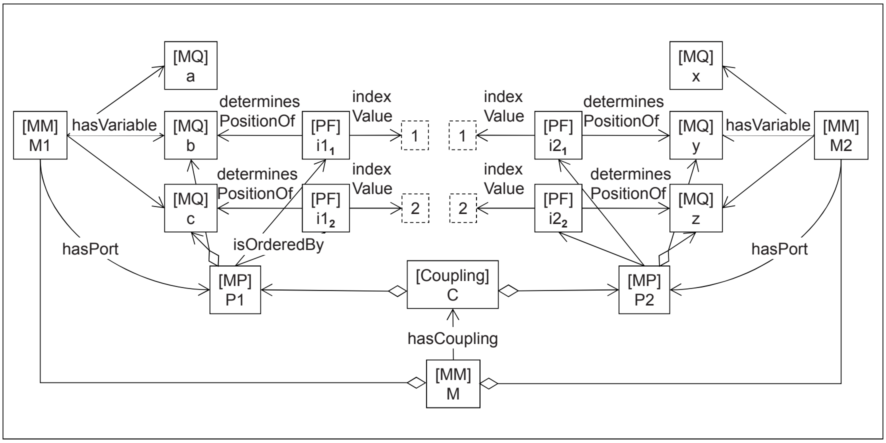

Fig. 9.6: Exemplary decomposition of model **M** into submodels **M1** and **M2**

9.1.4 Usage

The ontology module *mathematical_model* provides only the basic concepts for the description of mathematical models. For practical applications, further concepts may be required, which would typically be supplied by additional ontology modules located on the Application-Oriented Layer. Some possible extensions of the *mathematical_model* module are discussed next.

One possible extension would be the introduction of concepts suitable for representing model equations. Such an extension could be realized easily by reusing the concepts of the *mathematical_relation* module (cf. Sect. 6.1). However, such an extension is not required in practice, since specialized representation formats for mathematical equations are available, such as MathML (Ion and Miner 1999), CapeML (von Wedel 2002), or CellML (Lloyd et al. 2004).

Another possible extension would be the definition of different types (i.e., subclasses) of *model ports*. A particular *model port* type could, for example, prescribe the number of *model quantities* comprised in a *model port*, their types (i.e., *constant*, *parameter*, *input variable* or *state variable*), their *physical dimensions*, etc. Moreover, a *model port* type could be further characterized through attributes (e.g., assigning a direction to a *model port*, thus turning it into either an *inlet port* or an *outlet port*). That way, standardized model interfaces can be defined – for instance, one may define a standard *energy port*, which contains a single scalar *model quantity* with the *physical dimension* of an energy flow and must furthermore be tagged as an *inlet* or *outlet port*. Such standardization facilitates checking the feasibility of a *coupling*: A *coupling* of two *mathematical models* will be feasible if their *model ports* (a) are of the same type (e.g., *energy port*) and (b) have matching attributes (e.g., an *inlet port* can only be coupled to an *outlet port*).

In practice, a *mathematical model* often consists of several interconnected *submodels* of the same type – for example, the model of a distillation column contains several models of distillation column trays. An application-oriented extension of *mathematical_model* could apply the *loop* design pattern introduced in the Meta Model (Sect. 4.5.5) to define such repetitive model structures. An example is given in Fig. 9.7 and Fig. 9.8.

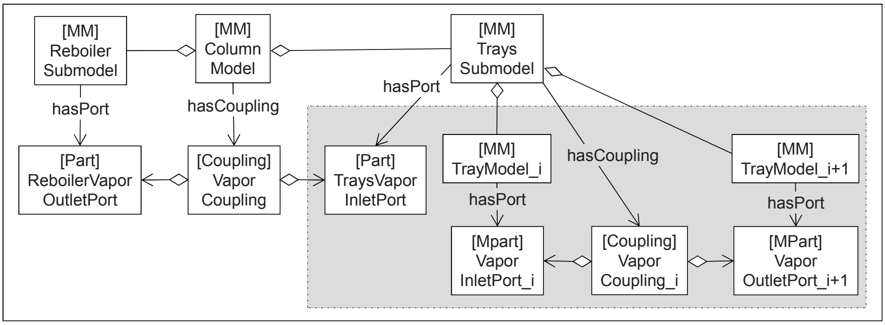

Fig. 9.7: Specification of the overall model

Fig. 9.7 specifies the overall structure of a **Column Model**. It consists of a **Reboiler Submodel** and a **Trays Submodel**, which are coupled via a **Vapor Coupling** (to simplify matters, the liquid phase is not considered in this example). The **Trays Submodel** is defined iteratively (see grey-shaded area in Fig. 9.7). It consists of several *submodels* of the same type, which are represented by the individual **TrayModel_i**. Each **TrayModel_i** has a **VaporInletPort_i**, which is coupled to the **VaporOutletPort_i+1** of the next **TrayModel_i+1**. This connectivity statement is included in a **ForLoop** that counts from 1 to 20, as shown in Fig. 9.8, to define a structure of 20 interconnected tray models. The vapor inlet port of the 20th tray model corresponds to the previously defined **TraysVaporInletPort**.

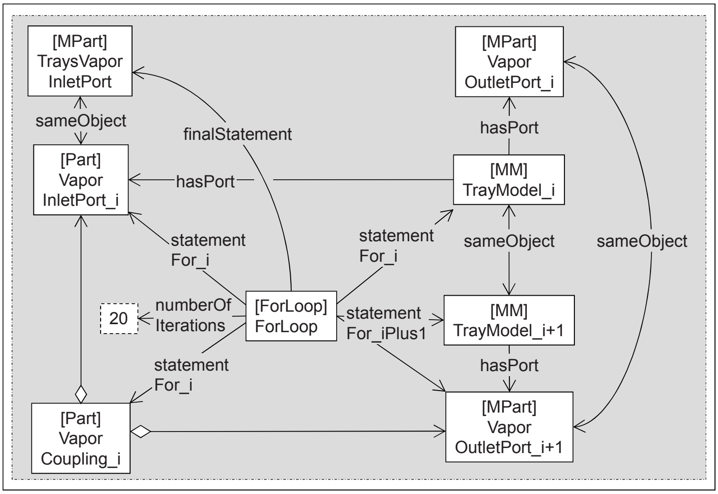

Fig. 9.8: Specification of the repetitive submodel structure

9.1.5 Concept Descriptions

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

Class Descriptions

Constant
A *constant* is a specific *model quantity*, the *model quantity specification* of which has a constant numericalValue in all simulation runs.

Coupling
A *coupling* connects two *model ports* of different *submodels*, thereby defining equality constraints between *model quantities* comprised in the two *model ports*.

Input variable
Input variables represent time- or space-depenent inputs, which have to be specified for dynamic and/or spatially distributed systems.

Lower limit
An *lower limit* is *model quantity specification* which defines an lower bound for the numericalValue of a *model quantity specification*.

Mathematical model
A *mathematical model* is a *model* that uses mathematical language to describe the modeled *system.*

Modeled object
A *system* that is modeled by means of a *model* is denoted as a *modeled object*.
Formal definition: A *modeled object* is a *system* that isModeledBy a *model*.

Model port
A *model port* is a collection of *model quantities* that can participate in a connection with another *mathematical model*. Thus, a *model port* has the function to identify and to bundle the "public" variables of a *mathematical model*. Optionally, a *model port* can be ordered by a *port index*.

Model quantity
A *model quantity* represents a *physical quantity* involved in a *mathematical model*, the *value* of which can be either supplied by the modeler or a computed from an evaluation of the *mathematical model*.
Formal definition: A *model quantity* is either a *state variable* or a *parameter* or a *constant*.

Model quantity specification
A *model quantity specification* specifies a *model quantity* in terms of its numerical value (or limits of its numerical value) and its unit of measurement.

Parameter
A *parameter* is a specific *model quantity* (i.e., an input variable), the *model quantity specification* of which may take different numericalValue in different simulation runs.

Port index
A *port index* orders the *model quantities* comprised in a *model port* by assigning each of them an indexValue. In a *coupling*, *model quantities* with the same indexValue are coupled to each other.

Submodel
A *mathematical model* can be decomposed into *submodels*.
Formal definition: A *submodel* is a direct subsystem of a *mathematical model*.

State variable
State variables constitute the fixed set of unknown variables which have to be computed by the model.
Its *model quantity specification* either indicates the upperLimit and lowerLimit of the *model quantity* (before solving the model) or its numericalValue (after solving the model).

Upper limit
An *upper limit* is a *model quantity specification* which defines an upper bound for the numericalValue of a *model quantity specification*.

Relation Descriptions

correspondsToQuantity
The relation denotes a one-to-one correspondence between a *model quantity* and a *physical quantity* of the *modeled object*.

hasCoupling
The relation indicates a *coupling* between two *submodels* of a *mathematical model*.

hasModelPort
The relation identifies the *model port* of a *mathematical model*.

hasModelVariable
The relation indicates the *model quantities* of a *mathematical model*.

determinesPositionOf
The one-to-one relation between a *port index* and the corresponding *model quantity*.

isIndexOf
The relation isIndexOf points from a *port index* to the associated *model port*.

isOrderedBy
The relation isOrderedBy points from a *model port* to its sorting *port index*.

Attribute Descriptions

indexValue
The attribute indexValue indicates the numerical value of a *port index*.

9.2 Equation System

The ontology module *equation_system* provides concepts for the description of the model equations that constitute a *mathematical model*. The model equations are not explicitly represented, only their *equation system characteristics* are specified. Moreover, the scope of *equation system characteristics* is confined to those characteristics that are of relevance for selecting an appropriate solver and/or solution strategy for the *mathematical model* (cf. Sect. 9.3).
A *mathematical model* can be classified according to different criteria including equation system type, *variables type*, *model representation form*, etc. Following the recommendations for ontology normalization[97] given by Rector (2003), *equation systems* are classified along a two axes (cf. Fig. 9.9): (1) using the equation system type as a differentiating criterion and (2) referring to the linearity of mathematical

[97] More details on this issue can be found in Sect. 4.2.

models, as described in Sect. 4.2.1. The other possible criteria are explicitly modeled as *equation system characteristics*, which are linked to a *mathematical model* via the relation hasCharacteristic (or one of its specializations, cf. upper left corner of Fig. 9.9). Note that some of the *equation system characteristics* can only be assigned to special types of *equation systems*; for instance, *DAE type* only applies to *differential algebraic equation systems*.

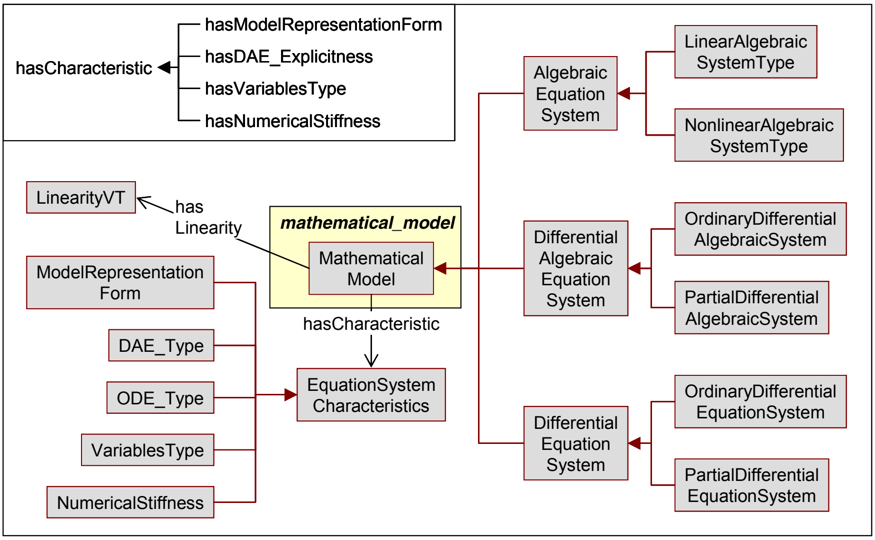

Fig. 9.9: Equation system characteristics

While the meaning of most concepts displayed in Fig. 9.9 should be evident from their names, the concept of *model representation form* requires some explanation. A *mathematical model* may appear in two representation forms, which are termed **open-form** and **closed-form**.

An **open-form** model does not provide a solution method to solve its model equations. A numerical solver needs to be applied to the model to obtain a solution solving the simulation. Hence, the **open-form** model must provide all the information required by the external numerical algorithm to solve the model. For example, a model representing a set of algebraic equations may provide equation residuals and derivatives to a Newton solver. Before an **open-form** model can be successfully solved, it has to be "squared", meaning that the number of its unknown variables must be the same as that of its equations. Among all the *model quantities* of an **open-form** model, those declared as *constants*, *input variables*, or *parameters* have to be given values (i.e., they need to be assigned a *model quantity specification* with a definite numericalValue) before the model can be evaluated (cf. Sect. 9.1). The other variables are *state variables*. If there are still more *state variables* than equations in a model, it is necessary to assign values to some selected variables (i.e., turn them into *parameters* or *input variables*). Generally, one can freely choose the set of

model variables of an **open-form** model to be specified, as long as the model remains solvable. The values of the remaining variables can be obtained by solving the model.
A **closed-form** model includes an underlying numerical algorithm, which solves its model equations. Thus, it does not require any external solver for obtaining the values of its unknown variables. The "execution" of the **closed-form** model yields the values of a set of selected unknown variables, the so-called outputs, based on the given values of the specified variables. In this process, the algorithm of a **closed-form** model accepts only a fixed set of input variables, and consequently returns a fixed set of output variables. No choice for specifying additional variables is available, as in the case of **open-form** models. Reflected in *model quantity* types, *constants* and *parameters* constitute the fixed set of specified variables, while the *state variables* constitute the fixed set of unknown variables (i.e. output variables).

9.2.1 Usage

The ontology module *equation_system* provides the basic concepts for the identification of mathematical models from a mathematical point of view (e.g., whether it is an ODE or DAE) This identification was primarily of concern in the COGents project (cf. Sect. 12.1.1), where this module was applied to specify the type of mathematical model to search for in various libraries.
Typically, mathematical models may be classified either by means of content (e.g., a mathematical model for a polyethene reactor) or simply by mathematical features, as it is done here. As an example, consider a process engineer who searches for a particular mathematical model, which is supposed to be applied for the calculations of a reactor. Depending on the software (e.g., he might have only a solver for ODEs available), a classification with respect to the characteristics (e.g., ODE type) is extremely helpful to identify the suitable mathematical model.

9.2.2 Concept Descriptions

Individual concepts of the module *equation_system* are defined below. For an extensive description of the introduced individuals, we refer to Morbach et al. (2008j).

Class Descriptions

Algebraic equation system
An *algebraic equation system* is a *mathematical model* which solely consists of algebraic equations.
Formal definition: An *algebraic equation system* is either a *linear algebraic system* or a *nonlinear algebraic system*.

DAE type
Characterizes the explicitness of a *differential algebraic equation system*.
Formal definition: The class *DAE type* is an exhaustive enumeration of the individuals **fully_implicit** and **semi-explicit**.

Differential algebraic equation system
A *differential algebraic equation system* (DAE system) is a *mathematical model* that comprises both algebraic and differential equations.
Formal definition: A *differential algebraic equation system* is either an *ordinary differential algebraic system* or a *partial differential algebraic system*.

Differential equation system
A *differential equation system* is a *mathematical model* that solely consists of differential equations.
Formal definition: A *differential equation system* is either an *ordinary differential equation system* or a *partial differential equation system*.

Equation system characteristics
The *equation system characteristics* characterize the model equations of a *mathematical model*.

Linear algebraic system type
A *linear algebraic system type* is an *algebraic system* which contains only linear equations.
Formal definition: A *linear algebraic system type* is an *algebraic system* that is characterized as **linear**.

Linearity VT
Linearity VT characterizes the linearity of a mathematical model.
Formal definition: Linearity is an exhaustive enumeration of the individuals **linear** and **nonlinear**.

Model representation form
A *mathematical model* may appear in two forms, as indicated by the *model representation form*:
- An **open-form** model is solved by an external algorithm. One can freely choose the inputs and outputs of the **open-form** model.

- A **closed-form** model includes an underlying numerical algorithm that solves the model equations. The algorithm accepts only a fixed set of input variables, and consequently returns only a fixed set of output variables.

Formal definition: The class *model representation form* is an exhaustive enumeration of the individuals **open-form** and **closed-form**.

Nonlinear algebraic system type
Formal definition: A *nonlinear algebraic system type* is an *algebraic equation system* that is characterized as **nonlinear**.

Numerical stiffness
In mathematics, stiff equations are equations where certain implicit methods, in particular BDF, perform better, usually tremendously better, than explicit ones (Hairer and Wanner 1996).
Formal definition: The class *numerical stiffness* is an exhaustive enumeration of the individuals **stiff** and **nonstiff**.

ODE_type
Characterizes the explicitness of an *ordinary differential equation system*, which can be given in **implicit_formulation** or **explicit_formulation**.
Formal definition: *ODE_types* is an exhaustive enumeration of the individuals **implicit_formulation** and **explicit_formulation**.

Ordinary differential algebraic system
An *ordinary differential algebraic system* comprises algebraic equations as well as ordinary differential equations, but no partial differential equations.

Ordinary differential equation system
An *ordinary differential equation system* (ODE system) is a *differential equation system* which solely consists of ordinary differential equations.

Partial differential algebraic system
A *partial differential algebraic system* is a *differential algebraic equation system* which comprises both partial differential equations and algebraic equations.

Partial differential equation system
A *partial differential equation system* (PDE system) is a *differential equation system* which consists of partial differential equations.

Variables type
A *variables type* indicates whether the *model quantities* of a *mathematical model* are all continuous, all discrete, or partly continuous and partly discrete.
Formal definition: The class *variables type* is an exhaustive enumeration of the individuals **continuous**, **discrete**, and **mixed**.

Relation Descriptions

hasDAE_Type
Indicates an *equation system characteristic* of type *DAE type.*

hasLinearity
Refers from a *mathematical model* to a *linearity value type*.

hasModelRepresentationForm
Indicates an *equation system characteristic* of type *model representation form*.

hasNumericalStiffness
Indicates an *equation system characteristic* of type *numerical stiffness*.

hasODE_Type
Indicates an *equation system characteristic* of type *ODE type.*

hasVariablesType
Indicates an *equation system characteristic* of type *variables type*.

Attribute Descriptions

differentialIndex
The attribute represents the differential index of an ordinary *differential algebraic equation system*, as defined by Gear and Petzold (1984) or of a partial differential algebraic system, as defined by Martinson and Barton (2000).

differentialOrder
The attribute differentialOrder denotes the order of a differential equation, which is defined as the order of the highest derivative of a *model quantity* appearing in the differential equation.

9.3 Numerical Solution Strategy

In this ontology module, strategies for solving *mathematical models* are defined. At present, it is confined to numerical solution strategies only. A classification of numerical solution techniques is given, and the ability of a strategy to solve a particular type of *mathematical model* is explicitly specified. The major concepts are shown in Fig. 9.10. A *model solution strategy* solves a *mathematical model*; the subclasses of *model solution strategy* represent different types of numerical algorithms, which are specifically designed to solve a certain type of *mathematical model* with certain *equation system characteristics*. To this end, a *model solution strategy* may apply some other, specialized *model solution strategy*. So far, only numerical solution strategies have been considered in OntoCAPE, but symbolic/analytical solution methods could be added in an analogous manner.

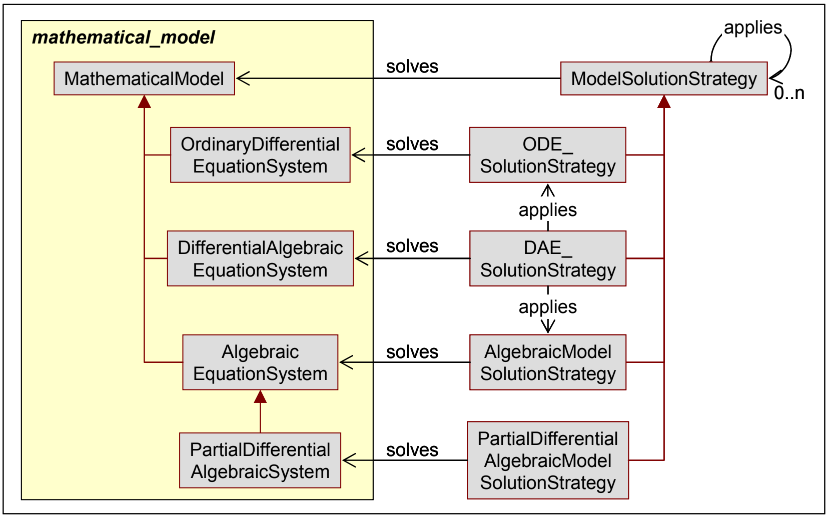

Fig. 9.10: Numerical solution strategy.

Fig. 9.11 shows the refinement of class *algebraic model solution strategy.* An exemplary *linear algebraic model solution strategy* is Gauss-elimination, an example of a *nonlinear algebraic model solution strategy* is Newton's method.

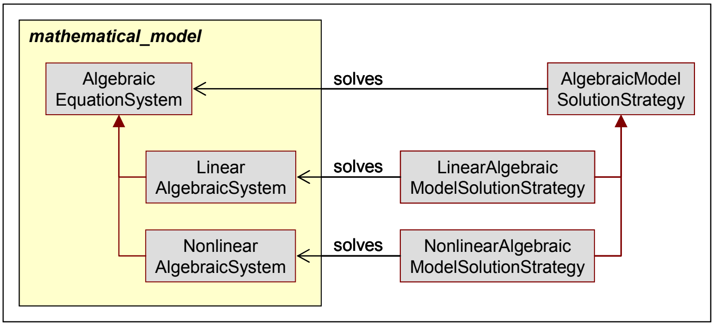

Fig. 9.11: Types of *algebraic model solution strategies*

An *ODE solution strategy* can be further characterized by indicating if the algorithm is a **one-step_method** (e.g., the classical Runge-Kutta methods) or a **multi-step_method** (e.g., the Adams-Bashforth methods). Moreover, it can be specified whether the algorithm is a *solution strategy for explicit ODEs* or *implicit ODEs* (cf. Fig. 9.12.

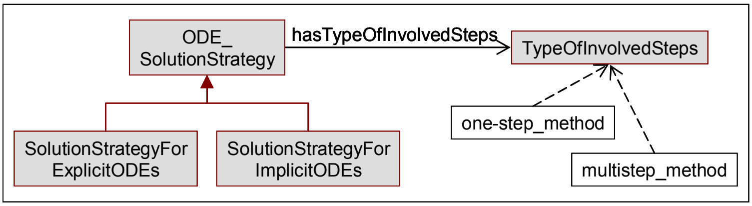

Fig. 9.12: Further specification of *ODE solution strategy*

9.3.1 Concept Descriptions

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

Class Descriptions

Algebraic model solution strategy
An *algebraic model solution strategy* is a *model solution strategy* for solving *algebraic equation systems*.

DAE solution strategy
A *DAE solution strategy* is a *model solution strategy* for solving *differential algebraic equation systems*. Examples are implicit Runge-Kutta, BDF, etc.

Linear algebraic model solution strategy
A *linear algebraic model solution strategy* is a *model solution strategy* for solving *linear algebraic systems.* An example is Gauss elimination.

Model solution strategy
A *model solution strategy* is a (typically numerical) algorithm that can be used to solve *mathematical models*.

Nonlinear algebraic model solution strategy
A *nonlinear algebraic model solution strategy* is a *model solution strategy* for solving *nonlinear algebraic systems*. An example is Newton's method.

ODE solution strategy
An *ODE solution strategy* is a *model solution strategy* for solving *ordinary differential equation systems*. An example is the Euler method.

Partial differential algebraic model solution strategy
A *partial differential algebraic model solution strategy* is a *model solution strategy* for solving *partial differential algebraic systems*. An example is a finite element method.

Solution strategy for explicit ODEs
A *solution strategy for explicit ODEs* is used to solve *ordinary differential equation systems* that are given in an **explicit_formulation**. Examples are explicit Euler, explicit Runge-Kutta, etc.

Solution strategy for implicit ODEs
A *solution strategy for implicit ODEs* is used to solve *ordinary differential equation systems* that are given in an **implicit_formulation**. Examples are implicit Euler, implicit Runge-Kutta, etc.

Type of involved steps
A *type of involved step* denotes whether an *ODE solution strategy* is a **one-step_method** or a **multi-step_method**.

- A **one-step_method** characterizes an *ODE solution strategy* that uses information of one integration step. Examples are various Runge-Kutta methods.
- A **multi-step_method** characterizes an *ODE solution strategy* that uses information of multiple integration steps. Examples are Adams, BDF, etc.

Formal definition: Exhaustive enumeration of the individuals **one-step_method** and **multi-step method**.

Relation Descriptions

applies
A *model solution strategy* may apply some other, specialized *model solution strategy* (e.g., for initialization, solving corrector equation, solution of a subproblem, etc.).

hasTypeOfInvolvedSteps
Indicates the *type of involved steps* of an *ODE solution strategy*.

solves
The relation indicates the type of *mathematical model*, for the solution of which a particular *model solution strategy* is designated.

Attribute Descriptions

handlesDifferentialIndexUpTo
A *DAE solution strategy* can only solve *differential algebraic equation systems* up to a certain differentialIndex. This restriction is specified through the attribute handlesDifferentialIndexUpTo.

9.4 Cost Model

The ontology module *cost_model* establishes some *cost models* for predicting the (investment) *costs* of chemical plants. A *cost model* is a special type of *economic performance model*, which models the *economic performance* of a *chemical process system*.
At present, the module merely holds a number of models for the estimation of the *fixed capital investment* (cf. Sect. 8.7.1.2); in the future, further types of *cost models* are to be added, and the existing ones are to be specified in detail. Fig. 9.13 gives an overview on the *cost models* defined so far. For an explanation of the individual classes, we refer to the concept definitions below.

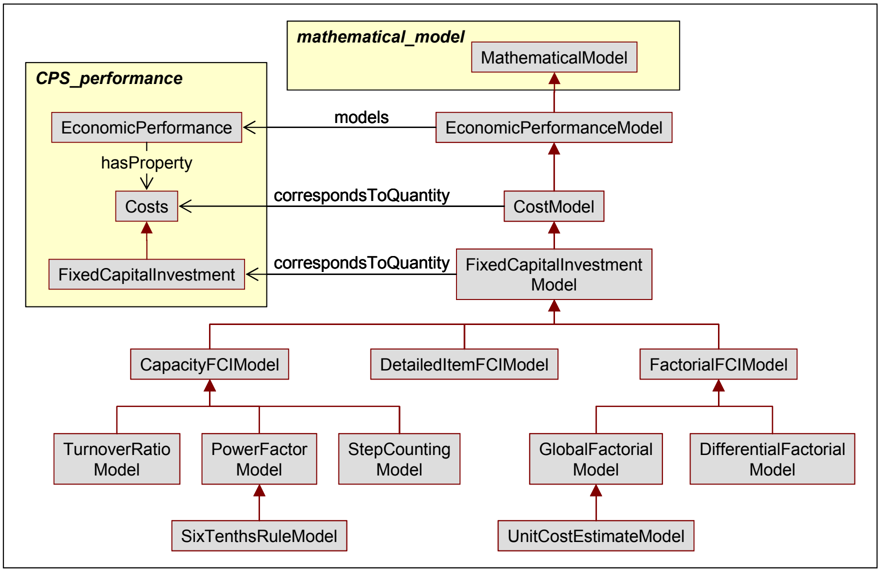

Fig. 9.13: Models for estimating the fixed capital investment

9.4.1 Concept Descriptions

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

Class Descriptions

Capacity FCI model
Capacity FCI models are based on *fixed capital investments* of past design projects that are similar to the current *chemical process system*. Besides, some relating factors (e.g., the turn-over ratio), exponential power ratios, or more complex relations are given.

Cost model
A *cost model* is a *mathematical model* to estimate the investment costs of a *chemical process system*.
Formal definition: A *cost model* is an *economic performance model* that has a *model quantity* which corresponds to the quantity of *costs*.

Detailed-item FCI model
A *detailed-item FCI model* requires careful determination of all individual direct and indirect cost items. For such models, extensive data and large amounts of engineering time are necessary. Therefore, this type of estimate is almost exclusively prepared by contractors bidding on complete and all-inclusive work from finished drawings and specifications.

Differential factorial model
Within *differential factorial models*, different factors are used for estimating the costs of the *fixed capital investment*. Examples are modular estimate models, where individual modules consisting of a group of similar items are considered separately, and their costs are then summarized (Guthrie 1969).

Economic performance model
An *economic performance model* models the *economic performance* of a *chemical process system*.
Formal definition: An *economic performance model* is a *mathematical model* that models some *economic performance*.

Factorial FCI model
Factorial FCI models rely on the fact that the percentages of the different costs within the *fixed capital investment* are similar for different *chemical process systems*. Based on one or several known costs (for example the *equipment costs*), the *fixed capital investment* is estimated using some factors that are derived from cost records, published data, and experience.

Fixed capital investment model
Fixed capital investment models (FCI models) are *mathematical models* that are used to estimate the *fixed capital investment* of a chemical process system.
Formal definition: A *fixed capital investment model* is a *cost model* which has a *model quantity* that correspondsToQuantity of *fixed capital investment*.

Global factorial model
A *global factorial model* estimates the *fixed capital investment* by multiplying the basic equipment cost by some factor. This factor depends, among other things, on the type of chemical process involved, on the required materials of construction, and on the location of the *chemical process system realization*. Examples for global factors are the ones proposed by (Lang 1947). This model can be extended to calculate the *total capital investment*.

Power factor model
The *power factor model* relates the *fixed capital investment* of a new *chemical process system* to the one of similar, previously constructed systems by an exponential power ratio (cf. Peters and Timmerhaus 1991).

Six-tenths rule model
The *six-tenths rule model* is a *power factor model* with x=0.6.

Step counting model
Step counting models are based on the assumption that the *fixed capital investment* can be estimated from the number of *process steps* (depending on the specific approach, *composite process steps* or *unit operations* and *reactions* are used), multiplied with the costs per *process step* and some correcting factors. The costs of the *process steps* are estimated from their capacity and some other factors (Vogt 1996).

Turnover ratio model
The *turnover ratio model* is a fast evaluation method for order-of-magnitude estimates. The turnover ratio is defined as the ratio of gross annual sales to *fixed capital investment*. Values of turnover ratios for different types of chemical processes are for example given by Schembra (1991) and Vogt (1996).

Unit-cost estimate model
Unit-cost estimate models are based on detailed estimates of the main *purchase costs for system realization* (either obtained from quotations or from cost records and published data).

9.5 Process Model

As an extension to *mathematical_model*, the ontology module *process_model* enables the definition of specialized *mathematical models* for the domain of chemical engineering. Such models, which model either *process units* (cf. Sect. 8.1.1) or *materials* (cf. Sect. 7.1) or subsystems of these, are called *process models* (cf. Fig. 9.14). The *modeling principle* based on which a *process model* is developed may also be indicated.

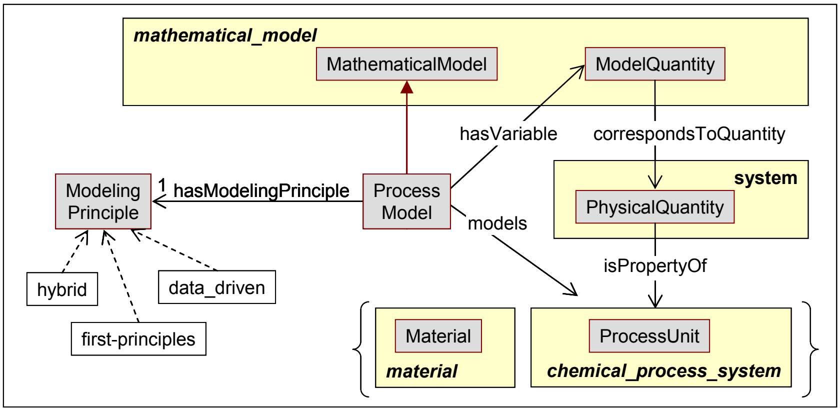

Fig. 9.14: Overview on *process_model*

A *process model* may contain other *process models*, particularly the established *laws* and *property models* (cf. Fig. 9.15). Neither *laws* nor *property models* are self-contained models, but form part of an overall *process model*, where they represent mathematical correlation between designated *model quantities*.

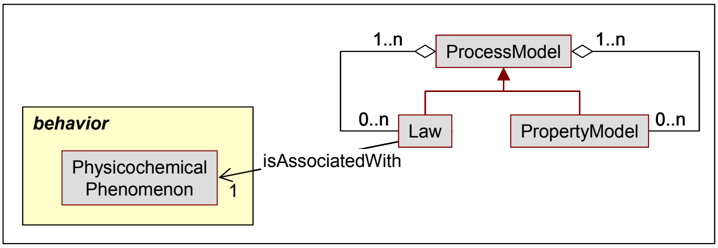

Fig. 9.15: Laws and property models

A *law* constitutes the mathematical representation of a scientific law, such as the law of energy conservation (cf. Sect. 9.6). Each *law* can be associated with a *physicochemical phenomenon* (cf. Sect. 8.6.1.6). The former gives a quantitative, the latter a qualitative description of a certain physical behavior. The correspondence between a *law* and a *physicochemical phenomenon* can be stated via the relation isAssociatedWith, as indicated in Fig. 9.15. Moreover, the *model quantities* of the *law* correspond to the *physical quantities* that are influenced by the *physicochemical phenomenon*, as exemplarily shown in Fig. 9.16.

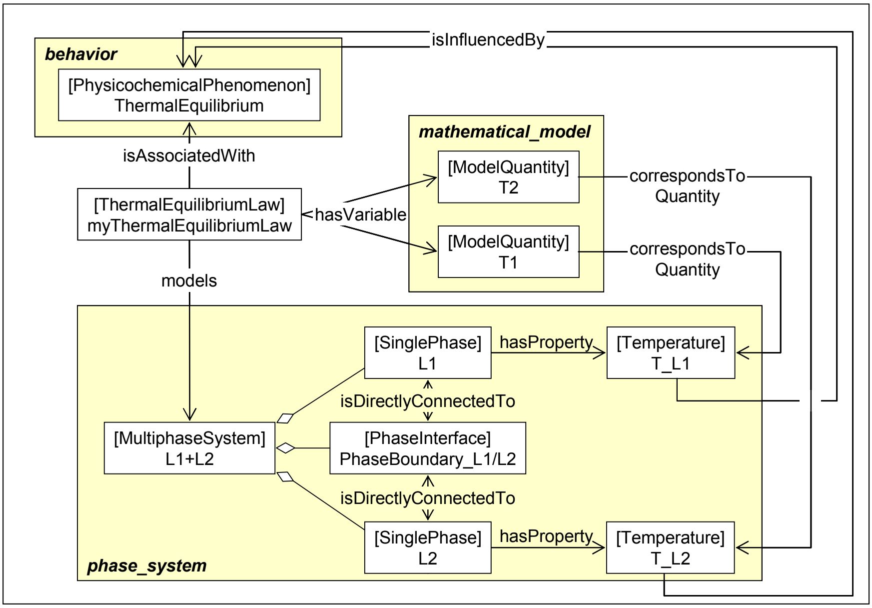

Fig. 9.16: Exemplary law modeling thermal equilibrium

A *property model* represents a mathematical correlation for the computation of one designated *model quantity*, which corresponds to one specific *physical quantity*. An example is given in Fig. 9.17: An *activity coefficient model* constitutes a correlation for the computation of *activity coefficients*. Consequently, an *activity coefficient model* comprises, among others, a *model quantity* which corresponds to an *activity coefficient*.

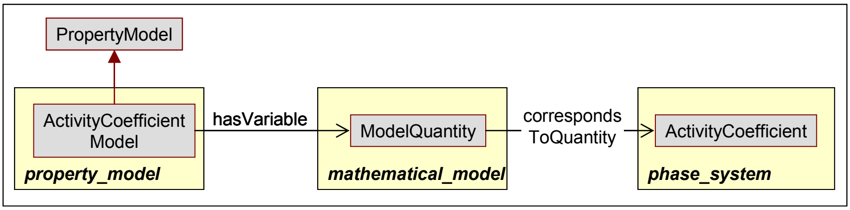

Fig. 9.17: Exemplary property model

9.5.1 Concept Descriptions

Individual concepts of the module *process_model* are defined below. For a description of the instances of *modeling principle*, we refer to Morbach et al. (2008j).

Class Descriptions

Law
A *law* constitutes the mathematical representation of a scientific law. It usually forms part of an overall *process model*.

Modeling principle
A *modeling principle* represents the principle on which the development of *process model* is based.

- Following the **data_driven** *modeling principle*, a *process model* is derived from the *values* of the *properties* of a *modeled object*. Examples of this type of models are neural network models.
- Following the **first-principles** *modeling principle*, the *process model* is based on established physical laws and mechanisms.
- A **hybrid** *modeling principle* applies both the **first-principles** and the **data_driven** approach.

<u>Formal definition</u>: *Modeling principle* is defined by an exhaustive enumeration of the individuals **data_driven**, **first-principles**, and **hybrid**.

Process model
A *process model* is a *mathematical model* that models a *process unit* or *material* (or *subsystems* of these).

Property model
A *property model* forms part of an overall *process model*. It represents a mathematical correlation for the computation of a designated *model quantity*, which corresponds to a specific *physical quantity*. Examples are vapor pressure correlations or activity coefficient models.

Relation Descriptions

hasModelingPrinciple
Indicates the *modeling principle* on which a *process model* is based.

isAssociatedWith
The relation denotes a correspondence between a *law* and a *physicochemical phenomenon*. The former gives a quantitative, the latter a qualitative description of a certain physical behavior.

9.6 Laws

The ontology module *laws*, located on the Application-Oriented Layer of OntoCAPE, introduces a hierarchical collection of *laws* that are frequently used in process modeling. The law hierarchy shown was originally presented by Marquardt (1995). A selection of the taxonomy related to physicochemical laws is given in Fig. 9.18 - Fig. 9.22. The high-level concepts include *balance laws*, *constitutive laws*, and *constraints* as shown in Fig. 9.18.

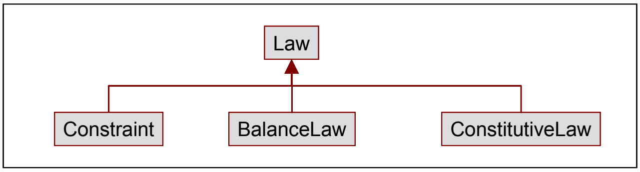

Fig. 9.18: High-level classification of *laws*

Balance laws generally represent the change of an extensive quantity in *process models*. This typically includes balances for total mass and mass of species in a mixture (*mass balance law*), for momentum (*momentum balance law*), for total or any other kind of energy (*energy balance law*), and for the particle number in case of a particulate system (*population balance laws*), as depicted in Fig. 9.19.

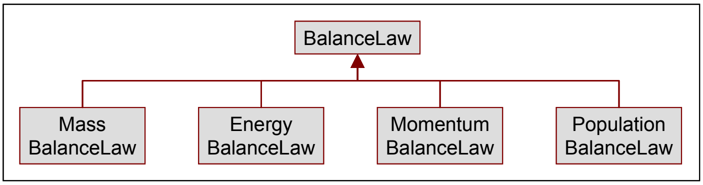

Fig. 9.19: Specialization of *balance laws*

However, balance equations do not suffice to describe the behavior related to a *process model*. Thus, *constitutive laws* have to be added in order to determine the *process model* completely. Three types of *constitutive laws* may be distinguished (compare Sect. 8.6.1.7 and Fig. 9.20), including *generalized flux laws*, *phenomenological coefficient law*, and *thermodynamic state function law*. *Generalized flux laws* describe the contribution to any kind of *balance law*. These laws are typically composed of a phenomenological coefficient and a driving force determined by some thermodynamic state function which are modeled by *phenomenological coefficient laws* and *thermodynamic state function laws*.

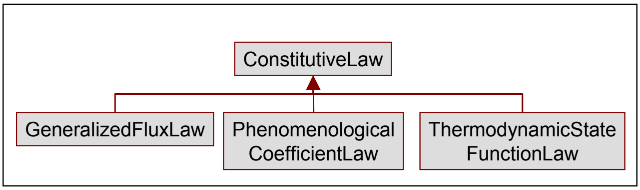

Fig. 9.20: Specialization of *constitutive law*

In Fig. 9.21, the specializations of *generalized flux law* are presented, including some further specializations associated to transport and exchange phenomena. These specific *laws* have to be considered before a concrete *process model* can be generated.

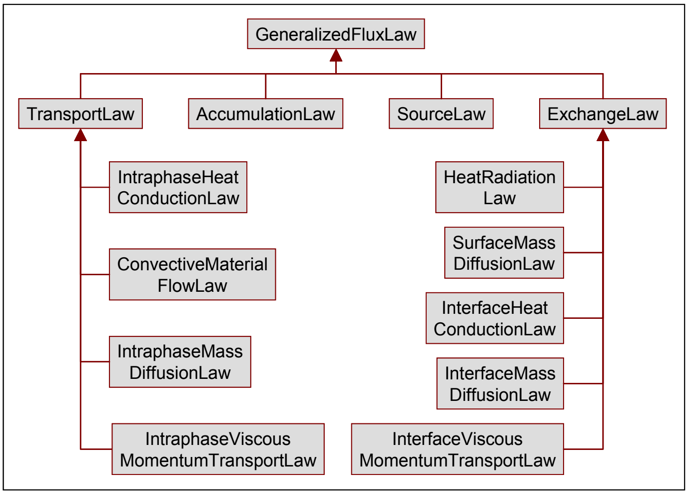

Fig. 9.21: Specialization of *generalized flux laws*

Finally, *constraints* describe all kinds of (algebraic) relations between *process quantities* which – literally or by assumption – have to hold at any time. Typical examples are *volume constraints* or *equilibrium constraints*, which specialize the class *constraint* in Fig. 9.18.

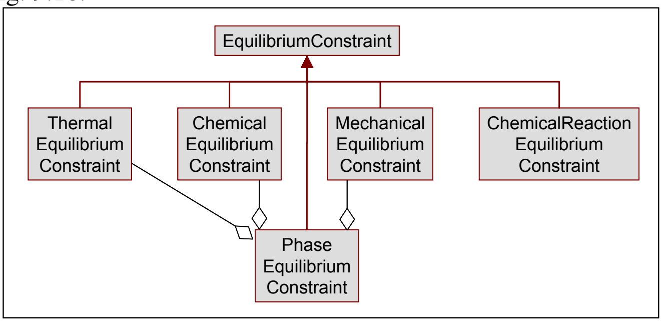

Fig. 9.22: Specialization of *equilibrium constraints*

In Fig. 9.22, *equilibrium constraints* are specialized into *thermal equilibrium*, *chemical equilibrium*, and *mechanical equilibrium* on the one hand, which refer to equal temperature, pressure, or chemical potential in adjacent phases. *Phase equilibrium* and *chemical reaction equilibrium* are considered on the other hand. It shows that *phase* *equilibrium* is just an aggregation of thermal, chemical, and mechanical equilibrium. *Chemical reaction equilibrium* refers to a network of chemical reactions residing in a *single phase*, where all forward reaction rates equal the backward reaction rates. The constraint *phase equilibrium* can also be formulated in various alternative ways which are fully equivalent[98].

Currently, only the hierarchy of *laws* and the associated *physicochemical phenomena* (cf. Sect. 8.6.1.6) are modeled. In future extensions of this ontology module, one may add further definitions and constraints in order to specify a *law*'s *model quantities* and their corresponding *physical quantities*.

For an exhaustive description of all concepts used in the module *laws*, we refer to Morbach et al. (2008j).

9.7 Property Models

The ontology module *property_models*, which is located on the Application-Oriented Layer of OntoCAPE, provides a hierarchically ordered collection of frequently used *property models*. As indicated in Fig. 9.23, a *property model* might be one of the following:

– A *chemical kinetics model*, which specifies how to calculate the rate coefficient of a homogenous or heterogeneous reaction.
– A *phase interface transport property model*, which provides a correlation for computing certain *phase interface transport properties*.
– A *thermodynamic property model*, which indicates the correlation between certain *intensive thermodynamics state variables* (cf. Sect. 7.3.3) and *intra-phase transport properties*.

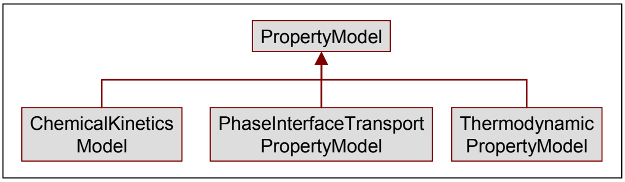

Fig. 9.23: High-level classification of *property models*

The classification of these specialized *property models* is given in Fig. 9.24 - Fig. 9.27:

[98] For example, a number of alternative formulations exist for chemical equilibrium, such as the equality of chemical potentials between two phases and the equality of fugacities between two phases. The equality of fugacities can further be written in different forms depending on what property models are to be used in conjunction with the law for the chemical equilibrium.

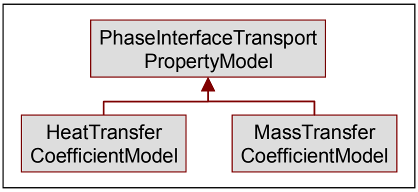

Fig. 9.24: Some *phase interface transport property models*

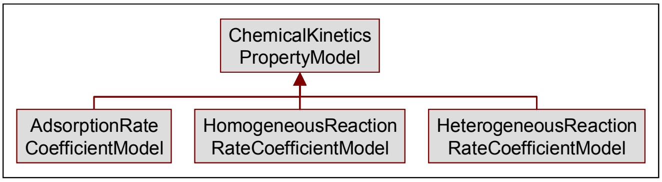

Fig. 9.25: Some *chemical kinetics models*

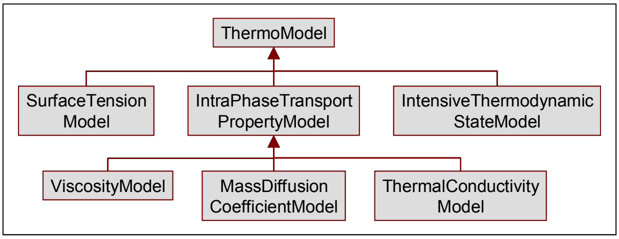

Fig. 9.26: Some *thermodynamic property models*

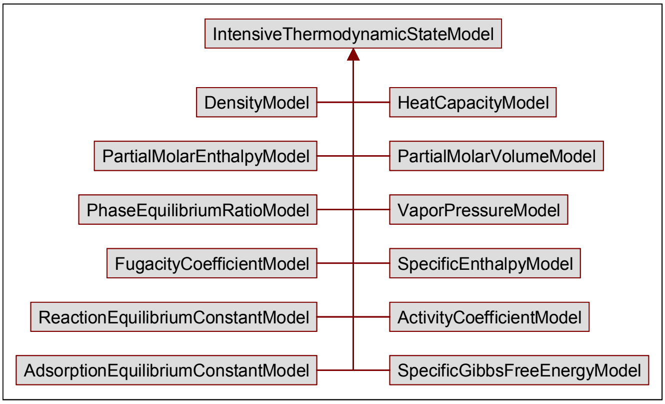

Fig. 9.27: Some *intensive thermodynamic state models*

Exemplarily, the definition of the class *density model* is shown in Fig. 9.28: A *density model* has some *model quantities*, one of which corresponds to a *physical quantity* of type *density*. The other *property models* are defined analogously.

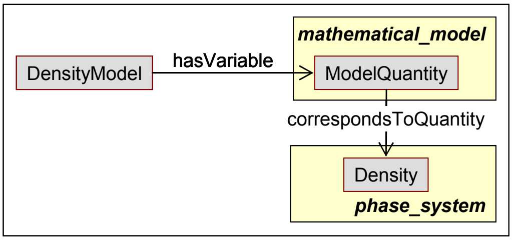

Fig. 9.28: Definition of the class *density model*

For an exhaustive description of all concepts used in the module *property_models*, we refer to Morbach et al. (2008j).

9.8 Process Unit Models

The ontology module *process_unit_models*, located on the application-oriented level of OntoCAPE, provides a collection of *mathematical models* that model the behavioral aspect of *process units*.

Please note that this module is introduced not for the purpose of providing a full account on this topic, but rather for suggesting a principle for defining various types of *process unit models* and illustrating the principle by means of only a few examples.

These exemplary *property unit models* are classified according to the modeled *process units* (Wiesner et al. 2008a): a *chemical reactor model* models a *chemical reactor behavior*, a *flash unit model* models a *flash unit behavior*, etc. (cf. Fig. 9.29).

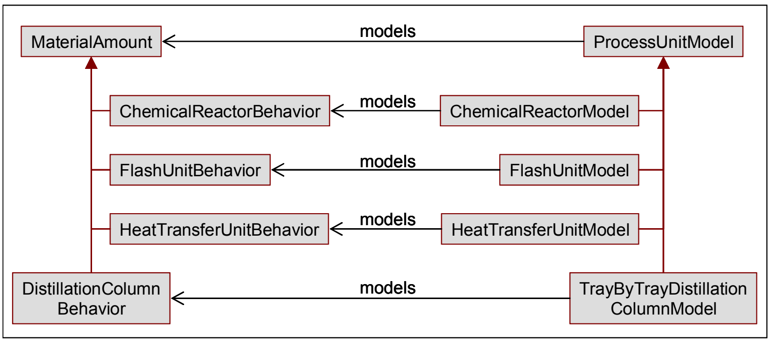

Fig. 9.29: High-level classification of *process unit models*

Beyond this high-level classification, the ontology module comprises some special types of *process unit models*. Fig. 9.30 exemplarily shows the definition of a *CSTR model*: A *CSTR model* is a *chemical reactor model* that models a *chemical reactor behavior* with the *physicochemical phenomenon* of phenomenon **ideally_mixed**. Furthermore, the *CSTR model* is a **first-principles** model and incorporates the following *laws*: *energy conservation law*, *mass conservation law*, and *reaction kinetics law*.

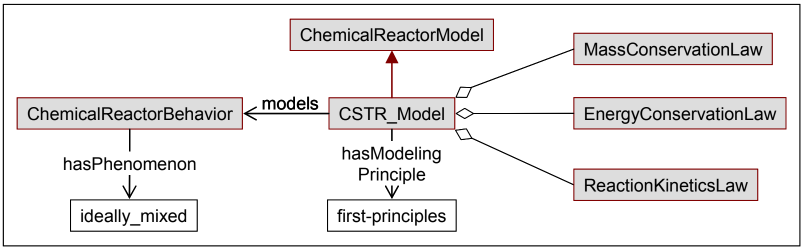

Fig. 9.30: Definition of the class *CSTR model*

Currently, the ontology module provides only a few of such specialized *process unit models*. In the future, it is to be extended to offer a substantial library of *process unit models*. As for now, the module merely provides the framework for establishing such a library.

9.9 References

Gear CW, Petzold L (1984) ODE methods for the solution of differential/algebraic systems. *Trans. Society Computer Simulation* **1**:27–31.

Guthrie K (1969) Data and techniques for preliminary capital cost estimation. *Chem. Eng. (New York)* **24** (3):114-142.

Hairer E, Wanner G (1996) *Solving Ordinary Differential Equations II – Stiff and Differential-Algebraic Problems*, Springer, Berlin.

Ion P, Miner R, eds. (1999) *Mathematical Markup Language (MathML) 1.01 Specification*. W3C Recommendation, revision of 7 July 1999. Online available at http://www.w3.org/TR/REC-MathML/. Accessed April 2007.

Lang HJ (1947) Engineering approach to preliminary cost estimates. *Chem. Eng. (New York)*:130-133.

Lloyd CM, Halstead MDB, Nielsen PF (2004) CellML: its future, present and past. Progress in Biophysics and Molecular Biology **85** (2-3):433-450.

Martinson WS, Barton PI (2000) A differentiation index for partial differential-algebraic equations. SIAM *J. Sci. Comput.* **21**:2295–2315.

Morbach J, Yang A, Marquardt W (2008j) *OntoCAPE 2.0 – Mathematical Models*. Technical Report (LPT-2008-28), Lehrstuhl für Prozesstechnik, RWTH Aachen University.

Peters MS, Timmerhaus KD (1991) *Plant Design and Economics for Chemical Engineers*, McGraw-Hill, New York.

Schembra M (1991) *Daten und Methoden zur Vorkalkulation des Anlagekapitalbedarfs von Chemieanlagen*. PhD thesis, Technische Universität Berlin.

Vogt M (1996) *Neuere Methoden der Investitionsrechnung in der Chemischen Industrie*. Diploma thesis, Technische Universität Berlin.

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-2008-29.