3

Representation of Physical Knowledge

This Chapter presents the basic ideas behind the PML language design. The object oriented modeling methodology will be analyzed in order to set the basis that has motivated this new language proposal. The object oriented methodology should support the structured development of models according to the system topology. The main goal of this approach to model construction is to reduce the effort associated to the model development and maintenance. Representations of systems as circuit diagrams or flowsheets are familiar to most engineers since they preserve the topological structure of a system. To support a modeling methodology where models can be structured according to the system structure, the language should provide with capability to reuse predefined modeling components (see Definition 1.3) and the consequent manipulation procedures required to achieve an adequate computational formalism or simulation model (see Definition 1.12). This modeling approach is based on libraries containing modular structures where the physical knowledge required to describe the system behaviour is represented. Fully reuse of such predefined modular structures is supported if it is possible to declare them without considering their reusing context. This chapter discusses which are the requirements for a modeling framework where physical knowledge could be represented in a modular way.

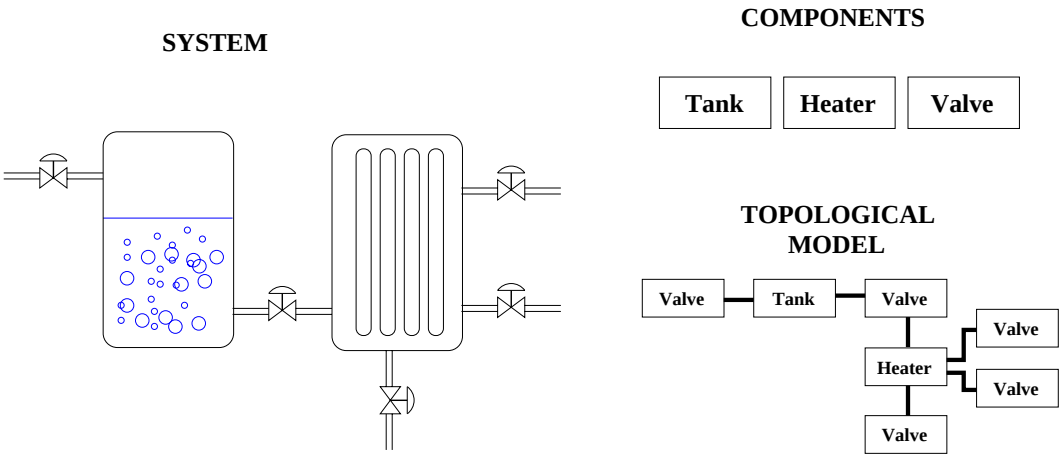


Figure 3.1. Topological description of a physical process

3.1. Support of structured modeling

A model is an abstraction of the world, i.e., a representation of our knowledge about the world within certain experimental framework. The basic objective of this chapter is to present a modeling framework to formalize the available knowledge about physical systems.

It is quite obvious that a structured approach where models can be specified at the topological level may reduce the modeling cost. We call this approach topological modeling, since the structured model should preserve the analogy with the system physical structure. Figure 3.1 illustrates the meaning of supporting the model construction according to the system topology. Modeling libraries with predefined and validated models should be provided to the user. These models have to represent a system component, e.g. a heat exchanger process unit, or a part of it, e.g. the cold circuit of a heat exchanger. Such model components can be reused and linked together to define new models. Unfortunately, a model definition according to system topology does not always preserve the computational structure suitable for simulation (Cellier and Elmqvist 1993, Ramos *et al.* 1998*a*).

Causal modeling tools provide with the capability of structuring models according to the computational structure for simulation. That is the reason why they hardly preserve the system topology and why their model building blocks can not be reused in an arbitrary context. The

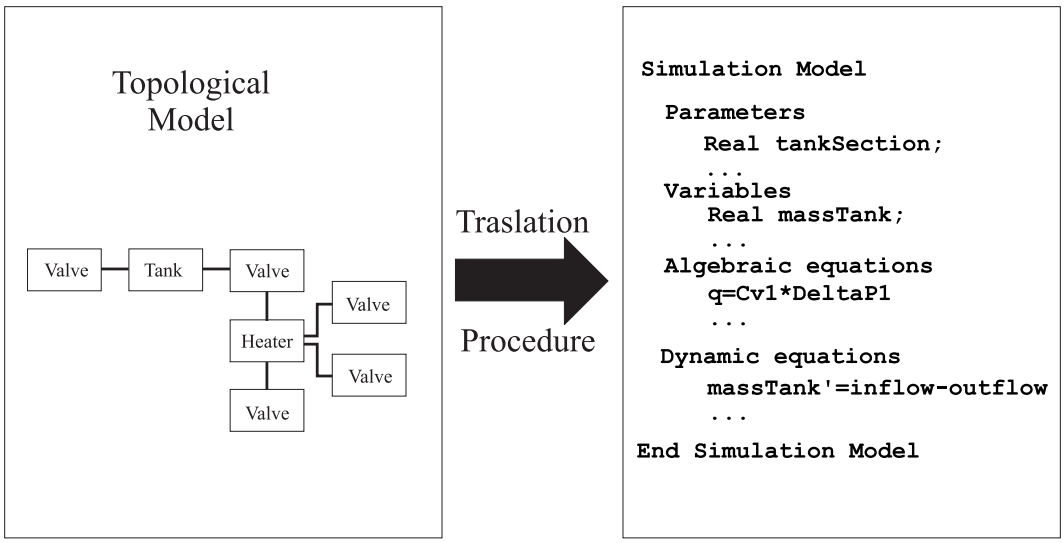
object oriented methodology tries to overcome these limitations providing with the capability of structuring models according to system topology together with the capability of translating those models into the mathematical formulation suitable for simulation. However, the system modeling domain introduces several constraints that make difficult to support the object oriented approach. These obstacles are a consequence the complicated *communication protocol* that has to be defined between the modeling classes. At some point in the translation from a structured model into a flat simulation model, the physical behaviour represented in each modeling class has to be manipulated in order to find the proper aggregated behaviour.

Most of the present objected oriented modeling tools are based on an equation representation of the physical behaviour. The unique modeling class defined by these tools is basically the *model class* as the representation of a system or a part of it. The model encapsulates a set of equations declaring the modeled physical behaviour. The modeling classes (models) communicate through a set of variables known as terminal variables. Such model interface defines an information stream between the equations encapsulated by the model and its environment. When models are connected through this interface, the terminal variables are related and interpreted in order to set the coupled behaviour. The following sections will try to show the limitations of this equation based approach. The analysis of the constraints shown up by the equation based object-oriented languages will serve us to state the basis for the new physical behaviour representation formalism. The PML language has been defined to implement this formalism under the object oriented paradigm.

3.2. Modular representation of the physical knowledge

In Section 2.3.3 the analogy between the object-oriented modeling and the object-oriented programming methodologies was presented. However, there are also significant differences between an object-oriented program and an object-oriented structured model operation.

A programming class is a modular structure representing the functionality and the attributes of some type of objects. An application is organized around a set of classes whose instances (objects) cooperate to perform some task. This cooperation consists basically in a communica-



(a) Modular representation of system behaviour

(b) DAE formulation of system behaviour

Figure 3.2. Translation from the topological model of the physical process at figure 3.1 into its simulation model.

tion process between the objects. The interaction between the objects is known as the *message passing* mechanism: an object A can send a message to an object B expecting a value to be returned immediately. The class interface declares the communication capabilities with other classes.

The object oriented modeling is also based on a classification method. The modeling classes should be modular structures representing some physical behaviour. However, modeling class instances (modeling objects from now on) and conventional programming objects interact in a very different way. A simulation model operates on a *time base*, i.e., a model represents the system dynamic evolution over time. Therefore, the modeling tool should be able to translate the modular system representation into a state-space , or a more generic DAE, formulation suitable for simulation engines (see Figure 3.2). This computational model must represent the overall system behaviour which depends on the *internal* behaviour encapsulated at each modeling object together with the physical interactions derived from their coupling. Hence, the modeling object

must define an interface able to represent the coupled behaviour. The modeling tool should be able to manipulate the object interface together with the internal physical representation in order to derive the simulation model, i.e., a procedural representation with the adequate computational structure.

The interaction between modeling objects involves a protocol much more complex than the message passing between programming objects. The translation from model based on modular structures into the simulation model introduces several requirements to both the interaction mechanisms and the modular physical behaviour representation. Nevertheless, many of the object oriented properties lean on the modeling object interface since it determines the way in which the physical behaviour can be encapsulated and retrieved to satisfy the experimental framework requirements.

3.2.1. Interaction between modeling objects.

Knowledge encapsulation, achieved by means of modular structures with abstract interfaces, is an essential property to support reusability. A main benefit of encapsulation is that any class can be readily, and independently, tested by putting the proper information in its interface. When new performance is required from a module, new knowledge can be supplied to attend new demand with minimum side effects to the rest of interacting objects.

In physical system modeling the capability to encapsulate the physical behaviour depends on the modeling class interface ability to represent the physical interaction relationships. Most of the existing modeling languages define structures to implement the concept of a *stream*, which is used to represent the connection between system components. The interaction between system components may be classified into the main three types of streams: material, energy and information.

The most complex streams to model are those which represent material flows. Consider for instance the matter flow between the process units of the topological model shown at the Figure 3.2 (a). Such streams must record sufficient information for all relevant properties that are carried between system components. Energy streams are used to represent energy flows in

the absence of corresponding material flows (for example, transfer of heat energy by conduction or transfer of mechanical energy through a shaft). Material or energy stream convey a direction flow, so that they have a source and a destination. This direction should not be predefined since it may change even at simulation time. Therefore, a successful modeling environment must provide support for computing flow directions without assuming a predefined nominal value.

The third type of stream is information and it is used to describe many other information flows which are not directly related to material or energy flows. This information stream contains only one data item, for example the mass flow rate of a sensor in a pipe or the control law computed by a controller. Information streams usually have a predefined in or out direction.

While information streams simply represent shared data between connected components, material and energy flows establish the physical interactions among the different system components. These interactions depend on the system topology, and its knowledge is of crucial importance for a modeling environment in order to be able to achieve the adequate simulation model. For instance, look at the system shown at Figure 3.1. The internal energy of the matter flowing from the tank into the heater defines a thermal energy interaction between them. The tank and the heater are not directly coupled, since there is a valve component between them. Assume this valve does not affect the matter internal energy. A difficult problem to solve when building the tank, heater and valve component models is to decide which of these component models should describe the thermal energy interaction.

This question appears when a stream through different system component implies physical interactions between components which are not one-to-one coupled. The stream properties propagation are related with its transport through system components, which often do not affect such properties (e.g. material component concentrations are not affected in a valve). A model decomposition preserving the system component connection structure will be possible depending on the capability to represent the physical relationships among the system component. The following discussion studies in depth how the modeling classes interfaces should be defined in order to represent the physical interaction between system components properly.

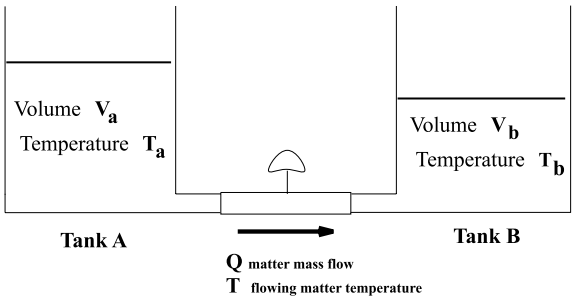


Figure 3.3. System of two tanks.

```
model TwoTanks
  Tank TankA,TankB;
  PipeLine pipe;
  MatterSource source;
  MatterSink sink;
equation
  connect(source.C,TankA.C1);
  connect(TankA.C2,pipe.C1);
  connect(pipe.C2,TankB.C2);
  connect(TankB.C1,sink.C);
end TwoTanks;
```

Listing 3.1. Modelica model of a Tank.

The modeling class interface.

The different modeling environments provide stream representation in a variety of reports. To illustrate the problem of representing physical interactions let's consider the system in Figure 3.3. In this system there is a material flowing between the two tanks through a pipe line.

In present equation-based object oriented modeling languages, as Modelica (Elmquist and Al. 1999) or EcosimPro (Cobas and Al. 1999) , models are defined as classes where physical behaviour is encapsulated as mathematical equations. In order to give a brief overview of these language types, a Modelica model of the two tank system is given in Listing 3.1. The composite (structured) model `Twotanks` specifies the system topology in terms of components (submodels) and the connections of their interfaces. The models of system components are given in Lsting 3.2. These models represent the internal component behaviour (a medium mass balance in the tank model and a mass transport rate in the pipe line model). Since in this type of languages behaviour is represented by means of equations, the model interface must declare which of the variables involved by the equations are related with the model environment. Hence, the component physical interaction represented at model interface is constrained to be a tuple of the variables shared by the encapsulated equations and the environment.

```
connector ProcesTerminal
  Pressure P;
  flow MassFlowRate W;
end ProcessTerminal; model PipeLine
  ProcesTerminal C1,C2;
  parameters  PipeCons Cv, Density rho;
equation
  P = C1.P - C2.P;
  C1.W + C2.W = 0;
  W = C1.W;
  P = rho/sqr(Cv)*abs(W)*W;
end PipeLine;
```

```
model Tank
  ProcesTerminal C1,C2;
  parameter Area A, Density rho,
     Pressure Patm, Gravity g;
equation
  C1.P = Patm;
  C2.P = sqrt(Patm+g*H);
  der(mass) = C1.W + C2.W;
  H = mass/rho/A;
end Tank;
```

Listing 3.2. Basic component models.

The model class interface at Listing 3.2 is defined by the connector `ProcessTerminal` by means of the pressure and mass flow rate variables. These variables are used to generate the link equations which relate the behaviour equations represented in each model. In this sense, they establish the information flow between models. For instance, when `TankA` and `pipe` models are connected, the pressure connector variables equals and mass flow rate connector variables sum to zero in the connection point. Therefore, when connecting the `ProcessTerminal` connectors of two components, the following coupling equations will be established:

$$P_A = P_B \tag{3.1}$$

$$W_A + W_B = 0 \tag{3.2}$$

With these two equations we distinguish between the representation of the physical properties having a direction from a source to a destination (e.g., the mass flow rate) and the physical properties which equals in the connection point. The two types of properties are represented by means of the *through* and *across* variables presented in Section 2.2.2 (see also (Cellier 1991)).

The coupling equations are required to establish the relationships between the equations of interconnected models. Hence, they are extremely dependent on the equations represented in

```
connector ProcesTerminal
  Pressure P;
  flow MassFlowRate W;
  Temperature T;
  Temperature Tq;
end ProcessTerminal;
model PipeLine
  ProcesTerminal C1,C2;
  parameters  PipeCons Cv, Density rho;
equation
  P = C1.P - C2.P;
  T1.W + T2.W = 0;
  W = T1.W;
  P = rho/sqr(Cv)*abs(Q)*Q;
% Medium temperature
  C1.Tq = C2.Tq;
  C1.Tq = Tq;
  Tq = IF W > 0 THEN T1.T ELSE T2.T;
end PipeLine;
```

Listing 3.3. Modified process connector and pipeline model.

the model which will become a component in a larger model. A main consequence is that, when the equations in a model are modified to include new dynamics, there will be side effects upon the rest of the component models interacting with it.

EXAMPLE 3.1 – *Modelling the matter temperature*

Consider that the tank model at Listing 3.2 has to be modified in order to include the medium temperature dynamics described by the heat balance shown in Equation 3.3

$$\frac{d}{dt}(\rho V c T) = c T_q W \tag{3.3}$$

where T_q is the temperature of the flowing medium and W is the mass flow rate. The temperature in a tank depends on the input/output medium temperature. This new information has to be added into the component model interface: the connector `processTerminal`.

If the matter flows outwards the tank, the medium temperature is determined in the tank and its value is computed by the heat balance at Equation 3.3. Therefore, the tank model should report the medium temperature (internal state) through the interface when the stream flows out. A variable, say `T`, is included in the connector and its value is assigned with the internal temperature state variable. If the stream flows inwards the tank, the medium temperature is determined in another system component. Since the value of the variable `T` has been already

assigned, a new variable, say `Tq`, is required to report the medium temperature when the stream flows into the tank. The new connector `ProcessTerminal` is shown at Listing 3.3.

Since the medium temperature is a property that has to "travel" with the medium when it is transported among the different process units, new equations will have to be defined to represent its propagation. In the system at Figure 3.3 the physical interaction appears between $Tank_A$ and $Tank_B$. Equation 3.4 can be used to describe how the temperature propagates depending on flow sense.

$$T_q = \text{if} \quad W > 0 \quad \text{then} \quad T_{Left} \quad \text{else} \quad T_{Right} \tag{3.4}$$

where T_{Left} and T_{Right} represent the temperature of the matter flowing into the pipeline (obviously it depends on the flow sense).

The thermal interaction between the two tanks derived from the matter transport through the pipeline is not modular to one single component, so it can not be established without analyzing the overall system topology. Furthermore, it is not a trivial question to decide in which component model must be represented this type of physical interactions since, as Equation 3.4 shows, the three system components are involved. Equation 3.4 is assuming that the components connected to the *left* and *right* determine the temperature of the flowing matter depending on the flow direction, and there is no reason for this being true.

In a system consisting of interconnected components there are physical interactions between one-to-one coupled components which can be predefined by means of the connector variables. For instance hydrostatic pressure equals in the connection between the tank and a valve. However, more powerful interface mechanism will be required to represent the physical interactions derived from aggregate components connections which are not one-to-one coupled components.

In difference with equal or sum to zero equations derived from the variables placed in the model connectors, it is not very clear where conditional expressions, such as Equation 3.4, should be placed without breaking the encapsulation feature and, consequently, with the modularity principle.

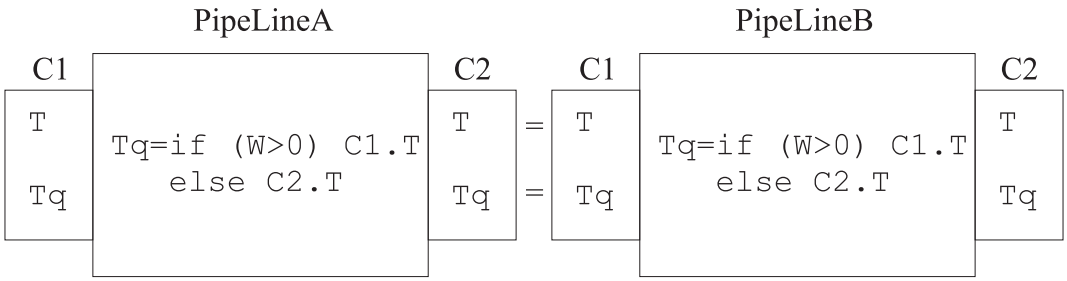


Figure 3.4. Connection topology of two `PipeLine` models illustrating the flowing matter temperature propagation.

For instance, the solution proposed in Listing 3.3 to model the medium property transport is assuming that the elements connected to the pipeline are determining these properties. The conditional expression

```
Tq = IF W > 0 THEN C1.T ELSE C2.T
```

assigns the medium temperature `Tq`, provided that `T1.T` and `T2.T` are determined by the connected models. Obviously, this model would not work if the neighboring models do not determine the temperature (Ramos 1994). For instance, let us consider that two `PipeLine` transport components are connected in series as Figure 3.4 shows. The terminals `T1` of `PipelineA` and `T2` of `PipelineB` are connected to temperature sources, i.e., components declaring the medium temperature dynamics as it has been described above. Assume $Q > 0$, then T_q equals to T at the `T1` terminal in both components. The variable `T1.T` at `PipeLineA` is properly assigned from the temperature source whereas the same variable `T1.T` at `PipeLineB` can not be assigned since the connected component is not a temperature source.

As this simple example shows, the assumption on the behaviour represented on other models breaks with the encapsulation principle and leads to incorrect models when the assumption is not fulfilled. ❑

As the previous example has illustrated, including more equations to describe the energy in the tank model implies that other library models must be also modified to describe the energy

interaction between the tanks. This side effect is extensible to any other matter property. Mathematical expressions such as Equation 3.4 break with the encapsulation principle and lead to serious constraints in model reuse. Note that the pipeline model that was defined in Listing 3.2 can not be reused when the tank model is modified to describe the temperature dynamics.

This problem is the consequence of a poor interaction description mechanism. A model interface defined by means of variables represents just the information exchange between the equation of two coupled models. Moreover, it can be considered that terminal variables also break the analogy with the physical system. System components exchange matter or energy. The terminal variables are only able to represent partially the physical relationships derived from that exchange.

Such type of interface is far from an efficient mechanism able to represent the physical interactions which appear due to the coupling of several system components. The aggregated physical behaviour, i.e. behaviour which is not modular to one single system component, introduces several requirements to the interface description mechanism of the modeling classes in order to:

i) minimize the side effects of modifying a component model,

ii) allow the automatic derivation of the physical behaviour not modular to one single component from the interface description mechanism, and

iii) preserve the analogy with the system physical connections.

In the system shown at Figure 3.3, the modification of the tank model to represent the temperature dynamics had side effects over the remaining models, transferring the problem of describing the thermal interaction between both tanks to the pipeline model. It can be argued that the side effects of adding new behaviour and describing the interactions of a component model may be avoided if the model anticipates the representation of all the possible behaviour from the very beginning. This solution leads to very complicate models which could be useless for many applications where it is not necessary to have such deep detailed behaviour representations.

Nevertheless, not even with models anticipating all the behaviour, the problem of representing the physical phenomena not modular to one single component can be solved by means of connectors defined in terms of variables (Ramos 1994). Assuming that Equation 3.4 is included in the pipeline model, the modified pipeline model will only be valid if it is connected to models that, as the tank models, set the terminal variable `T` value.

Modeling the stream properties with variables leads to complicated connectors since, in general, physical interaction modeling can not be simplified to a pair of connection equation classes (*equal* and *sum to zero* equations). A third type of *conditional* equation is required to model the interactions appearing from the whole system component coupling. This is the case when a stream property is modeled (Ramos 1994) but also in certain model connection topologies which lead to modeling constraint (Ramos *et al.* 1995) which will be discussed in more detail in the Chapter 5.

Variable connectors fall short in modeling the interactions derived from a material transport since they depend on the overall topology of the system and not only on the one-to-one connections between component models. Modularity needs of abstract interfaces able to describe the physical interactions and not just the information flow represented by the terminal variables of one-to-one coupled models.

The coupling mechanism in modular systems should provide a level of *delayed binding* (Zeigler 1990): a system model places in its interface the interaction information, the actual destination of this information can not be determined until the model becomes a component in a larger model and the coupling scheme is specified. Otherwise model reusability can not be assured. The modeling class interfaces in PML will provide the level of delayed binding required to assure the modeling class reusability.

3.2.2. Internal behaviour representation.

As important as the class interface as the means to achieve the physical behaviour encapsulation is the capability to formulate the encapsulated behaviour according to the modeling class reusing context. According to the model construction criterion, a second issue expected from

encapsulation is to make the modeling class reuse independent from the internally represented physical behaviour. The interface is the mechanism used to describe the physical interactions without predefining the model reusing context. Hence, the modeling tool should be able to adapt the physical behaviour into the reusing context.

In the equation based languages this representation is done by means of mathematical expressions. This mathematical modeling uses non-causal (behavioural) equations representing physical knowledge (conservation laws, constitutive equations, etc). Consider the tank model shown in Listing 3.2. The internal behaviour representation consists of equations describing the phenomena (medium storage), medium properties (density) and unit geometrical aspects (tank area). This type of physical behaviour representation is still much more concerned with the procedural knowledge than with the physical knowledge. For instance, the behaviour of a tank system where a medium is accumulated can be described in terms of expressions as Equation 3.5:

$$\frac{\mathrm{d}m}{\mathrm{d}t} = m_i - m_o \tag{3.5}$$

where m is the matter mass and m_i and m_o are the mass flow rates.

The tank model shown at Listing 3.2 uses this equation to represent a tank system where a single medium is stored. Equations describe how a phenomenon can be computed, but they are not able to describe the phenomenon itself. There are changes introduced by the model reusing context that will require different mathematical formulations of the behaviour even when the same phenomena are occurring. For instance, if the single medium is replaced by a composite medium in some reusing context, new expressions as Equation 3.5 could be required for each medium component in order to describe the component concentrations. In this case, the tank model declared at Listing 3.2 can not be reused, despite the same system with the same phenomena is being reused.

There are very efficient solutions when the reusing context of a structured model involves the exchange of a submodel. For example, when a simple resistor model is substituted by a

temperature dependent resistor model in an electrical circuit. This feature is supported by Modelica (Elmqvist et Al. 2000) with the model parameterization capability.

However, when the reusing context requires different mathematical formulations to describe the same phenomenon, the only solution implies the change of the equations. Variations of the mathematical behaviour representation may also occur when we want to adapt a predefined model according to different adequacy levels. In this sense, equations are a *static* formulation of behaviour (see Definition 1.7) since, once the equation is postulated, all the physical knowledge and the physical reusing context becomes determined (Ramos *et al.* 1998*b*, Piera *et al.* 1996).

Despite the fact that solutions as medium separation from equipment and proposals for the primitive behaviour decomposition (Nilsson 1993) have been thoroughly investigated in object oriented modeling, the equation based languages do not, nowadays, offer to the model user a clear feedback to decide how to reuse a model when the reusing context involves modifications on the set of equations used to formulate the physical behaviour.

The mathematical formulation of the laws required to compute physical behaviour is context dependent, even when the phenomena remain unaltered. Hence, every aspect related with the context where the phenomenon occurs has to be known before the law can be properly formulated. Nevertheless, the phenomenon is context independent. A tank accumulates matter independently of the stored material and independently on the mathematical formulation of the matter balance law. This knowledge can not be retrieved from its mathematical formulation (static formalism). However, this knowledge can be easily coded by means of a modeling language designed to represent explicitly the occurring phenomena. The behaviour mathematical formulation can be dynamically obtained once the model reusing context is determined (see Definition 1.8 for dynamic modeling formalism).

Assume that the modeling language can explicitly represent the following physical knowledge: *"model tank stores matter"*, *" the matter storage phenomenon can be described by a matter balance law"* and *"the mass balance law can be computed with Equation 3.5"*. This model representation collects all the concepts about the physical behaviour: the involved entity (matter),

the phenomenon (matter storage), the law which describes it (matter conservation principle) and the computable property of the matter (mass of matter). This knowledge can be used to obtain the following simulation model for a tank where water is stored:

```
model Tank
  Real waterInflow,waterOutflow,waterMass
equation
  der(waterMass) = waterInflow - waterOutflow;
end Tank;
```

Assume now the same tank is reused with a solution of water and ethanol. The phenomenon is also the same, what changes is the formulation of the matter balance since it is medium dependent:

```
model Tank
  Real waterInflow,ethanolInflow,mixOutflow,mixMass
equation
  der(mixMass) = waterInflow + ethanolInflow - mixOutflow;
end Tank;
```

Of course, it is necessary to have a law declaration in order to describe properly the matter storage phenomenon when the matter is a solution of water and ethanol.

In the same way as the model interface demands a sort of delayed binding, the behaviour representation inside a model has to provide with the capability to achieve the mathematical behaviour formulation according to the model reusing context. This capability can be compared with the *dynamic binding* feature in conventional object oriented programming languages. While objects execute the proper method realization depending on the call parameterization, the proper law instance describing a phenomenon can be set depending on the involved entity. A proper separation between the physical knowledge representation (e.g. phenomena and medium) and computational knowledge (e.g. law formulation) will be used in PML in order to be able to postulate dynamically the adequate mathematical formulation. The implementation of the dynamic binding in PML will be presented in detail at Section 4.5.7.

3.2.3. Modular representation of physical knowledge

Previous discussion has introduced several requirements which should be fulfilled to guarantee model reusability. An object oriented modeling language should provide with structures supporting the physical knowledge modular representation by means of:

- Abstract interfaces which describe physical interaction between subsystems. The interface definition should allow the derivation of the one-to-one submodel coupling relationships and the overall physical interactions which appear once the topological scheme of the final model is specified.

- Declarative physical knowledge offering the explicit information required to adapt a model into a particular reusing context. This representation must be familiar to engineers, providing a complete uncoupling between declarative and procedural knowledge.

In the equation based object-oriented approach, the property of modularity is associated to the model class concept. All the physical knowledge is decomposed in model classes since the model class is the only structuring unit. The model class can be considered modular when the information hiding property is taken into account. But the mathematical interface provided by the model class does not assure a free context reusability. As the matter temperature modeling example 3.1 has illustrated, it can not be guaranteed that aggregating models to define a new model will lead to a proper aggregated behaviour representation, at least without establishing some type of model reusing rules defining the context of each model reuse. For instance, no more than one transport components can be connected between two temperature source components.

The PML language will introduce new modeling classes in order to predefine the physical knowledge and behaviour in a modular way. The new modeling classes will be modular structures satisfying the following requirements:

- *Modular decomposability*: the representation of the knowledge required to represent the system physical behaviour can be decomposed into the modular structuring units provided by the language.

- *Modular composability*: the modular structuring units can be combined with each other to produce new system behaviour representations.
- *Modular understandability*: the modules can be can be easily understood without having to know the others or, at the worst case, by having to examine a few of the others.
- *Modular continuity*: a small change in a modular structuring unit may affect just one module, or a small number of modules.

Analogous requirements of *Modular decomposability*, *Modular composability*, *Modular understandability* and *Modular continuity* are defined in the object-oriented software construction domain (Meyer 1997) as the basic criteria used to consider a design method worthy of being called *modular*.

The modular structuring unit is named PML modeling class or, in a shorter form, modeling class. It responds to the modeling component concept described in Definition 1.3. In order to fulfill with these modularity requirements, the following rules (Meyer 1997) have been considered in the design of the modeling classes (structuring units) provided by PML:

- *Direct mapping*. The modular structure devised in the model should be compatible with the modular structure devised in the modeled system. This rule applies to the concept of structured model and topological modeling. But it also applies to the modular structures (modeling classes) provided by PML to represent the physical knowledge (see Sections 4.2 and 4.3).
- *Information hiding*. The design of a modeling class establishes which set of class attributes or operations are made available to the other modeling classes. This public part of the modeling class constitutes its interface (see Sections 4.5.1 and 4.5.3).
- *Few interfaces*. Every modeling class should communicate with as few others as possible. This rule follows from the modular continuity criteria: if there are too many relations between modeling classes, then a small change in a class may propagate to a large number of classes. PML defines a communication protocol with two rules (see Section 4.5.1).

- *Small interfaces.* If two modules communicate, they should exchange as little information as possible. Actually, the PML class interfaces do not define the exchange of information in terms of variables shared between a module and its environment. Instead, they define the physical relationships between the communicating modeling classes (see Sections 4.5.1 and 5.2).

- *Explicit interfaces.* When two classes communicate, this must be obvious from the roles played by the classes. This rule follows from the decomposability and the composability criteria in order to alleviate the modeling burden. In PML, the communication between the modeling classes always reflects a physical concept, making easy to the model user and developer the definition of the relationships among the classes.

It can be noticed that most of the previous rules are related with the modeling class interface. This is quite obvious since the object oriented model construction is basically a structuring method. Hence, the success of the method depends to a great extend on the capability of the structuring units to cooperate between them.

3.2.4. Discussion

This section has been focused on the discussion of the modularity requirements which should be imposed to a modeling environment in order to increase as much as possible the reusing capabilities of the modeling classes.

Many of these capabilities depend on the modeling class interface, since it is the public part on the class and establishes the communication with the other classes. Its definition depend on the formalism used to represent the physical behaviour encapsulated in the modeling class (e.g. the model). From Example 3.1 it can be concluded that model interfaces in the equation based object-oriented modeling languages become more and more complicated as the number of equations describing new phenomena in a model is increased. The problem is that the increasing complexity frequently affects models which are not related to the phenomena that motivated the interface modification. This fact punishes the four modularity criteria described above.

To have structures with a poor modularity to represent the physical behaviour reduces the model reusability to the context where they were designed. This is one of the main reasons for proposing a new formalism to represent the physical behaviour. PML will introduce new structures (modeling classes) to accomplish with the modularity criteria. The expected main benefit is to extend the reusing context of the modeling classes in different meanings:

- Reuse of every modeling classes: all the physical knowledge is represented by means of a class structure. A PML modeling class can be reused in different contexts to declare new physical knowledge. This subject is developed in Chapter 4.

- When creating new models: topological modeling is supported, so the direct mapping rule is preserved.

- Reusing a topological model for different simulation purposes: a very important feature is the ability of the modeling environment to allow the topological model manipulation in order to fit with different experimentation goals. This subject is developed in Chapter 5.

3.3. Explicit representation of the physical knowledge

When a new language is designed to reduce the modeling burden, two main questions should be answered: does the language provide with reusable representation structures?; Does the language make easy the reuse of the representation structures?.

Leaving aside the requirements to be fulfilled by the representation structures in order to assure a good reusability degree, a not less important aspect is the facility given to the user in order to reuse them. This later question is related with the language pragmatics ,i.e., the expressiveness understood as the language ability to express the system behaviour in terms close to the physical knowledge. This aspect is also a consequence of the direct mapping rule.

The language expressiveness becomes of major relevance considering that two type of tasks, likely involving two type of users, are present in the modeling domain: the model building process itself and the model use procedure. In both cases the user deals with the physical knowledge

representation: first, when new system models are set up, second when ready-made topological models are used for some experimentation purpose. This section presents the very basic ideas which are behind the proposed modeling language PML to achieve these two main objectives:

- **Easy physical knowledge representation**. This assertion is somehow deceitful. The knowledge representation is not an easy task. But it is not less true that a proper language choice may reduce the representation difficulties. PML tries to accomplish with this intention by preserving the topological structure of the system in the model construction (model interactions representing the analogous system interactions is a way to represent physical behaviour) and by providing with modeling classes to express explicitly the physical knowledge. The modeller will find in these classes the mechanism to represent his physical knowledge about the system behaviour.

- **Easy use of models**. When a topological model is going to be used for some experimentation purpose two questions have to be solved:

    1. Structured models with declarative behaviour are not suitable for the simulation purpose, so they have to be manipulated. The derivation of the simulation model should be automated and the user should not be involved. However, if for any reason the user must take part in this procedure, the modeling tool should be able to advise him in order to take the proper decisions.

    2. The experimentation purpose makes advisable to operate with a simulation model where only the phenomena of interest (experimental framework) are formulated with the desired adequacy level. This second question depends on the language capability to discard or neglect those phenomena which are not relevant for the simulation goal and on the capability to formulate the selected phenomena of interest by means of the adequate equations.

An automated modeling tool should be able to deal with both questions. By considering that the final goal of modeling is to set up a proper mathematical formulation of the system

behaviour, the procedure followed by the modeling tool should not differ very much from the way an engineer proceeds to formulate the simulation model. The engineer analyzes the system and applies to Physics in order to make an abstraction of the laws ruling the phenomena and the physical relationships present in the system. From such analysis, both the phenomena occurring in every system component or unit and the phenomena derived from the system topology have to be formulated. The right side of Figure 3.5 shows the human procedure (black arrows) according to the modeling process described in Chapter 1, which can be characterized by the modular approach rules:

- *Direct mapping.* The system topology (system components and connections) defines a scheme which is advisable to use in order to structure the modeling problem. This modular system specification should be present in mind to determine which phenomena should be described (related to the experimental framework) and where they occur (related to the system topology). The system topology is used to set part of the phenomenological specification where the phenomena of interest are established.

- *Few, small and explicit interfaces.* The system components interact by exchanging matter or energy. These exchanges establish the physical interactions among the components which are represented in the simulation model. The physical relationships among the system components can be derived from these component physical interfaces by analyzing the topological schema. This will complete the phenomenological specification.

- *Information hiding.* The topological and phenomenological specifications are an abstract representation of the system behaviour that describes what is happening in the system. The formulation of the laws ruling the phenomena of interest leading to the simulation model can be stated afterwards. Hence, it can be considered that the structure specifications encapsulate (hides) the system behaviour formulation.

The next steps in the modeling trajectory will depend on the modeling tool. If a causal tool is used (equation or block oriented), the remaining steps are the mathematical formulation of

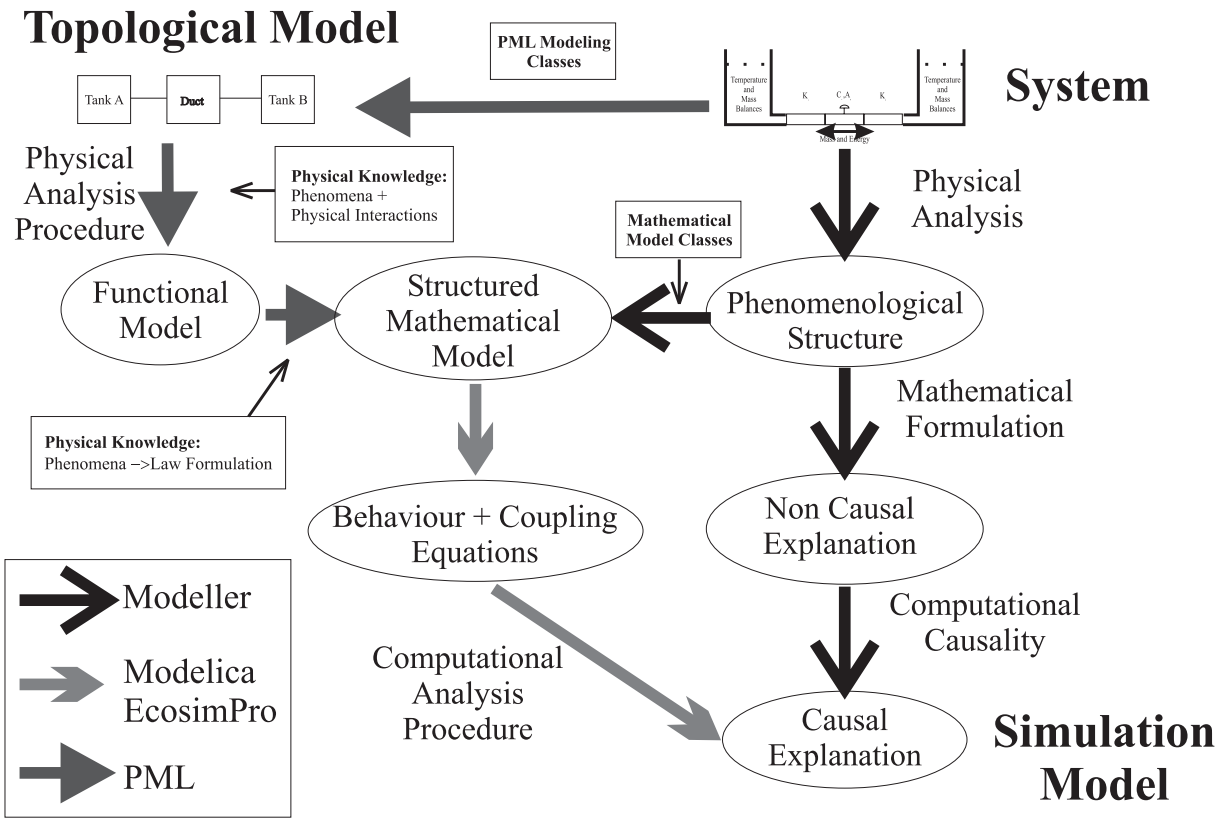


Figure 3.5. Different paths to be followed by the modeller to set up the simulation model depending on the modeling approach.

the phenomenological structure according to the adequacy level and the computational analysis to obtain the causal explanation fitting with the simulator relation. If a equation based object-oriented tool is used, the modeller will try to find in the modeling libraries the model classes able to represent the phenomenological specification according to the experimental framework and with the desired adequacy level. If they exist, he can reuse them to build an aggregated model matching with his phenomenological structure. If they do not exist, he will have to adapt, or even define from the scratch, the model classes suitable for the phenomenological specification. We have tried to formalize the human way to proceed because this procedure is in the aim of the PML modeling methodology. The modeller procedure is illustrated in the following example where the modular nature of the system is used to formulate its simulation model.

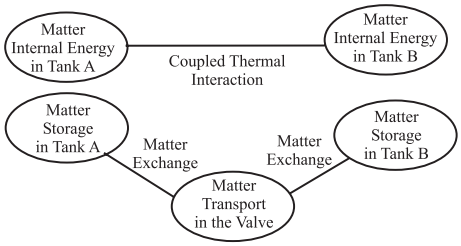


Figure 3.6. Phenomenological structure of the two tank system in Example 3.2.

EXAMPLE 3.2 – *Simulation model of the two tank system*

By analyzing the topology of the system at Figure 3.3 it is observed that there are two system components where matter is stored and a system component where matter is transported. From this topological specification it is also deduced that there exist physical interactions between both tanks due to the matter transport (matter properties carried with the flow). The formulation of these interactions will depend on the experimentation framework. The thermal interaction will be considered in this example. So, we have to consider the matter and energy storage phenomena. The phenomenological specification is shown in Figure 3.6.

Applying to the physical laws, the matter storage and transport phenomena can be formulated by using the mathematical expressions:

$$\begin{aligned} m'_A &= -m_q \\ m'_B &= m_q \\ m_q &= Ksign(\Delta P)\sqrt{abs(\Delta P)} \end{aligned} \tag{3.6}$$

where m_A and m_B are the matter masses in each tank and m_q is the mass flow rate between them (the relationship between stored mass and pressure drop ΔP has not been included for simplicity reasons). It has been considered that matter flows from tank A to tank B if ΔP is positive.

The following causal explanation is deduced from the phenomenological specification when the temperature dynamics is included (Equation 3.3 at Example 3.1 has been used to represent

the temperature dynamics):

$$\begin{aligned} m'_A &= -m_q \\ m'_B &= m_q \\ m_q &= Ksign(\Delta P)\sqrt{abs(\Delta P)} \\ m_A T'_A &= m_q(T_A - T_q) \\ m_B T'_B &= m_q(T_q - T_B) \\ T_q &= \text{if} \quad m_q > 0 \quad \text{then} \quad T_A \quad \text{else} \quad T_B \end{aligned} \tag{3.7}$$

This mathematical model can be represented in a equation oriented language (e.g. ACSL) or a block oriented environment (e.g. Simulink) without any further manipulation. ❑

To sum up this modeling procedure, the engineer has visualized the system by means of the topological specification. He has considered the phenomena occurring at each system component and the physical interactions derived from those phenomena. He has used the system topology to derive the thermal interaction between both tanks. It should be noticed that the matter temperature dynamics has been established by considering the whole system topological structure. The key factor in this modeling procedure is that the phenomenological structure, which must be stated before any consideration about the behaviour mathematical formulation, can be obtained from the physical analysis of the topological specification. To do this, we need to focus on the phenomena occurring at each system component together with the physical interactions established from the matter and/or energy transfers defined by their connection topology.

In order to automate this modeling procedure, it would be necessary to make the same type of physical analysis. That is, it should be possible to obtain the *functional model* (see Definition 1.10) from the phenomena declared at each *topological model* component (see Definition 1.9) together with the physical interactions defined by their connection topology. Hence, we must determine how the *topological model* should articulate the representation of the physical knowledge and behaviour in order to design the equivalent *Physical Analysis Procedure*.

The mathematics language is far to give a the level of expressiveness required to support

model construction at the topological specification level as it is defined in this thesis. It should be remarked that the interpretation of the equation as a static modeling formalism is not related with the capability to manipulate the equation in order to find a proper computational assignment from the reusing context. This capability is called declarative behaviour representation (see (Andersson 1994) for instance). The static characteristic is a consequence of the poor equation capability to report the represented physical behaviour. This *information* is required to analyze a model from a physical perspective. For instance, the equation $e = R * f$ may be used to represent a resistive relationship between the e and f physical quantities. This equation can be used both to model the Ohm's law in an electrical resistor and the matter transport in a fluid pipe. However, the aggregated behaviour derived form a resistor inserted in a electrical circuit or from a fluid pipe between two process units are completely different. The resulting coupled behaviour can only be stated from the system physical analysis. As it has been illustrated through Examples 3.1 and 3.2, the physical analysis of a system where matter flows through a pipe may lead to different phenomenological structures depending on the elements which are connected to the pipe and also depending on the experimentation purposes. Such physical analysis can not be automated by using a mathematical formalism. Equations describe how a phenomenon can be computed but they are not able to describe the phenomenon itself, hence they do not represent the physical knowledge used by the engineer to deduce the phenomenological structure from the system topology.

Therefore, the physical analysis of the system must be performed by the modeller before building the proper object-oriented mathematical model (see the modeling path at the center of Figure 3.5). The modeller must go inside the equations of the predefined models in order to decide how to reuse them in the physical context stated by his experimental framework. That is, he builds the model at the phenomenological level.

In order to provide with a modeling environment able to replicate the physical analysis performed by the engineer at the topological level, the modeling language must be able to represent the analogous physical knowledge used by the engineer. A modeling language where the physical knowledge is explicitly represented is needed because of two reasons:

- Technical aspect: the modeling tool should be able to perform the physical analysis of the topological specification in order to find the proper phenomenological structure. This procedure should be based on deterministic physical knowledge that is explicitly represented in the modeling classes.

- Pragmatics aspect: when the model user has to go inside a model, the modeling language expressiveness allows him to retrieve the behaviour represented in the model, without needing to interpret the physical meaning of an equation.

These considerations have conditioned the design of the PML language. In PML the system behaviour is described with modeling classes where physics knowledge is explicitly represented. The PML modeling class is a modular representation structure according to the modular rules in the previous section. This makes an important difference from existing object oriented modeling languages where behaviour is described in terms of mathematical equations. In PML, the physical behaviour is mathematically represented just when it is required to formulate the non causal explanation to the phenomena of interest.

The modeling trajectory followed by the PML modeling tool (PMT) is illustrated at the left side of Figure 3.5. First, the structure of a PML topological model and the physical knowledge expressed by the language are used to analyze the whole (coupled) system behaviour in order to deduce the phenomenological structure (the functional model). For example, the pipe model of the two coupled tank system at Figure 3.3 would describe the matter transport through the pipe. It is deduced that there is a convection flow related to the mass transport through the pipe. This will lead to the phenomenological structure shown at Figure 3.6. Such interaction does not depend just on one single component but on more subsystems which are not directly interconnected. Therefore, such kind of interactions can be formulated only after the whole system connection structure is considered. This analysis is based on the physical knowledge represented by the PML modeling classes (they are introduced in the next section). In a second term, the physical behaviour represented in the functional model is mathematically formulated according to the physical laws ruling the phenomena. Currently, the PML modeling tool performs this

formulation by means of an equation-based object oriented modeling language (current version generates EL, the EcosimPro Language).

With respect to the language pragmatics, the modeller may observe in the modeling classes the physical concepts related to the system behaviour representation. The PML language expressiveness has been designed to attend this demand by defining the modeling classes as a representation structure of the involved physical concepts such as phenomena or laws (the complete language specification is developed in Chapter 4).

Finally, the PML design has considered the support of model manipulations in order to generate efficient models where the complexity is adapted to the experimentation purpose. A representation level that makes easy to understand which manipulations have to be performed in order to adapt the simulation model to the experimental framework with the desired adequacy level is therefore needed. For example, it is not easy to anticipate the consequences derived from the elimination of an equation from the set of equations declared in a mathematical model. It will be much easier to say that the energy storage phenomenon will not be considered for some experimentation purpose. This model adaptation facility depends on several factors:

i) The modeling language expressiveness which makes possible to easily identify which phenomena are not relevant.

ii) The ability to check the consistence of a model where part of the represented behaviour is neglected.

iii) The capability to dynamically find the mathematical formulation of the laws which describe the remaining phenomena.

All these considerations have been taken into account in the PML language design. The next section gives an introduction to the modular modeling structures defined by PML and to the physical analysis procedure applied to derive the simulation model.

3.4. Modular modeling with PML

PML is a modeling language designed on the basis of physical knowledge modular representation stated in Section 3.2. The language introduces modular structures to represent system behaviour and interactions in such a way that the equation oriented formalism can be dynamically stated once the model reusing context is analyzed. The proposed modeling methodology is based on the object oriented and modular model development paradigm, preserving the system ↔model structure analogy.

3.4.1. Representation of system behaviour in PML.

The language has been designed to support the knowledge needed to analyze the model behaviour from a physical approach rather than from a computational causality analysis perspective. Hence, different representation structures have been defined to declare the physical concepts involved in a modeling problem: system and subsystems, system processing subject (basically matter or energy), system component interactions, phenomena and laws. Considerations taking into account the physical aspects in a modeling problem instead of the computational aspects have aimed at the definition of these structures, so called PML modeling classes.

The main modeling classes are shown in Figure 3.7 (the complete set of PML modeling classes will be presented in Chapter 4). In order to preserve system↔model analogy, the PML model class is tightly related to the system component. Therefore, the end user may readily identify the duality system↔model component in the library of predefined models. The behaviour representation in a model has been split into two main context independent structures: entities and phenomena. The entity is the subject of the physical process carried out by a system component, usually matter or energy. The phenomena represent the system physical behaviour which is declared in the model.

The phenomena affect or modify the entity properties, and a law structure is used to formulate such dynamics. For example, a tank (model) is a system component where matter (entity) is stored (phenomenon) and can be formulated by means of a matter balance (law).

It should be noticed that the model only declares entities and phenomena. The knowledge

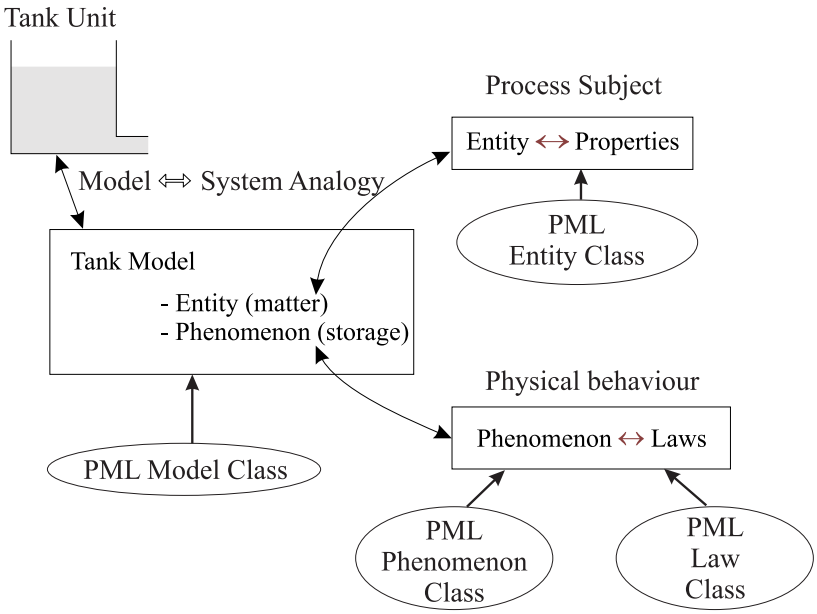


Figure 3.7. Tank PML model: involved representation structures

of the phenomena represented in a model makes possible to find the adequate law to describe it: a mass balance can be applied to state the matter mass property dynamics. This step, basically consisting in the behaviour mathematical formulation, is known in PML as the translation procedure from the physical to mathematical level (see sections 3.5 and 5.2.2).

The are two main reason to represent phenomena and laws in two separated modeling classes: the first one is the phenomenon declaration is context independent but its formulation through a law depends on its reusing context; the second one is the same phenomenon may be described or formulated in different ways (support to adequacy variation). This is the role played by the PML law class, so that different law classes may be defined when different formulations are required. For example, a matter transport phenomenon occurring in a duct will require different formulations depending on the laminar or turbulent flow conditions, or even on the state (solid, liquid or gas) of the fluid being transported. Actually, the mathematical description of a phenomenon can not be properly determined until the physical context where the phenomenon occurs is fully specified.

These representation structures introduce several advantages over equation based modeling languages where behaviour is particularized in terms of mathematical equations:

- The PML models specify phenomena and entities, giving to the model user a clear information about the modeled physical behaviour.
- The language semantics is very closed to the user's physical understanding of the system.
- This information is completely uncoupled from the computational aspects, which are stated afterwards as a set of equations when the simulation model is required.

The procedure of determining the adequate law instances, which describe a phenomenon in a particular reusing context, is called dynamic binding (this PML property is developed in Section 4.5.7). This feature eliminates the necessity of having different models to represent the same system component when the process entity is changed (this question was discussed in Section 3.2.2). With PML, a unique model class specifying the process entity is required. The proper law instances are dynamically selected before equations are generated.

A second main contribution is related to the simulation model generation, where only those phenomena of interest will be considered in order to achieve an efficient simulation code. The model user may derive efficient simulation models by selecting those phenomena suitable for the experimentation goal. For instance, let us consider that a tank model represents all the thermodynamical phenomena related with matter storage. Only mass dynamics should be considered when the simulation goal is to test a level controller, neglecting the rest of irrelevant phenomena.

Finally, the separation between the phenomenon and law structures supports multi-faceted models. Several authors define *multi-faceted* as an intrinsic characteristic of models since a model is an abstraction of the real world (Zeigler 1984, Marquardt 1991). Therefore, the behaviour incorporated in a model may depend on many factors, varying the required degree of detail or complexity. Among the causes for this variety of models are the experimentation goal, the incomplete knowledge of complex phenomena or even the background of the modeller. These factors usually lead to a family of models for the same process or system, breaking with the

system⇔model duality. In order to make possible the system⇔model unambiguous relationship, the PML model representation structure should support this multi-faceted nature (Nilsson 1993, Stephanopoulos *et al.* 1990*b*, Bogusch *et al.* 2001).

Basically, multi-faceted models are models including several representations of behaviour (realizations) with different degrees of detail and complexity. For instance, a medium transport in a pipe line may be represented as $w = K\sqrt{\Delta\ P}$ or as a simpler linear expression $w = K(\Delta\ P)$ (w is the mass flow rate and $\Delta\ P$ the pressure drop in the pipe). The model user may select one of these realizations depending on his simulation interest, particularized by the experimental framework and adequacy level. In PML this selection is straightforward since different laws or, more precisely, different mathematical law descriptions can be related to a phenomenon. Hence, since the model declares phenomena and these phenomena can be formulated in different ways, the variety of behaviour formulations can be considered as the different facets of the system model. This PML feature will be developed in Section 4.2.2.

3.4.2. Representation of system interaction in PML.

Models are defined by specifying the physical behaviour in terms of phenomena and involved entities, but they can also be recursively built as the connection of already built models. In this case, the user specifies the relationships among submodels by preserving the analogy with the system topology, i.e., he builds the topological model according to the modeling process described in Chapter 1. Applying to system modeling, physics could be seen as a problem of trading matter and energy (Cellier 1991). Systems exchange matter and energy between them. It would be desirable that models representing systems could describe mass and energy exchange in a similar way. In PML the interaction between connected submodels is defined in terms of entity exchange (for instance, matter or energy). This definition of the model interface is independent of the variables needed to establish the relationships between the internal behaviour equations that compute the physical phenomena and even from the phenomena declared in the model. The concept of **port** is introduced as an interface defining an exchange of an entity (for instance, matter or energy). For example, a PML model interface declaration would say that a tank model

exchanges matter through a certain number of ports.

This interface definition is guided by the fact that the development of basic component models should not be subject to the analysis of the context in which they will be reused (Ramos *et al.* 1998*b*). Hence, the model interface definition has been made independent from the phenomena declared in the model. The connections by means of entity exchange preserve the system↔model analogy, describing the analogous connections between subsystems. This abstract interface provides the desired level of delayed binding. When model interface is specified, only physical considerations are taken into account. Once the computational behaviour equations are derived in a model, the port is used as the path to find those physical interactions which are not modular to the model.

3.4.3. The modular modeling rules in PML

With the PML modeling classes two main goals will be accomplished (its fully understanding is developed in chapters 4 and 5): the first one is the modular modeling rules fulfillment; the second one is the replication of the modeling procedure followed by an engineer which was illustrated in Example 3.2. The PML modeling procedure, in analogy with the engineer modeling procedure (see Figure 3.5), is characterized by the following modular modeling rules:

- *Direct mapping.* The system topology (components and connections) defines the scheme used in order to structure the modeling problem. A PML model class is a structured representation defined by preserving the analogy with the system. Every PML model component may either represent a system component or represent the physical behaviour of interest in terms of phenomena and entities. In the first case the model component is named *submodel*. In the second case, the model component should be an instance of a PML phenomenon class or a PML entity class.

- *Few, small and explicit interfaces.* The PML model class interface defines the exchange matter or energy. The formulation of the physical relationships among the model classes is stated from these physical interfaces by analyzing the connection topology. The PML model interface can be considered explicit since it represents the same type of connection

that the system has, small because it declares just the exchange of matter or energy with no further information (the model interface does not depend on the phenomena declared in the model). A PML model is said to have few interfaces since their number equals to the number of the modeled system connections.

- *Information hiding.* A PML model is an abstract representation of the system behaviour that describes *what* is happening in the system. The formulation of the laws ruling the system behaviour (leading to the simulation model) is stated afterwards. Hence, it can be considered that the model encapsulates (hides) the system behaviour formulation.

3.5. Levels of system specification with PML

The previous section has introduced the PML modeling classes designed to represent the physical knowledge and behaviour. The PML model class is close to the physical and structural aspects of the system, but it is far from the computational aspects required for simulation.

The structural and computational aspects of a model may be considered at two different levels of system specification. The structure level represents the components and how they are coupled together, while the computational level represents the states and their transition mechanisms (how states are affected by inputs and which are the new states after the input stimulus is over). This is an interpretation of the structure and generative levels of system knowledge introduced in General System Theory (Klir 1985) which has been used in Chapter 1 to define the modeling process adopted in this work. Many of the modeling approaches (see for instance (Marquardt 1991, Zeigler *et al.* 2000)) claim for the necessity of being able to go from one level to the other, since the structural level makes the model construction easier and the generative level is required for simulation.

In the PML modeling environment, there is a translation from the structural or coupled component level into the generative level. The system specification at the topological level is built with the PML modeling classes by reflecting the system structure (system topological representation). Hence, the PML language defines the system specification formalism at the

topological level. The causal generative specification level is what we call simulation model (according to Definition 1.12). The differential equation formalism is used at this level.

Nevertheless, obtaining the aggregated behaviour from the component coupling is not a trivial task that must be carefully planned, as it has been discussed through Examples 3.1 and 3.2. The aggregated behaviour can be defined as the set of phenomena occurring at each component together with the coupling phenomena derived from the component connection topology. The set of phenomena and their physical relationships conform the phenomenological structure of the system specification. As the previous examples have illustrated, there is not always a perfect match between the model phenomenological structure and its topological structure. For instance, according to the model multi-faceted nature, the phenomenological structure of the two tank system model differs whether the thermal behaviour is considered or not.

This is the reason why in PML two different system specifications are considered at the structural level. These are the PML topological model (which is built by the modeller) and the PML functional model (which is obtained from the physical analysis of the former).

DEFINITION 3.1 – *The PML topological model*

The PML topological model represents the system topology (physical components and connections). It can be represented as a graph structure where the nodes represent the submodels and the edges represent the exchange of matter or energy among nodes. It is declared by means of the PML model class and it should preserve the analogy with the system both in components and their connections. ❑

Through this system representation the user interacts with the modeling environment. The topological model offers the user explicit documentation about the system: the model preserves the analogy with the system, both in internal physical knowledge and interaction specification. This information, which is familiar to engineers and uncoupled from the computational knowledge, can be used to specify all the hypothesis, assumptions and simplifications that give rise to a particular simulation model because of the multi-faceted nature of a PML model and because of the capability to analyze it from the physical behaviour perspective.

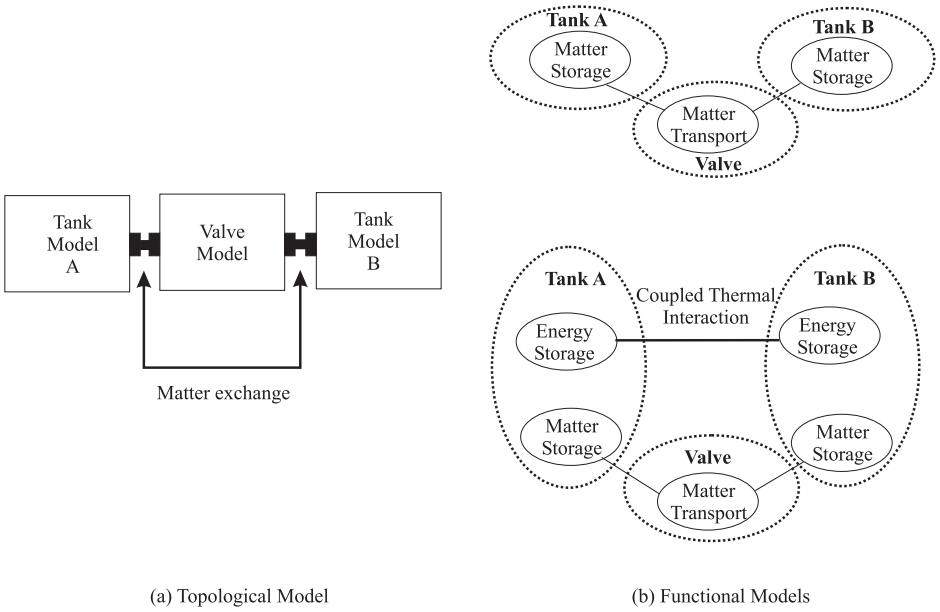


Figure 3.8. The PML structural level of the two tank systems: (a) topological model; (b) functional model. The dotted ellipses indicate the relationships between the phenomena and the coupled components.

Definition 3.2 – *The PML functional model*

The PML functional model represents the phenomenological structure of the PML topological model attending upon the experimental framework. It can be represented as a graph structure where the nodes represent the phenomena and the edges represent the physical interactions among the nodes. ❑

The functional model is closer to the model computational functionality than the topological model. This is the reason for naming the phenomenological structure as functional model. There are two important characteristics in the functional model. The first one is the functional model is automatically obtained from the topological model, avoiding to involve the model user in the problem of determining the proper phenomenological structure of the given topological model. The second one is there is not a unique association between the topological and the functional models and the phenomenological structure can be adapted to the experimentation purposes. This feature fulfills with the considerations about the reusability extension discussed in Section 1.2 and it is a consequence of the physical analysis capabilities. Figure 3.8 shows

the topological model of the two tank systems and two possible associated functional models. The functional model on the top does not include the thermal relationships, while the thermal tank interactions derived from the matter transport have been included in the bottom functional model. As it can be observed in the bottom functional model, the connection topology does not match with the connections in the topological model.

Some analogies between the functional model and a bondgraph may be found. Actually, they are quite similar in concept: the nodes in the functional model represent phenomena (which can be considered of capacitance or resistive nature by using bondgraph terminology) and the edges represent occasionally a flow/effort relationship similar to the bonds (see Section 4.3.1 for more details). However, the functional model connection structure is not tight to the rules guiding the bond definition, i.e., an edge in the functional model does not always represent a flow/effort couple.

Functional model generation

Both system specifications at the structural level (topological and functional models) are necessary to describe the system but they are not independent (they are not *orthogonal* in the sense described at (Marquardt 1991)). The topological model is given by means of the PML model class and it is used to derive the functional model representing the phenomenological structure. The PML modeling tool (PMT) analyzes the topology of connections defined by the PML model class in order to establish, from a physical perspective, the overall physical interactions. This translation procedure, from the topological to the functional model, is named as *Physical Analysis Procedure* (Ramos *et al.* 2001) and it will be described in detail in Chapter 5 according to the PML language semantics. At this point, a short description can be given.

The physical causality analysis procedure examines the topological model deriving the physical interactions from interconnected ports together with the phenomena described at each submodel. On a first stage of the procedure, the topological model consistency is checked by matching each model component declaration with the knowledge defined in the libraries. The procedure analyzes the model structure in a top-down sense, retrieving the topology of sub-

model connections and the entity exchange through the ports. Then, each submodel is analyzed in order to determine which laws are going to describe the phenomena. Applying to physical knowledge, the used relationships are: submodels represent entities and phenomena; the laws ruling the phenomena are found and, consequently, the dynamics of the involved entity properties can be described by these laws. In the two tank model shown at Figure 3.8, the matter mass can be stated from the mass balance (law) related to storage phenomenon and the matter exchange defined by ports. The remaining behaviour representation (the thermal interaction), which is not modular to the tank submodel (see Example 3.1), is derived by the physical analysis of the topological connections. In the PML model, the thermal interaction between tanks (not explicitly described by ports) is automatically deduced from the matter stream path defined by the valve and matter exchange defined by the port connections. This phenomenological structure is represented by the functional model (see the bottom graph at Figure 3.8(b)).

Dynamic modularity

The functional model represents the phenomenological structure in a modular way. This assertion can be easily proved by recalling the modularity rules posed in Section 3.2.3:

- There is a direct mapping between the functional model nodes (phenomena) and the specified system behaviour.
- The functional model edges define the minimum set of phenomena interactions derived from the specified system component structure (few, small and explicit interfaces).
- The functional model encapsulates the behaviour mathematical formulation since it describes the behaviour in terms of phenomena and physical interactions.

This modular structure is not predefined in the PML modeling classes, since it is the result of the topological model physical analysis. Hence, it may be asserted that the modular structure is dynamically obtained from the physical analysis procedure. The phenomenological structure is a system *dynamic representation*: the system behaviour is formulated once the physical context of every component has been considered.

Definition 3.3 – *Dynamic modularity*

Dynamic modularity is the PML capability to state the proper phenomenological structure from the topological model physical analysis by preserving the modularity criteria. This structure is represented by means of the functional model. ❑

The dynamic modularity is a consequence of the PML capability to analyze the model from a physical perspective. The capability to set automatically the phenomenological structure from the topological model makes possible to reuse a PML model independently from the context. A free context model definition means an important increase of the model reusability. For example, the same valve model representing the matter transport can be reused when the tank models just describe the storage phenomenon and when they also include the thermal dynamics. This feature is very related to the PML capability to dynamically bind the adequate formulation which describes a phenomenon in a particular reusing context. The dynamic binding allows the separation between the system specification at the structural level and the generative level. While the differential equation or, generically, the equation based formalism is a static behaviour formalization in the sense of Definition 1.7, the PML language establishes a dynamic formalism in the sense of Definition 1.8, since the behaviour is formulated once the physical context has been considered.

Despite the *Functional Model* represents the adequate aggregated behaviour (e.g., see Figure 3.8(b)), it is still far from the causal generative level (simulation model) required to fulfill with the simulator relation. Hence, the functional model has to be translated into this system specification level. Due to the functional model modularity, it can easily be described by using an equation-based object-oriented modeling language(such as Modelica or EcosimPro). The equations formulating the behaviour are obtained from the laws associated to the phenomena represented at each functional model node. The physical interactions represented by the functional model edges are translated into the mathematical terminals provided by those languages. This stage in the translation procedure from the structural level into the generative level is tackled in Chapter 5. Because of the multi-faceted nature of the PML models, the mathematical formulation can be adapted to the required adequacy level by selecting the proper law instance.

3.6. Summary

This chapter has presented the formal framework where the modeling language PML has been designed. The object-oriented paradigm has been adopted as the model construction methodology. The PML modeling structures (or classes) have been designed to support the physical knowledge modular representation.

The need of a new formalism supporting the physical knowledge modular representation is motivated by the limitations of present equation based object oriented modeling languages. The differential equation is qualified in this work as a static formalism, since the physical context where the model can be reused must be assigned before the behaviour and interactions of the model are declared. This results in model structures which are not modular in the sense of physical knowledge and behaviour representation. As it has been illustrated with a very simple system, modeling difficulties arise when the behaviour is represented using the differential equation formalism since the behaviour of aggregated or structured models (coupled phenomena and property propagation) can not be statically predefined in a modular way.

It has been shown the necessity of a modeling formalism providing with modular physical knowledge and behaviour representation structures in order to support the object oriented methodology and make possible a hierarchically structured model construction according to the system topology. The PML language defines a modeling framework where the system is specified at the structural level according to the system component coupling structure. This system specification has been called the *topological model* and it is supported by means of the PML model class. Since the topological model does not always preserve the phenomenological structure, a second system specification has been introduced in order to represent the system aggregated behaviour in a modular way. This structure has been called *functional model*, and it is automatically generated from the topological model by means of the *Physical Analysis Procedure* (see Chapter 5). The result of this procedure is a system specification at the structural level where the dynamics of interest for the experimentation purpose are represented in a modular way. The analysis of models from a physical perspective is feasible due to the dynamic binding

feature provided by the PML modeling classes. Thanks to the functional model modularity, this structure can be easily represented by an object oriented modeling language based on equations, leaving to these modeling tools the translation into the system specification at the generative level required for simulation.

The two following chapters will present the language PML and the way in which PML models are manipulated in order to find a representation suitable for simulation.

4

The Modeling language PML

With PML, the system representation is structured in two main aspects: on the one hand the representation of the system components (e.g. tanks, pumps, valves, etc.) and the processing subjects (basically entities such as energy or matter); on the second hand, the representation of the system behaviour (basically phenomena and laws). This chapter introduces the formal framework used to represent these modeling concepts. The PML syntax is specified in Appendix A. This chapter is focused on the description of the language semantics and the model construction procedure. Sections 4.1, 4.2 and 4.3 present the PML classes and the modeling procedure. In Section 4.5 the main PML object-oriented characteristics are analyzed and Section 4.7 goes through the most relevant quality factors of the methodology. Finally, Section 4.6 deals with some advanced modeling concepts.

4.1. Overview of the language

The *Physical Modeling Language* (PML) is an object-oriented language designed to represent the physical behaviour of systems. PML introduces a new formalism to define explicitly the physical knowledge required to represent the system behaviour. PML has been conceived to overcome the main limitations of the equation based approach discussed in previous chapters. The main consideration that has aimed the PML design has been the improvement of model reusability according to different interpretations of this property:

- *Technical aspects.* The lack of modularity and the derived constrains to reuse models of the equation based modeling languages has been already discussed in the previous chapter (see the Example 3.1). The organization of the physical knowledge in PML provides with modular structures assuring their reuse in different scenarios.

- *Pragmatics aspects.* The PML language offers to the model user and to the model developer a framework to represent the system behaviour in terms of physical concepts. The PML language expressiveness increments model reusability by establishing a direct association between the represented behaviour and the physical concepts which are familiar to the model user. For instance, a tank system may be described at different levels of abstraction:

  - Descriptive Level: The tank is a system where matter is stored.
  - Phenomenological Level: A tank stores matter. The matter accumulation rate is a function of the input and output flows.
  - Computational Level: $m' = \sum_i m_i$, where m is the mass holdup and m_i are the matter flows.

  The PML language is close to the phenomenological level. It is clearly more readable to see that a model is explicitly representing a matter storage phenomenon than to interpret that a differential equation is describing such phenomenon.

The model construction in PML, as an object oriented language, is a classification method. The formalism defined by PML offers a classification method supporting the *physical knowledge organization* according to physical concepts totally uncoupled from the mathematical foundation required for the physical behaviour computation.

The physical knowledge is organized around five main *modeling classes*. The definition of these classes is based on a very simple idea, a *model* represents a physical device (e.g. a process unit, an electrical component, etc) where some *phenomena* occur. Depending on the physical context, there is a physical *law* which describes how such phenomena affects the *entities* (e.g. some kind of matter or energy). Probably the physical device is a part of a larger system and it

interacts with other devices exchanging matter or energy through some type of *port*. According to this, the PML basic classes are:

- *Entity*. It represents some type of matter or energy defining its properties.
- *Phenomenon*. It is used to declare the physical behaviour occurring in a system.
- *Law*. Describes how a phenomenon occurs representing the law which rules it. The law class is close to the computational aspects of the represented behaviour since it is used, among other functions, to obtain the behaviour mathematical formulation.
- *Model*. It is used to represent some physical device or a part of it.
- *Port*. It describes the physical interactions between physical devices.

With these class structures, PML extends the classical *model* class reusability supported by the equation based approaches to the new modeling classes, such as *entity*, *phenomenon*, *law* or *port* classes. All these classes can be declared independently and can be reused in different contexts. For instance, once a matter storage phenomenon class is defined, it can be reused to specify this behaviour in different models. There are several semantic rules that must be observed to reuse a modeling class, but it is important to note that they will not depend on mathematical considerations. Every PML modeling class has an interface which specifies how the class interactions are defined. The most obvious modeling interaction is the aggregation of (sub)models to define structured models. But the PML language defines other interaction protocols among classes in order to support the object-oriented characteristics.

Another advantage derived from the PML knowledge representation is the improvement of the assistance capabilities provided to both the model developer and the model user. The model reusability is extended by supporting a computer-aided model manipulation to cope with different simulation or experimentation purposes. For instance, models can be simplified by setting hypothesis in order to obtain a simulation model according to the experimentation goal.

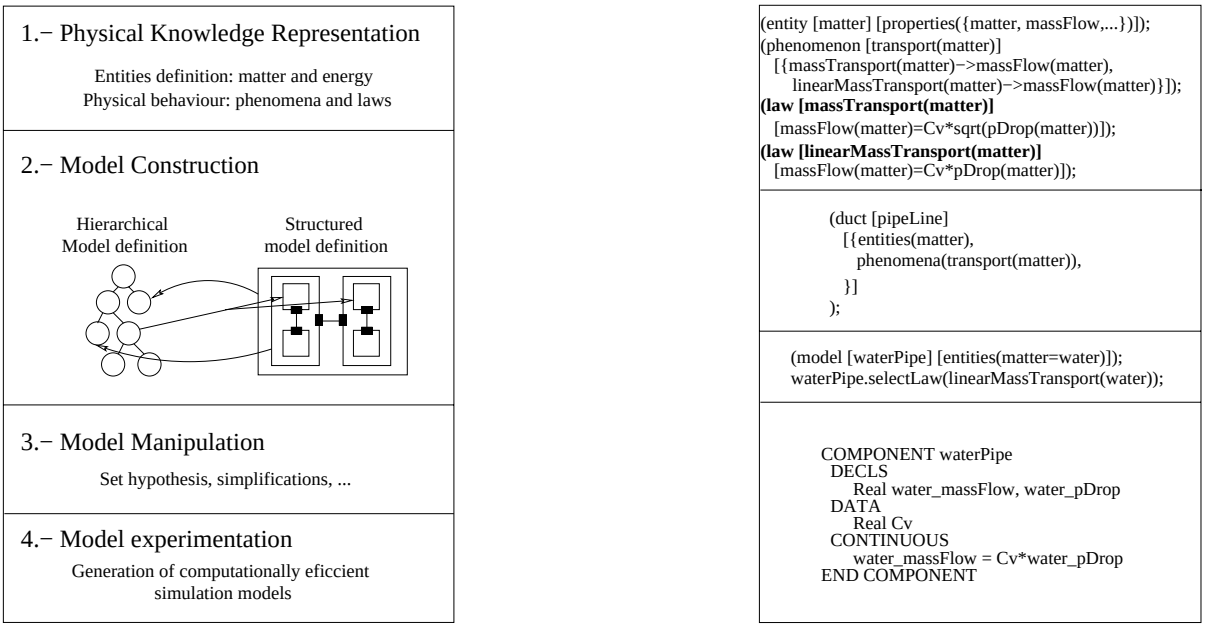


Figure 4.1. Layers in the construction of a PML model library: behaviour representation, model construction and model usage (adaptation to the experimental framework and simulation model generation).

All the PML classes are declared in the modeling libraries. The modeling library construction in PML can be viewed as a four layered task (see Figure 4.1(a)). No reference to PML syntax is required at this point:

- *First layer:* the basic physical knowledge is represented by means of the entity, phenomenon and law classes. The *entity* class represents the processed elements (basically matter and energy). The different properties of an *entity* (e.g. matter mass, mass flow or temperature) are specified in its declarations. The *phenomenon* declaration expresses which entity properties are affected and the *law* describes how this behaviour can be quantified and computed. This knowledge will be used in the following layers to build models first and to adapt such models to different simulation purposes afterwards.

- *Second layer:* the model and port classes can be defined. Models express system behaviour by declaring the phenomena. In PML the models can be atomic or aggregated (structured

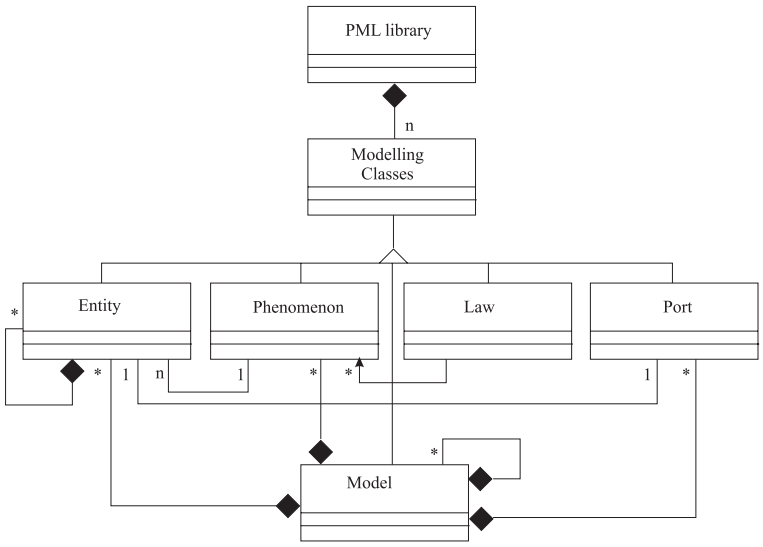


Figure 4.2. Partial UML static diagram of PML basic modeling classes

models). Structured models are build at the topological specification level. Also, new models can be defined by inheritance.

- *Third layer:* model manipulation. In this layer models already defined can be manipulated to adapt them to some experimentation purpose. The model user can set hypothesis in order to discard the behaviour which is not of interest for the experimentation goal avoiding the inclusion of unnecessary equations formulating the neglected behaviour.

- *Fourth layer:* model experimentation. The model specified at the third level is analyzed in order to formulate the mathematical model required for simulation. This step will be discussed in Chapter 5 in detail.

These four layers or steps which constitute the model construction in PML are based on the class organization shown at Figure 4.2. The two first layers are related to the model definition, while the next layers are related to the model operation. A discussion on how the basic modeling classes are declared in PML follows (the syntax is introduced Appendix A). All the aspects related to the model operation (third and fourth layers) will be presented the Chapter 5.

Entity modeling class	
`properties:`	`*property`
`components:`	`*entity`

Table 4.1
UML description of the PML entity class showing its attributes.

4.2. Basic knowledge representation

What is considered as basic knowledge in PML are the phenomena and the laws required to describe them. The class definition of phenomena and laws represents the physical knowledge which is independent from any system component model. For instance, the matter storing phenomenon is general and may be reused in any component model. Thus, the classes representing phenomena and laws can be predefined independently of their reusing context. Nevertheless, a phenomenon occurs on an entity (some type of matter or energy) affecting to a set of its properties. That is the reason for considering the modeling class entity as basic knowledge too. Therefore, the entity modeling classes should be declared together with the phenomenon and law modeling class declarations.

4.2.1. The Entity PML class

This modeling class is used to represent the processed physical object, usually some type of matter or energy. Its declaration consists in the specification of two types of attributes as Table 4.1 shows. PML entity class declaration follows semantic rule S1.

Semantic rule S1 – *Entity class declaration*

The modeling class *entity* represents the processed physical object of a particular process. An entity class is defined in terms of its properties. A property can be any physical quantity characterizing the entity. An entity can be also defined in terms of its components, i.e., as an aggregation of entities. ❑

The *properties* attribute is a list of names used to represent the properties of the entity. Establishing an analogy with the equation based languages, the list of properties is the set of variables used to represent the physical quantities in a model. For example, assume that *matter*

is an entity involved in our modeling library. To declare the properties of interest for the matter entity , the following class may be defined:

```
(entity [matter]
      [properties({mass,temperature,density,...})]
);
```

The property declaration is an entity class attribute which means that a particular entity object is characterized by its list of properties. The references to properties have to be performed through a particular entity instance

SEMANTIC RULE S2 – *Definition of entity properties.*
The properties of an entity can only be defined as an attribute of the entity class. ❑

With this rule, PML coerces to have a unique name to represent a physical quantity everywhere. Hence, the consistency of a property reference can be guaranteed in any class where the entity instance is used. The list of properties is considered in PML as the entity class interface, which makes possible to check the consistency of any reference made to a physical property. The scope of a property reference is defined by semantic rule S4.

The second attribute of an entity class may be a list of components. This attribute can be of special interest in the chemical process domain where the process substances are usually multi-component products.

SEMANTIC RULE S3 – *Multicomponent entities*
A multi-component entity is an entity class where the `components` attribute defines a collection of entities. Thus, an aggregation relationship between the component entities and the multi-component entity is established. ❑

For example, a chemical mix of ethanol and water can be defined with the following class:

```
(matter [ethanolSolution]
      [components({water,ethanol})]
);
```

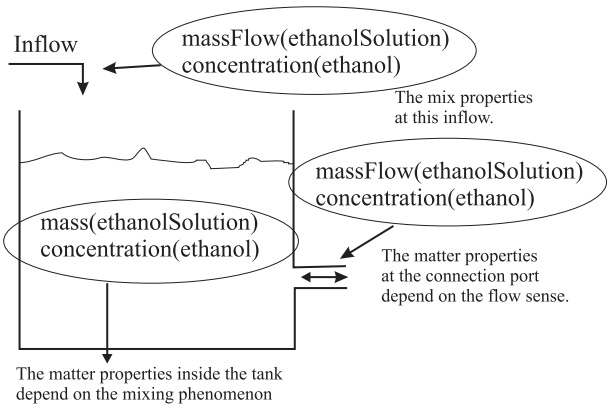


Figure 4.3. Matter property scope.

Semantic rule S4 – *Access to an entity property (property scope and bindings).*

An entity property can only be accessed making a reference to the owner object: an entity instance. The access domain of a property may be:

- *Global.* If the property P of an entity E is referenced in a phenomenon or in a law the reference will be observed in all the instances of the entity class.
- *Local.* If the property P of an entity E is referenced in a model, in a port or as a part of another entity (multi-component entities) its domain is confined to the behaviour declared in the model, in the port and, in multi-component entities, to the aggregated entity.

❑

For example, the assertion "a matter storage phenomenon affects the property mass of matter" holds for the mass property in every instance of the matter entity affected by the storage phenomenon. Let us consider now that an object of the mix class declared above (`ethanolSolution`) is used in a tank model where a storage phenomenon occurs (see Figure 4.3). Three different references to the mix entity should be made: the mix inside the tank, the mix at the inflow and the mix at the tank connection port. The PML language defines internally three objects to represent the mix depending on the scope of their use:

- The properties of the mix inside the tank are affected by the phenomena described in the tank model.

- The mix properties at the inflow stream which will be considered as boundary conditions of the model.

- The mix properties at the tank connection port. They depend on the flow sense at this point. Their values are the same as the values of the mix inside the tank if the stream flows outwards. If the stream flows inwards the tank, the mix properties at the connection should be assigned as boundary conditions.

It should be noticed that these aspects may change depending on the tank model reusing context and even depending on the experimentation conditions. Hence, PML should formulate the proper mathematical model to represent the properties of the three mix objects. This subject is discussed with detail at Section 5.2.2.

4.2.2. The Phenomenon PML class

The *phenomenon* class is used to express some type of physical behaviour that occurs in a system. Basically consists in the declaration of the attributes shown in Table 4.2. The `entities` attribute is used to define the entities classes involved by the physical phenomenon. The `law` attribute defines the laws which can be used to formulate the phenomenon and the `affected properties` attribute declares the affected entity properties. PML phenomenon class declaration should follow semantic rule S5.

Semantic rule S5 – *Phenomenon class declaration*

The modeling class *phenomenon* represents a physical phenomenon which occurs in a system. A phenomenon class is defined in terms of the involved entities, the laws used to describe the phenomenon and the entity properties affected by the phenomenon.

A phenomenon class declaration is context free since it makes reference just to a non empty collection of entity classes and to a non empty collection of law classes. No assumption on where a phenomenon class will be used to define behaviour can be made in its declaration. ❑

Phenomenon modeling class
entities: *entity
laws: *law
affected properties:*properties of entities

Table 4.2
UML description of the PML phenomenon class showing its attributes.

For example , let us consider a storing matter phenomenon. This phenomenon can be described by a matter balance and quantified in terms of the mass of the stored matter. In PML the following class can be declared to express this behaviour:

```
(phenomenon [store(matter)]
      [massBalance(matter)->mass(matter)]
);
```

where `massBalance(matter)` is the reference to the law class ruling this phenomenon and `mass` is the matter property affected by the phenomenon. The inclusion of the affected properties attribute in the phenomenon class declaration is demanded by the physical analysis performed on a PML model: it is necessary to know which and where the entity properties are affected by the different phenomena in order to assure the proper aggregated behaviour description in the functional model.

Multi-faceted behaviour declaration

In the phenomenon class declaration, several laws may be used to formulate the behaviour. This is one aspect of the multi-faceted nature supported by the PML models since any of the attached laws may be used according to the experimentation goal (this feature is discussed in more detail bellow).

The phenomenon establishes the relationship between what is happening in some system and how the behaviour can be described by a physics law. It is very important to note that a phenomenon is independent from any consideration about the system where the phenomenon occurs, so the phenomenon class declaration is also independent from any consideration about the model of the system. The reuse of a phenomenon class in different contexts can be ensured

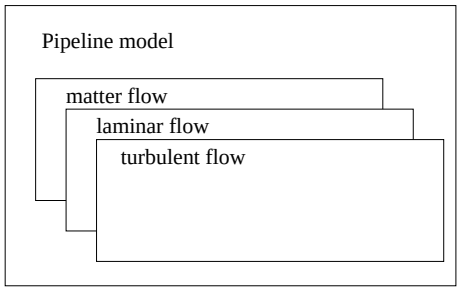


Figure 4.4. A model of a pipeline with multiple realizations with different mathematical descriptions of the transport phenomena

since its declaration can not make any assumption on its using context (see semantic rule S5). This means, for instance, that the `store(matter)` phenomenon can be reused to define the matter storage behaviour in any model where it makes sense. This feature is closely related with the definition of modularity given in Section 3.2.3 and will be discussed later in more detail in Section 4.5.1 .

Usually models have to answer to a number of different demands. Traditionally it has been considered not practical and difficult to develop a model able to solve many type of problems. This can be true if we think of mathematical models, since the complexity of the model increases as the set of problems to cover increases. The possibility of having models with multiple realizations has been pointed out in the object-oriented equation-based approaches (see for instance (Nilsson 1993)). The idea is illustrated at Figure 4.4 and basically consists in having different mathematical formulations (realizations) of the same behaviour declared in a model. One of these realizations can be selected depending on the problem to be solved by the model.

The possibility in PML to have different formulations of the same behaviour comes up in a natural way. As it has been introduced at the beginning of this section, a model defines which phenomena are represented. Since the law class defines how a phenomenon occurs, it is a matter of having different law class declarations associated to a phenomenon and then, it will be possible to select which is the appropriated law formulation to solve a particular problem.

SEMANTIC RULE S6 – *Multi-faceted phenomenon declaration*

A multi-faceted phenomenon can be ruled by different law formulations. Its class declaration consist in setting a collection of law classes in its law attribute. Any of the laws can be selected in a particular reusing context, provided the selection is compatible with the rest of declared behaviour. ❑

With this rule, the number of *realizations* of a phenomenon may grow arbitrarily without side-effects on the already defined behaviour. It is a simple question of declaring new law classes and establishing the new references to them by modifying the phenomenon class law attribute. For example, the following phenomenon class defines that the matter transport phenomenon can be described by three different law formulations:

```
(phenomenon [transport(matter)]
   [{matterFlow(matter)->massFlow(matter),
      laminarFlow(matter)->massFlow(matter),
      turbulentFlow(matter)->massFlow(matter)
   }]
);
```

The selection of the most proper law formulation is let to the model user (see Section 5.1.2). He may decide which one is the adequate behaviour formulation for his specific simulation purpose (see Definition 1.4). The multi-faceted phenomena will make possible to obtain different simulation models in a simple way. According to the experimentation purpose the appropriated law formulation can be selected. It should be also noticed that the entity property dynamics which are described (formulated) may depend on the selected law. It is clear that a physical phenomenon simply occurs and the system behaviour is affected. However, the system behaviour will by formulated in terms of the entity property dynamics, i.e., it is formulated by means of the physical property evolution over the time giving a measure of the system behaviour. Hence, when a new law class is attached to the phenomenon class, the affected property dynamics described by the law must be specified since the physical analysis procedure should be informed about the properties used to evaluate the phenomenon effects.

```
(phenomenon [store(ethanolSolution)]
  [{massBalance(ethanolSolution)->mass(ethanolSolution),
    massBalanceWithReaction(ethanolSolution)->mass(ethanolSolution)
  }]
);
```

Listing 4.1. PML class code of a multi-faceted storage phenomenon of an ethanol solution.

Relationships between phenomena

The way a phenomenon has to be formulated may depend on other phenomena declared in the model. For example, the storage phenomenon may be extended in order to describe the behaviour when a chemical mix (`ethanolSolution`) is used by declaring as multi-faceted the storage of an ethanol solution (the PML code is shown at Listing 4.1). Now, the `ethanolSolution` mass balance formulation should differ if the storage phenomenon class is reused in reservoir model or is reused in a reactor process unit coexisting with a chemical reaction phenomenon. The relationships between a particular phenomenon formulation with other phenomena are established in the law class (this feature is described in the section bellow).

4.2.3. The Law PML class

The *law* class is used to define the mathematical formulation of the law which rules a phenomenon. The law class is defined by the two attributes shown in Table 4.3 and its declaration should follow semantic rule S7.

Semantic rule S7 – *Law class declaration*

The modeling class *law* defines a particular mathematical formulation of the law which rules a physical phenomenon. The law class is defined in terms of the involved entities and the law formulation.

The law formulation is a PML sentence (see Appendix A) and may also include a reference to a set of phenomena required to postulate the law properly.

The law formulation can only make reference to the properties of the involved entities. The involved entities must match with the entities defined by the phenomenon the law is attached

Law modeling class
entities: *entity
formulation: law formulation

Table 4.3
UML description of the PML law class showing its attributes.

to or, in the case of referring other required phenomena, with the entities defined by such phenomena. If the referenced entity is multi-component, its components can be also referenced by the law. ❑

For example, the matter storage phenomenon may be described by a mass balance of matter. The following law class declares this behaviour:

```
(law [massBalance(matter)]
     [der(mass(matter))=massFlow(matter)]
);
```

In this formulation of the mass balance it is not specified how the `massFlow` matter property is defined. Usually, the law must refer to properties of entities which are exchanged between the model where the law is used and other models. This reference can not be predefined when the law class is declared since the context where it will be used is unknown.

SEMANTIC RULE S8 – *References to an undetermined number of properties in the law*
The law class formulation attribute can make reference to properties of entities declared in a class port by using the language sentence `portInstances`. This reference will be properly expanded in the model depending on the defined ports (see semantic rule S4 for entity access domain). ❑

The matter mass balance law class can be now declared to consider the mass flows through an undetermined number of ports which will affect the storage phenomenon (see Listing 4.2). When this law class becomes an instance in a model, the ports declared in the model are analyzed in order to determine which of them define a exchange of matter. Then the references to the `massFlow` property may be established at each port. Suppose that the law is used in a model

```
(law [massBalance(matter)]
     [der(mass(matter))=sum(portInstances(massFlow(matter)))]
);
```

Listing 4.2. PML law class describing a matter balance.

with two matter ports `mP1` and `mP2`. The following equation would be automatically generated:

```
massMatter' = mP1.massFlowMatter + mP2.massFlowMatter
```

PML equations (see Appendix A) can also be used as the argument in the `portInstances` sentence which should indicate . For example, the expression:

$$\sum_j \texttt{massFlow(ethanolSolution)} \times \texttt{concentration(water)}, \quad \forall j \in J \tag{4.1}$$

where J makes reference to an undetermined number of ports exchanging `ethanolSolution`, can be formulated by the following PML sentence:

```
sum({portInstances({ethanolSolution,
     prod({massFlow(ethanolSolution),concentration(water)})})
})
```

This expression will be expanded on each `ethanolSolution` port instance, once they can be determined. According to the entity property scope defined in the semantic rule S4, the `portInstances` sentence establishes the references to an undetermined number of ports exchanging the `ethanolSolution` entity.

The law can also refer to the involved entities components, whenever they exists, in the terms stated by next semantic rule:

SEMANTIC RULE S9 – *Entity components referenced by the law class*

The law class formulation attribute can make reference to properties of entities which are components of the involved entity. A component property can be referenced explicitly or by means of the expansible language sentence `componentInstances` in a n-termed math operator. ❑

For example, the following law formulation

```
(law [perfectMix(ethanolSolution)]
      [generatedMass(ethanolSolution) = sum({mass(water),mass(ethanol)})]
);
```

can be written in a more compact version

```
(law [perfectMix(ethanolSolution)]
     [generatedMass(ethanolSolution) =
        sum(mass(componentInstances(ethanolSolution)))]
);
```

As it has been mentioned in the previous section, a phenomenon formulation (law instance) may be related to other phenomena which must be declared in the model in order to be applicable. For example, the matter storage phenomenon has to be formulated in different ways whether a chemical reaction takes place or not. When the applicability of a law requires of other phenomena to be declared, it should be specified by means of the `PHENOMENON` sentence. For example, Listing 4.3 shows a law class describing the `ethanolSolution` balance in a system where a mixing phenomenon takes place.

With the `PHENOMENON` sentence it is guaranteed that the law is only used if the model declares the referred phenomena (the user can not select a law referring to phenomena not declared in the model). Additionally, this sentence helps to the physical analysis procedure to decide which is the most proper law to be used in the behaviour mathematical formulation: an instance of the law class at Listing 4.2 is used if the model just declares the ethanol solution storage phenomenon; an instance of the law class at Listing 4.3 is used when the mode declares the mix reaction phenomenon in addition to the storage phenomenon.

Relationships among phenomena do not necessarily have to be symmetric. In our example, the storage phenomenon has two different formulations depending on whether the mix reaction phenomenon is declared in the model or not. This is the reason why the previous law should include the `PHENOMENON` sentence. The opposite may not be true: the mix reaction phenomenon has a formulation which does not depend on the presence of other phenomena.

```
(law [massBalanceWithReaction(ethanolSolution)]
  [{mass(ethanolSolution) = generatedMass(ethanolSolution),
   der(mass(water))=sum({portInstances(massFlow(water)),
                         portInstances({ethanolSolution,
                                        prod({massFlow(ethanolSolution),
                                              concentration(water)
                                            })})
                        }),
   der(mass(ethanol))=sum({portInstances(massFlow(ethanol)),
                           portInstances({ethanolSolution,
                                          prod({massFlow(ethanolSolution),
                                                concentration(ethanol)
                                              })})
                          }),
   PHENOMENON(mixReaction(ethanolSolution))}]
);
```

Listing 4.3. PML law class describing the matter balance of an ethanol solution by considering a reusing context where a mixing reaction takes places.

4.3. Model construction

The model construction corresponds to the second layer in the PML modeling library construction (see Figure 4.1). At this layer, the basic physical knowledge required to define a model must be already declared, since the model definition basically consists in the specification of entities and phenomena. This means that a model class aggregates two predefined modeling classes: the phenomenon and the entity classes. In addition to this, two classes can be involved in the model definition: the port class and the model class itself. The ports are used to define the physical interactions between the system components and the model class defines the model components or submodels.

4.3.1. The Port PML class

The *port* class is used to represent the physical relationships between system components in terms of the exchange of a physical entity modeled with an entity class. The ports constitute the model physical interface. When a model declares a port, there is an implicit definition of an entity exchange phenomenon. Note that there is a straightforward analogy between the PML

Port modeling class
ent: entity
flowProperties:*entity property
effortProperties:*entity property

Table 4.4
UML description of the PML port class showing its attributes.

port class and the physical port of a system component.

Semantic rule S10 – *Port class declaration*

The modeling class `port` defines the exchange of an entity between interconnected models. The port class is defined in terms of two attributes: the exchanged entity and the properties which define the cause-effect relationships needed to describe the exchange phenomenon. ❑

The port class attributes are shown in Table 4.4. The flow and effort attributes have to be defined as couples describing the cause↔effect physical quantities which are used to represent the exchange phenomenon. For example, the mass flow or the volume flow are usually used to quantify the exchange phenomenon in a point of a process unit where matter is exchanged. Determining the mass or volume flow involves another physical quantity such as the pressure at the exchange point, which can be considered as its counterpart in the cause↔effect couple. These couples can be established for the different physical domains: voltage and current in electricity, force and position or velocity in mechanics, heat flow and temperature or specific enthalpy in thermodynamics, etc.

Semantic rule S11 – *Port cause-effect relationships*

The couples of flow-effort properties declared in a port must define a cause-effect relationship able to describe the entity exchange phenomenon represented by the port.

The physical quantities used to represent the exchange phenomena are circumscribed to the port, i.e., to the point where the exchange phenomenon takes place. ❑

For example, the following port class declares the exchange of matter entity:

```
(port [matterPort(matter)]
      [{massFlow,pressure}]
);
```

The `massFlow` and `pressure` properties are the *flow* and *effort* quantities respectively. These properties are interpreted as physical quantities of the entity at the port. Which means they are not carried out with the stream since they make sense within the port. The effort property must be physically related to the flow property as a cause-effect relationship.

An implicit assumption follows from the second assertion of the semantic rule S11: when two models are connected through a port, it should be possible to formulate the entity exchange phenomenon from the laws describing the phenomena declared at both models. For example, if a tank and a valve models are connected, it is reasonable to assume that the pressure at the connection point is determined from the storage phenomenon and the mass flow through the port is determined from the matter transport phenomenon occurring in the valve. If not, it would not be possible to formulate the matter exchange at the port. This assumption must be taken into account when models are connected through ports and can be generalized to an entity stream path defined by several interconnected models: it should be possible to set the cause↔effect relationship from the phenomena declared in those models.

The rest of the entity properties and the associated physical interactions are carried out through the port and, consequently, through the interconnected models. It is very important to remember that the physical interactions among submodels in an aggregated model are not, in general, local to the coupled models (connected through a port). As it was discussed in Section 3.2 (see Example 3.1), the physical interactions due to the entity streams in an aggregated model are global to the whole model. The physical analysis performed by PML to derive them uses the flow properties defined in a port to trace the paths followed by an entity among the submodels (see Chapter 5).

It is important to point out that the flow and effort properties attributes are needed to derive the mathematical formulation required for the simulation model. Analogous representation mechanisms of the model interaction are defined by the bonds in the Bond-Graph formalism and

by the terminal variables in the object oriented modeling environments based on the equation formalism. However, there are significant differences between the PML port and those mechanisms. As it was discussed in Section 3.4, in Bond-Graph every model interaction is based on a bond, i.e., on a flow-effort couple, leading to a model structure which is usually closer to our functional model than to the system topology (our topological model). In the equation-based object oriented modeling languages, the model interface must include all the variables shared by the model and its environment. Hence, when the model interface is defined, we have to think in terms of the local variables which can not be solved from the equations declared in the model. Thus, their solution will be set when the model becomes a component in an aggregated model. It is more a mathematical problem than a physical problem.

The PML port establishes a direct analogy with the system connection ports since it defines the exchange of the entity in the process (matter or energy). Hence, the port assumes an exchange phenomenon and, in the same way as a PML phenomenon class sets the properties affected by the physical phenomenon, the port sets the properties involved directly in the exchange phenomenon. The flow and effort attributes play two roles: the representation of the exchange phenomenon as a cause↔effect relationship between two physical quantities and the representation of the exchange direction by means of the flow property sign.

When a system consists of a set of interconnected components, entities (matter or energy) are exchanged between them. Hence, there are properties carried out with the entity streams and it is necessary to know the direction of the streams in order to establish the physical interactions between the system components. These physical interactions between models are derived by the *Physical analysis procedure* (introduced in the previous chapter). The stream direction is mathematically represented by the sign of the properties declared in the `flowProperties` attribute. The sign of the flow property (e.g. `massFlow`) determines the direction of the entity exchange (e.g. `matter`). The second important aspect of the flow property sign is its contribution to an entity balance (e.g., the matter balance in the tank). The sign of the `massFlow` property is used to know if the matter is going into the tank or if its flowing out from the tank. Hence, the `massFlow` property sign in a port of a tank model will determine if the mass flow adds up

Model modeling class
phenomena: *phenomenon
entities: *entity
ports: *port
submodels: *model
connection topology: *PML connection sentence
local behaviour: PML sentence

Table 4.5
UML description of the PML model class showing its attributes.

or subtracts in the mass balance. Nowadays, PML does not generate a flat simulation model, but generates a model according to an equation-based object oriented language (EcosimPro in current version). The `flowProperties` and `effortProperties` attributes are used to define the *through* or *flow* and the *across* variables (Cellier 1991) of the mathematical terminals required by these languages. The connection equations derived from these model terminal variables propagate the sign of a flow property between interconnected models.

4.3.2. The Model PML class

The *model* class is used to define a system component or a part of system component. For instance, a tank model may represent a stand-alone reservoir system or may become the vessel of a chemical reactor. The model class can be considered as an aggregation of entity, phenomena and port classes (see Figure 4.2). The model class is characterized by the attributes shown in Table 4.5.

Semantic rule S12 – *Model class declaration*

The *model* class is used to represent a system (physical device) or a part of a system. The model class may contain the following three sections:

- *Physical behaviour.* It defines the behaviour of the physical system that the model will represent. Its declaration is made by specifying the involved phenomena classes.

- *Topology.* This section allows the definition of *structured models* by specifying its submodels

and the topology of submodel connections. This topology is supposed to match the system components topology of connections.

- *Local behaviour.* This section is used to represent aspects which are particular to the represented system as, for instance, geometrical attributes of the system. A PML sentence must be used and references to the properties of the entities represented in the model, if any, can be made.

❑

The model class is a container where both the phenomena occurring and the characteristics of the physical component are represented. The model class captures the essential behaviour of a system by specifying the phenomena occurring in the system, the topological structure of the system or the equations which are particular to the represented system (e.g., the relation between volume and level in a tank). For example, the model of a tank where some matter is stored can be defined by the following model class declaration:

```
(model [tank]
       [phenomena(store(matter))]
);
```

It is possible to declare a model class, as the previous one, without specifying any entity.

Semantic rule S13 – *Abstract Model class declaration*

If the entity attribute is not defined, the model class is said to be an *abstract model* and no class instance can be created. ❑

The abstract model class is analogous to the programming abstract classes. They are useful to organize the hierarchy of model classes in order to increase the maintainability of the model library.

The structure of a PML model class

Two types of models may be distinguished, the *atomic* model which represents an isolated system component (e.g. a valve or a pump) and the *composite* or *aggregated* model type whose

structure represents a collection of interconnected submodels which in turn can be atomic or composite models (e.g., a chemical reactor model composed by a vessel and a jacket models).

SEMANTIC RULE S14 – *Atomic model class*

A model class defines an *atomic* model if it has an empty topology section. ❑

The *aggregation* facility gives a powerful mechanism for reusing models. Complex models can be built in a bottom-up sense by specifying submodel connection topology. It is important to note here that, since the PML model interface represents the exchange of a physical entity, the topology of an aggregated or composite model will preserve the analogy with the component connection topology of the system being modeled.

SEMANTIC RULE S15 – *Structured model class*

A model class defines a *structured* model if it has a non empty topology section. The topology section declares the submodels and their connection topology. When a predefined PML model class is used as a submodel, it becomes an attribute of the structured model class named as model instance (or model object). ❑

Submodels are connected by means of the `connections` attribute of the PML model class (for syntax see Appendix A). The model coupling through port connections must accomplish with the semantic rule S18.

4.4. PML modeling libraries

The PML modeling classes are organized in libraries supported by plain text files. In the current version of the PML modeling environment there is not any restriction in the order in which the classes are defined. The following example, filling the rest of this section, illustrates the construction of a modeling library where the physical knowledge required to model very simple matter storage and transport components is represented.

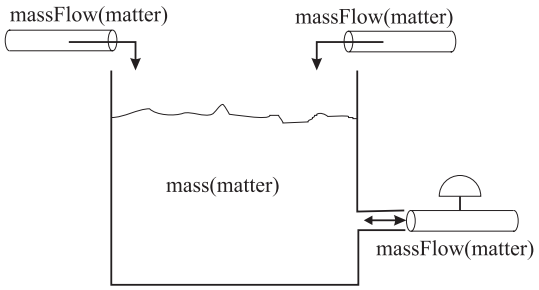


Figure 4.5. A tank process unit

Matter entity class
properties: mass, massFlow, pressure, pressureDrop
components: none

Table 4.6. Matter class attributes.

EXAMPLE 4.1 – *Modelling of a simple process unit*

The model of the system shown in Figure 4.5 will illustrate the two first layers in PML model construction. A modeling library for representing the matter balance behaviour will be developed. The definition of the PML modeling classes will be guided through this simple system. However, it will be shown in forward examples how this library may grow to afford the modeling of more complex systems with minimum, even null, side effects on the already defined classes.

First layer: basic knowledge definition.

In the first layer the entity, phenomenon and law classes are declared. The definition of this basic physical knowledge is independent from any system component model. For instance, the matter storing phenomenon is general and may be reused in any component model. Thus, classes representing entities, phenomena and laws can be predefined independently of their reusing context.

First, the matter entity class is defined. The attributes required to represent the dynamics involved in the example are shown in Table 4.6. The PML code of this class is shown at Listing 4.4. Any other property associated with the matter may be aggregated in the future if new dynamics, not considered at the moment, become necessary. For reasons of simplicity, only these four have been specified, since they are enough for the example purpose.

Two phenomena are needed to represent the physical behaviour: the matter storage in the tanks and the matter transport in the ducts. The matter storage and transport phenomenon class attributes are defined at Tables 4.7 and 4.8.

Store phenomenon class
entities: matter
laws: massBalance
affected properties:mass

Table 4.7
UML description of the store phenomenon class.

Transport phenomenon class
entities: matter
laws: matterTransport
affected properties:massFlow,pressureDrop

Table 4.8
UML description of the transport phenomenon class.

The PML code of these classes is shown at Listing 4.4. It should be noticed that these phenomenon classes do not express explicitly how the phenomena may be computed or calculated. They are needed because a PML model represents the physical behaviour as an aggregation of the phenomena happening in the system. The way in which the behaviour has to be computed (mathematically formulated) depends on the whole physical context of the model and its knowledge is not necessary to enunciate the phenomena.

Another relevant aspect can be observed at the transport phenomenon class. Both the `massFlow` and `pressureDrop` have been included in the affected properties attribute since there is a cause-effect physical relationship between them. The knowledge of cause-effect relationships in a phenomenon can be very useful to obtain efficient simulation models (Ramos *et al.* 1998*a*).

It should be noticed that, up to this point, only physical concepts have been used to build the modeling classes. It is still the same when declaring the laws, since only considerations about the relationship between the properties involved in the phenomenon are made. The mass balance law class attributes define the matter balance law are shown in Table 4.9.

The `massBalance` law declaration makes reference to an arbitrary number of mass flows with the language operator `portInstances`. The number of ports exchanging matter (involved `massFlow` properties) will be determined in the models where this law is used. The matter transport class attributes are shown in Table 4.10.

Listing 4.4 shows the modeling library where the entity, phenomena and laws required to model the matter balance in the tank system of Figure 4.5 are defined.

massBalance law class
entities: matter
formulation: $mass(matter)' =$ $\Sigma\ portInstances(massFlow(matter))$

Table 4.9
UML description of the balance law class.

matterTransport law class
entities: matter
formulation: `massFlow(matter)` $=$ $Rfluid * \sqrt{pressureDrop(matter)}$

Table 4.10
UML description of the transport law class.

```
(entity [matter]
  [properties({mass,massFlow,pressure,pressureDrop})] );
(phenomenon [store(matter)]
   [massBalance(matter)->mass(matter)]);
(phenomenon [transport(matter)]
   [matterTransport(matter)->{massFlow(matter),pressureDrop(matter)}]);
(law [massBalance(matter)]
    [mass(matter) = intgr(sum(portInstances(massFlow(matter))))]);
(law [matterTransport(matter)]
    [{DATA(Rfluid = 1.0),
      massFlow(matter) = prod({Rfluid,sqrt(pressureDrop(matter))}),
      sum(portInstances(massFlow(matter))) = 0.0}]);
```

Listing 4.4. PML classes representing the basic physical knowledge.

Second layer: model construction.

In this second layer of the PML modeling library construction, the port and model classes are defined. The port class defines the interaction between models as the exchange of some entity (usually some type matter or energy). The exchange of the matter entity can be defined with the port class attributes shown at Table 4.11. The PML code of the port class is shown at Listing 4.5. The `massFlow` and `pressureDrop` matter properties are defined as the cause↔effect relationships in the port (see the semantic rule S11). This declaration is needed since the matter exchange direction, determined by the flow property sign, must be known in order to analyze the physical interactions derived from the exchange of an entity between models. The sign determines the influence over a balance equation of the flow property. A simple case can be observed in the example being developed (see Figure 4.5). The `massFlow` property sign at the

matter port class
entity: matter
flowProperties: massFlow
effortProperties: pressure

Table 4.11
UML description of the matter port class.

```
(port [matterPort(matter)]
   [{massFlow,pressure}
]);
```

Listing 4.5.
PML code of the matter port.

tank ports will determine the matter mass dynamics. Note that all the `massFlow` at the port instance are summed up in the mass balance law (see Listing 4.4). Hence, the matter mass in the tank would decrease because of the matter leaving the tank (`massFlow` negative sign) and would increase when the flow sense reverses (`massFlow` positive sign).

Furthermore, it is easy to realize that there will be many physical interactions between models that will depend on the entity stream direction. In such physical relationships, it will be necessary to know the entity flow direction. Note that at each connection point the flow changes from *out* to *in* (e.g. flows out from the tank and flows into the valve). Hence, a signed property must be used in order to represent these changes. This is also a reason to declare the `massFlow` property as a flow property in the port. Its sign will be used in the mathematical model as the matter flow direction indicator. In order to consider the outflow-inflow changes, the flow properties will be declared as *flow* or *through* variables in the mathematical model terminals generated from the PML models (e.g. see the EcosimPro code generated by PMT at Section C.1).

Resuming now the library construction, the model classes will be defined. The PML atomic model classes will represent the basic units (equipment) of the system. In a PML model class it is defined the component interaction with its environment, the geometric aspects of the component, which phenomena take place and the involved entities. For the specified modeling purposes, the tank model should represent a matter balance. The tank model class attributes are defined in Table 4.12.

The PML code of the tank model is shown at Listing 4.6. This model class defines a matter balance behaviour by aggregating the store and matter predefined classes. The port

tank model class
phenomena: store(matter)
entities: matter
ports: matterPort(p1),
matterPort(p2),
matterPort(p3)
submodels: none
connection topology: none
local behaviour: *PML sentence

Table 4.12
UML description of the tank model class.

```
(model [tank]
   [{ entities({matter}),
      ports({
        matterPort(P1(matter)),
        matterPort(P2(matter)),
        matterPort(P3(matter))}),
      phenomena(store(matter)),
      equations
      ({DATA({section=1.0,pTop=1.0,g=9.8}),
        mass(matter)=prod({volume(matter),
                           density(matter)}),
        volume(matter)=prod({section,level}),
        P3(pressure(matter))=
           sum({pTop,
                prod({density(matter),
                      g,level})}),
        P1(pressure(matter))=pTop,
        P2(pressure(matter))=pTop})
   }]
);
```

Listing 4.6.
PML code of the tank model.

attributes constitute the model class interface to other model classes. This model physical interface, which is analogous to the physical system interface, is used to analyze the physical interactions with other models once the model becomes a component in a larger structured model. This question is discussed at Section 5.2. The only, so to say, mathematical face of a model class is the PML sentence used to declare behaviour local to the model. In the tank class the local behaviour expresses the relationships between matter properties and the geometrical attributes of the tank such as volume and level (see the code at Listing 4.6). The rest of the behaviour mathematical formulation is obtained from the physical knowledge represented by the phenomena. The phenomenon class definition establishes the link between the phenomenon and the law used to describe it. In the tank the mass balance rules the storing matter phenomenon. This relationship is modeled as:

```
(phenomenon [store(matter)]
   [massBalance(matter)->mass(matter)]
);
```

duct class
phenomena: matterTransport(matter)
entities: matter
ports: matterPort(T1),matterPort(T2)
submodels: none
connection topology: none
local behaviour: PML sentence

Table 4.13
UML description of the duct model class.

```
(model [duct]
   [{
    phenomena(transport(matter)),
    entities(matter),
    ports({matterPort(P1(matter)),
           matterPort(P2(matter))}),
    equations({
      DATA({ductVolume=5.0}),
      pressureDrop(matter) =
        sum({P1(pressure(matter)),
            -P2(pressure(matter))}),
      volume(matter) = ductVolume
    })
  }]
);
```

Listing 4.7.
PML model of the duct unit.

The law class describes how the matter mass property affected by the storage phenomenon can be quantified (see Listing 4.4). The `massBalance` law instance is created to represent the effects of the phenomenon declared in a model and it is used to formulate the simulation model.

The other basic unit to be modeled in this example is the duct. Its model class attributes are represented at Table 4.13. The PML code of the duct model is shown at Listing 4.7. This class aggregates those phenomena and entities which declare the behaviour that the model is going to represent: the matter transport. The local behaviour here sets the direction of the flow since the duct system establishes how the cause-effect relationship in a transport phenomenon takes place.

The structured model of the system in Figure 4.5 is defined by specifying the submodel components and their connection topology. This model class illustrates the concept of structured models. Its class attributes are defined at Table 4.14. Listing 4.8 shows the `process` PML model class code. Note that the model topology is a replication of the system topology, where `dE`, `dW` and `v` are the names given to the instances of the `duct` model class and `mTA` is the name given to the `tank` instance. The connection sentence indicates which are the matter port couplings, e.g. `dE.P2-mTA.P1` sets the connection of the duct port `P2` to the tank port `P1`.

process class
phenomena: none
entities: none
ports: none
submodels: mTA:tank, dE,dW,v:duct
connection topology: dE,dW,v ducts to mTA tank
local behaviour: none

Table 4.14
Process model class UML description.

```
(model [process]
   [{submodels({
        duct({dE,dW,v})
        tank(mTA)}),
     connections({
        dE.P2-mTA.P1,dW.P2-mTA.P1,
        v.P1-mTA.P3})
    }]
);
```

Listing 4.8.
PML model of the tank system.

Figure 4.6 shows the modeling object diagram. This diagram represents the relationships between the modeling class instances created to formulate the mathematical model required for simulation. Two type of relationships are distinguished:

- Explicit relations defined by the model. The `process` model is composed by four submodels, three are instances of the `duct` model class and one is an instance of the `tank` model class. Every submodel uses matter. The tank instance represents a storing phenomenon and the duct instances represent a transport phenomenon.
- Relations derived from the physical analysis procedure (see Section 5.2). They consist basically in the behaviour formulation by using the $phenomenon + entity \rightarrow law$ relationship. For instance, in the tank model it is inferred that the phenomenon store happening with matter is described by a mass balance law instance.

The main task of the physical analysis procedure is to formulate a mathematical model according to the reusing context of the model and its submodels. The reusing context is characterized by two main aspects: the entities involved in the phenomena represented at the model and the topology of the submodel connections. The mathematical model generated by PMT is shown at Section C.1 in appendix C. ❑

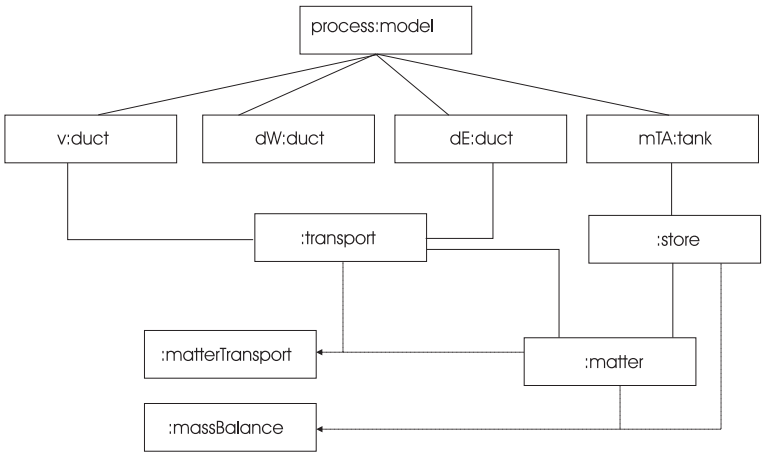


Figure 4.6. Modeling object diagram of the tanks model. Lines indicate the modeling class instances aggregated by the model definition. Arrows indicate the law class instances created by the physical analysis procedure.

4.5. PML Object-Oriented properties

This section is devoted to illustrate how PML accomplishes with the main object oriented characteristics already introduced in Section 2.3.2.

4.5.1. Classification

The construction of an application in an object oriented approach is performed around the structure named class. The class plays two important roles: the first one is that the class structure becomes a type definition mechanism by capturing and representing the properties of similar objects; the second one is that the class constitutes an organization mechanism since every object definition in the application should be based on a class. The PML class structure fulfills the following rules:

Semantic rule S16 – *Modularity*

The PML classes are modular units since object orientation is primarily an architectural technique whose main contribution is the modular construction of systems. This basically means that every definition must remain local to the class declaration or make a reference to the class interface. ❑

SEMANTIC RULE S17 – *Typing*

The PML class is the type definition mechanism. The representation of every object in the application domain should be based on a PML class. ❑

As it has been presented in the previous sections, PML articulates the representation of the physical knowledge in terms of five main classes (model, entity, phenomenon, law and port classes). These classes can be considered as *super-classes* since every new modeling class must be derived (inherited) from one of these classes. Hence the *typing* rule is fulfilled. The preservation of modularity in the representation of the physical behaviour should be considered under the point of view of modularity that was given in Section 3.2. The following five points will illustrate how the PML modeling classes preserve the modularity design principle:

- **The entity class**. Every reference to a physical property must be made to an attribute of the entity class. Hence, the entity class must be previously defined. Its definition is context independent. When an entity class is defined, the only physical question to be taken into account is the representation of the physical quantities which characterize the entity. Any later modification, e.g. a new property definition, does not have side-effects on the rest of defined modeling classes (provided that the modification does not eliminate attributes referred by other modeling classes already defined). Entity classes can be combined in two senses to build new behaviour representation: with other entity classes to define a new class (multi-component entities) or aggregated in a model and related to phenomena to define the model physical behaviour.

- **The phenomenon class**. What happens in a system depends on the phenomenon itself and not on the way the phenomenon is formulated for simulation purposes. For example, in a tank some matter is stored and is described by a matter mass balance considering the matter exchanges through ports. If a chemical reaction is undergoing in the tank, a term representing the reacted matter must be added to the matter balance equation (as well as the formulation of the chemical reaction). So, what changes in both cases is the way in which the matter balance is formulated to describe the storage phenomenon, which is still

the same. That is the reason why in PML the phenomenon and the law are represented by different classes. The phenomenon modeling class definition is context independent since it only represents some physical behaviour without considering its formulation (which is context dependent).

- **The law class**. A law class is complementary to a phenomenon class. The definition of this class is context independent, even the use of a law instance to formulate the behaviour of a model is context dependent. To support a free context definition, the language offers the possibility to make reference into the undetermined aspects related to the reusing context (see the semantic rule S8). For instance, a law class may ask the model how many ports declaring the exchange of the involved entity has (see the use of `portInstances` in the law class at Listing 4.4). The reusing context of a law instance is set during the physical analysis procedure (see Section 5.2).

- **The port class**. This is the class with the simpler structure, however it plays a very important role since it defines the model class physical interface. Nevertheless, when the port classes are defined it is not necessary to know any feature of the model class where it will be used (aggregated). In this sense, its definition is context independent.

- **The model class**. The PML model class structure also accomplish with the modularity requirements established in Section 3.2. The model represents those phenomena which are particular to the modeled system component. As it has been discussed in Example 3.1 at Section 3.2, the mathematical model of a system component can not be predefined in a modular way since, in general, there will be coupling phenomena which are unknown without considering all the system components and the connection topology. With PML the model class modular definition is possible because the representation of the phenomena and their mathematical formulation have been separated. The physical interactions with other models are represented by means of the port structure which gives the mechanism to derive the proper mathematical formulation of the coupled behaviour. Section 5.2 will come back to this important contribution of PML.

Communication among PML modeling classes

Because of the modular nature of classes, the primary communication mechanism among objects is: giving a certain object, which (because of rules S16 and S17) is an instance of some PML class, uses a feature of some class on that object. Within the system modeling context, the *communication protocol* between objects should be used to obtain a suitable mathematical formulation of the physical behaviour (state-space or DAE simulation model). In PML, the communication depends on the involved class types. In addition to the inheritance mechanism to exchange information between classes, PML introduces two more rules to communicate classes:

Semantic rule S18 – *Communication between model classes*

The model-to-model communication is described by means of the port class. The connection of two models through their ports establishes the physical relationships between the coupled models required to formulate the whole aggregated model behaviour in terms of the exchange of an entity. Two connected ports must define the same entity class. ❑

This communication protocol among models preserves the analogy with the physical unit connections since the PML port class defines the exchange of an entity between two models. This feature increases the model reusability since in the connection mechanism there is not any reference to the phenomena represented in the models and, therefore, there is not any dependence between the model interface and the behaviour represented by the model.

It should be remarked here that, as a consequence of the semantic rule S11 of the current PML language specification, it should be possible to set the cause↔effect relationship in an entity stream path from the phenomena declared in the interconnected models. Consider the following port class declaration:

```
(port [matterPort(matter)]
      [{massFlow,pressure,
        volumetricFlow,pressure}]
);
```

The physical interpretation of this port establishes that the exchange phenomenon can be described both with the mass or volumetric as the flow property and with the pressure as the

effort property. However, from semantic rule S11 it follows that either the mass flow or the volumetric flow are required by the models involved in the stream path. In other words, if a tank connected to a pump asks for the mass flow at the port and the pump sets the volumetric flow, it will not be possible to generate a valid simulation model, despite the semantic rule S18 makes possible the connection of these models.

This restriction will be relaxed in future versions by introducing the facility to describe relationships among properties in the entity class declaration. For instance:

```
(entity [matter]
        [{properties({mass,density,massFlow,volumetricFlow,...}),
          constraint({massFlow=prod({density,volumetricFlow})})
        }]
);
```

The second important association of PML modeling classes is the communication between the phenomenon, entity and law. The protocol is governed by the following rule:

SEMANTIC RULE S19 – *Communication between phenomenon, entity and law classes*
When a model declares a phenomenon and an entity, the relationship between the phenomenon and the entity is used to make the instance of the proper law class giving the mathematical formulation of the represented behaviour. ❑

This rule makes possible the dynamical mathematical formulation of the behaviour declared in the model in terms of the aggregation of phenomena and entity classes. This feature is discussed in detail in Section 4.5.7 bellow. The combination of the communication rules S18 and S19 are used by the physical analysis procedure in order to build the PML functional model.

4.5.2. Genericity

The concept of *generic* classes is closely related to the flexibility and extendibility of the object oriented design (these quality factors will be discussed in the next section). An object oriented language should support the definition parameterized classes, also known as generic. A generic class defines some functionality but does not predefine which objects are manipulated by

the class since they are specified as the parameter to the class. A typical example in programming could be a class Vector defined to store any type of object (e.g. an integer basic type or a user defined object).

The concept of generic classes has been adopted and adapted in PML model classes. Any model class, except an abstract class, may be considered generic. There is a simple physical reason to support this idea. For instance, a tank may contain any product, at least any liquid or gas. Therefore, a tank model should be prepared to contain any liquid or gas matter entity. A PML model class is a generic class from the entity representation point of view since the PML model class is an aggregation of entity classes . Nevertheless, it is not as simple as this definition may seem. A model class is also an aggregation of phenomena, but it is not a generic class from the phenomena point of view. The characteristic of genericity is related to the capacity to replace a model class attribute without affecting the behaviour defined by the model. This is the reason for not supporting the substitution of phenomena, since it would change the represented behaviour. On the contrary, an entity may be replaced without affecting the represented behaviour. There is only one rule that must be observed when replacing entities:

Semantic rule S20 – *Generic model class*

Any non abstract model class may be considered an *entity-generic* class. An entity-generic model class can be reused with an arbitrary number of entities.

An entity-generic model class may be reused with any entity if, and only if, the new entity is an heir of the entity declared in the model class. ❑

The following example will illustrate this PML characteristic, which gives to the model reusability a great strength since a model can be designed without caring about the entities used to describe each particular reusing context, provided that the semantic rule S20 is fulfilled.

Example 4.2 – *Reuse of a model with new entities.*

This example illustrates the PML capability to support generic model classes with respect to the entity class. Let us consider the tank model class representing a matter storage phenomenon

```
(law [massBalance(ethanolSolution)]
    [{
      der(mass(ethanolSolution)) =
          sum(portInstances(massFlow(ethanolSolution))),
      der(moles(water)) =
          sum(portInstances({ethanolSolution,
                             prod({molarFlow(ethanolSolution),
                                   molarFraction(water)})
                            })
          ),
      der(moles(ethanol)) =
          sum(portInstances({ethanolSolution,
                             prod({molarFlow(ethanolSolution),
                                   molarFraction(ethanol)})
                            })
          )
    }]
);
```

Listing 4.9. New PML law class added to the modeling library developed in Example 4.1.

in two different reusing contexts where the entity is changed (the PML tank model class was developed in Example 4.1 and the code is shown at Listing 4.6).

This model class may be reused to contain different entities provided they inherit from the matter entity (see Section 4.6 for the explanation of model parameterization). For example, water and the ethanol solution can be used to generate two different simulation models:

```
> analyze tank(tk1(entities(matter=water)))
> analyze tank(tk2(entities(matter=ethanolSolution)))
```

It is important to note that this is not a simple change of the names given to the entities. The phemenon+entity→law communication rule S19 will lead to different `tk1` and `tk2` mathematical models provided that there is a law class specific for the storage of the ethanol solution. For example, let us consider a law class is declared to describe the following molar balances of the

solution components (the class code is shown at Listing 4.9):

$$n_E = \int_0^t \sum_i nF_{ES} * x_E$$
$$n_W = \int_0^t \sum_i nF_{ES} * x_W$$

where n_j is the number of moles of each substance, x_j is the molar fraction, nF_{ES} is the molar flow of the ethanol solution and i indexes the ethanol solution ports of the model where the law instance is created.

By specifying different entities (`water` and `ethanolSolution`) with the same tank model, two different mathematical formulations of the behaviour described in the model are obtained according to the law classes defined in the library. ❑

Note that the tank model behaviour does not change with the two different entity parameters. Still a matter storage phenomenon is represented in both cases. The important change is the mathematical formulation of both tanks which differs because a specific law is defined for the ethanol solution.

The capacity of PML to use the proper law formulation makes possible to have a complete separation between the entity as a representation of matter or energy and the model as a representation of a physical device. We will come back to this question in Section 4.6.

4.5.3. Information hiding

In class definition two different facets should be distinguishable in the represented objects: the part which is accessible to other objects and the part which remains hidden and whose implementation details are not relevant to the rest of the interacting objects. This rule is known as *information hiding* and makes the class modular structure to be *encapsulated*.

A main consequence of this rule is that communication between objects should be strictly limited to the class interface. In the software domain, classes will exchange information exclusively through feature calls (message passing) and through the inheritance mechanism. In the modeling domain, because of the requirements imposed by the simulation application (see Section 5.2),

the communication between the modeling classes need a much more sophisticated manipulation. The two communication rules S18 and S19 defined in PML make this manipulation possible. The model class port interface is the mechanism used to describe the communication between models, by considering that model the communication usually represents physical relationships. Any interaction between model classes is analyzed through the port connections, hence the port encapsulates the physical behaviour represented in the model.

The physical behaviour section of a model (phenomena and entities) is used to set the proper law formulation (use of the adequate law class instance) during the physical analysis procedure of the model. The selection of the proper law class can be hidden to the model user, except in the case of multi-faceted models where the user has to select the law class according to his experimentation interest. Hence, it is possible to consider the phenomenon-entity class association as the encapsulation mechanism of the behaviour mathematical representation.

4.5.4. Inheritance

Inheritance is a powerful mechanism for sharing similarities among classes while preserving their differences. A class will be an heir of another if it incorporates the other's features in addition to its own.

Semantic rule S21 – *Class inheritance*

Every PML class is defined as an heir of another class known as a *super class*. Single inheritance is supported and every class has as ancestor one of the five basic modeling classes. ❑

The inheritance mechanism is usually used to define more specific classes. In the following example, a mixing tank model class refines a more generic tank model class by adding new behaviour describing the mix of ethanol and water.

Example 4.3 – *Model of a mix reactor*

Let us consider the model of a process unit where a mix of ethanol and water takes places giving an ethanol solution. The best way to represent this type of entities is using the multi-component entities such as the `ethanolSolution` entity class already defined in Section 4.2.1.

```
(phenomenon [mixReaction(ethanolSolution)]
   [perfectMix(ethanolSolution)->{generatedMass(ethanolSolution),
                                  concentration(componentInstances(ethanolSolution))}]
);
(law [perfectMix(ethanolSolution)]
    [{
    generatedMass(ethanolSolution) = sum(mass(componentInstances(ethanolSolution))),
    concentration(water)=div({mass(water),mass(ethanolSolution)}),
    concentration(ethanol)=div({mass(ethanol),mass(ethanolSolution)}),
    }]
);
(tank [mixTank]
   [{
     ports({
        matterPort(P1(ethanol)),
        matterPort(P2(water)),
        matterPort(P3(ethanolSolution))}),
     entities(matter=ethanolSolution),
     phenomena(mixReaction(ethanolSolution))
   }]
);
```

Listing 4.10. PML modeling classes to represent the behaviour of a mix reactor process unit

The tank representing a matter storage which was defined in Example 4.1 (see Listing 4.6, will be now specialized by means of inheritance in order to represent a process unit where two substances are mixed. New modeling classes representing the basic behaviour (the mix reaction) should be added to the modeling library. The new modeling classes are shown at Listing 4.10. As it can be seen, just the `mixReaction` phenomenon and the `perfectMix` law classes must be declared to model the mixing dynamics. The `mixTank` model class is declared as a heir of the `tank` class. The `matter` entity is redefined by the `ethanolSolution`, the `mixReaction` phenomenon is aggregated and the proper entity exchanges at ports are specified.

Due to the multi-faceted declaration of the `store(ethanolSolution)` phenomenon (see Listing 4.1) and to the relationship between the store and mixing reaction phenomena, an instance of the `massBalanceWithReaction` law class will be created (see Listing 4.3) when the mathe-

matical model is generated as a consequence of the inclusion of the mixing phenomenon in the `mixTank` model. ❑

In the language current specification, only entity and model class inheritance is supported. The phenomenon class inheritance will be implemented in future versions. The advantages derived from phenomena inheritance are discussed in Section 4.6 bellow. The law and port classes are defined as final since no heir of them can be declared. The model and entity class inheritance must be guided by the two following rules.

Semantic rule S22 – *Model class inheritance*

A model class B can be declared as an heir of class A, establishing a *"B is a A"* relationship, with the following type of valid specializations:

- *Phenomena*: class B inherits the phenomena defined by the class A. In addition, new phenomena can be declared to represent new physical behaviour.
- *Entities*: the class B inherits the entities defined by the class A. In addition, new entities can be declared.
- *Ports*: class B inherits the ports defined by class A. In addition, new ports can be declared.
- *Submodels and connections*: class B inherits the topological section defined by class A. If class B defines a new topological section then, the topological section of A is overridden with the exception of submodel redefinition (see semantic rule S24).
- *Local behaviour*: class B inherits the equation section defined by class A. If class B defines a new equation section then, the equation section of A is overloaded.

❑

By using programming terminology, the model basic behaviour (entities and phenomena) in addition to the model to model interface (ports) can be overloaded by means of the inheritance mechanism.

However, attention must be paid when using this mechanism. The addition of new physical behaviour (phenomena and entities) should not cause the invalidity of the inherited behaviour. For instance, it is meaningless to say that a `tankHeir` declaring a matter transport phenomenon is a `tank` declaring a matter storage phenomenon. This type of inconsistencies can not be controlled by the language, so it is a matter left to the modeller. Nevertheless, it should be possible to specialize a model by adding new coherent phenomena. A similar reason justifies the addition of new entities in a inherited model class. Actually, both phenomena and entities constitute the basic physical behaviour represented by the model, so the specialization of the physical behaviour represented by the model must be supported in PML by means of inheritance.

The model port section defines its interface. Since the port defines the exchange of entities, the model class interface must be adapted as new entities are declared. The validity of new port by means of inheritance can be controlled by the language since a port in a model has to make reference to one of the entities declared in the model.

On the contrary, the topology section can only be inherited as it is defined by the superclass or redefined in other case (with the exception of semantic rule S24). This is for model consistency reasons since the inheritance mechanism may preserve consistency as far as possible. For instance, if an heir class modifies a connection of the topology defined by its super class, which is considered to be a valid model, the validity of the heir model can not be guaranteed.

SEMANTIC RULE S23 – *Entity class inheritance*

An entity class B can be declared as an heir of the entity class A, establishing a *"B is a A"* relationship, with the following type of valid specializations:

- *Properties*: class B inherits the properties defined by class A. In addition, new properties can be declared.
- *Components*: class B inherits the components defined by class A. No new components can be added or redefined in class B.

❑

Because of the entity propagation semantic rule S26 (see Section 4.6), if a phenomenon F is valid for an entity E and the entity G inherits from E, the phenomenon F is also valid for G. Therefore, this rule could not be applied if the components of an entity could be changed by inheritance. For example, consider that the phenomenon `binaryReaction` is valid for some entity E with two components and a new entity G is declared to be heir of E. It is obvious that the addition of new components in G by inheritance can not be allowed. Otherwise, the entity propagation rule S26 would lead to an invalid model.

4.5.5. Polymorphism.

Within the system modeling context the polymorphism characteristic defines the ability for a modeling class to be substituted by another class without affecting the consistency of the represented behaviour. The polymorphism is closely related with the capacity of redefinition to be discussed in the next subsection and with genericity characteristic.

In the equation based languages the polymorphism is related to the model class. Two models are polymorphic if the following demands are satisfied (see for instance (Nilsson 1993)):

- Terminals of polymorphic models must be compatible. This means that terminals must declare the same connection variables.
- Parameters whose values are assigned by the super model must be identical.
- The degree of freedom of polymorphic models must be the same. The degree of freedom is the difference between the number of variables (including terminal variables and parameters) and the number of constraints (equations and parameter assignments).

Different arguments are used in PML since the behaviour representation is not based on equations. The PML language introduces the following demands on classes to be polymorphic:

Definition 4.1 – *Polymorphic PML model classes*

In the PML language, polymorphism is related to the model class interface and to the inheritance model class hierarchy:

- Polymorphic model classes are those classes which have the same model interface. This means that they must have the same model ports and operate on the same entities.

- Polymorphic model classes must have a common ancestor different from the model base class.

❑

The polymorphism check is under the inheritance control since it is necessary to guarantee that the model classes have declared a common physical behaviour to be polymorphic.

In PML, the polymorphism characteristic is also extended to the entity class. If a phenomenon is defined on an entity, for example store matter, the phenomenon can be also applied to any heir of the entity. For example, if water and ethanol inherits from the matter entity the store phenomenon can be also used with them.

The demands for entity classes to be polymorphic is to be declared as heirs of the entity referenced by the phenomenon. Hence, a more accurate definition of polymorphic entity classes is:

DEFINITION 4.2 – *Polymorphic PML entity classes*
Two entity classes are polymorphic to a phenomenon if they are heirs of the entity referenced by the phenomenon. ❑

A consequence of polymorphism among entity classes is the genericity characteristic of the PML model classes described by the semantic rule S20 and illustrated in Example 4.2.

4.5.6. Redefinition

When a class is an heir of another, it may need to redefine some of the inherited features. The redefinition mechanism means that part of the inherited behaviour may be changed affecting the represented behaviour. In PML redefinition in inherited model classes is supported in different aspects. Semantic rule S24 guides the redefinition mechanism.

SEMANTIC RULE S24 – *Redefinition of model classes*

The redefinition mechanism is supported in the following model class attributes:

- *Submodel redefinition.* Assume A is a structured model and B is an heir of A. A submodel SA of A may be replaced in B by a model SB if, and only if, SA and SB are polymorphic. The name of the SB instance in model B must be equal to the name given to the SA instance in model A.

- *Entity redefinition.* An heir class may redefine the inherited entities if an only if the new entities are heirs of the replaced ones. An entity may be replaced by more than one entity in an atomic model. If the atomic model inherits ports referring to the redefined entity classes, it must to redefine also these ports establishing the proper reference to the new entities.

❑

The redefinition mechanism can be very useful to declare new model classes without starting from scratch. The following example illustrates how to use the redefinition mechanism combined with the inheritance mechanism in order to build new models by reusing predefined ones. Firstly, a model with heat capacity will be created overloading the inherited `tank` model class behaviour. Secondly, an aggregated model will be redefined by substituting one of its components.

EXAMPLE 4.4 – *Model of a tank with heat capacity*

Let us consider the model of a tank storing matter able to describe the internal energy of the stored medium. Since we have already developed the model of a tank in Example 4.1 (see Listing 4.6), we can overload the `tank` model class basic behaviour by adding a new phenomenon describing the internal energy associated to the stored matter.

We will consider the internal energy to be described in terms of the medium enthalpy because of simplicity reasons. Actually, this assumption is true if the stored matter is in liquid phase (a

more rigorous library development should have contemplated this fact by establishing a different entity class hierarchy including `liquid` as `matter` heir and declare it in the `tank` model class).

Hence, let us consider that the internal energy balance of the stored matter can be described in the following terms (see for instance (Stephanopoulos 1984)):

$$\begin{aligned} \frac{dH}{dt} &= \sum_i mF_i \times h_i \\ H &= m \times h \\ h &= c_p \times T \end{aligned}$$

where H, h, T and m are the total stored matter enthalpy, its specific enthalpy, its temperature and the stored mass respectively; i indexes the ports of the model where the law instance will be created and mF_i and h_i are the mass flow and the specific enthalpy at each port.

The new `storeInternalEnergy` phenomenon and `enthalpyBalance` law classes listed at Listing 4.11 are included in our modeling library to represent the matter thermal dynamics. At the same listing it is shown the new `tankWithHeatCapacity` model class. It is declared as an heir of the `tank` model class and the inherited basic physical behaviour has been overloaded by including the `storeInternalEnergy` phenomenon.

The `tank` and the `tankWithHeatCapacity` classes are polymorphic since they have the same interface and operate with the same entities (recall model polymorphism definition 4.1). Therefore, it is possible to use instances of these two classes without distinction in an aggregated model according to the semantic rule S24.

For instance, the `process` model class in Example 4.1 (see Listing 4.8) may be redefined as a model where the `tank` model is replaced by a `tankWithHeatCapacity` model

```
(process [newProcess]
      [submodels(tank(mTA)=tankWithHeatCapacity(mTA))]
);
```

Note that the `tank` instance name `mTA` is preserved in the `newProcess` heir since the likely inherited equation section could make reference to attributes of the `tank` instance which must

```
(phenomenon [storeInternalEnergy(matter)]
   [enthalpyBalance(matter)->enthalpy(matter)]
);
(law [enthalpyBalance(matter)]
     [{
        der(Enthalpy(matter)) = sum(portInstances({matter,
                                        prod({massFlow(matter),enthalpy(matter)})
                                 }),
        Enthalpy(matter)=prod({mass(matter),
                               enthalpy(matter)}),
        enthalpy(matter) = prod({specificHeat(matter),temperature(matter)}),
        PHENOMENON(store(matter))
     }]
);
(tank [tankWithHeatCapacity]
   [{
      phenomena(storeInternalEnergy(matter))
   }]
);
```

Listing 4.11. New PML classes added into the library in order to describe the matter enthalpy as a measure of its internal energy.

also be valid in `newProcess` (e.g. changing the tank section by means a PML sentence such as `mTA.section=2`). ❑

As the example has illustrated, the new model may declare new behaviour with respect to the replaced one. It is possible that the new behaviour requires of entity properties which were not involved in the former model and whose dynamics are not described in the rest of components of the aggregated model. For instance, the `tankWithHeatCapacity` defines a new phenomenon with respect to the `tank` in order to represent the matter internal energy involving new matter properties such as the matter enthalpy. Since there is an exchange of matter between the `mTA` instance and the rest of connected duct model instances, the enthalpy of the matter at each conduction is required when the matter flows into `mTA`. However, there is no phenomenon declared in the duct model to represent the matter enthalpy dynamics. If the matter flows out from the heater, the matter enthalpy at the corresponding duct will be the enthalpy of the

matter in the heater, whose dynamics is set by the `storeInternalEnergy` phenomenon. The only thing that PML can do is to warn the model user that there is a submodel redefinition that may lead to incomplete simulation models since behaviour that was not defined in the replaced submodel appears in the new submodel. In order to get a correct simulation model, the physical analysis procedure sets the unknown flowing matter enthalpies as boundary conditions for the simulation experiment.

With respect to the entity redefinition mechanism of rule S24, it is the combination of the inheritance mechanism together with the genericity characteristics discussed in Section 4.5.2.

4.5.7. Dynamic binding

In the object oriented programming context, dynamic binding is the capability of calling the feature of the object attached in run time, which is not necessary the same in different executions of the call. Somehow, the behaviour of the application is being adapted continuously at runtime depending on the object owner of the called feature.

In PML, as in any of the object oriented modeling languages, speaking about runtime is meaningless. The modeling objects, i.e. the modeling class instances, are created at compilation time when the object oriented model is translated into the simulation model. In this translation procedure, the fourth layer of PML model construction, the mathematical model is formulated according to the reusing context of every class used in the model.

Definition 4.3 – *Behaviour dynamic binding in PML*

PML defines the *behaviour dynamic binding* as the ability to adapt automatically the behaviour mathematical representation to the physical context. ❑

This ability is a consequence of the possibility to analyze the PML models from the physics perspective by means of the physical analysis procedure. The *dynamic* adjective is used to express that the behaviour formulation is performed once the physical reusing context of every modeling classes has been taken into account. Therefore, PML defines a dynamic formalism according to Definition 1.8.

This capability is an important contribution of PML, since the behavior mathematical formulation is not included in the model declaration and it is not performed until the fourth design layer (see Figure 4.1). At the fourth layer the mechanisms discussed previous sections have been already executed, and the physical analysis procedure has taken into account the possible model manipulations made by the user at the third layer (the third layer is presented in the next chapter). Hence, a great separation between the PML model and the experimentation purposes can be achieved.

The capability to dynamically adapt the behaviour representation to the context embraces several aspects of the model mathematical formulation. On the one hand, the relationships between the basic physical behaviour declared in the models (modular local behaviour) and its formulation by means of the proper law set selection. On the second hand, the determination of the phenomenological structure of an aggregated model, which has been defined as dynamic modularity (see Definition 3.3 in the section 3.5).

Modular local behaviour

The capability of the PML modeling classes to represent the system behaviour in a modular way (introduced in Section 3.4) can be explained here in terms of the dynamic mathematical behaviour formulation (behaviour dynamic binding).

In difference with the static formalisms (according to Definition 1.7), the PML modeling classes permit a clear separation between the physical aspects of a model and its mathematical facet. As a consequence of that, only the behaviour which is local to the modeled system component has to be declared in terms of the phenomenon and entity PML classes (see Section 3.4.1 and semantic rule S12).

With PML the separation is possible since the mathematical facet is derived from the model physical analysis. The way in that the behaviour dynamic binding is performed to formulate the local behaviour is guided by the communication semantic rule S19. According to this rule, each phenomenon and entity pair is used to create the proper law instance coming to the behaviour mathematical formulation.

Therefore, one of the tasks of the behaviour dynamic binding is to implement the phenomenon-entity-law communication rule.

The redefinition mechanism (see semantic rule S24), illustrated by Example 4.4, is possible since the behaviour formulation is dynamically adapted to the reusing physical context. In this case, the redefinition of a submodel or an entity in a new model is considered as a change of the former model physical context. The availability of generic model classes (see semantic rule S20) is also a consequence of the behaviour dynamic binding.

Coupled behaviour

As it was said in section 3.4.2, the PML port has been designed to provide with a level of delayed binding enough to set the proper coupled behaviour once the model reusing context has been analyzed. The port is used to find the physical interactions which are not modular to one single component of an aggregated model through the entity streams paths defined by the port connection topology. This capability has been named as dynamic modularity and its analysis gives rise to the functional model (see definitions 3.2 and 3.3).

The way in which the behaviour dynamic binding is performed to formulate the coupled behaviour is guided by the communication semantic rule S18. According to this rule, the entity exchange phenomena defined by the port connection is used to find the proper coupled behaviour.

Therefore, the second task of the behaviour dynamic binding is to implement the model-to-model communication rule.

4.6. Advanced modeling concepts in PML

The previous sections in this chapter have introduced the basic semantic rules to build a PML modeling library. These rules will permit the declaration of the physical knowledge and behaviour required to define the model of a system either aggregating predefined basic knowledge (entity, phenomenon and law classes) or aggregating predefined model classes (topological modeling). In this section we present two more mechanism to increase the reusability of predefined modeling classes.

Model parameterization

One way to extend the reusability of predefined models is the separation between the system representation (process unit or device) and the medium representation (the product or processing subject). This separation would permit the reuse of a process unit model with different medium models. This has been the traditional approach in flowsheeting causal modeling. The medium models are functions to calculate physical medium properties. So in certain cases it is possible to reuse a process model with different products since the behavioural aspects related to the medium are resolved by means of function calls accessing into a medium database.

This behaviour decomposition has also been thoroughly explored in the equation based object-oriented modeling approach. For instance, within the chemical process modeling domain, very interesting ideas are proposed by (Nilsson 1993) in order to decompose the media and units representation. Basically, they consist in the definition of modeling components where every aspect related to the medium behaviour is encapsulated. For example, a modeling component describing a matter balance. These modeling components are aggregated into a model class. The class should be parameterized in terms of the medium, so it can be reused with different media.

The decomposition presented in (Nilsson 1993) was not fully supported by the Omola language (Andersson 1990, Andersson 1994), which he uses to illustrate his proposals. Current languages, such as EcosimPro and Modelica, provide with sentences which permit to some extent this separation. For example, the `EXPAND` operator in EcosimPro permit to expand equations over a index which can be provided as a parameter to the model class (Pérez *et al.* 1999). Replaceable (`replaceable` and `redeclare` operators) classes can used to medium separation with the Modelica language (Mattsson and Elmqvist 1998, Modelica-Association 2000).

The unit-medium separation in both languages is based on the model parameterization. The model parameter makes reference to the medium and the model equations are adapted to it. In EcosimPro, the adaptation is restricted to the number of the equations that must be used to formulate certain dynamic. This is useful when, for instance, we have to work with medium with

different number of components. For example, the calculation of the mass of each component by multiplying their concentrations times the total mass of product, i.e., $w_i = w \times x_i$. The number of equations depends on the number of medium component. This can be formulated by means of the `EXPAND` operator (e.g. `EXPAND (j in mix) w[j] = w*x[j]` where `mix={N2,O2,H2O}`). In Modelica, the consistency of a model can not be assured when part of the formulated behaviour is overridden by means of the `redeclare` mechanism (Modelica-Association 2000).

The separation between the model and the medium in PML comes up in a natural way since they are represented by two different modeling classes. Their consistency relationship is controlled by semantic rule S19 and by the inheritance rules. It is assumed that, if the behaviour represented by a model declaring a set of phenomena is consistent with an entity E, it will be also consistent with any inheritor entity. Of course, the behaviour formulation may vary and rule S19 takes care of setting the proper law formulation.

The possibility to parameterize a PML model class in terms of the PML entity model class is a consequence of semantic rules S20 and S24. Model parameterizations are governed by the following rule:

Semantic rule S25 – *Model Parameterization.*

Any non abstract PML model class may be considered as a parameterized class with respect to the entities which it defines since the entities may be redefined when the model class instance (model object) is created. A PML model object (see semantic rule S15) may redefine its entities according to semantic rule S20. A parameterized model object must indicate which is the redefined entity and by which entity is replaced. Redefinition by multiple entities are not allowed. ❑

The example bellow illustrates the use of model parameterizations in order to extend the model reusability while preserving its consistency.

Example 4.5 – *Reuse of predefined models with different entities.*

By reusing the tank model class (see Listing 4.6) and the valve model class (see Listing 4.7) developed in Example 4.1, we may define the model class representing two interconnected tanks

```
(model [twoTanks]
  [{entities({liquid}),
    submodels({tank({tk1(matter=liquid),tk2(matter=liquid)}),
               valve(v(matter=liquid))}),
    connections({tk1.P3-v.P1,v.P2-tk2.P3})}]
);
```

Listing 4.12. Two tank system with parameterized submodels.

where the submodel declaration is prepared to propagate the possible entity redefinitions. The code is shown at Listing 4.12.

According to rule S25, the `twoTanks` model class declaration can be considered to be parameterized with respect the `liquid` entity. This is because of the possibility to redefine the entity `liquid` by an inheritor entity when an instance of `twoTanks` is created. For example, we may now define new classes by redefining the `liquid` entity with inheritors entities (`water` and `ethanolSolution`):

```
(twoTanks [twoTanksWater] [{entities(liquid=watter)]);
(twoTanks [twoTanksEthanol] [{entities(liquid=ethanolSolution)]);
```

This declarations will add two new model classes to the modeling library. If we are not interested in new class declarations, we can also create a model instance (model object) for simulation purposes by means of the `analyze` PMT command (see Section 5.3):

```
> analyze twoTanks(twoTankSim(entities(liquid=watter)))
```

where `twoTankSim` is the name given to the instance created to generate the simulation model. The entity redefinition must be properly propagated among the submodels according to the specification made at each model object declaration (e.g. `tk1(matter=liquid)`). The corresponding semantic rule is enunciated bellow (see S26). Once the entity redefinition is propagated, the behaviour formulation is adapted to the redefined entities by selecting the proper law formulation (semantic rule S19). ❑

An entity redefinition may affects to the formulation of the behaviour declared in the model. Hence, it is necessary to define how the entity redefinition is propagated through the submodels in a structured model and through the phenomena in the atomic models. The capability to propagate entities among the submodels of a topological model and among the phenomenon declared at the atomic models is governed by the following rule:

SEMANTIC RULE S26 – *Propagation of redefined entities.*

When an entity is redefined in a parameterized model according to semantic rule S25, every reference to the redefined entity is replaced by the new entity. The change affects to the phenomena making reference to the replaced entity and to the model objects if it is a structured model.

The following conditions should be fulfilled to allow the entity propagation:

1. Propagation on submodels: given a model object m where the entity E is aggregated, E can be replaced by an entity G if, and only if, E is ancestor of G (S20) and m is properly parameterized.

2. Propagation on a phenomenon: given a phenomenon F valid for an entity E, the phenomenon is also valid, and therefore applicable, to an entity G if, and only if, E is ancestor of G.

❑

The parameterization mechanism is similar to the redefinition mechanism (see semantic rule S24) except for its applicability (redefinition is exclusively used to declare new model classes while parameterized classes can also be used to create model objects) and for its more restrictive use (only one by one entity redefinitions are allowed). The redefinition of an entity in a parameterized model class is not simply a matter of renaming the entity. According to semantic rule S19, the behaviour mathematical formulation may be significantly different. This is the case of the models in Example 4.5. The EcosimPro code of Example 4.5 is shown in Appendix C.2.

Phenomenon inheritance

The possibility to establish a relationship between the applicability of a law and the presence of dependent phenomena in the model class declaration was indicated in sections 4.2.2 and 4.2.3. The law class shown in Listing 4.3 illustrates the use of the `PHENOMENON` sentence to set such relationship. The dependency relation ensures that the `massBalanceWithReaction` law can only be applied in those models where the `mixReaction` phenomenon class is declared. The law declaration and the dependent phenomenon make reference to the `ethanolSolution` entity class (note that every law declaration in Section 4.2.3 make reference explicitly to the `ethanolSolution` entity class). Therefore, they are applicable in a model class declaring the `ethanolSolution` entity class. So, the model class can only be reused with this entity or, by semantic rule S26, with his heir entity classes. But in this case, the heir classes can not redefine the `ethanolSolution` component attribute (see semantic rule S23). Let us consider the following physical behaviour declaration. Let P be a phenomenon and E_1 and E_2 two entity classes representing multi-component products.

In first term, we will consider that the phenomenon P can be formulated by means of a law which does not refer to the entity components. E_1 and E_2 are declared as inheritors of the entity E (e.g. `matter` entity class declared in Listing 4.4 in Example 4.1). For instance, the storage phenomenon shown at Listing 4.4. The tank model declared in Listing 4.6 is reused by giving the entities E_1 and E_2 as parameters and, according to semantic rule S26, the storage phenomenon can be properly propagated since E_1 and E_2 are `matter` inheritors (see Example 4.5).

In second term, we will consider that the P is formulated by means of a law which makes reference to the entity components. According to semantic rule S19, the behaviour is formulated by:

$$P + E_1 \longrightarrow L_1$$

$$P + E_2 \longrightarrow L_2$$

where L_1 and L_2 are the law classes declared to formulate P according to the entities E_1 and E_2 respectively. Let us assume that E_1 and E_2 have different number of components so that

L_1 and L_2 are completely different formulations. If we want to build a model class M declaring the phenomenon P able to be reused indistinctly with E_1 and E_2, we should be able to find law class in such a way that:

$$P + E \longrightarrow L$$

It should be noticed that L, L_1 and L_2 can not be altogether valid behaviour formulations since if L is valid, it should refer to the E components and, therefore, L_1 and L_2 are not valid since E_1 and E_2 can not redefine the E components. If we define L as a *dummy* law enunciated without referring to the E components, the model M is invalid, and only its parameterizations through E_1 and E_2 could be valid. This is not a good solution and it has been one of the main reasons for postulating semantic rule S23 in order to prevent the definition of invalid models due to the parameterization.

However, semantic rule S23 introduces a serious constraint to model reusability since the behaviour declaration in terms of phenomena is medium independent in spite of law formulations are medium dependent. For example, we would like to be able to declare a model class for a mix reactor without being worried about the medium used to parameterize it. This is not possible with the semantic rules defined up to this point. The proposed solution is given by the two semantic rules defined bellow.

Semantic rule S27 – *Phenomenon abstract class*

A phenomenon class is considered to be abstract when it does not establish any relationship with a law class. An abstract phenomenon class must declare the set of properties affected by the physical phenomena it represents. The aggregation of an abstract phenomenon class in a model class makes this class abstract too. ❑

A sort of delayed binding is achieved with this rule. The abstract phenomenon class declares certain behaviour without referring explicitly to any law. However, it represents the involved entity together with the affected properties. This will permit to analyze all the behaviour represented in the model and determine if it is properly defined. The analysis of the abstract phenomena is reduced to the correctness determination of the referred entity and properties.

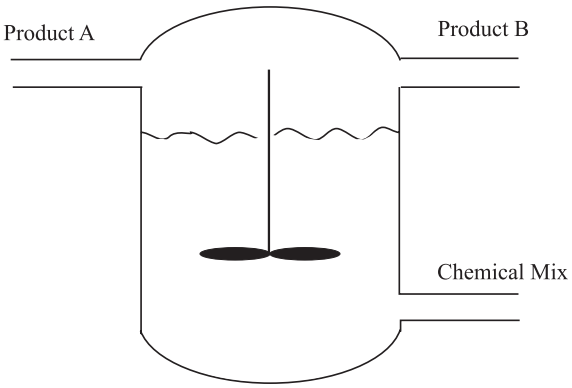


Figure 4.7. Process unit where two product are mixed.

The use of a law class whose applicability is dependent on the occurrence of the phenomena represented by the abstract phenomenon class is therefore allowed. The complete analysis is performed once the model class is parameterized and the link to the law class formulating the abstract phenomenon can be set. The inheritors of the abstract phenomenon will establish the links to the law classes according to the following rule:

Semantic rule S28 – *Phenomenon class inheritance*

A phenomenon class may be declared as an heir of an abstract phenomenon class. The entity declared by the inheritor phenomenon must be an heir of the entity referred by the abstract phenomenon. An inheritor phenomenon class must establish, at least, one relationship with a law class formulating the dynamics of the properties declared at the ancestor abstract class. Any non abstract phenomenon class declaration is considered as final, i.e, non inheritors can be declared. ❑

The example bellow illustrates the use of these two rules in order to extend the reusability of predefined modeling classes.

Example 4.6 – *Use of abstract phenomenon class*

Figure 4.7 shows a process unit where a chemical mix of different substances takes place giving rise to a multi-component product. The modeling classes required to represent this unit are

m	mass
rho	density
x	concentration
v	volume
gM	generated mass
w	mass flow
q	volumetric flow
p	pressure
dP	pressure drop

Table 4.15. Symbols used to describe the matter properties.

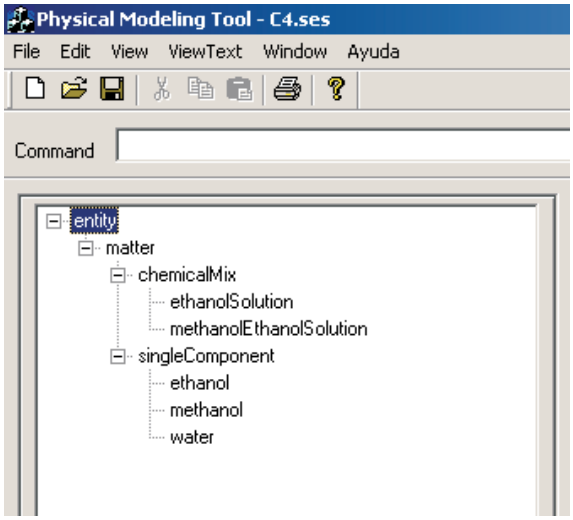


Figure 4.8. Entity class inheritance tree at Example 4.6.

developed in this example. The objective is to declare a PML model class named `mixReactor` able to represent the process unit behaviour operating with different chemical products. The entity classes are defined in first term (Table 4.15 shows the meaning of the symbols used to describe the entity properties):

```
(entity [matter]
  [properties({m,gM,w,p,rho,v,dP,x})]);
(matter [singleComponent] []);
(matter [chemicalMix] []);
(singleComponent [water] []);
(singleComponent [ethanol] []);
(singleComponent [methanol] []);
(chemicalMix[ethanolSolution] [components({water,ethanol})]);
(chemicalMix[methanolEthanolSolution] [components({water,ethanol,methanol})]);
```

The entity class hierarchy can be observed at Figure 4.8. The `chemicalMix` entity class is introduced in order to declare the phenomena related to a multi-component product in a system

where chemical reactions may occur. This is a design decision that, as it shown bellow, will increase the reusability of the modeling classes. What it should be achieved is a model which can be parameterized in such a way that the execution of the following commands:

```
> analyze mixReactor(myMixReactor(entities(chemicalMix=ethanolSolution)))
> analyze mixReactor(myMixReactor(entities(chemicalMix=methanolEthanolSolution)))
```

will lead in both cases to valid simulation models. Hence, the `mixReactor` PML model class should be prepared to propagate properly the medium parameterization.

The phenomenon and law classes needed to represent the storage and chemical mix phenomena can be now declared. In relation with matter storage, the more generic `matter` is referred in order to represent physical behaviour which can be reused with any `matter` inheritor:

```
(phenomenon [store(matter)] [massBalance(matter)->m(matter)]);
(law [massBalance(matter)] [der(m(matter)) = sum(portInstances(w(matter)))]);
```

The model class to represent the matter storage behaviour and the matter exchange port class can be now declared:

```
(port [matterPort(matter)] [{w,p}]);
(model [vessel]
  [{entities({matter}),
    phenomena(store(matter))
  }]
);
```

The tank model class shown in Listing 4.6 is not reused since it declares ports which are annoying for the model parameterization purposes. It is not a good decision to aggregate port declarations in the more generic model classes (upwards in the model class inheritance tree) since it can reduce the model parameterization capabilities.

The classes required to represent the physical behaviour related to the chemical mix phenomenon will be now defined. These classes have to be prepared to support the entity propagation due to model parameterizations. That is, we want to have a mix reactor model class which can be reused with different chemical mixes and not different mix reactor model classes for each chemical mix.

The first defined class is a phenomenon to represent the chemical mix. This class, named `mixReaction`, should be declared abstract in order to support the parameterization of the models aggregating this class. There are two causes to declare `mixReaction` as an abstract phenomenon class where we only refer to the entity affected properties without establishing a link with any law class. The first one is a consequence of semantic rule S23 which says that the components of an entity class can not be redefined by the inheritance mechanism. Hence, if `mixReaction` is declared to occur with the entity `chemicalMix` and makes the link with a law (non abstract phenomenon class), we were forced to declare the `chemicalMix` components. These components can not be redefined in its inheritor entity class so, in fact, the `mixReactor` model class can not be adapted to different chemical mixes which would require different law formulations of the `mixReaction` phenomenon. For instance binary or ternary chemical mixes with perfect and non perfect mixes. The second one is motivated by consistency reasons (the correctness quality factor to be discussed in Section 4.7). We should guarantee that if model class is valid for certain entity, say `chemicalMix`, it should be also valid for the entity `ethanolSolution` since it is inheritor of `chemicalMix`. This consistency check will require the use of abstract phenomenon classes.

The abstract phenomenon can be declared as:

```
(phenomenon [mixReaction(chemicalMix)]
   [{gM(chemicalMix), x(componentInstances(chemicalMix))}]
);
```

and the `mixReactor` model class as an heir of `vessel` model class:

```
(vessel [mixReactor]
   [{entities(matter=chemicalMix),
     phenomena(mixReaction(chemicalMix))
   }]
);
```

New physical behaviour should be declared since the mix reaction phenomena affects to the matter balance and the predefined `massBalance(matter)` law does not contemplate it. The following law class declaration makes its application dependent on the occurrence of the mix reaction phenomenon:

```
(law [massBalanceWithReaction(chemicalMix)]
    [{der(m(chemicalMix)) =
          sum({portInstances(w(chemicalMix)),gM(chemicalMix)}),
      PHENOMENON(mixReaction(chemicalMix))
    }]
);
```

The storage phenomenon class is made multi-faceted in order to set the link to the previous law when the `matter` entity is replaced by the `chemicalMix`:

```
(phenomenon [store(matter)]
   [{massBalance(matter)->m(matter),
     massBalanceWithReaction(matter)->m(matter)}]
);
```

The `mixReactor` model is abstract since it aggregates an abstract phenomenon class (see semantic rule S27). Hence, no instances (model objects) for simulation purposes can be created without a proper model parameterization. This model class declaration provides the required degree of delayed binding. In order to use an instance of this model, it must be parameterized with

an entity (e.g. `ethanolSolution`) in such a way that the `mixReaction` phenomenon together with the entity sets the link to the law which formulates the phenomenon. It is accomplish by defining the following modeling classes. In first term, the non abstract heir phenomenon class:

```
 (mixReaction [mixReaction(ethanolSolution)]
    [perfectMix(ethanolSolution)->{gM(ethanolSolution),
                                   x(componentInstances(ethanolSolution))}]
 );
```

and the law class in second term:

```
(law [perfectMix(ethanolSolution)]
   [{gM(ethanolSolution) =
         sum({portInstances(w(water)),portInstances(w(ethanol))}),
     x(water)=div({m(water),m(ethanolSolution)}),
     x(ethanol)=div({m(ethanol),m(ethanolSolution)}),
     der(m(water)) = sum({portInstances(w(water)),
                          portInstances({ethanolSolution,
                                         prod({w(ethanolSolution),x(water)})
                                        })
                         }),
     der(m(ethanol)) = sum({portInstances(w(ethanol)),
                            portInstances({ethanolSolution,
                                           prod({w(ethanolSolution),x(ethanol)})
                                           })
                           }),
     rho(ethanolSolution) = sum({prod(q,x(ethanol)),b})
   }]
);
```

The `mixReactor` model class can be now used to generate a simulation model with the `ethanolSolution` entity by executing the command:

```
> analyze mixReactor(myMixReactor(entities(chemicalMix=ethanolSolution)))
```

The entity parameter is considered invalid if there is not defined any non abstract `mixReaction` phenomenon for the entity. In this case, an error message can be reported to the model user advising him of the incorrect model reuse. ❑

The semantic rules related to phenomenon inheritance have not been implemented at the moment of writing this text, even we are working on it for the next upgrade of the PML language. This is the reason for not having included in Appendix C the EcosimPro code of the example above.

4.7. Object-Oriented quality factors

The most important quality factors of the object-oriented methodology were introduced in Section 2.3. Their fulfillment has been a central task of the PML design. A discussion of how PML deals with these quality factors follows.

Correctness.

As it has been defined, correctness is the ability of models to represent systems performing their exact task, as defined by their specification. In PML the approach to correctness is viewed in a *conditional* manner, since different steps or layers are involved before the simulation model can be operated (see Section 4.1 and Figures 2.13 and 4.1). The conditional approach concentrates in guaranteeing that each layer is correct on the assumption that the previous levels are correct. Correctness is verified in terms of the language semantic rules (see Appendix B).

At the first layer of the modeling library construction (physical knowledge representation), correctness is analyzed by checking the consistency of the entity (S1, S3 and S23), phenomenon (S5 and S6) and law (S7,S4 and S9) class declarations. The cross-references established among the entity-phenomenon-law classes are also checked (S19).

At the second layer, model classes are declared. The PML atomic model (S14) correctness analysis consists in determining that the referenced entities and phenomena are properly declared in the modeling library and the corresponding laws can be formulated (S12). The PML structured model (S15) correctness verification should additionally validate the submodel connections (S18). Any inheritance (S22) and redefinition (S24) relationships must be also validated.

During the third and fourth steps several checks are made in order to guarantee a proper simulation model generation (see Figure 2.13). For instance, the correct propagation of model parameterization, feasibility of the hypothesis imposed at the third layer (e.g, the selection of one of the two transport laws shown at Figure 4.1(b) to describe the transport phenomenon) and the possibility to reach the an adequate simulation model by means of the *Physical Analysis Procedure* (Ramos *et al.* 1998*a*, Ramos *et al.* 2001) and Chapter 5.

Extendibility.

It has been defined as the ease of adapting a model to changes of specification, i.e., the variation of the experimental framework and the adequacy level. This often will occur with models in different aspects according to the variety of their applications. For instance, the change of a submodel in a model or the addition of new behaviour in a model class should be propagated to the simulation model without side effects on the rest of defined model classes. The fulfillment of two principles has been essential to support extendibility:

- *Design simplicity.* The capability of building structured models according to the system topology has been shown as a simple approach suitable for the modeling purpose (topological modeling);
- *Decentralization.* The ability to circumscribe the effects of a change in a modeling class. The fulfillment of this principle is a consequence of the modular property that has been imposed to the PML modeling classes.

Reusability.

It has been defined as the ability of modeling classes to serve for the construction of many different models. Reusability is a quality factor whose main benefit is probably the reduction

of the model construction cost, but also it influences to improve other quality factors such as correctness or extendibility. In addition to the model reusability by means of aggregation and inheritance supported by other object-oriented modeling approaches, the formalism defined by PML extends to the other modeling classes. This is the consequence of the scope to reusability defined in this work and its implications on the reusing context. The entity-phenomenon-law relationship makes possible to reuse theses classes in different contexts, giving in each case the adequate behaviour formulation.

Efficiency

In the modeling domain means the ability of a model to place as few demands as possible on complexity considering a particular application for model experimentation. An ideal situation will be to have one unique model for a system and provide with the capability to adapt its mathematical formulation to the application purpose, simplifying this representation as much as possible according to the required complexity. We have related efficiency to the adaptation of the topological model into the experimental framework and adequacy level defined by the experimentation purpose.

This ability is mainly supported at the third design layer where, for instance, the user may choose from several laws to describe a phenomenon (e.g. a linear description of the matter transport) or even he may neglect a set of phenomena which are not interesting for his particular experimentation purpose. The convenience of efficient models is also considered at the fourth design layer to obtain simulation models where physical knowledge is used to avoid algebraic loops and high index problems (Ramos 1995, Ramos *et al.* 1998*a*).

4.8. Summary

This chapter has presented the object-oriented modeling language PML. The construction of PML modeling libraries is based on five modeling classes: entity, phenomenon, law, port and model. The physical knowledge and behaviour required to specify the system at the topological structure level can be represented by means of theses classes. The modular structure given to

the modeling classes allows the extension of reusability to the scope defined in Chapter 1. Four layers have been distinguished in the construction of a modeling library. Physical knowledge and behaviour are declared at the two first layers in terms of the five PML modeling classes. These classes must be declared according to the language semantic rules defined in this chapter. The two last layers are related to the usage of the predefined models for simulation purposes. This subject is treated in next chapter.