5

PML model physical analysis

This chapter describes the different aspects involved in the simulation model generation. The simulation model is the system specification where the expected behaviour is represented and has the adequate structure able to play the prescribed system behaviour (Klir 1969). In this chapter the third and fourth layers of PML model construction are presented (see Figure 4.1(a)). These steps are prior to the simulation activity and basically comprise the adaptation of the PML models to the experimentation purpose by setting the phenomena of interest defined by the experimental framework and by selecting the law formulations which cope with the desired adequacy level, as well as the simulation model generation.

There are many of the modeling applications where a reduced or simplified system behaviour representation should be obtained in order to apply certain system analysis theories. One very important application domain is, for instance, the control system domain. Most of the control design techniques require of linear models, or low order models, external models (transfer functions) or internal models (state-space form), etc. This sort of models are usually obtained by making different hypothesis such as to neglect certain dynamics, linearize the selected dynamics and/or reduce the order of the linear model (Johansson 1993). Some of the simplifications made on the system representation lead to models which still preserve some physical meaning. For instance, a linearized model obtained from the basic physical principles. But there are also model reduction techniques where the resulting model can be hardly related to the physical

phenomena happening in the system. For example, a model which is defined to match with a known system frequency response within a frequency interval. It is rather difficult to relate this model structure and its parameters with the physical system structure and parameters. So in fact, such type of models do not really represent the system behaviour, but they represent certain system dynamics under very precise operation conditions.

The performance of the controller facing the real system is highly dependent on the goodness of the reduced model used in the analysis and design stages. To validate the controller with a more precise system simulation model before its operation is a good, and recommended, practice. Usually, such a realistic model does not exists since the reduced model development has been guided by the control design purpose and, likely, the applied modeling procedure is not the most convenient to obtain such a realistic model. Even when the effort to develop a realistic model is made, it will be quite difficult to establish an analogy between the realistic model and the reduced model, which could be really useful if the reduced model must be refined in order to iterate on the control law design. For example, consider that a parametric model m_r has been identified to design the controller and a model M is postulated from the basic physical principles for the controller validation. Generally, there is not a direct relationship between models m_r and M. This fact makes difficult to establish any guideline to improve the reduced model m_r adequacy (made for design purpose) from the realistic model M (made for validation purpose).

This application case illustrates the benefits which can be obtained when a unique system specification built at the topological level (a PML model class) can be adapted to different experimentation purposes (different simulation models). The PML model adaptation into the experimental framework and adequacy level is discussed in Section 5.1. Section 5.2 presents how the functional model is obtained by means of the physical analysis procedure. It is also shown how the non causal mathematical model is generated by creating the law formulation instances. Finally, Section 5.3 will present a brief overview of the modeling tool PMT.

5.1. Model manipulation

The third and fourth layers in the PML model construction process are related to the reuse of a predefined PML model class and its adaptation to some specific experimentation purpose. The adaptation of this model involves several aspects with a main objective: the generation of an adequate simulation model for the experimental framework. Three different types of model adaptation will be considered:

- Behaviour pruning: A PML model declaring a set of phenomena may be simplified by discarding the phenomena which is not of interest for the experimental framework. We call this behaviour pruning. The model user may also parameterize a ready-made model with different entity classes as it was discussed in the previous chapter.

- Adequacy set up: The adequacy of a model was related in Chapter 1 to its capability to be the simplest causal explanation to the phenomena of interest with the desired accuracy (see Definition 1.4). Different levels of accuracy may be achieved by the use of multi-faceted phenomena. This mechanism provides with the possibility to formulate the behaviour represented by a phenomenon with different degrees of accuracy.

- Extension of the experimental framework: the experimental framework may be modified by the addition of new physical behaviour to the already defined model classes. This adaptation is not so immediate since it requires the modification of the modeling library with new modeling classes to represent the additional physical behaviour. Even so, these modifications have minimum side effects on the already defined modeling classes.

Additionally, there are aspects related to the behaviour mathematical formulation which can be taken into account in order to increase the simulation model adequacy. Different hypothesis may be set to increase the simulation model efficiency in terms of computation time. For example, the number of conditional equations to set the properties of a flowing matter may be significantly reduced if we can assign the flow sense (see Section 5.2.2).

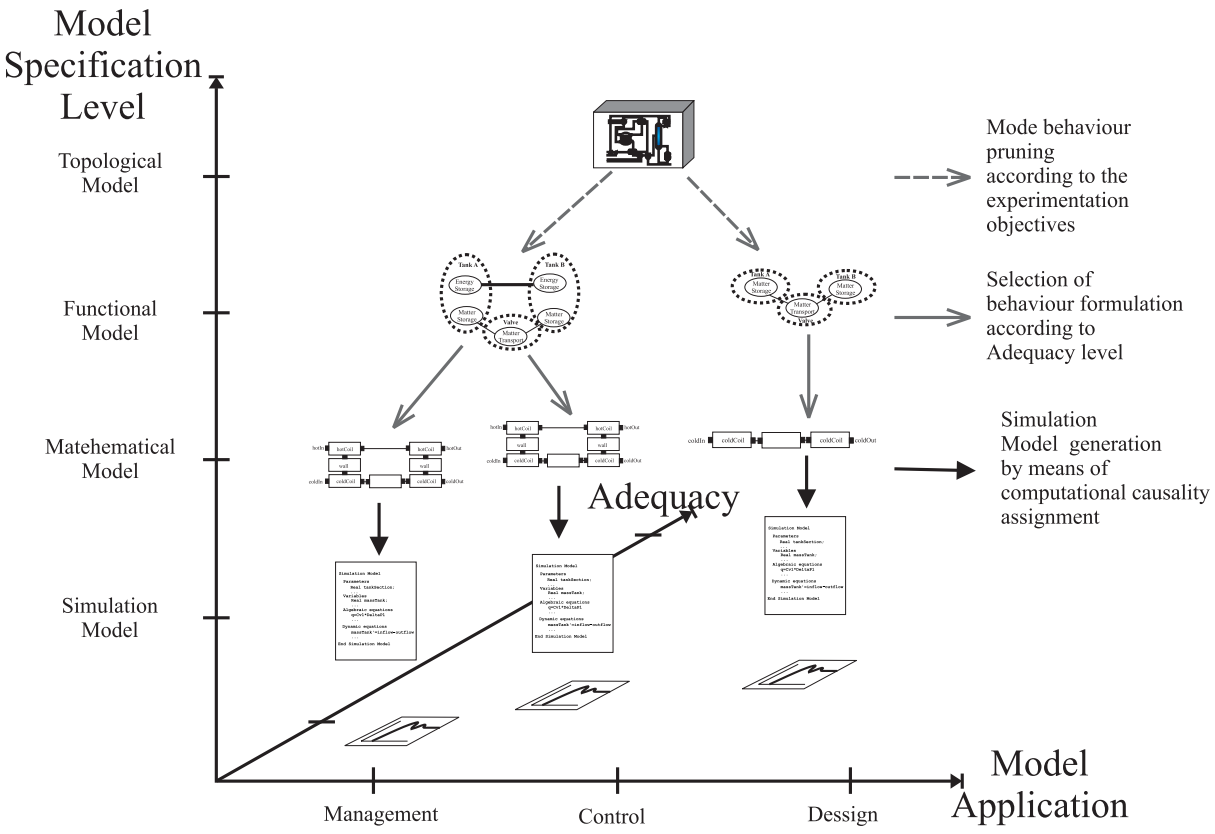

Figure 5.1. Modeling trajectories in the third and fourth layers defined by the PML environment.

Figure 5.1 illustrates these manipulation steps made through the third and fourth layers in the PML model construction process (see the Figure 4.1). The starting point in this procedure is the reuse of a predefined PML topological model to generate the simulation model. According to Definition 3.1, the topological model may be represented by an atomic or a structured PML model class (see semantic rules S14 and S15). The topological model has been drown within a cube to symbolize the capability to extend its behaviour representation capabilities along the adequacy and experimental framework coordinates.

5.1.1. Model behavior pruning

When a model includes a detailed and extensive representation of the system behaviour, it is quite possible that the model results too complex for many experimentation purposes where such a precise representation is not necessary. Hence, it may become inefficient if part of the

represented behaviour can be neglected. In such situation it could be very useful to have the possibility to prune all the behaviour which is not of interest. This model reduction is error prone, so the modeling tool should provide with mechanism to assure that the reduced model is still valid or, at least, it does not contain incoherences from the correctness point of view, as it has been defined in the Section 2.3.

The importance of having one single system model, preferably built at the topological specification level, which can be adapted to the experimentation purpose has been shown above within the control design domain. This situation can be extended to other related domains such as operator training or production planning. Different simulation models will be required at each domain, so such a topological model must be adaptable to the experimentation purpose. The model adaptation procedure, which probably means a model simplification and reduction, should not imply the definition of a reduced simulation model where the formulated dynamics can not be related to the physical behaviour, since it would not permit to easily refine the reduced simulation model when required.

A main advantage of the PML representation framework is the possibility to obtain reduced or simplified models by neglecting the behaviour which is not of interest for some experimentation purpose. Considering a PML system model as an aggregation of phenomena classes, the model behaviour pruning can be performed at a physical level by selecting only the phenomena required for experimentation. The PML model of a heat exchanger unit (see the Figure 5.2) will be used to illustrate the simplification procedure in order to generate different simulation model where the phenomena of interest is mathematically formulated. A simulation model, including the formulation of the whole behaviour represented in the PML model of the heat exchanger unit, will be generated in a first term. Afterwards, a reduced simulation model is generated by considering an experimental framework where the thermal phenomena occurring in the process are considered to be irrelevant. So, a simplified model describing the fluid transport phenomenon is more convenient for the experimentation purpose.

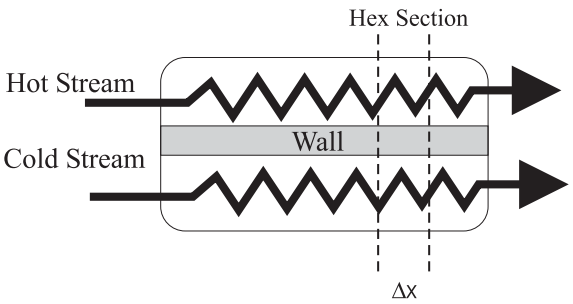

Figure 5.2. Representation of a heat exchanger process unit.

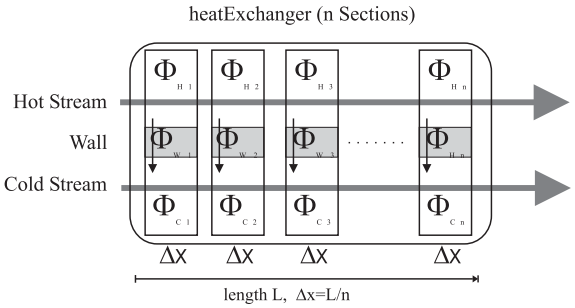

Figure 5.3. Conceptual representation of a heat exchanger. The unit is divided into sections in order to avoid the use of partial differential equations to represent the thermal energy dynamics.

EXAMPLE 5.1 – *Model of a heat exchanger*

The PML model of the heat exchanger based on the unit TS30 presented at (Mattsson 1997) is built in this example, although some small simplifications are introduced since the main objective of this example is to illustrate some of the PML language characteristics.

The basic idea behind the operation of a heat exchangers is to let two media, which flow on the two sides of the wall, exchange heat through the wall without being mixed. The model of the heat exchanger can be conceived as two ducts with a common wall through which heat can flow (see Figure 5.2). The model will be based on this conceptual view. A common approach, to avoid the complexity of using partial differential equations to describe the heat transfer from the hot side to the cold side, consists in dividing the heat exchanger into a number of sections of length Δx along the direction of the flow (see Figure 5.3). The energy dynamics within this section can be considered independent from the spatial coordinates when Δx is small enough.

Such a section consists of two duct sections and one wall section. The matter transport through the ducts and the thermal energy conduction though the wall should be modeled. By following the steps defined by the PML methodology, the basic physical knowledge (entity, phenomenon and law classes) is defined in first term. Many of the PML classes built at Chapter 4 can be reused by extending their declaration to the new experimental framework.

cp	specific heat
T	temperature
Tref	reference temperature
rho0	density at Tref
h	specific enthalpy
H	Enthalpy

Table 5.1
Additional symbols used to describe the matter thermal properties (see the Table 4.15).

phi	heat flow rate
thE	thermal effort (associated to temperature)
dTh	thermal gradient

Table 5.2
Symbols used to describe the thermal energy properties.

To represent the matter thermal characteristics, new properties should be declared for the `matter` entity class with respect to its declaration at Example 4.6 (see Table 5.1). Here follows the PML code:

```
(entity [matter]
  [properties({m,gM,w,p,rho,v,dP,x,mol,mF,
               h,H,cp,T,Tref})]
);
(matter [singleComponent] []);
(singleComponent [water] []);
```

Additionally, an entity class is defined to describe the thermal energy. This class will be used to model the energy transfer through the wall (see Table 5.2):

```
(entity [energy] [] );
(energy [thermalEnergy] [properties({phi,thE,dTh})]);
```

Finally, an entity class is defined to describe wall material thermal properties related to the heat conduction (see Table 5.3). This class is not derived from the matter class because it should be avoided that the phenomena applied to matter (e.g. matter transport or heat convection) become also applicable to the material entity. Here follows the PML code:

```
(entity [material]
  [properties({R,Rc,Rh,Rw,Rf,lambda,d,Aw,Y})]
);
```

R	Thermal resistance between cold and hot sides
Rc,Rh	Thermal contact resistances (cold and hot surfaces)
Rw	Material thermal resistance
Rf	Thermal fouling resistance (deposits and dirt)
lambda	Thermal conductivity
d	Material thickness
Aw	Contact area
Y	Corrugation correction factor

Table 5.3
Symbols used to describe the wall material thermal properties.

The phenomena and law class definition follows. The transport phenomenon and law classes defined at Example 4.1 can be reused with no modification. The remaining phenomena which should be represented are related to the thermal energy entity: the thermal energy balance and transport in a duct section and the heat conduction through the wall (see Listing 5.1). The `storeInternalEnergy` phenomenon class is used to describe the change of the thermal energy in the duct by means of the matter enthalpy balance (observe the link to the law class). This phenomenon is defined for `matter` and `thermalEnergy` to guarantee its usage in a model where these entities are aggregated since the enthalpy property in absence of matter can not be referred. Additionally, to relate the matter enthalpy with the thermal energy is required since the heat conduction is usually formulated in terms of the temperature of two media in contact. The definition of a multiple entity phenomenon class is the way to assure with PML that these relationships are fulfilled when the phenomenon class is aggregated in a model class.

The `heatConduction` refers also to multiple entity since the thermal energy transfer depends on the `material` thermal properties. The thermal energy transport may affect to some of the material properties depending on the modeled behaviour (e.g. certain wall thermal properties may vary with its temperature). In spite of the material thermal properties are considered to be constants in this example (no wall heat capacity is considered), future extensions of the experimental framework make advisable to define this dependency by declaring the `heatConduction` as a multiple entity phenomenon. The resolution of this dependency is let to the model classes

```
(phenomenon [storeInternalEnergy({matter,thermalEnergy})]
   [enthalpyBalance({matter,thermalEnergy})->
      {h(matter),cp(matter),rho(matter),thE(thermalEnergy)}]
);
(phenomenon [heatConduction({material,thermalEnergy})]
   [heatFlow({material,thermalEnergy})->
      {phi(thermalEnergy),dTh(thermalEnergy)}]);
```

Listing 5.1. Phenomenon classes to represent the thermal behaviour.

where this phenomenon can be reused, in a similar way as the assignment of the parameter values which relate the pressure drop with the matter flow is let to the duct model (e.g. the duct section).

The law modeling classes ruling previous phenomena should be defined to complete the basic physical behaviour representation. When a law class is designed, certain hypothesis related to the wanted experimental framework and the adequacy level must be stated. It does not imply a restriction over these two factors since different law classes can be associated to a phenomenon class. Nevertheless, the law class is used to formulate the phenomenon within certain experimental framework and adequacy level, so its declaration is conditioned by them.

According to the heat exchanger model presented at (Mattsson 1997), the energy associated to the matter will be described by means of its specific enthalpy (`h` matter property). Proceeding this way, the specific enthalpy property can be used to represent both the thermal energy dynamics in a duct (heat balance) and the thermal energy transport through the duct section. The heat balance in a duct section can be represented by:

$$\frac{dH}{dt} = \sum_i \Phi_i + \Phi_W \tag{5.1}$$

where H is the matter enthalpy in the system, Φ_i is the enthalpy flowing through a matter port, i is an index referring the matter ports of the system and Φ_W is the rate of the heat flow across the system contact surface.

```
(law [enthalpyBalance({matter,thermalEnergy})]
     [{DATA({cp=4180,rho=1000}),
       rho(matter)=rho,
       cp(matter)=cp,
       der(H(C5::matter)) = sum({portInstances({matter,
                                          prod({w(matter),h(matter)})}),
                           +portInstances(phi(thermalEnergy))}),
       H(matter)=prod({rho(matter),v(matter),h(matter)}),
       h(matter) = prod({cp(matter),T(matter)}),
       thE(thermalEnergy)=T(matter)
    }]
);
```

Listing 5.2. PML law class to declare a feasible formulation of an enthalpy balance.

The matter enthalpy is now related to the rest of properties to be represented:

$$\begin{aligned} H &= \rho V h \\ h &= c_p T \end{aligned} \tag{5.2}$$

where ρ is the matter density (assumed constant in this law formulation), V is the matter volume (will depend on the process unit model) and h is the specific enthalpy, c_p is the matter specific heat capacity (assumed constant in this law formulation) and T is the matter temperature. The thermal energy transport through the duct section (heat convection) does not require any explicit phenomenon class since it is described by the implicit matter exchange phenomenon through the duct matter ports. The thermal energy dynamics in a duct is formulated by the law class shown at Listing 5.2.

The heat transfer through a material is focused on the heat flow across it in the direction from the hot surface to the cold surface (heat flow along the material is neglected as well as its heat capacity), which means that

$$\sum_i \Phi_i = 0 \tag{5.3}$$

```
(law [heatFlow({material,thermalEnergy})]
     [{sum(portInstances(phi(thermalEnergy))) = 0.0,
       phi(thermalEnergy) = div({dTh(thermalEnergy),R(material)})
     }]
);
```

Listing 5.3. PML law class to declare a feasible formulation of the heat conduction through a solid material.

where Φ_i is the heat flow through the material contact points (thermal energy ports) and i is an index referring to the thermal energy ports used to represent the contact surface. The heat flow rate across the material (i.e. the thermal energy transfer) is modeled as

$$\Phi_W = \frac{\Delta T_h}{R} \tag{5.4}$$

where R is the overall thermal resistance between the contact surfaces and ΔT_h is a suitable representation of the thermal gradient across the material (it will be calculated in the wall model as the mean temperature difference of the matter flowing through the ducts in contact with the wall). This behaviour formulation is represent by the law class shown at Listing 5.3.

The calculation of the `dTh` property (thermal gradient) is let to the wall model since it can not be formulated by the law class in a general sense. That is, it can not be formulated without considering the way in that the model is conceptually structured to avoid the use of partial differential equations (see Figure 5.3).

It should be noticed that the phenomenon and law modeling classes (see Listings 5.1 to 5.3) have been defined keeping in mind the heat exchanger unit but, actually, there is no assumption in their declaration invalidating their reuse in a different process unit model where the thermal behaviour could be be represent by the same phenomena (e.g. the vessel and the jacket of a heated reaction process unit).

Once the basic knowledge has been defined, we may proceed with the second layer in construction procedure of the PML modeling library. The PML port classes to represent the exchange

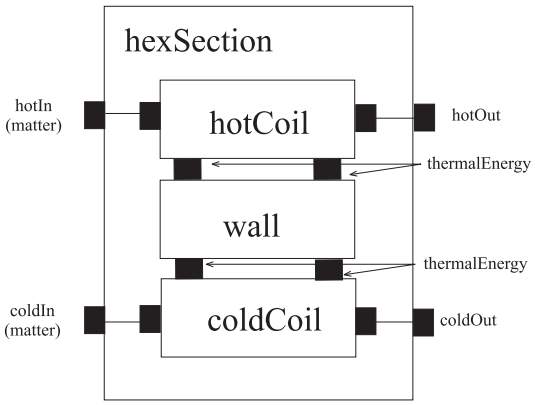

Figure 5.4. Topology of the heat exchanger section.

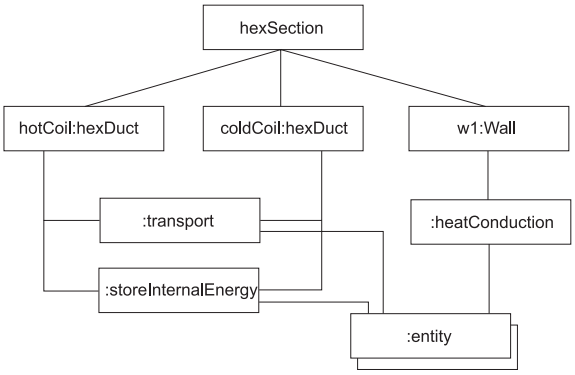

Figure 5.5. Modeling object relationships at the PML model of the heat exchanger section.

of matter and thermal energy are defined in first term:

```
(port [matterPort(matter)] [{w,p}]);
(port [thermalPort(thermalEnergy)] [{phi,thE}]);
```

The `matterPort` class is reused from examples in Chapter 4. It should be remarked that the port class declaration is independent from the phenomena aggregated into a model class. The `thermalPort` class represent the exchange of thermal energy at the connection point. This exchange phenomenon is described by the cause-effect relationship among the thermal effort `thE` (associated to the temperature at the connection point) and the heat flow `phi`.

According to the approach adopted in (Mattsson 1997), where the heat exchanger is divided into sections (see Figure 5.3), the model of a section should be defined. A heat exchanger section may be viewed in turn as a structure composed by two duct sections (the hot and cold streams) and a wall section. This topological model structure is shown in Figure 5.4. The heat exchanger section model is defined by the interconnection of two modeling objects (`hotCoil` and `coldCoil`) which are instances of the `hexDuct` model class and one modeling object (`W1`) which is an instance of the `wall` model class. The `hexDuct` model class will describe the matter transport and the thermal energy balance phenomena by aggregating the `matter` and `thermalEnergy` entity classes as well as the `matterTransport` and `storeInternalEnergy`

```
(duct [ductWithHeatCapacity]
   [{entities({thermalEnergy}),
     phenomena(storeInternalEnergy({matter,thermalEnergy}))
   }]
);
```

Listing 5.4. PML model class of a duct specialized by aggregating the thermal behaviour of the matter entity being transported.

phenomenon classes. The `wall` model class will describe the heat transfer by aggregating the `thermalEnergy` entity class as well as the `heatConduction` phenomenon class. The `hexDuct` and `Wall` modeling objects are connected by the `thermalPort` ports to represent the thermal energy interactions. Inner connections between the section and the coils matter ports are defined. This type of connection serves to propagate a submodel unconnected port to the equivalent port of the structured model. Hence, the heat exchanger model can be defined as the connection of the required number of sections through its matter ports. The object diagram of the `hexSection` model is shown at Figure 5.5.

The model of a duct able to represent the energy associated to the matter transfer can be defined by extending the duct model defined at Example 4.1 (see Listing 4.7). Note that, because of equation readability reasons, shorter names to the matter properties are given in this example (see Tables 4.15 and 5.1). By means of inheritance, the `storeInternalEnergy` phenomenon is aggregated to the `duct` model class as Listing 5.4 shows.

This model class does not declare the thermal energy exchange because it depends on the approximation adopted in this specific example. Two thermal energy ports are defined in the duct section according to the approach in (Mattsson 1997). Each of them will represent the thermal effort (related to the matter temperature) at both ends of the duct. The heat exchanger duct model can be defined by extending the `ductWithHeatCapacity` model in order to aggregate the thermal energy interface with the wall. Listing 5.5 shows the model class which will be used to build the heat exchanger section model. Note that this model class definition is a consequence of the approach adopted to avoid the use of partial differential equations.

```
(ductWithHeatCapacity [hexDuct]
   [{ports({thermalPort(PE1(thermalEnergy)),
            thermalPort(PE2(thermalEnergy))}),
     equations({
      DATA({ductVolume=0.001})
      PE1(thE(thermalEnergy))= div({P1(h(matter)),P1(cp(matter))}),
      PE2(thE(thermalEnergy))= div({P2(h(matter)),P2(cp(matter))})})
   }]
);
```

Listing 5.5. PML model class of a duct representing the matter thermal behaviour and the interaction with the wall through heat flow.

The model of the wall is also conditioned by the way in that the dimensional dependence of the thermal behaviour has been discretized to formulate it by means of differential equations. As Equation 5.4 describes, the heat transfer Φ through the wall depends on the thermal gradient ΔT_h and the thermal resistance R between the hot and cold surfaces. The formulation of ΔT_h and R shown bellow follows the approach given in (Mattsson 1997).

The thermal resistance is decomposed into four terms:

$$R = R_h + R_c + R_w + R_f \tag{5.5}$$

where R_h and R_c are the thermal contact resistances between the liquid in the ducts and the wall, R_w is the thermal resistance of the wall, R_f (zero valued in this example) is the thermal fouling resistance due to deposits and dirt on the wall. The contact resistances between a liquid and the wall are modeled as:

$$R_i = (h_i A_w)^{-1},\ i = \{h, c\} \tag{5.6}$$

where h_i is the surface coefficient of heat transfer (considered to be a constant in our example) and A_w is the area of the common wall between the ducts. The thermal resistance of the wall is calculated as:

$$R_w = \frac{d}{\lambda Y A_w} \tag{5.7}$$

```
(model [materialProperties]
  [{entities(material),
    equations({
      DATA({pRf=0,pLambda=387.48,pd=0.1,pAw=0.025,pY=1,hC=1e6,hH=1e6}),
      Rf(material) = pRf,
      lambda(material) = pLambda,
      d(material) = pd,
      Aw(material) = pAw,
      Y(material) = pY,
      R(material)=sum({Rc(material),Rh(material),Rw(material),Rf(material)}),
      Rc(material)=div({1,prod({hC,Aw(material)})}),
      Rh(material)=div({1,prod({hH,Aw(material)})}),
      Rw(material)=div({d(material),
                        prod({lambda(material),Y(material),Aw(material)})})
    })
  }]
);
```

Listing 5.6. PML model class declared to compute the thermal coefficient of a material.

where d is the thickness of the wall, λ is the thermal conductivity of the wall material and Y is a correction factor for the corrugation of the wall. The formulation of the material thermal properties is declared in the model class shown at Listing 5.6.

Note that this model aggregates no phenomena since we are only interested in the computation of the thermal properties from physical parameters which, in this example, are considered to be independent of any phenomenon occurrence.

The heat conduction phenomenon should be considered to complete the wall model class as well as the formulation of the thermal gradient across the wall. The adopted approach is to take ΔT_h as the log-mean temperature difference, defined as

$$\Delta T_h = \frac{\Delta T_1 - \Delta T_2}{ln \Delta T_1 / \Delta T_2} \tag{5.8}$$

where ΔT_1 and ΔT_2 are the thermal effort differences at the two ends of the duct.

```
(materialProperties [wall]
   [{phenomena(heatConduction({material,thermalEnergy})),
     entities({thermalEnergy}),
     ports({thermalPort(Ph1(thermalEnergy)),
            thermalPort(Ph2(thermalEnergy)),
            thermalPort(Pc1(thermalEnergy)),
            thermalPort(Pc2(thermalEnergy))}),
     equations({
        deltaT1=sum({Ph1(thE(thermalEnergy)),Pc1(thE(thermalEnergy))}),
        deltaT2=sum({Ph2(thE(thermalEnergy)),Pc2(thE(thermalEnergy))}),
        IF(EQ({prod({deltaT1,deltaT2}),0.0})) THEN
          (dTh(thermalEnergy)=div({sum({deltaT1,-deltaT2}),2}))
        ELSE IF(GT({abs(sum({deltaT1,-deltaT2})),
                    prod({0.05,max({abs(deltaT1),abs(deltaT2)})})})) THEN
          (dTh(thermalEnergy)=div({sum({deltaT1,-deltaT2}),
                                   ln(div({deltaT1,deltaT2}))}))
        ELSE
          (dTh(thermalEnergy)=
            prod({0.5,sum({deltaT1,deltaT2}),
                  sum({1,-prod({div({pot({sum({deltaT1,-deltaT2}),2}),
                                      prod({12,deltaT1,deltaT2})}),
                                sum({1,-div({pot({sum({deltaT1,-deltaT2}),2}),
                                             prod({12,deltaT1,deltaT2})
                                    })})
                           })
                     })
                })
            ),
        Ph1(phi(thermalEnergy))=div({phi(thermalEnergy),2}),
        Ph2(phi(thermalEnergy))=div({phi(thermalEnergy),2}),
        sum({Ph1(phi(thermalEnergy)),Pc1(phi(thermalEnergy))})=0,
        sum({Ph2(phi(thermalEnergy)),Pc2(phi(thermalEnergy))})=0
     })
   }]
);
```

Listing 5.7. PML model class of a wall where a heat transfer phenomenon occurs.

```
(model [hexSection]
  [{entities({hotMedium(matter),
              coldMedium(matter)}),
    ports({
      matterPort(hotIn(hotMedium)),
      matterPort(hotOut(hotMedium)),
      matterPort(coldIn(coldMedium)),
      matterPort(coldOut(coldMedium))
    }),
    submodels({
      hexDuctWater(hotCoil(hotMedium)),
      hexDuctWater(coldCoil(coldMedium)),
      wall(W1)
    }),
    connections({
      hotIn-hotCoil.P1,hotOut-hotCoil.P2,
      coldIn-coldCoil.P1,coldOut-coldCoil.P2,
      hotCoil.PE1-W1.Ph1,hotCoil.PE2-W1.Ph2,
      W1.Pc1-coldCoil.PE1,
      W1.Pc2-coldCoil.PE2
    })
  }]
);
```

Listing 5.8. Section of a heat exchanger model.

```
(model [heatExchanger]
  [{entities({hM(matter),cM(matter)}),
    ports({
      matterPort(hotIn(hM)),
      matterPort(hotOut(hM)),
      matterPort(coldIn(cM)),
      matterPort(coldOut(cM))
    }),
    submodels({
      hexSection({
        S1({hotMedium=hM,coldMedium=cM}),
        S2({hotMedium=hM,coldMedium=cM}),
        S3({hotMedium=hM,coldMedium=cM})
      })
    }),
    connections({
     hotIn-S1.hotIn,S1.hotOut-S2.hotIn,
     S2.hotOut-S3.hotIn,S3.hotOut-hotOut,
     coldIn-S1.coldIn,S1.coldOut-S2.coldIn,
     S2.coldOut-S3.coldIn,S3.coldOut-coldOut
    })
 }]
);
```

Listing 5.9. Topological model of a heat exchanger composed by three sections.

Formula 5.8 is badly conditioned if $\Delta T_1 \approx \Delta T_2$ so when $|\Delta T_1 - \Delta T_2| < 0.05 max(|\Delta T_1|, |\Delta T_2|)$ it is better to use:

$$\Delta T_h = 0.5(\Delta T_1 + \Delta T_2) \times \left(1 - \frac{1}{12}\frac{(\Delta T_1 - \Delta T_2)^2}{\Delta T_1 \Delta T_2}\left[1 - \frac{1}{2}\frac{(\Delta T_1 - \Delta T_2)^2}{\Delta T_1 \Delta T_2}\right]\right) \tag{5.9}$$

This approach is represented by the model class shown at Listing 5.7. The formulation of Equation 5.9 has not been included at the `heatTransfer` law class because of its dependence between ΔT_h thermal gradient and the number of thermal energy ports used in the approach adopted to model the wall section (it should be recalled that the law class formulation can not make explicit references to port classes). Note that ΔT_h is not computed directly from the effort properties

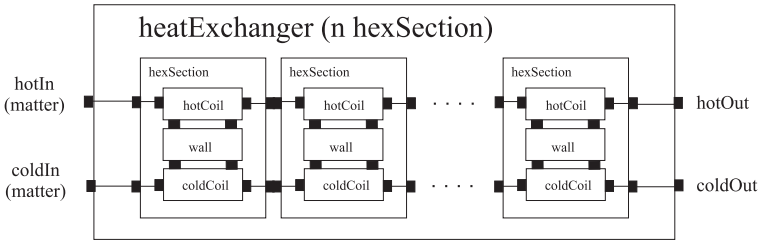

Figure 5.6. Heat exchanger PML model topology.

declared by the thermal port class and a particular combination of `thE(thermalEnergy)` at each of the four ports in the model is used instead. This type of problems will be treated in future releases of the language since a more generic solution will increase the law class reusability as well as it will reduce the number of equations formulated inside the model class to resolve certain cause-effect physical relationships.

Once the basic knowledge and the atomic model classes are defined, the remaining work is the topological specification of the heat exchanger section model and the heat exchanger model itself.

According to the structure illustrated at Figure 5.4, the matter and thermal energy paths between the duct and the wall in a section is defined by the model class at Listing 5.8. Finally, Listing 5.9 shows the model of the heat exchanger composed by the interconnection of three sections (according to the topology shown at Figure 5.6). The heat exchanger mathematical model can be now generated by executing the following PMT command:

```
>analyze C5::heatExchanger(TS30(entities({hM=water,cM=water})))
```

The mathematical model is coded with the EcomsimPro language. The generated code is listed at Appendix C.3. ❑

The mechanism provided by PML to generate simplified simulation models from a predefined PML model class is based on the phenomena pruning. When a model class has been defined to represent accurately the system behaviour, it is quite possible that the model results too complex for many experimentation purposes where such a precise representation is not necessary. Since

the physical behaviour declaration in a PML model consists in the aggregation of phenomenon classes, the behaviour which is not of interest for the experimentation framework can be neglected by removing from the model the references to the unwanted phenomenon classes. The example bellow illustrates the behaviour pruning procedure.

EXAMPLE 5.2 – *Behaviour pruning of the heat exchanger model*

Once a PML model class is defined to represent a system physical behavior it can be reused to generate different simulation models. This example illustrates how to prune the unwanted behaviour declared in a predefined model class. The heat exchanger model developed at Example 5.1 will be used for this purpose by assuming that the thermal behaviour of the hot and cold streams may be neglected for certain experimental framework (e.g. the design of a flow rate controller). The thermal behaviour of the matter entities has been defined in terms of the `storeInternalEnergy` phenomenon class (see Listing 5.1).

The purpose of pruning a phenomenon is to achieve a simplification of the behaviour formulation. However, there is information about the physical behaviour which should not be lost in the simplification process. This is the role of the list of affected properties included in the phenomenon class declaration (`h`,`cp`, `rho` for matter and `thE` for thermal energy). The properties affected by the neglected phenomenon should be known to properly set the functional model since they may be referenced by the model equations and by other coupled models even when the phenomenon is neglected. This list is the mechanism to inform to the physical analysis procedure which of the `matter` and `thermalEnergy` properties are affected by the `storeInternalEnergy` phenomenon in a model where it is aggregated (this will explained in Section 5.2). They will be considered as boundary conditions in the experimentation since no behavior related to the neglected phenomena will be formulated. The PMT provides the user with a check box dialog to select the phenomena which are of interest for his experimental framework (see Figure 5.7) and remove the `storeInternalEnergy` phenomenon.

The mathematical model resulting from the duct model class simplification is shown at Listing 5.10. Note that it just describes the matter transport phenomenon, which has been con-

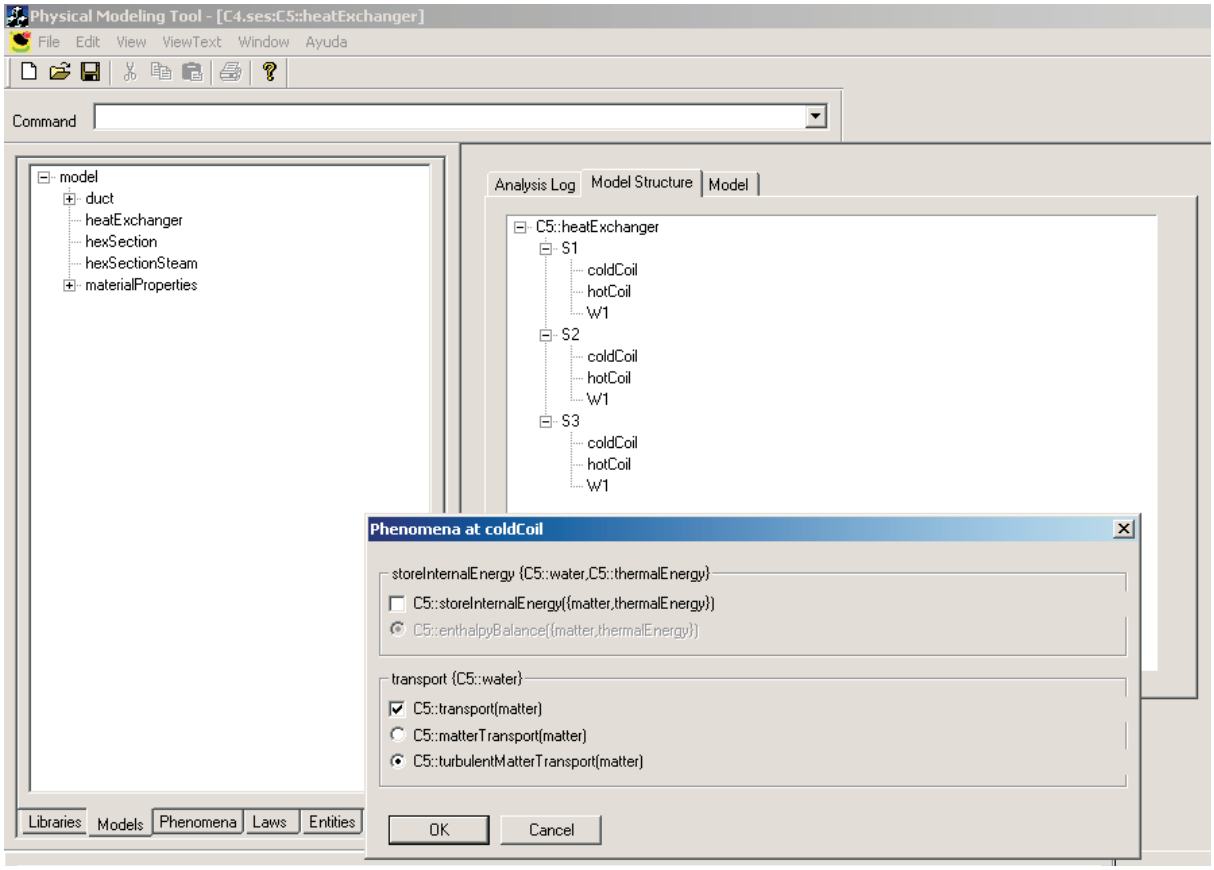

Figure 5.7. Pruning of the neglected behaviour and selection of the law.

sidered the behavior of interest for the experimentation purposes. By comparing the simplified duct section mathematical model with the mathematical model in Appendix C.3, it may be observed that it contains one less derivative variable. Hence, the reduced version of the three section heat exchanger model avoid the evaluation of six derivative variables. Under the same numerical conditions, the whole heat exchanger simulation time takes twice the reduced model simulation time. The reduction of the simulation time may be a very important issue in certain applications such as real time simulators. ❑

```
COMPONENT   C5HexDuctWater --IS_A C5::hexDuct
  PORTS
     IN C5WaterPort P1 -- On Entity: C5::water
     IN C5WaterPort P2 -- On Entity: C5::water
     IN C5ThermalPort PE1 -- On Entity: C5::thermalEnergy
     IN C5ThermalPort PE2 -- On Entity: C5::thermalEnergy
     IN waterProperties PAP_boundaryProp -- added by physical analysis procedure
  DATA
  -- Data declared by laws
     REAL cp=4180
     REAL rho=1000
  -- Data declared by local equations
     REAL Rfluid=0.025
     REAL ductVolume=0.001
  DECLS
  -- Entity properties referenced by local equations
     REAL C5Water_w
     REAL C5Water_cp
     REAL C5Water_dP
     REAL C5Water_h
     REAL C5Water_rho
     REAL C5Water_v
  CONTINUOUS
  -- PHENOMENON: C5::storeInternalEnergy({matter,thermalEnergy})
  --        LAW: C5::enthalpyBalance({matter,thermalEnergy})
  -- NEGLECTED
  -- PHENOMENON: C5::transport(matter)
  --        LAW: C5::matterTransport(matter)
     C5Water_w=Rfluid*C5Water_dP
     P1.C5Water_w+P2.C5Water_w=0.0
  -- Local model equations
     PE1.C5ThermalEnergy_thE=P1.C5Water_h/P1.C5Water_cp
     PE2.C5ThermalEnergy_thE=P2.C5Water_h/P2.C5Water_cp
     C5Water_dP=P1.C5Water_p-P2.C5Water_p
     C5Water_v=ductVolume
     C5Water_w=P1.C5Water_w
  -- added by physical analysis procedure
     PAP_boundaryProp.C5Water_h = C5Water_h
     PAP_boundaryProp.C5Water_cp = C5Water_cp
     PAP_boundaryProp.C5Water_rho = C5Water_rho
END COMPONENT
```

Listing 5.10. Mathematical model of a the duct section neglecting the water enthalpy dynamics.

As the Example 5.2 has shown, one of capabilities of PML to produce reduced simulation models is the pruning of the irrelevant behaviour represented by the model. However, this is not only manipulation capability required to adapt a ready-made PML model class into different experimentation frameworks with the desired adequacy level. Next section describes how to deal with the simulation model adequacy.

5.1.2. Setting the adequacy level

The concept of model adequacy has been defined as a combination of accuracy and efficiency. According to Definition 1.4, an adequate model should give the simplest formulation able to represent the phenomena of interest with enough accuracy. Different simulation models can be generated from a PML model class by setting adequacy level suitable to the simulation purpose (see Figure 5.1). This second possible model adaptation is a consequence of the multi-faceted nature of the phenomenon class. Multi-faceted phenomenon classes (see Semantic Rule S6) permit the selection of the behaviour formulation which better suits with the desired model adequacy. Therefore, the adequacy of the simulation model can be achieved by properly combining the behaviour pruning mechanism together with the selection of the law formulation which better suits with the desired accuracy. For example, a linear formulation of the matter transport has been selected for the duct section mathematical model shown at Listing 5.10. The selection is made through the check box shown at Figure 5.7.

The PML model classes representing the system at the topological level can be manipulated to obtain, in an easy way, different simulation models which are adequate for the simulation purpose, i.e., they are accurate and efficient enough for the model application. The decision about how the model should be simplified is taken by the user according to his simulation interest. He is helped in this task by the language expressiveness since he makes the model manipulation operating with representation structures which explicitly describe the physical behavior he wants to neglect or formulate in certain way.

5.1.3. Extending the model reusability

As it has been stressed in Section 1.2, different scopes to model reusability were considered in the PML modeling framework design. In order to be realistic, certain experimental framework and adequacy level should be defined when a modeling library is built from scratch. Therefore, the possibility to extend the reusing capabilities of a predefined model along the experimentation framework and adequacy coordinates becomes a modeling features of major relevance. The extension of the reusability of a ready-made PML model class can be achieved by the addition of new modeling classes able to afford system simulation applications which were not considered at the moment of designing a modeling library. Obviously, the addition of new modeling classes should no have side effects on the already defined classes or, at least, these effects should be minimized. Different ways to extend the model application reusability are provided by the PML language:

- Extension of the experimental framework:
  - Addition of phenomenon and law classes to operate with new entities. No side effects on predefined model classes are expected since the reusability extension can be achieved by means of the model parameterization (see Semantic Rule S25. For example, the heat exchange model developed at Example 5.1 can be extended to operate with steam as the hot stream (instead of water) by adding the law classes which properly formulate the `storeInternalEnergy` and `transport` phenomena.
  - Aggregating new phenomenon classes to a predefined model class. May have side effects on the modified model class. Should not have side effects on the structured (composite) models where the modified model class is a component if the new and old model versions are polymorphic. For example, the wall model defined at Example 5.1 could be extended by aggregating the heat capacity phenomenon which was not considered in first term. This change would also imply the aggregation of a new law class to formulate the heat conduction phenomenon (see Listing 5.1). However, no side effects on the rest of model classes in the heat exchanger are expected.

- Extending the adequacy coordinate: adding new law classes to predefined phenomenon classes (no effects on predefined model classes because of the PML phenomenon class is multi-faceted).

5.2. Simulation model generation

The fourth layer (model experimentation) in the PML modeling approach (see Figure 4.1) involves the simulation model generation. In first term, the physical analysis procedure determines the proper physical phenomenological structure of a *PML topological model* (see Definition 3.1). The resulting structure is represented by the *PML functional model* (see Definition 3.2), which was introduced in the Section 3.5 and which is described with more detail here. The functional model generation is a step previous to the mathematical formulation (the mathematical model according to Definition 1.11) and the later simulation model generation (the simulation model according to Definition 1.12). Previously, the PML topological model has been semantically analyzed (see Figure 5.8). This analysis validates the model declaration by certifying that the language semantic rules are fulfilled in the model class declaration.

5.2.1. The Physical Analysis Procedure

The *Physical Analysis Procedure* (*PAP*) is used to state the functional model describing the phenomenological structure of the topological model, i.e., it translates the system specification at the topological level into the phenomenological level (see Definitions 1.9 and 1.10). The functional model can be generated once the experimental framework and adequacy level have been set (as it is discussed in previous section) by executing the menu option shown in Figure 5.9.

The main task performed by *PAP* is the setting of the system aggregated physical behaviour as it is specified at the topological level (PML topological model) by resolving the physical causality among its components. This task is performed in two main steps:

- Local behaviour setting: determination of the behaviour locally declared by the PML model classes (only the phenomenon and laws classes selected by the user will be mathematically formulated).

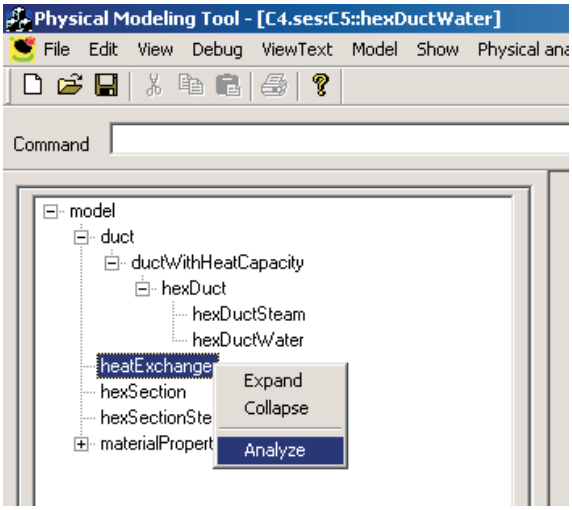

Figure 5.8. PMT analyze menu: performs the semantic analysis of a PML topological model.

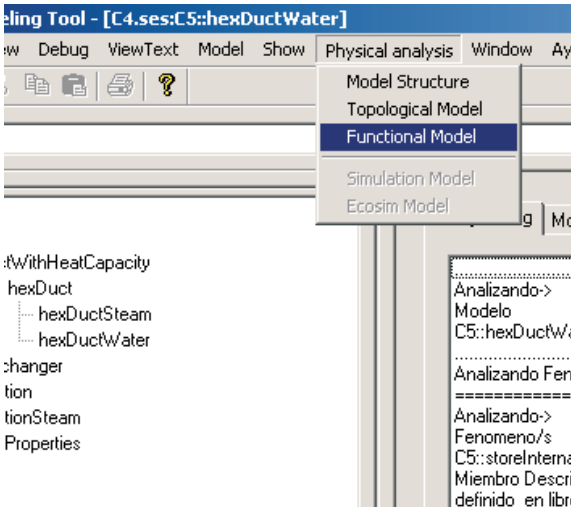

Figure 5.9. PMT functional model menu: generates the functional model of a PML topological model.

- Physical interaction setting: resolution of the physical interactions among the coupled model objects from the analysis of the entity exchanges declared by their port connection topology.

The main differences of *PAP* with respect the computational analysis procedure (*CAP*) at the equation-based object oriented modeling languages are the mechanisms used to obtain the aggregated behaviour of a structured model. The equation-based mechanisms are the coupling equations (equality and sum up to zero equations) derived from the connection of the variables declared at the model interface. The PML mechanisms are based on the two communication rules defined by PML: communication between phenomenon, entity and law classes (Semantic Rule S19) and communication between model classes (Semantic Rule S18).

Local behaviour setting

This step of the *PAP* is performed during the semantic analysis of the model. The physical behaviour represented by the model is collected and sorted. During the behaviour collection, all the sections in a model class declaration (phenomena, entities, ports, submodels, connections and

equations) are retrieved by tracking the model class inheritance path. The collected behaviour is then sorted according to the inheritance hierarchy and each modeling class is analyzed to validate its applicability within the model class. PMT organizes the representation of the physical behaviour according to the dynamic structures shown at Figure 5.10. These structures serve to handle the declared and, likely, inherited local behaviour . They represent the phenomena and the entities, by considering the behaviour redefinition mechanisms provided by inheritance (see S22, S20), redefinition (see S24) and parameterization (see S25, S26). According to this mechanisms, phenomena can be propagated to entities added by inheritance (entity overloading) and/or to entities redefined (entity overriding).

A list *L1* is created from the phenomena declared (aggregated) in the model class. A new entry is inserted at the list *L1* of applicable phenomena for each phenomenon aggregated in the model (e.g. `transport(matter)`).

A second list *L2* is created for each *L1* entry from the communication between phenomenon, entity and law classes (Semantic Rule S19) by tracking the inheritance hierarchy of the entities aggregated in the model class. Each node at this list has in turn two more lists: a list containing the pointers to the phenomenon classes which are applicable to the declared entities and a list containing the propagated entities (according to Semantic Rules S22, S25 and S26). The involved entity (or entities in the case of multi-entity phenomenon) causes a new entry at the list *L2* (e.g. `matter`). The PML phenomenon classes which can be applicable are searched at the phenomena tree (constructed when a modeling library is opened) and are added to the new entry in *L2*. Up to this point, the physical behaviour declared by the model has been identified and all the involved entity and phenomenon classes have been validated (semantically analyzed).

The entities aggregated in the model are propagated once every phenomenon object at the phenomena section has been processed. Entity overloading and/or overriding history is determined by following the model inheritance hierarchy. The resulting structure is then processed by starting at the entities declared by the most ancient model and continues to the end of the model definition hierarchy. Each time an entity class is processed, it is determined at the phenomena tree if there exists an specific phenomenon predefined for the entity. If that holds, a

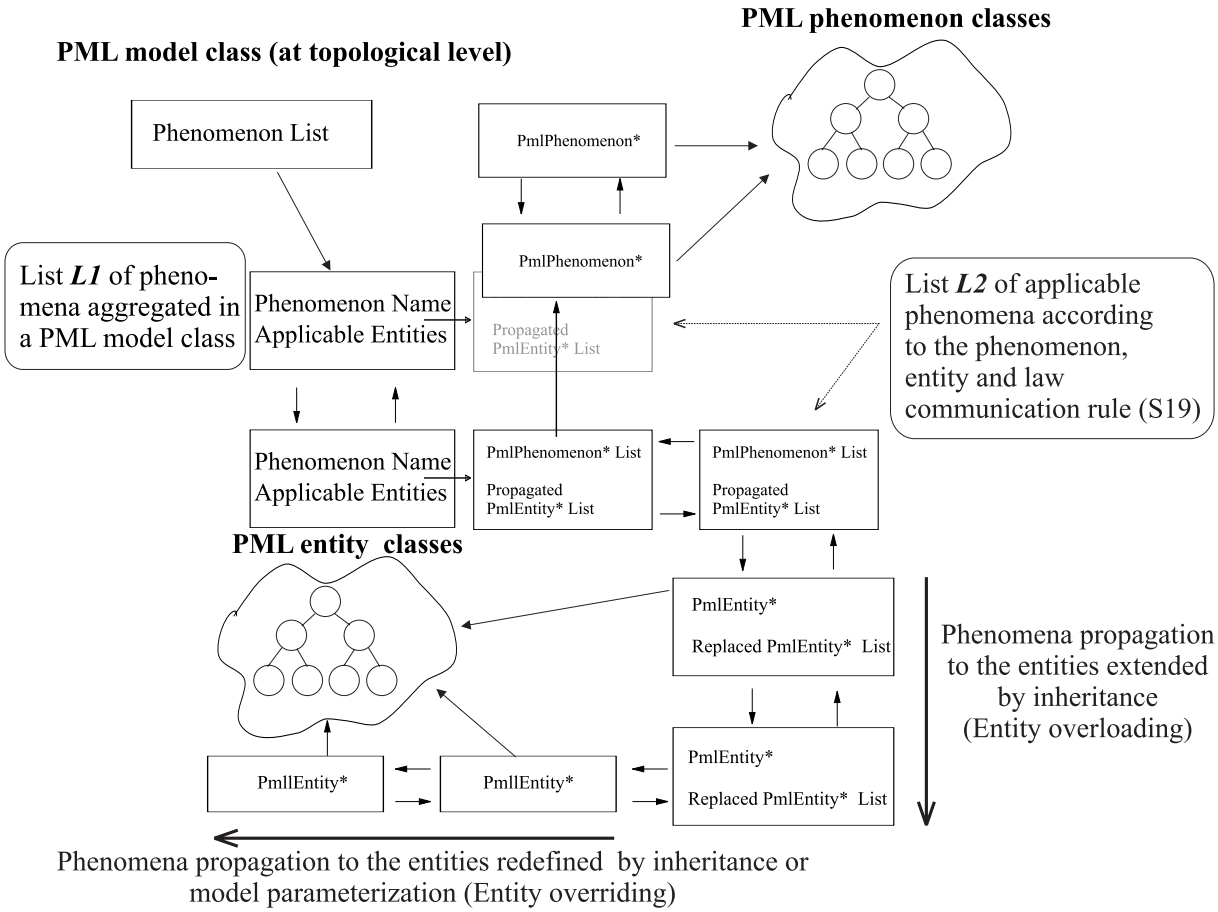

Figure 5.10. Dynamic structures used by PMT to represent and manipulate the physical behaviour declared by a PML model class.

new entry is created at the *L2* list with the involved entity establishing the pointers with the found phenomena. This is the mechanism defined PML in order to specialize the behaviour of a model by means of inheritance. If no specific phenomenon classes are found, the applicable phenomena determined at the previous stage are extended to the new entity class. Consider that, for example, the `transport(matter)` phenomenon and the entity `matter` are aggregated in a model. This model is then specialized by means of inheritance and the entity `water` redefines `matter`. If no specific phenomena are predefined to represent the transport of water, the `transport(matter)` phenomenon and their related law classes will be reused by replacing the references to the `matter` entity class with references to the `water` entity class.

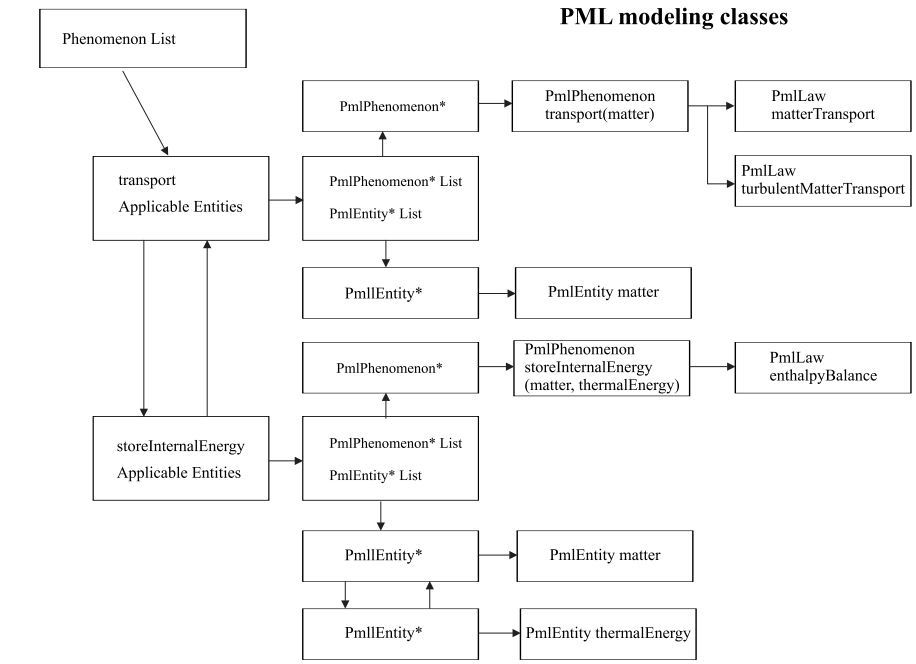

Figure 5.11. Dynamic structures used by PMT to represent the physical behaviour declared by the `hexDuct` PML model class.

Consider the `hexDuct` model shown at Listing 5.5 to illustrate the whole local behaviour setting procedure. This model inherits from the `ductWithHeatCapacity` (see Listing 5.5) which, in turn, inherits from the `duct` model (see Listing 4.7). The resulting behaviour described by the inherited phenomena is:

```
transport(matter)
storeInternalEnergy({matter,thermalEnergy})
```

being `matter` and `thermalEnergy` the involved entities. Two entries at list *L1* are created to register the aggregated phenomena `transport` and `storeInternalEnergy`. Two list *L2* are created at both entries. The first one contains one node for the `transport` PML phenomenon class defined for the `matter` entity class found in the modeling library. The second list *L2* contains one node for the `storeInternalEnergy` PML multi-entity phenomenon class defined for `matter` and `thermalEnergy` (see Listing 5.1). Figure 5.11 shows the representation structure

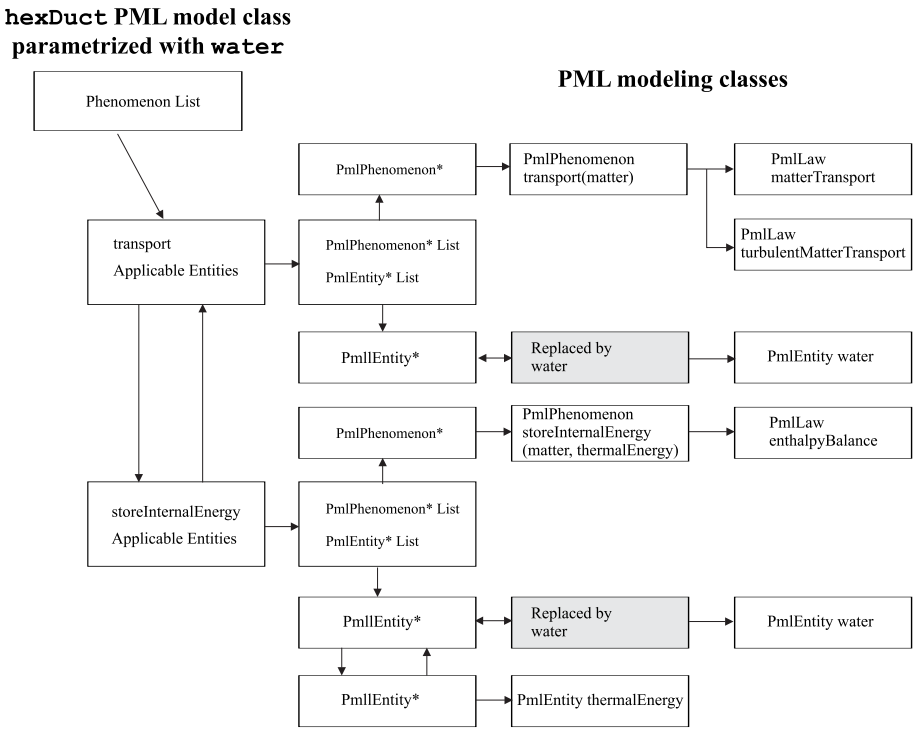

Figure 5.12. Dynamic structures representing the physical behaviour of the `hexDuct` PML model class parameterized with water.

in PMT of the `hexDuct` model class. Note that the communication rule S19 is applied to set the link with the predefined PML law classes which can be reused to formulate the physical behaviour represented by the `hexDuct` model class.

When the `hexDuct` model is reused to define the heat exchanger model, the `matter` entity is redefined by means of model parameterization (see Semantic Rules S25 and S26). Hence, in this reusing context the declared phenomena should be applied to the `water` entity. Model parameterization is used when a model class instance is created either to generate a simulation model or to define a component in an aggregated model (e.g. see Listings 5.8 and 5.9). Model parameterization has similar effects to the redefinition mechanism in model inheritance (with the exceptions discussed at Section 4.6). Figure 5.12 shows the representation structure in PMT of the `hexDuct` model class when an instance is created by redefining `matter` with `water`.

```
-- PHENOMENON: C5::storeInternalEnergy({matter,thermalEnergy})
--        LAW: C5::enthalpyBalance({matter,thermalEnergy})
   C5Water_rho=rho
   C5Water_cp=cp
   C5Water_H'=P1.C5Water_w*P1.C5Water_h+P2.C5Water_w*P2.C5Water_h+\
              PE1.C5ThermalEnergy_phi+PE2.C5ThermalEnergy_phi
   C5Water_H=C5Water_rho*C5Water_v*C5Water_h
   C5Water_h=C5Water_cp*C5Water_T
   C5ThermalEnergy_thE=C5Water_T
-- PHENOMENON: C5::transport(matter)
--        LAW: C5::matterTransport(matter)
   C5Water_w=Rfluid*C5Water_dP
   P1.C5Water_w+P2.C5Water_w=0.0
```

Listing 5.11. EcosimPro code of the `hexDuct` local behaviour formulated from the PML modeling classes.

Listing 5.11 shows part of the mathematical model generated from the structure at Figure 5.12 (the complete mathematical model is listed at Section C.3). The `matterTransport` law has been selected in this case to formulate the `transport` phenomenon (note that this phenomenon is declared as multi-faceted). The declared behaviour is expanded to formulate the water transport and internal energy dynamics.

The model user can easily identify each equation at the simulation model with the formulation of a physical phenomenon. The simulation model readability becomes a very important issue in order to get acquainted with the relationship between the PML model class and its usage for experimentation purposes when the simulation model is generated in a particular reusing context. For example, when the generated simulation model needs to be simplified as it was discussed in Section 5.1. The procedure of behaviour pruning and law class selection is performed over the local behaviour representation structure shown at Figure 5.10. The proper simulation model can be achieved in a small number of iterations till the desired adequacy level formulating the phenomena of interest is reached.

Once the local behaviour of each model component in a structured model has been set, *PAP* will proceed with the determination of the aggregated behaviour by setting the proper physical interactions derived from the entity transfers among model components.

Physical interaction setting

The functional model structure is determined by the physical interactions which derived from the coupled models because of the exchange of the entity class instances through port connections together with the occurring phenomena of interest. Hence, the functional model structure depends on the topological model structure as well as on the experimental framework defined for its usage. *PAP* uses the following physical knowledge in order to set the functional model structure:

- The entity exchange phenomenon represented by the model port connections (see Semantic Rule S18).
- The set of entity properties affected by the phenomena of interest aggregated in the model. This is derived from the local behaviour stated a the first step of *PAP* .
- The entity scope rule (see Semantic Rule S4). This rules determines which is the scope of a reference made to an entity property.

The functional model is represented by means of a graph where the nodes are related with the phenomena occurring at each component of the topological model and the edges represent the physical interactions among nodes. The functional model edges consist in a set of the variables required to represent the physical interactions which are detected among two functional model nodes. The node interface is constituted by the variables contained at all the connected edges. Figure 5.13 shows the relationship between the topological model decomposition structure and the functional model structure. The structured topological model being processed to generate the simulation model is represented at the tree top level (level 0). This model is composed by the connection of submodels (PML model classes) which are represented at the next level. Each submodel is recursively decomposed into submodels until atomic models are reached, giving rise

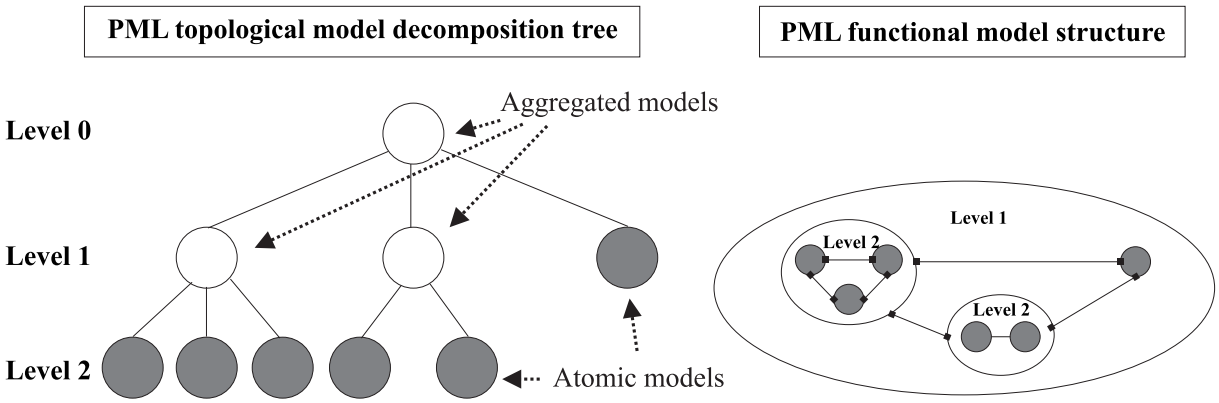

Figure 5.13. Decomposition structure of the topological model and its corresponding functional model representation structure.

to the decomposition tree shown at Figure 5.13. The physical interactions among submodels must be resolved at each tree level since the port connection structure is always defined at a unique level. That is, the submodel ports of a model are not accessible for the model connection with other models. For example, the link `hotIn-S1.hotCoil.P1` is not allowed in a PML connection sentence. That was the reason for defining the inner connections which can be observed at the `hexSection` and `heatExchanger` PML model classes at Example 5.1. An inner connection is not interpreted as a coupling in the sense of the model communication rule S18. It is used to propagate an unconnected submodel port to one of the ports of the aggregated model (e.g., inner connections can be observed at the `hexSection` model topological view shown at Figure 5.4).

Consequently, the functional model is also structured, i.e., a functional model node representing a non atomic (sub-)model at a given level of the tree may contain a functional model representing the aggregated behaviour of its submodels. The functional model structure corresponding to the topological model decomposition tree may be observed at Figure 5.13. The number of nodes at the functional model equals the number of model components at the topological model. There is a pragmatic reason for this and is to preserve to a great extent the analogy between the structure of the mathematical model and the structure of the topological model.

Hence, the user may have a better understanding of the mathematical model when required (see Definition 1.11). What may likely differ from the topological model structure are the functional model node connections representing the physical interactions among the model components in the topological model. The physical interactions among coupled model are the consequence of the entity exchange through ports. The entity exchange may cause a one-to-one coupled model interaction, but also may cause interactions among model components which are not directly connected (this matter was discussed at Section 3.2). Two types of information structures can be associated to a functional model edge: the set of variables used to represent the entity exchange phenomenon and/or the set of variables used to represent the physical interactions appearing due to the entity exchange.

PAP begins the functional model generation at the top level of the topological model decomposition tree and proceeds to the lowers levels (the inner model components of the topological model). The functional model structure is created through this top-down path. Then, *PAP* starts resolving the physical causality at the atomic model classes and process back the model structure in a bottom-up way. The procedure executed by *PAP* to set the physical interactions is summarized by the algorithm shown at Listing 5.12. The heat exchanger model developed in Example 5.1 will be used to illustrate this procedure. The topological model decomposition structure is shown at the PMT `Model Structure` window (see Figure 5.7).

The functional model nodes are defined at the first procedure step by following the `heatExchanger` decomposition tree (top-down path). The interface definition of each functional model node (*fmN* from now on) starts at this step by establishing the variables which will be used to represent the entity exchange phenomena at each model port. These variables are obtained from the properties declared at the corresponding port class. According to the port classes defined in the modeling library developed in Example 5.1, the `w` and `p` properties will describe the `water` exchange and `phi` and `thE` the `thermalEnergy` exchange. The `w` and `phi` properties define the corresponding entity exchange (see Semantic Rule S11). According to this rule, the scope of cause-effect properties is bound to the point where the exchange phenomenon occurs, i.e., these properties can not be propagated with the entity stream.

1. *A functional model node is created for each model component of the topological model at the decomposition tree. The physical analysis is moved to the level n, being n the deepest tree level.*
2. *Physical interactions are resolved for each model component at the assigned level:*
   (a) *if the visited component is atomic then the functional model interface is created from the physical behavior declared by model component (phenomena classes and model component ports).*
   (b) *else the connection topology of each model component at this level is analyzed in order to derive their submodel physical interactions. The corresponding functional model node interface is created according to the model component ports.*
3. *The analysis is moved up on level in the decomposition tree.*
4. *if level > 0 then go to 2*
5. *Physical interactions are resolved for the root of the decomposition tree.*

Listing 5.12. Functional model generation procedure.

Once the functional model structure has been created, the physical analysis starts at the bottom level of the decomposition tree in order to determine the node connection which properly describes the aggregated behaviour. Instances of the `wall` and `hexDuct` atomic models model are found at this level. The local physical behaviour collected at the first main step of *PAP* is now used to determine which variables are required to represent the exchanged entity properties. Consider for example the `hexDuct` model (see Listing 5.5). This model has four ports declaring the exchange of `water` and `thermalEnergy`. The `water` and `thermalEnergy` set of properties affected by the aggregated phenomena and their corresponding selected laws is deduced from the phenomenon class declaration (see Listing 5.1). On the one hand, the *fmN* associated to this model component can be inquired by other model components about the values of this set of of properties as far as the entities are exchanged through the model port. On the other hand, the formulated local behaviour asks for different properties of the exchanged entities at the different ports: the `enthalpyBalance` law selected to formulate the `storeInternalEnergy`

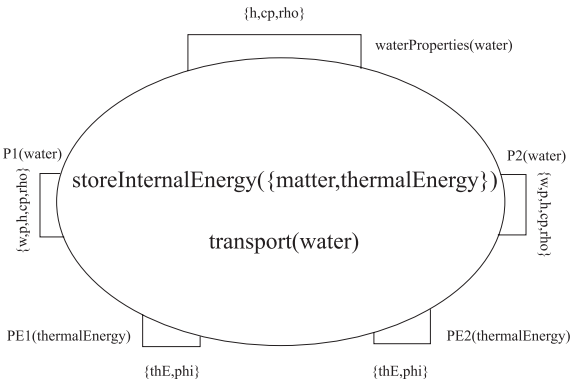

Figure 5.14. Example of a functional model node representing the `hexDuct` section model.

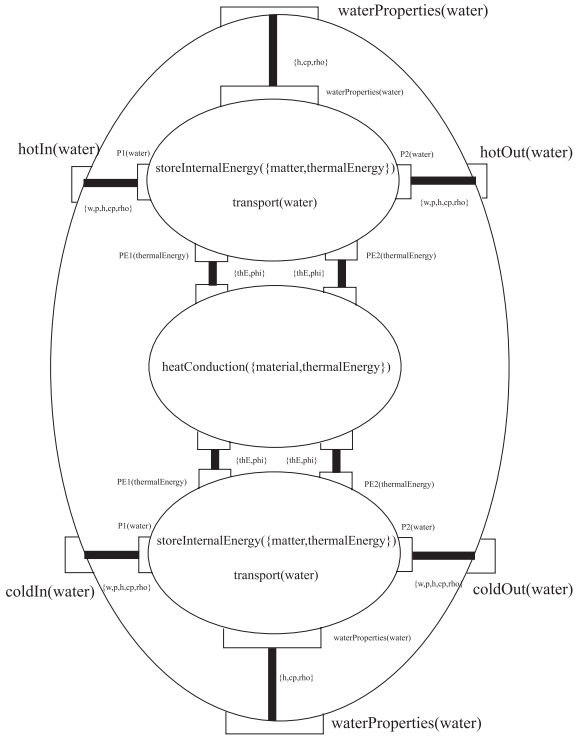

Figure 5.15. Functional model node representing the `hexSection` section model.

phenomenon refers to the water specific enthalpy `h`; the local equation to the specific heat `cp`, etc. The question is that, except for the cause-effects properties, the values of the entity properties referred at ports depend on the exchange sense. Hence, it should be distinguished if the entity flows inwards or outwards the port. This question can not be locally solved since it depends on the aggregated behaviour. Nevertheless, the *fmN* interface can be defined in order to solve the undetermined coupled behaviour once the aggregated behavior is analyzed in a further step of the procedure. It is concluded from this analysis that the {`w,p,h,cp,rho`} set of properties are required at `P1` and `P2` ports (`w` and `p` are cause-effect properties), the {`h,cp,rho`} is the set of `water` properties affected by local phenomena and {`phi,thE`} is the set of `thermalEnergy` properties required at `PE1` and `PE2` ports (they are cause-effect properties at these ports so they will not be carried with the entity stream). Figure 5.14 shows the *fmN* and its interface representing the `hexDuct` model. The same procedure leads to the `wall` *fmN*.

Once the analysis of all the model components at the bottom level is finished, the procedure moves up one level. Instances of the `hexSection` are found at this level. This is an aggregated model, so the objective here is to determine the couplings (edge connections) between the functional model nodes associated to its submodels. The functional model edges connections are defined according to the requirements imposed by the derived physical interactions with other nodes. The edges which are defined in first term represent the exchange phenomenon through connected ports. The cause-effect properties declared at the port classes are associated to these edges. The edges representing the properties carried with the entity stream are defined in second term by following the exchange paths. The `hexSection` model class connection structure describes the set of entity exchange paths between its submodels. These paths are used by *PAP* to resolve where an entity property referred in a port is affected by a phenomenon. This information has been registered in the previous procedure step at the functional model nodes associated to the submodels (`hotCoil`, `coldCoil` and `W1`). The physical interactions derived from the `water` exchange can not be resolved at the `hexSection` level and the unattached sub-node interfaces are propagated to the *fmN* according to the inner connections declared at the model class. Only the edges representing the `thermalEnergy` exchange can bet set. Figure 5.15 shows the *fmN* and its interface representing the `hexSection` model. The rest of functional model nodes at this level 1 are identical.

The procedure iterates once again arriving at the level 0 of the `heatExchanger`. At this level `water` is exchanged between the connected `hexSection` components. The `water` properties carried with the stream can be resolved now by following its exchange paths. It is known from the previous iterations of the functional model generation procedure that the `water` properties are affected by the physical behaviour associated to the sub-nodes (e.g. the water specific enthalpy) when the `water` flows outwards the `hexSection`. One section is connected to the next one by means of a `matterPort`, so when `water` flow outwards from the first section then flown inwards the second section. Therefore, the remaining unattached interfaces of the *fmN* associated to the `hexSection` model can be now linked by means of an edge representing the carried properties depending on the flow sense (note that the sense is determined by the sign of

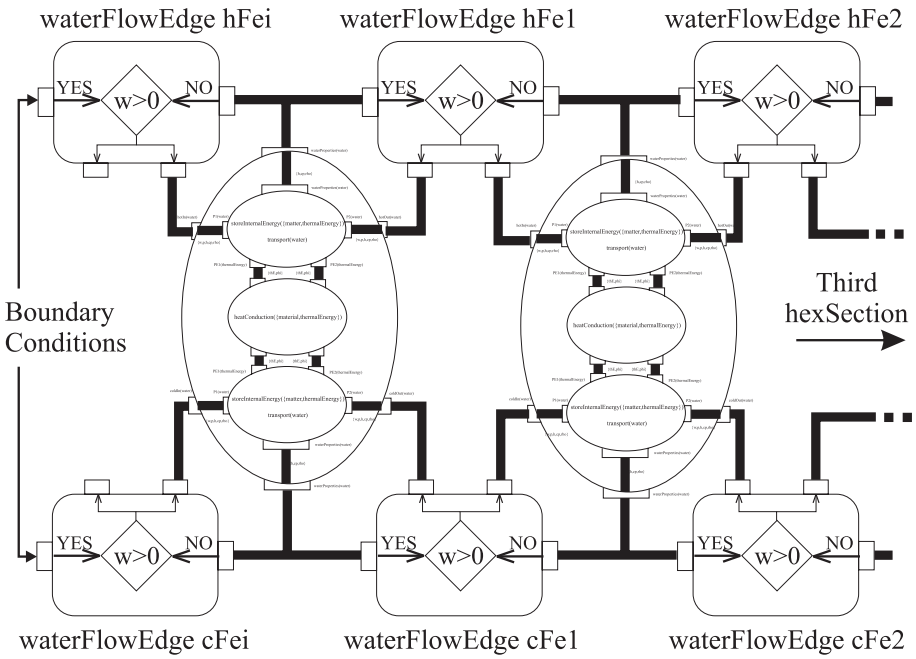

Figure 5.16. Functional model of the three sections heat exchanger developed at Example 5.1.

the flow property `w` declared at the port). Figure 5.16 shows the functional model derived from the `heatExchanger` model class. The heat exchanger functional model is represented in PMT by the dynamic structures (the graphs shown at Figures 5.14 to 5.16) found after the completion of this procedure. PMT offers the possibility to generate a debug file where significant information about the functional model generation procedure is registered.

There is a very important condition for this procedure success: the paths followed by an entity stream must be unambiguous, i.e., if *PAP* is tracing the properties of an entity through different model components, no more that one alternative places where the properties are affected by a phenomenon can be found. In other case, *PAP* can not resolve properly the physical interaction between the model components causing variations on the involved entity properties. That is the reason why no port multiple connections are allowed. Hence, physical devices such as a bifurcation pipe must represent by means of phenomenon classes how the entity properties are propagated through the flow split. This condition can not be formulated as a semantic rule, which would prevent from errors, since its validation can not be assured in every context.

5.2.2. The physical level to mathematical level translation

Next step in the simulation model generation is the mathematical formulation of the model aggregated physical behaviour (see Figure 5.1). The functional model generated by *PAP* is used to set the mathematical model structure.

Note that the functional model represents the modeled behavior in a modular way. The phenomenological structure represented by the functional model is dynamically obtained by *PAP*. This PML feature was defined as Dynamic Modularity (see Definition 3.3) and was discussed at Section 3.5. *PAP* detects the behaviour which is modular to each topological component (that is what each *fmN* represents) and then it resolves the physical interactions among the components (represented by the functional model edges). The edges define the minimum set of physical interactions since they are obtained from the requirements introduced by the physical behaviour contained at the nodes. Hence, the functional model fulfills the modularity criteria discussed thoroughly in Chapter 3.

The mathematical model can be easily formulated by means of an equation-based object oriented modeling language as a consequence of the functional model modularity. Current version of PMT generates EL code (EcosimPro Language), even future implementations will also generate Modelica. Using these well established commercial tools as the modeling language target at the non causal specification level is advisable because of their capability to perform the resolution of the computational causality required to translate the mathematical model into the simulation model. The basic principles of this modeling approach were introduced in Chapter 2.

Two basic main representation structures are provided by EL (analogous structures are found in Modelica): the EL `PORT` is used to define the mathematical coupling mechanism (mathematical interface) by defining the external or *manifest* variables; the EL `COMPONENT` is used to encapsulate the inner or *latent* variables, the behavior equations which formulate the dynamics of interest and the possible submodels and their connection topology. Hence, structured mathematical models can be built from these modeling components according to the functional model structure and connection topology.

The mathematical formulation procedure begins at the inner nodes of the functional model (associated to the bottom level of the topological model decomposition tree) and ends at the outer node (associated to the tree top level). This procedure is quite simple since it is almost a flat translation of the aggregated behaviour determined by *PAP* into its mathematical formulation according to the target language syntax. On the one hand, the dynamics of interest, represented by the functional model nodes, are formulated by considering that a node may be atomic (related to atomic model PML classes) or may point to a functional (sub)model (related to aggregated PML model classes). On the other hand, the mathematical interfaces are defined according to the information contained at the functional model edges.

Atomic mathematical model components

The inner nodes of the functional model are atomic, so they register the selected phenomena of interest. The mathematical representation of the atomic functional model nodes is based on the EL `COMPONENT`. Its declaration consist in the definition of the component mathematical interface and the formulation of the selected behaviour, i.e, the formulation of the dynamics of interest.

The declaration of the mathematical interface binds the property scope of the entity classes which are exchanged through PML port classes. They are derived from the functional model edges. The properties declared as flow in the PML port class will be defined as `SUM` at the EL mathematical port. The rest of properties used to describe the physical interactions among PML model class components will be defined as `EQUAL` at the EL mathematical port. Note that the `SUM` and `EQUAL` qualifiers correspond to the *Through* and *Across* external variable types defined in Section 2.2.2.

Consider for example the *fmN* shown at Figure 5.14. *PAP* has defined three type of node interfaces to connect the edges required to describe the physical interactions with other nodes. The mathematical `PORT` structures are declared directly from this node interfaces according to EL syntax. For instance, the `hexDuct` node interface to the `wall` node is declared in EL by the `PORT` structure shown at Listing 5.13.

```
PORT C5ThermalPort
   EQUAL REAL C5ThermalEnergy_thE
   SUM   REAL C5ThermalEnergy_phi
END PORT
```

Listing 5.13. EcosimPro mathematical port describing the thermal energy coupling.

```
COMPONENT   C5HexDuctWater --IS_A C5::hexDuct
   PORTS
     IN C5WaterPort P1 -- On Entity: C5::water
     IN C5WaterPort P2 -- On Entity: C5::water
     IN C5ThermalPort PE1 -- On Entity: C5::thermalEnergy
     IN C5ThermalPort PE2 -- On Entity: C5::thermalEnergy
     IN waterProperties PAP_innerProp -- added by physical analysis procedure
   DATA
     ...
END COMPONENT
```

Listing 5.14. EcosimPro code describing the `hexDuct` mathematical interface.

The rest of `PORT` structure declarations for the heat exchanger can be consulted at Section C.3. The mathematical model interface is defined by creating the proper mathematical `PORT` instances according to the node interface defined by *PAP*. For instance, the `hexDuct` node interface leads to the EL `COMPONENT` mathematical interface declaration shown at Listing 5.14.

As it can be observed, the mathematical model connectivity responds to the phenomenological structure and, in this case, differs from the topological structure (compare to the `hexDuct` PML model class shown at Listing 5.5).

The formulation of the dynamics of interest is made by translating the PML law class behaviour formulation according to EL syntax. PML model equations are also translated. Every generated expression is parsed in order to extract the inner or *latent* variables. An example of this translation procedure may be observed at the `CONTINUOUS` and `DECLS` sections of the mathematical model shown in Listing 5.10. The `C5HexDuctWater` and `C5Wall` mathematical models (EL atomic components) can be found in Section C.3.

```
COMPONENT   C5HexSection --IS_A C5::hexSection
   ...
   TOPOLOGY
     C5HexDuctWater hotCoil
     C5HexDuctWater coldCoil
     C5Wall W1
     CONNECT hotCoil.PE1 TO W1.Ph1
     CONNECT hotCoil.PE2 TO W1.Ph2
     CONNECT W1.Pc1 TO coldCoil.PE1
     CONNECT W1.Pc2 TO coldCoil.PE2
   ...
END COMPONENT
```

Listing 5.15. EcosimPro code describing the `hexSection` mathematical components and their connection topology.

Structured mathematical model components

The non atomic nodes formulation is based on the EL structured `COMPONENT`. A structured component declares the set of aggregated `COMPONENT's` and their connection topology. The aggregated components are instances of the mathematical models generated from the atomic nodes. Listing 5.15 shows part of the EL code corresponding to the *fmN* associated to the `hexSection` model class.

The connection topology defines the couplings between mathematical EL ports of the declared component instances. This topology responds to the edge connection structure of the non atomic node. As it has been illustrated by the heat exchanger functional model, there are two types of node connections defined by edges: physical interactions which do not depend on the exchange sense and connections which depend on the sense. It might be recalled that the sense of the entity exchange is represented by means of the flow property declared at the port classes.

The physical interaction among two functional model nodes is formulated as the connection of the associated mathematical ports when it does not depend on the exchange sense. This is the case of the thermal interaction between the ducts and the wall (see functional model at Figure 5.15) as the use of the `CONNECT` EL sentence in Listing 5.15 illustrates.

A more complex mathematical structure is required when the physical interaction represented by the edge depends on the exchange sense. In these cases, the external variables used to represent the properties carried with the flow must be duplicated in order to consider when the exchanged entity flows inwards or outwards the connected nodes. This problem is taken into account when the functional model node interfaces are created (it has been discussed in Section 5.2.1). The solution adopted when using EL is the definition of a component where the effects of the flow sense change over the carried properties are formulated. Listing 5.16 shows the component code used to represent the physical interactions among the `hexSection` *fmN* in the `heatExchanger` functional model (see Figure 5.16). This component acts like a switch between the connected water streams. When the water flow sense is positive, the mathematical port (instances of `C5WaterPort`) representing the stream properties takes the values of the `water` leaving the section to the left (instances of `waterProperties` mathematical port). The opposite assignment is made when the water flow sense changes.

Note that neither the PML law class formulation nor the equations declared at the PML model classes should assign the values at a PML port of the properties of the exchanged entity which are carried with the stream. As it was discussed in Chapter 3, a free context reusable model can not make reference to physical behaviour happening outside the system it represent (see Example 3.1). This is not necessary anyway, since this information is deduced by *PAP* from the properties affected by the aggregated phenomena together with the implicit exchange phenomenon defined by the PML port class connection at the topological model. This type of physical interactions is then properly formulated by the mathematical model as the heat exchanger example has illustrated.

The mathematical model could be significantly simplified if a fixed flow sense can be assumed. Actually, the benefits of this simplification have influence on the simulation model efficiency since the computing time can be reduced if no state events have to be detected to propagate the effects of the flow sense change. Current version of PMT does not support this type of simplification, although the future research related to the generation of efficient simulation models will contemplate it (see Section 6.2).

```
COMPONENT waterFlowEdge
  PORTS
    IN C5WaterPort PA
    IN C5WaterPort PB
    IN waterProperties PAprop,PBprop
  DECLS
    REAL C5Water_h
    REAL C5Water_cp
    REAL C5Water_rho
    REAL alpha
  CONTINUOUS
    alpha = ZONE (PA.C5Water_w>=0.0 TOL 1e-4) 1 OTHERS 0
    C5Water_h = alpha*PAprop.C5Water_h+(1-alpha)*PBprop.C5Water_h
    C5Water_cp = alpha*PAprop.C5Water_cp+(1-alpha)*PBprop.C5Water_cp
    C5Water_rho = alpha*PAprop.C5Water_rho+(1-alpha)*PBprop.C5Water_rho
    PA.C5Water_h = C5Water_h
    PA.C5Water_cp = C5Water_cp
    PA.C5Water_rho = C5Water_rho
    PB.C5Water_h = C5Water_h
    PB.C5Water_cp = C5Water_cp
    PB.C5Water_rho = C5Water_rho
    PA.C5Water_p = PB.C5Water_p
    PA.C5Water_w+PB.C5Water_w=0
END COMPONENT
```

Listing 5.16. EcosimPro code of the component used to describe the `water` properties carried by the flow among the heat exchanger sections.

5.2.3. Resolution of the computational causality

The causal explanation is obtained once the non causal mathematical model has been generated. The system specification at this level should have the computational structure which properly describes the cause-effect physical relationships in the formulated formulated behaviour. This causal explanation is obtained by means of an external tool able to perform the formula manipulation required to solve the computational causality. EcosimPro is used in current version of PMT. This tool translates the system specification at the non causal level generated by PMT into the specification at the causal level where the simulator relation is fulfilled.

Some very well known problems may arise during this translation procedure. This problems are related to the model mathematical structure and may lead to a high index DAE or simultaneous systems of non linear equations. Even present tools are able to solve these problems (numerically and/or symbolically), the model user may be disconcerted by their presence. Some of this problems may be avoided by using the physical knowledge represented by the PML modeling classes. This feature is discussed in other author works such as (Ramos 1995, Ramos *et al.* 1995, Ramos *et al.* 1998*a*).

5.3. The PMT modeling tool

PMT is the modeling tool developed to process the modeling libraries developed with PML. PMT has been developed with C++ and the GUI is based on the Windows platform in current version (Visual C++ and MFC).

Some of the PMT functional issues have been introduced at previous sections of this chapter. This section describes the main options to operate with a PML modeling library. A deeper explanation is left to the *PMT user's guide* (in elaboration process at the moment of writing this document).

Figure 5.17 shows the PMT GUI. The file menu is used to open, parse and process the PML modeling libraries. A PML library is a plain text file containing the modeling class declarations. Only syntax analysis is performed when the library is opened, reporting to the user the possible errors present in the modeling class declarations. The modeling class hierarchies are constructed during the library processing. They are represented by means of five graphical trees, one for each base PML class. The trees are shown at the left frame once a library is opened and parsed (see Figure 5.17).

The semantics analysis of a modeling class is performed by means of the PMT `analyze` command. This command may be executed either by selecting a node in the hierarchy trees with the mouse right button (see Figure 5.8) or by using the command line placed bellow the menu toolbar (see Figure 5.17). Every PML class can be analyzed at user demand. Possible semantic errors (violations of the related semantic rules) can be separately detected and corrected for each

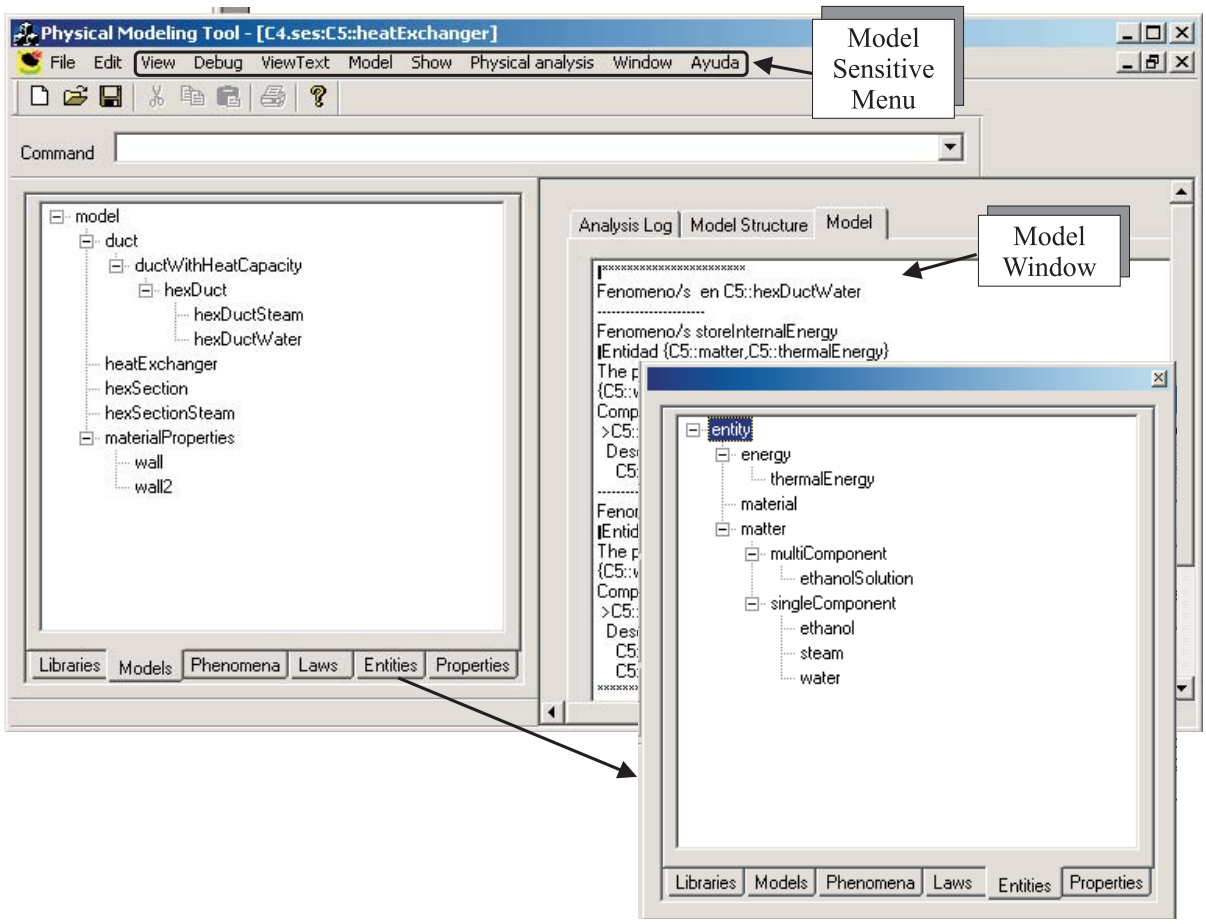

Figure 5.17. Heat exchanger PML model and entity class hierarchies.

modeling class. It should be noticed that if a modeling class semantic analysis has dependencies with other modeling classes, the last will be also analyzed automatically. For example, every involved entity classes and related law classes will be analyzed if the user executes the analyze command with a phenomenon class.

The generation of the simulation model starts when the analyze command is executed with a PML model class. A child window is created on the right frame each time a model class instance is created. Any operation on the processed model class is performed through this window and through the sensitive menus which are activated at this moment (see Figure 5.17). The main operations have been described at previous sections (e.g. model behaviour pruning, law selection

or functional model generation). Additionally, the user may obtain relevant information about the processed model class in this window. Examples are the result of the semantics analysis, visualizing the graphical representation of the model class decomposition tree or to list the applicable phenomenon and law classes for the aggregated or parameterized entities.

6

Conclusions and Future Research

During the last decade, very much research effort has been put in design of modeling methodologies able to minimize the cost of the model development task. Automating the modeling process has been a common objective in many of the present modeling tools. The object oriented methodology is a widely accepted approach to achieve the modeling automation by means of reusability. Several current modeling environments have adopted this approach by defining representation structures, which in some approaches can be assimilated to the concept of class, able to encapsulate by means of mathematical interfaces the non causal equations representing the system dynamics. However, encapsulation is not by itself the means to assure the model reusability.

"Encapsulation for objects, a valid technique to hide complexity, does not sufficiently support model reuse, since complying interfaces are only a necessary condition for reuse. More often, the implementation of a model decides on possible reusability in a new context." [1]

The modeling environment reported in this thesis emerges from the need to improve the object oriented modeling mechanisms which can be used to support a guided and automated modeling procedure. The reasons to justify a new modeling language have been thoroughly treated in this document. PML was originally conceived as a layer of physical knowledge built on top of an equation-based object oriented modeling language, such as Modelica or EcosimPro,

[1] (Marquardt 1996)

in order to overcome the limitations to reusability shown by these tools. However, the limitations shown by the mathematical formalism to represent the physical knowledge required for a free context model reusability led us to the definition of the current PML language. The burden to come up with a satisfactory solution is thus removed from the modeller and left to the designer of the modeling tool.

6.1. Conclusions

This work has reported the design and implementation of PML-PMT. PML is a modeling language conceived to embody the physical knowledge required to represent the behaviour of a physical system. Some important conclusions and contributions can be drawn from the way to the final result:

- The language PML defines a new dynamic modeling formalism designed to extend reusability both in the model construction concerns and in the model usage concerns. The defined formalism supports the explicit representation of the physical knowledge required to analyze models from a physical point of view. The physical knowledge makes a difference between system behaviour representation (description of the system physical behaviour) and system dynamics representation (mathematical formulation of the phenomena of interest). PML has been designed to fit with the Object Oriented paradigm. Recursive structured model development is supported by means of reusability.

- PML defines a modeling framework where the system can be specified at the topological level by connecting the models of the system components in the same way as they are physically connected. The system topological model can be built without taking into account its application purpose, focusing its construction on the representation of all the relevant information about the physical behaviour of the system. This specification, which responds to the modeller demands of easy while robust model construction methods, is automatically manipulated in order to generate the simulation model, which is the most simplified representation of the system dynamics that preserves the experimental frame-

work and responds to the simulator relation. This translation is performed by the *Physical Analysis Procedure*.

- There are two intermediate steps in this translation procedure. These steps responds to the phenomenological structure and to the non causal explanation system specification levels. The phenomenological structure is derived from the topological model and represents the aggregated physical behaviour. It is automatically obtained by the physical analysis of the interactions derived from the exchange of matter and energy defined by the topological model connection structure together with the modular local behaviour represented at the coupled submodels. This feature has been defined as dynamic modularity, since the physical behaviour represented by the reused model classes is dynamically formulated according to their reusing physical context. The derived phenomenological structure is represented by the PML *functional model* and it is mathematically formulated at the next step (non causal specification level). This formulation is made according to the adequacy level demanded by the user. This feature has been named as behaviour formulation *dynamic binding*. The dynamic modularity and dynamic binding capabilities are important contributions of PML since it is the mechanism that supports the separation between the system behaviour representation (defined by the phenomenon and law classes) and its mathematical formulation, i.e., the system dynamics representation (obtained from the law class attributes).

- An important contribution are the PML features supporting the definition of different experimental frameworks and adequacy levels during this translation. The very important consequence is that the same topological model can be reused for different experimentation purposes. Hence, reusability is extended to the model usage in addition to the model construction purpose. The model user can select the phenomena of interest for his experimental framework before generating the functional model. Then, he may easily select the proper laws applicable to the selected phenomena in order to set the desired adequacy level.

- The model reusability feature is extended also to the other PML modeling classes. This is a consequence of the modular characteristic of every PML modeling class. Strict modularity requirements have been defined in this thesis in order to permit the definition of the modeling classes without considering their reusing context. Consequently, the PML modeling libraries may grow with minimum side-effects over the already defined modeling classes.

- An important consequence of the stated modularity rules is that the experimental framework and adequacy level can be extended with minimum, even null, side effects on the already defined modeling classes. Model usage capabilities are therefore extended.

- With respect to the PML language pragmatic, the semantics is very closed to the user's physical understanding of the system, which leads to the improvement of the assistance capabilities provided to both the model developer and the model user. The satisfactory transparency of all details of the process model as well as of the modeling process is an essential requirement for model reuse. The PML language set of linguistic symbols allows an unambiguous identification of the physical concepts involved in the modeling process.

6.1.1. PML-PMT state of the art

The language and the modeling tool current version supports the modeling features reported through the thesis (with the exception of the phenomenon inheritance discussed at Section 4.6). Different academic examples have been developed, in addition to the ones formulated in this work, in order to test the modeling environment correct operation. PML-PMT has been presented in different conference contributions, such as (Ramos and Piera 1999, Ramos *et al.* 2001, Ramos *et al.* 2003*b*). An application example can be found in (Ramos *et al.* 2003*a*). It should be noticed that the current software implementation main objective has been the validation of the modeling approach proposed in the thesis. Wide use of the PML-PMT modeling environment for realistic problems will require of certain modifications in order to contemplate modeling features whose implementation was not relevant for this thesis purposes. Here follows a list of the foreseen main modifications:

- Implementation of semantic rules S27 and S28 in order to support phenomenon inheritance to extent model reusability by means of parameterization.
- The PML language syntax is being redefined in order to make more readable the modeling code.
- New attributes should be added to the entity modeling class in order to characterize the properties by means of physical units. The inclusion of equations to describe the change of units is being considered.
- New programming structures should be included in order to allow the definition of user functions. New data structures such as n-dimension vectors will be also included.

6.2. Future Research

The physical knowledge represented with the PML language define a very promising research field which should be explored. Also, certain modeling questions have not been dealt with depth because they were out of the thesis scope. Feasible directions for future research include the following:

- Generation of efficient models. The PML model classes representing the system at the topological level can be manipulated to obtain, in an easy way, different simulation models which are adequate for the simulation purpose, i.e., they are accurate and efficient enough for the model application. Despite of this manipulation process is automated by the PMT tool, the decision about how the model should be simplified is taken by the user. A major issue for future research will be the automation of the model adaptation procedure in order to avoid the user intervention in the generation of efficient simulation models. Future work will be developed to identify which mechanisms and manipulation rules can be designed to use the physical knowledge represented in the PML modeling classes in order to generate efficient simulation models according to the model application demands. According to the PML modeling environment architecture, this automated model manipulation can be implemented by specific software tools designed to interact with PMT.

- Modeling of hybrid systems. Except for the possibility to represent state event detection in the behaviour formulation at law classes and model equation section, PML does not currently contemplate the modeling of hybrid systems in a feasible manner. Future language specification should contemplate how the definition of models for hybrid systems with event propagation among model components may affect to the physical analysis procedure.

- Modeling library development. Every PML example developed in the thesis fits within the chemical process domain. Other application domains should be explored in the future.

References

Acebes, L.F. (1996). SIMPD: Sistema Inteligente de Modelado de Procesos Dinmicos. Ph.d-thesis. Dpt. de Ingeniera de Sistemas y Automtica. Universidad de Valladolid. Spain.

Andersson, M. (1990). Omola—An Object-Oriented Language for Model Representation. Lic. technical thesis. Department of Automatic Control. Lund Institute of Technology.. Lund, Sweden.

Andersson, M. (1994). Object-Oriented Modeling and Simulation of Hybrid Systems. Ph.d-thesis. Department of Automatic Control. Lund Institute of Technology.. Lund, Sweden.

Åström, K.J., H. Elmqvist and S.E. Mattsson (1998). Evolution of continous-time modeling and simulation. In: *The 12th European Simulation Multioconference, ESM98*. Manchester, UK.

Bogusch, R., B. Lohmann and W. Marquardt (2001). Computer-aided process modeling with MODKID. *Computers Chemical Engineering* **25**, 963–995.

Brenan, K.E., S.L. Campbell and L.R. Petzold (1989). *Numerical Solution of Initial-Value Problems in Differential-Algebraic Equations*. North-Holland. New York.

Broenink, J. F. (1990). Computer – Aided Physical System Modeling and Simulation: A Bond-Graph Approach. Ph.d-thesis. University of Twente. Enschede, The Netherlands.

Cellier, F. and H. Elmqvist (1993). Automated Formula Manipulation Supports Object-Oriented Continuous-System Modeling. *IEEE Control Systems*.

Cellier, F.E. (1991). *Continuous System Modeling*. Springer-Verlag. New York.

Cobas, P. and Al. (1999). *EcosimPro Simulation Language*. Empresarios Agrupados. http://www.ecosimpro.com.

Elmquist, H. and Al. (1999). *Modelica–A Unified Object-Oriented Language for Physical System Modeling. Tutorial and Rationale*. Modelica Design Group.

Elmqvist et Al., H. (2000). Modelica TM – A Unified Object-Oriented Language for Physical System Modelling. In: *http://www.modelica.org* (Modelica Association, Ed.).

Elmqvist, H., K.J. Åström, T. Schönthal and B. Wittenmark (1990). *Simnon User's Guide*. SSPA. Göteborg, Sweden.

Glaser, J.S., F.E. Cellier and A.F. Witulski (1995). Object-Oriented power system modeling using the Dymola modeling language. In: *Object Oriented Simulation Conference*. OOS'95. Las Vegas, Nevada. pp. 141–146.

Goldberg, A. (1983). *Smalltalk-80: The Interactive Programming Language*. Addison-Wesley.

Gustafsson, K. (1992). Control of error and Convergence in ODE solvers. Ph.d-thesis. Department of Automatic Control. Lund Institute of Technology.. Lund, Sweden.

Hahn, M. (1995). Object-oriented modelling of mechatronic systems. *Mathematical Modeling of Systems* **1**(4), 286–303.

Hopcroft, J.E. and J.D. Ullman (1979). *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley.

Jensen, K. (1997). *Coloured Petri Nets*. Vol. 1. Springer-Verlag.

Johansson, R. (1993). *System Modeling and Identification*. Prentice Hall.

Kailath, T. (1980). *Linear Systems*. Prentice-Hall.

Karnopp, D. and R.C. Rosenberg (1968). *Analysis and Simulation of Multiport Systems. The Bond Graph approach to Physical System Dynamics*. The M.I.T. Press.

Kheir, N.A. (1986). *Systems Modeling and Computer Simulation*. Marcel Dekker Inc.

Klir, G.J. (1969). *An approach to General System Theory*. Van Nostrand Reinhold. New York.

Klir, G.J. (1985). *Architecture of System Problem Solving*. Plenum Press. New York.

Kreutzer, W. (1986). *System Simulation - Programming Styles and languages*. Addison-Wesley.

Kuipers, B. (1986). Qualitative simulation. *Aritficial Intelligence* **29**, 289–338.

Kuipers, B. (1989). Qualitative reasoning: Modeling and simulation with incomplete knowledge. *Automatica* **25**(4), 571–585.

Lien, K. and T. Perris (1996). Future directions for CAPE research. perceptions of industrial needs and opportunities. *Computers Chemical Engineering* **20S**, S1551–S1557.

Lohmann, B. and W. Marquardt (1996). On the systematization of the process of model development. *Computers Chemical Engineering* **20S**, S213–S218.

Marquardt, W. (1991). Dynamic process simulation – recent progress and future challenges. In: *Chemical Process Control CPC-IV* (Y. Arkun and W.H. Ray, Eds.). pp. 131–180. CACHE, Austin. AICHE. New York.

Marquardt, W. (1996). Trends in Computer-Aided Process Modeling. *Computers Chemical Engineering* **20**(6/7), 591–609.

Mathworks (2000*a*). *Matlab Reference Manual*. The Mathworks.

Mathworks (2000*b*). *Simulink Reference Manual*. The Mathworks.

Matko, D., R. Karba and B. Zupancic (1992). *Simulation and Modelling of Continuous Systems*. Prentice Hall.

Mattsson, S.E. (1992). Modelling of power systems in omola for transient stability studies. In: *IEEE Symposium on Computer-Aided Control System Design,CACSD'92*. Napa, California.

Mattsson, S.E. (1993). Towards a new standard for modelling and simulation tools. In: *the 35th SIMS Simulation Conference - Applied Simulation in Industry*.

Mattsson, S.E. (1997). On modeling of heat exchangers in modelica. In: *9th European Simulation Symposium*. Passau, Germany.

Mattsson, S.E. and H. Elmqvist (1998). An overview of the modeling language modelica.. In: *Eurosim'98*. Helsinki, Finland.

Meyer, B. (1997). *Object-Oriented Software Construction*. Prentice Hall.

Minsky, M. (1965). Matter, mind and models. In: *Information Processing 1965* (W.A. Kalenich, Ed.). Vol. I of *Proceedings of IFIP Congress*. Spartan Books. Washington.

Mitchel and Gauthier Associates (1986). *Advanced Continuous Simulation Language (ACSL), Reference Manual*. Concord. Massachusetts.

Modelica-Association (2000). *Modelica–A Unified Object-Oriented Language for Physical System Modeling. Tutorial*. Modelica Design Group.

Nayak, P.Pandurang (1995). Automated Modeling of Physical Systems. Ph.d-thesis. NASA Ames Research Center. Moffet Field, CA (USA).

Nilsson, B. (1993). Object-Oriented Modeling of Chemical Processes. Ph.d-thesis. Department of Automatic Control. Lund Institute of Technology.. Lund, Sweden.

Pantelides, C.C. (1988*a*). The consistent initialization of differential-algebraic systems. *SIAM Journal of Scientific and Statistical Computing* **9**, 213–231.

Pantelides, C.C. (1988*b*). SpeedUp – recent advances in process simulation. *Computers Chemical Engineering* **12**, 745–755.

Pantelides, C.C. and P.I. Barton (1993). Equation-Oriented dynamic simulation, current status and future perspectives. *Computers Chemical Engineering* **17S**, S263–S285.

Pérez, R., P. Cobas and A. García (1999). Ecosim: State-of-the-art continuous simulation tool for RCS and ECLSS. In: *International Conference on Environmental Systems*. Denver, Colorado.

Piera, M.A. (1993). PMT: Un Entorno de Modelado en la Industria de Procesos. Ph.d-thesis. Unitat d'Enginyeria de Sistemes i d'Automàtica.. Universitat Autònoma de Barcelona, Bellaterra. España.

Piera, M.A., J.J. Ramos and D. Luque (1998). Object-oriented modelling of process industry systems: Constrains and expectations. In: *Advances in Systems, Signal, Control and Computers*. Vol. 1. Durban, South Africa.

Piera, M.A., J.J. Ramos, C. de Prada and I. Serra (1996). Several advantages when using PMT as a Dymola front-end.

Ramos, J. J., M. Gutierrez, R. Buil, M.A. Piera and M. Narciso (2003*a*). Generation of efficient dynamical models for simulation optimization. In: *XXV Jornadas de Automática*. León. Spain.

Ramos, J.J. (1994). Object-Oriented Modelling of Flows in Process Systems. Technical Report TFRT-7521. Department of Automatic Control. Lund Institute of Technology. Lund, Sweden.

Ramos, J.J. (1995). Una metodología de modelado orientado a objetos para la representación del conocimiento físico. Master's thesis. Unitat d'Enginyeria de Sistemes i d'Automàtica.. Universitat Autònoma de Barcelona, Bellaterra. España.

Ramos, J.J. and M.A. Piera (1999). Need of Oject-Oriented languages for Physics Knowledge representation in the Simulation field. In: *Technology of Object-Oriented Languages and Systems TOOLS 29* (R. Mitchell, A.C. Wills, J. Bosch and B. Meyer, Eds.). pp. 162–171. TOOLS Conferences, Nancy. IEEE Computer Society. Los Alamitos, California.

Ramos, J.J., M.A. Piera and D. Alsina (2001). Pml: an object-oriented modelling language for physics knowledge representation. In: *Eurosim'01*. Delft, The Netherlands.

Ramos, J.J., M.A. Piera and I. Serra (1995). A modelling tool to guide computational causality assignment through physical causality analysis. In: *Eurosim'95*. Vienna.

Ramos, J.J., M.A. Piera and I. Serra (1998*a*). The use of physical knowledge to guide formula manipulation in system modelling. *Simulation Practice and Theory* **6**(3), 243–254.

Ramos, J.J., Piera M.A and D. Alsina (2003*b*). Ecosimpro code generation from the physical analysis of object-oriented models. In: *II Jornadas de usuarios de EcosimPro*. UNED. Madrid, Spain.

Ramos, J.J., Piera M.A and I. Serra (1998*b*). A free context physics knowledge representation suitable for object-oriented modeling tools. In: *Eurosim'98*. Helsinki, Finland.

Rimvall, M. and F.E. Cellier (1986). Evolution and Perspectives of Simulation Languages following the CSSL Standard. In: *Modelling, Identification and Control*. Vol. 6. pp. 181–199.

Rumbaugh, J., M. Blaha, W. Premernali, F. Eddy and W. Lorensen (1991). *Object Oriented Modeling and Design*. Prentice Hall.

Sargent, R.W.H. (1983). Advances in modelling and analysis of chemical process systems. *Computers Chemical Engineering* **7**(4), 219–237.

Stephanopoulos, G. (1984). *Chemical Process Control. An Introduction to Theory and Practice*. Prentice Hall.

Stephanopoulos, G., G. Henning and H. Leone (1990*a*). Model.la. a modeling language for process engineering—I. The formal framework.. *Computers Chemical Engineering* **14**(8), 813–846.

Stephanopoulos, G., G. Henning and H. Leone (1990*b*). Model.la. a modeling language for process engineering—II. multifaceted modeling of processing systems.. *Computers Chemical Engineering* **14**(8), 847–869.

Strauss, J.C. (1967). The SCi Continuous System Simulation language (CSSL). *Simulation* **9**, 281–303.

Stroustrup, B. (1987). *The C++ Programming Language*. Addison-Wesley.

Thoma, J.U. (1990). *Simulation by bondgraphs: introduction to a graphical method*. Springer.

Urquía, A. (2000). Modelado Orientado a Objetos y Simulación de Sistemas Híbridos en el ámbito del Control de Procesos Químicos. Ph.d-thesis. Facultad de Ciencias. UNED. Madrid. Spain.

Westerberg, A.W. and D.R. Benjamin (1983). Thoughts on a future equation-oriented flowsheeting system. *Computers Chemical Engineering* **9**(5), 517–526.

Willems, J.C. (1991). Paradigms and puzzles in the theory of dynamical systems. *IEEE Transactions on Automatic Control* **36**(3), 259–294.

Willems, J.C. (1997). On Interconnections, Control and Feedback. *IEEE Transactions on Automatic Control* **42**(3), 326–339.

Woods, E.A. (1993). The Hybrid Phenomena Theory. PhD thesis. Department of Engeneering Cybernetics. Norwegian Institute of Technology, NTH. Trondheim, Norway.

Zeigler, B.P. (1984). *Multifacetted modelling and discrete event simulation*. Academic Press.

Zeigler, B.P. (1990). *Object-Oriented Simulation with Hierarchical, Modular Models: Intelligent Agents and Endomorphic Systems*. Academic Press.

Zeigler, B.P., H. Praehofer and T.G. Kim (2000). *Theory of Modeling and Simulation*. Academic Press.

A

PML Grammar

A.1. Lexical conventions

Used metasymbols (extended BNF):

[ ] optional item

{ } repeat zero or more times

PML defines the following lexical units:

DIGIT = **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9**

LETTERS = letters "**a**" to "**z**" | letters "**A**" to "**Z**"

A-CHAR = "_" | LETTERS

UNSIGNED_INTEGER = DIGIT { DIGIT }

UNSIGNED_NUMBER = UNSIGNED_INTEGER ["**.**" [UNSIGNED_INTEGER]]

[("**e**" | "**E**") ["+" | "-"] UNSIGNED_INTEGER]

STRING = """ { any printable character } """

CONSTANT = (["+" | "-"] UNSIGNED_NUMBER | STRING)

CONSTANT_ID = LETTERS { A-CHAR }

The PML language is case sensitive. PML keywords and built-in operators are bold-faced and they can not be used as a constant identifier. PML uses the same comment syntax as C++.

A.2. Grammar

A.2.1. Class definition

```
entity_definition:
   "(" ( entity | class_identifier )
      "[" class_identifier "]" "[" entity_body "]" ");"
entity_body:
   [ properties "(" constantId_list ")" ]
   [ components "(" classId_list ")" ]
pehomenon_definition:
   "(" phenomenon "[" class_identifier "(" classId_list ")" "]"
      "[" phenomenon_body "]" ");"
pehomenon_body:
   ( class_reference "->" property_reference |
      "{" class_reference "->" property_reference
       { "," class_reference "->" property_reference } "}" )
law_definition:
   "(" law "[" class_reference "]"
      "[" law_formulation "]" ");"
law_formulation:
   sentence |
   "{" sentence "," phenomenon"(" classId_list ")}"
port_definition:
   "(" ( port | class_identifier )
      "[" class_reference"]" "[" [ constantId_list ] "]" ");"
model_definition:
   "(" (model | class_identifier )
      "[" class_identifier "]"
```

```
      "[" [entity_clause] [phenomenon_clause] [port_clause]
          [submodel_clause] [connect_clause] [equation_clause]
      "]"
  ");"

entity_clause:
  entities "(" entity_clause_argument |
      "{" entity_clause_argument { "," entity_clause_argument } "}"
  ")"

entity_clause_argument :
  classId_list |
  class_identifier "=" classId_list |
  class_identifier "=" "0"

phenomenon_clause:
  phenomena "(" class_reference |
      "{" class_reference { "," class_reference } "}"
  ")"

port_clause:
  ports "(" class_reference |
      "{" class_reference { "," class_reference } "}"
  ")"

submodel_clause:
  submodels "(" submodel_clause_argument |
      "{" submodel_clause_argument { "," submodel_clause_argument } "}"
  ")"

submodel_clause_argument :
  class_identifier "(" constantId_list ")"|
  class_identifier "(" submodel_paramerization ")"
```

```
submodel_parameterization :
   CONSTANT_ID "(" class_identifier "=" classId_list ")" |
   CONSTANT_ID "(" "{" class_identifier "=" classId_list
                  { "," class_identifier "=" classId_list } "}" ")"

equation_clause:
   equations "(" equation ")"

property_reference:
   CONSTANT_ID "(" class_identifier ")"

class_reference:
   class_identifier "(" class_identifier ")"

constantId_list:
   ( CONSTANT_ID | "{" CONSTANT_ID { "," CONSTANT_ID } "}")

classId_list:
   ( class_identifier | "{" class_identifier { "," class_identifier } "}")

class_identifier:
   CONSTANT_ID [ "::" CONSTANT_ID ]
```

A.2.2. Equations

```
equation:
   (sentence |
    expand "({" CONSTANT_ID "," sentence "})" |
    "{" equation {"," equation } "}")

sentence:
   (data_sentence | equality_sentence | if_sentence | zone_sentence |
    procedural_sentence)
```

```
data_sentence:
  DATA "(" (def_data | "{" def_data { "," def_data } "}" ) ")"
def_data:
  CONSTANT_ID ["=" CONSTANT]
equality_sentence:
  simple_expression "=" term
if_sentence:
  IF "(" expression ")" THEN "(" sentence ")"
  [ELSE "(" sentence ")"]
zone_sentence:
  (CONSTANT_ID | property_reference) "="
  ZONE "(" expression ")" math_operation
  { ZONE "(" expression ")" math_operation }
  OTHERS "(" expression ")" math_operation
procedural_sentence:
  procedural "(" equation ")"
```

A.2.3. Expressions

```
expression:
  (NOT "(" expression ")" |
  LOGIC_OPERATOR "(""{" term "," term "}"")")
term:
  (CONSTANT | simple_expression)
simple_expression:
  ["+"|"-"] ( property_reference |
  property_in_port | port_instances | math_operation)
property_in_port:
  CONSTANT_ID "(" CONSTANT_ID "(" class_identifier ")" ")"
```

```
port_instances:
   portInstances "(" expand_port ")"
math_operation:
   MATH_OPERATOR "(" ( (["+"|"-"]UNSIGNED_NUMBER | simple_expression ) |
   "{" (["+"|"-"]UNSIGNED_NUMBER | simple_expression )
    { "," (["+"|"-"]UNSIGNED_NUMBER | simple_expression ) } "}") ")"
expand_port:
   ( property_reference | "{" CONSTANT_ID "," port_expression "}" )
port_expression :
   (property_reference |
   (sum | prod) "({" property_reference { "," property_reference }"})")
```

LOGIC_OPERATOR = **LT** | **LE** | **EQ** | **GE** | **GT**

MATH_OPERATOR = **sin**|**cos**|**tan**|**asin**| **asin**|**acos**|**atan**|**abs**| **log**|**ln**|**exp**|**sqrt**| **instances**|
div|**pot**|**sum**|**prod**|**max**|**min**|**intgr**|**der**

B

PML Semantic Rules

B.1. Entity class

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. ❑

Semantic rule S2 – *Definition of entity properties.*

The properties of an entity can only be defined as an attribute of the entity class. ❑

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. ❑

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.

❑

B.2. Phenomenon class

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. ❑

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. ❑

B.3. Law class

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 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. ❑

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). ❑

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. ❑

B.4. Port class

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. ❑

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. ❑

B.5. Model class

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.

❑

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. ❑

Semantic rule S14 – *Atomic model class*

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

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). ❑

B.6. Object Oriented features

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. ❑

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. ❑

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. ❑

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. ❑

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. ❑

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.

❑

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.

❑

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.

❑

B.7. Advanced modeling concepts

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.
❑

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 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.

❑

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. ❑

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. ❑

C

EcosimPro examples

This appendix includes the EcosimPro models of some of the examples developed in the text. The code has been generated by PMT.

C.1. Example 4.1

This section shows the mathematical model generated by PMT of the process unit shown at the Figure 4.5 (see section 4.3). The representation of the port exchanging matter is:

```
PORT C4MatterPort
  SUM REAL C4Matter_massFlow
  EQUAL REAL C4Matter_pressure
END PORT COMPONENT
```

The mathematical model of the tank is:

```
C4Tank --IS_A C4::tank
  PORTS
     IN C4MatterPort P1 -- On Entity: C4::matter
     IN C4MatterPort P2 -- On Entity: C4::matter
     IN C4MatterPort P3 -- On Entity: C4::matter
  DATA
  -- Data declared by local equations
     REAL section=1.0
     REAL pTop=1.0
     REAL g=9.8
  DECLS
  -- Variables declared by local equations
     REAL level
  -- Entity properties referenced by local equations
     REAL C4Matter_density
```

```
    REAL C4Matter_mass
    REAL C4Matter_volume
  CONTINUOUS
  -- PHENOMENON: C4::store(matter)
  --        LAW: C4::massBalance(matter)
    C4Matter_mass'=P1.C4Matter_massFlow+P2.C4Matter_massFlow+P3.C4Matter_massFlow
  -- Local model equations
    P1.C4Matter_pressure=pTop
    P2.C4Matter_pressure=pTop
    P3.C4Matter_pressure=pTop+g*level*C4Matter_density
    C4Matter_mass=C4Matter_volume*C4Matter_density
    C4Matter_volume=section*level
END COMPONENT
```

The mathematical model of the duct is:

```
COMPONENT   C4Duct --IS_A C4::duct
  PORTS
    IN  C4MatterPort P1 -- On Entity: C4::matter
    IN C4MatterPort P2 -- On Entity: C4::matter

  DATA
  -- Data declared by laws
    REAL Rfluid=1.0
  -- Data declared by local equations
    REAL ductVolume=5.0
  DECLS
  -- Entity properties referenced by laws
    REAL C4Matter_massFlow
  -- Entity properties referenced by local equations
    REAL C4Matter_pressureDrop
    REAL C4Matter_volume
  CONTINUOUS
  -- PHENOMENON: C4::transport(matter)
  --        LAW: C4::matterTransport(matter)
    C4Matter_massFlow=Rfluid*ssqrt(C4Matter_pressureDrop)
    C4Matter_massFlow=P1.C4Matter_massFlow
    P1.C4Matter_massFlow+P2.C4Matter_massFlow=0.0
  -- Local model equations
    C4Matter_pressureDrop = P1.C4Matter_pressure-P2.C4Matter_pressure
    C4Matter_volume=ductVolume
END COMPONENT
```

Finally, the mathematical model of the process is:

```
COMPONENT   C4process --IS_A C4::model
  TOPOLOGY
    C4Tank mTA
    C4Duct dE,dW,dV
```

```
    CONNECT dE.P2 TO mTA.P1
    CONNECT dW.P2 TO mTA.P2
    CONNECT dV.P1 TO mTA.P3
END COMPONENT
```

C.2. Example 4.5

This section shows the mathematical model generated by PMT from the `twoTanks` PML topological model developed at Example 4.5 (see section 4.6) when it is parameterized with the `water` entity. The representation of the port exchanging matter is:

```
PORT C4MatterPort
  SUM REAL C4Water_massFlow
  EQUAL REAL C4Water_pressure
END PORT
```

The tank mathematical model is given by:

```
COMPONENT   C4Tank --IS_A C4::tank
   PORTS
     IN C4MatterPort P1 -- On Entity: C4::water
     IN C4MatterPort P2 -- On Entity: C4::water
     IN C4MatterPort P3 -- On Entity: C4::water

  DATA
  -- Data declared by local equations
     REAL section=1.0
     REAL pTop=1.0
     REAL g=9.8
  DECLS
  -- Variables declared by local equations
     REAL level
  -- Entity properties referenced by local equations
     REAL C4Water_density
     REAL C4Water_mass
     REAL C4Water_volume
  CONTINUOUS
  -- PHENOMENON: C4::store(matter)
  --        LAW: C4::massBalance(matter)
     C4Water_mass'=P1.C4Water_massFlow+P2.C4Water_massFlow+P3.C4Water_massFlow
  -- Local model equations
     P1.C4Water_pressure=pTop
     P2.C4Water_pressure=pTop
     P3.C4Water_pressure=pTop+C4Water_density*g*level
     C4Water_mass=C4Water_volume*C4Water_density
     C4Water_volume=section*level
END COMPONENT
```

The valve mathematical code generated is:

```
COMPONENT   C4Valve --IS_A C4::valve
   PORTS
     IN C4MatterPort P1 -- On Entity: C4::water
     IN C4MatterPort P2 -- On Entity: C4::water

  DATA
  -- Data declared by local equations
     REAL Ap=1.0
     REAL Cv=1.0
     REAL ductVolume=5.0
  DECLS
  -- Variables declared by local equations
     REAL Rfluid
  -- Entity properties referenced by local equations
     REAL C4Water_massFlow
     REAL C4Water_pressureDrop
     REAL C4Water_volume
  CONTINUOUS
  -- PHENOMENON: C4::transport(matter)
  --        LAW: C4::matterTransport(matter)
     C4Water_massFlow=Rfluid*ssqrt(C4Water_pressureDrop)
     P1.C4Water_massFlow+P2.C4Water_massFlow=0.0
  -- Local model equations
     Rfluid=Ap*Cv
     C4Water_massFlow=P1.C4Water_massFlow
     C4Water_pressureDrop=P1.C4Water_pressure-P2.C4Water_pressure
     C4Water_volume=ductVolume
END COMPONENT
```

Finally, the aggregated system mathematical model is

```
COMPONENT   C4TwoTanks --IS_A C4::twoTanks
  TOPOLOGY
    C4Tank tk1,tk2
    C4Valve v

    CONNECT tk1.P3 TO v.P1
    CONNECT v.P2 TO tk2.P3
END COMPONENT
```

C.3. Example 5.1

This section shows the mathematical model generated by PMT from the heat exchanger topological model developed at Example 5.1 when it is parameterized with the `water` entity. The mathematical model of the `hexDuct` is:

```
COMPONENT   C5HexDuctWater --IS_A C5::hexDuct
   PORTS
     IN C5WaterPort P1 -- On Entity: C5::water
     IN C5WaterPort P2 -- On Entity: C5::water
     IN C5ThermalPort PE1 -- On Entity: C5::thermalEnergy
```

```
    IN C5ThermalPort PE2 -- On Entity: C5::thermalEnergy
    IN waterProperties PAP_innerProp -- added by physical analisys procedure
  DATA
  -- Data declared by laws
    REAL cp=4180
    REAL rho=1000
  -- Data declared by local equations
    REAL Rfluid=0.025
    REAL ductVolume=0.001
  DECLS
  -- Entity properties referenced by laws
    REAL C5Water_H
    REAL C5Water_T
    REAL C5ThermalEnergy_thE
  -- Entity properties referenced by local equations
    REAL C5Water_w
    REAL C5Water_cp
    REAL C5Water_dP
    REAL C5Water_h
    REAL C5Water_rho
    REAL C5Water_v
  CONTINUOUS
  -- PHENOMENON: C5::storeInternalEnergy({matter,thermalEnergy})
  --        LAW: C5::enthalpyBalance({matter,thermalEnergy})
    C5Water_rho=rho
    C5Water_cp=cp
    C5Water_H'=P1.C5Water_w*P1.C5Water_h+P2.C5Water_w*P2.C5Water_h+\
              PE1.C5ThermalEnergy_phi+PE2.C5ThermalEnergy_phi
    C5Water_H=C5Water_rho*C5Water_v*C5Water_h
    C5Water_h=C5Water_cp*C5Water_T
    C5ThermalEnergy_thE=C5Water_T
  -- PHENOMENON: C5::transport(matter)
  --        LAW: C5::matterTransport(matter)
    C5Water_w=Rfluid*C5Water_dP
    P1.C5Water_w+P2.C5Water_w=0.0
  -- Local model equations
    PE1.C5ThermalEnergy_thE=P1.C5Water_h/P1.C5Water_cp
    PE2.C5ThermalEnergy_thE=P2.C5Water_h/P2.C5Water_cp
    C5Water_dP=P1.C5Water_p-P2.C5Water_p
    C5Water_v=ductVolume
    C5Water_w=P1.C5Water_w
  -- added by physical analisys procedure
    PAP_innerProp.C5Water_h = C5Water_h
    PAP_innerProp.C5Water_cp = C5Water_cp
    PAP_innerProp.C5Water_rho = C5Water_rho

END COMPONENT
```

The wall mathematical model is:

```
COMPONENT   C5Wall --IS_A C5::wall
   PORTS
     IN C5ThermalPort Ph1 -- On Entity: C5::thermalEnergy
     IN C5ThermalPort Ph2 -- On Entity: C5::thermalEnergy
     IN C5ThermalPort Pc1 -- On Entity: C5::thermalEnergy
     IN C5ThermalPort Pc2 -- On Entity: C5::thermalEnergy
  DATA
  -- Data declared by local equations
     REAL hC=1e6
     REAL hH=1e6
     REAL pRf=0
     REAL pLambda=397.48
     REAL pd=0.1
     REAL pAw=0.025
     REAL pY=1
  DECLS
  -- Variables declared by local equations
     REAL deltaT1
     REAL deltaT2
  -- Entity properties referenced by local equations
     REAL C5Material_Aw
     REAL C5ThermalEnergy_phi
     REAL C5Material_R
     REAL C5Material_Rc
     REAL C5Material_Rh
     REAL C5Material_Rw
     REAL C5Material_Rf
     REAL C5Material_d
     REAL C5Material_lambda
     REAL C5Material_Y
     REAL C5ThermalEnergy_dTh
  CONTINUOUS
  -- PHENOMENON: C5::heatConduction({material,thermalEnergy})
  --        LAW: C5::heatFlow({material,thermalEnergy})
     Ph1.C5ThermalEnergy_phi+Ph2.C5ThermalEnergy_phi+\
     Pc1.C5ThermalEnergy_phi+Pc2.C5ThermalEnergy_phi=0.0
     C5ThermalEnergy_phi=C5ThermalEnergy_dTh/C5Material_R
  -- Local model equations
     C5Material_Aw=pAw
     Ph1.C5ThermalEnergy_phi=C5ThermalEnergy_phi/2
     Ph2.C5ThermalEnergy_phi=C5ThermalEnergy_phi/2
     C5Material_R=C5Material_Rc+C5Material_Rh+C5Material_Rw+C5Material_Rf
     C5Material_Rc=1/(hC*C5Material_Aw)
     C5Material_Rf=pRf
     C5Material_Rh=1/(hH*C5Material_Aw)
     C5Material_Rw=C5Material_d/(C5Material_lambda*C5Material_Y*C5Material_Aw)
     C5Material_Y=pY
     C5Material_d=pd
     C5ThermalEnergy_dTh=\
```

```
    ZONE(abs(deltaT1*deltaT2)<1e-4 TOL 1e-4)
        (deltaT1-deltaT2)/2
    ZONE(abs(deltaT1-deltaT2)>0.05*max(abs(deltaT1),abs(deltaT2)) TOL 1e-3)
        (deltaT1-deltaT2)/log(deltaT1/deltaT2)
    OTHERS
        0.5*(deltaT1+deltaT2)*(1-(deltaT1-deltaT2)**2/ \
        (deltaT1*deltaT2)*(1 - (deltaT1-deltaT2)**2/ \
        (deltaT1*deltaT2)/2)/12)
    deltaT1=Ph1.C5ThermalEnergy_thE-Pc1.C5ThermalEnergy_thE
    deltaT2=Ph2.C5ThermalEnergy_thE-Pc2.C5ThermalEnergy_thE
    C5Material_lambda=pLambda
    Ph1.C5ThermalEnergy_phi+Pc1.C5ThermalEnergy_phi=0
    Ph2.C5ThermalEnergy_phi+Pc2.C5ThermalEnergy_phi=0
END COMPONENT
```

The mathematical model of a heat exchanger section is:

```
COMPONENT   C5HexSection --IS_A C5::hexSection
   PORTS
     IN C5WaterPort hotIn -- On Entity: C5::water
     IN C5WaterPort hotOut -- On Entity: C5::water
     IN C5WaterPort coldIn -- On Entity: C5::water
     IN C5WaterPort coldOut -- On Entity: C5::water
     IN waterProperties PAP_hotCoilProp -- added by physical analisys procedure
     IN waterProperties PAP_coldCoilProp -- added by physical analisys procedure
   TOPOLOGY
     C5HexDuctWater hotCoil
     C5HexDuctWater coldCoil
     C5Wall W1
     CONNECT hotIn TO hotCoil.P1
     CONNECT hotOut TO hotCoil.P2
     CONNECT hotCoil.PAP_innerProp TO PAP_hotCoilProp
     CONNECT coldIn TO coldCoil.P1
     CONNECT coldOut TO coldCoil.P2
     CONNECT coldCoil.PAP_innerProp TO PAP_coldCoilProp
     CONNECT hotCoil.PE1 TO W1.Ph1
     CONNECT hotCoil.PE2 TO W1.Ph2
     CONNECT W1.Pc1 TO coldCoil.PE1
     CONNECT W1.Pc2 TO coldCoil.PE2
END COMPONENT
```

Finally, the mathematical model of a three section heat exchanger:

```
COMPONENT   C5TS30 --IS_A C5::heatExchanger
   PORTS
     IN C5WaterPort hotIn -- On Entity: C5::water
     IN C5WaterPort hotOut -- On Entity: C5::water
     IN C5WaterPort coldIn -- On Entity: C5::water
     IN C5WaterPort coldOut -- On Entity: C5::water
   TOPOLOGY
```

```
    C5HexSection S1
    C5HexSection S2
    C5HexSection S3
    waterFlowEdge hFei,hFe1,hFe2,hFeo
    waterFlowEdge cFei,cFe1,cFe2,cFeo
    -- water PATHS
    CONNECT hotIn TO hFei.PA
    CONNECT hFei.PB TO S1.hotIn
    CONNECT S1.hotOut TO hFe1.PA
    CONNECT hFe1.PB TO S2.hotIn
    CONNECT S2.hotOut TO hFe2.PA
    CONNECT hFe2.PB TO S3.hotIn
    CONNECT S3.hotOut TO hFeo.PA
    CONNECT hFeo.PB TO hotOut
    -- property flow propagation
    CONNECT S1.PAP_hotCoilProp TO hFei.PBprop
    CONNECT S1.PAP_hotCoilProp TO hFe1.PAprop
    CONNECT S2.PAP_hotCoilProp TO hFe1.PBprop
    CONNECT S2.PAP_hotCoilProp TO hFe2.PAprop
    CONNECT S3.PAP_hotCoilProp TO hFe2.PBprop
    CONNECT S3.PAP_hotCoilProp TO hFeo.PAprop
    -- Water PATHS
    CONNECT coldIn TO cFei.PA
    CONNECT cFei.PB TO S1.coldIn
    CONNECT S1.coldOut TO cFe1.PA
    CONNECT cFe1.PB TO S2.coldIn
    CONNECT S2.coldOut TO cFe2.PA
    CONNECT cFe2.PB TO S3.coldIn
    CONNECT S3.coldOut TO cFeo.PA
    CONNECT cFeo.PB TO coldOut
    -- property flow propagation
    CONNECT S1.PAP_coldCoilProp TO cFei.PBprop
    CONNECT S1.PAP_coldCoilProp TO cFe1.PAprop
    CONNECT S2.PAP_coldCoilProp TO cFe1.PBprop
    CONNECT S2.PAP_coldCoilProp TO cFe2.PAprop
    CONNECT S3.PAP_coldCoilProp TO cFe2.PBprop
    CONNECT S3.PAP_coldCoilProp TO cFeo.PAprop
END COMPONENT
```

The representation of the mathematical interfaces is:

```
PORT C5ThermalPort
   EQUAL REAL C5ThermalEnergy_thE
   SUM   REAL C5ThermalEnergy_phi
END PORT
PORT C5WaterPort
   EQUAL REAL C5Water_p
   SUM   REAL C5Water_w
   EQUAL REAL C5Water_cp
   EQUAL REAL C5Water_rho
```

```
   EQUAL REAL C5Water_h
END PORT
PORT waterProperties
  EQUAL REAL C5Water_h -- flowing Water enthalpy auxiliar
  EQUAL REAL C5Water_cp -- flowing Water enthalpy auxiliar
  EQUAL REAL C5Water_rho -- flowing Water density
END PORT
COMPONENT waterFlowEdge
  PORTS
      IN C5WaterPort PA
      IN C5WaterPort PB
      IN waterProperties PAprop,PBprop
  DECLS
      REAL C5Water_h
      REAL C5Water_cp
      REAL C5Water_rho
      REAL alpha
  CONTINUOUS
      alpha = ZONE (PA.C5Water_w>=0.0 TOL 1e-4) 1
              OTHERS 0
      C5Water_h = alpha*PAprop.C5Water_h+(1-alpha)*PBprop.C5Water_h
      C5Water_cp = alpha*PAprop.C5Water_cp+(1-alpha)*PBprop.C5Water_cp
      C5Water_rho = alpha*PAprop.C5Water_rho+(1-alpha)*PBprop.C5Water_rho

      PA.C5Water_h = C5Water_h
      PA.C5Water_cp = C5Water_cp
      PA.C5Water_rho = C5Water_rho

      PB.C5Water_h = C5Water_h
      PB.C5Water_cp = C5Water_cp
      PB.C5Water_rho = C5Water_rho

      PA.C5Water_p = PB.C5Water_p
      PA.C5Water_w+PB.C5Water_w=0
END COMPONENT
```

Figures C.1 and C.2 show some simulation results of the heat exchanger model once the causal explanation is generated by EcosimPro. They are included to illustrate the simulation model operation, although no validation against real experimental data has been made.

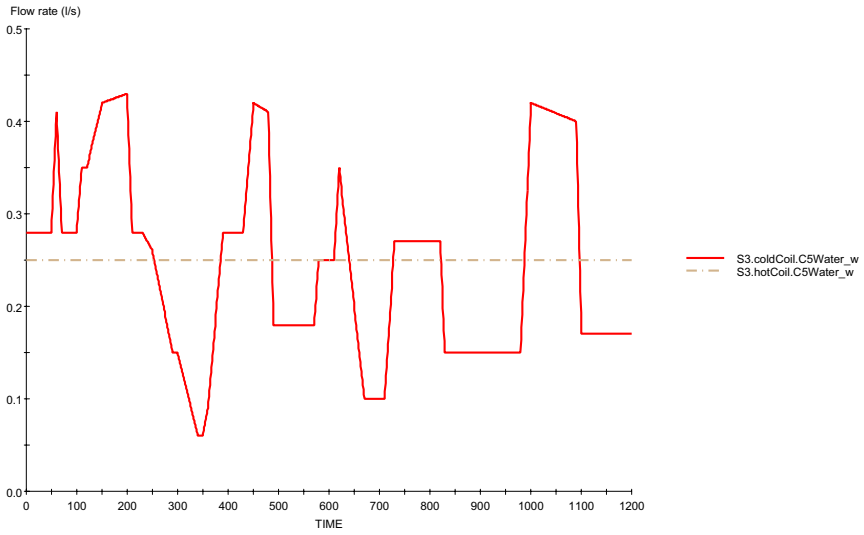

Figure C.1. Water mass flow rates through the hot and cold ducts of the heat exchanger.

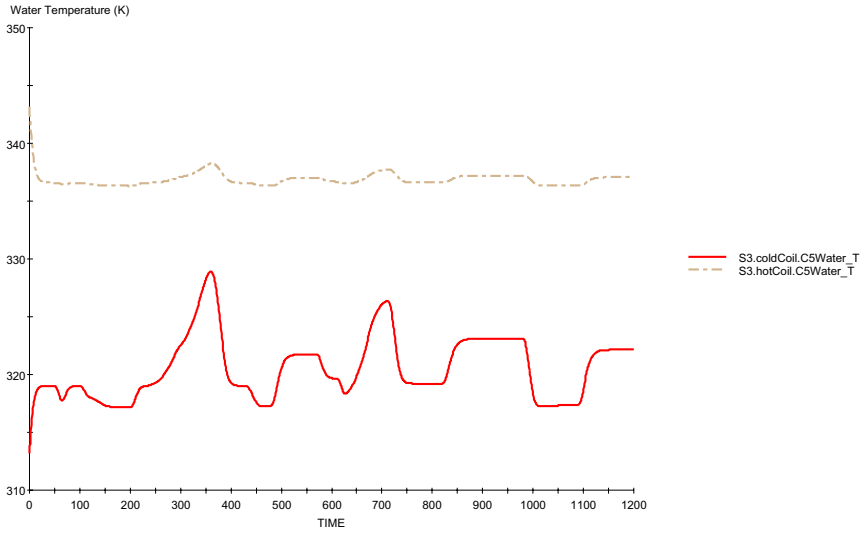

Figure C.2. Temperatures of the water at the hot and cold ducts of the heat exchanger.