Chapter 7: **Tank System**

This example illustrates how you can build a hierarchical model using *MathModelica System Designer* as well as make new libraries. A flat-tank model is first developed, followed by a similar component-based tank model. We then see the flexibility that this gives us to test new scenarios.

7.1 Flat tank

The system we will begin with is a one-tank system with a controller, as illustrated in the picture below.

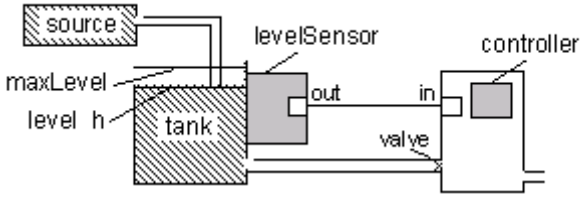

Figure 7-1: A graphical representation of a tank system.

To implement the model we need to set up the system equations. The water level, h, in the tank is a function of the flow in and out of the tank and the tank area:

$$\dot{h} = \frac{q_{in} - q_{out}}{A}$$

In this example we choose an input flow that is constant the first 150 seconds after which it triples:

$$q_{in} = \begin{cases} flowLevel, & t < 150\text{s} \\ 3(flowLevel), & t \geq 150\text{s} \end{cases}$$

where flowLevel is a parameter. By controlling the output flow we will try to keep the tank level at a desired reference value, ref. In order to do this we implement a PI controller:

$$q_{out} = K\left(error(t) + \frac{1}{T}\int_0^t error(s)ds \right)$$

where K is the controller gain and T is the time-constant of the controller. Finally, we limit the output flow to a minimum value, minV, and a maximum value, maxV. With this information we can implement the flat Modelica code.

```
model FlatTank
  parameter Real flowLevel(unit="m3/s")=0.02;
  parameter Real area(unit="m2")=1;
  parameter Real flowGain(unit="m2/s")=0.05;
  parameter Real K=2 "Gain";
  parameter Real T(unit="s")=10 "Time constant";
  parameter Real minV=0,maxV=10;
  parameter Real ref=0.25 "Reference level for control";
  Real h(start=0,unit="m") "Tank level";
  Real qInflow(unit="m3/s") "Flow through input valve";
  Real qOutflow(unit="m3/s") "Flow through output valve";
  Real error "Deviation from reference level";
  Real outCtr "Control signal without limiter";
  Real x "State variable for controller";
equation
  assert(minV >= 0, "minV must be greater or equal to zero");
  der(h)=(qInflow - qOutflow)/area;
  qInflow=if time > 150 then 3*flowLevel else flowLevel;
  qOutflow=Functions.LimitValue(minV, maxV, -flowGain*outCtr);
  error=ref - h;
  der(x)=error/T;
  outCtr=K*(error + x);
end FlatTank;

function LimitValue
  input Real pMin;
  input Real pMax;
```

```
  input Real p;
  output Real pLim;
algorithm
  pLim:=if p > pMax then pMax else if p < pMin then pMin else p;
end LimitValue;
```

By simulating the model for 250 seconds we can see that the tank level starts to increase, reaching and then surpassing the desired reference level. Once the desired level is surpassed, the outflow is opened and after 150 seconds the level is stabilized. However, at this moment the input flow is suddenly increased, thus increasing the water level before the controller manages to stabilize it again. This is illustrated in the figure below.

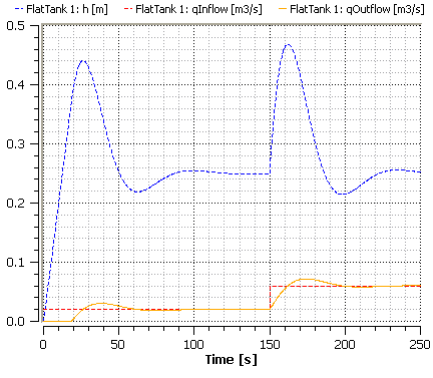

Figure 7-2: Plotting the tank level and the flows through in and out the flat tank with default parameters values.

7.2 Component-based tank

Implementing a component-based tank will require a bit more work to begin with, but as soon as we start experimenting with the tank and testing different scenarios we will regain the invested time.

When using the object-oriented component-based approach to modeling, we first try to understand the system structure and decomposition in a hierarchical top-down manner. Once the system components and interactions between these components have been roughly identified, we can apply the first traditional modeling phases of identifying variables and equations to each of these model components.

By studying Figure 7-1 we see that the tank system has a natural component structure.

We can identify five components in the figure: the tank itself, the liquid source, the level sensor, the valve, and the controller. However, since we will choose very simple representations of the level sensor and the valve, i.e. just a simple scalar variable for each, we let these variables be simple Real variables in the tank model instead of creating two new classes, each containing a single variable.

The next step is to determine the interactions and communication paths between the components. It is fairly obvious that fluid flows from the source tank via a pipe. Fluid also leaves the tank via an outlet controlled by the valve. The controller needs measurements of the fluid level from the sensor. Thus, a communication path from the sensor of the tank and the controller needs to be established.

In order to connect communication paths, connector instances must be created for those components which are connected, and connector classes must be declared when needed. In fact, the system model should be designed such that the only communication between a component and the rest of the system is via connectors.

Finally, we should think about reuse and generalizations of certain components. For example, do we expect that several variants of a component will be needed? In the case of the tank system we expect to plug in several variants of the controller, starting with a PI controller. Thus, it is useful for us to create a base class for tank system controllers.

The structure of the tank system model developed using the object-oriented component-based approach is clearly visible in Figure 7-3 below.

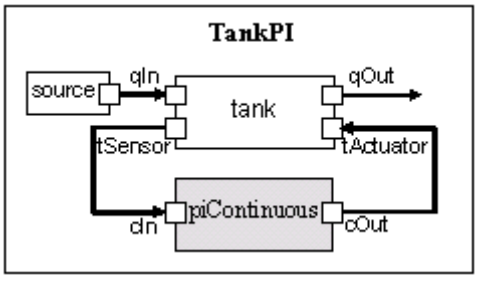

Figure 7-3: A graphical representation of an object-oriented component-based tank system.

We can identify three different types of classes that will be used in the model: interfaces, functions and components. Therefore, we develop a package containing three sub packages. To create a package right-click on the tree root in the library browser and select New Class as shown in Figure 7-4. You can as well right-click on the package you want to add your package to and select New Class.

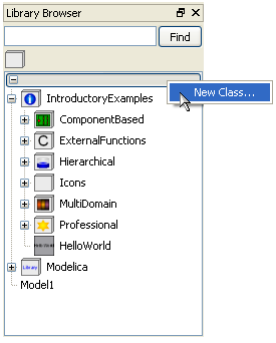

Figure 7-4: Menu to create a new class

In the dialog box that opens, set the class restriction to package and give the package the name "Hierarchical". Click the OK button to create the new package. The package will appear in the main tree of the library browser. By right clicking the name of the new package we can create and add models and packages to it.

7.2.1 Interfaces

We are now ready to create the interfaces, called connectors. Begin by creating a new package Interfaces within the Hierarchical package. Unless already expanded, expand the Hierarchical package in the library browser to view its contents. Create connector classes within the Interfaces package by specifying connector as the class restriction in the new class dialog box.

Create one connector class for reading the fluid level (see the Hello World example to see how to edit models textually and make model icons):

```
connector ReadSignal "Reading fluid level"
   Real val(unit="m");
end ReadSignal;
```

Create a second connector class for the signal to the actuator for setting valve position.

```
connector ActSignal "Signal to actuator for setting valve
position"
   Real act;
end ActSignal;
```

Finally create a connector class for the liquid flow at inlets and outlets:

```
connector LiquidFlow "Liquid flow at inlets or outlets"
   Real lflow(unit="m3/s");
end LiquidFlow;
```

7.2.2 Tank components

The next step is to create the three components of the system. Begin by creating a Components package within the Hierarchical package. In the Components package, create a tank model named Tank. The tank model has four interfaces (connectors in Modelica): qIn for input flow, qOut for output flow, tSensor for providing fluid level measurements, and tActuator for setting the position of the valve at the outlet of the tank. The central equation regulating the behavior of the tank is the mass balance equation, which in the current simple form assumes constant pressure. The output flows are related to the valve position through the flowGain parameter and the LimitValue function. This function guarantees that the flow does not exceed what corresponds to the open/closed positions of the valve:

```
model Tank;
   parameter Real area(unit="m2")=0.5;
   parameter Real flowGain(unit="m2/s")=0.05;
   parameter Real minV=0,maxV=10;
   Real h(start=0.0,unit="m") "Tank level";
   Hierarchical.Interfaces.ReadSignal tSensor "Connector, sensor
reading tank level (m)";
   Hierarchical.Interfaces.LiquidFlow qIn "Connector, flow (m3/s)
through input valve";
   Hierarchical.Interfaces.LiquidFlow qOut "Connector, flow (m3/s)
through output valve";
   Hierarchical.Interfaces.ActSignal tActuator "Connector,
actuator controlling input flow";

equation
assert(minV >= 0, "minV - minimum Valve level must be >= 0 ");
  der(h)=(qIn.lflow - qOut.lflow)/area;

equation
  qOut.lflow=Functions.LimitValue(minV, maxV, -
flowGain*tActuator.act);
  tSensor.val=h;
end Tank;
```

The model uses the already defined connector as well as the LimitFunction, which has not been defined yet. This is defined by creating the following function within a new package, named Functions. As it is a function we set its class restriction to Function when creating it.

```
function LimitValue;
  input Real pMin;
  input Real pMax;
  input Real p;
  output Real pLim;
algorithm
  pLim:=if p > pMax then pMax else if p < pMin then pMin else p;
end LimitValue;
```

The fluid entering the tank must originate somewhere. Therefore we have a liquid source component in the tank system Flow which increases sharply at t=150 to three times the previous flow level. This creates an interesting control problem that the tank controller must handle. The following model is created in the Components package:

```
model LiquidSource;
  parameter Real flowLevel=0.02;
  Hierarchical.Interfaces.LiquidFlow qOut;

equation
  qOut.lflow=if time > 150 then 3*flowLevel else flowLevel;
end LiquidSource;
```

7.2.3 Controllers

Finally the controllers need to be specified. We will initially choose a PI controller but later replace it with other kinds of controllers. The behavior of a PI (proportional and integrating) controller is primarily defined by the following two equations:

$$\frac{dx}{dt} = \frac{error}{T}$$

$$outCtr = K*(error + x)$$

Here x is the controller state variable, error is the difference between the reference level and the actual level of liquid obtained from the sensor, T is the time constant of the controller, outCtr is the control signal to the actuator for controlling the valve position, and K is the gain factor. These two equations are placed in the controller class PIcontinuousController, which extends the BaseController class defined later:

```
model PIcontinuousController
  extends BaseController(K=2,T=10);
  Real x "State variable of continuous PI controller";
```

```
equation
  der(x)=error/T;
  outCtr=K*(error + x);
end PIcontinuousController;
```

Both the PI and PID controllers to be defined later inherit the partial controller class Base-Controller, containing common parameters, state variables, and two connectors: one to read the sensor and one to control the valve actuator.

```
partial model BaseController;
  parameter Real Ts(unit="s")=0.1 "Time period between discrete
samples";
  parameter Real K=2 "Gain";
  parameter Real T(unit="s")=10 "Time constant";
  parameter Real ref "Reference level";
  Real error "Deviation from reference level";
  Real outCtr "Output control signal";
  IntroductoryExamples.Hierarchical.Interfaces.ReadSignal cIn
"Input sensor level,  connector";
IntroductoryExamples.Hierarchical.Interfaces.ActSignal cOut
"Control to actuator, connector";

equation
  error=ref - cIn.val;
  cOut.act=outCtr;
end BaseController;
```

7.2.4 Small tank system

When this is finished we can compose our tank system model with drag-and-drop.

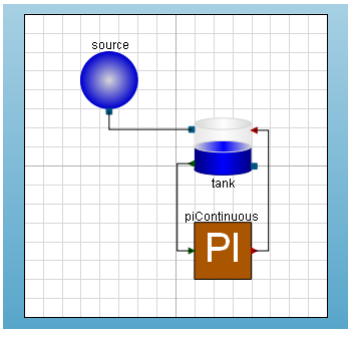

Figure 7-5: The diagram view of the IntroductoryExamples.Hierarachical.TankPI model.

Simulating for 250 seconds yields the same result as the flat-tank system.

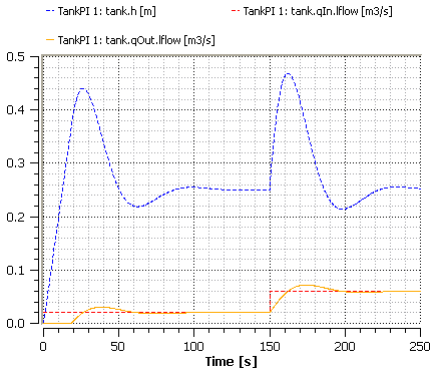

Figure 7-6: Plotting the tank level and the flows through in and out the PI tank with default parameters values.

7.3 Tank with continuous PID controller

We now define a TankPID system, which is the same as the TankPI system except that the PI controller has been replaced by a PID controller. Here we see a clear advantage of the object-oriented component-based approach over the traditional model-based approach, since system components can easily be replaced and changed in a plug-and-play manner.

A PID (proportional, integrating, derivative) controller model can be derived in a similar way as the PI controller. The basic equations for a PID controller are the following:

$$\frac{dx}{dt} = \frac{error}{T}$$

$$y = T\frac{derror}{dt}$$

$$outCtr = K(error + x + y)$$

Using these equations and the BaseController class we create the PID controller:

```
model PIDcontinuousController
  extends BaseController(K=2,T=10);
  Real x "State variable of continuous PID controller";
  Real x "State variable of continuous PID controller";

equation
```

```
  der(x)=error/T;
  y=T*der(error);
  outCtr=K*(error + x + y);
end PIDcontinuousController;
```

We can now compose a PID controlled-tank system using drag-and-drop.

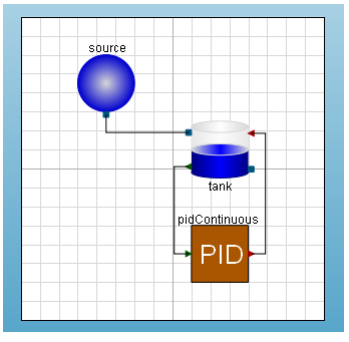

Figure 7-7: The diagram view of the IntroductoryExamples.Hierarachical.TankPID model.

We simulate for 250 seconds again and compare with the previous result.

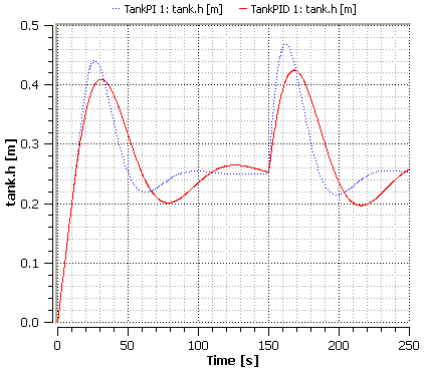

Figure 7-8: Comparison of tank levels between the TankPI model and a TankPID model.

7.4 Three tanks system

Finally, thanks to the object-oriented component-based approach, we can compose a larger system with ease.

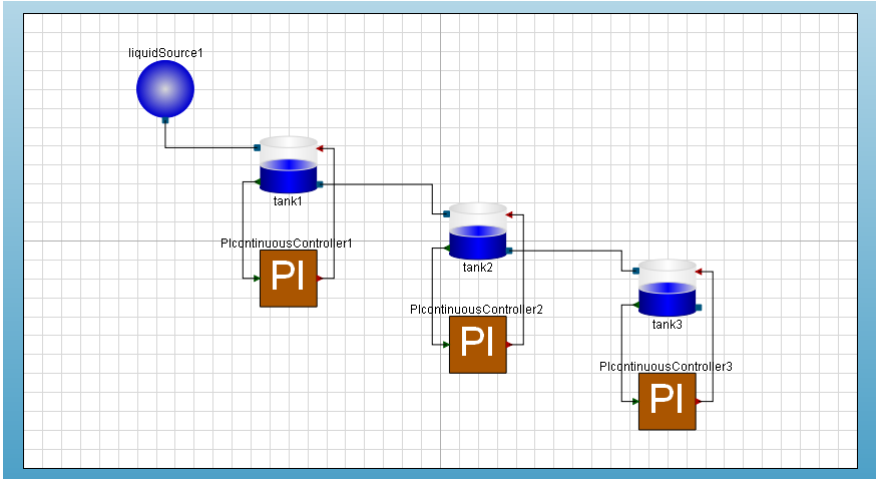

Figure 7-9: The diagram view of the IntroductoryExamples.Hierarachical.TankSystem model.

Simulating this system, we can now study how the tank level of each tank is controlled.

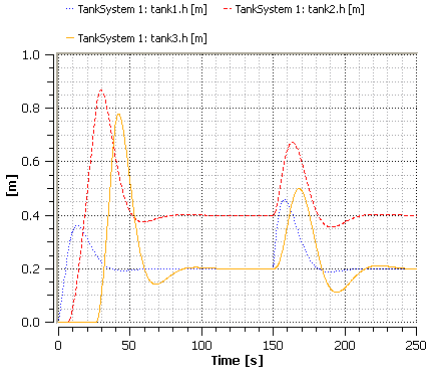

Figure 7-10: Evolution of the tank levels from the TankSystem model.

Note that the second tank has a reference level of 0.4 meters while the other tanks only have a reference level of 0.2 meters.