



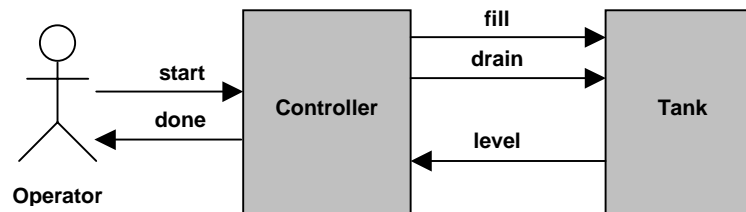
► ► ► Part 2: Case Study Labs

Application Overview

The purpose of the dyeing system is to dye fabric with a specified color. This process is accomplished by filling a tank with liquid dye, placing the fabric in the tank, and letting the fabric soak for a preset time, letting the dye penetrate the fabric and change its color.

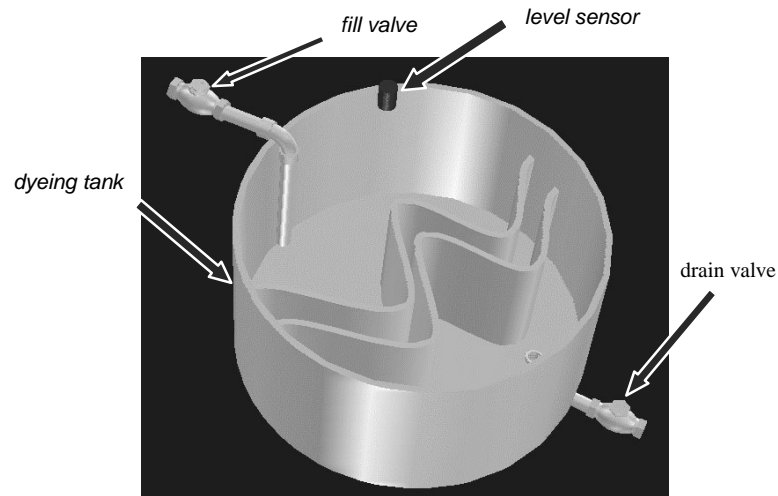
Dyeing System

The diagram below shows a high-level view of a dyeing system. It is composed of a *tank*, within which the dyeing takes place, and a *controller*, which is responsible for sequencing the tank through the dyeing process. An *operator*, external to the system, sets up the controller and then starts the dyeing process.



Tank

The tank holds the fabric to be dyed and the dye solution. The fill valve allows solution to enter the tank, and the drain valve permits its removal. The current level in the tank is reported continuously via the level sensor. The valves are two-state devices (on or off), where the level sensor is a transformational or continuous device, meaning that its output continuously varies as a function of its input.

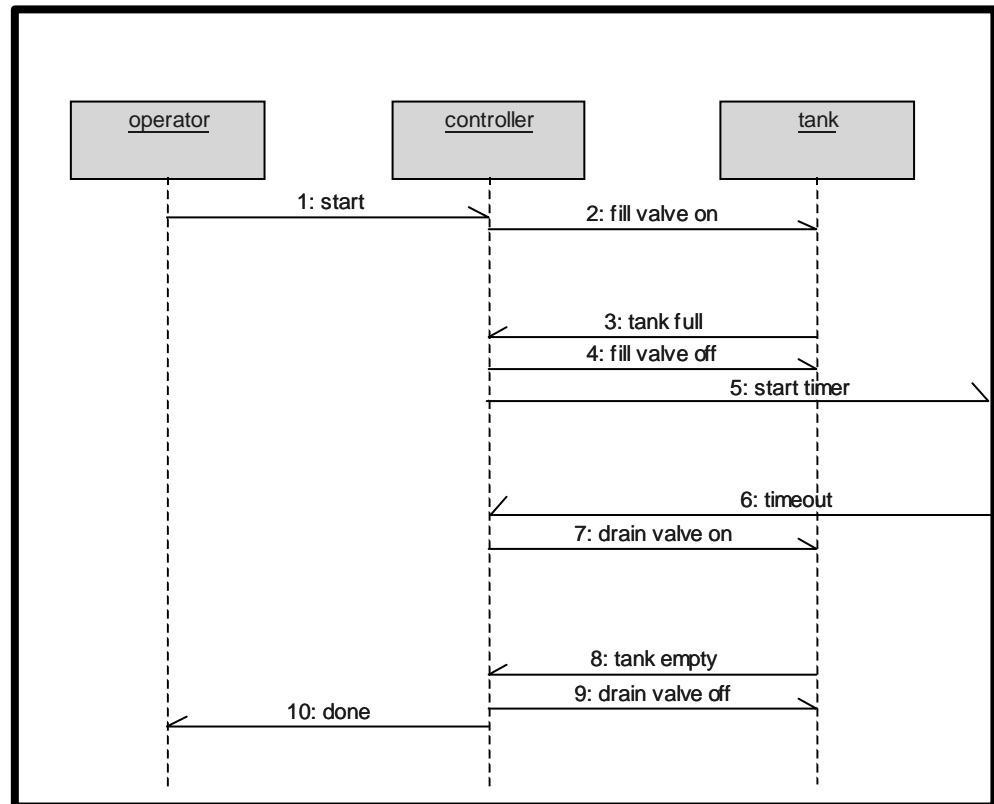


The level sensor provides the necessary feedback to correctly and safely control the dyeing process. This example is a closed-loop system. Conversely, open-loop systems contain no feedback mechanism and must rely on assumptions about the state of the system to control processing.

Controller

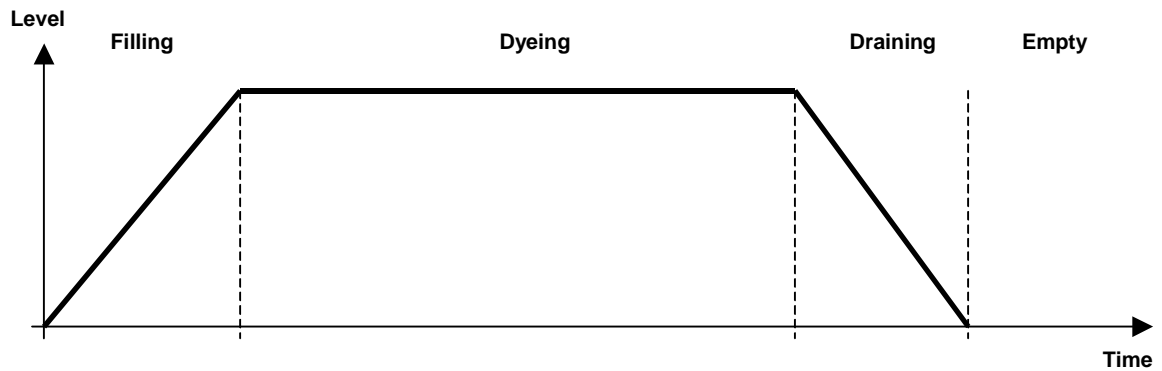
The controller has two primary responsibilities: first, to act on commands from the operator; and second, to fill and drain the tank, and monitor the amount of dye that is in the tank and the amount of time the fabric is exposed to the dye.

Dyeing Run Scenario



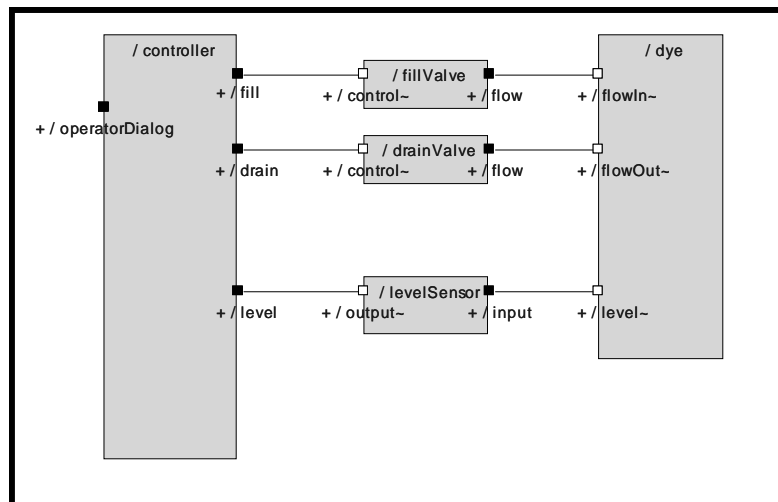
The operator sets up the controller for a dyeing run by specifying a fabric type and dye color, and then initiates the process. The controller immediately opens the fill valve, allowing dye to begin filling the tank. When the controller senses that the tank is full, it closes the fill valve and starts a timer. A predetermined period of time later, the timer fires. This event alerts the controller to open the drain valve, which begins emptying the dye from the tank. When the tank is empty, the controller closes the drain valve, and the process is complete. At this point, the system is ready to process more fabric.

Another View – Level vs. Time



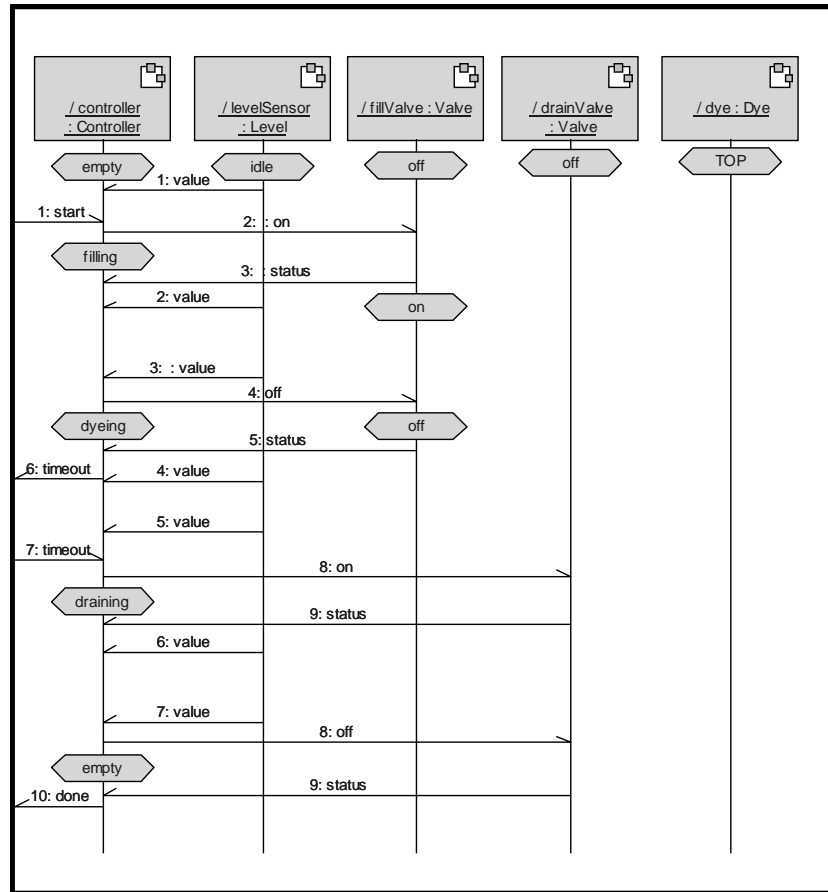
Viewing a process as a function of time can add valuable information about the functioning of a system. The graph above shows the level of the dye as it varies over time during the dyeing process. The major states of the dyeing process are also shown.

Basic Structure of the Dyeing System



There are five basic elements in the dyeing system: the *controller*, *fill valve*, *drain valve*, *level sensor*, and liquid *dye*. The last four (the two valves, sensor, and dye) are elements of the tank. Their interconnection is shown in the diagram above.

Detailed Dyeing System Scenario





▶ ▶ ▶ Lab 1: Controller/Tester

This lab accompanies Module 4, “Capsule Structure: Ports, Protocols, and Capsule Roles.”

In this lab, you will create a simplified version of *DyeingSystem* by using the existing capsule classes, *Controller* and *Tester*. You will create capsules with simple structure and no behavior.

Objectives

When you have successfully completed this lab, you will be able to:

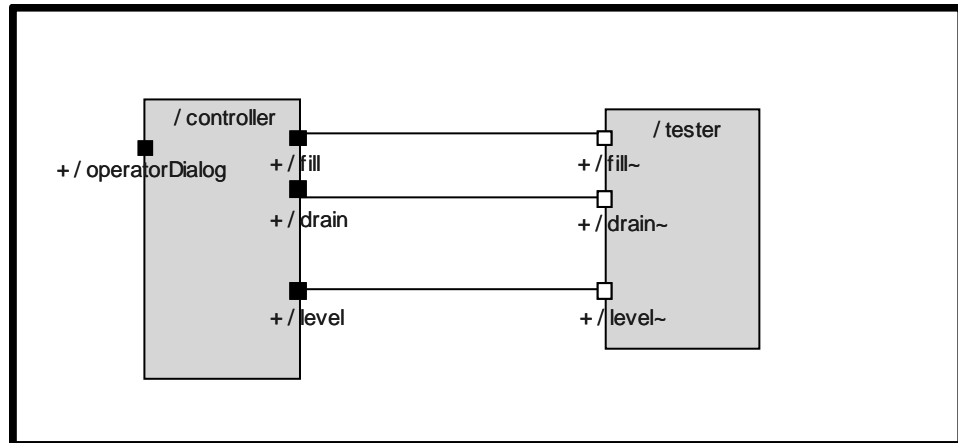
- ▶ Use the basic tool features
- ▶ Build a simple model from existing elements
- ▶ Compile, run, test, and debug the model
- ▶ Generate a sequence diagram from the running model
- ▶ Create a class diagram of the model

Instructions




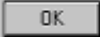

1. Open the model C:\MRRT\CaseStudy\DyeingSystem.rtmml.
 - 1.1. Open the model using **Open** on the File menu.
2. Save the model
 - 2.1. Use **Save Model** on the File menu to save the model in your StudentWork directory.
 - 2.2. When the workspace dialog appears, click to create a workspace.
3. Create a new capsule class named *DyeingSystem*.
 - 3.1. Right-click the Logical View folder and select **New > Capsule**.
 - 3.2. Type the name of the capsule followed by **↵** (Enter) .

Continued on next page.

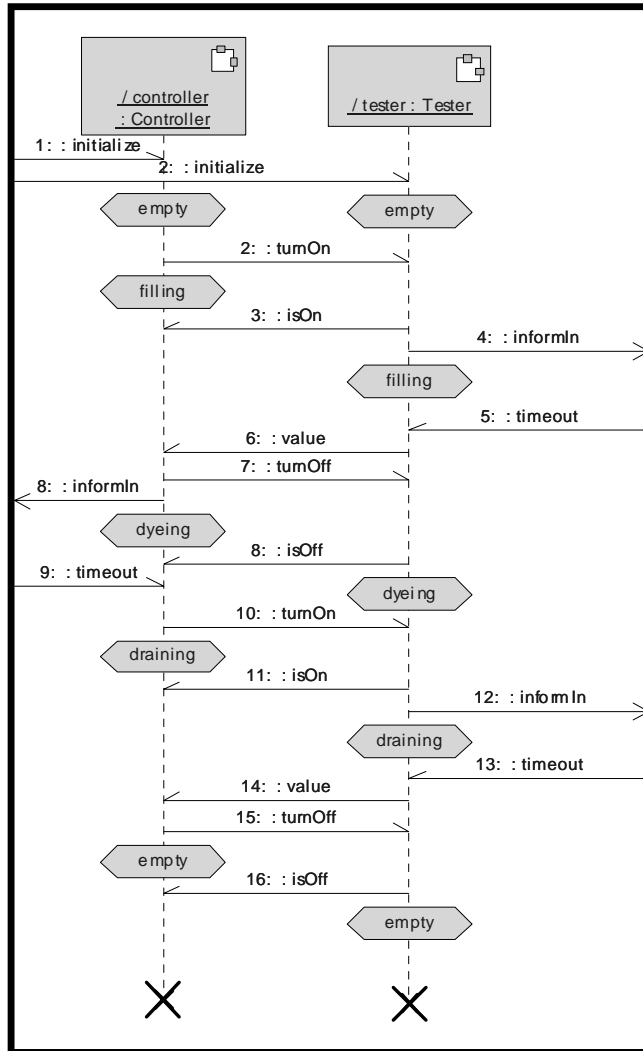
4. In the *DyeingSystem* structure diagram, create the model shown below. Insert one *Controller* and one *Tester* capsule, and connect them as shown.



- 4.1. Right-click *DyeingSystem* in the Logical View folder and select **Open Structure Diagram**.
- 4.2. Drag the *Controller* and *Tester* capsules onto the diagram.
- 4.3. Using the Connector tool, click the *fill* port on the *controller* capsule role, and then drag a line to the *fill~* port on the *tester* capsule role.
- 4.4. Repeat the previous instruction for the *drain* and *level* ports.
5. Build and run the model and correct any errors.
 - 5.1. Create a new component named *DyeingSystem*.
 - 5.2. Drag the top capsule onto the component.
 - 5.3. Set the component to *active*.
 - 5.4. Set the *Top Capsule* and *Target Configuration*.
 - 5.5. Create a new processor named *DyeingSystem*.
 - 5.6. Draw a dependency between the *DyeingSystem* component and *RTComponent*.
 - 5.6.1. Double-click the *Main* component diagram to open it.
 - 5.6.2. Drag the *DyeingSystem* component and the *RTComponent* from the *RTComponents* package onto the diagram.
 - 5.6.3. Use the Dependency tool to draw a dependency relationship from *DyeingSystem* to *RTComponent*.
 - 5.7. Create a component instance.
 - 5.8. Set the component instance to *active*.
 - 5.9. Build and run the component instance.

6. Create a run-time sequence diagram.
 - 6.1. Open the *DyeingSystem* structure monitor window.
 - 6.1.1. Right-click the *DyeingSystem* folder, and select **Open Structure Monitor**.
 - 6.2. Place a probe on the *operatorDialog* port.
 - 6.2.1. Using the Probe tool , click the *operatorDialog* port in the structure monitor. (See Rose RealTime Help topic, "Creating a port probe.")
 - 6.3. Open an inject window on the probe and create a *start* message within it.
 - 6.3.1. Expand the *Probes* folder and double-click the probe to open its specification dialog box.
 - 6.3.2. On the Detail tab, right-click the entry field, and select **Insert**.
 - 6.3.3. Use the **Edit Inject Message** specification to select the signal to be injected.
 - 6.4. Open a trace window on *controller* and *tester*.
 - 6.4.1. Open the Structure Monitor for *DyeingSystem*.
 - 6.4.2. Click on *controller*, then control-click on *tester*.
 - 6.4.3. Right-click on an empty spot on the diagram and select **Open Trace**.
 - 6.4.4. Set the configuration threshold to 250 by right-clicking the Trace window and selecting **Configure**. Change the number in the **Buffer Size** box to 250 and click  .
 - 6.5. Open the *controller* state monitor (maximize).
 - 6.5.1. Right-click the *controller* and select **Open State Monitor**.
 - 6.6. Start the model and inject the *start* message.
 - 6.6.1. Click Start  to run the model.
 - 6.6.2. Double-click *operatorDialog/start* in the probe folder.
 - 6.7. When the *controller* cycles back to the *empty* state, open a sequence diagram from the trace window.
 - 6.7.1. Right-click in the trace window and select **Open Sequence Diagram**.
 - 6.7.2. You can choose whether to save the diagram, and then click  .
7. Click the **Shutdown** button  to halt the execution of the model.

8. Compare the generated sequence diagram with the one shown below.





▶ ▶ ▶ Lab 2: Valve, Start Low-Temp System

This lab accompanies Module 5, “State Modeling.”

This exercise involves creating capsules with simple structure and behavior. You will build the *Flow* protocol and *Valve* capsule class, and then compile and debug the model by removing any compile and run-time errors. Then you will add *Valve* capsule roles to the *DyeingSystem*.

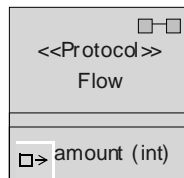
Objectives

When you have successfully completed this lab, you will be able to:

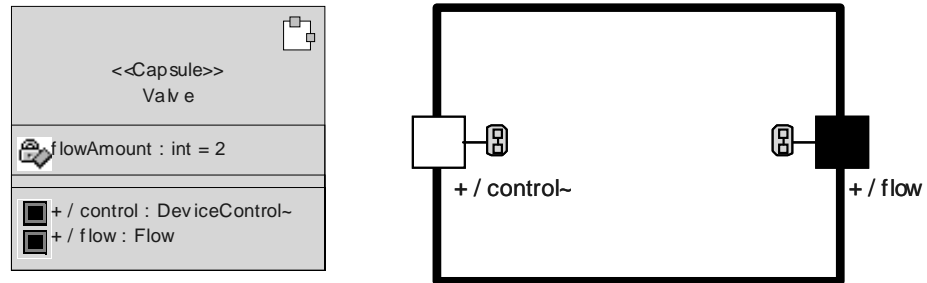
- ▶ Create a capsule with simple structure and simple behavior
- ▶ Begin construction of a complex model (the case study)
- ▶ Create a simple protocol class

Instructions

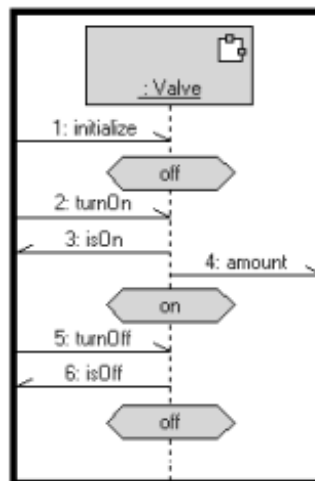
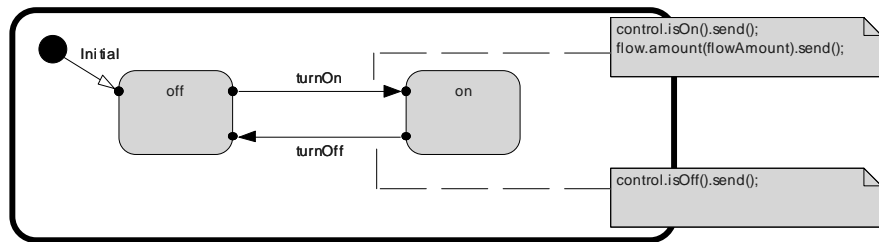
1. Remove *tester* from the *DyeingSystem* structure diagram. It is no longer needed.
2. Create the *Flow* protocol class with an out-signal called *amount* as shown below.



3. Create a capsule class named *Valve* and define its structure and behavior.
 - 3.1. Create the capsule and an integer attribute called *flowAmount*. Set the initial value of *flowAmount* to 2.
 - 3.2. Define *Valve* structure with two ports as shown below:

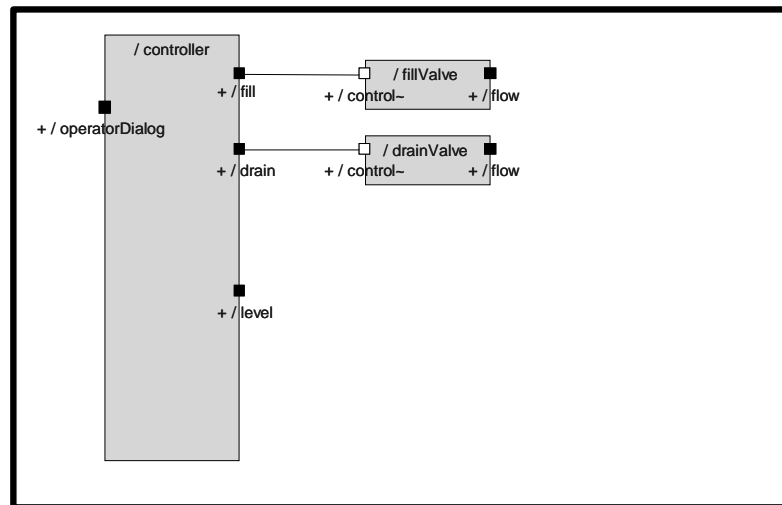


- 3.3. Define *Valve* behavior as shown below. The notes show the code to add to the transitions. Both transition triggers are on the *control* port.



4. Save the model.

5. Test the *Valve* class.
 - 5.1. Compile the *Valve* class. Correct any compilation errors.
 - 5.2. Run the model.
 - 5.3. Add probes to the *control* and *flow* ports, and then open trace windows on them.
 - 5.4. Open an inject window on the *control* port. Create two messages, *turnOn* and *turnOff*.
 - 5.5. Open a state monitor window on *Valve*.
 - 5.6. Start the model and verify that it is running properly by injecting messages and monitoring the trace windows.
 - 5.7. Generate a sequence diagram from the running *Valve* and compare it with the one in the previous lab.
6. Add capsule roles to the *DyeingSystem* class.
 - 6.1. Add two *Valve* references to the *DyeingSystem* structure. Name them *fillValve* and *drainValve*.



- 6.2. Connect them to the appropriate ports on the *controller*.
- 6.3. Compile the *DyeingSystem* and remove any compilation errors.
- 6.4. The entire *DyeingSystem* will be completed and tested in the next exercise.



▶ ▶ ▶ Lab 3: Complete Low-Temp System

This lab accompanies Module 6, “System Services.”

To complete the low-temp dyeing system, you will integrate a timer into the design of the *Valve* capsule class to periodically send *Flow::amount* messages. You will also create the *AcquiredValue* protocol class and *Level* and *Dye* capsule classes. Finally, you will add *Level* and *Dye* capsule roles to the *DyeingSystem*, and then compile and test the *DyeingSystem* as a complete system.

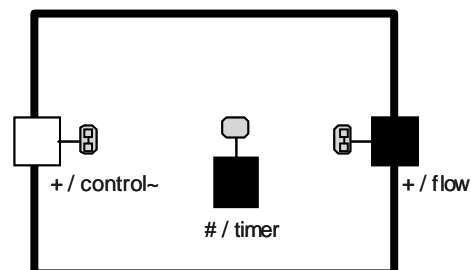
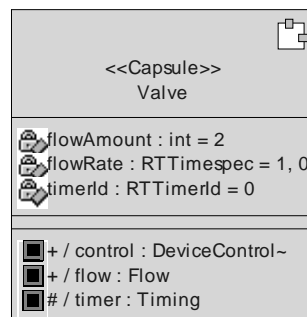
Objectives

When this exercise is successfully completed, you will be able to:

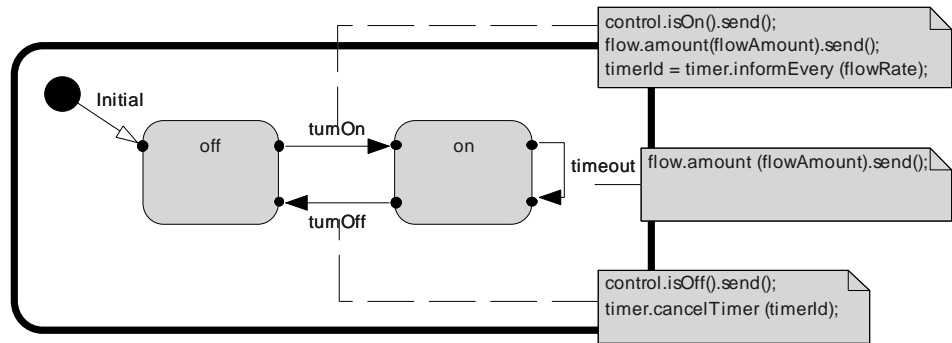
- ▶ Complete construction of a complex model
- ▶ Make use of timers in a model with the Timing service in the services library

Instructions

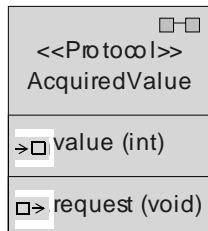
1. Modify the *Valve* capsule class to periodically send *Flow::amount* messages out its *flow* port. Implement this modification using a timer.
 - 1.1. Modify the *Valve* structure by adding an unwired end port based on *Timing*.



1.2. Modify the *Valve* behavior.

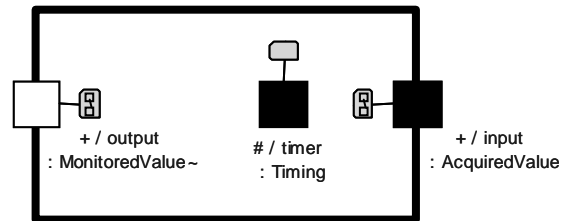
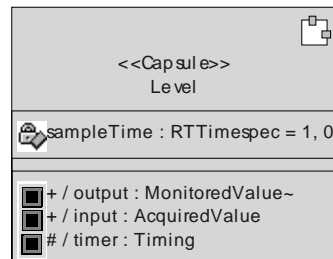


2. Create the *AcquiredValue* protocol class.

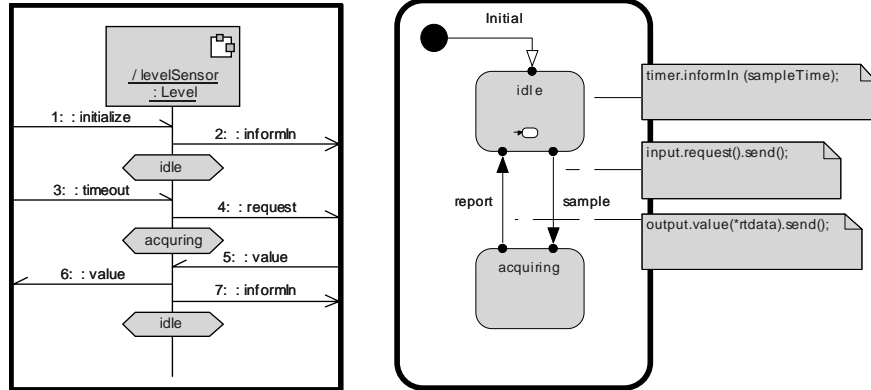


3. Create the *Level* capsule class.

3.1. Define the *Level* structure.

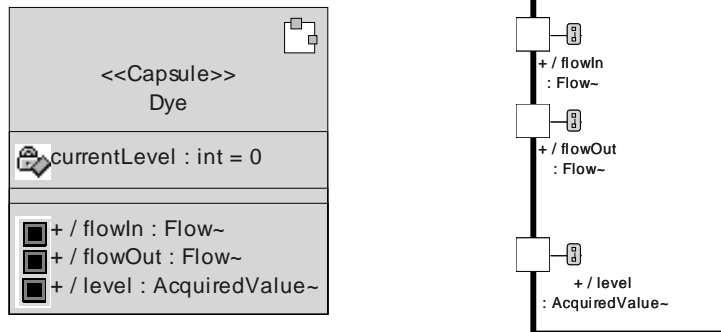


3.2. Define the *Level* behavior as shown before. Notice that there is entry code for the *Idle* state. The trigger for *report* is on the *input* port, the trigger for *sample* is on the *timeout* port.

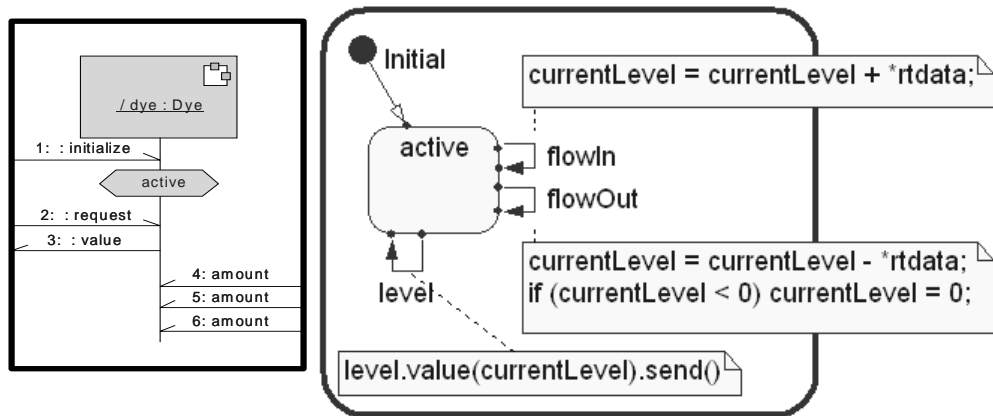


4. Create the *Dye* capsule class.

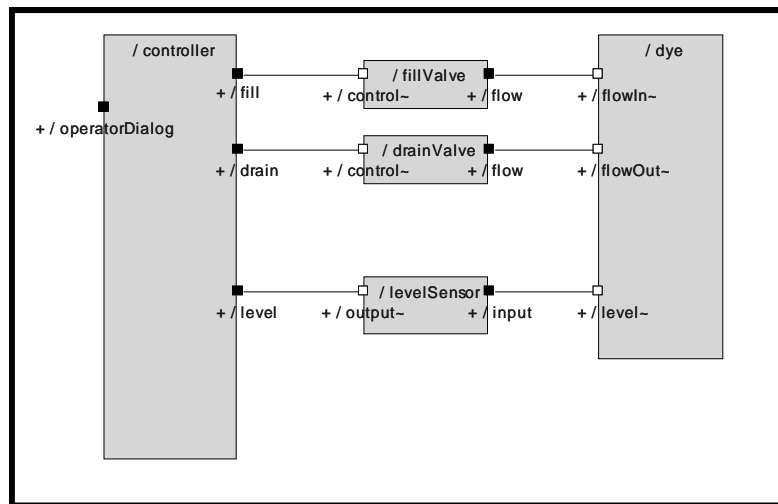
4.1. Define the *Dye* structure.



4.2. Define the *Dye* behavior. Triggers are on the port with the same name as the transition.




5. Add one *Level* and one *Dye* role to the *DyeingSystem*. Name them *levelSensor* and *dye*, respectively. Connect the appropriate ports to complete the definition of the *DyeingSystem* as shown below.



6. Compile the *DyeingSystem*. Remove any compilation errors.
7. System test the *DyeingSystem* by using the “30-Second Test” (instructions below).
8. Generate a sequence diagram from the running model and compare it to the one shown in “Dyeing Run Scenario” on page 3.

The 30-Second Test

1. Load the model into the *Runtime View* (do not run it yet).
2. Open the structure monitor for the *DyeingSystem*.
 - 2.1. Place a probe on *controller's operatorDialog* port.
 - 2.2. Place a probe on *controller's level* port.
 - 2.3. Close the *DyeingSystem's* structure monitor.
3. Expand the *Probes* folder.
 - 3.1. Open an inject window on the *operatorDialog* probe.
 - 3.2. Create a *start* message.
 - 3.3. Close the inject window.
 - 3.4. Open a trace window on the *level* probe.
 - 3.5. Maximize the window size, if it is not maximized.
4. Run the model by clicking **Start** . If your model is working properly, you should see *value* messages being received from the *levelSensor*, indicating a “0” solution level for the *dye*.
5. Expand the *operatorDialog* probe folder so that you can see the individual signals.
6. Start the dyeing process by injecting the *start* message (double-click it).
7. The level should rise from 0 to a maximum of 10 (5 seconds), remain constant at 10 (20 seconds), and then drain back to 0 (5 seconds)—hence, the 30-second test! This is the fastest and simplest way to test this model. Using it throughout the case study will reduce your test and debug times. Don't forget to set the trace threshold to 250. See Lab 1, step 6.4 for instructions.



▶ ▶ ▶ Lab 4: Master and Tank Containers

This lab accompanies Module 9, “Structure Hierarchies.”

This is a two-part lab in which you will reorganize the dyeing system model to make it easier to work with. In the first part, you will use the Rose RealTime *aggregation* feature to create container capsule classes *Tank* and *Master*. Remember that the aggregation feature is a tool that allows you to combine capsules. It is not the same as the UML aggregation relationship.

In the second part of the lab, you will use replication to create multiple tanks and controllers.

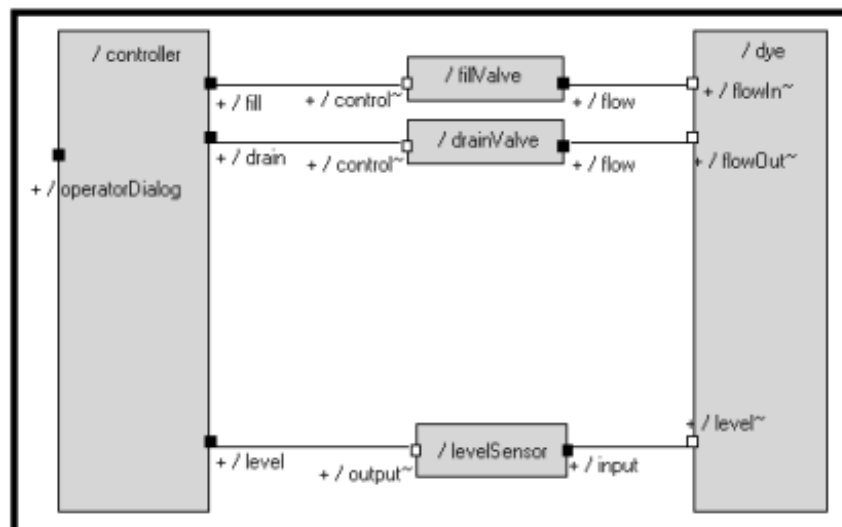
Objectives

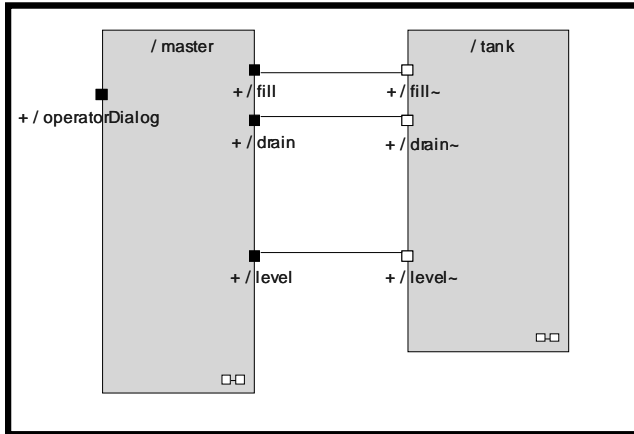
When you have successfully completed this lab, you will be able to:

- ▶ Create structure hierarchies (capsules containing other capsules) in an existing model
- ▶ Apply the technique of aggregation
- ▶ Apply replication (cardinality) to capsules and ports
- ▶ Compile, run, and debug a structurally complex model

Instructions: Part 1 — Aggregation (of Structure)

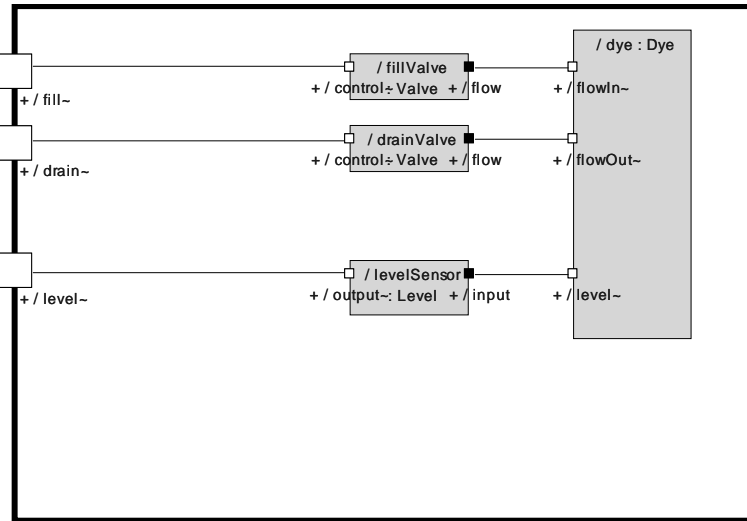
Here is the *DyeingSystem* capsule structure as it is now.





1. Open the *DyeingSystem* model if it's not already open.
2. Aggregate *fillValve*, *drainValve*, *dye*, and *levelSensor* into a new capsule class. The new *DyeingSystem* capsule structure looks like the illustration above.
 - 2.1. Open the *DyeingSystem* structure diagram and select the valves, sensor and *dye* capsule roles. (You can either click and drag or Shift+click.)
 - 2.2. Right-click the empty part of the diagram and select **Parts > Aggregate**. (If you click the selected capsule roles you will get the wrong menu.) The result should look similar to the diagram above.
 - 2.3. Right click the *NewCapsule1* in the browser and select **Rename** to change the name to *Tank*.

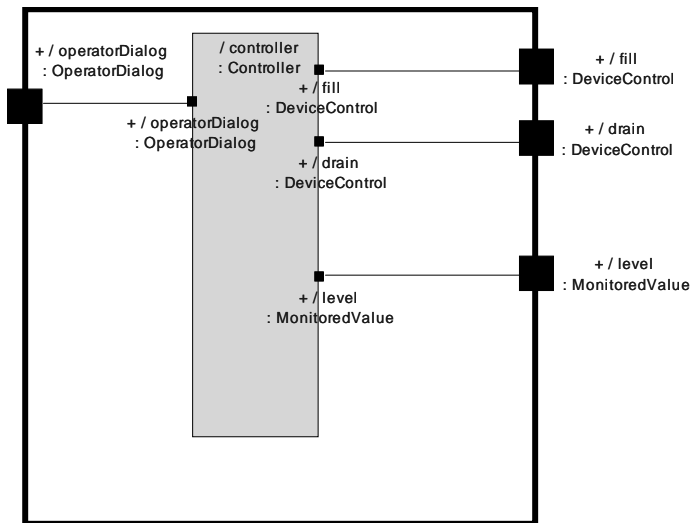
2.4. Double-click the title of the new capsule role and name it *tank*. If you double-click the new tank capsule role you will see its structure, which looks like the following diagram.



3. Aggregate the *controller* into a new capsule class.

3.1. Follow the directions in step 1 and name the new capsule class *Master*.

3.2. Name the new capsule role *master*.




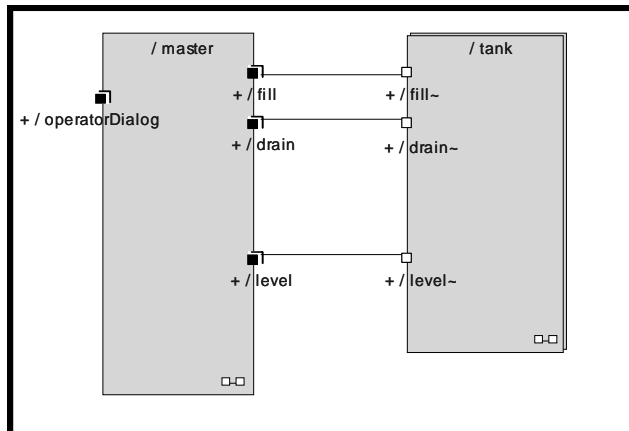
4. Add a relay port named *operatorDialog* to *Master* based on the protocol class *OperatorDialog*. Connect it to *operatorDialog* on *controller*.

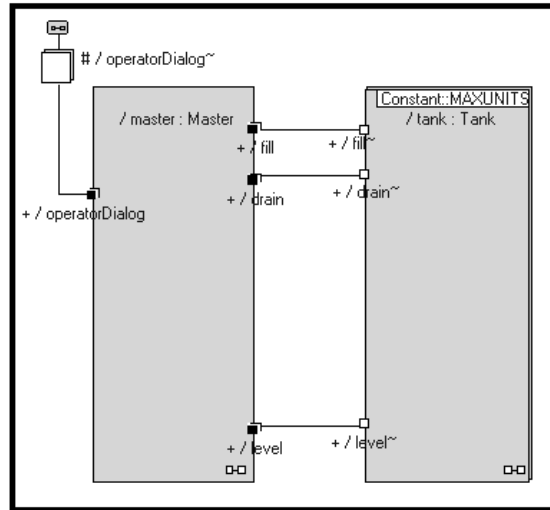
5. Save the model.

6. Perform the “30-Second Test” on *DyeingSystem* to ensure that it still functions properly. To perform the 30-Second Test for this lab, the model must be running in order to add the probe to the *fill* port. Also, place the probe to start the system on the *operatorDialog* port on *master* instead of on *controllerR1*.

PART 2 – Replication

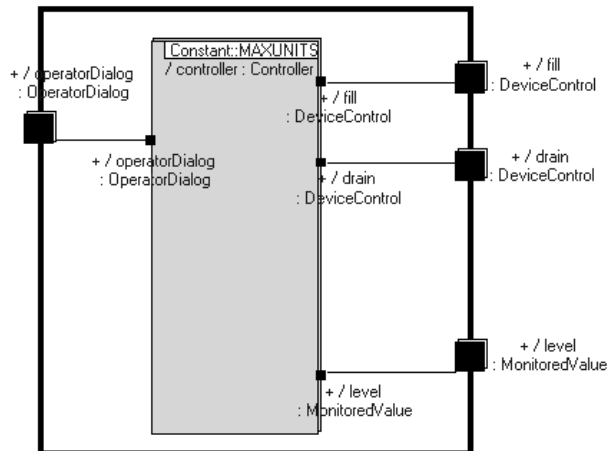
1. Create a global constant named *Constant::MAXUNITS*.
 - 1.1. Create the passive class *Constant*.
 - 1.1.1. Right-click the Logical View folder and select **New > Class**.
 - 1.2. Add an attribute named *MAXUNITS* based on *int*.
 - 1.2.1. Open the Class Specification dialog box to the General tab. Set **Visibility** to *Public* and **Type** to *Class*.
 - 1.2.2. Click the Attribute tab. Right-click in the entry area and select **Insert**. Name the attribute *MAXUNITS*.
 - 1.2.3. Double-click the name of the attribute to open the Class Attribute Specification dialog box. On the Detail tab, set **Type** to *int*, **Initial value** to 1, and **Changeability** to *Frozen*.
 - 1.2.4. On the C++ Tab, set *AttributeKind* to global.
 - 1.2.5. Click  on both dialogs.
2. Replicate the specified capsule roles within *DyeingSystem*, using a cardinality of *Constant::MAXUNITS*.
 - 2.1. Add a conjugated, wired, protected end port based on *operatorDialog* to *DyeingSystem*'s structure and connect it to the matching port on the *master* capsule role. Set the cardinality to *Constant::MAXUNITS*.
 - 2.2. Set the *tank* capsule role within *DyeingSystem* to a cardinality of *Constant::MAXUNITS*.







2.3. Set the *controller* capsule role within *master* to a cardinality of *Constant::MAXUNITS*.

2.4. Set the relay ports on *master* to a cardinality of *Constant::MAXUNITS*.



3. Save the model.
4. Perform the “30-Second Test” on *DyeingSystem*.
5. Change *Constant::MAXUNITS* to 3.
6. Save the model.

7. Perform the “*30-Second Test*” on *DyeingSystem*. Be sure to monitor all 3 controller/tank pairs for proper operation.
 - 7.1. Put a probe on the *level* port of each controller.
 - 7.1.1. Click **Start**  .
 - 7.1.2. Expand the *0/master:Master* folder.
 - 7.1.3. For each of the controllers, open a Structure Monitor and put a probe on the *level* port.
 - 7.2. Open a trace window on each probe.
 - 7.3. To simplify starting all three controllers, place a single probe on the conjugated *operatorDialog* port connected to *master*, and then create a single *start* message. To do this, open the Edit Inject Signal Specification as before, and change the Direction selection to Out. Select the *Start* signal and continue as usual.s
 - 7.4. Click **Start**  .
 - 7.5. Inject a single *start* message through the *operatorDialog* probe. This action has the advantage of broadcasting *start* to each of the (replicated) *operatorDialog* ports.



▶ ▶ ▶ Lab 5: Start High-Temperature System

This lab accompanies Module 10, “Inheritance Hierarchies.”

In this lab you will learn to manage model complexity by using inheritance hierarchies. You will create a high-temperature dyeing system by superclassing and subclassing existing capsule, protocol, and passive classes in *DyeingSystem*.

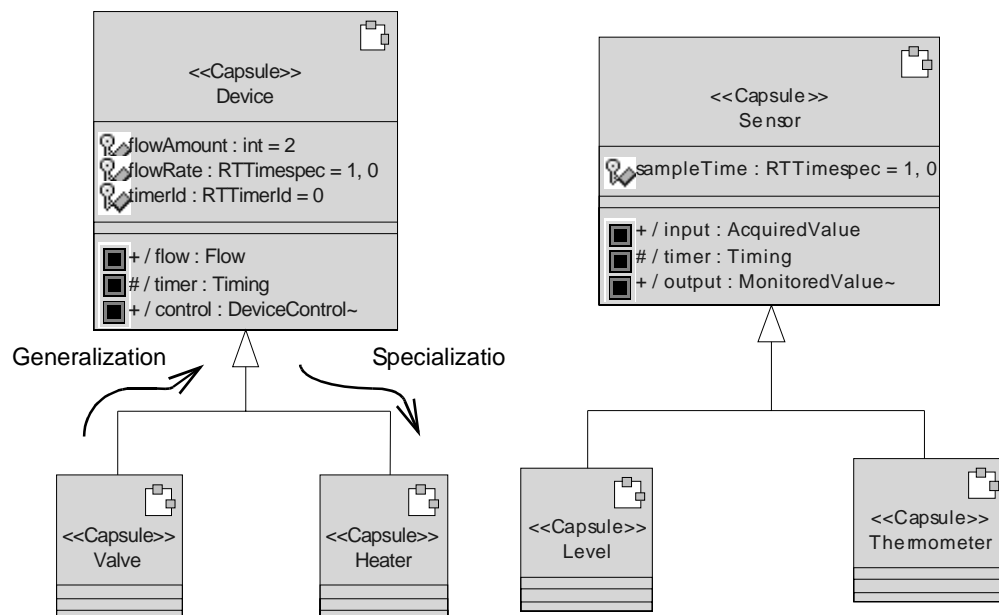
Objectives



When you have successfully completed this lab, you will be able to:

- ▶ Create an entirely new system by constructing inheritance hierarchies
- ▶ Apply bottom-up design to an inheritance hierarchy (*generalizing* an existing class)
- ▶ Apply top-down design to an inheritance hierarchy (*specializing* an existing class)
- ▶ Compile, run, and debug a model containing inheritance hierarchies

Instructions

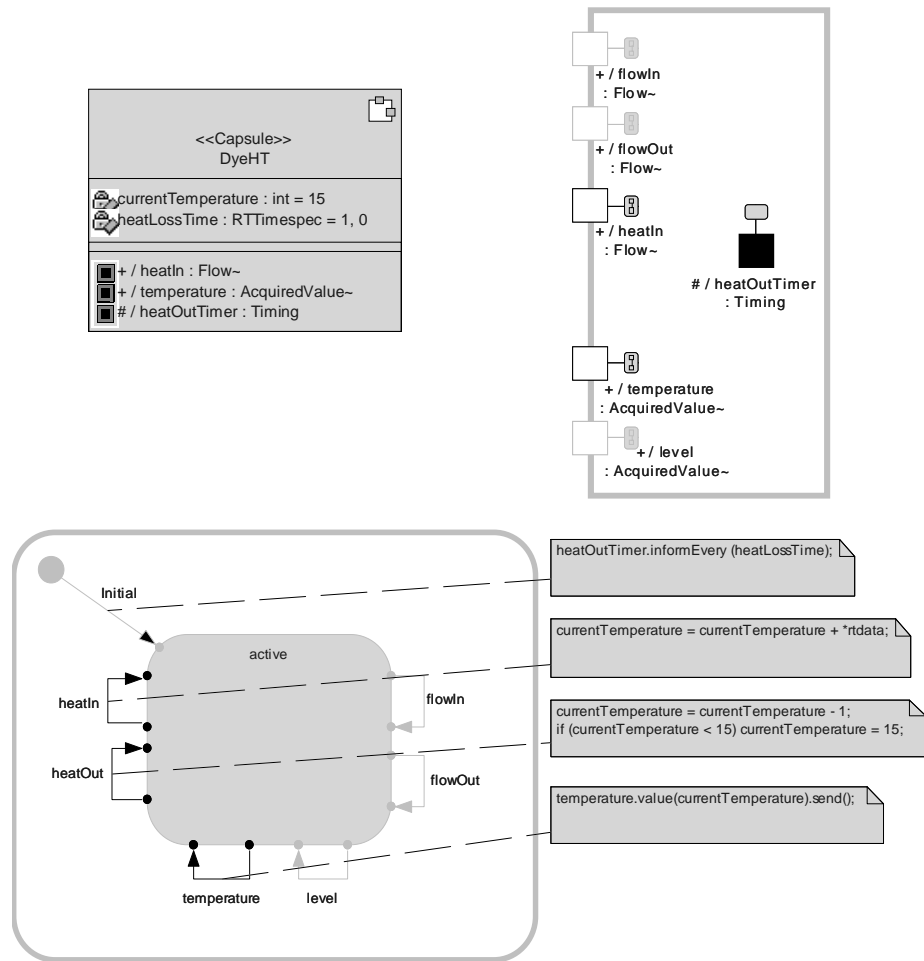
PART 1 – Create Inheritance Hierarchies



1. Start with the interaction of *DyeingSystem* you completed in the last exercise.
2. Change *Constant::MAXUNITS* to 1. This will simplify testing this phase of the design.
3. Make *Valve* a subclass of a new capsule called *Device*.
 - 3.1. Create a new capsule class named *Device*.
 - 3.2. In the *Logical View*, create a class diagram named *Inheritance*.
 - 3.3. Drag both the *Device* and *Valve* capsules onto the class diagram.
 - 3.4. Using the Generalization tool , draw a line from *Valve* to *Device*. You will see the Inheritance Rearrangement dialog box. Click  .
 - 3.5. Promote all the structure elements of *Valve* up into *Device*.
 - 3.5.1. Open the structure diagram for *Valve*.
 - 3.5.2. Select any ports, capsule roles, and connectors.
 - 3.5.3. Right-click the diagram background and select **Parts > Promote**.
 - 3.6. Promote all the behavior elements of *Valve* up into *Device*.
 - 3.6.1. Open the State diagram for *Valve*.
 - 3.6.2. Select all states, right-click the diagram background, and select **Parts > Promote**.
 - 3.6.3. Select all transitions and choice points, right-click the diagram background, and select **Parts > Promote**. (States have to be promoted first so that the transitions have definition.)
 - 3.7. Promote attributes and operations
 - 3.7.1. Open the Capsule Specification dialog box for *Valve* and click the Attribute tab.
 - 3.7.2. For each attribute, right-click the attribute name and select **Promote**. (If there were operations, you would go to the Operation tab and promote each operation in the same way.)
 - 3.7.3. Open the Specification window for *Device*. Set the visibility of all attributes to *Protected* (allows them to be inherited).
4. Create a new capsule class, *Heater*, as a subclass of *Device* (*Class Diagram*). This step is done in the same way as steps 3.1 through 3.4.
 - 4.1. In *Heater*'s initial transition, set *flowAmount* to 3.
5. Make *Level* a subclass of a new capsule class named *Sensor*. Follow all instructions as in step 3.
6. Create a new capsule class named *Thermometer* as a subclass of *Sensor*.
 - 6.1. In *Sensor*'s initial transition, set *sampleTime* to 0.5 second (0s, 500000000 ns).
7. Build the model and remove any errors.

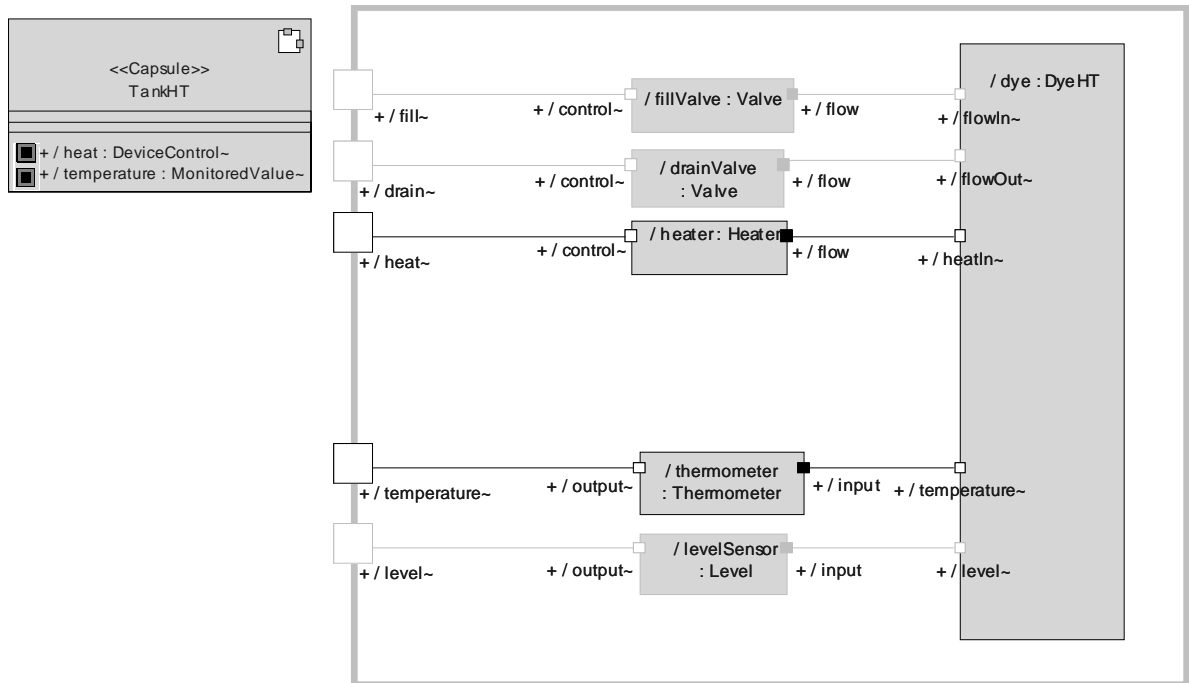
PART 2 – Use Subclasses

1. Create a subclass of *Dye* named *DyeHT*.
 - 1.1. Modify the *DyeHT* structure and behavior.



- 1.2. Create a new component, *DyeingSystemHT*, based on the capsule class *DyeHT* (the top-level capsule).
- 1.3. Create a new component instance, *DyeingSystemHT*, based on the component *DyeingSystemHT*.
- 1.4. Build *DyeingSystemHT* and remove any compilation errors.

2. Create a subclass of *Tank* named *TankHT*.

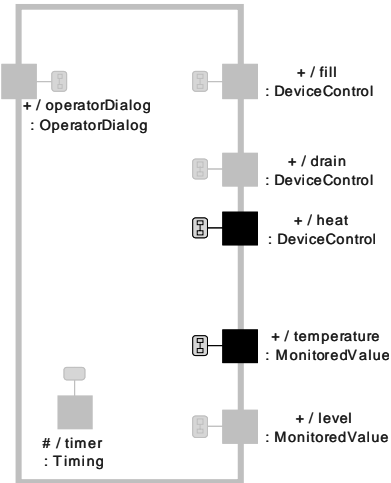


- 2.1. In *TankHT*, override the *dye* capsule role's class by changing it from *Dye* to *DyeHT*.
 - 2.1.1. Open the structure diagram for *TankHT*.
 - 2.1.2. Right-click the *dye* capsule role and select **Open Specification**.
 - 2.1.3. On the **General** tab, change the **Class** selection in the drop-down box to *DyeHT*.
- 2.2. Add a *Heater* capsule role named *heater*.
- 2.3. Add a *Thermometer* capsule role named *thermometer*.
- 2.4. Create relay ports on *TankHT* to pass heater control commands to the heater and temperature readings from the thermometer to the outside world.
- 2.5. Connect *heater* and *thermometer* to the appropriate ports on *dye* and to the relay ports on *TankHT*.
3. Save the model.
4. Unit test *TankHT* to ensure that it functions properly.
 - 4.1. Build and test *TankHT*.
 - 4.1.1. Add probes.
 - 4.1.2. Turn the *heater* on at its *control* port, and watch the temperature rise on *thermometer's* input.

4.1.3. Turn the *heater* off and watch the temperature drop to room temperature (15° C).

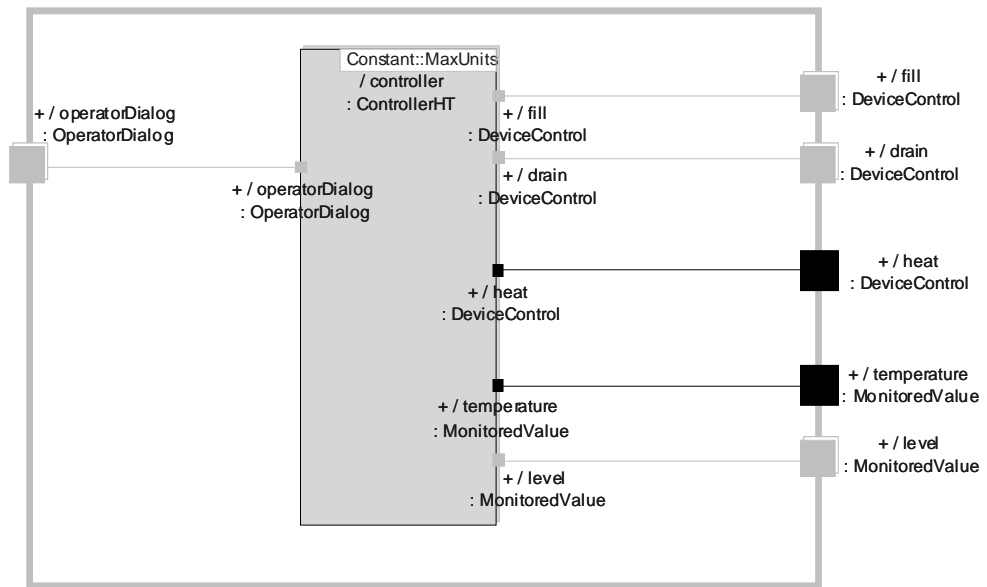
5. Create a subclass of *Controller*, named *ControllerHT*.

5.1. Modify the *ControllerHT* structure (add *heat* and *temperature* end ports).



5.2. Create a subclass of *Master* named *MasterHT*.

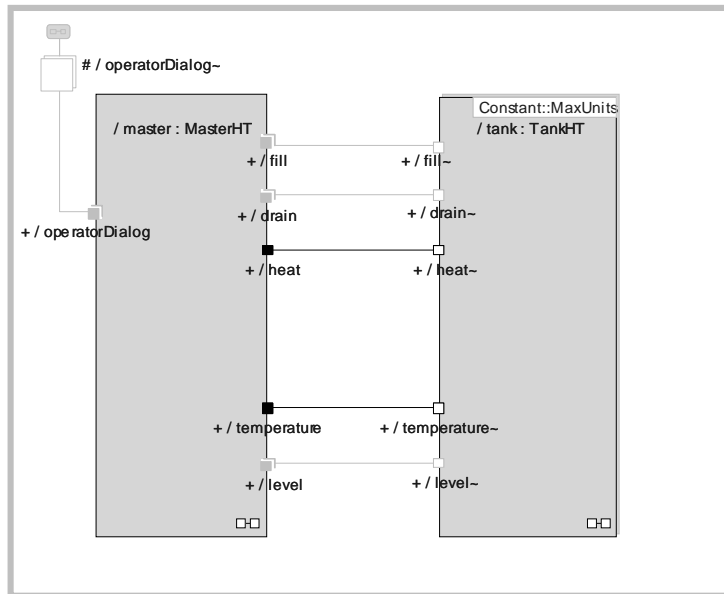
5.3. Modify the *MasterHT* structure (override *controller* capsule role with *controllerHT*, add relay ports).



6. Create a subclass of *DyeingSystem* named *DyeingSystemHT*.

6.1. Modify the *DyeingSystemHT* structure.

6.1.1. Override the *tank* capsule role with *tankHT* and the *master* capsule role with *masterHT*.



6.2. Connect the high-temperature ports between *master* and *tank*.

7. Save the model.

8. Compile the *DyeingSystemHT*. Remove any compilation errors. Do *not* test the model; *it will not run*—the *ControllerHT* behavior requires modification in the next lab before it will function properly.



▶ ▶ ▶ Lab 6: Complete High-Temperature System

This lab accompanies Module 11, “Behavior Hierarchies.”

In this lab you will learn to manage behavioral complexity by constructing hierarchical finite state machines (FSMs). You will modify the behavior of *Controller* so that it becomes a hierarchical state machine. Then you will add thermostatic behavior to *ControllerHT* so that it can properly regulate the temperature of the liquid dye in the tank.

Objectives

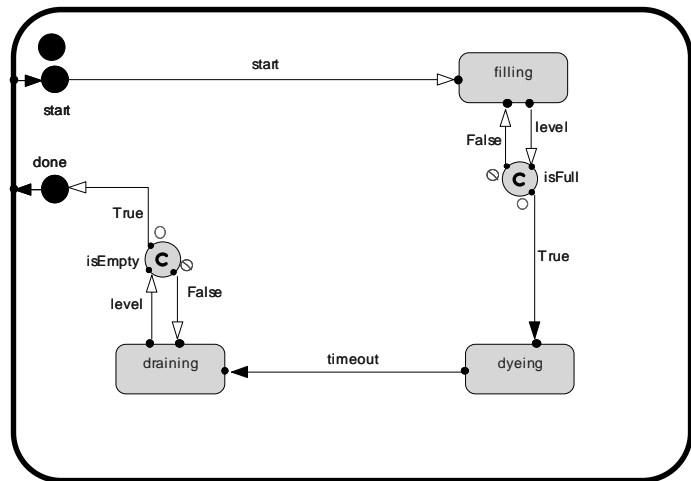
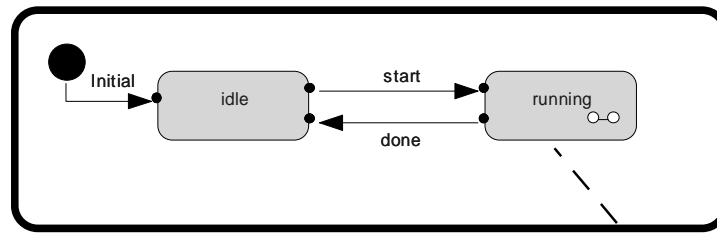
When you have successfully completed this lab, you will be able to:

- ▶ Create a hierarchical FSM in an existing model
- ▶ Distribute behavior across a model’s hierarchical FSM
- ▶ Compile, run, and debug a model containing a hierarchical FSM

Instructions

PART 1 – Aggregation (of Behavior)

1. Modify the behavior of *Controller*.
 - 1.1. Aggregate the *filling*, *dyeing*, and *draining* states and choice points into a state named *running*.
 - 1.2. Rename the state *empty* to *idle* (see below).
2. Save the model.
3. Perform the “30-Second Test” on *DyeingSystem* to ensure that it still functions properly.



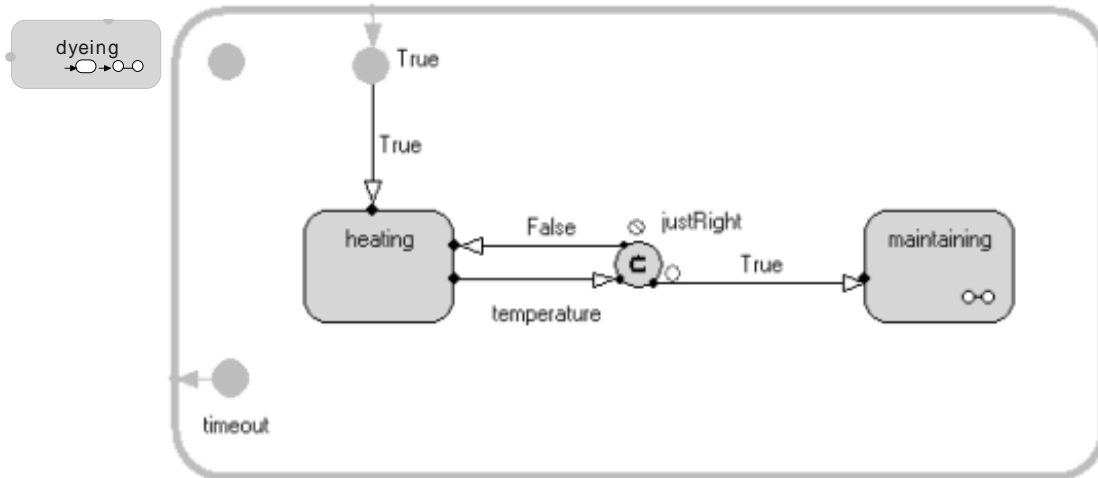
PART 2 – Create Behavior Hierarchy

1. To *ControllerHT*, add an attribute named *dyeingTemperature* (*int*), with an initial value of 30.
2. Modify *ControllerHT*'s behavior to maintain the dye at a fixed temperature.
 - 2.1. Add code in *dyeing*'s entry code to turn on the heater:
 - 2.1.1. Open the state diagram for *ControllerHT*.
 - 2.1.2. Double-click *running* to get to the next level in the state hierarchy.
 - 2.1.3. Right-click *dyeing* to open the State Specification dialog box and go to the Entry Actions tab.
 - 2.1.4. Type the following code in the **Code** area:


```
heat.turnOn().send();
```
 - 2.2. Select the Exit Actions tab of the State Specification dialog box, and place the following code in the **Code** area to turn off the heater:

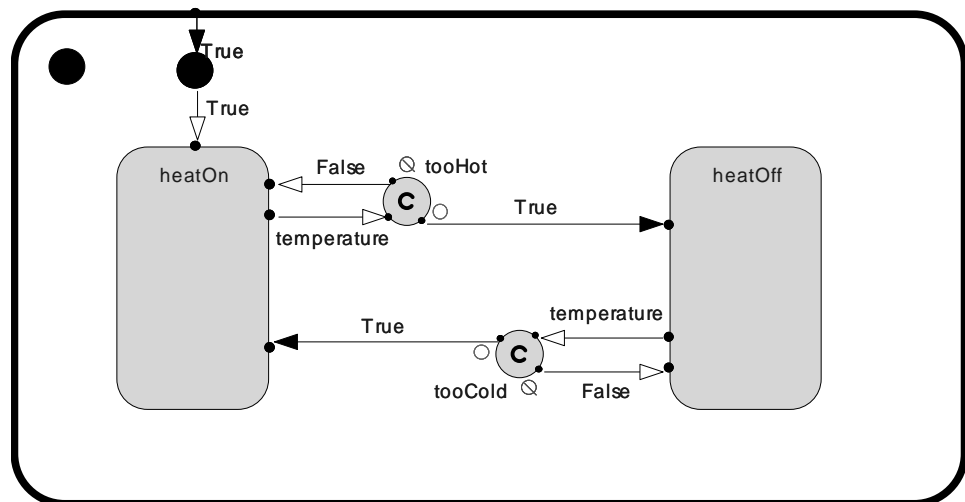

```
heat.turnOff().send();
```

3. Create two new states within *dyeing* named *heating* and *maintaining*.



- 3.1. Draw a transition from *dyeing's* *True* junction to *heating*.
- 3.2. Create a choice point named *justRight* between *heating* and *maintaining*. Connect them with the appropriate transitions.
- 3.3. Place code in the choice point to test when the temperature is greater-than-or-equal-to *dyeingTemperature*.

```
return (*rtdata >= dyeingTemperature);
```
4. Create two states within *maintaining*, named *heatOn* and *heatOff*.



- 4.1. Draw a transition from *maintaining's* *True* junction to *heatOn*.
- 4.2. Create two choice points, named *tooHot* and *tooCold*, between *heatOn* and *heatOff*. Connect them with the appropriate transitions.

- 4.3. Place code in the choice points *tooHot* and *tooCold* to maintain the temperature within a 2-degree margin above and below *dyeingTemperature*.
return (*rtdata >= (dyeingTemperature + 2));
return (*rtdata <= (dyeingTemperature - 2));
- 4.4. Turn the heater off when *tooHot* is *True*. Turn the heater on when *tooCold* is *True*.
5. Save the model.
6. System test *DyeingSystemHT* using the “30-Second Test.” Be sure to monitor both the level and temperature to ensure the temperature is being properly maintained (it should toggle around 30° C).



▶ ▶ ▶ Lab 7: Deliverable and Test Harness / Low and High Temperature

This lab accompanies Module 12, “System Hierarchies: Packaging and Layering.”

In this lab you learn to manage model complexity by using packages. You will organize the Dyeing System application into two categories, Deliverable and Test Harness, and Low Temperature and High Temperature.

Objectives

When you have successfully completed this lab, you will be able to:

- ▶ Create packages and place existing model elements within them
- ▶ Create a containing capsule
- ▶ Add simple structure to that capsule by adding capsule roles
- ▶ Compile, run, and debug the model

Instructions

Create packages named *Deliverable*, *TestHarness*, and *Common*. With the exception of *DyeingSystem* and *DyeingSystemHT*, place all the model elements in the appropriate packages. Within each of the packages (except *Common*), create a high-temperature package and place the high-temperature model elements with them.

