The Coffee Machine

1 Behaviour of the coffee machine

The coffee machine in the example provides the user with a cup of coffee or a cup of hot water (for tea), given that the customer has inserted the required amount of money. The coffee machine can handle coins of the values 5 and 10, where 5 is the price for a cup of tea while a cup of coffee costs 10. The following holds:

- If a coin with the value of 10 is inserted and the Coffee button is pressed, the customer receives a cup of coffee.
- If a coin with the value of 10 is inserted and the Tea button is pressed, the customer receives a cup of hot water plus change.
- If a coin with the value of 5 is inserted and the Coffee button is pressed, the money is returned.
- If a coin with the value of 5 is inserted and the Tea button is pressed, the customer receives a cup of hot water.

2 Model and Diagrams

2.1 General

A diagram is a view of a UML model showing a set of elements and their relations. In this tutorial you will use the capabilities of the tool to work both directly in the model and through diagrams. You will work with different diagrams where each of these diagrams present a certain view of the model. When a new entity, for example a class, is drawn in a diagram, it becomes part of the model. To present different views, the same class may be drawn again in the same diagram or in a different diagram. The information stored in the model is the sum of all descriptions made of an entity.

2.2 Deleting entities

The distinction between the model and the diagrams must be kept in mind when designing in Tau/Modeler. As stated above, an entity, for example a class, is included in the model as soon as it is added to a diagram. Deleting a class from a diagram does not mean that the class is removed from the model! The reason for this is that the same class could be used in other diagrams. It is however possible to delete an entity from the model. By right- clicking an entity there will appear a context-sensitive shortcut menu. From this menu *Delete from Model* can be clicked instead of *Delete*.

2.3 Diagram element creation toolbar

The Diagram element toolbar is only active when the diagram is used. Click in the diagram to activate the toolbar. Buttons in the *Diagram element creation* toolbar are normally handled so that you click on the toolbar button, then click the diagram to position the entity controlled by the button. Toolbar quick-buttons are in sometimes context-sensitive, so the result may depend on selections in the current diagram and the relation between the selection and the button entity. It is often possible to access a shortcut menu (right-click) to get assistance from the model semantics. An example of this is when you draw messages in sequence diagrams, you click on the Message Line button, click on the sender lifeline and then right-click on the receiver lifeline and the shortcut menu will have a drop-down box with all signals in the current scope.

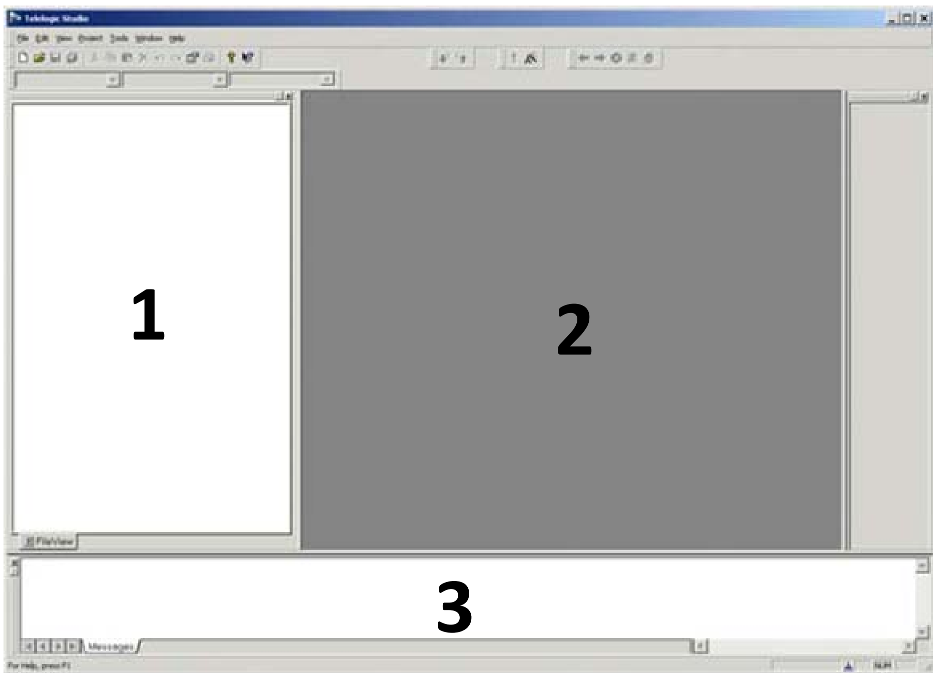
3 User Interface

3.1 General

The Tau/Modeler user interface main areas:

1. the Workspace window
2. the Desktop
3. the Output window

In this tutorial you will sometimes have other areas active, for example watch windows during testing. It is also possible to drag an area to alter its position within the window or even to position it outside the main window of the user interface.



3.2 Workspace window

The Workspace window presents the entities contained in a model. Different views are available for displaying different kinds of information. The *File View* shows all elements that are represented as files. The *Model View* contains all UML elements. This is the view to select when adding diagrams or working directly in the model.

3.3 Desktop

The Desktop is the area where diagrams and documents appear when opened. This is where you work with your model as it is viewed from different diagrams.

3.4 Shortcut bar

The Shortcut bar, which is optional, gives you the possibility to put any frequently used toolbar in a special area, or to create shortcuts to your own scripts.

3.5 Output window

The Output window is used for logging events and displaying errors and warnings.

3.6 Shortcut menu

By right-clicking an entity you will get a shortcut menu. This menu will frequently contain context-sensitive commands.

4 Workspace

4.1 General

A workspace is your personal working area, where you can work in separate projects but also include files that are not related to a specific project. You can define more than one workspace, but you can only work in one workspace at a time. You cannot share a workspace with other users. The information contained in a workspace is stored in a text file with the extension *.ttw.

4.2 Creating a workspace

You will now create a workspace for the development of the coffee machine example. Do the following:

1. Start Tau/Modeler.
2. On the *File* menu, click *New...* The dialog that appears contains four tabs: File, Project, Template and Workspace.
3. Select the *Workspace* tab.
4. Name the workspace “Tutorial” and select the location where the work- space file (Tutorial.ttw) will be stored. Click *OK*. Your new workspace appears in the *File View* of the Workspace window.

The next step is to add a project.

5 Projects

5.1 General

Using projects is a way of grouping the contents within your workspace. You can for example let your workspace “Tutorial” contain several different examples, one being this coffee machine, using one project for each example. Diagrams and documents can be moved between projects. A project is not individual and can therefore be shared between users. The information contained in a project is stored in a text file with the extension *.ttp.

5.2 Creating a project

You will now create a project for the coffee machine example. Do the following:

1. On the *File* menu, click *New...* Select the *Project* tab.

2. A dialog with a number of choices for creating various types of applications appear. Select *UML for Modelling*.
3. Name the project “CMdesign” and select the location where the project files will be stored. Select *Add to current workspace* and click *OK*.
4. A second dialog appears, suggesting a name for the file representing your model and a location for this file. Make sure that *Project with one file and one package* is selected and click *Next*.
5. A third dialog appears, displaying the name of the file representing the project and the name of the file representing the model. Click *Finish*. Your project appears in the Workspace window.
6. Expand[1] the tree structures. The *File View* displays the created files and the *Model View* displays an empty package. In the Model View you will also find information from the internal structures of the Tau/Modeler representation of UML. This information is found in the packages called *Library* and *Predefined*.

6 Use Case Diagrams

6.1 General

Use case diagrams describe the relationships between use cases and actors for a system. In a use case diagram it is possible to group use cases with a subject frame.

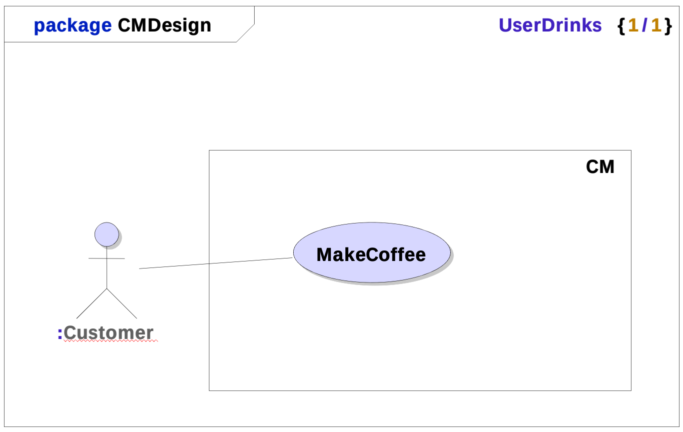
6.2 Creating use case diagrams

You will now create a use case diagram for the use cases for the coffee ma- chine example. These cases were described earlier, see 1.

1. Make sure that the Workspace window displays the *Model View*.
2. Select your package *CMdesign* in the Model View (**not** CMDesign.ttp). Right-click and from the shortcut menu point to *New Diagram* and then click *Use case Diagram***.** Name the use case diagram *UserDrinks*. An icon for the use case diagram appears in the *Model View*. The diagram is now open on the Desktop. Click in the diagram to make it active. The available symbols are now high-lighted in a toolbar. When resting the mouse pointer on a symbol a tooltip appears, indicating the name of the symbol.
3. A symbol is inserted in a diagram by clicking the quick-button representing the symbol, then clicking the diagram on the Desktop. Place an *actor* in the diagram. Give the actor the class *Customer* (for this the class name shall be preceded by a colon according to the syntax). It is possible to give the actor a name, but it is not necessary. The class Customer will not be bound at this stage in the design. This can be observed in two ways. First the type name is underlined with a red wave line indicating a name binding error; secondly there will be some error messages in the Output window (Autocheck tab).

[1] A plus sign (+) to the left of an icon in the Workspace window indicates that the icon is collapsed, i.e. more information can be displayed. To expand the structure for this icon, click the plus sign. An entire substructure can be expanded by selecting a collapsed icon in the Workspace window and pressing the multiplication key (*) on your numeric key pad. To collapse a substructure click on the minus (-) sign for its root icon.

4. Click on the Use case symbol and place it in the diagram. Name the use case *MakeCoffee*. Select the use case and create an association line to the actor. To do this drag the leftmost (*Association line*) of the three line handles from the use case symbol to the actor.
5. Click on the subject symbol and place it in the diagram. Name the subject *CM*.
6. Save your work. You have now completed a use case diagram containing one of the possible use cases for this system. You will add some more use cases to this diagram later when working with sequence diagrams.



7 Class Diagrams

7.1 Class diagram

Class diagrams describe the types of objects that a system consists of, and the relationships between them. They also show attributes and operators of the classes.

7.2 Creating class diagrams

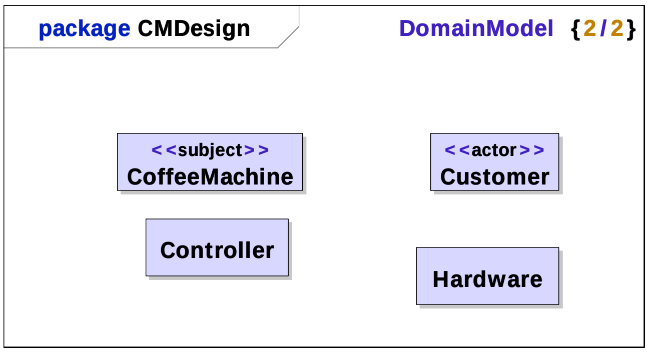
You will now use class diagrams for modelling the coffee machine example.

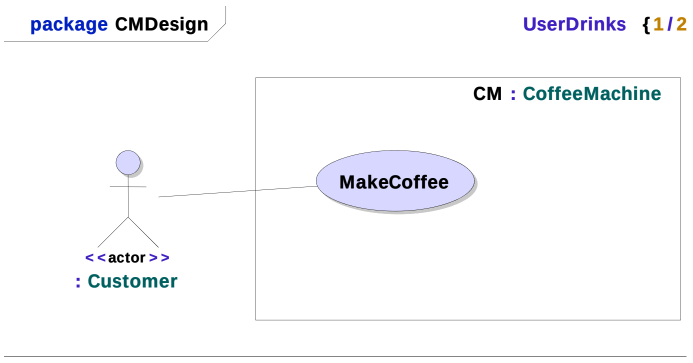
1. Make sure that the Workspace window displays the *Model View*.
2. Select your package *CMdesign* in the Model View. Right-click and from the shortcut menu point to *New Diagram* and then click *Class diagram*.
3. An icon for the class diagram appears in the *Model View*. The diagram is now open on the Desktop. Name the diagram "DomainModel".
4. Drag Customer actor and CM subject from Model View and drop it in the DomainModel diagram in the Desktop. They appear as stereotype of class.
5. Name the class of CM "CoffeeMachine".

7.3 Decomposing a class

You will now refine your model by adding two new classes representing the controller part and the hardware part of the coffee machine. *Controller* will contain the logic for the system, while *Hardware* will simulate the hardware behaviour. When adding these classes, different methods will be used. You will add the classes directly to the model and then use some modelling features of the editor. Do the following:

1. In the Model View of the Workspace window, locate package *CMdesign*. Right-click package *CMdesign*. On the shortcut menu, point to *New* and click *Class*. A class icon appears in the Model View.
2. Name the class *Controller*.
3. In the Model View, select package *CMdesign*.
4. Right-click package *CMdesign*. On the shortcut menu, point to *New Model Element* and click *Class*. A class icon appears in the Model View.
5. Name the class *Hardware*.





8 Paquetage

8.1 General

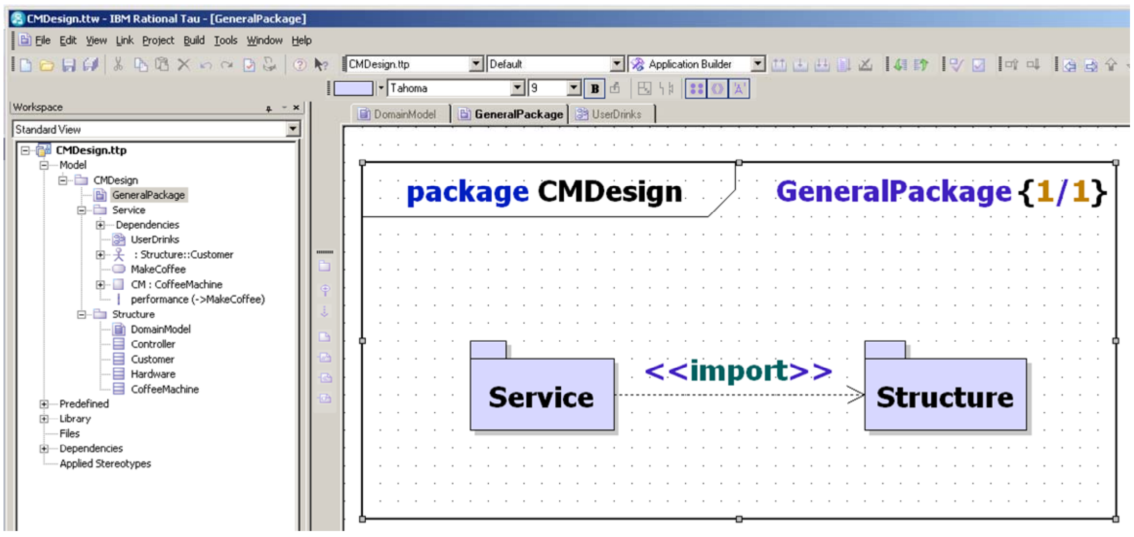
UML packages are used to manage model components.

8.2 Create packages

1. Make sure that the Workspace window displays the *Model View*.
2. Select your package *CMdesign* in the Model View. Right-click and from the shortcut menu point to *New Model Element* and then click *Package*.
3. Name the package *Service*
4. Drag and drop <u>all</u> parts concerned with *Service*
5. Create a second package named *Structure*
6. Drag and drop <u>all</u> parts concerned with *Structure*

Note that one error appears in the use case diagram: *CoffeeMachine* is unknown!

1. Select your package *CMdesign* in the Model View. Right-click and from the shortcut menu point to *New Diagram* and then click *Package diagram*.
2. Name the diagram GeneralPackage
3. Drag and drop the packages *Service* and *Structure*
4. Add an *import* relation from *Service* to *Structure*
5. The use case diagram error disappears!



9 Signals, Interfaces and Ports

9.1 General

UML has a specific class symbol for defining signals. This symbol is one of the predefined stereotypes, indicated by the *<<signal>>* heading. All signals used in the UML model must be defined. A signal defined on a certain level can be seen and used by all entities on lower levels.

9.2 Defining signals

You will now define all signals needed to implement the behaviour of the coffee machine. You will use a class diagram to draw the signal definitions in. Do the following:

1. Create a new package named *Communication*
2. Select your package *Communication* in the Model View. Add a class diagram by right-clicking and from the shortcut menu point to *New Diagram* and then click *Class diagram*. The diagram appears in the Model View of the Work- space window.
3. Name the diagram *Signals and Interfaces*. The diagram is now open on the Desktop.
4. From the toolbar, select a Signal symbol and place it in the diagram. Define the signals (one Signal symbol per signal) from the table below.

Interfaces	To Customer	From Customer	To Hardware	From Hardware
	CupOfCoffee	Coffee	FillCoffee	CoffeeOK
Signals	CupOfWater	Tea	HeatWater	Warm
	ReturnChange	Coin (Value : Integer)	-	-

5. Signal *Coin* will carry one signal data of type integer. This signal data will hold the value of the inserted coin(s). Add the parameter *Value* of type *Integer* in the middle compartment of the symbol representing signal *Coin*.
6. Save your work.

Note A colouring of an entity, for example on a signal name or a type, indicates that Tau/Modeler has found the matching definition. This is called name resolution and appears on all levels when designing in Tau/Modeler. Integer is one of the predefined types in UML. If no definition is present the name will be underlined with red.

9.3 Defining interfaces

You will now create interfaces containing the signals for the interaction with the environment. Do the following:

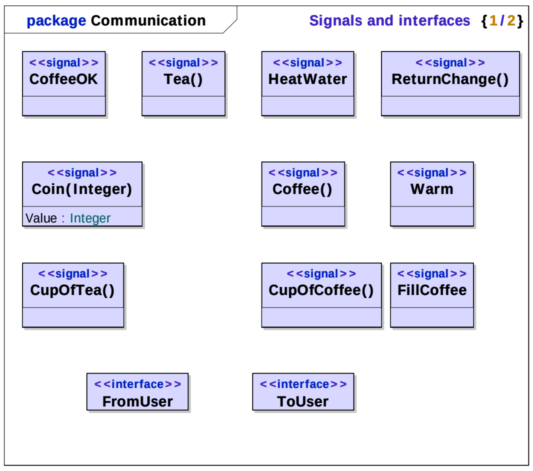
1. Open the class diagram *Signals and Interfaces*.
2. From the toolbar, use the *Interface symbol* and define the following two interfaces:
    a. FromUser
    b. ToUser
3. In the Model View drag the signals to the interfaces
    a. Signals *Coin*, *Coffee* and *Tea* should go to the interface *FromUser*.
    b. Signals *ReturnChange*, *CupOfCoffee* and *CupOfWater* should go to the interface *ToUser*.
4. Save your work.

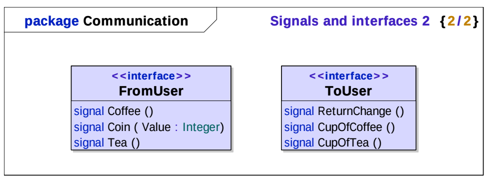
Note To view the signals in the interface symbols you can right-click and on the shortcut menu point to Show all operations. This will also let you modify signal parameters.

9.4 Ports

All signals going to or from an instance of a class pass through a port. Separate ports can be used for each entity with which the part communicates. You will now add ports to classes in your model to enable communication, starting with ports for external communication. Do the following:

1. Go to the diagram *DomainModel*. Select class *CoffeeMachine*. Hold down SHIFT and in the toolbar select the *Port symbol*. A port appears on the border. This port represents the communication to and from the class. Name the port *P1*.
2. Make sure that the port P1 is selected.
    a. Add a realized interface to P1 by clicking the toolbar button. Name the interface *FromUser*.
    b. Add a required interface name *ToUser*.
3. Select the class Controller, add a port "P2". Add required and realized interfaces to port P2 in the same way as for P1. Observe that the signals must go in the same direction.





10 Sequence Diagrams

10.1 General

A sequence diagram describes the behaviour of use cases. A sequence diagram is commonly built up with instances of active classes in your system and messages (signal instances) between them.

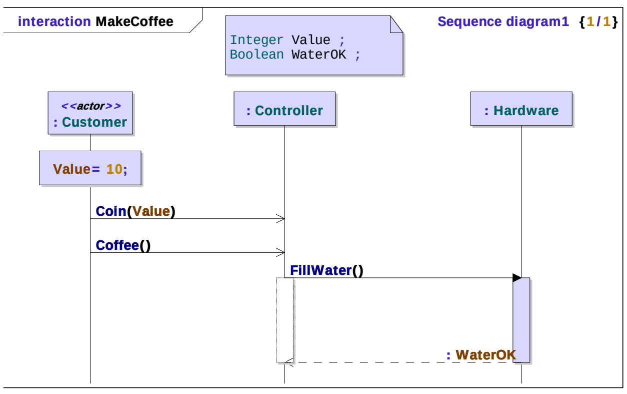
10.2 Creating sequence diagrams

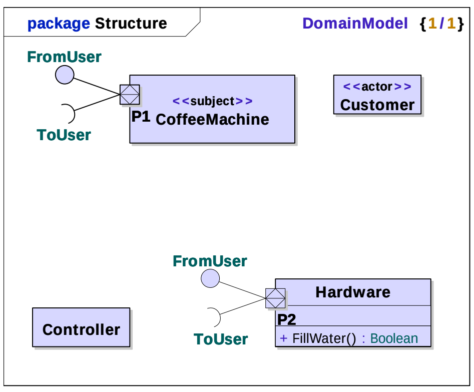
You will now create a sequence diagram for the use case *MakeCoffee*. Make sure that the Workspace window displays the Model View.

1. Locate the package *Services* in the Model View. Right-click the use case *MakeCoffee*, from the shortcut menu point to *New* and click *Sequence diagram*.
2. The diagram is now open and an icon for the sequence diagram appears inside in the Model View. The sequence diagram is contained in an *Interaction* which can contain one or more diagrams describing the use case.
3. Locate the class *Customer* in the Model View and drag it to the sequence diagram. The class appears as a lifeline.
4. Drag the class *Controller* to the sequence diagram.
5. Drag the class *Hardware* to the sequence diagram.

In the following instructions you will create a synchronous and an asynchronous messages in the sequence diagram, the *Message line* and *Method Call* buttons in the element toolbar are the starting points.

6. Draw a message from Customer to Controller. Drag the signal **Coin** onto the message line. The signal name appears with parameter list containing the types of the parameters, in this case one integer parameter. Replace the parameter type information with the integer value 10.
7. Add on Hardware class, the method FillWater as shown in the figure. From the element toolbar select the *Method Call* button. Click on *Controller* and then on *Hardware*. Drag the method *FillWater* on the *Method Call*.
8. Add Action Symbol and Text Symbol to complete the diagram as depicted in figure bellow.



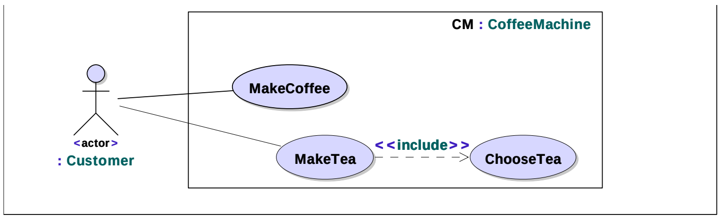


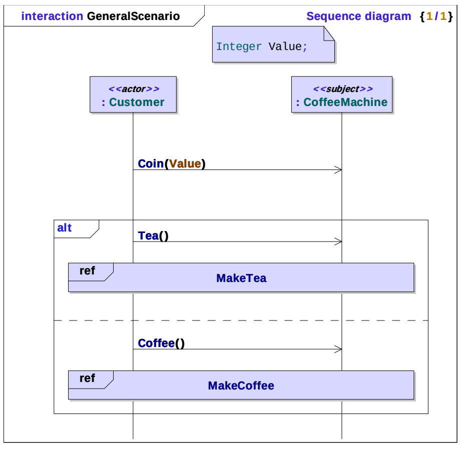
10.3 Sequence diagrams with references and alt frames

1. Add the use case MakeTea in the usecase diagram. Add an include use case ChooseTea with a *include* line of MakeTea use case.
2. Create a new sequence diagram named GeneralScenario.

In this sequence diagram, 2 references and 1 alternative are used.

3. From the toolbar, select the *Inline Frame Symbol* and click in the desktop.
4. Write *alt* in the left top of the frame. To add alternatives, use the *Separator Line* (the black square) on the right of the frame, a new compartment appears in the frame.
5. From the toolbar, select the *Reference Symbol* and click in the desktop.
6. Drag from the model view the use case *MakeTea* and drop in the appropriated Reference Symbol.
7. Repeat this operation for the *MakeCoffee* use case.





To add some *loop* frames, the steps are similar to the *alt* frames (see Lecture for example!).

11 State Machine Diagrams

11.1 General

A state machine diagram describes the behaviour of an active class. A state machine has one or more possible states and a change of state is triggered by a signal reception. In UML, the state machine concept has been extended with data handling, meaning that signal data and other variables can be declared and handled. Two different notations are supported: **transition-oriented syntax** and **state-oriented syntax**. Both syntaxes can be used for describing the behaviour of a state machine, but the state-oriented syntax is more suitable for getting an overview of a large design. The transition-oriented syntax is suitable for detailed design.

11.2 State machine diagram for class Hardware

11.2.1 Transition-oriented syntax

When using transition-oriented syntax, have in mind the following:

- To add a symbol, click the corresponding quick-button in the toolbar, then click the desktop.
- To connect two symbols, select the first symbol. Two handles appear below the symbol. Grab the handle represented by a square and drag a line to the second symbol. Click the symbol.
- Autoflow: Select a symbol in the flow and hold down SHIFT, then click on one of the symbols in the toolbar. The new symbol is automatically connected to the selected symbol.

11.2.2 Composite state

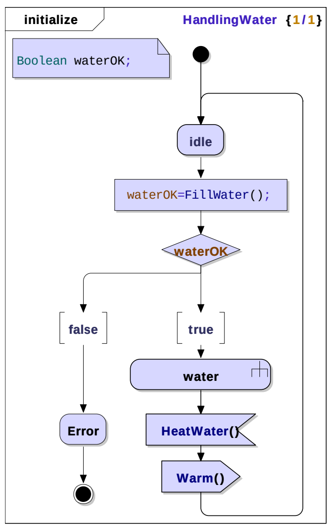
It is possible to have composite states in a UML state machine. This is a way of defining a sub-state with a state machine of its own. A composite state can be used for example when a set of actions tend to al- ways lead back to one identifiable state. This can be identified in the state machine for **Hardware**. When producing tea or coffee the signal exchange between Controller and Hardware is very similar but there is one signal exchange more when the user requires a cup of coffee. This signal exchange can then be put in a composite state.

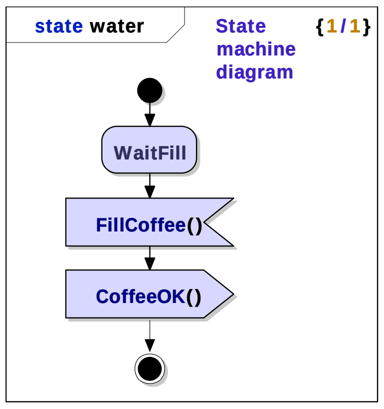
You will now describe the behaviour of class Hardware in a state machine. Do the following:

1. Add a state machine to class Hardware. Right-click and from the shortcut menu point to **New** and then click **State machine diagram**. Name the state machine “HandlingWater”. The state machine is contained in a **state machine implementation** in turn contained in a state machine, by default named to **initialize**.
2. Implement the behaviour as depicted in the figure

You will now use the transition-oriented syntax to describe the behaviour in the composite state.

1. Open the state machine “HandlingWater” in class Hardware.
2. Double-click on state Water. The Create Presentation dialog opens. Go to tab New diagram, point to State machine diagram. A new diagram opens in the desktop.
3. Draw the sub-state definition as depicted below.

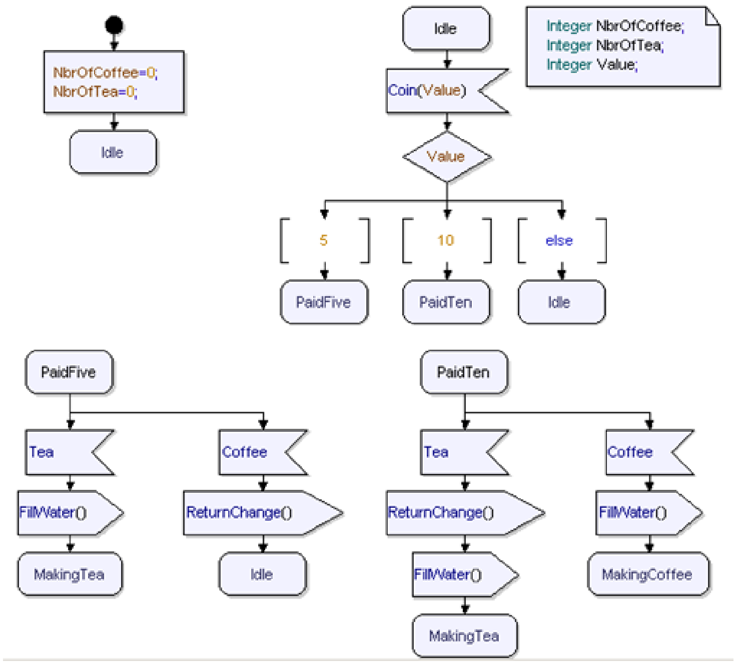




11.3 State machine diagram for class controller

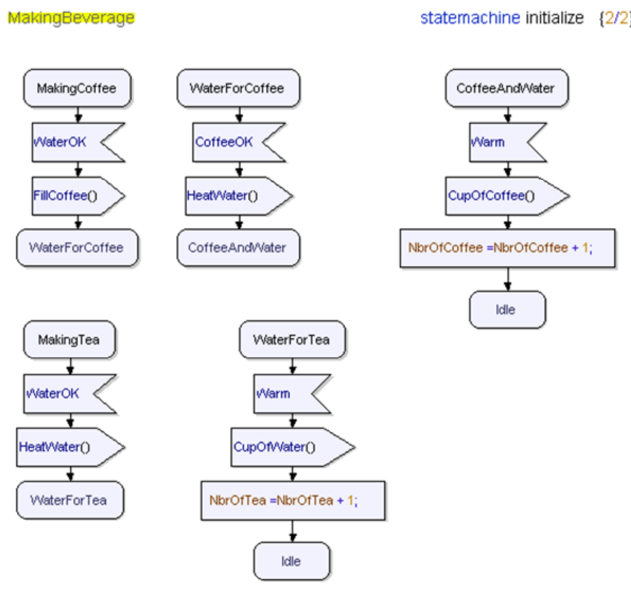
You will now describe the behaviour of class Controller in a state machine, using the transition-oriented syntax. To add a state machine, do the following:

1. In the Model View of the Workspace window, select the icon representing class Controller and create a new state machine diagram for Controller.
2. Rename the state machine diagram to "NewOrder" to indicate that this diagram will describe the behaviour when a new coffee or tea order is received.
3. Implement the behaviour as depicted in the following figure.



The state machine describes the reception of a coin, the reception of a coffee or tea order and the start of the hardware communication. Two variables are used for counting the number of cups of coffee or tea that have been served. Another variable is needed for holding the value of the coin.

4. To get more drawing space, you can now extend the state machine for the rest of the behaviour. In the Model View, select the icon representing the **state machine implementation** and create another state machine page for Controller.
5. Name the new state machine page "MakingBeverage".
6. Implement the rest of the behaviour according to figure.



12 Composite Structure Diagrams

12.1 General

The next step is to add a composite structure diagram. A composite structure diagram displays instances of active classes, and the communication between them. This is the diagram for showing how your objects from your model should be instantiated and built together to form a system. The instances are called **parts**. A part communicates with other parts or with the environment through **ports**. Ports are connected through **interfaces** or **connectors**.

12.2 Creating a composite structure diagram

You will now instantiate your classes and describe the communication be- tween the customer, the controller and the hardware in a composite structure diagram. To add a composite structure diagram, do the following:

1. In the Model View, select the icon representing class CoffeeMachine.
2. Right-click on class CoffeeMachine and select **New** and then **Composite structure diagram**.
3. The composite structure diagram appears in the Model View. Name the diagram “Communication”.

12.2.1 Parts

A part is an instantiation of an active class.

Add the parts by doing the following:

1. Make sure that the diagram Communication is open on your desktop. From the toolbar, select the symbol representing a **part** and place it in the diagram.
2. Click inside the symbol to activate the text area. Name the instance **Ctrl:Controller**, where Ctrl is the name of the part and Controller is the name of the class it instantiates.
3. Add a part representing the instantiation of class Hardware. Name the part **Hw:Hardware**.

Note It is also possible to drag-and-drop the active classes to the composite structure diagram.

12.2.2 Ports

You will now add some more ports to your model to enable communication, starting with the port representing the customer. Do the following:

1. Select the part Ctrl. Hold down SHIFT and in the toolbar, select the **Port symbol**. A port appears on the border. Name the port “P3”. This port rep- resent the communication with the Hw part.
2. Select the Hw part. Add one port and name it “P4”. This port represents the communication with the Ctrl part.
3. Add required and realized signals for the ports P3 and P4. Observe that the signals in the ports must correspond to the direction of the connection line. The following signals can be received by Ctrl: – CoffeeOK, WaterOK, Warm

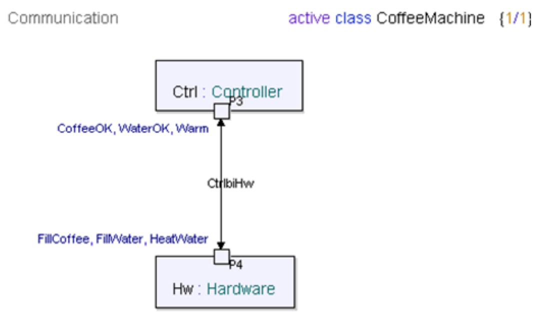
The following signals can be received by Hw: – HeatWater, FillWater, FillCoffee

12.2.3 Connectors

A connector is a signal path which can be bidirectional or unidirectional. Connectors connect the ports in the composite structure diagram. You will now add a connector to your model.

Do the following:

1. Add a connector between ports P3 and P4. This is done by selecting one of the ports, then dragging the handle to the other. Three text fields ap- pear. Name the connector “CtrlbiHw“.
2. Right-click on the connector line and select **Show all signals** from the shortcut menu. For each of the connectors the signals corresponding to the Realizes and Requires properties will be filled in. Your composite structure diagram should now look as depicted in figure.

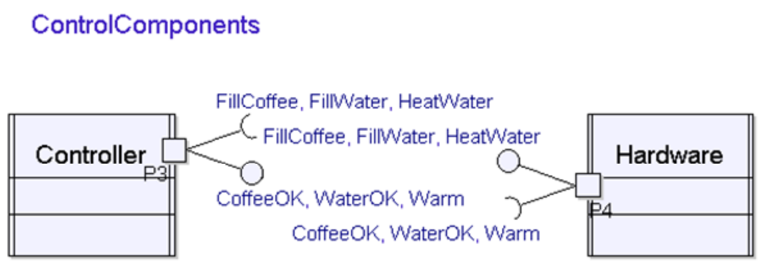


Your composite structure diagram is now complete. Ports and parts have been added to the model as you have edited the diagram.

12.2.4 Ports and interfaces

You will now add your ports to the classes in the component diagram.

1. Open your component diagram (ControlComponents).
2. Drag and drop port P3 to Controller.
3. Drag and drop port P4 to Hardware. The interfaces and signal lists will be shown as in figure


ControlComponents
FillCoffee, FillWater, HeatWater
FillCoffee, FillWater, HeatWater
Controller
P3
CoffeeOK, WaterOK, Warm
CoffeeOK, WaterOK, Warm
Hardware
P4