Rational. Tau and DOORS Analyst

UML Tutorial

IBM Rational Modeler Tau Edition 4.3

UML Tutorial

This edition applies to IBM® Rational® Tau® version 4.3 and to all subsequent releases and modifications until otherwise indicated in new editions.

Notices

This information was developed for products and services offered in the U.S.A. IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send written license inquiries to the following:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte character set (DBCS) information, contact the IBM Intellectual Property Department in your country or send written inquiries to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106-0032, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions. Therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

Intellectual Property Dept. for Rational Software|
IBM Corporation
1 Rogers Street
Cambridge, Massachusetts 02142
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

Copyright license

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

© (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs. © Copyright IBM Corp.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Trademarks and service marks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at www.ibm.com/legal/copytrade.html.

Third-party Trademarks

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Java and all Java-based trademarks and logos are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Other company, product or service names may be trademarks or service marks of others.

Contacting Customer Support

If the self-help resources have not provided a resolution to your problem, you can contact IBM® Rational® Software Support for assistance in resolving product issues.

Before you contact Customer Support

To submit your problem to IBM Rational Software Support, you must have an active Passport Advantage® software maintenance agreement. Passport Advantage is the IBM comprehensive software licensing and software maintenance (product upgrades and technical support) offering. You can enroll online in Passport Advantage from lhttp://www.ibm.com/software/lotus/passportadvantage/howtoenroll.htm

To learn more about Passport Advantage, visit the Passport Advantage FAQs at http://www.ibm.com/software/lotus/passportadvantage/brochures_faqs_quickguides.html.

For further assistance, contact your IBM representative.

To submit your problem online (from the IBM Web site) to IBM Rational Software Support, you must additionally:

- Be a registered user on the IBM Rational Software Support Web site. For details about registering, go to http://www.ibm.com/software/support/.
- Be listed as an authorized caller in the service request tool.

Submitting Problems

To submit your problem to IBM Rational Software Support:

1. Determine the business impact of your problem. When you report a problem to IBM, you are asked to supply a severity level. Therefore, you need to understand and assess the business impact of the problem that you are reporting.

Use the following table to determine the severity level

Severity	Description
1	The problem has a *critical* business impact: You are unable to use the program, resulting in a critical impact on operations. This condition requires an immediate solution.
2	This problem has a *significant* business impact: The program is usable, but it is severely limited.
3	The problem has *some* business impact: The program is usable, but less significant features (not critical to operations) are unavailable.
4	The problem has *minimal* business impact: The problem causes little impact on operations or a reasonable circumvention to the problem was implemented

2. Describe your problem and gather background information, When describing a problem to IBM, be as specific as possible. Include all relevant background information so that IBM Rational Software Support specialists can help you solve the problem efficiently. To save time, know the answers to these questions:
   - What software versions were you running when the problem occurred?

   To determine the exact product name and version, use the option applicable to you:
   - Start the IBM Installation Manager and select **File** > **View Installed Packages**. Expand a package group and select a package to see the package name and version number.
   - Start your product, and click **Help** > **About** to see the offering name and version number.
   - What is your operating system and version number (including any service packs or patches)?
   - Do you have logs, traces, and messages that are related to the problem symptoms?
   - Can you recreate the problem? If so, what steps do you perform to recreate the problem?
   - Did you make any changes to the system? For example, did you make changes to the hardware, operating system, networking software, or other system components?
   - Are you currently using a workaround for the problem? If so, be prepared to describe the workaround when you report the problem.

3. Submit your problem to IBM Rational Software Support. You can submit your problem to IBM Rational Software Support in the following ways:
   - **Online**: Go to the IBM Rational Software Support Web site at https://www.ibm.com/software/rational/support/ and in the Rational support task navigator, click **Open Service Request.** Select the electronic problem reporting tool, and open a Problem Management Record (PMR), describing the problem accurately in your own words.
   - For more information about opening a service request, go to http://www.ibm.com/software/support/help.html
   - You can also open an online service request using the IBM Support Assistant. For more information, go to http://www.ibm.com/software/support/isa/faq.html.
   - **By phone**: For the phone number to call in your country or region, go to the IBM directory of worldwide contacts at http://www.ibm.com/planetwide/ and click the name of your country or geographic region.
   - **Through your IBM Representative**: If you cannot access IBM Rational Software Support online or by phone, contact your IBM Representative. If necessary, your IBM Representative can open a service request for you. You can find complete contact information for each country at http://www.ibm.com/planetwide/.

If the problem you submit is for a software defect or for missing or inaccurate documentation, IBM Rational Software Support creates an Authorized Program Analysis Report (APAR). The APAR describes the problem in detail. Whenever possible, IBM Rational Software Support provides a workaround that you can implement until the APAR is resolved and a fix is delivered. IBM publishes resolved APARs on the IBM Rational Software Support Web site daily, so that other users who experience the same problem can benefit from the same resolution.

Table of Contents

Introduction

Purpose of this tutorial

The purpose of this tutorial is to make you familiar with Tau/Modeler and the UML language. This tutorial primarily addresses persons with no or little experience of Tau/Modeler, but with knowledge of the basic concepts of UML and state machines.

This tutorial provides step-by-step instructions on how to start up a workspace and create a project from scratch. You will produce a UML specification.

The instructions in this tutorial should be complete to let you perform all steps. More information on the various work procedures can be found in the Tau/Modeler On-line help.

In the margin of the textual description you will sometimes find pictures of toolbar buttons. The purpose of this is to make it easier for you to identify the button that are referred to in the text. The button pictures will in most cases only be present the first time you are instructed to use a button. These buttons belongs to various toolbars which may be activated or deactivated by the user, therefore it can happen that a button referred to is not visible in the current set of toolbars. The toolbar must then first be activated. This can be done in the Customize dialog which is opened from the Tools menu.

When you are instructed to insert symbols in diagrams make sure that the diagram is active by clicking in it. In a recently opened diagram it is not always possible to directly click on a toolbar to insert symbols, first click anywhere in the diagram then select the desired symbol from the toolbar.

The Coffee Machine

Behavior 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.

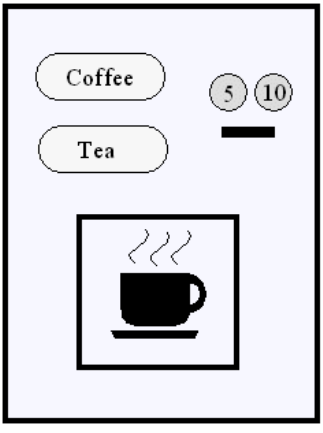

Figure 1: The coffee machine

Model and Diagrams

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.

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**.

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.

User Interface

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.

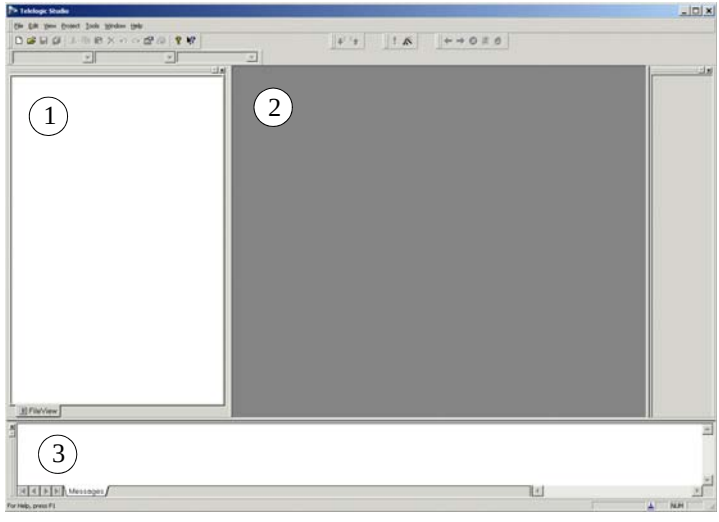

Figure 2: The user interface

The 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.

The 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.

The 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.

The Output window

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

The Shortcut menu

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

Workspace

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.

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 workspace 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.

Projects

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.

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.

Note

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.

6. Expand 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**.

Use Case Diagrams

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.

Creating use case diagrams

You will now create a use case diagram for the use cases for the coffee machine example. These cases were described earlier, see "Behavior of the coffee machine" on page 2.

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** and then click **Collaboration**. Name the collaboration **UserDrinks**.

Note

*Renaming of model entities is done by selecting the element and pressing F2, **or** by selecting the element and clicking once on the name. This will open the name string for editing.*

You will use a collaboration in the model to encapsulate the use case diagram, and also later use cases represented as sequence diagrams.

3. Select the collaboration. Right-click and from the shortcut menu point to **New** and then click **Use case diagram** to insert a use case diagram.

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.

4. 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 (See Figure 3 on page 9) indicating a name binding error, secondly there will be some error messages in the Output window (Autocheck tab).

5. 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. See Figure 3 on page 9.

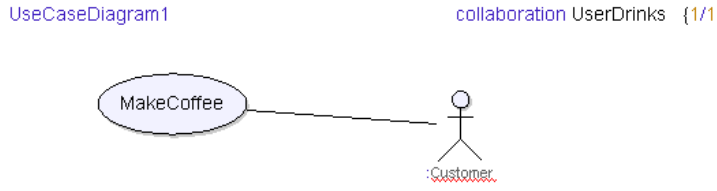

Figure 3: Use case and actor

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.

Class Diagrams

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.

Creating class diagrams

You will now use class diagrams for modeling 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** 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.  Use the **Class symbol** quick-button to add two classes, click on the button, then click in the diagram to position the class. Name the classes **CoffeeMachine** and **Customer**. These classes represents the coffee machine and a prospective user, see Figure 4 on page 10.

Figure 4: The classes CoffeeMachine and Customer

Note

When class customer is created any errors concerning the actor type in the autocheck tab are resolved.

Active or passive classes

A class can be either active or passive. An active class contains behavior, while a passive class merely contains definitions. An example of a passive class is a data type. Your class CoffeeMachine will contain behavior, and must therefore be set to active. Do the following:

1. Right-click the CoffeeMachine class symbol.
2. On the shortcut menu that appears, click **Properties**. The properties dialog opens and displays the properties for class CoffeeMachine. In **Filter** select Class.
3. Select the **Active** check box. You may leave the dialog open.

Note

The properties dialog can be opened by pressing ALT + Enter.

An active class is displayed in the diagram with double vertical border lines, see Figure 5 on page 11.

Figure 5: Active and external classes

Informal class

The class Customer is an entity which is not a part of the coffee machine design itself. You will give it the stereotype **informal**. Do the following:

4. Right-click class Customer. On the shortcut menu that appears, click **Stereotypes...** .
5. Select the **TTDPredefinedStereotypes:informal** check box. The stereotype is activated as soon as the dialog is closed.
6. Right-click the class symbol for Customer and select **Active**.

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 behavior.

When adding these classes, different methods will be used. You will add the classes directly to the model and then use some modeling 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** and click **Class**. A class icon appears in the Model View.
5. Name the class **Hardware**.
6. Select the class Hardware in the Model View. Right-click class Hardware. On the shortcut menu, change the properties to make Hardware to an active class. Do the same for class Controller.

Note

Observe that you do not need to go to the properties dialog to set a class to active. The shortcut menu contains some context-sensitive choices and one of these is controlling whether a class is active or not.

Component diagram

Component diagrams are closely related to and can be considered a subtype of class diagrams. Component diagrams describe the identifiable and possibly replaceable components of a system. Just like a class diagram the component diagram can also show ports and interfaces of the classes.

Creating component diagrams

You will now use a component diagram for further modeling of the coffee machine example.

1. Make sure that the Workspace window displays the **Model View**.
2. Right-click class **Controller**. On the shortcut menu, point to **Create Presentation**. In the New Symbol tab of the dialog click on the table row with **New ComponentDiagram**.
3. An icon for the diagram appears in the **Model View** and the diagram is opens on the Desktop. Name the diagram "**ControlComponents**".
4. Drag and drop the Hardware class icon from the Model View into the open component diagram (**ControlComponents**) on the Desktop.
5. Save your work.

Figure 6: Components for CoffeeMachine

Your component diagram should now look as depicted in Figure 6 on page 13.

Signals

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.

Defining signals

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

1. Select your package **CMdesign** in the Model View. Add a class diagram by right-clicking and from the shortcut menu point to **New** and then click **Class diagram**. The diagram appears in the Model View of the Workspace window.
2. Name the diagram “Signals”. The diagram is now open on the Desktop.
3.  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.

Note

Press and hold CTRL when placing a symbol is a shortcut to keep the selection in the toolbar. This makes it possible to position multiple symbols of the same sort without going back and forth between the diagram and the toolbar.

To Customer	From Customer	To Hardware	From Hardware
CupOfCoffee	Coffee	FillWater	WaterOK
CupOfWater	Tea	FillCoffee	CoffeeOK
ReturnChange	Coin (Integer)	HeatWater	Warm

4. Signal **Coin** will carry one signal data of type integer. This signal data will hold the value of the inserted coin(s). Add the type **Integer** in the middle compartment of the symbol representing signal **Coin**, see Figure 7 on page 15.
5. Save your work.

Note

A coloring 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.

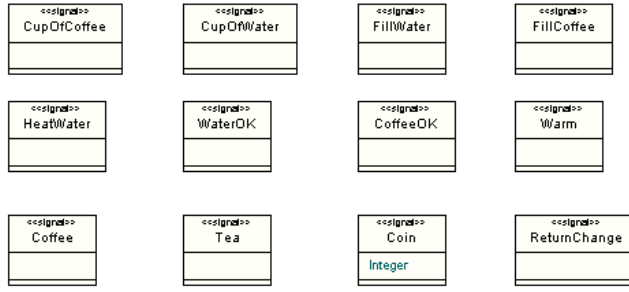

Figure 7: Signal definitions

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

Note

Each signal will be a feature of the interface and the text line will be treated as a presentation element. In addition to being editable like text it will be possible to delete the entire line (presentation element) if you position the cursor first on the line and press backspace.

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.

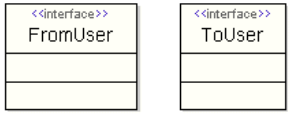

Figure 8: Interface definitions

Interface relations

The interfaces are now ready to be used. You will now create an **Association** between the two interfaces and a **Dependency** from ToUser to Customer.

1. Select **ToUser** interface symbol, drag the Association line handle from **ToUser** to **FromUser**. This will implicate that wherever one of the interfaces is used, the other will also be available.
2. Remove the navigability of the association. Right-click on the line close to its connection to the interface, select Target and in the sub-menu deselect **Navigable**. Right-click on the line close to the connection to the other interface, select Source and in the sub-menu deselect **Navigable**.
3. Go to the class diagram DomainModel. You can navigate with the **Previous** button on the Navigation toolbar or select the diagram tab in the top of the desktop area.
4. Drag the interface **ToUser** from the Model View into the class diagram DomainModel. Select Customer class symbol, drag the Dependency line handle to ToUser interface symbol. This will indicate that Customer has a dependency ToUser.

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. Add a realized interface to P1 by clicking the toolbar button. Name the interface **FromUser**.
3. With P1 selected press ALT + Enter to open the properties dialog. Make sure that Port is selected in Filter field. Note that the interface name is inserted in the **Realizes** field. Add the interface name **ToUser** in the **Requires** field.

4. Go to the diagram DomainModel again. Select the port P1 and click the tool bar button for the **Required** interface. Observe that the name is automatically inserted on the interface symbol.

Note

It is not necessary to add the required interface, because of the association between the interfaces. This is only done to improve readability.

5. Drag and drop the Controller class icon from the Model View into the open class diagram (DomainModel) on the Desktop.
6. Drag and drop the Hardware class icon from the Model View into the open class diagram (DomainModel) on the Desktop.
7. 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.
8. Save your work

Your diagrams should now contain ports and interfaces as shown in Figure 9 on page 18. Note that the diagrams are rearranged with respect to earlier pictures and that only a subset of the symbols are shown.

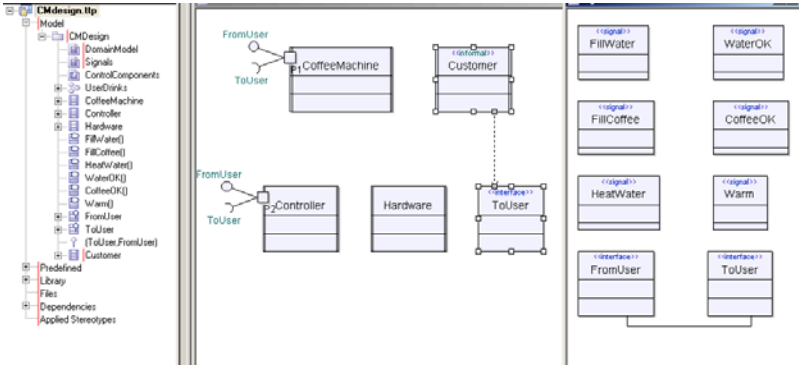

Figure 9: Interfaces and ports

Sequence Diagrams

General

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

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 collaboration **UserDrinks** in the Model View. Right-click the use case **MakeCoffee**, from the shortcut menu point to **New** and click **Sequence diagram**.

Note

The actor type is now bound to class Customer. In the use case diagram the class name is no longer underlined with red, but is now colored green. This is an example of the work done in the background analyzing your model changes and updating your presentation elements to keep your model consistent at all times.

2. The diagram is now open and an icon for the sequence diagram appears inside the collaboration 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 use three different ways of creating messages in the sequence diagram. For each of these note especially the binding of signal names from your model. The **Message line** button in the element toolbar is the common starting point for all three ways.

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. From the element toolbar select the **Message line** button. Click on Customer and then right-click when placing the receive event on Controller. From the shortcut menu point to **Reference existing** and click **Coffee**.

Note

Reference existing can be used for classes with ports and interfaces that defines a set realized of signals.

8. Draw a message from Controller to Hardware. Type the signal name **FillWater** onto the message line.

Note

CTRL + Space is a shortcut command that performs name completion. If you start typing a name, this will complete the name or present a list with entities allowed in the scope.

9. Continue to draw the sequence depicted in Figure 10 on page 20
10. Save your work.

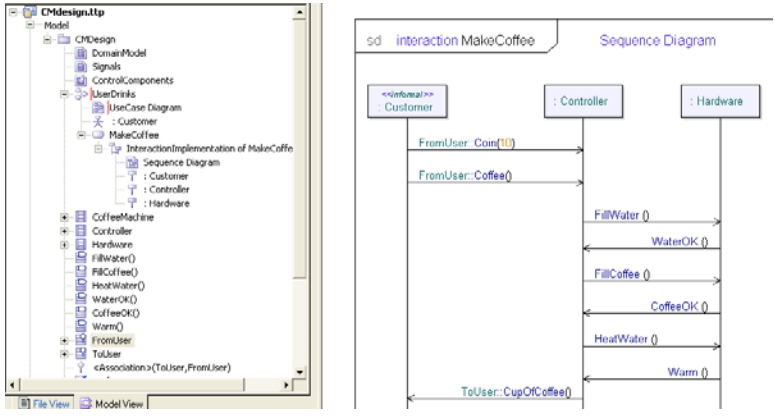

Figure 10: Sequence diagram for the use case MakeCoffee

Use cases and subject frame

1. Open the use case diagram in the collaboration **UserDrinks**. Create a new use case and name it **MakeTea**.
2. Create a new use case and name it **TeaMaking**.
3. Draw a dependency line from MakeTea to TeaMaking. Create an association line to the actor from MakeTea.

MakeTea should be one of the main use cases describing a complete scenario for the system to produce a cup of hot water. The use case TeaMaking is a sub-scenario describing filling of the cup and heating the water.

4. Create a **subject symbol** around the use cases. After selecting the symbol in the toolbar, drag to enclose the use case symbols with the subject symbol. Name the frame and identify the frame to belong to class CoffeeMachine. See Figure 11 on page 21.
5. Save your work.

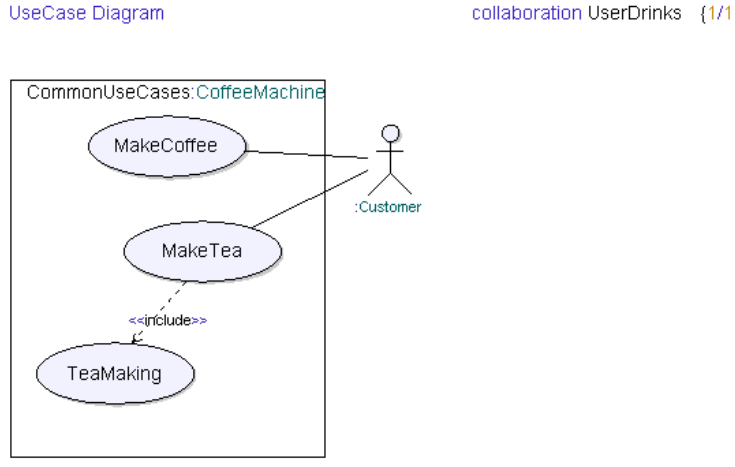

Figure 11: Use cases in a subject frame

The subject frame is optional but it encapsulates your use cases to belong to one particular class in your model allowing you to draw complex use case diagrams with common actors to one or more systems.

Sequence diagrams with references

1. Locate the collaboration **UserDrinks** in the Model View. Right-click the use case **MakeTea** and from the shortcut menu point to **New** and click **Sequence diagram**.

   The diagram is now open and an icon for the sequence diagram appears inside the collaboration in the Model View.

2. Locate the class **Customer** in the Model View and drag it to the sequence diagram. The class appears as a lifeline. Drag the class **CoffeeMachine** to the sequence diagram.
3. Draw a message from Customer to CoffeeMachine. Type the signal name **Coin** with parameter **5** onto the message line. Draw a new message from Customer to CoffeeMachine. Type the signal name **Tea** onto the message line.
4. Place a **reference** symbol below the signals. Name the reference symbol **TeaMaking**. See Figure 12 on page 22.
5. Save your work.

Figure 12: Sequence diagram with reference

You can use reference symbols as build blocks to create longer and more complex sequence diagrams from a common set of simple sequences.

State Machine Diagrams

General

A state machine diagram describes the behavior 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 behavior 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.

State machine diagram for class Hardware

State-oriented syntax

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

- When connecting states, select the first state. Two handles appear. Grab the handle represented by a filled circle. Drag the line to the second state and click the symbol.
- To change the shape of the connecting line, click it. Square-shaped handles appear. Drag the handles to form the requested shape.
- When drawing a line from a state back to the state itself, grab the handle represented by a filled circle and drag the line away from the state. Click to change directions, drag the line back to the state and click the symbol.
- The following notation is used on transition lines:
  <input>/ <transition statement>;
  where <transition statement> can be for example: ^ **<signal name>**. Signals without parameters should be followed by an empty set of parenthesis. There can be multiple statements in a transition, they are then separated by semicolon.

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 always 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.

Starting with the state-oriented syntax, you will now describe the behavior 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 behavior as depicted in Figure 13 on page 24.
3. Save your work.

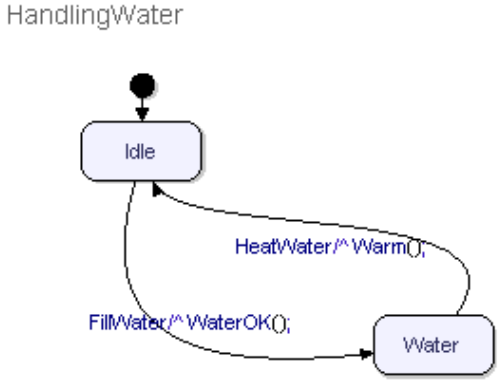

Figure 13: State machine diagram HandlingWater

Note

Name resolution, visualized as coloring of entities, appears whenever the matching definition of an entity is found. If the definition is included in the model, navigation is possible. To navigate to the definition, hold down CTRL and click the name.

UML is case-sensitive. In order for name resolution to be performed the case must match.

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.

You will now use the transition-oriented syntax to describe the behavior 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 in Figure 14 on page 26. The history state implies a return to the state WaitFill.
4. Save your work.

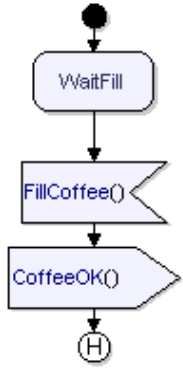

Figure 14: Composite state

State machine diagram for class controller

You will now describe the behavior 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 behavior when a new coffee or tea order is received.
3. Implement the behavior as depicted in Figure 15 on page 27.

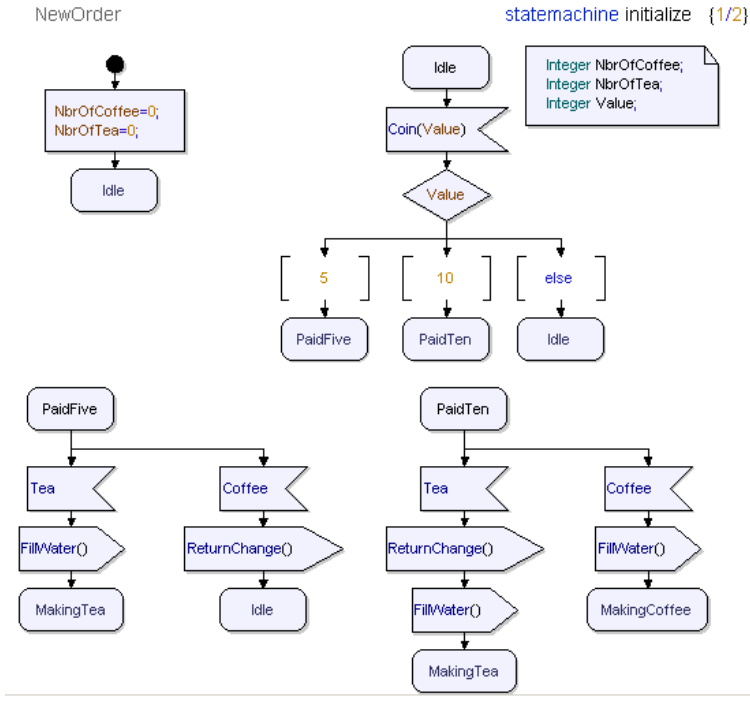

Figure 15: State machine diagram NewOrder

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 behavior. 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 behavior according to Figure 16 on page 28.
7. Save your work.

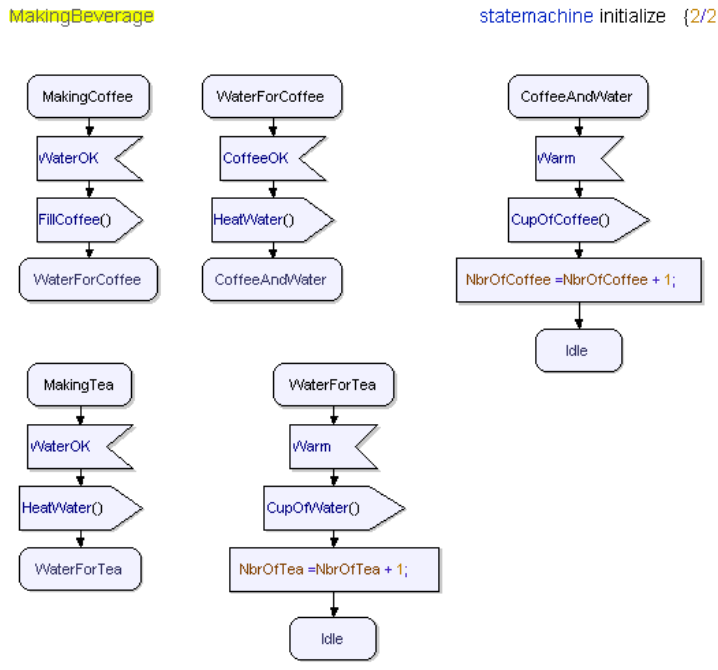

Figure 16: State machine diagram MakingBeverage

The state machine diagrams are now complete.

Composite Structure Diagrams

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**.

Creating a composite structure diagram

You will now instantiate your classes and describe the communication between 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”.

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.

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 represent 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

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 appear. 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 17 on page 31.

3. Save your work.

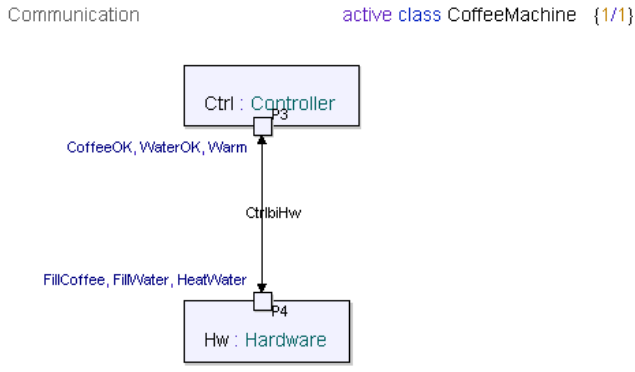

Figure 17: The composite structure diagram

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

Ports and interfaces

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

1. Open your component diagram (ControlComponents).

Figure 18: The component diagram with ports and interfaces

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 18 on page 31.

Relations

General

The next step is to add relations between your classes. Relations between classes in a model are best illustrated in the class diagram.

Associations

An **association** represents a relationship between objects. Each association has two roles, represented by the directions of the associations. You have earlier created an association between the interfaces ToUser and FromUser in the model.

Compositions

Controller and Hardware “live and die” with class CoffeeMachine, which means that their relation to class CoffeeMachine is a **composition**.

1. In the diagram DomainModel, select the class CoffeeMachine. Three handles appear below the symbol.
2. Grab the handle represented by a square (Association/Aggregation/Composition line), drag the line to class Controller and right-click to the class symbol. Select **Reference Existing**. The drop-down box should contain the Ctrl part.

Note

It is possible to change a composition into an association or an aggregation from the shortcut menu, by right-clicking on the line close to the connection to CoffeeMachine.

3. Add a **composition** in the same way between class CoffeeMachine and class Hardware.
4. Save your work.

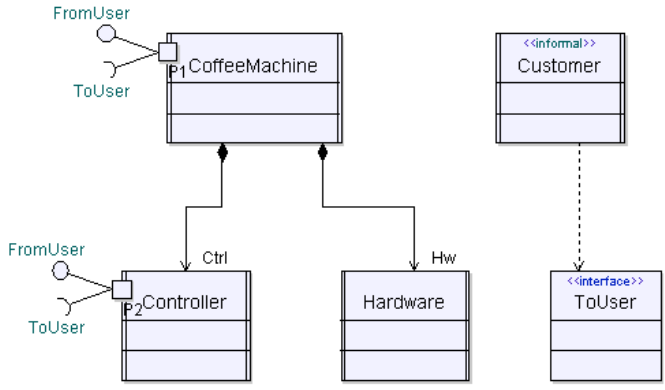

Figure 19: Compositions in the domain model

Your class diagram should now look as depicted in Figure 19 on page 33.

Iterations and additions

Purpose

This next section cover some additions to the model, showing some more features of UML. You will also get a repetition of some of the editors that you have worked with along the tutorial.

Some of the descriptions of what to do will in this section be shorter and not as detailed as before. This will especially be so when it concerns activities similar to those earlier described.

Timer

Introduce a timer into your coffee machine. The timer should expire after 10 time units and it is to be set in conjunction with the activity of heating the water.

1. Go to class Hardware in the Model View. Add a timer called **Heater** to the class. Right-click class Hardware and from the shortcut menu point to **New** and click **Timer**.
2. Go to the state machine for Hardware. The timer signal should be received in such a way that it triggers the sending of signal Warm. Replace the signal **HeatWater** with **Heater** in the transition from state Water to state Idle.

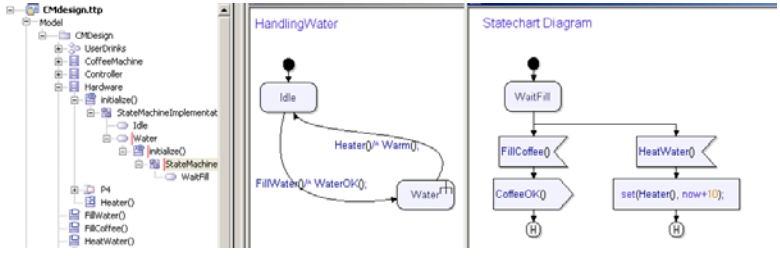

Figure 20: Timer declaration and use

3. Go to the state diagram for composite state Water. Add the signal HeatWater in parallel to the signal receipt of FillCoffee. This transition should contain the **set** statement of timer Heater, "`set(Heater(),now+10);`". See Figure 20 on page 34.

4. Save your work.

Structured data

Introduce a class representing a set of structured data into your coffee machine. The contents of this data model will be milk and sugar to be put into the coffee. This will require several steps to be performed as it will affect receiving and sending of signals as well as some internal behavior.

1. Open the class diagram **Signals**. Add a class in the diagram and name the class **Additives**.
2. Add two attributes to the class, **Milk** of type **Boolean** and **Sugar** of type **Integer**. Declare the attributes to be public.

Hint

To set the visibility to public type a '+' sign in front of the attribute name in the interface (class) symbol.

Note

*Next you will add a parameter of type Additives to the signal **Coffee** and to the signal **CupOfCoffee**. To be able to edit parameters for signals in interfaces you must first make the signals visible in the interface symbol.*

3. Select the signal **Coffee** in the Model View, drag it to **FromUser** interface symbol in the class diagram. Type in the parameter and declare it as a part.
4. Select the signal **CupOfCoffee** in the Model View, drag it to **ToUser** interface symbol in the class diagram. Type in the parameter and declare it as a part. See Figure 21 on page 36.
5. Open the state diagram **NewOrder** for class Controller. Declare a variable called **Add** of type Additives. Declare the variable as a part.

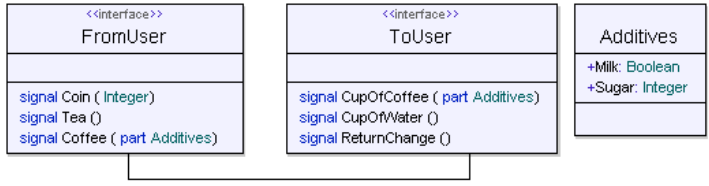

Figure 21: Class Additives

6. Use the variable Add to receive request for milk and sugar whenever the signal coffee is received.
7. Use the value of the variable Add as parameter when sending CupOfCoffee to the environment. A possible solution is shown in Figure 22 on page 37
8. Save your work.

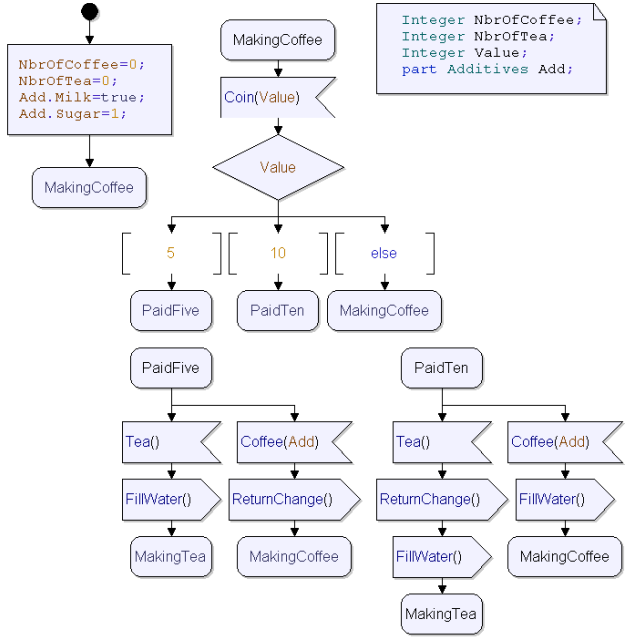

Figure 22: State machine for Controller with Additives

Conclusions

Model

The diagrams in UML are views of your model. Whenever possible the model will work for you, creating definitions and instantly binding these to elements as they are created. To your assistance you have the Model Navigator which allows you to browse the model bindings of your choice.

Editors

You should now feel familiar with working with the editors in Tau/Modeler. The example you have built is simple to its functionality but contains examples of a large part of the possible symbols in UML. The tutorial has in some situations shown different ways of drawing similar constructions. Which to use in a given situation depends on the characteristics of the problem but is also many times a personal preference. Important to understand is the underlying model support to speed up your work and make it more precise.

Workflow

The workflow in this tutorial is intended to be aligned with a possible workflow in a larger software project. The scope of the tutorial is focused on demonstrating the tool rather than imposing a methodology. The order of the activities described should however fit well with the design phase of a larger software project. It is in this case important to remember that the analysis is already done and even if you have edited the complete model from scratch all the design decisions was made in advance. You can find some more information on workflow methodology in the chapter Description of Workflow in the online help of Tau/Modeler.

What’s next?

You have now completed the tutorial and are ready to start working on your own with UML and Tau/Modeler. If you would like to have more information on model driven work it is recommended that you study the chapter “Working with Models” in the online help.