Chapter 1. Introduction to iUMLite 2.20

Through this lecture we will explore the executable UML formalism and how it is supported by iUML.

We will also develop a model based on a case study.

Case Study – A Simple Gas Station System

Mission Statement

This (simple) Gas Station System manages the dispensing of fuel, customer payments and tank levels.

A computer-based system is required to control the dispensing of fuel, to handle customer payment and to monitor tank levels. The system must be "best of breed", easy to use, reliable, fast and easy to modify to incorporate requirements yet to be conceived.

- **Enabling the Pump & Delivering Fuel**
  Before a customer can use the self-service pumps, the pump must be enabled by the attendant. When a pump is enabled, the pump motor is started, if it is not already on, with the pump clutch free. When the trigger in the gun is depressed, closing a micro switch, the clutch is engaged and fuel pumped. When it is released, the clutch is freed. There is also a micro switch on the holster in which the gun is kept which prevents fuel being pumped until the gun is taken out. Once the gun is replaced in the holster, the delivery is deemed to be completed and the pump disabled. Further depressions of the trigger in the gun cannot dispense more fuel. After a short stand-by period, the pump motor will be turned off unless the pump is re-enabled.

- **Measuring the Delivery**
  A metering device in the fuel line sends a pulse to the system for each hundredth of a litre dispensed. The cost of the fuel is calculated using the amount delivered and unit cost which is displayed on the pump.

- **Payment**
  Transactions are stored until the customer pays. Customers sometimes abscond without paying and the operator must annotate the transaction with any available information, the vehicle's registration number for example. At the end of the day, transactions are archived and may be used for ad-hoc enquiries on sales.

- **Fuel**
  At present, two grades of fuel are dispensed from three pumps on the forecourt. Each pump takes its supply from one of two tanks, one tank for each grade. The tank level must not drop below 4% of the tank's capacity. If this happens, the pumps serviced by that tank cannot be enabled to dispense fuel.

Chapter 2. Managing your Database

Basics of iUML database:

1. it is a multi-user database
2. can create as many databases as you want
3. each of these databases can be accessed by one or more users

This section describes how to create and edit databases.

1. Creating an iUML database –

a. Pull-down **File** > **New Repository ...** .
b. Navigate to the folder/directory that you wish to store your database in.
c. Enter the name of the database:
   Simple_Gas_Station
d. Click **OK**.
e. Enter the following fields in the Database Details dialog:

Pressing the **Tab** key will cycle through the text entry boxes.

In field	Type
Administrator	<Your name>
Database	Simple Gas Station System
Description	Gas Station case study for the iUML tutorial

f. Click **OK**.

You have now created a database in the directory you specified.

Acquiring the model for edit

Before you can edit the items in the database, you must acquire a lock. When you exit the database and re-enter you will need to acquire the model for edit each time.

To acquire the Top Level Lock for the database:

From the tree RMB pull-down Database name > **Acquire Lock** > **Top Level Lock**

Opening, Exiting and Saving Models

In order to work through the tutorial at your own pace, you will need to save your work, exit and re-enter iUML Modeller.

To Save and Exit and iUML Database

a. Pull-down **File** > **Exit**.
b. Click **Save**.
c. You can enter an comment to describe your additions to the database.
d. Click **Publish**.

Confirm your exit of iUML by clicking **OK**.

To Open an iUML Modeller Database

e. Start iUML Modeller
f. Pull-down **File** > **Open**.
g. Select the location of the iUML Database.
h. Click **OK**.

To Save an iUML Database

To save changes and continue to work through the tutorial:

a. Pull-down **File** > **Save**.
b. You can enter an comment to describe your additions to the database.
c. Click Publish.

NOTE: After saving you will need to acquire a lock to continue editing.

Chapter 3. Using iUML to create and manage the components of your system

Important terms:

1. Components – Domains
2. Integrated assemblies of these components (Domains) – Projects

In this section you will

- Create a Project for the case study.
- Construct a set of Use Cases for the case study.
- Partition the Gas Station System into a number of subject matter areas (Domains) and represent them on a Domain model.

Domains in our case will be:

1. Gas Station Control
2. Attendant Interface
3. Pump Interface
4. Operating System

Projects can be defined as: - *A project will specify an assembly of domains that constitute the system.*

How to create/add a project ?

Add a Project

You can now create the Gas Station System project.

a. Acquire a lock if necessary,
b. From the Tree RMB pull-down **Projects** > **Add Project** > **<New>**.
c. Fill in the details of the Project as follows:

Number	2
Name	Gas Station System
Key Letter	GSS
Short Description	Simple Gas Station Case Study
Mission Statement[a]	To develop a model of a Gas Station in order to develop an understanding of xUML models and to learn about the features of iUML.
Build Area	Leave blank for now

a. Of course, in a real system this would reflect the system mission.

d. Click OK.

Expand the tree for **Projects** to see the projects you have created, click on the project name to see its fine details.

Use Cases for the Gas Station System

In simple words – it is a *black box* view of the required functionality of the system.

Creating a Use Case Diagram

a. Expand the tree to **Projects** > **Gas Station System** > **Initial Version**.
b. RMB pop-up Initial Version and select **Show Use Case Diagram** > **<New Diagram>**.
c. Type the name Fuel Delivery and Purchases and click **OK**.

The diagram frame now shows an empty Use Case Diagram.

Add an Actor

a. RMB pop-up menu from the Diagram window and select **Add** > **Actor**.
b. Enter the name of the Actor **Attendant** and click **OK**.
c. Click where you want the Actor to be placed.
d. In the same way, add the Actors, **Customer** and **Tanker Driver**.

Add a Use Case

a. RMB pop-up menu and select **Add** > **Use Case**.
b. LMB click where you want the Use Case to be placed.
c. Enter the name of the Use Case **Customer Fuels Car** and click **OK**.
d. Now add the Use Case **Fuel Delivery** in the same way.

Add a Communication

A communication signifies a link between an Actor and a Use Case.

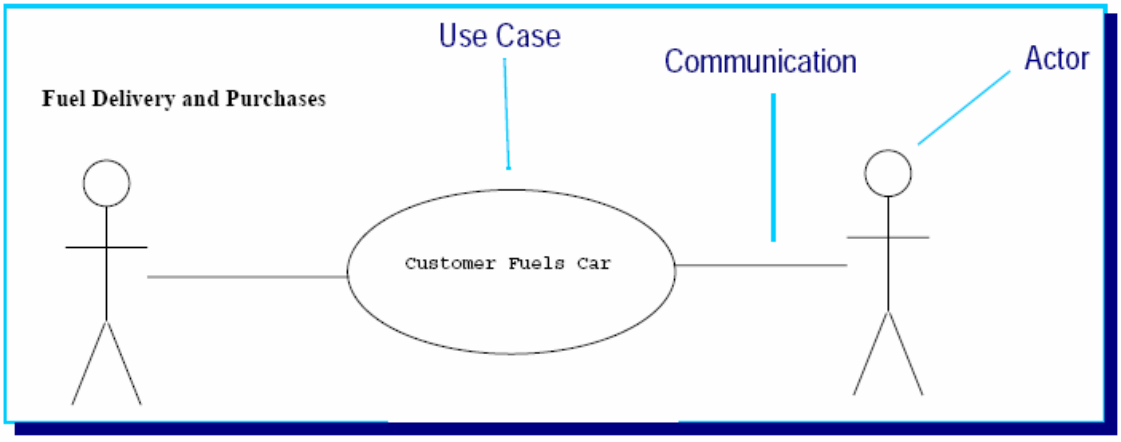
a. LMB select the Actor Attendant.
b. RMB pop-up menu and select **Add** > **Communication**.
c. LMB click on the Use Case Customer Fuels Car.
d. Add further communications from Customer to Customer Fuels Car and from Tanker Driver to Fuel Delivery

Note that Communications can be added in either direction. i.e. they can be added *from* the Use Case *to* the Actor, or vice versa.

Arrange the Use Case Diagram

To arrange the Use Case diagram, select Actors and Use cases and drag them into position on the screen. Hold the MMB and drag the target to view different areas of the model.

Use Cases should be moved to the middle of the diagram and Actors placed around them.



Creating a Domain Model

Types of Domains in xUML

1. Application Domains – represents the purpose of the system form end users point of view. Like Gas Station Control etc.
2. Service Domains – provide generic services to support the application domains. Like, Attendant Interface, Pump Interface, etc.

3. Architecture Domains – represents the globally applied design and coding strategies. Like Software Architecture, etc.
4. Implementation Domains – represent pre-existing software components. Like, C, Hardware Interface, etc.

Add a Domain to the Domain Model

The aim of this exercise is to represent the identified Domains on a Domain Model. A domain is represented as a package on the Domain Model.

a. Expand the tree to **Projects** > **Gas Station System.**
b. Click **Initial Version**.
c. From an empty space in the Diagram Window and RMB pop-up **Add** > **Domain Package**.
d. Enter the name Gas Station Control and description...:
The Gas Station Control domain manages the dispensing of fuel, customer payments and tank levels.

...in the Domain Details dialog and click **OK**.

e. Point to where you wish to place a domain symbol and click.

Adding a Domain Dependency to the Domain Model

The domains are organised on the Domain Chart into a hierachy of Client Server Dependencies. In each Client/Server pair the Client is the Domain which requires the services of the Server in order to fulfil its mission.

Each dependency between a Client/Server Domain Pair has a description which explains the purpose of the dependency.

Now that the Domains have been added to the Domain Model you can add the Client/Server Dependencies.

In the Diagram window:

a. From the Client Domain Gas Station Control RMB pop-up **Add** > **Dependency**.
b. Click on the server domain Attendant Interface.

The Client/Server dependency has now been added to the diagram.

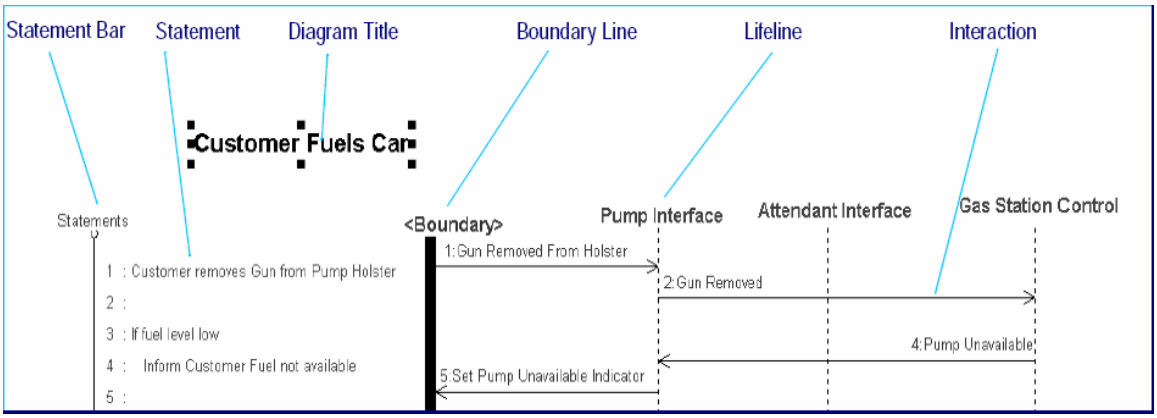
Adding a Description to the Dependency

a. From the **Gas Station Control** to **Attendant Interface** Dependency line on the Package Diagram, RMB pull-down **Modify** > **Edit Description**.
b. Enter the following description in the edit field and click **OK**.

Sequence Diagrams

Using Sequence Diagrams to model the interaction between domains.

Creating Sequence Diagrams: - it captures the system threads which corresponds to a single Use Case.



To Show the Sequence Diagram

a. Expand the tree to **Projects** > **Gas Station System** > **Initial Version** > **Use Case Model** > **Use Cases** > **Customer Fuels Car** > **Sequence Diagrams**.
b. Single click Customer Fuels Car.
c. On the Diagram window, click the pencil icon to acquire the lock if required.

The Diagram Frame will display an empty sequence diagram for Customer Fuels Car, containing a vertical Boundary Line and an empty Statements column..

A Lifeline represents a Domain involved in the Use Case.

a. From an empty area of the Sequence Diagram, RMB pop-up menu and select **Add** > **Lifeline**.
b. Select the domain name Pump Interface from the pick list.
c. Place the lifeline a few cms to the right of the Boundary Line.

Add an Interaction

Each interaction represents a service required by the source domain and a service provided by the destination domain or system boundary. In this exercise you will add the sequence of domain interactions for part of the thread corresponding to the 'Customer Fuels Car' Use Case.

a. LMB Select the source Lifeline or Boundary Line - in this case, select the Boundary Line.
b. RMB pop-up menu **Add** > **Interaction**.
c. LMB Click on the destination lifeline - in this case the Pump Interface.
d. Select **New Interaction** from the pick list.
e. Enter the Interaction Name **Gun Removed from Holster** (you can leave the Provided and Required service fields blank) and click **OK**.
f. Enter the corresponding statement text **Customer removes Gun from Pump Holster** and click **OK**.
g. Repeat this sequence for the interaction from Pump Interface to Gas Station Control called **Gun Removed**, leaving the Statement text blank.

Add a statement

a. LMB select the vertical dotted 'Statement Bar'.
b. Ctrl. RMB pop-up **Add** > **Statement**.
c. Select <End> from the **Position before Statement...** pick list.
d. Enter the statement **If fuel level low** and click **OK**.

Chapter 5. Static Modeling of Domains

So far you have seen how a set of Use Cases:

- Expose the usage of the Gas Station System from an external viewpoint.
- Supplement the system requirements so that the initial domain partitioning can be specified using a Domain model.
- Describe the interactions amongst the domains (domain level threads), using a number of sequence diagrams.

From this analysis, an analyst may:

- make a qualitative assessment of the viability of the domain model;
- establish the interfaces between the domains;
- identify the required dependencies between domains.

What you will do in this section

- Learn about the purpose of Static Modelling
- Add Classes, Associations, Attributes and Identifiers to the Static Model
- Explore the viewing and diagram editing facilities of iUML

Static Model Overview

A static model is represented on a class diagram which is enhanced with supportive descriptions and data definitions. By contrast, the dynamic model describes the events and operations which will cause the data described in the static model to change.

Domain Creation

Each of the Domains on the Domain model will require a corresponding Domain within the database in which the analysis can be conducted - as such you will be required to add one or more domains to the database.

In this exercise you will add the 'Gas Station Control' domain to the database which has been identified on the Domain model.

Add a Domain to the database

a. Select **Domains** > **Acquire Lock** > **Top Level** if necessary.
b. Select **Domains** > **Add Domain** > **<New>**.

Fill in the details of the Domain as follows:

Number	3
Name	Gas Station Control
Key Letter	GSC
Short Description	Simplified Gas Station Control
Mission Statement	To control the dispensing of fuel, customer payments and tank levels.
Type	Application
Initial Version	Standard
Build Area	leave blank

c. Click **OK** to add the Domain to the Database.

Classes

The first stage of editing the static model involves adding the classes to the class diagram and the associations between them.

Showing the Class Diagram

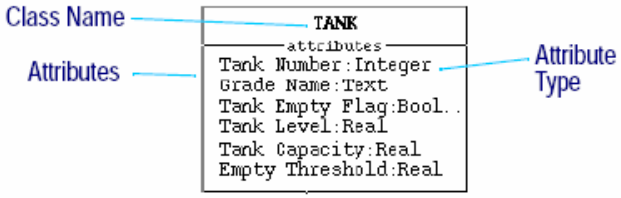
The Class Diagram 'belongs' to the domain version.

a. Expand the tree to Domains > Gas Station Control.
b. Click **Initial Version**.

The Class Diagram, which will be initially blank, is shown in the Diagram Window.

Adding a Class

This is the graphical representation of the TANK class:



a. On an any empty area of the Diagram Window RMB pop-up **Add** > **Class**..
b. Enter the Class Details for the TANK class in the table and click **OK**..
c. Click to place the Class
d. Repeat for the other classes shown in the table.

Number	Name	Key Letter
4	TANK	TNK
3	PUMP	PMP
2	DELIVERY	DLV
5	FUEL GRADE	FGR
6	TRANSACTION	TRN

Attributes

Adding Attributes to Classes

a. From the Class RMB pop-up **Add** > **Attribute.**
b. Enter the attributes and details from the table and click OK (leaving the **Default Value** blank).

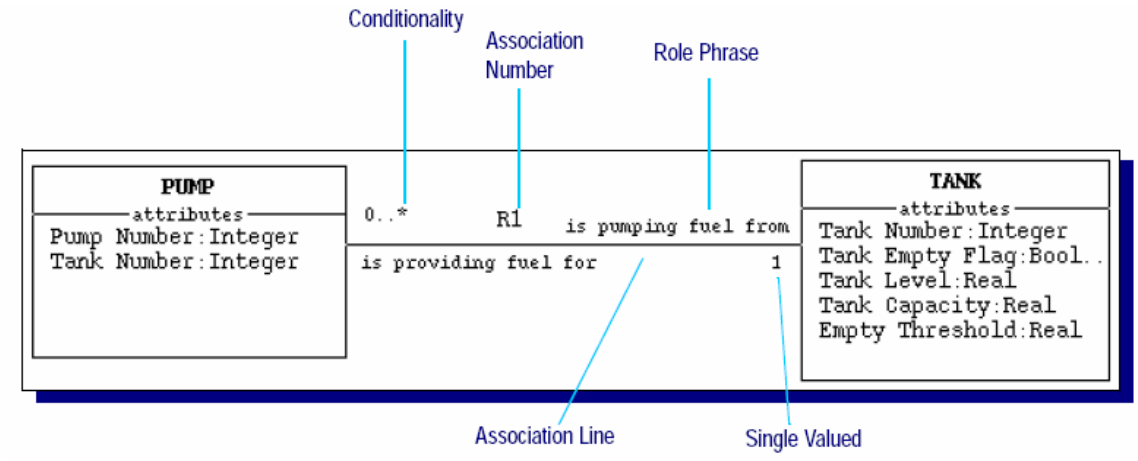
Identifiers

Adding an Attribute to the Preferred Identifier

a. From the attribute RMB pop-up **Modify** > **Add to Preferred Identifier**.
b. Here are the other attributes which are the preferred identifiers of their Classes:

Class	Attribute to make preferred
TANK	Tank Number
FUEL GRADE	Grade Name
TRANSACTION	Transaction number
DELIVERY	Delivery Time
PUMP	Pump number

Associations



Add an Association between TANK and PUMP

a. From the PUMP class RMB pop-up **Add** > **Association**
b. Click on the TANK class.

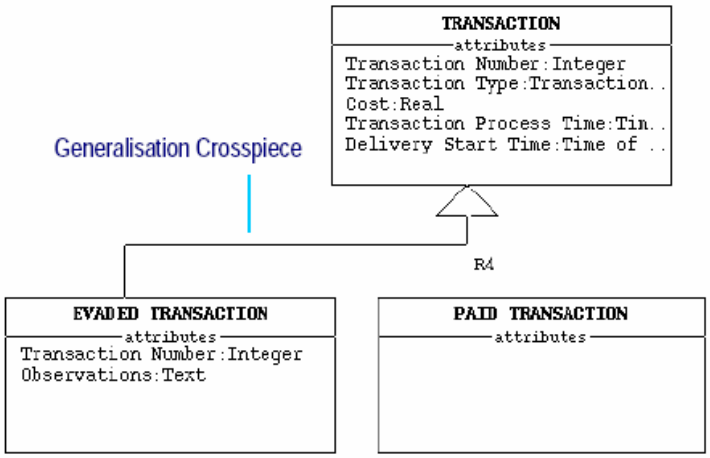
Generalizations

To create a Generalisation:

a. From PENDING TRANSACTION class and RMB pull-down **Add** > **Generalisation**.
b. Click on the TRANSACTION class to create the association between them.

To add the other Subclasses to the super/sub class hierarchy:

a. From the classes RMB pull-down **Add** > **Generalisation**.



b. Click on TRANSACTION and select the existing Generalisation from the pick list.

Adding new types

The Class Collaboration Diagram

The Class Collaboration Diagram (CCD) summarises the interactions between:

- **Classes** - Representing classes whose dynamic behaviour is specified only by operations - such classes are stereotyped <<class>> on the CCD;
- **Classes with State Machines** - Representing classes whose dynamic behaviour is specifed both in terms of operations, and whose state-dependent behaviour is defined by a state chart and a corresponding state transition table - such classes are stereotyped <<state machine>> on the CCD;
- **Terminators** - which are abstractions of entities outside of this domain. There are a number of different types of terminator (the semantics of which are detailed in the iUML Modeller Manual and the ASL Reference Guide):
  - **Non Counterpart Terminators** - which are stereotyped <<terminator>> on the CCD;
  - **Association Terminators** - stereotyped <<association terminator>> on the CCD;
  - **Specialisation Terminators** - stereotyped <<specialisation terminator>> on the CCD.

There are two forms of interaction supported:

- **Synchronous** interactions - an operation invocation, represented by an arrow with a filled arrowhead. Note that the direction of the arrow reflects the invocation direction, not the data flow (which could be in both directions if there are both input and output parameters associated with the operation);
- **Asynchronous** interactions - the transmission of a signal, represented by an arrow with an open arrowhead. Note that the direction of the arrow reflects the transmission from sender to receiver.

Adding a State Machine to the Pump class

a. Expand the tree to **Domains** > **Gas Station Control** > **Initial Version**.
b. RMB pull-down **Initial Version** > **Show Class Collaboration Diagram**.
c. In the Diagram Window, RMB pop-up PUMP class > **Modify** > **Create State Machine**.
d. Confirm by clicking **Yes**.

The PUMP class now has <<state machine>> annotated at the top of the class. This indicates that the PUMP class has a State Machine.

e. You can also add state machines to the classes TRANSACTION, TANK and DELIVERY.

Terminators

Interactions

Add a Signal Transmission

a. From Class Terminator from which the signal is to be transmitted, RMB pop-up **Add** > **Send Signal**.
b. Click on Class Terminator to transmit to.
c. Click <**New Signal**>.
d. Enter the name of the Signal Transmission and click **OK**.

The Signal Transmission is added to the Class Collaboration Diagram.

Add an Operation Invocation

a. From the Class Terminator from which the Operation Invocation is to be transmitted, RMB pop-up **Add** > **Call Operation**.
b. LMB click on Class Terminator to transmit to.
c. Click **New Operation**.
d. Enter the name of the Operation Invocation and click **OK**

The Operation Invocation is added to the Class Collaboration Diagram

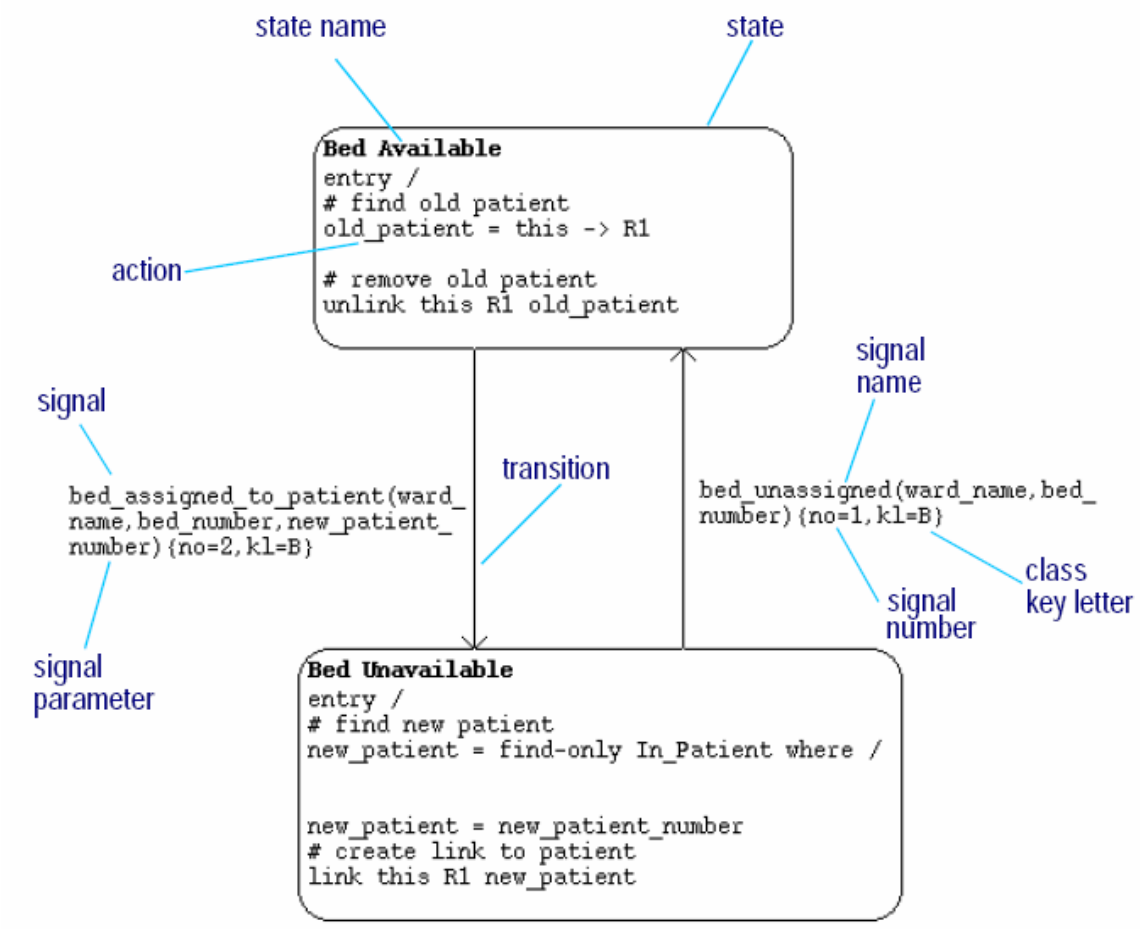
Adding Operations

You can write Operation Actions for the Operation Invocations you have added to the Class Collaboration Diagram. The Operation you will load for Create Transaction creates a pending transaction which a customer needs to pay for.

a. From the Operation Invocation name Create Transaction on the Class Collaboration Diagram RMB pop-up **Show Details**.
b. RMB pop-up <Method> text beneath **Method** and select **Load**.
c. Navigate to the <iUML release directory>/Manual/Tutorial_Files/ASL_code directory and double click createTransaction.txt.
d. Click **OK** on the Action Description dialog. The new Action is shown in the Text Window.

State Chart

Example-



Add a State

A state represents a condition of the class, subject to a defined set of rules, policies or physical laws.

a. Expand the tree to **Domains** > **Gas Station Control** > **Initial Version** > **Classes**.
b. From PUMP, RMB pop-up **Show State Chart**.
c. From the empty chart RMB pop-up **Add** > **State**.
d. Enter the state name Waiting For Customer then click **OK**. The state will be shown on the diagram.
e. LMB click on an area of the frame to place the State.
f. You can add some or all of the states to the diagram in this way. As the states form a cycle of behaviour, add their boxes clockwise round the screen.

Name
Waiting For Pump Enable
Ready To Pump
Pumping
Pumping Paused
Fuel Delivery Complete

Edit Actions

Add Transitions

Attach a Signal