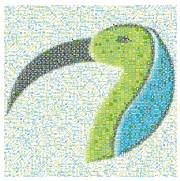


Papyrus Tutorial:
How to use Sequence Diagrams in Papyrus MDT?

Editor	LANUSSE Agnes, CEA LIST
Status	First version
Version number	0.1
Date of preparation	26/08/10

Authors

Editor name (first/last name)	Company	E-mail	Initial
LANUSSE Agnès	CEA LIST	agnes.lanusse@cea.fr	AL

Authors name (first/last name)	Company	E-mail	Initial
	CEA LIST		SL

Revision chart and history log

Version	Date	Reasons
0.1	10/08/26	First contribution.

Table of contents

1 Introduction and initial steps

The purpose of this document is to provide a tutorial for Papyrus users explaining how to use UML2 Sequence Diagrams. In depth knowledge on Sequence Diagrams in UML2 specification that can be found on the OMG site at http://www.omg.org/spec/UML/2.3/.

A Sequence Diagram is one way to describe interactions in UML2. The Behavior part of the specification has a section devoted to interactions which provides a set of concepts to describe interactions between parts in a system.

Basically, Lifelines represent parts life cycle, communication between parts are represented by message exchanges and executions are represented by Execution specifications that can be Action or Behavior execution.

Several diagrams are available in the specification :

- Sequence Diagram
- Communication Diagram
- Interaction Overview Diagram
- Timing Diagram

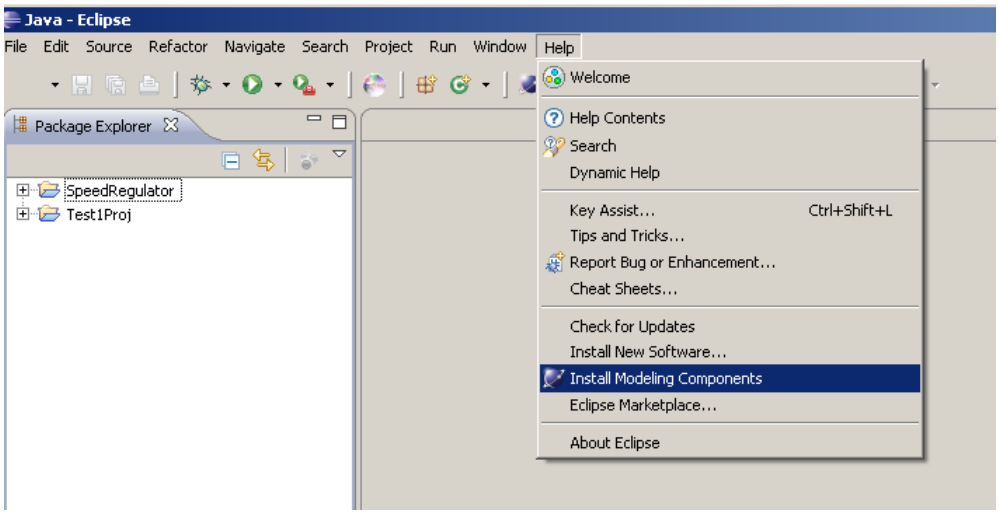
Papyrus MDT implements support for the first two diagrams. In this document we describe Sequence Diagram utilization.

We list here under a few prerequisites to start this tutorial.

1. getting Papyrus
2. creating a first papyrus project
3. creating a new model

1.1 Getting Papyrus

Before starting , you must have a version of Papyrus MDT installed. Papyrus MDT is one of the incubating projects of Eclipse MDT. The official release (0.7.0) is now public and available under Helios. You can easily download it using the Helios Modeling Discovery UI and selecting the papyrus Component.

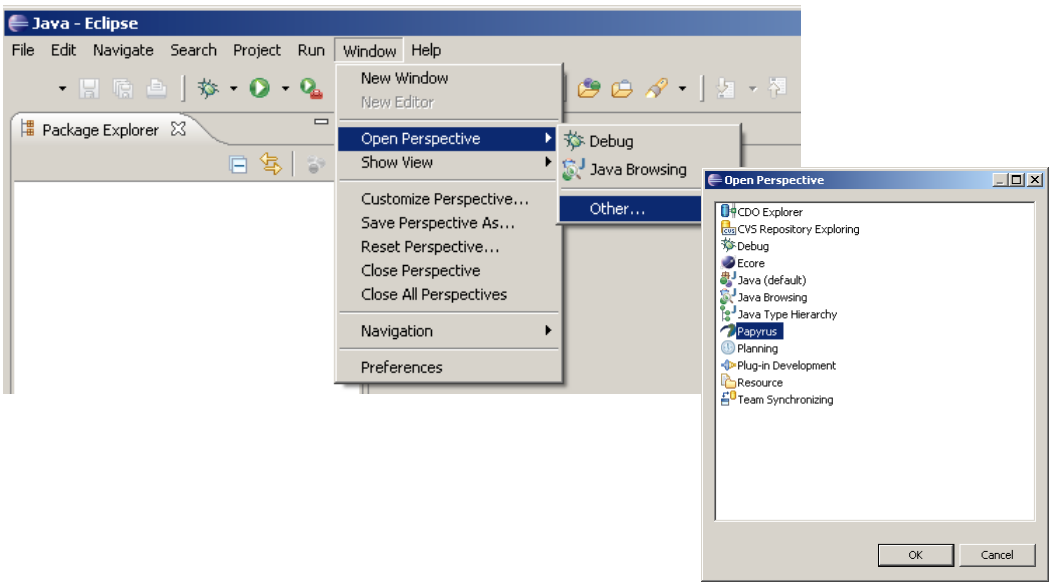


In this release, you will find all latest features: the main UML diagrams, several SysML diagrams, advanced profile support, customization framework, XText and Modisco integration and many other cool things! You can find documentation on the Papyrus website:

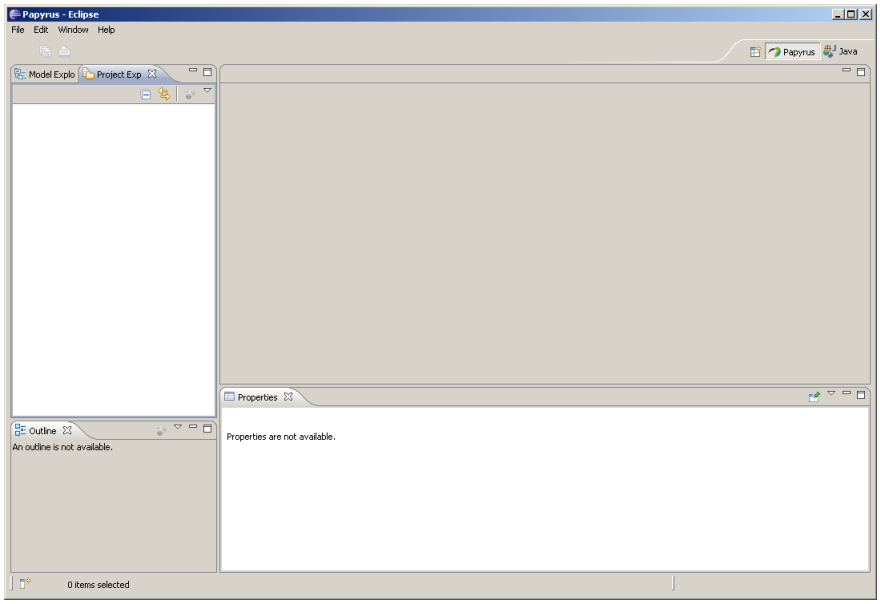
http://www.eclipse.org/modeling/mdt/papyrus/

1.2 Creating a Papyrus project

Once papyrus is installed, you can open the papyrus perspective.



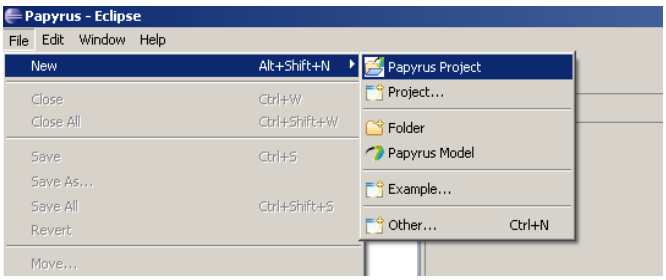
…. and get the following display



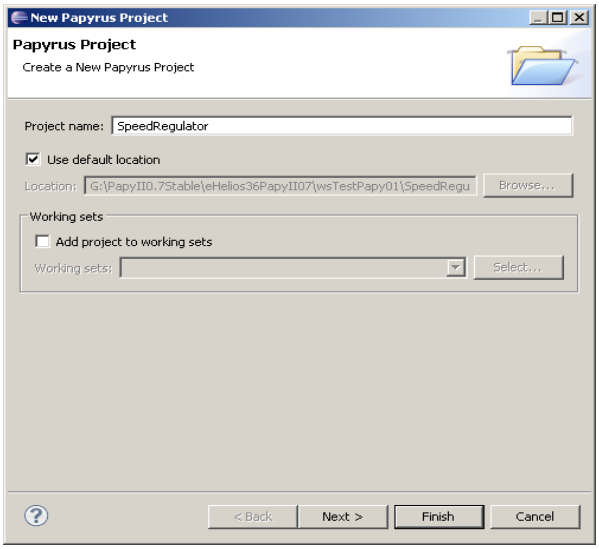
Now we can create a new project; this takes 4 steps.

1. In the main tool bar menu

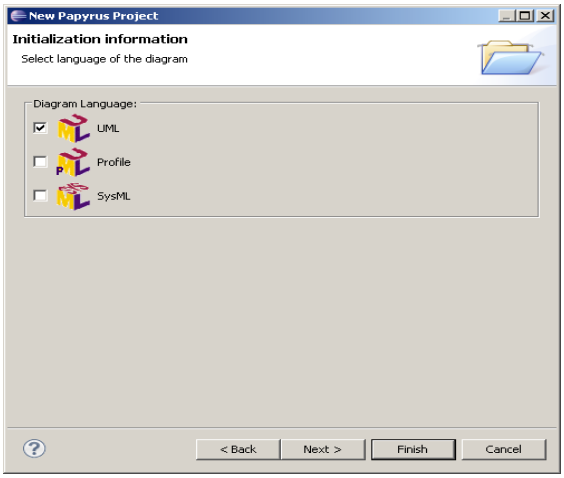
Select File > New > Papyrus Project



2. Set project Name

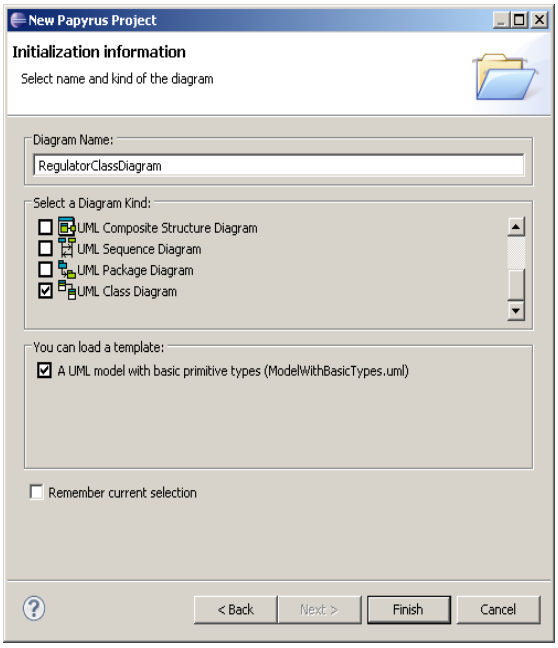


3. Select Language (UML or SysML)

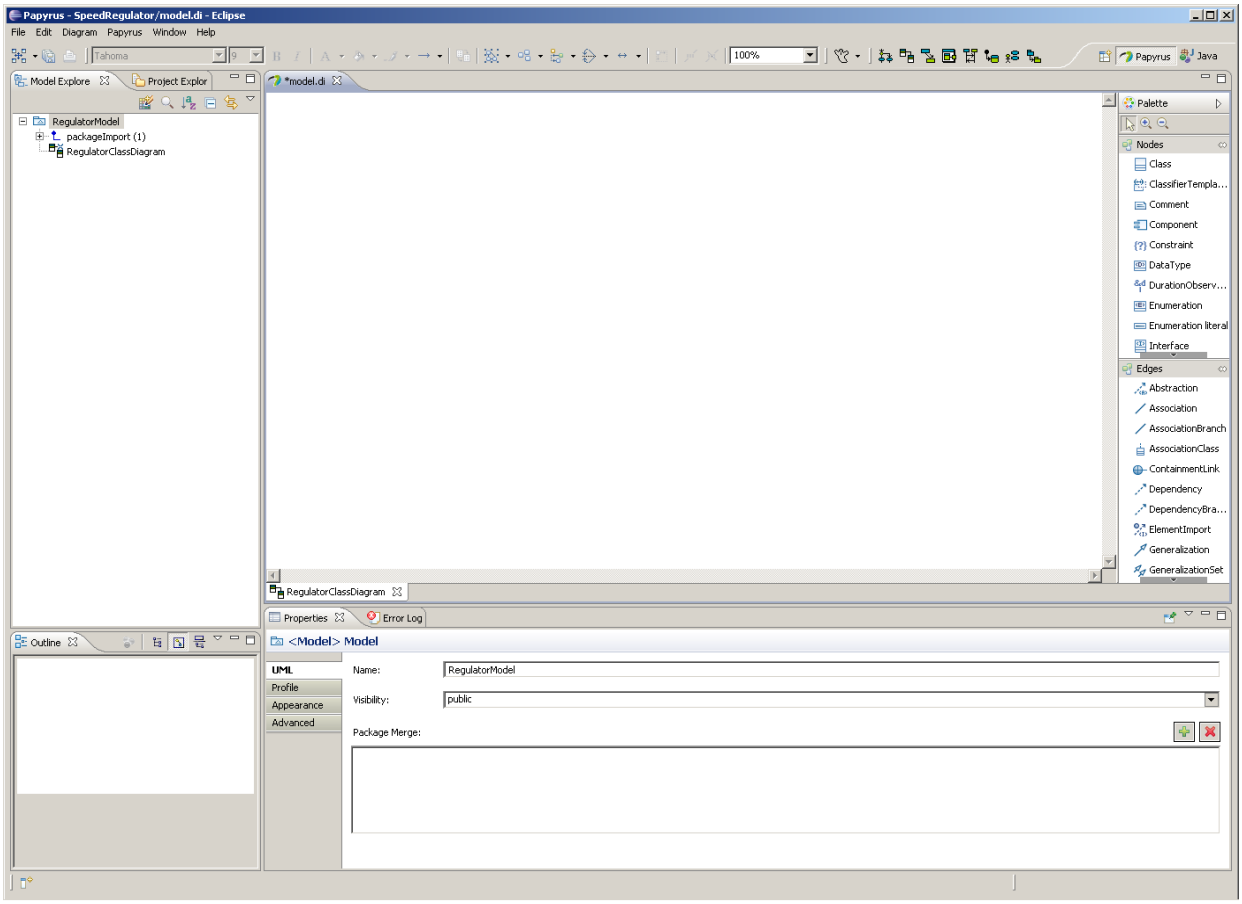


4. Create first Diagram

The project is ready to be created, the wizard asks for a first diagram to be created. Usually we start with a ClassDiagram, we can ask to pre-install basic types (template section) and provide a name for the diagram.



The project and a first model is created and we get the following display.

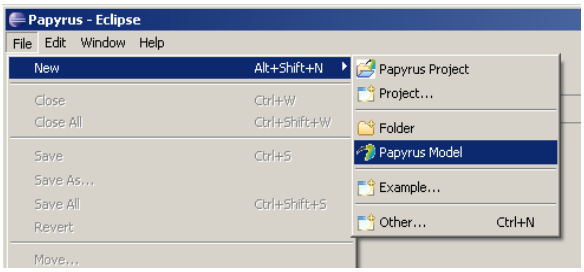


1.3 Creating a new model

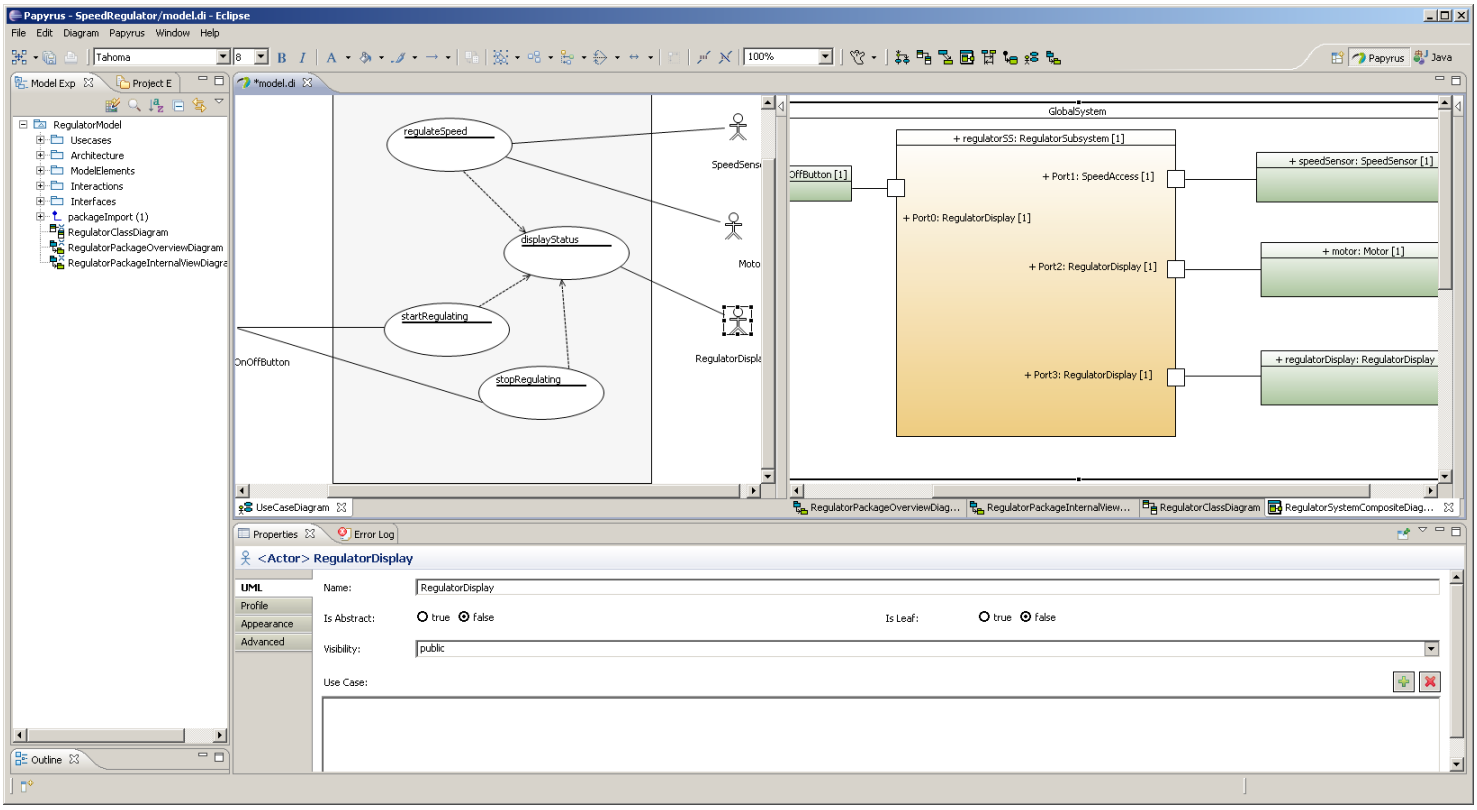
Following the wizard, we have created directly a model.

We can also create a model from the menu wizard

File > New > Papyrus Model



Normally we could create directly a sequence diagram, but since the sequence diagram describes interactions between parts of the model. It is recommended to start by creating at least a class diagram and a composite diagram (the composite diagram describes a system architecture obtained by assembly of parts representing more or less instances of classes defined in the class diagram.). In the following figure, we show a simple example of CruiseControl Subsystem.

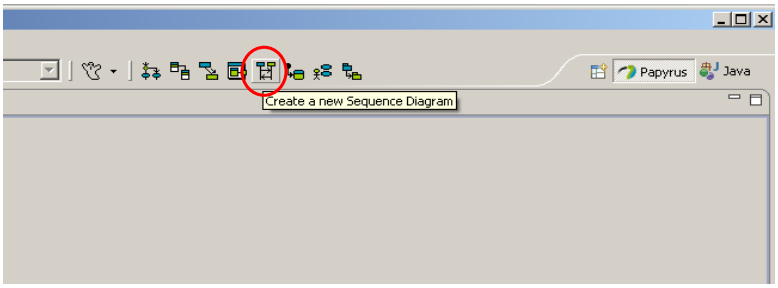


For this system, we have defined a Use Case Diagram (left view) and a Composite Diagram (right view) corresponding to an external view of the system to illustrate its integration within the environment. The Subsystem will itself be refined. We will now create a sequence diagram to illustrate interactions between the subsystem and the environment.

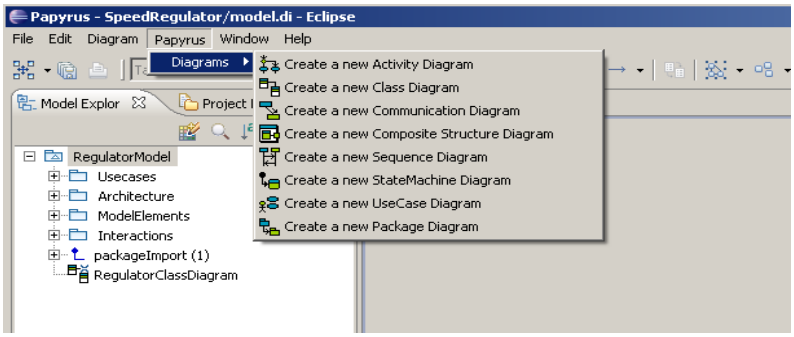
2 Creating a simple sequence diagram

As any other diagram in Papyrus MDT, a sequence diagram can be created trough three different ways :

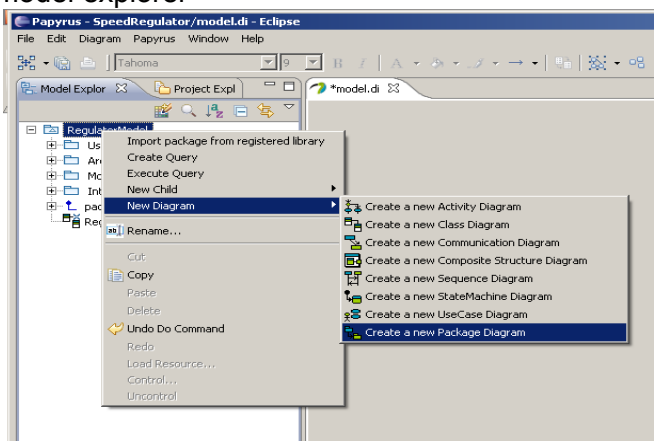
- from toolbar



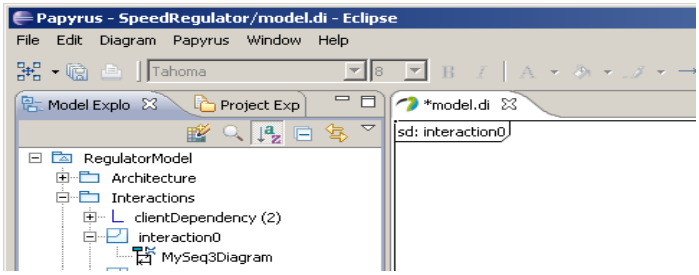
- from main menu



- from model explorer



The result is the creation of an interaction element and a diagram.



Remark: If an interaction exists already and it is selected when creating the diagram, a new diagram is created for this interaction, so several diagrams can be created providing different views of a same interaction.

We recommend to create the diagram and the corresponding interaction in a separate package named Interactions for instance, since many events will be created when we start creating elements in the diagram.

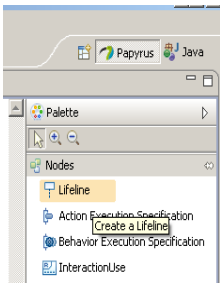
Once the diagram is created, we can start populate it. The three main elements of such diagrams are lifelines, messages and Execution specifications. Let's see how to create them.

2.1 Adding lifelines

In order to create a lifeline we have to do three things

1. Select the lifeline tool from the palette on the right side of the graphic view
2. Place the lifeline in the diagram (click in the diagram)
3. Set properties in the properties view

2.1.1 Select LifelineTool in the palette

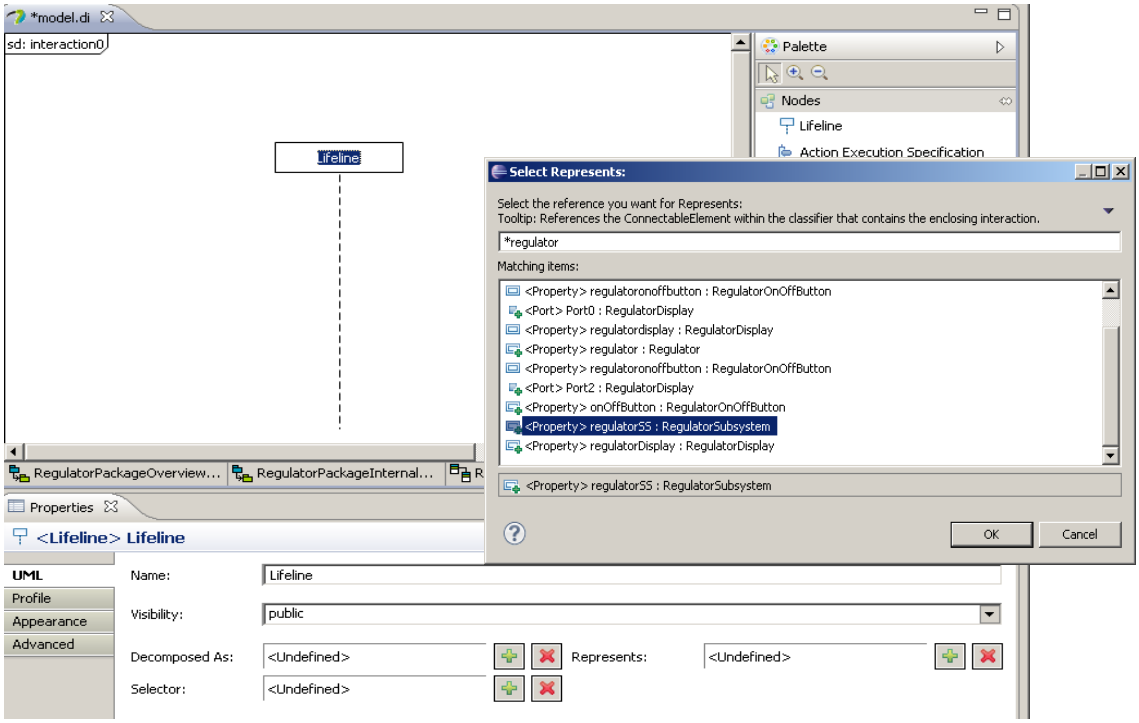


2.1.2 Set properties in the properties view

The property view displays properties attached to lifeline, support is provided to select .

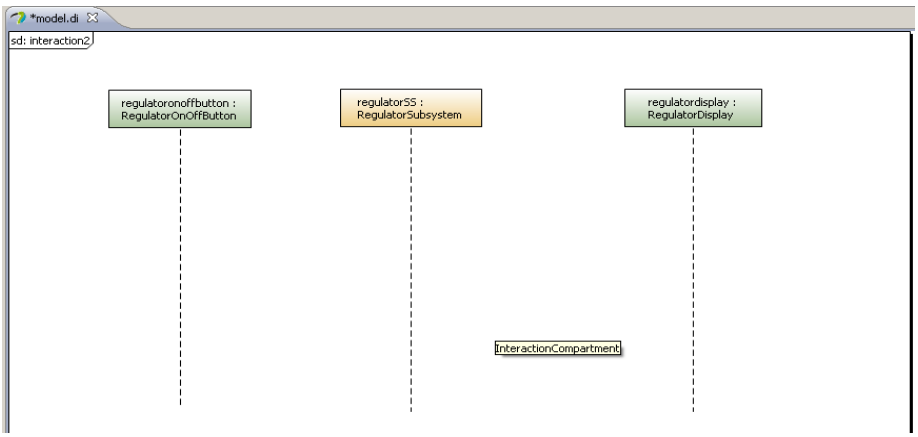
- ***name.*** This is the name of the lifeline itseltf (notice that what generally appears in the header of the lifeline is not it's name but the name of the part it represents when set).

- ***represents.*** This is a reference to the part represented by the lifeline, a popup window helps select it from a list of parts.
- ***selector***. This is used when arity of part is greater than one in the composite diagram, the selector helps specify which instance is represented by the part.
- ***decomposed as*** . Is used to refine Lifelines (this is not detailed here).



Clicking the green "+" sign activates a selection pop-up. Just select the desired element.

Repeat the operation for each Lifeline to be created in the diagram



Remark: You can change the appearance of the lifelines using the appearance tab as in any other diagram. Here we choose the green colour to represent environment and orange colour to represent the regulator subsystem.

2.2 Adding messages (basic - asynchronous)

Communication between parts is described through message exchanges.

Messages can represent ***operationCall*** (synchronous or asynchronous) or **signal** (asynchronous) communication.

When a ***synchronousCall*** is created a ***replyMessage*** is expected.

Specific messages ***createMessage*** and ***deleteMessage*** denote dynamic management of Lifelines during application lifecycle; they are respectively devoted to LifeLineCreation and LifeLineDeletion.

Creating a message with Papyrus consists in:

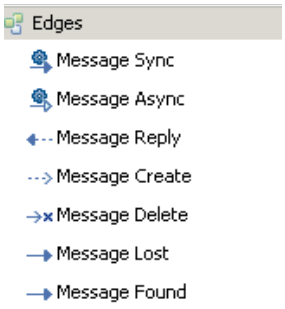
1. selecting the desired tool from the palette;
2. clicking in the source Lifeline to set starting anchor point
3. clicking in the target lifeline to set the ending anchor point
4. select/create the corresponding signal/operation from the appearing pop-up

Notations

- Asynchronous Messages have an open arrow head.
- Synchronous Messages typically have a filled arrow head.
- The reply message has a dashed line.
- Object creation Message has a dashed line with an open arrow.
- Lost Messages are described as a small black circle at the arrow end of the Message.
- Found Messages are described as a small black circle at the starting end of the Message.

2.2.1 Select desired tool from the palette

Possibilities offered by the palette are the following:



Notice that some restrictions apply.

- Up to now, we have not created Execution Specifications so Sync Message and Reply cannot be used (we will see this later on).
- Message create can be created only if the target point is already present (the Lifeline has been drawn before).
- Message Delete must have a delete event as target point.
- Message Lost and Message Found management is not stable (still under development)

2.2.2 Select source and target points

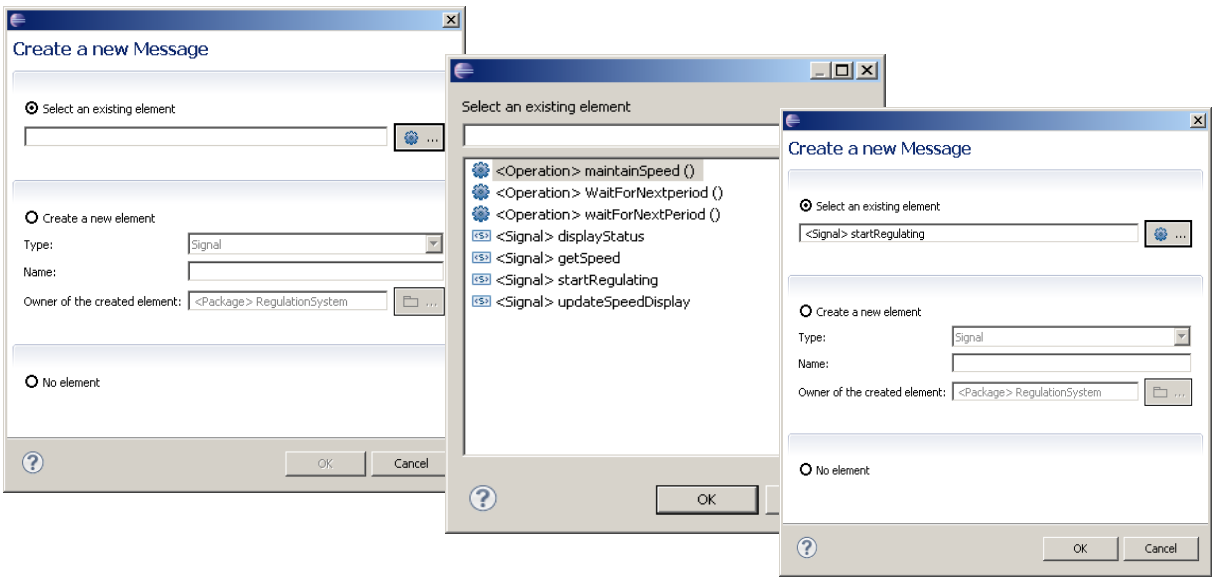
Source and target points can be selected on two different lifelines (normal case) or on the same lifeline in which case it denotes a “recursive” call (see figure below).

Restrictions on target points depend on the type of target element; for instance as said in the previous section, a Synchronous Call must start from an Execution Specification.

Known problem: It may happen that a message cannot be created if target point is a selected element. If this happens, just deselect the element and try again.

2.2.3 Select/Create Operation/Signal attached to message

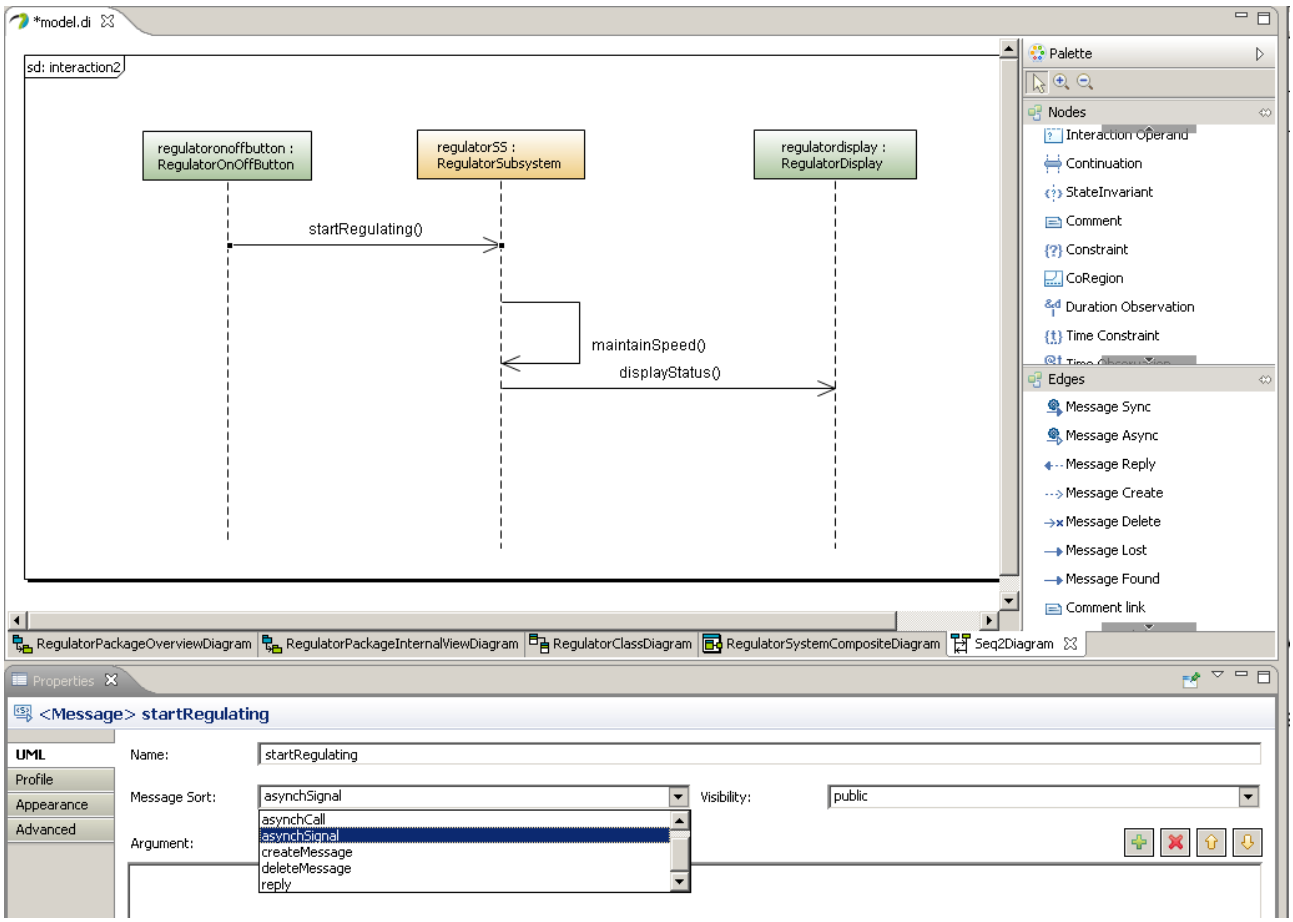
Papyrus provides support to create/select dynamically operation or signal to be attached to the message. This is provided through pop-up windows.



You can select an existing element (signal or operation), create a new one or do nothing (in case of a preliminary general overview scenario).

If a creation is required then the signal will be created after a name is set and a directory selected.

We can obtain a diagram like the one hereunder.



Notice that the nature of the message sort property can be set from the property view.

2.2.4 Creating Create and Delete Message

The process of creation is exactly the same as above but restrictions apply.

- Create Message

  The message must target towards a lifeline already existing in the diagram.

- Delete Message

  The message must target a destruction event

2.3 Adding execution specification

Until now we have used Sequence diagram only to show communication scenarios. But, sequence diagram can represent complete execution trace including communication events as long as execution events. The Execution Specification concepts is an abstraction to represent execution. It can be of two sorts: Action Execution Specification or Behavior Execution Specification. Actually, what is really handled are events attached to start and end of these execution specification.

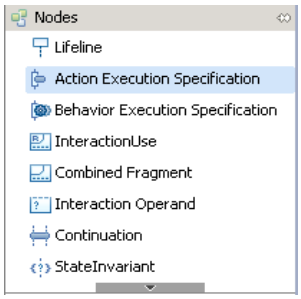
So a lifeline can be considered as a trace of events corresponding to communication and/or executions. In the diagram, execution occurrences are represented by one of these two UML entities. The notation - a rectangle - is similar in both cases.

Creating an Execution Specification with Papyrus consists in:

1. selecting the desired tool from the palette;
2. clicking in the desired Lifeline to set ES
3. if necessary resize ES
4. select/create the corresponding signal/operation from the appearing pop-up

2.3.1 Select desired tool from the palette

Select Action Execution Specification or Behavior Execution Specification.



2.3.2 Place the ES on the Lifeline

To place the ES on the desired Lifeline proceed quite naturally:

- Click within the Lifeline at the point where you want to add an Execution Specification
- Maintain the mouse down and drag it downward on the lifeline
- Release the mouse

Remark : You can also place an ES on an existing ES, in this case it represents recursive execution.

2.4 Adding messages (complements)

Once Execution Specification exist we can improve the first design and add synchronous messages to the sequence diagram.

In a development process one could consider that at preliminary design all messages are asynchronous and that at design we start more detailed specification.

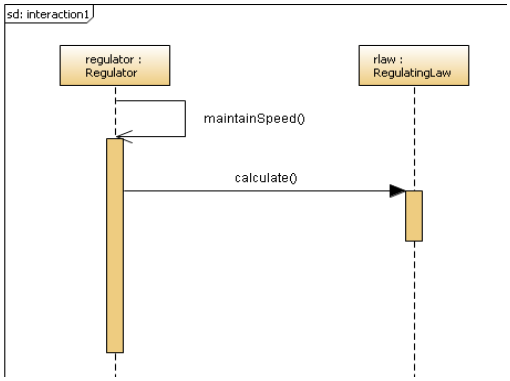
2.4.1 Adding synchronousCall (regular)

Adding a synchronous call is done in the same way as for an asynchronous message.

1. Select tool in the palette
2. Select source point
3. Select target point
4. Select operation

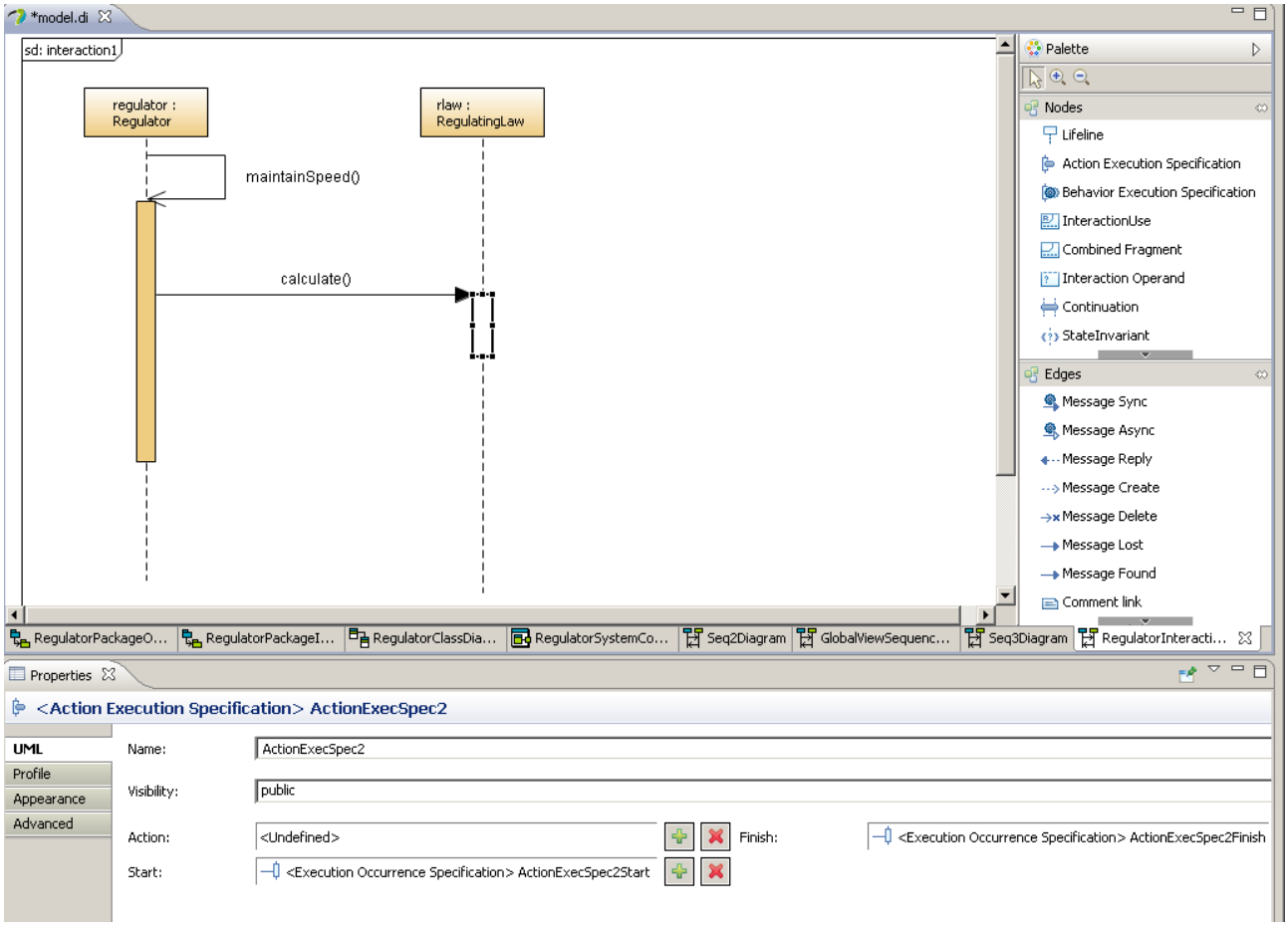
However there are two differences.

- The starting point must be within an ES.
- The ending point should normally be also an ES in which case the target anchor must coincide with the target ES start point as shown hereunder. However, Papyrus MDT provides support for dynamic creation of target ES (see next section).



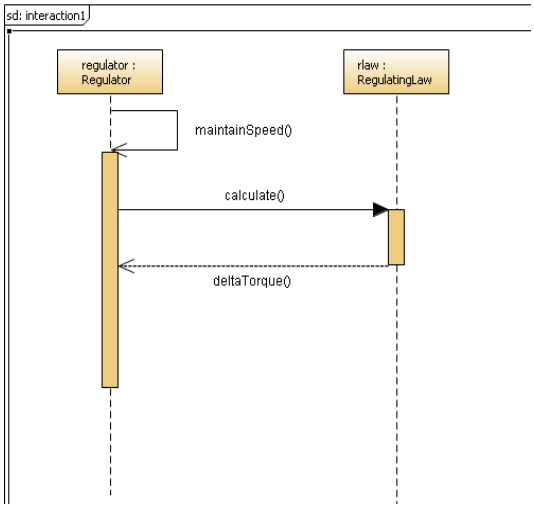
2.4.2 Adding synchronousCall (with dynamic ES creation)

In this case the target point at step 2 can be one point on the target Lifeline not covered by an ES. The message pop-up helps select/create the operation involved, then an Action Execution Specification is created.



2.4.3 Adding a Reply message

Adding a reply message is done in a similar way as any other message, but the message must start from an ES (involved in a Synchronous Call) and arrive to the initiating ES .



3 Combined Fragments

3.1 Introduction

The introduction of combined fragments within Sequence Diagram results from the will to write complex traces in a synthetic way that is to say in a rather caricatural way “grouping several scenarios within one with more expressive power”. Through the use of Combined Fragments the user can describe a number of traces in a compact and concise manner.

Formally, a combined fragment defines an expression of interaction fragments.

In the specification, the Combined Fragment is characterized by an interaction operator.

The interaction operator tells the kind of combined fragment is considered. Several generic types have been defined to express loops, alternatives, parallel sections etc...

Depending on the kind of interaction operator, one or several operands are defined. For instance in alternatives CFs, there must be several operands, while in Loops CFs, only one operand is required.

Operands contain fragments representing the events included in the CF. It is important to notice that only events are considered and not elements such as ES or messages. These events represent occurrence specifications (start or finish event corresponding to an execution, send or receive events corresponding to messages). However, these events are not represented graphically, only messages and ES are.

The different CFs are considered as sub parts of the diagram and may appear in sequence (or imbricated) in a sequence diagram. This leads to very expressive general scenarios, while basic sequence diagrams represent only one particular scenario.

3.1.1 The different types of Combined Fragments

The UML2 specification describes generic Combined Fragments (see section 14.3.3 of the specification UML Superstructure Specification, v2.2). The kind of Combined Fragment is determined by setting the Interaction Operator attribute.

The possible values are : alt, opt, par, loop, break, critical, neg, assert, seq, strict, ignore, consider.

These different kind of Combined Fragments are of different nature.

For instance, seq, loop, alt represent classical programming control structure for sequential, loops, conditional structures.

Opt represents an optional section. That is depending on the value of a guard, the internal is done or nothing happens.

Seq is a weak sequential section. Occurrence specifications is strict within one operand) but occurrence Specifications on different lifelines from different operands may come in any order. A variant is a Strict combined fragment where a strict order is obsereved between the behaviors of the internal operands.

3.2 Creating CF with papyrus

Papyrus provides a first implementation of combined fragments that is described below. We first describe the general principles for CF construction then we detail specific issues for each type of interaction operator.

3.2.1 General principles

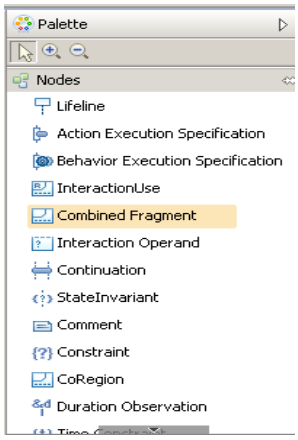
Creating a Combined Fragment consists in 6 steps

1. select the CF tool in the palette
2. select the type of combined fragment consider/ignore or other
3. place the CF on the diagram
4. select the type of Interaction operator (by default a Seq CF is created).
5. create the operands if necessary (by default one is created)
6. set operand properties in the property view

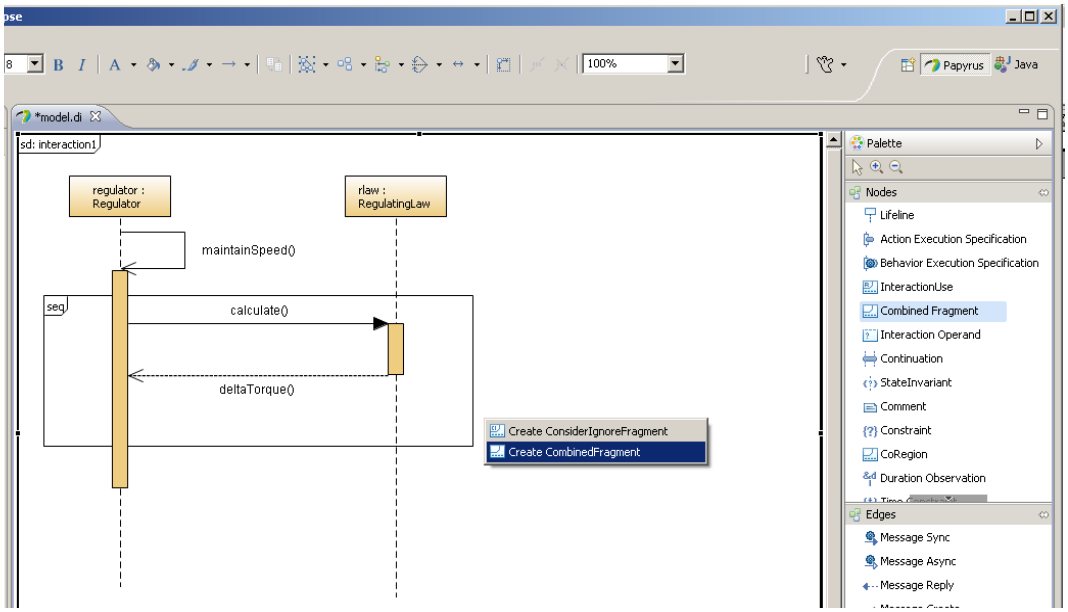
Steps 4., 5., 6. vary according to the interaction operator selected and specific rules may apply.

We illustrate hereunder the first three steps.

1. Select CombinedFragment in the palette



2 + 3 Select the type Consider/Ignore or regular CF, and place CF over Lifelines (covered)



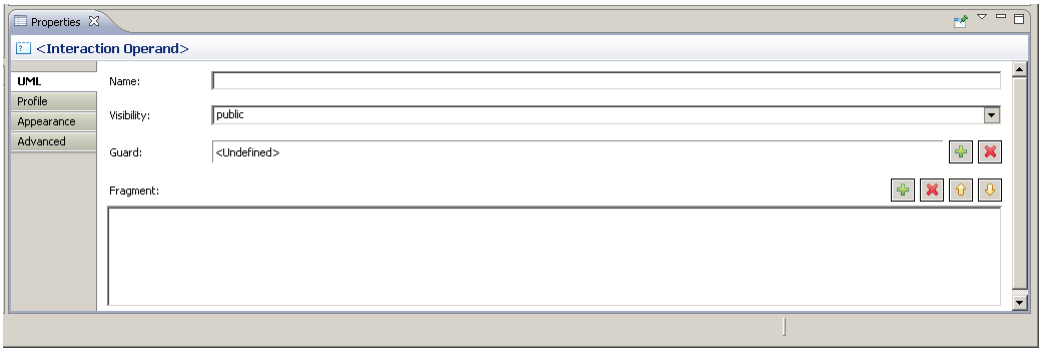
4. The resulting CF is a Seq combined fragment. By default an interaction operand has been created also.

5. Do nothing only one operand must be created

6. Set operand properties in the property view.

To do so it is necessary to click within the CF in order to select the operand, then the property view correspond to the operand selected (if several operands areas are separated by discontinuous lines).

The appearance of the property view depends on the type of operand. In the default case, we have to set the name of the operand.



In the current implementation, property views do not handle yet all operands initialization. So we have to use specific turnarounds. This should be fixed soon. In the mean time we suggest to follow the procedure given in annex.

The main point concerns the specification of the guard which cannot be created directly from the property view but must be defined by an Interaction Constraint and then selected through the property view. See details in Annex.

3.2.2 Seq

The Seq CF describes an ordered sequence of events. The order can be partial or strict.

Seq creation is the simplest one since it is the default case and has been described in the previous section. Nothing else has to be done.

3.2.3 Loop

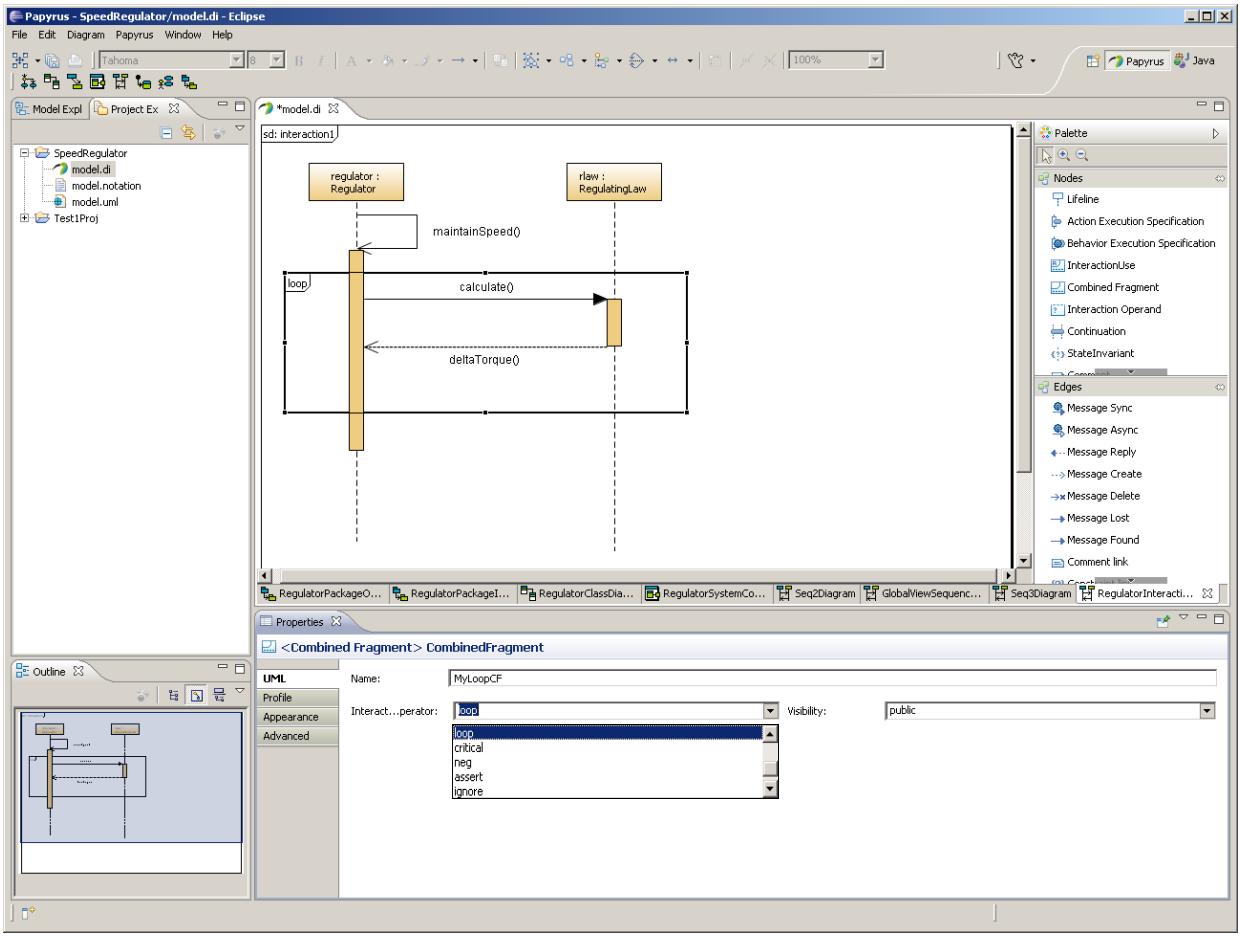
A Loop describes an iteration structure. Internal events are repeated a certain number of times depending on loop parameters and a guard condition.

A Loop CF has exactly one operand, this operand contains the events enclosed within the area drawn by the CF.

Creating a Loop follows the same algorithm as the Seq CF until step 4.

1., 2., 3 Apply generic procedure

4. Change the interaction operator through the property view.



5. Nothing special has to be done for this step

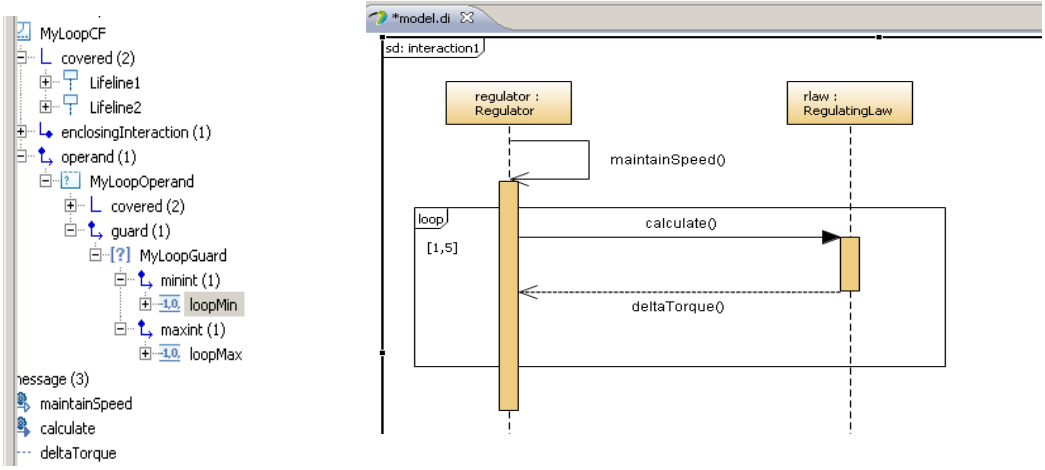
Loop requires one unique operand as Seq.

6. Set Loop Operand parameters

In the current implementation, property views do not handle yet all operands initialization.

So we have to use specific turnarounds (as described in the annex). This should be fixed soon.

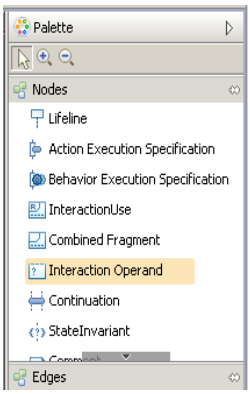
We finally get the Loop CF.



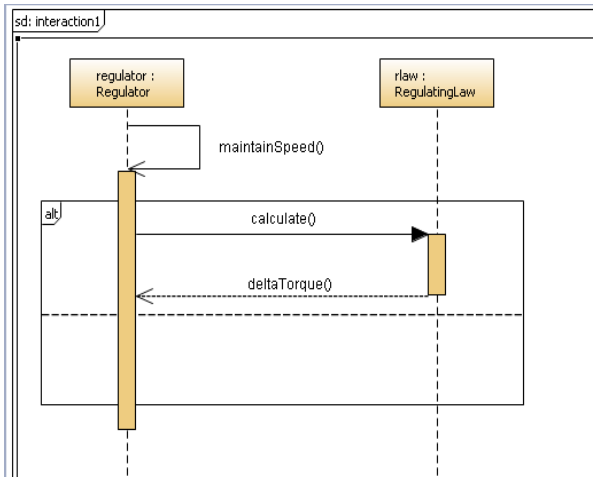
3.2.4 Alt

To obtain a parallel CF, just proceed as in general case except that an Alt CF has several operands.

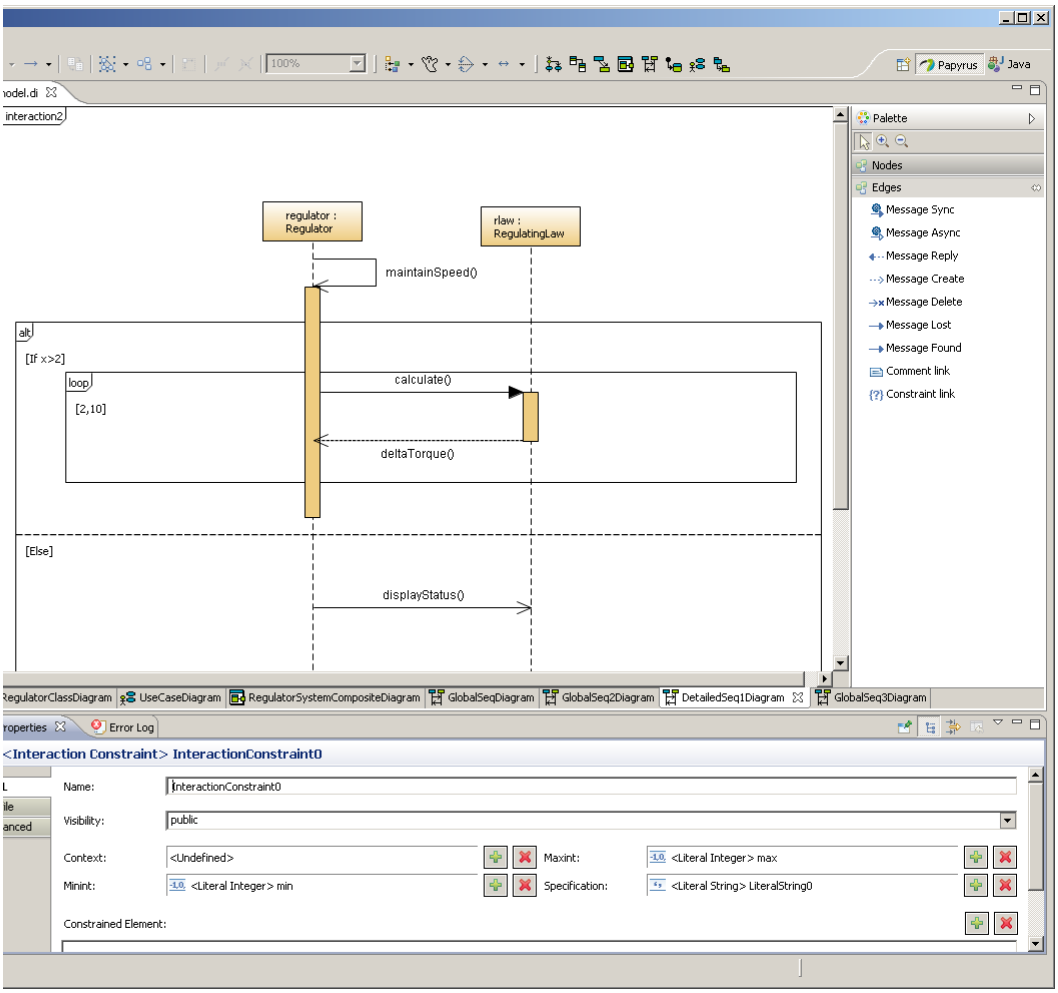
So after having created the Alt CF, you must add a new operand. This is done by selecting the operand tool in the palette.



Then click within the CF area, a new area is created for the new operand.



Then we have to set the guards for If and Else conditions of the Alt sections. For the moment we have to follow the same procedure as described for the Loop CF.



3.2.5 Par

To obtain a parallel CF, just proceed as for a Seq CF, then change Seq to Par then add as many operands as wanted.

3.2.6 Break

The break CombinedFragment represents “a break scenario”. The operand (it must be unique) represents a “scenario that is performed instead of the remainder of the enclosing InteractionFragment”.

Normally the break combined fragment has a guard. If the guard is set to false, then the operand is ignored and the rest of the enclosing operand is executed. If it evaluates to true then the break scenario is executed.

If no guard is specified, the behaviour is not defined deterministically. This should be avoided.

Rules:

A break CombinedFragment should cover all Lifelines of enclosing Interaction Fragment.

A break CF is normally included within an other CombinedFragment.

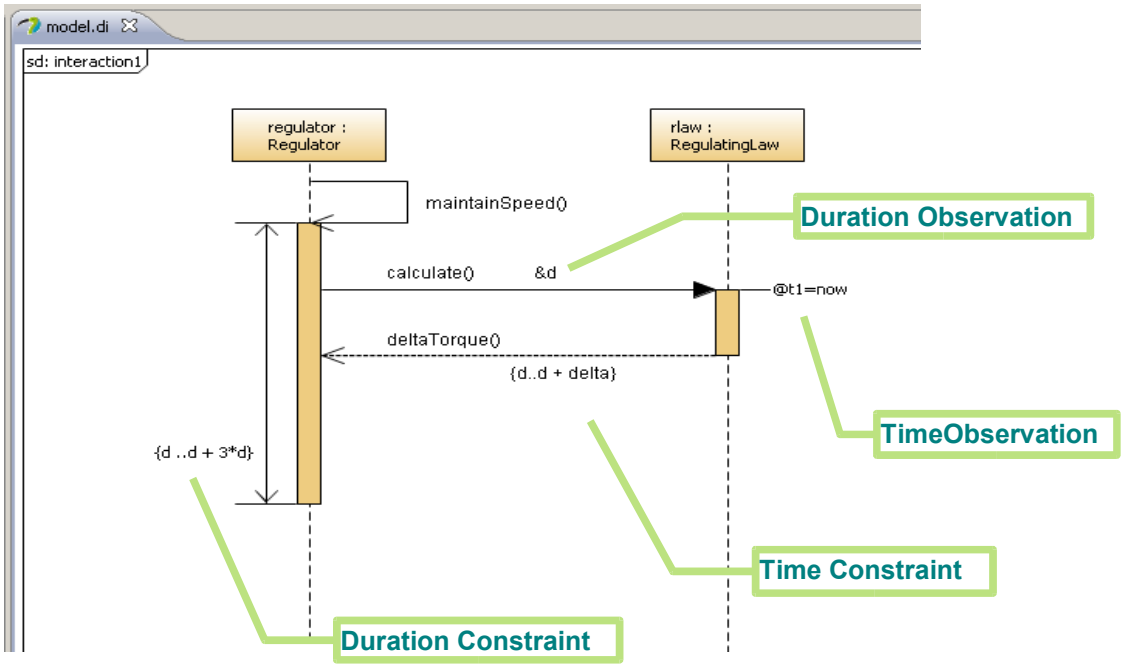
To create a Break CF, just do as for a Loop CF and set the guard constraint instead of min and max properties.

4 Adding Timing Information to Sequence Diagrams

Two classes of concepts are used to set temporal information onto diagrams.

The first one is the concept of Observation, it covers two types of information: Time Observation to tag instants on a lifeline, and Duration Observation to tag a duration (for instance the transmission delay of a message).

The second one is the concept of Timing Constraint, they can be a constraint on an instant on a lifeline or a constraint on a duration. Both constraints are expressed as a Time Interval.



4.1 Adding Observations

Adding Observations can be achieved directly from the palette

4.1.1 Time Observations

To set a Time Observation on a sequence diagram, just select the tool from the palette then click on the target location on a lifeline. This must coincide with an occurrence specification (Execution Start, Finish, MessageEnd).

4.1.2 Duration Observations

Duration Observation can be set only on messages in the current implementation.

Select the tool in the palette click on the message starting point maintain the mouse down and follow the message, release mouse button on the message end. A “&<duration_label>” (here &d) is displayed above the message. This label can be used in constraints expressions.

4.2 Adding Constraints

Constraints can concern durations (duration constraints) or instants (time constraints).

4.2.1 Time Constraints

Time constraints may be set using the tool in the palette and clicking on the event corresponding to the constrained event. An interval appears. The values can be either constants or expressions involving observations. Up to now, nothing is provided in Papyrus to set these values, it is thus necessary to use a similar procedure as the one described in the annex to set parameters of Combined Fragments.

4.2.2 Duration Constraints

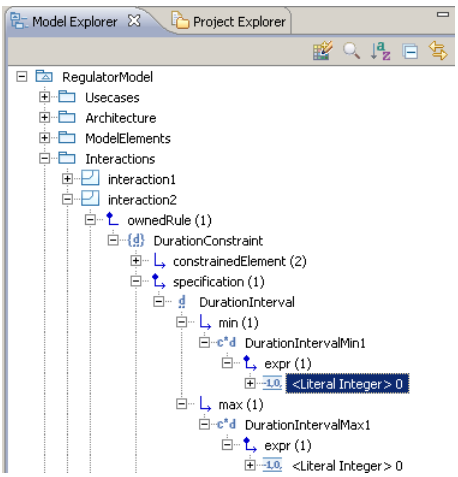
Duration constraints apply on interval between two events. In the current implementation, duration constraints can be set between events located on the same lifeline, or to specify a communication delay constraint on a message.

Select the tool from the palette, click at starting position within Lifeline, maintain the mouse down and drag it downward until the target event (end of the duration interval), then release mouse button. A figure appears showing top, bottom of the interval with a vertical arrow between and a label in which default interval bounds are displayed. In the case of a duration constraint on a message, select the message ends instead and the display appears as an interval under the message.

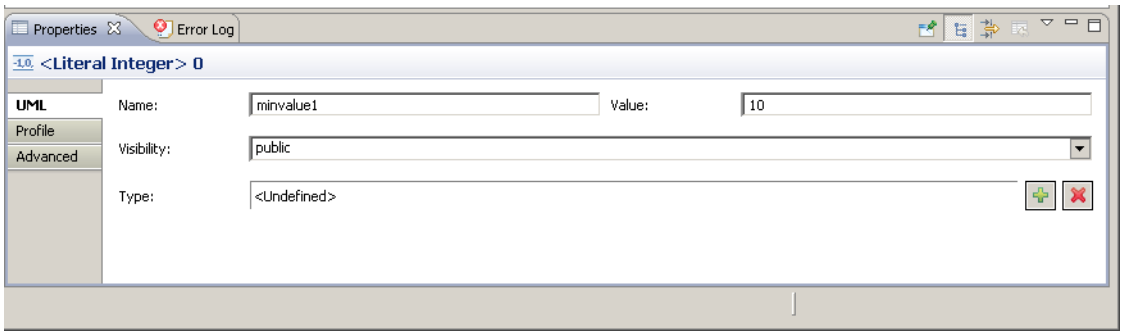
By default the values min and max are integer set to 0. They can be changed via the property view. To do so, follow the procedure hereunder:

1. go the model explorer
2. select the Duration elements attached to the DurationInterval of the constraint.

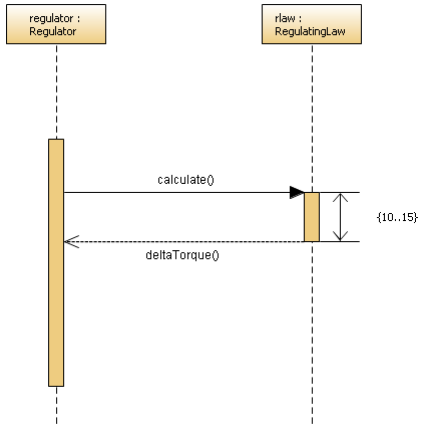
   It is located in the rule entry of the interaction enclosing the duration constraint.

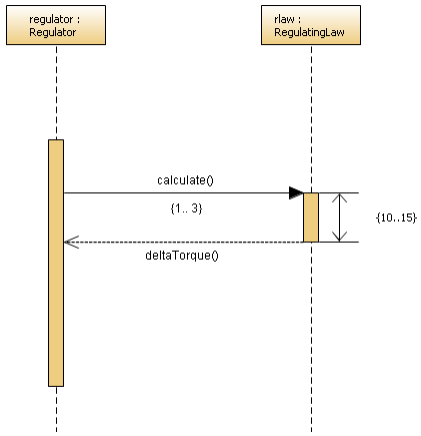


3. in the property view change the values (min and max)



4. The duration constraint is updated in the diagram





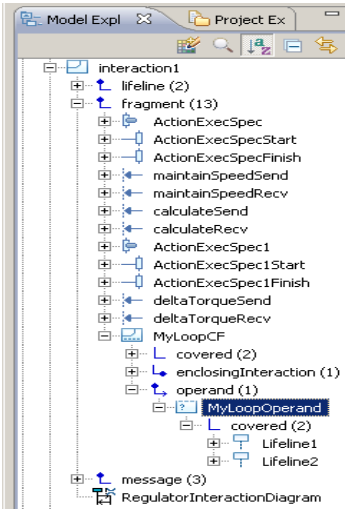
5 ANNEX

This annex describes turnarounds to palliate current lacks in Papyrus MDT to set properties of Combined Fragments. This should be solved soon, but for the moment, we describe two possibilities : the first one is to use direct UML edition of the model following the procedure provided hereunder; the second one is to create some elements from the model explorer and then to reference them in the properties view

5.1 Building elements from UML editor

a) go to model explorer and select interaction operand

You can see the tree structure of interaction1 whith MyLoopCF and its operand.



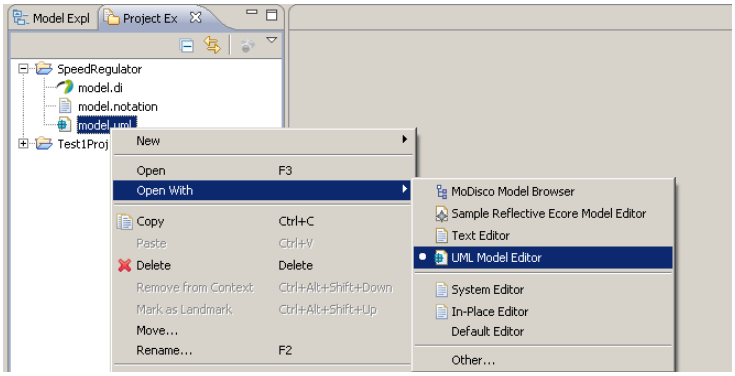
Normally, we should be able to add a guard as a child of this element using a right click menu, but this is not available either. So we will have to build it directly through UML.

b) close the model

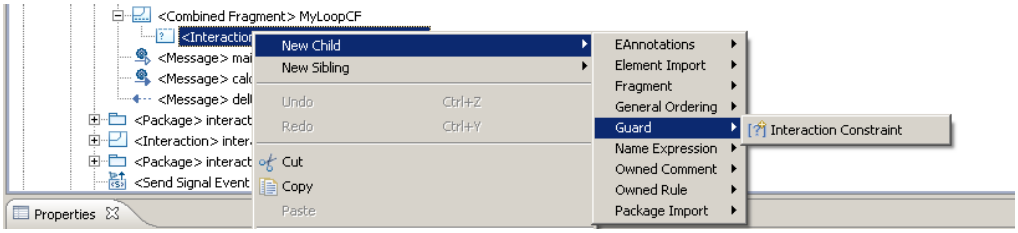
c) go to ProjectExplorer

d) select model.uml then right click to get the pop-up menu

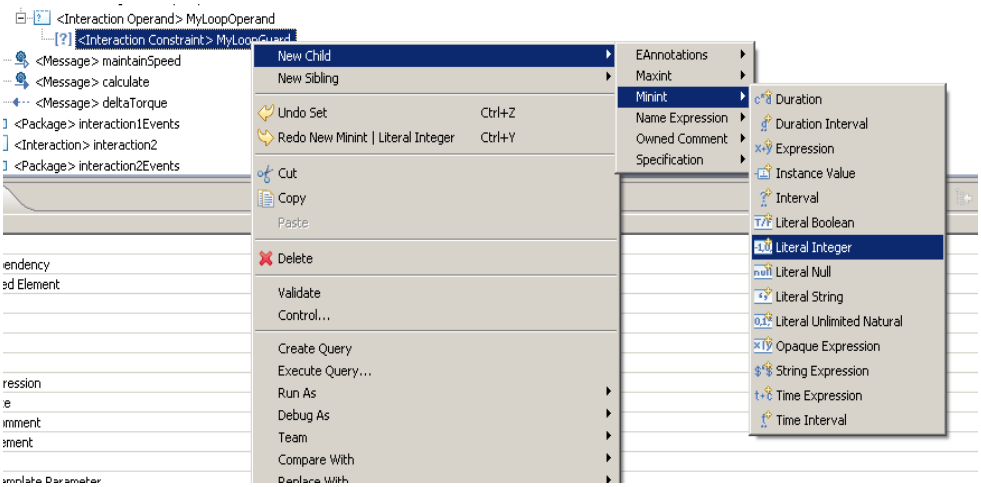
e) select open with Uml Model Editor



f) create a guard (interaction constraint).



Create the min and max value



Set the values of min and max

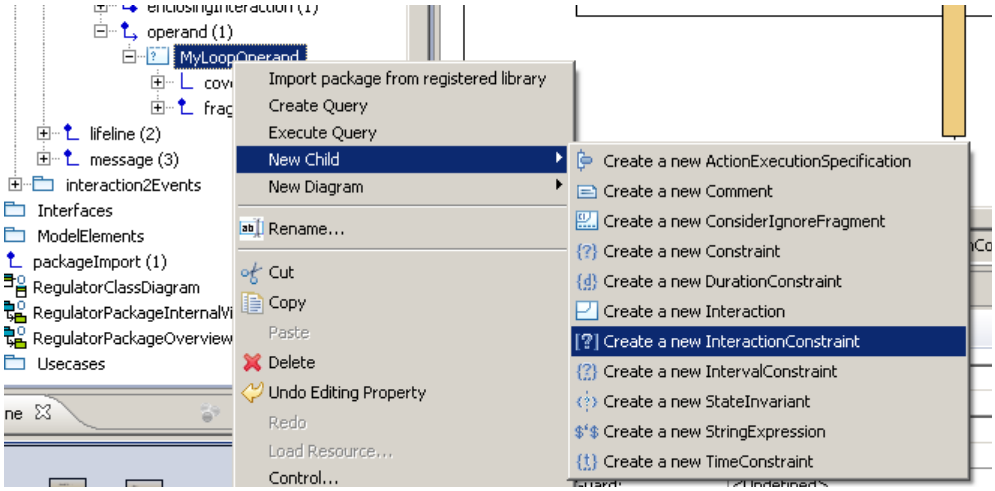
g) save the model

h) close and come back to papyrus editor

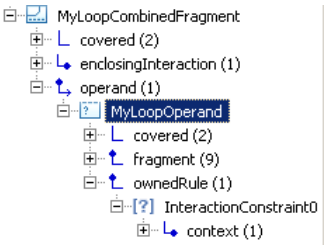
5.2 Creating Elements from Model Explorer

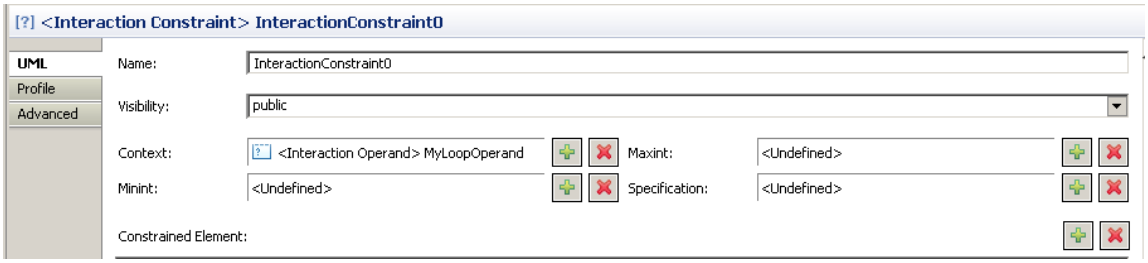
Another possible procedure is now also available

1. Go to the interaction operand in the model explorer

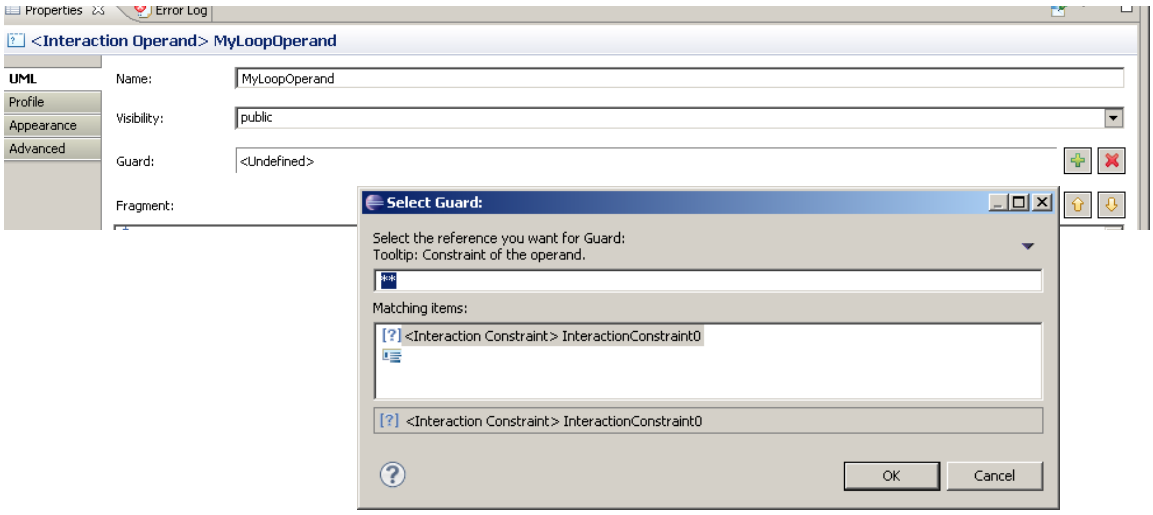


2. Create an interaction constraint
   - you get an owned rule

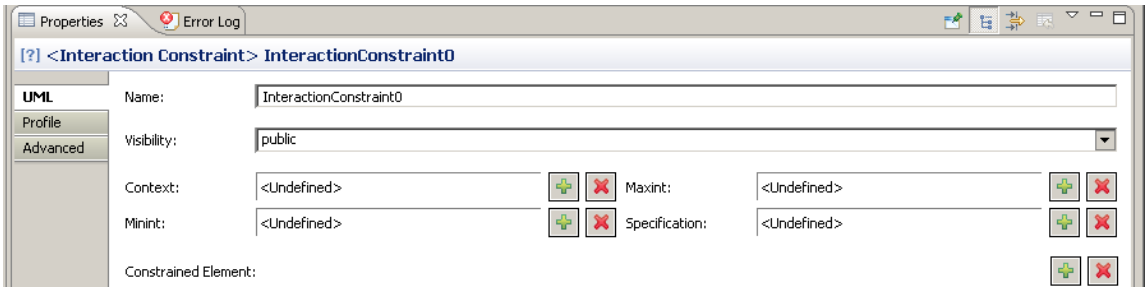




3. Select this interaction constraint from the properties view to set the guard attribute



We get the constraint attached to the guard of the operand.

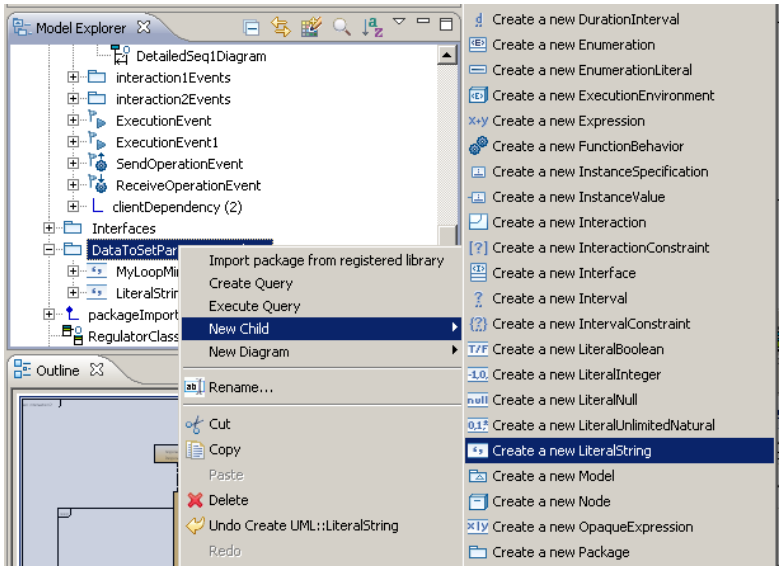


4. Set the attributes of the guard

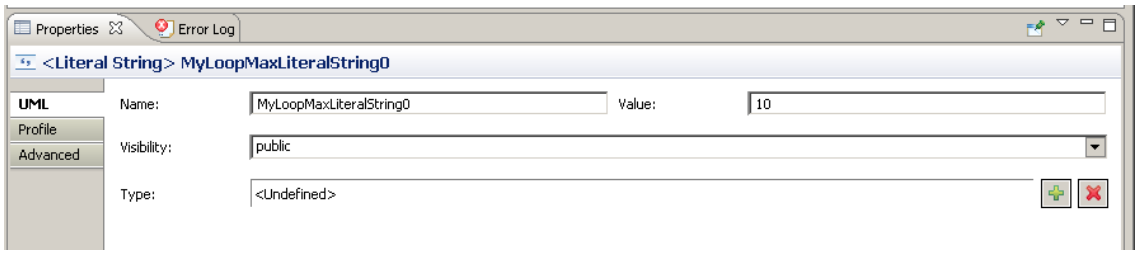
The current implementation of properties view does not support element creation. So we create the literals corresponding to values in a separate Package (DataToSetParameters).

Then we select these elements from the properties view to set the guards attributes.

5. Create literals fro min and max attributes



set values in the properties view.

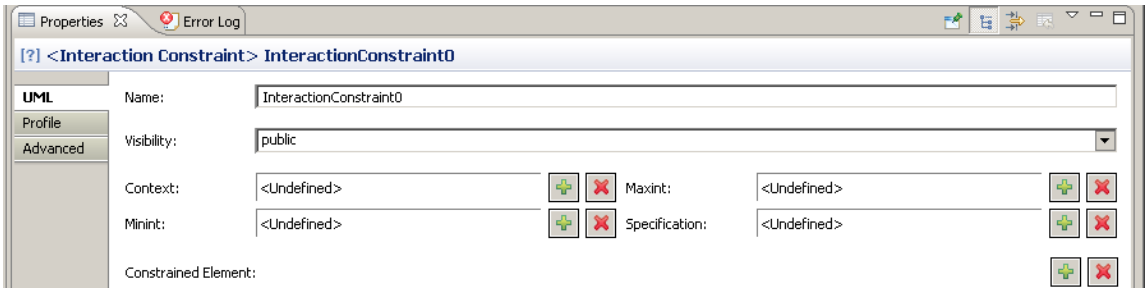


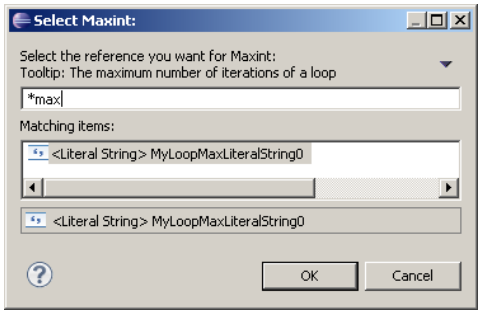
Remark : semantics of attributes actually depend on the kind of combined fragment.

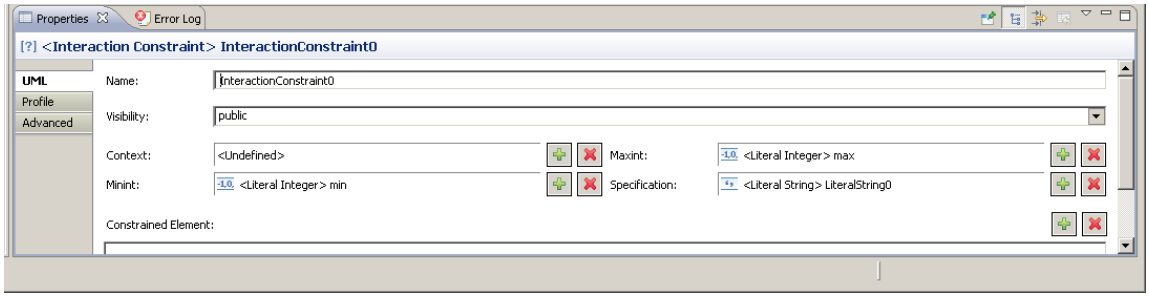
Min and max values are used for Loop , the specification is used as a constraint applied to the combined fragment. The sepcification can be created directly from the guard in the Model Explorer with a right click (using contexte menu).

6. Reference attributes from the guard properties view

Select elements on pop-up after clicking on green “+” sign.







Remark: the Specification property is set directly from the model explorer with the contextual menu. We temporarily use a string to set the constraint.