EclipseCon EU 2016 - Tutorial Guide

Sirius Concepts

During this tutorial, you will mainly use these Sirius concepts:

- **Viewpoint Specification Project**
  - The Eclipse project that defines a Sirius modeling tool
  - Contains a **odesign** file that describes the representations and Java services used by the tool
- **Viewpoint**
  - A viewpoint defines Sirius representations (diagrams, tables, matrices, trees) dedicated to a specific need
- **Diagram Description**
  - Describes a kind of graphical representation for your model
  - It defines which elements to display on the diagram, how (style) and the tools to edit them
- **Node**
  - Describes model elements displayed via an image or a simple shape
  - It defines how to find the model elements to display
  - It defines a graphical style (shape, label, color, ...)
- **Relation Based Edge**
  - Describes the relation between two objects
  - The relation can be computed
  - It defines graphical style (color, routing style, ...)
- **User Colors Palette**
  - Defines specific colors that you can use for the diagram elements (background, foreground, border, ...)
- **Container**
  - Describes graphical model elements that can contain other elements (as free form, lists or compartments)
  - Defines how to find the container elements to display and their sub-elements (nodes, containers, ports)
  - Defines a graphical style (shape, label, color, ...)
- **Double-Click tool**
  - Defines which actions to execute when the user double-click on a graphical element
- **Direct Edit Label tool**
  - Defines how to interpret the label value changes that are made directly on the diagram
- **Reconnect Edge tool**
  - Defines how to interpret the modification of edges ends (origin or destination) made directly from the diagram
  - Provides three variables :
    - element (the origin of the edge)
    - source (the destination of the edge before the execution of the tool)
    - target (the destination of the edge after the execution of the tool)
- **Properties Views Description**
  - Describes how model element properties are edited in the Eclipse Properties Views

Sirius expressions syntaxes

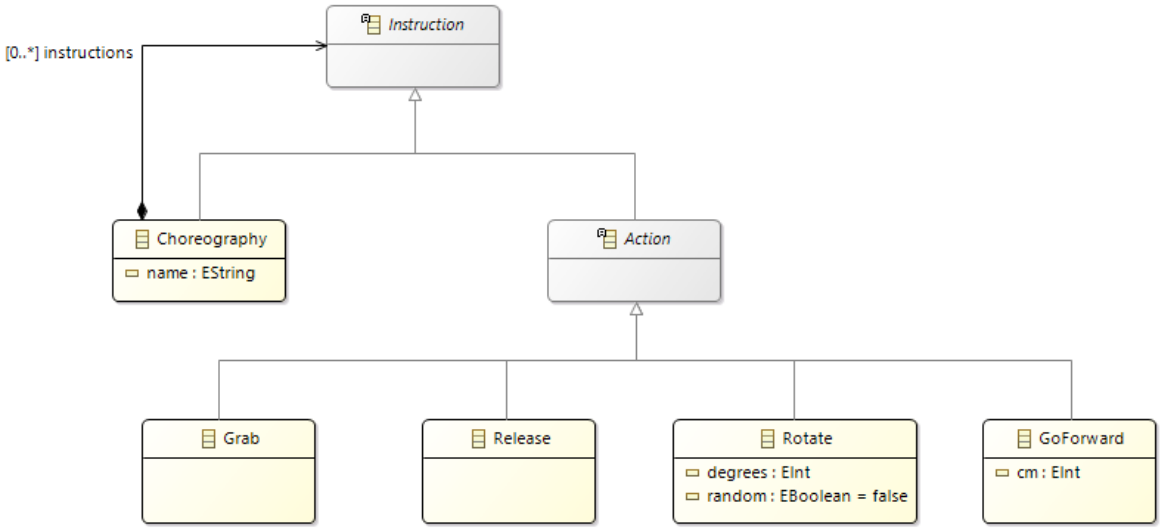
Dynamic parts of a modeling tool created with Sirius require you to write expressions that will be evaluated at runtime. Some of these expressions return model elements while others simply produce text.

Sirius proposes four main syntaxes to write these queries:

- **var:**
  - allows Sirius to evaluate a variable
  - examples:
    - `var:self`
    - `var:container`
- **feature:**
  - allows Sirius to evaluate an EMF feature (property or reference) on the current context
  - examples:
    - `feature:name`
    - `feature:instructions`
- **service:**
  - allows Sirius to evaluate a Java method defined in a Class declared as an extension
  - examples:
    - `service:getNextInstruction()`
    - `service:setNextInstruction(i)`
- **aql:**
  - allows Sirius to evaluate an expression written in AQL (Acceleo Query Language)
    - https://www.eclipse.org/acceleo/documentation/aql.html
  - examples:
    - `aql:self.instructions->at(1)`
    - `aql:self.oclIsKindOf(mindstorms::Rotate) and self.degrees >= 0`

Metamodel

For this tutorial we use this metamodel:



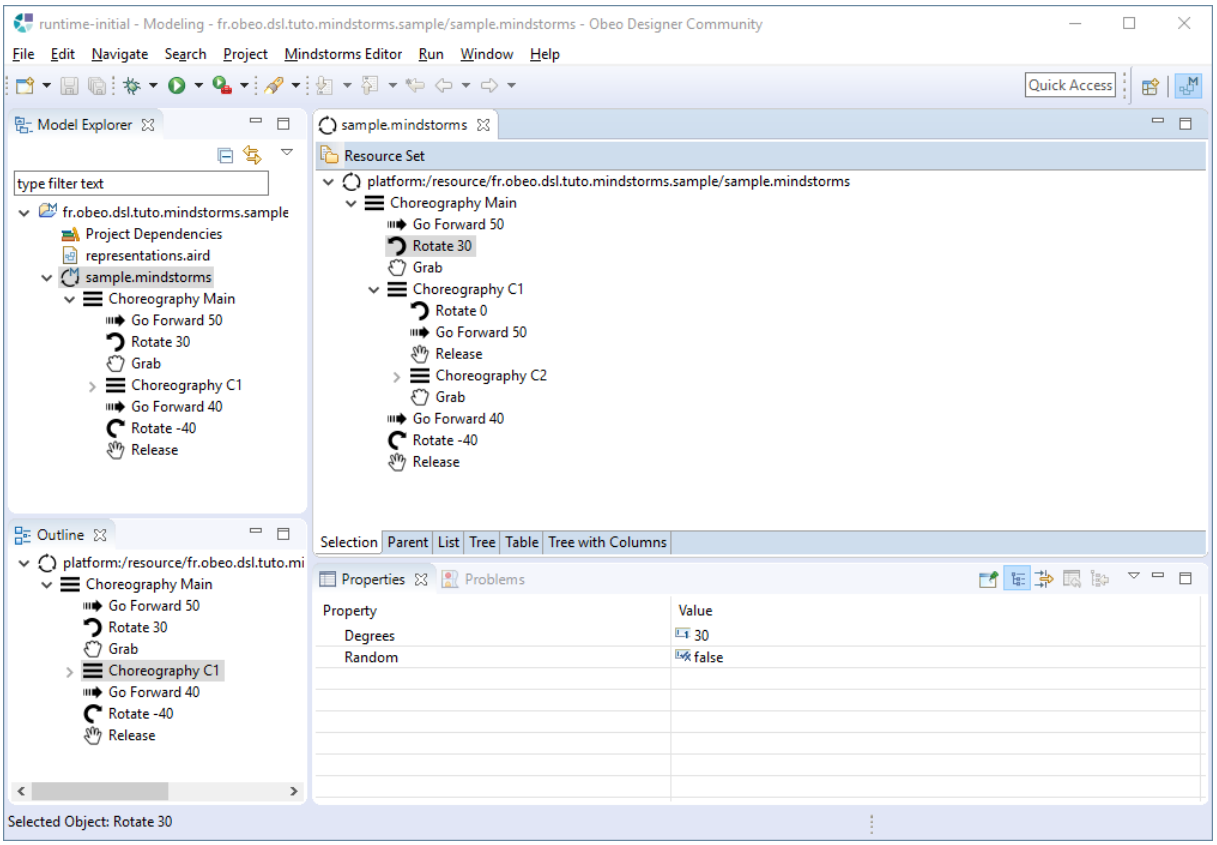
Preparation

Objectives

- Prepare the environment for the tutorial

Instructions

- Launch **Obeo Designer**
- Select the **Sirius** perspective
- Import the three Eclipse projects contained in the archive **initial.zip**
  - They define the **Mindstorms** metamodel
- Create and launch a new **Eclipse Launch Configuration**
  - In this new runtime, the **Mindstorms** metamodel will be available for execution
- In the new runtime, import the Eclipse project contained in the archive **sample.zip**
  - It contains a sample **Mindstorms** model that will be used to test your modeling tool
  - You can open this model with the default editor generated by EMF
- Your environment should look like this:



Step1: Basic Visualization Tool

Objectives

Create a basic Diagram to display the instructions of a Mindstorms choreography:

- **Nodes**
  - Start of the choreography (the choreography itself)
  - Grab actions
  - Release actions
  - GoForward actions
  - Rotate actions
  - Sub-choregraphies
- **Edges**
  - Link from the start and the first instruction of the choreography
  - Link between an instruction and its next instruction

Instructions

Note: all the following actions have to be performed in the Eclipse **runtime** launched previously

- Create a **Viewpoint Specification Project** named `fr.obeo.dsl.tuto.mindstorms.design`
  - *Viewpoint Specification Model name*: `mindstorms.odesign`
- Import the Archive File **icons-modeler.zip** into this project

Note: all the next actions have to be performed in the **.odesign** file (except the creation of Java services)

- Update the **Viewpoint** created by default (**%viewpointName**)
  - *Id* = `Mindstorms`
  - Remove the *Label* value
  - *Model File Extension* = `mindstorms`
- In this viewpoint, create a representation of type **Diagram Description**
  - *Name* = **`Choreography Diagram`**
  - *Domain Class* = **`mindstorms.Choreography`**
  - Reference the **mindstorms** metamodel
    - in *metamodels* tab, add the **mindstorms package** from the registry

Note: from now, you should be able to create a blank **Choreography Diagram** on the sample model:
- Activate the **Mindstorms** viewpoint from the sample project (right-click on the project + menu **Viewpoint Selection**)
- Create a diagram from the sample model (right-click on the root Choreography in the Model Explorer and select "**New representation**")

- Create a **Composite Layout** to force linked objects to be displayed from left to right
  - *Direction* = `Left To Right`
- In the **Default Layer,** create a **Node** representing the current Choreography

  - *Name* = `CD_Start`
  - *Domain class* = `mindstorms.Choregarphy`
  - *Semantic Candidate Expression* = `var:self`
  - Create a style **Workspace Image** for this Node
    - *Image Path* = `Start.svg` (prefixed by its path)
    - *Label Size* = `12`
    - *Label Format* = `Bold`
    - *Show Icon* = `false`
    - *Label Expression* = `Start`
    - *Label Position* = `border`

- Create a **Node** to display the Instructions of the current choreography
  - *Name* = `CD_Instruction`
  - *Domain class* = `mindstorms.Instruction`
  - *Semantic Candidate Expression* =
    - `feature:instructions`
  - Create a default **Workspace Image** for this Node
    - *Image Path* = `Instruction.svg` (prefixed by its path)
    - *Label Size* = `10`
    - *Show Icon* = `true`
    - Remove the value of *Label Expression*
    - *Label Position* = `node`

- On the Default layer, create a **Style Customizations** rubric to set specific labels for some kinds of Instructions
  - Create a **Style Customization** for **Choreography**
    - *Predicate Expression* = `aql:self.oclIsKindOf(mindstorms::Choreography)`
    - Create a **Property Customization Expression (by expression)**
      - Applied On = `Instruction.svg`
      - *Property Name* = `labelExpression`
      - *Value Expression* = `feature:name`

  - Copy / Paste the previous Style Customization and adapt it for **GoForward**
    - *Predicate Expression* = `aql:self.oclIsKindOf(mindstorms::GoForward)`
    - Update the **Property Customization Expression (by expression)**
      - *Value Expression* = `aql:self.cm + ' cm'`

  - Copy / Paste the previous Style Customization an adapt it for **Rotate**
    - *Predicate Expression* = `aql:self.oclIsKindOf(mindstorms::Rotate)`
    - Update the **Property Customization Expression (by expression)**
      - *Value Expression* = `aql:if (self.random) then '?' else self.degrees + '°' endif`

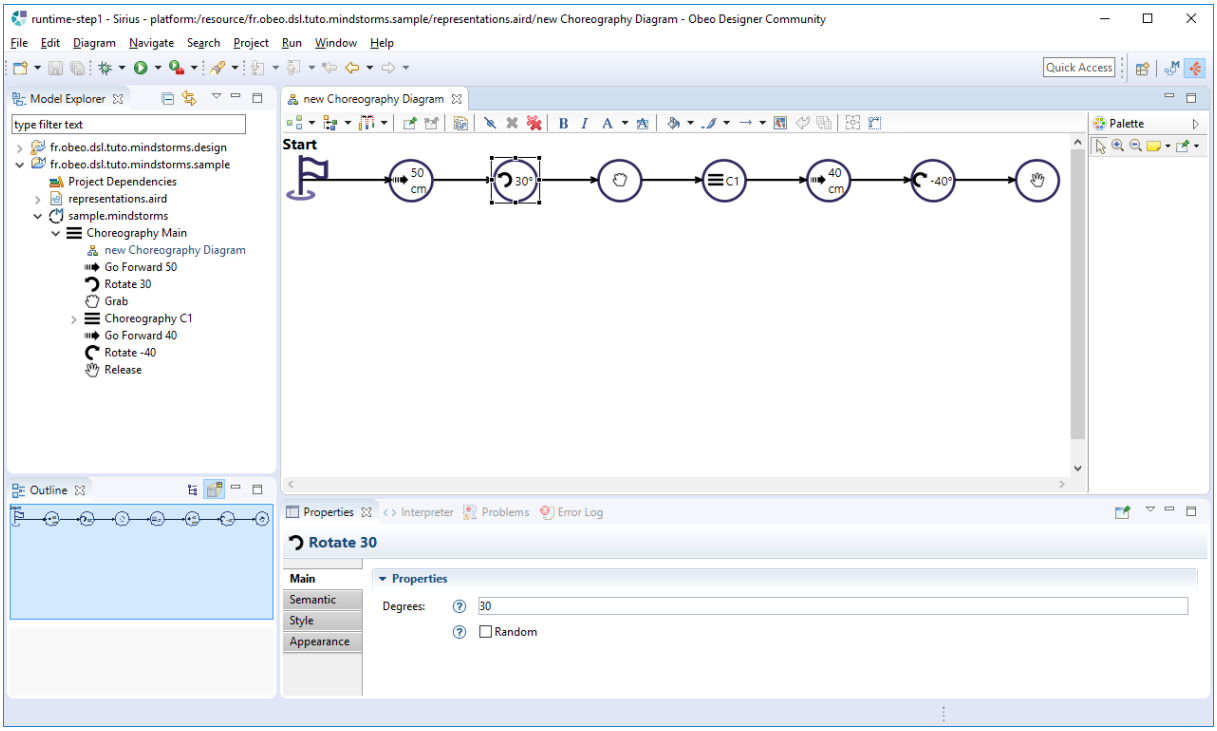
- Create a **Relation Based Edge** named `CD_First` to display a link between the **Start** node and the node representing the **first** Instruction
  - *Target Finder Expression* = `aql:self.instructions->at(1)`
  - *Change the style*
    - *Routing Style* = `Manhattan`
    - *Stroke Color* = `black`

- Define a service that computes the next instruction of a given instruction.
    - Declare the EMF mindstorms metamodel in the **META-INF/MANIFEST.MF** file
        - In the **Dependencies** add `fr.obeo.dsl.tuto.mindstorms`
    - Copy the source code of the **Method** named **`getNextInstruction`** from the file **methods-step1.txt** into the class **Services.java**

```
public Instruction getNextInstruction(Instruction instruction) {
	Choreography
parentChoreography=(Choreography)instruction.eContainer();
	List<Instruction> actions=parentChoreography.getInstructions();
	int position=actions.indexOf(instruction);
	if (position==actions.size()-1) {
		return null;
	}
	else {
		return actions.get(position+1);
	}
}
```

- Create a **Relation Based Edge** named `CD_Next` that displays the links between an Instruction and its next Instruction
    - *Target Finder Expression* = **`service:getNextInstruction()`**

- The diagram on the sample model should look like this:



Step 2: Nicer Visualization Tool

Objectives

Improve the graphical rendering of instructions:

- **Specific SVG images for actions**
  - Full size image
  - Label on the border
  - For Rotate : an image depending on the rotation direction (left or right)
- **Containers for sub-choregraphies**
  - Specific colors for background and border
  - Sub-instructions displayed as a list

Instructions

- Set specific images for each kind of **Action**
  - Update the Instruction.svg Workspace Image (defined previously)
    - *Show Icon* = `false`
    - *Label Size* = `12`
    - *Label Format* = `Bold`
    - *Label Position* = `border`
  - Extend the **GoForward**'s Style Customization
    - Create a **Property Customization Expression (by expression)**
      - Applied On = `Instruction.svg` (prefixed by its path)
      - *Property Name* **=** `workspacePath`
      - *Value Expression* = **`GoForward.svg`**
  - Create two Style Customizations for **Grab** and **Release**
  - Create a specific Style Customizations for **Rotations** to right
    - *Predicate Expression* = `aql:self.oclIsKindOf(mindstorms::Rotate) and self.degrees < 0`
  - Copy/Paste and adapt the GoForward's **Property Customization** Expression for **Grab, Release and Rotate** (left and right)

- Update the node previously defined for Instructions (CD_Instruction) to restrict this node to Actions (Choregraphies are going to be displayed with a dedicated Container)
  - Domain Class = `mindstorms.Action`

> Note: to improve the performances, you could also update the *Semantic candidate Expression* with this new expression `aql:self.instructions->filter(Mindstorms::Action)`

- At the .odesign root, create a **User Colors Palette** to define specific colors for **Choregraphies**
  - Create a **Used Fixed Color** named `MindstormColor1` (R=`186`, G=`223`, B=`225`)
  - Create a **Used Fixed Color** named `MindstormColor2` (R=`0`, G=`119`, B=`136`)

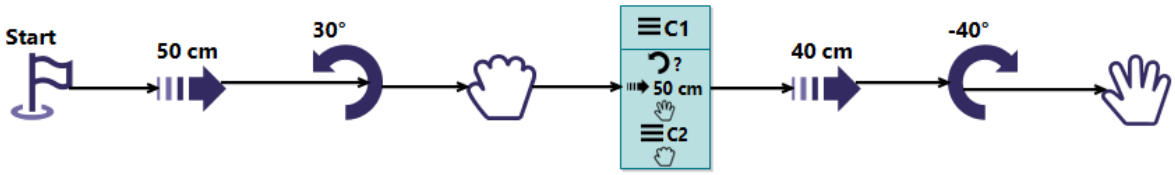
- In the **Choreography Diagram**, create a **Container** `CD_SubChoreography` to represent the instances of **Choreography**
  - *Domain Class* = `mindstorms.Choreography`
  - *Semantic Candidate Expression* = `feature:instructions`
  - *Children Presentation* = `List`
  - Create a **Style** of type **Gradient**
    - *Label Size* = `12`
    - *Label Format* = `Bold`
    - *Background Color* = `MindstormColor1`
    - *Foreground Color* = `MindstormColor1`
    - *Border Color* = `MindstormColor2`

  - Create a **Sub Node** named `CD_SubInstruction`
    - *Domain class* = `mindstorms.Instruction`
    - *Semantic Candidate Expression* = `feature:instructions`
    - Define a default style (any style with a label, for example Square)
      - *Show Icon* = `true`
      - *Label Size* = `10`
      - *Label Format* = `Bold`
      - Remove the *Label Expression*

  - Update all the Property customizations related to the labelExpression (defined previously), to also apply to the sub-instructions of the container which have a label (Choreography, GoForward Rotate and RotateRight)
    - Add the style defined for `CD_SubInstructions` into each *Applied On* property

  - Update the edges CD_First and CD_Next to take the container CD_Choreography as a potential source or target

- The diagram on the sample model should look like this:



Step 3: Tools

Objective

Create tools to create, modify or navigate the model elements directly from the diagram:

- **Node creation**
  - Create a Grab action
  - Create a Release action
  - Create GoForward action (initialized to 50 cm)
  - Create a Rotate to left action (initialized to 90°)
  - Create a Rotate to left action (initialized to -90°)
  - Create a Choreography (initialized to NewChoreography)
- **Redirect**
  - Change the destination of the first edge
  - Change the destination of the other edges
- **Navigation**
  - Double-click on a Choreography to create/open a new diagram
- **Direct edit**
  - Change the properties of objects from their label

Instructions

- In the Default layer, create a **Section** named `New` to provide a palette for the creation of objects

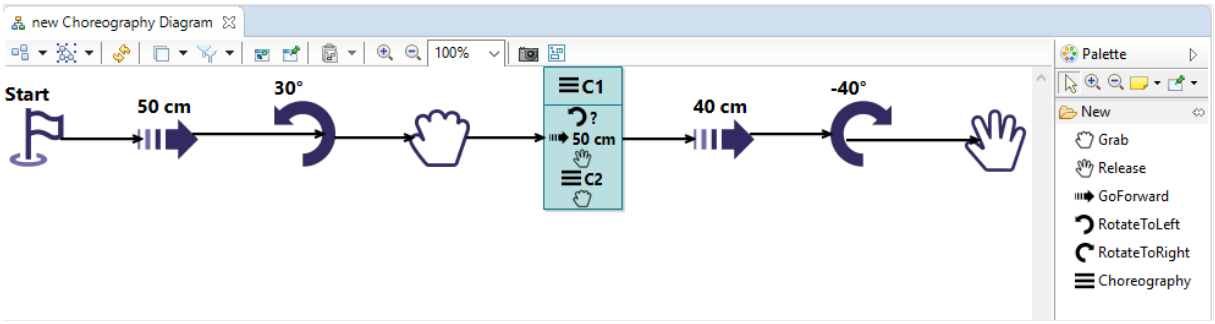
  - Create a **Node Creation** to create instances of **Grab**
    - *Node Mapping* = `CD_Instruction`
    - *Icon Path* = `Grab_16px.png` (prefixed by its path)
    - To allow the user to also create a Grab into a sub-Choreography add `CD_SubChoreography` in the *Extra Mappings* property
    - Under the **Begin** node, create a **Change Context** to set on which objects the instructions will be executed
      - *Browse Expression* = `var:container` (this is the current Flow)
    - Add a **Create Instance**
      - *Reference Name* = `instructions`
      - *Type Name* = `mindstorms.Grab`

  - Copy/Paste and adapt the previous tool for **Release**, **GoForward**, **Rotate to Left** and **Rotate to Right**

  - Create a **Container Creation** for **Choreography**

  - After the Create Instance, add a **Set** operation for:
    - **GoForward**: *Feature Name* = `cm` and *Value Expression* = `50`
    - **Rotate to left**: *Feature Name* = `degrees` and *Value Expression* = `90`
    - **Rotate to right**: *Feature Name* = `degrees` and *Value Expression* = `-90`
    - **Choreography**: *Feature Name* = `name` and *Value Expression* = `NewChoreography`

- The **New** Palette shoul look like this:



- In the Layer, create a **Section** named `Edit` for the other tools not visible in the palette
  - Copy/paste three new methods from the file **methods-step3.txt** into Services.java
    - `setFirstInstruction`
    - `setNextInstruction`
    - `editRotateLabel`
  - Create a **Reconnect Edge** to allow the user to change the end of the first edge
    - Mappings = `CD_First`
    - Under the **Begin** node, create a **Change Context** to execute a service
      - *Browse Expression* = `aql:element.setFirstInstruction(target)`

> The Reconnect Edge tool provides three important variables:
> - *Element* = the object attached to the edge's start
> - *Source* = the initial object at the edge's end (before the user moves this end)
> - *Target* = the new object at the edge's end (after the user moved this end)

  - Create a **Reconnect Edge** to allow the user to change the end of the other edge
    - *Mappings* = `CD_Next`
    - Under the **Begin** node, create a **Change Context** to execute a service
      - *Browse Expression* = `aql:element.setNextInstruction(target)`
  - Create a **Double-Click** for **Choreography** to navigate to its detailed diagram
    - Mapping = `CD_SubChoreography`
    - After the Begin, create a **Navigation** to **Choreography Diagram**
      - *Create if not Existent* = **`true`**
  - Create a unique **Direct Edit Label** for **Rotate, GoForward** and **Choreography** instances
    - *Mappings* = **`CD_Instruction, CD_SubChoreography, CD_SubInstruction`**
    - Under the **Begin** node, create a **Switch** with three **Cases**
      - ***`aql:self.oclIsKindOf(mindstorms::Choreography)`***
        - Create a **Set** operation for **name**
          - *Value Expression* = ***`var:arg0`***
      - ***`aql:self.oclIsKindOf(mindstorms::GoForward)`***
        - Create a **Set** operation for **cm**
          - *Value Expression* = ***`var:arg0`***
      - ***`aql:self.oclIsKindOf(mindstorms::Rotate)`***
        - Create a **Change Context**
          - Value Expression = ***`service:editRotateLabel(arg0)`***

Step 4: Custom Properties Views

Objectives

Replace the default properties views (dynamically generated by Sirius) by custom ones:

- **Rotate**
  - Degrees field disabled if Random is checked
- **Choreography**
  - Editable list of Instructions
  - Error if Name is used by another Choreography
- **GoForward**
  - Specific background color if Cm value is negative
  - Warning if Cm value is null

Instructions

- At the root of the modeler definition create a **Properties View Description** to define custom properties views for the instructions
  - Update the first **Page** (created by default)
    - *Domain Class* = `mindstorms.Instruction`
    - *Label Expression* = `General`
  - Create a **Group** to display and edit the **cm** property of **GoForward**
    - Add this new **Group** to the first **Page**
    - *Domain Class* = `mindstorms.GoForward`
    - *Label Expression* = `Properties`
    - Add a **Text** for the **name** property
      - *Label Expression* = `Cm`
      - *Value Expression* = `feature:cm`
      - Add a **Begin** and a **Set** operation (set `var:newValue` to `cm`)
    - Create a **Conditional Style** for the **cm** Text in order to color the text background when cm is lower than 0
      - *Precondition Expression* = `aql:self.cm<0`
      - Create a **Style** with *Background Color* = `MindstormsColor1`
  - Create a **Group** to display and edit the **degrees** and **random** properties of **Rotate**
    - Add this new **Group** to the first **Page**
    - *Domain Class* = `mindstorms.Rotate`
    - *Label Expression* = `Properties`
    - Add a **Text** for the **degrees** property
      - *Label Expression* = `Degrees`
      - *Value Expression* = `aql:self.degrees`
      - *Is Enabled Expression* = `aql:not self.random`
      - Add a **Begin** and a **Set** operation (set `var:newValue` to `degrees`)
    - Add a **Checkbox** for the **random** property
      - *Label Expression* = `Random`

- *Value Expression* = `aql:self.random`
- Add a **Begin** and a **Set** operation (set `var:newValue` to `random`)

- Add a **Group Validations** to warn the user when *degrees* is null and *random* is false (useless rotate instruction).
  - Create a **Semantic Validation Rule**
    - *Id* = `UselessRotation`
    - *Level* = `Warning`
    - *Message* = `This rotation is useless`
    - Create an **Audit**
      - *Audit Expression* = `aql:self.degrees<>0 or self.random`
    - Create a **Fix** that sets the random property to `true`

- Create a **Group** to display and edit the **name** and **instructions** properties of **Choreography**
  - Add this new **Group** to the first **Page**
  - *Domain Class* = `mindstorms.Choreography`
  - *Label Expression* = `Properties`
  - Add a **Text** for the **name** property
    - *Label Expression* = `Name`
    - *Value Expression* = `feature:name`
    - Add a **Begin** and a **Set** operation (set `var:newValue` to `name`)

  - Add a **Reference** for the **instructions** property
    - *Label Expression* = `Instructions`
    - *Reference Owner Expression* = `var:self`
    - *Reference Name Expression* = `instructions`

  - Add a **Group Validations** to warn the user when the **name** is already used by a sibling choreography.
    - Create a **Property Validation Rule**
      - *Targets* = `Text Name`
      - *Id* = `UniqueName`
      - *Level* = `Error`
      - *Message* = `Name must be unique`
      - Create an **Audit**
        - *Audit Expression* =
```
aql:not self.siblings()
->filter(mindstorms::Choreography)
->excluding(self)
->collect(i|i.name)
->includes(self.name)
```