Scenario tutorial day two

Hands-on Sirius, the logo turtle language

This document indicates the steps to follow in order to reproduce the tutorial.

The result of this tutorial is available in the folder part2-mmfirst-solution in githug repo or as a download in the following zip.

Installation

- If you haven't followed the previous tutorials [tutorial_scenario_part1.asciidoc] or [tutorial_scenario_part1postprocess.asciidoc]:
  - Install an Oracle Java JDK (minimum 8, max 10, Oracle is preferred due to support of openFX used for the tutorial part 3)
  - grab and unzip the latest dev version of GEMOC Studio http://download.eclipse.org/gemoc/packages/nightly/
- If you have followed one the previous tutorial [tutorial_scenario_part1.asciidoc] or [tutorial_scenario_part1postprocess.asciidoc]:
  - Either use a brand new workspace or clean it by deleting the projects in it since the projects we will create will have the same names.

Then

- download the file **https://dvojtise.github.io/mde-crashcourse-logo/zips/part2-mmfirst-base.zip**
- in Eclipse,
  - *File → Import... → General → Existing projects into Workspace → Next*
  - *Select archive file → Browse* and select the file *part2-mmfirst-base.zip* you've downloaded
  - Finish

Sirius graphical editor

We will create a block based graphical representation for Logo.

Create the project for the graphical editor:

- *File → new → Viewpoint Specification Project*
  - name: `fr.inria.sed.logo.design`
- on the project (plugin.xml or manifest.mf) add a dependency to *fr.inria.sed.logo.model* project
- open the odesign
  - rename the viewpoint from *MyViewpoint* to *LogoBlockViewpoint*

Let's open some example using our representation. This can be done in the **Modeling Workbench** like xtext directly on .logo files.

Most parts of Sirius are interpreted, a big part of the diagram specification can be done directly in the **Language Worbench** and changes in the diagram specification are directly reported to the opened model. This greatly simplifies the design of the diagram editor.

However, since xtext does not work this way we need to convert our .logo files into .xmi that don't require xtext.

You can directly create an xmi test file by :

- open the ecore file → Select the LogoProgram class → right click → Create dynamic instance

This is the recommended way if you do not have an xtext representation.

In some situation you may wish to convert an xtext represensation into xmi. IE. convert a logo file into an xmi file. To do that:

- in the **Modeling workbench** (with xtext support available…)
- right click on the *.logo* file → Open with → Other → Sample Reflective Ecore Model Editor
- *File* → *save as* → choose a name ending with **.xmi**
- verify that the file is correctly encoded in xmi (ie. a xml flavor) by opening it with the generic text editor

- change the header in order to be [source>

```
<logo:LogoProgram xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:logo="http://www.inria.fr/sed/logo/Logo"
    xsi:schemaLocation="http://www.inria.fr/sed/logo/Logo
../fr.inria.sed.logo.model/model/Logo.ecore">
```

copy or import the file in some test project in the **Language Workbench**

For best result if working with both Sirius and Xtext. I recommend to open (import the project) the project containing the *design* file in both the Language workbench AND Modeling workbench.

Using this technique you can directly add a Sirius representation on top of the *.logo* files without converting them in xmi.

NOTE when using the second workbench, make sure to create the representation using the correct viewpoint, since it will appear twice.

For our logo example, we'll mostly design the graphical representation from the **modeling workbench**.

In the odesign:

- on the viewpoint; *right click → new representation → diagram description*
- on the diagram description;
  - on the metamodel tab: add a reference to the ecore file (*add from registry* if you work with xtext and are working in the **modeling workbench**, otherwise use *add from workspace*)
  - on the general tab:
    - Domain class = LogoProgram (the completion should work)
    - give an ID = LogoBlockDiagram (change the label for "Logo Block Diagram")
    - tick "Initialization" and "show on startup"

Create a test model with it representation

Create a test project and copy one or several *.logo* files for testing the representation.

Right click on the *.logo* file → New → Representation file

This allows to create one file containing the representations (ie. the diagrams) for the given *.logo* file. These representations will be contained in an *.aird* file.

Sirius support another mode for the diagram using a *project session*:

When creating the project you can use the *Modeling project* wizard. Projects with this nature do not require to create manually the *.aird* file because it will create one by default for the project.

However in this case, all representations of all models in the current project will be contained in a single "representation.aird" file. While being convenient for some purposes, this behavior may not be suitable for all cases.

Display all root instructions:

- _New diagram element → Node then in the properties view
  - Id: PrimitiveInstructionNode
  - domain class: logo::PrimitiveInstruction (you can try with Instruction but you'll probably have to change it later ;-))
  - semantic candidate expression: `aql:self.eContents()` then use this alternative to reject some kinds : `aql:self.eContents()->reject(x | x.oclIsKindOf(logo::ProcDeclaration))`
  - *New style → Square*
  - Label tab: Label expression: `aql:self.eClass().name` (for a start, will be improved later)
  - advanced tab: size computation expression: `aql:self.eClass().name.size()`

If you have nice default icons defined in the *.edit* project, they'll be directly displayed.

I recommend to use explicit names as IDs in Sirius. I usually start by the represented model element (ie. metaclass name) followed by the kind of representation (Container, Node, or edge) using camel case text.

Display all root instructions:

- *New diagram element → Node* then in the properties view
  - Id: PrimitiveInstructionNode
  - domain class: logo::PrimitiveInstruction (you can try with Instruction but you'll probably have to change it later ;-))
  - semantic candidate expression: `aql:self.eContents()` (alternative to reject some kinds : `aql:self.eContents()->reject(x | x.oclIsKindOf(logo::ProcDeclaration))`
  - *New style → Dot*
  - Label tab:
    - Label expression: `aql:self.eClass().name` (for a start, will be improved later)
    - Label position: border
  - Advanced tab:
    - allow resizing : uncheck
    - size computation expression: 1

you can try with a more generic type such as *Instruction* and then reject some elements using a query such as: `aql:self.eContents()->reject(x | x.oclIsKindOf(logo::ProcDeclaration))`

However, this will not fit our final design. and using the *PrimitiveInstruction* and *ControlStructureInstruction* structure of the metamodel allow to factorize some representation rules for each group.

Reference documentation for writing queries https://www.eclipse.org/sirius/doc/specifier/general/Writing_Queries.html https://www.eclipse.org/acceleo/documentation/aql.html https://www.eclipse.org/acceleo/documentation/

Display all instructions of the procedure declaration:

We will indicates to the ProcedureDeclaration container that we want to reuse some display rules.

- On the procDeclNode,
  - *Import tab*, Reused Node Mapping: *PrimitiveInstructionNode*

Add a link representing the sequence of instructions

- *New diagram element → Relation based Edge* then in the properties view
  - Id: instructionSequenceEdge
  - source mapping: InstructionNode
  - target mapping: InstructionNode
  - Target finder expression:

```
aql:let i = self.eInverse('instructions').instructions->asSequence() in i->at(i->indexOf(self)+1)
```

Add a link between procedure call and the procedure declaration:

- *New diagram element → Relation based Edge* then in the properties view
  - Id: procCallEdge
  - source mapping: InstructionNode
  - target mapping: procDeclNode
  - Target finder expression: `aql:if self.oclIsKindOf(logo::ProcCall) then self.oclAsType(logo::ProcCall).declaration else null endif`
  - make this link use dashed line

```
https://www.eclipse.org/sirius/doc/specifier/general/Writing_Queries.html
```

```
You can test your queries in order to write them: use the "Acceleo Model to Text > Interpreter" view
then switch to "Sirius" mode instead of "Acceleo" mode.
```

```
Warning: When using the Interpreter view from an element selected in a Sirius representation, the context of the expression is not the semantic element, but the view model element used internally by Sirius.
```

```
In the interpreter view, to get the semantic element, you must enter _aql:self.target_
```

Move procedure call - procedure declaration link into a separate layer

on the Logo Block Diagram

- *New diagram element → additional layer* then in the properties view
  - Id: ProcedureCall

move procCallEdge to this layer

In the diagram, observe how to enable/disable the layer.

Add a default layout

on the Logo Block Diagram

- *New layout → Composite layout* then in the properties view
  - Padding: 20
  - top to bottom

Create representation for If

- *New diagram element → Node* then in the properties view
  - Id: IfNode
  - domain class: logo::If
  - semantic candidate expression: `aql:self.eContents()`
  - *New style → Diamond*
  - Label tab:
    - Label expression: `aql:self.eClass().name` (for a start, will be improved later)
    - Label position: border
  - Advanced tab:
    - allow resizing : uncheck
    - size computation expression: 3
- *New diagram element → Container* then in the properties view
  - Id: thenPartContainer
  - domain class: logo::Block
  - semantic candidate expression: `aql: self.eContents()->filter(logo::If)->collect(i | i.thenPart))`
  - _New style → Gradient
  - Label tab:

        - Label expression: `aql:'then'`
    - Color tab
        - Foreground color: light_green
- *New diagram element → Container* then in the properties view
    - Id: elsePartContainer
    - domain class: logo::Block
    - semantic candidate expression: `aql: self.eContents()->filter(logo::If)->collect(i | i.elsePart))`
    - _New style → gradient
    - Label tab:
        - Label expression: `aql:'else'`
    - Color tab
        - Foreground color: light_red

in the following containers: procDeclContainer, thenPartContainer, and elsePartContainer;

- Import tab:
    - Reused Node Mapping: PrimitiveIntrustionNode, IfNode
    - Reused Container Mapping: elsePartContainer, thenPartContainer
- *New diagram element → Relation based Edge* then in the properties view
    - Id: IfThenEdge
    - source mapping: IfNode
    - target mapping: thenPartContainer
    - semantic candidate expression: `aql: self.thenPart`
- *New diagram element → Relation based Edge* then in the properties view
    - Id: IfElseEdge
    - source mapping: IfNode
    - target mapping: elsePartContainer
    - semantic candidate expression: `aql: self.elsePart`
- *New diagram element → Relation based Edge* then in the properties view
    - Id: EndIfSequenceEdge
    - source mapping: thenPartContainer, elsePartContainer
    - target mapping: PrimitiveInstructionNode, IfNode
    - semantic candidate expression:

```
aql:let i = self.eContainer().eInverse('instructions').instructions->asSequence() in
i->at(i->indexOf(self.eContainer())+1)
```

Exercise for the motivated: reproduce similar structure for Repeat and While control structure

Improve labels and xtext integration

We will create some java services to be used by sirius

Add xtext aware service static methods

close the **modeling worbench** (will need to be restarted in order to take into account the new methods)

in the **Language workbench**.

in the *xxx.design* project open plugin.xml file, add a plugin dependency to *org.eclipse.xtext*, *org.eclipse.ui.ide*, *org.eclipse.ui.workbench.texteditor*, and *org.eclipse.ui.workbench*.

copy the file InfoPopUp.java in the package next to the Services.java class.

add the following methods in the Services.java file. (or copy the file from Services.java).

```
    /**
     * Try to retrieve an xtext resource for the given element and then get its String
representation
     * @param any EObject
     * @return the xtext representation of the EObject or an empty string
     */
    public String xtextPrettyPrint(EObject any) {
        if (any != null && any.eResource() instanceof XtextResource && any.
eResource().getURI() != null) {
            String fileURI = any.eResource().getURI().toPlatformString(true);
            IFile workspaceFile = ResourcesPlugin.getWorkspace().getRoot().getFile(new
Path(fileURI));
            if (workspaceFile != null) {
                ICompositeNode node = NodeModelUtils.findActualNodeFor(any);
                if (node != null) {
                    return node.getText().trim();
                }
            }
        }
        return "";
    }

    public EObject openTextEditor(EObject any) {
        if (any != null && any.eResource() instanceof XtextResource && any.
eResource().getURI() != null) {

            String fileURI = any.eResource().getURI().toPlatformString(true);
```

```
            IFile workspaceFile = ResourcesPlugin.getWorkspace().getRoot().getFile(new
Path(fileURI));
            if (workspaceFile != null) {
                IWorkbenchPage page = PlatformUI.getWorkbench
().getActiveWorkbenchWindow().getActivePage();
                try {
                    IEditorPart openEditor = IDE.openEditor(page, workspaceFile,
                            "fr.inria.sed.logo.xtext.Logo", true);
                    if (openEditor instanceof AbstractTextEditor) {
                        ICompositeNode node = NodeModelUtils.findActualNodeFor(any);
                        if (node != null) {
                            int offset = node.getOffset();
                            int length = node.getTotalEndOffset() - offset;
                            ((AbstractTextEditor) openEditor).selectAndReveal(offset,
length);
                        }
                    }
                    // editorInput.
                } catch (PartInitException e) {
                    Activator.error(e.getMessage(), e);
                }
            }
        }
        System.out.println(any);
        return any;
    }

    public EObject openBasicHoveringDialog(EObject any) {
        String xtextString = xtextPrettyPrint(any);
        if (xtextString != null && !xtextString.isEmpty()) {
            IEditorPart part = PlatformUI.getWorkbench().getActiveWorkbenchWindow
().getActivePage().getActiveEditor();
            InfoPopUp pop = new InfoPopUp( part.getSite().getShell() , "Textual
representation of the element","press ESC to close");
            pop.setText(xtextString);
            pop.open();
        }
        return any;
    }
```

Use services to improve labels

restart the **modeling workbench**

If you start it in debug mode, small changes (code in an existing method) can be taken into account without a full restart.

On the IfNode

- Label tab
  - Label expression: `aql:self.condition.xtextPrettyPrint()`

On PrimitiveInstructionNode

- *New conditional style*
  - Predicate expression: [self.oclIsKindOf(logo::Left) or self.oclIsKindOf(logo::Right) /]
  - copy the style of the PrimitiveInstructionNode into this new conditional style
    - Label tab
    - Label expression :

```
aql:self.eClass().name+' '+self.angle.xtextPrettyPrint()
```

do the same for other types such as Forward, Backward, ProcCall ...

service calling xtextPrettyPrint() might be usefull too in the *tooltip expression* on the General tab of the styles.

Add actions that open xtext editor

- *new tool → Section*
  - Id: edition

Open xtext editor via right click popup

- *new menu → Popup menu*
  - Id: OpenInTextEditorPopUp
  - Icon: add an icon from your own (or get one from the solution)

in the Begin element:

- *new operation → change context*
  - browse expression: `service:self.openTextEditor()`

Add action that create elements (Palette)

- *new element creation → node creation*
  - Id: addPenUp (also change the label for a nicer name in the Paletter)
  - Node Mappings: PrimitiveInstructionNode

on Begin

- *new operation → change context*

- browse expression: `var:container`
  - *new operation → create instance*
    - reference name: *instructions*
    - Type name: *logo::PenUp*

Add Validation rule (error marker)

Sirius provide a way to define rules that'll report errors. (Markers)

It is useful for example when creating element in sirius may lead to models that cannot be serialized in xtext.

The validation rule can also contains quickfix actions.