Notes on creating an OSATE2 AADL annex plug-in with XTEXT

Technical Note

Ernesto Posse

Modelling and Analysis in Software Engineering
School of Computing – Queen's University
Kingston, Ontario, Canada

June 17, 2014

Abstract

These notes describe how to setup Eclipse to develop an OSATE2 plug-in to support an AADL annex sub-language with XTEXT.

Contents

1 Assumptions and conventions

These notes assume that the reader is familiar with JAVA, ECLIPSE, AADL, OSATE2, XTEXT and EMF/ECORE. It is recommended that the reader follows some ECLIPSE plug-in development tutorial, as well as the basic XTEXT tutorials found on http://www.eclipse.org/Xtext/documentation.html. For the AADL standard, see [1].

1.1 Conventions

1.1.1 Paths

In most of these notes we will use the Unix-style notation for paths, *e.g.*, `/home/user/somefolder`, which is common to Linux and MacOSX. Converting this path to Windows-like notation is usually straight-forward: replace every forward slash '/' with a backslash '\' and prepend the drive letter followed by a colon. So the path `/one/two/three` becomes the path `C:\one\two\three`.

In these notes we will use the path `/home/user/osate-dev` for the home path where we will install the OSATE2 development environment, the user workspace, the GIT repositories, and the OSATE2 runtime workspace. However the reader can any suitable paths instead of these.

1.1.2 Code samples and meta-variables

We will use `fixed size courier font` for code samples and paths, and *`italic courier font`* for meta-variables that should be replaced by concrete names or other strings specific to the user's project. In larger pieces of code we use the prefix "`sample`" as a generic name prefix for different entities such as class names or method names, to be substituted by the user's specific names.

2 Installation

Install Osate2 from sources. You can follow the instructions from the OSATE2 wiki website: https://wiki.sei.cmu.edu/aadl/index.php/Getting_Osate_2_sources but you might run into problems with XTEXT if you obtain it from the source provided there, hence the following is recommended instead:

1. Install ECLIPSE MODELING TOOLS (MDT) version 4.3 (KEPLER) or later. It can be obtained at https://www.eclipse.org/downloads/:

   (a) Download the apropriate archive (`eclipse-modeling-kepler-SR2-`*`platform`*`.zip` or `eclipse-modeling-kepler-SR2-`*`platform`*`.tar.gz`) where *`platform`* is the platform where you are going to install Eclipse, for example, for the standard 64 bit linux distribution *`platform`* is `linux-gtx-x86_64`, for the standard MacOSX 64 bit distribution, it is `macosx-cocoa-x86_64`, and for the standard Windows distribution it is `win64`, etc.

   (b) Extract this archive[1] in some folder, *e.g.*, `/home/user/osate-dev`. Once extracted it should have created a folder called "`eclipse`" (`/home/user/osate-dev/eclipse`) with an executable called "`eclipse`" inside it.[2]

   (c) Run the ECLIPSE executable. It will ask you for a workspace folder. For example, `/home/user/osate-dev/workspace`.

2. Install XTEXT:

   (a) Click on "Help ▷ Eclipe Marketplace..."

   (b) In the "Find" search box type "Xtext" and hit "Enter" or click on "Go".

   (c) The first entry should show the latest version of XTEXT (2.6.0 at the time of this writing). Click on "Install".

   (d) When it asks you to confirm the selected features, make sure that "Xtext" and "Xtext SDK" are ticked. Click on "Confirm>"

   (e) Select "I accept the terms of the licence agreement" and click "Finish".

   (f) It will pop up a security warning window, saying that you are installing software that contains unsigned content. Click on "OK".

   (g) It will pop up a window asking if you would like to restart now. Click on "Yes".

3. Install GOOGLE GUICE:

   (a) Click on "Help ▷ Install New Software..."

   (b) Click the "Add..." button.

   (c) In the Name field enter "Google Guice".

   (d) In the Location field enter:

   http://guice-plugin.googlecode.com/svn/trunk/eclipse-update-site/

[1] If you are on MacOSX, avoid using the built-in unarchiver facility because due to a bug, it will corrupt the executable when extracting it. You will have to use a third-party archive extractor instead.

[2] Called "`eclipse.exe`" under Windows.

(e) Click "OK".

(f) Under the table an entry "Guice" should appear with a checkbox. Tick the checkbox to select it.

(g) Click "Next" and "Next" again.

(h) Select "I accept the terms of the licence agreement" and click "Finish".

(i) It will pop up a security warning window, saying that you are installing software that contains unsigned content. Click on "OK".

(j) It will pop up a window asking if you would like to restart now. Click on "Yes".

4. Install the Graphical Editing Framework Zest Visualization Toolkit (GEF-ZVT):

(a) Click on "Help ▷ Install New Software..."

(b) Click the "Add..." button.

(c) In the Name field enter "GEF".

(d) In the Location field enter

http://download.eclipse.org/tools/gef/updates/releases/

(e) Click "OK".

(f) Under the table an entry "GEF (Graphical Editing Framework)" should appear with a checkbox and a right-pointing arrow. Click on the right-pointing arrow to unfold a list of sub-components.

(g) Select (tick) the box for "Graphical Editing Framework Zest Visualization Toolkit".

(h) Click "Next" and "Next" again.

(i) Select "I accept the terms of the licence agreement" and click "Finish".

(j) It will pop up a window asking if you would like to restart now. Click on "Yes".

5. Install SLF4J:

(a) Click on "Help ▷ Install New Software..."

(b) In the "Work with:" field, click the down-pointing arrow to unfold the list of update sites.

(c) Choose "Kepler - http://download.eclipse.org/releases/kepler".

(d) Look for "slf4j" either by typing it in the text-box field below the "Work with:" field, or by unfolding "General Purpose Tools". The feature's full name is "m2e - slf4j over logback logging (Optional)". Select (tick) it.

(e) Click "Next" and "Next" again.

(f) Select "I accept the terms of the licence agreement" and click "Finish".

(g) It will pop up a window asking if you would like to restart now. Click on "Yes".

6. Install the core OSATE2 plug-in sources:

(a) Once you restart ECLIPSE, if there is a "Welcome" tab in the ECLIPSE environment, close it.

(b) Select "File ▷ Import..."

(c) Unfold the option "Git"

(d) Select "Projects from Git" and click "Next".

(e) Select "Clone URI" and click "Next".

(f) Under "Location", in the "URI" field, enter "`https://github.com/osate/osate2-core.git`" and click "Next".

(g) Make sure that the baranch "master" is selected[3] and click "Next".

(h) Under "Destination", in the "Directory:" field enter a directory where you want to download and store your local copy of the OSATE2 source repositories. For example,

`/home/user/osate-dev/git/osate2-core`

(i) Click "Next".[4]

(j) When asked for the wizard for project import, choose "Import existing projects" and click "Next".

(k) In the following dialog box all projects should be selected. Click "Finish". In the "Package Explorer" view of your ECLIPSE workspace all core OSATE2 plugins should appear.

(l) IMPORTANT: under the plugin `org.osate.aadl2` there should be a file called

"`aadl2-nouml.genmodel`"

This file should be inside the "`model`" folder. This file is essential to be able to create XTEXT languages that have access to AADL meta-model elements. Unfortunately, in the latest version of OSATE2 at the time of this writing (June 14, 2014), this file was removed. You will have to ask the OSATE2 developers to either put the file back, or provide you with an alternative. However, since at the time of this writing there is no alternative, the rest of these notes assume that you have a copy of this file and when you create a plugin called `sampleplugin` you create a "prerequisites" folder "`prereqs`" inside the plugin, and a "`models`" folder inside `prereqs`, where you should store a copy of `aadl2-nouml.genmodel`, as well as copies of the files "`aadl2.ecore`" and "`aadl2.genmodel`" found under `org.osate.aadl2/model`. Summarizing, you should have in your plugin, the following files:

i. `sampleplugin/prereqs/models/aadl2.ecore`
ii. `sampleplugin/prereqs/models/aadl2.genmodel`
iii. `sampleplugin/prereqs/models/aadl2-nouml.genmodel`

7. Setup a Run configuration:

(a) On the ECLIPSE menu select "Run ▷ Run Configurations...".

(b) Select "Eclipse Application".

(c) Right-click on it and select "New".

[3] These procedure has been tested only with the master branch. If you want to reproduce this procedure, uncheck the annex and develop branches.

[4] Usually the download gets slow at around 92%, but it will eventually end.

(d) In the field "Name:" enter "Osate2"

(e) In the field "Location:" choose a location for the runtime OSATE2 workspace. For example `/home/user/osate-dev/runtime-osate-workspace`.

(f) In the field "Run a product:" click the down-pointing arrow to unfold and select the option "org.osate.branding.osate2".

(g) Click on the "Arguments" tab.

(h) Under "VM Arguments" add "-XX:MaxPermSize=256m -Xmx1200m"

(i) Click "Apply".

(j) Click "Run". OSATE2 starts, and when you close it, the run configuration will appear under the list or Run Configurations as well as the list that appears when you click on the down-pointing arrow next to the "Run" button on the ECLIPSE toolbar.

3 Outline

Here we give an overview of the whole process of creating the plugin(s). The following sections ellaborate on the details.

1. Create the syntax and UI plugins: (Section 4)
   (a) Create an XTEXT project.
   (b) Write an initial XTEXT grammar.
   (c) Configure the main XTEXT project (workflow and dependencies)
   (d) Generate XTEXT artifacts (produces three or four additional plugin projects, incuding the "UI" project).
   (e) Configure the XTEXT UI project (setup extensions, create custom activator class)
   (f) Add parser class for your annex and corresponding extension.
2. Create the analysis plugins: (Section 5)
   (a) Create a Java plug-in project
   (b) Configure the project (dependencies, actions, extensions, etc.)
   (c) Create actions.
   (d) Create visitors.
   (e) Set up a run configuration.

4 The core XTEXT project

4.1 Creation

1. Create a new XTEXT project:

   (a) Go to "File ▷ New ▷ Project..."

   (b) Unfold "Xtext" and choose "Xtext Project" (don't choose "Xtext Project from Existing Ecore models"[5]) and click "Next".

   (c) Enter a project name. While any name should work, it is suggested to use a name within the OSATE2 namespace, for example `org.osate.xtext.aadl2.sampleannex`.

   (d) Enter a name for your annex language. Again, it is suggested to use a name within the OSATE2 namespace, for example `org.osate.xtext.aadl2.sampleannex.SampleLang`.

   (e) Enter a file extension for your annex language. For example `samplang`.

   (f) Click "Finish".

   This will create four plug-in projects:

   (a) `org.osate.xtext.aadl2.sampleannex`: the core XTEXT project which contains the grammar and once compiled, will contain the main syntax artifacts, namely the ECORE meta-model (abstract syntax classes) for your language, the runtime features and other supporting infrastructure such as a serializer, validator, formatter, as well as skeletons for generators, etc.

   (b) `org.osate.xtext.aadl2.sampleannex.ui`: the user-interface plugin which provides a link between your language and the ECLIPSE/OSATE2 user-interface.

   (c) `org.osate.xtext.aadl2.sampleannex.tests`: the plugin for unit-tests.

   (d) `org.osate.xtext.aadl2.sampleannex.sdk`: the SDK plugin for creating update-sites.

   Each of the plugins contains three source folders:

   (a) `src`: where where hand-written source XTEXT, XTEND and JAVA packages and class files are placed.

   (b) `src-gen`: where JAVA packages and class files generated from XTEXT are placed.

   (c) `xtend-gen`: where JAVA packages and class files generated from XTEND are placed.

   Initially only the core project (`org.osate.xtext.aadl2.sampleannex`) cointains something, namely the files:

   (a) `src/org.osate.xtext.aadl2.sampleannex/SampleLang.xtext`: the source file for your XTEXT grammar, and

   (b) `src/org.osate.xtext.aadl2.sampleannex/GenerateSampleLang.mwe2`: the workflow (written in the MWE2 workflow language) that generates all runtime artifacts and infrastructure from your grammar. This is the file that you run to generate the infrastructure and where most configuration is done.

[5] Creating the XTEXT project from an existing ECORE meta-model seems to be incompatible with the AADL meta-model, so it is not recommended.

2. Edit the XTEXT source file (`src/org.osate.xtext.aadl2.sampleannex/SampleLang.xtext`) and enter something like the sample shown in Figure 1 on page 11. Pay attention to the places where the annex name and language name are used. You will likely need to inherit from some classes from the AADL metamodel. To do that, it is necessary to import the AADL ecore metamodel (line 5), and then you refer to the AADL classes by using the prefix "`aadl2::`". You will need to define at least, a subclass for "`aadl2::AnnexLibrary`" and "`aadl2::AnnexSubclause`" as shown in lines 10 and 13 respectively.

3. Look in the folder `model` folder of the `org.osate.aadl` plugin project. If there is a file called `aadl2-nouml.genmodel`, then skip the following step.

4. In the main project `org.osate.xtext.aadl2.sampleannex`create a folder called `prereqs/models` and copy into it the files:

   (a) `aadl2.ecore`: found in the folder `model` folder of the `org.osate.aadl` plugin project.

   (b) `aadl2.genmodel`: found in the folder `model` folder of the `org.osate.aadl` plugin project.

   (c) `aadl2-nouml.genmodel`: it was previously found in the `model` folder of the `org.osate.aadl` plugin project but was removed from the master branch of the sources. At this point there is no alternative solution but to ask the OSATE2 developers to provide this file.

4.2 Workflow configuration

1. Open the workflow file `src/org.osate.xtext.aadl2.sampleannex/GenerateSampleLang.mwe2` and under the `StandaloneSetup` section, comment out or remove the following lines:

```
registerGeneratedEPackage = "org.eclipse.xtext.xbase.XbasePackage"
registerGenModelFile = "platform:/resource/org.eclipse.xtext.xbase/
    model/Xbase.genmodel"
```

   Then, if the file `aadl2-nouml.genmodel` is in the `model` folder of the `org.osate.aadl2` plugin project, then add in their place the following line:[6]

```
registerGenModelFile = "platform:/resource/org.osate.aadl2/
    model/aadl2-nouml.genmodel"
```

   But if it is not, and you had to do step 4 in the previous section, then use the following line instead:

```
registerGenModelFile = "platform:/resource/org.osate.xtext.aadl2.sampleannex/
    prereqs/models/aadl2-nouml.genmodel"
```

2. (Optional) if (and only if) you created the XTEXT project from an ECORE model, then under "`language`", add or uncomment the following line (after the `GrammarAccess` fragment):

```
fragment = ecore.EcoreGeneratorFragment auto-inject {}
```

[6] Write it in one line.

```
grammar org.osate.xtext.aadl2.sampleannex.SampleLang
with org.eclipse.xtext.common.Terminals

generate SampleLang "http://www.osate.org/xtext/aadl2/sampleannex/SampleLang"
import "http://www.eclipse.org/emf/2002/Ecore" as ecore
import "platform:/resource/org.osate.aadl2/model/aadl2.ecore" as aadl2

SampleLangGrammarRoot :
      'library' lib = SampleLangAnnexLibrary
    | 'subclause' subclause = SampleLangAnnexSubclause
;

AnnexLibrary returns aadl2::AnnexLibrary:
    SampleLangAnnexLibrary
;

AnnexSubclause returns aadl2::AnnexSubclause:
    SampleLangAnnexSubclause
;

SampleLangAnnexLibrary returns SampleLangAnnexLibrary:
    {SampleLangAnnexLibrary}
    // your syntax here
    things += Thing+
;

SampleLangAnnexSubclause returns SampleLangAnnexSubclause:
    {SampleLangAnnexSubclause}
    // your syntax here
    goods += Stuff+
;

Thing:
    'thing' name=ID ';'
;

Stuff:
    'stuff' name=ID ';'
;
```

Figure 1: A sample XTEXT grammar.

And comment or remove the line

```
fragment = ecore2xtext.Ecore2XtextValueConverterServiceFragment auto-inject {}
```

4.3 Plug-ins configuration

1. Set the plug-in dependencies:
   (a) In project `org.osate.xtext.aadl2.sampleannex`:
      i. Open (double-click) on the file `META-INF/MANIFEST.MF`.
      ii. Open the "Dependencies" tab.
      iii. Click on "Add..." and enter "org.osate.aadl2"
      iv. Click on "Add..." and enter "org.osate.core"
      v. Save all ("File▷Save All" or Shift+Ctrl+S)
   (b) In project `org.osate.xtext.aadl2.sampleannex.ui`
      i. Open (double-click) on the file `META-INF/MANIFEST.MF`.
      ii. Open the "Dependencies" tab.
      iii. Click on "Add..." and enter "org.osate.aadl2"
      iv. Click on "Add..." and enter "org.osate.aadl2.modelsupport"
      v. Click on "Add..." and enter "org.osate.annexsupport"
      vi. Save all ("File▷Save All" or Shift+Ctrl+S)
2. Run the workflow:
   (a) Select the file `src/org.osate.xtext.aadl2.sampleannex/GenerateSampleLang.mwe2`
   (b) Right click and select "Run As▷MWE2 Workflow"

If you don't get any exception messages on the console and you see the message "Done", then the generation was succesful. If you had any trouble reported, go back to the previous steps and make sure that you have followed the steps precisely.

4.4 Setup the activator

You need to create an activator class for the UI plugin.

1. Create subclass of

   `org.osate.xtext.aadl2.sampleannex.ui.internal.SampleLangActivator`

   and save it in `src/org.osate.xtext.aadl2.sampleannex.ui`. Call it, for example,

   `org.osate.xtext.aadl2.sampleannex.ui.SampleLangCustomActivator`

```
1  package org.osate.xtext.aadl2.sampleannex.ui;

2  import org.osate.xtext.aadl2.sampleannex.ui.internal.SampleLangActivator;

3  import org.apache.log4j.Logger;
4  import org.osate.core.OsateCorePlugin;
5  import org.osgi.framework.BundleContext;
6  import com.google.inject.Injector;

7  public class SampleLangCustomActivator extends SampleLangActivator {

8      @Override
9      public void start(BundleContext context) throws Exception {
10         super.start(context);
11         try {
12             registerInjectorFor(ORG_OSATE_XTEXT_AADL2_SAMPLEANNEX_SAMPLELANG);
13         } catch (Exception e) {
14             Logger.getLogger(getClass()).error(e.getMessage(), e);
15             throw e;
16         }
17     }
18     @Override
19     public Injector getInjector(String languageName) {
20         return OsateCorePlugin.getDefault().getInjector(languageName);
21     }

22     protected void registerInjectorFor(String language) throws Exception {
23         OsateCorePlugin.getDefault()
24             .registerInjectorFor(language, createInjector(language));
25     }

26 }
```

Figure 2: Activator for the UI plugin:
`src/org.osate.xtext.aadl2.sampleannex.ui/SampleLangCustomActivator.java`.

This class should create and register an injector for the language, which is done with the "`createInjector`" method inherited from the parent `SampleLangActivator`, and the method "`registerInjectorFor`" from the instance of the default OSATE2 plugin. An example is found in Figure 2 on page 13. Pay attention to use the right annex and language names in the package and class names as well as in the constant used when invoking "`registerInjectorFor`" in line 12.

2. Register the activator:

   (a) Open (double-click) the file `META-INF/MANIFEST.MF` of the project

   `org.osate.xtext.aadl2.sampleannex.ui`

   (b) Go to the "Overview" tab.

   (c) Click "Browse..." in the "Activator" field and locate the file

   `org.osate.xtext.aadl2.sampleannex.ui.SampleLangCustomActivator`

created in the previous step. (When you type the name of the class it should appear in the list of options.)

(d) Save all ("File▷Save All" or Shift+Ctrl+S)

4.5 Setup the parser

Here we create the class that links the OSATE2 parser to the annex parser generated by XTEXT for your annex language.

1. Create a package `org.osate.xtext.aadl2.sampleannex.parsing` inside the project `org.osate.xtext.aadl2.sampleannex.ui`.

2. Inside this package create a class that implements `AnnexParser` as shown in Figure 3 on page 15. Make sure that you replace the appropriate names for `sampleannex` and `SampleLang`, and in particular in the calls "`getGrammarAccess().getSampleLangAnnexLibraryRule()`" (line 36) and "`getGrammarAccess().getSampleLangAnnexSubclauseRule()`" (line 47).

3. Create an extension to the `org.osate.annexsupport.parser` extension point:

   (a) Open (double-click) the file `META-INF/MANIFEST.MF` of the project

   `org.osate.xtext.aadl2.sampleannex.ui`

   (b) Go to the "Extensions" tab.

   (c) Click on "Add..." and enter "org.osate.annexsupport.parser" and click "Finish".

   (d) Select the new `org.osate.annexsupport.parser` extension and enter in its fields:

      i. field "id": org.osate.xtext.aadl2.sampleannex.ui.parser
      ii. field "name": SampleLang Annex Parser
      iii. field "class": `org.osate.xtext.aadl2.sampleannex.parsing.SampleLangAnnexParser`
      iv. field "annexName": SampleLang
      v. field "annexNSURI": http://www.osate.org/xtext/aadl2/sampleannex/SampleLang

   (e) Save all ("File▷Save All" or Shift+Ctrl+S)

You can test the syntax now, by running OSATE2(click the down-pointing arrow next to the "Run" button on the toolbar and select "Osate2"). If you get a conflict with other plugins, just select and close any conflicting plugin.

You can try a test such as the one in Figure 4 on page 16. If the keywords you used in your grammar appear highlighted, the parser is working.

```java
package org.osate.xtext.aadl2.sampleannex.parsing;

import org.osate.aadl2.AnnexLibrary;
import org.osate.aadl2.AnnexSubclause;
import org.osate.aadl2.modelsupport.errorreporting.ParseErrorReporter;
import org.osate.annexsupport.AnnexParseUtil;
import org.osate.annexsupport.AnnexParser;
import org.osate.core.OsateCorePlugin;
import org.osate.xtext.aadl2.sampleannex.parser.antlr.SampleLangParser;
import org.osate.xtext.aadl2.sampleannex.services.SampleLangGrammarAccess;
import com.google.inject.Injector;
import antlr.RecognitionException;

public class SampleLangAnnexParser implements AnnexParser {

    private Injector injector = OsateCorePlugin.getDefault()
        .getInjector("org.osate.xtext.aadl2.sampleannex.SampleLang");

    private SampleLangParser theSampleLangParser ;

    protected SampleLangParser getParser() {
        if (theSampleLangParser == null) {
            if (injector == null) {
                 injector = OsateCorePlugin.getDefault()
                    .getInjector("org.osate.xtext.aadl2.sampleannex.SampleLang");
            }
            theSampleLangParser = injector.getInstance(SampleLangParser.class);
        }
        return theSampleLangParser;
    }

    protected SampleLangGrammarAccess getGrammarAccess() {
        return getParser().getGrammarAccess();
    }

    @Override
    public AnnexLibrary parseAnnexLibrary(String annexName, String source,
            String filename, int line, int column,
            ParseErrorReporter errReporter) throws RecognitionException {
        AnnexLibrary annex_lib_ast_root =
                (AnnexLibrary) AnnexParseUtil.parse(getParser(),
                        source,
                        getGrammarAccess().getSampleLangAnnexLibraryRule(),
                        filename, line, column, errReporter);
        return annex_lib_ast_root;
    }

    @Override
    public AnnexSubclause parseAnnexSubclause(String annexName, String source,
            String filename, int line, int column,
            ParseErrorReporter errReporter) throws RecognitionException {
        AnnexSubclause annex_clause_ast_root =
                (AnnexSubclause) AnnexParseUtil.parse(getParser(),
                        source,
                        getGrammarAccess().getSampleLangAnnexSubclauseRule(),
                        filename, line, column, errReporter);
        return annex_clause_ast_root;
    }

}
```

Figure 3: The Annex parser.

```
package test1
public

annex SampleLang {**
    thing hello ;
    thing goodbye;
**};

thread T
annex SampleLang {**
    stuff one;
    stuff two;
**};
end T;

end test1;
```

Figure 4: An annex test.

5 An analysis plug-in

An analyzer for your annex language should be in its own separate plugin. Here we show how to do a simple analyzer with one action. This adds one new menu entry and one button to execute on a model.

5.1 Create a new plugin

Make a new plug-in as a Java plug-in project called, for example:

`org.osate.xtext.aadl2.sampleannex.analysis`

1. Go to "File▷New▷Project..." and select "Plug-in Project" and click "Next".
2. Enter the project name, *e.g.*, `org.osate.xtext.aadl2.sampleannex.analysis`, and click "Next" and "Next" again.
3. Select the "Hello, World" wizard, and click "Next" and "Finish".

5.2 Create an action set

1. Open (double-click) on the file `META-INF/MANIFEST.MF` of the project

   `org.osate.xtext.aadl2.sampleannex.analysis`

2. Open the "plugin.xml" tab and replace its contents with the code shown in Figure 5 on page 18.
3. Save all ("File▷Save All" or Shift+Ctrl+S)

5.3 Setup the dependencies

1. Open (double-click) on the file `META-INF/MANIFEST.MF` of the project

   `org.osate.xtext.aadl2.sampleannex.analysis`

2. Open the "Dependencies" tab.
3. Click on "Add..." and enter "org.osate.aadl2.modelsupport"
4. Click on "Add..." and enter "org.osate.aadl2"
5. Click on "Add..." and enter "org.osate.ui"
6. Click on "Add..." and enter "org.osate.xtext.aadl2.sampleannex"
7. Save all ("File▷Save All" or Shift+Ctrl+S)

```
<?xml version="1.0" encoding="UTF-8"?>
<?eclipse version="3.4"?>
<plugin>
   <extension
         point="org.eclipse.ui.actionSets">
      <actionSet
            label="OSATE Sample Analysis Action Set"
            visible="true"
            id="org.osate.xtext.aadl2.sampleannex.analysis.actionSet">
         <menu
               id="menu.osate"
               label="OSATE"
               path="project">
            <groupMarker name="file.grp"/>
            <separator name="instance.grp"/>
            <separator name="general.grp"/>
         </menu>
         <menu
               id="menu.analyses"
               label="Analyses"
               path="menu.osate">
            <groupMarker
                  name="top.grp">
            </groupMarker>
            <groupMarker
                  name="bottom.grp">
            </groupMarker>
         </menu>
         <menu
               id="menu.sampleanalysis"
               label="Sample Analyses"
               path="menu.analyses/top.grp">
            <groupMarker
                  name="stuff.grp">
            </groupMarker>
         </menu>
         <action
               label="&amp;SampleAnalysis"
               icon="icons/sample.gif"
               class="org.osate.xtext.aadl2.sampleannex.analysis.actions.SampleAction"
               tooltip="Sample Analysis"
               menubarPath="menu.analyses/menu.contracts/stuff.grp"
               toolbarPath="fault.toolbar"
               id="org.osate.xtext.aadl2.sampleannex.analysis.actions.SampleAction">
         </action>
      </actionSet>
   </extension>
</plugin>
```

Figure 5: plugin.xml for the actionSet of `org.osate.xtext.aadl2.sampleannex.analysis`

```java
package org.osate.xtext.aadl2.sampleannex.analysis.actions;

import org.eclipse.core.runtime.IProgressMonitor;
import org.osate.aadl2.Element;
import org.osate.aadl2.instance.InstanceObject;
import org.osate.aadl2.modelsupport.util.AadlUtil;
import org.osate.ui.actions.AaxlReadOnlyActionAsJob;
import org.osate.ui.dialogs.Dialog;
import org.osate.xtext.aadl2.sampleannex.analysis
    .visitors.SampleLangSampleAnalysisSwitch;

public class SampleAction extends AaxlReadOnlyActionAsJob {

    @Override
    protected String getActionName() {
        return  "SampleAction";
    }

    @Override
    protected void doAaxlAction(IProgressMonitor monitor, Element root) {
        // We only work on Declarative models...
        if (root instanceof InstanceObject) {
            Dialog.showError("Sample Analysis", "This analysis is done on "
                + "declarative models only, not on instance models. Please "
                + "select the appropriate model (.aadl file) in the AADL "
                + "Navigator view.");
        } else {
            monitor.beginTask(getActionName(), IProgressMonitor.UNKNOWN);
            // do real work here...
            SampleLangSampleAnalysisSwitch myt =
                new SampleLangSampleAnalysisSwitch(monitor);
            myt.processPreOrderAll(root);
            monitor.done();
        }
    }
}
```

Figure 6: Sample action.

5.4 Implement an action

The code for the action is implemented in a subclass of "`AaxlReadOnlyActionAsJob`". Go to

`src/org.osate.xtext.aadl2.sampleannex.analysis.actions/SampleAction.java`

and replace the code with the one shown in Figure 6 on page 19. This class delegates the actual processing to an ECORE AADL switch, described in the next subsection.

5.5 Implement a switch

The actual processing of declarative (syntactic) or instance (semantic) models is done by an ECORE "switch". See the EMF/ECORE documentation on switches.

- In the case of declarative models, such a switch contains a "`case`X" method for each type of syntactic construct X in your language (each return ECORE class in your XTEXT grammar). If there is no case method for a given construct, then the switch will return the result of the "`defaultCase`" method. Each"`case`X" method receives as input an instance of the node in the model's AST, and its features can ge accessed with "`get`X" methods. The return type must be `String` (a design decision of the OSATE2 developers).
- In the case of instance models, the switch contains a"`case`X" method for each type of semantic object (*e.g.* component instance).
- The code in Figure 7 on page 21 shows a typical switch to traverse a declarative model and perform actions on the annex elements.
- If you created the switch by subclassing "`AadlProcessingSwitchWithProgress`", you will have access inside the "`case`X" to a `monitor` object that can be used to report to the user the progress in the action with the methods "`subtask`" and "`worked`", and which should be polled by calling method "`isCanceled`" to check if the user has clicked the "Cancel" button.
- In the case of `AnnexLibrary` and `AnnexSubclause` nodes, the OSATE2 parser will produce a node of type "`DefaultAnnexLibrary`" and "`DefaultAnnexSubclause`" which do not represent the root of your particular annex, but have a method "`getParsedAnnexLibrary`" and "`getParsedAnnexSubclause`" respectively that return the actual root of your annex. To avoid executing twice an action on an annex library or subclause, the tests shown in lines 22-26 and 35-39 are necessary, to ensure that you are on the right node of the AST.
- In the "`case`X" methods for `AnnexLibrary` and `AnnexSubclause` nodes you can access the features of the root element in your grammar (`SampleLangGrammarRoot` in Figure 1 on page 11), and from it you can access the AST nodes of sub-terms in your language.
- A preferred and clean approach to process your particular language is to create a switch for your language which should be an instance of the class `SampleLangSwitch<T>` automatically generated by XTEXT for you and defined in the package

  `org.osate.xtext.aadl2.sampleannex.SampleLang.util`

  found in the folder

```java
package org.osate.xtext.aadl2.sampleannex.analysis.visitors;

import org.eclipse.core.runtime.IProgressMonitor;
import org.osate.aadl2.AnnexLibrary;
import org.osate.aadl2.AnnexSubclause;
import org.osate.aadl2.DefaultAnnexLibrary;
import org.osate.aadl2.DefaultAnnexSubclause;
import org.osate.aadl2.modelsupport.modeltraversal.AadlProcessingSwitchWithProgress;
import org.osate.aadl2.util.Aadl2Switch;
import org.osate.xtext.aadl2.sampleannex.SampleLang.SampleLangAnnexLibrary;
import org.osate.xtext.aadl2.sampleannex.SampleLang.SampleLangAnnexSubclause;

public class SampleLangSampleAnalysisSwitch extends
        AadlProcessingSwitchWithProgress {

    public SampleLangSampleAnalysisSwitch(IProgressMonitor pm) {
        super(pm);
    }

    @Override
    protected void initSwitches() {
        aadl2Switch = new Aadl2Switch<String>() {
            public String caseAnnexSubclause(AnnexSubclause obj) {
                monitor.subTask("AnnexSubclause " + obj.getName());
                if (monitor.isCanceled()) return null;
                if (obj instanceof DefaultAnnexSubclause)
                    return "";
                if (!(obj instanceof SampleLangAnnexSubclause))
                    return "";
                if (obj.getName().equals("SampleLang")) {
                    // do something
                }
                monitor.worked(1);
                return obj.toString();
            }
            public String caseAnnexLibrary(AnnexLibrary obj) {
                monitor.subTask("AnnexLibrary" + obj.getName());
                if (monitor.isCanceled()) return null;
                if (obj instanceof DefaultAnnexLibrary)
                    return "";
                if (!(obj instanceof SampleLangAnnexLibrary))
                    return "";
                if (obj.getName().equals("SampleLang")) {
                    // do something with
                }
                monitor.worked(1);
                return obj.toString();
            }
        };
    }

}
```

Figure 7: An ECORE switch for AADL.

`src-gen/org.osate.xtext.aadl2.sampleannex.SampleLang.util`

inside the project

`org.osate.xtext.aadl2.sampleannex`

For example, you can define a switch like the one shown in Figure 8 on page 23. such switch works on AST nodes specific to your language, rather than the full AADL. Note that in such switch, you have only "`case`X" specific to your grammar. and that you can use the standard ECORE operations on AST nodes. In particular, you can obtain the features or subterms of an AST node "`obj`" by invoking the method "`obj.get`Y" for a feature or sub-term Y, as shown in lines 9 and 15. In this example, we explicitly call the `doSwitch` method on subterms, in order to control exactly which nodes will be visited. A common alternative is to invoke a method such as "`someSwitch.processPreOrderAll(root)`" applied to a switch and passing as argument the root of the AST. The "`processPreOrderAll`" method is defined in the "`AadlProcessingSwitch`" class, together with many other traversal methods. Note, however that such methods are defined for classes derived from "`AadlProcessingSwitch`", and not for those switches generated by XTEXT such as `SampleLangSwitch<T>`. If you want more traversal methods for your specific sub-language, you have to write them yourself.

Assuming you defined this class from Figure 8 on page 23 in package

`org.osate.xtext.aadl2.sampleannex.analysis.visitors`

then the updated main switch would look like Figure 9 on page 24. Note that it declares a field for the new sub-switch (line 14), creates an instance of `SampleLangSpecificSwitch` in the "`initSwitches`" method (line 20), and then the new sub-switch is invoked with the "`doSwitch`" method, passing the AST node as argument (lines 31,45).

- If you wish to do analysis on an instance model instead, in the "`initSwitches`" method you must initialize the "`instanceSwitch`" field with an instance of class "`InstanceSwitch`", and in the action of Figure 6 on page 19, you should make the creation and invocation to the switch in the first branch of the conditional that tests if the `root` is an instance of `InstanceObject`.

```java
package org.osate.xtext.aadl2.sampleannex.analysis.visitors;

import org.osate.xtext.aadl2.sampleannex.SampleLang.Stuff;
import org.osate.xtext.aadl2.sampleannex.SampleLang.Thing;
import org.osate.xtext.aadl2.sampleannex.SampleLang.SampleLangAnnexLibrary;
import org.osate.xtext.aadl2.sampleannex.SampleLang.SampleLangAnnexSubclause;
import org.osate.xtext.aadl2.sampleannex.SampleLang.util.SampleLangSwitch;

public class SampleLangSpecificSwitch extends SampleLangSwitch<String> {

    public String caseSampleLangAnnexLibrary(SampleLangAnnexLibrary obj) {
        for (Thing t : obj.getThings()) {
            doSwitch(t);
        }
        return "";
    }

    public String caseSampleLangAnnexSubclause(SampleLangAnnexSubclause obj) {
        for (Stuff s : obj.getGoods()) {
            doSwitch(s);
        }
        return "";
    }

    public String caseThing(Thing obj) {
        System.out.println("found a thing: " + obj.getName());
        return "";
    }

    public String caseStuff(Stuff obj) {
        System.out.println("found stuff: " + obj.getName());
        return "";
    }

}
```

Figure 8: A (sub)language-specific switch.

```java
package org.osate.xtext.aadl2.sampleannex.analysis.visitors;

import org.eclipse.core.runtime.IProgressMonitor;
import org.osate.aadl2.AnnexLibrary;
import org.osate.aadl2.AnnexSubclause;
import org.osate.aadl2.DefaultAnnexLibrary;
import org.osate.aadl2.DefaultAnnexSubclause;
import org.osate.aadl2.modelsupport.modeltraversal.AadlProcessingSwitchWithProgress;
import org.osate.aadl2.util.Aadl2Switch;
import org.osate.xtext.aadl2.sampleannex.SampleLang.SampleLangAnnexLibrary;
import org.osate.xtext.aadl2.sampleannex.SampleLang.SampleLangAnnexSubclause;
import org.osate.xtext.aadl2.sampleannex.SampleLang.util.SampleLangSwitch;

public class SampleLangSampleAnalysisSwitch extends
        AadlProcessingSwitchWithProgress {

    protected SampleLangSwitch<String> samplangSwitch;

    public SampleLangSampleAnalysisSwitch(IProgressMonitor pm) {
        super(pm);
    }

    @Override
    protected void initSwitches() {
        samplangSwitch = new SampleLangSpecificSwitch();
        aadl2Switch = new Aadl2Switch<String>() {
            public String caseAnnexSubclause(AnnexSubclause obj) {
                monitor.subTask("AnnexSubclause " + obj.getName());
                if (monitor.isCanceled()) return null;
                if (obj instanceof DefaultAnnexSubclause)
                    return "";
                if (!(obj instanceof SampleLangAnnexSubclause))
                    return "";
                if (obj.getName().equals("SampleLang")) {
                    // do something
                    return samplangSwitch.doSwitch(obj);
                }
                monitor.worked(1);
                return obj.toString();
            }
            public String caseAnnexLibrary(AnnexLibrary obj) {
                monitor.subTask("AnnexLibrary" + obj.getName());
                if (monitor.isCanceled()) return null;
                if (obj instanceof DefaultAnnexLibrary)
                    return "";
                if (!(obj instanceof SampleLangAnnexLibrary))
                    return "";
                if (obj.getName().equals("SampleLang")) {
                    // do something with
                    return samplangSwitch.doSwitch(obj);
                }
                monitor.worked(1);
                return obj.toString();
            }
        };
    }

}
```

Figure 9: An ECORE switch for AADLwith a sub-switch for the sub-language.

Description	Link
Main wiki page	https://wiki.sei.cmu.edu/aadl/index.php/Osate_2
Getting Osate 2 sources	https://wiki.sei.cmu.edu/aadl/index.php/Getting_Osate_2_sources
Links to several pages on developing OSATE2 plug-ins.	https://wiki.sei.cmu.edu/aadl/index.php/OSATE_Tool_Developer
Information on sub-language extensions. It gives information on how to link the parser for your language with OSATE2.	https://wiki.sei.cmu.edu/aadl/index.php/Sublanguage_extensions
Outdated tutorial for sublanguage extensions This uses ANTLR directly instead of XTEXT, and ECORE directly for the meta-model.	https://wiki.sei.cmu.edu/aadl/index.php/Sublanguage_Example
Information about types of OSATE2 plug-ins	https://wiki.sei.cmu.edu/aadl/index.php/OSATE_2_Plug-ins
Some limited information about the OSATE2 API to work with models.	https://wiki.sei.cmu.edu/aadl/index.php/OSATE_V2_Plugin_Development
Implementing the action class.	https://wiki.sei.cmu.edu/aadl/index.php/Replacing_AaxlReadOnlyAction_with_AaxlReadOnlyActionAsJob_in_OSATE_1.2.4
Old OSATE2 development guide.	http://www.aadl.info/aadl/currentsite/downloads/Plug-in%20Guide%202005-06-16%201030.pdf

Table 1: OSATE2 wiki links.

6 Other references

The OSATE2 wiki (https://wiki.sei.cmu.edu/aadl/index.php/Osate_2) contains many references on how to develop plug-ins, but unfortunately a great deal of information is outdated, not directly applicable to an XTEXT annex or scattered and difficult to find. That's why I have written these notes. Some useful links from the wiki are summarized in Table 1 on page 25.

References

[1] SAE International. Architecture Analysis & Design Language (AADL). SAE Standard AS5506b, 10 September 2012.