SINTEF

Developer's Manual

SiSaS Studio v 2.0

SINTEF ICT

6/11/2012

DEVELOPER'S MANUAL

FOR SISAS STUDIO V2.0

Version: 1.0

Contributors:

- Franck Chauvel – SINTEF / ICT

Summary:

This document briefly explains how to add new features in the SiSaS studio v1.0. It first outlines the basic software toolset needed, and depicts the main architecture of the SiSaS Studio. The addition of a simple feature is given as a tutorial and some common issues are discussed to conclude.

Change History:

Version	Date	Changes Description	Author
1.2	June 11, 2012	Add section about bundle creation	F. Chauvel
1.1	June 8, 2012	Introduce the new architecture	F. Chauvel
1.0	Nov 29, 2011	First Version	F. Chauvel
0.1	Oct 17, 2011	Initial Outline	F. Chauvel

1 CONTENTS

2 INTRODUCTION

SiSaS Studio [5] is an Eclipse bundle that enables to generate code (Java [2] among others) from various models, especially from UML models [7]. The core part of the SiSaS studio [1] is basically a set of model transformations, written in MOFScript [4], and some additional Java classes that ensure the proper integration of these transformations within the Eclipse GUI [1].

The SiSaS studio bundle also includes other Eclipse plugins that support common development tasks associated with the model-driven methodology: UML modelling using the Papyrus UML Editor [10], Mathematical modelling using the OpenModelica plugin [8, 12], etc. Although these plugins are bundled within the SiSaS Studio, they must be considered as third party elements, and are maintained separately.

This document is aimed at people who need to extend or to fix bugs in the SiSaS Studio. It is NOT aimed at people who merely want to use it. Please consult the User Guide [5] in this case.

The aim of this document is to introduce the tool set used to develop and extend the SiSaS Studio. The first section describes precisely how to setup your development environment. The second part then briefly describes the architecture of the model transformations that have been developed in the SiSaS Studio.

3 SETTING UP YOUR DEVELOPMENT ENVIRONMENT

3.1 INSTALLING THE DEVELOPMENT TOOLSET

Below are the main steps to follow in order to configure your SiSaS development environment. These steps are only relevant to setup the environment needed to develop the SiSaS Studio itself, and is NOT relevant if you only want to use it.

1. **Java Development Kit** (JDK v1.6). The first element to install is the Java Development Kit [2], which includes the Java virtual machine, and the Java development tools. This is needed to run the Eclipse IDE. It is worth to note that you can run Eclipse over newer version of the JDK (e.g., on Java 7) but the code of the SiSaS Studio itself requires Java 6.
2. **Eclipse-Modelling** (v3.5.x, so called "Galileo", NOT higher). The second element needed is a distribution of the Eclipse Modelling Framework. We recommend the version 3.5 as some of the plugins that will be later added require this specific version.
3. **Papyrus UML** [10] is the UML editor advocated by the SiSaS studio. It comes as an Eclipse plugin and can be installed in Eclipse using the update site.
4. **MOFScript** [4] (v1.4 or higher) is the programming language used to write the model transformations that are provided by the SiSaS Studio. The MOFScript language comes as well as an Eclipse plugin providing syntax highlighting and automatic completion. The MOFScript plugin can be installed using the update site.
5. **SVN Plugin.** You might want as well to install an additional SVN plugin, which permits to directly commit the changes you made to the SiSaS source code into the SiSaS repository (So they become available for other developers). Basically, two main plugins are

available: *Subclipse* and *Subversive*. Both of them are compatible with the rest of the toolset.

3.2 Checking out the SiSaS Studio Source Code

The source code of the SiSaS studio is available on a SVN server. You can retrieve it from the following URL:

https://cloud.catenda.no/svn/sisas/sisasstudio/

If you have not yet got proper credential to access the SiSaS Studio, you may contact the Catenda staff in charge of the SVN repository.

4 Architecture of the SiSaS Studio

4.1 A Set of Eclipse Plugins

The SiSaS Studio is made of four main Eclipse plugins:

- **Profiles plugin** defines a set of UML profiles that can be used within the SiSaS Studio, e.g., SoaML, migration, etc.
- **Transformations plugin** binds the Eclipse graphical interfaces to the set of MoFScript transformations. Basically, it drives the MoFScript engine and selects the proper transformation according to the UI events.
- **ProjectTemplates** provides the user with the ability to define his own project templates, assuming that the transformation he needs already exist in the SiSaS Studio.
- **Features plugin** is basically a collection of plugins. It is used as a façade/container by the Eclipse Framework. It depends on the two other plugins
- **Site plugin** permits to create an Eclipse *update-site* from which the SiSaS Studio plugin can be retrieved and install. It is used when building the SiSaS Studio bundle.

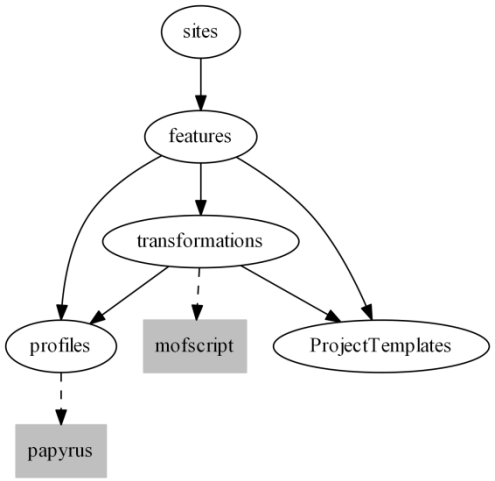


Figure 4.1 Dependencies between the plugins that compose the SiSaS Studio

The dependencies between these four plugins are depicted by Figure 1, aside. For a better understanding of the Eclipse plugin framework, the interested reader may consult [5].

4.2 Internal Architecture

The SiSaS Studio is built as a sort of "product line" of models transformations. The basic idea is to separate the idea definition of specific model transformations from their combination to generate complex code structures (*e.g.*, Maven projects). To this end, the SiSaS studio let the user define "project templates" where she specifies the organization of the code she needs in terms of a directory structure containing the "to-generate" artefacts.

4.2.1 Template Projects

Template Projects are separated models (bundle in a separate Eclipse models) which capture the organization of the code to generate, especially complex structures including multiple generated artefacts. Although the user can define is own project templates, the SiSaS Studio comes with a set of predefined project templates supporting (POJO projects, OGC/WPS projects, EJB, *etc.*)

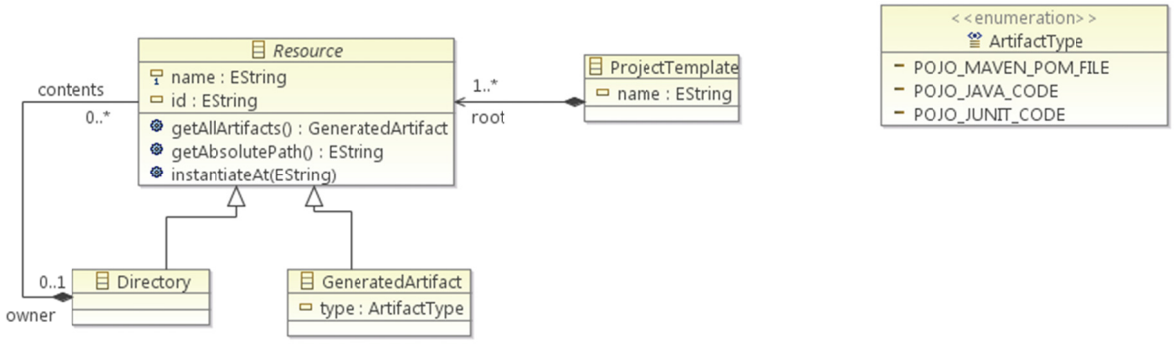


Figure 4.2 The Project Template metamodel

As shown by Figure 4.2, "project templates" basically describe a directory structure in which the needed artefacts can be place so as to ensure the proper generate of the complete project.

Project template defines a ***fixed*** set of generated artefact types. Adding a new type of generated artefact requires to modify the model of project template and hence to regenerate and recompile the code the related Eclipse plugin.

4.2.2 Model Transformations

The SiSaS Studio is actually a registry of models transformation, each of them targeting a specific type of generated artefact. While instantiating a specific project template, the sisas studio merely trigger the transformation relevant for each generated artefacts.

Each transformation registered in the SiSaS Studio made of two parts: generators and checkers. Generators are in charge of effectively generating the code, where as a checker is in charge of verifying whether the model given as input by the user contains sufficient information for the generator to run properly.

It is worth to note that ***the consistency between a generator and its related checker is the responsibility of the transformation developer***.

Figure 4.3, below illustrates how model transformation registered with the SiSaS Studio. The SiSaS Studio explicitly declares the transformations that are available, including the two separate MofScript files and the type artefact they generate. Invoking the "instantiate" will then pass through the template that trigger the generation of each artefact using the relevant transformation (or raise an error if not transformation are available to handle one of the artefact).

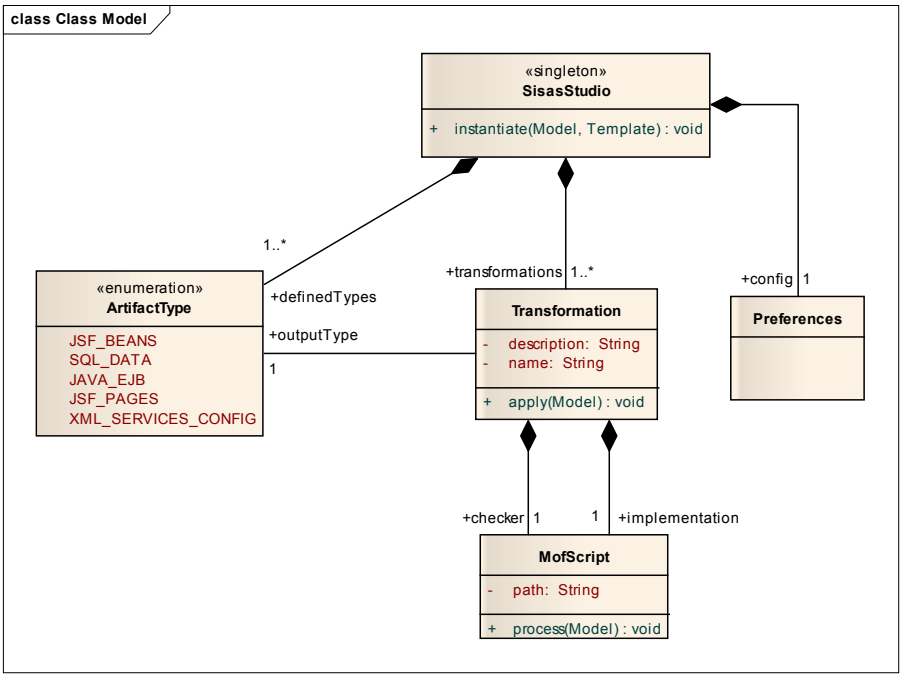


Figure 4.3 Internal representation of model transformation within the SiSaS Studio

Registering a new transformation in the SiSaS Studio is hence a matters of a few lines, in the initialization of the SisasStudio singleton class. In the following code snippet, we registered one transformation that aims at generated POJO (plain old java projects).

```
private SisasStudio() {
                transfoRegistry = new Hashtable<ArtifactType, Transformation>();

                // ----
                // ADD BELOW ANY TRANFORMATION THAT MUST BE AVAILABLE BY DEFAULT OR ANY
                // NEW TRANSFORMATION THAT MUST BE SUPPORTED
                //
                List<Transformation> transformations = new LinkedList<Transformation>();
                transformations.add(new Transformation("UML TO POJO",
                                ArtifactType.POJO_JAVA_CODE,
                                "uml to pojo/plain java checker.m2t",
                                "uml to pojo/plain java generator.m2t"));
                transformations.add(new Transformation("UML to MAVEN POM FOR Java",
                                ArtifactType.POJO MAVEN POM FILE,
                                "uml_to_pojo/maven_pom_checker.m2t",
                                "uml_to_pojo/maven_pom_generator.m2t"));

                // ----
                // ...

        }
```

4.2.3 *Existing Model Transformations*

Scope	Input	Required Profile	Output
Storage	UML	SoaML / Persistence	XSD Schema SQL Schema Creation SQL Schema Deletion
Persistence	UML	SoaML / Persistence	Java Classes (POJO) JEE 6 – Entity Beans JEE 6 – Entity Managers
Business	UML	SoaML	WS - WSDL WS – Skeleton (JAX-WS) JEE 6 – Stateless Session Beans

Presentation	UML		JSP Web Service Client JSF Sample Site
Development	UML	None	Maven JEE Project Structure
Import/Export	Enterprise Architect	None	ECore Model

Table 1. Overview of the model transformations included in the SiSaS Studio

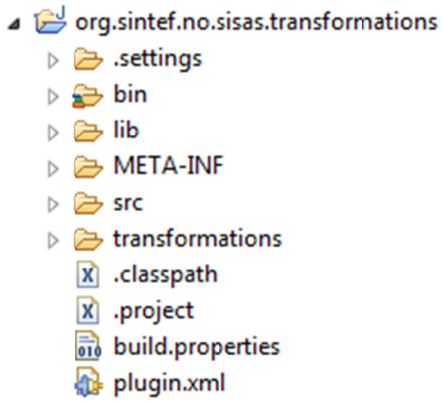
As shown in the table above, the SiSaS Studio contains a set of model transformations that be applied on various models, mainly on UML model, extended with *ad hoc* profiles. The source code of all these transformations is contained into the "transformation" plugin.

The transformations included in the SiSaS Studio can be divided in three categories. The first one contains all the transformations that produce "production" code, *i.e.*, code that will be later integrated in the final application. In Table 1, this first category corresponds to first set of transformation (including storage, persistence, business and presentation). The second set contains transformations that generate code useful to compile and package the final application: basically the transformation generating the Maven Infrastructure. Finally, the last group of transformation contain transformations providing conversion means between different types of models (*e.g.*, converting Enterprise Architect Model into ECore models).

5 TUTORIALS

5.1 DEVELOPING A NEW TRANSFORMATION

A good practice to follow while adding new transformations into the SiSaS Studio, is to first develop and test the transformation aside, and then to integrate it within the SiSaS Studio. This section describes briefly the second parts, namely ***how to integrate a new transformation***. Interested readers may refer to [4] for additional details about how to write MoFScript transformation.



This done is three main steps:

1. Develop the **generator**, a separate MoFScript transformation that generates the needed code
2. Develop the ***checker***, a MoFScript transformation that checks whether a given UML model is proper regarding the requirements of the generator
3. Register a new transformation in the SiSaS Studio that bind together the generator, the checker and the type of artefact that is produced.

5.1.1 DEVELOPING THE GENERATOR SCRIPT

The code snippet below is a very simple model transformation that merely prints a welcoming message on the Eclipse console. It is written in the MOFScript language [4]. The next paragraph shows how to integrate this transformation into the SiSaS Studio.

```
/**
 * Test.m2t
 *   Test whether the MoFScript Engine is ready for use.
 *
 * date: 29/10/2011
 *
 * author: Franck Chauvel - SINTEF
 */

texttransformation FooGenerator (in mdl:"http://www.eclipse.org/uml2/2.1.0/UML") {

  mdl.Model::main () {
     println("This is a test:" )
     println("If you can read this message, MofScript is operational")
  }

}
```

The file that contains this code snippet must be placed into the transformation directory of the transformation plugin (or one of its sub directories). For the sake of consistent organisation, we recommend to place it in "transformations/uml_to_foo/foo_generator.m2t"

5.1.2 *Developing the Checker Script*

The following code snippets illustrates how to write a checker, and especially how to return a values that can be understood in the Java layer of the SiSaS Studio. In this simple example, the checker actually checks nothing, and replies that the models conforms the requirements, sending the "CHECKER_PASS" message.

```
import "../configuration.m2t"


texttransformation FooChecker(in mdl:"http://www.eclipse.org/uml2/2.1.0/UML") {

  mdl.Model::main () {
        print(CHECKER_PASS);
  }

}
```

This file must be saved in the transformation plugin in a separated directory (for sake of organization). For instance, we may place it into "transformation/uml_to_foo/for_checker.m2t"

5.1.3 *Registering the Transformation*

The second step aims at writing a piece of Java code that registers the generator and the checker in the SiSaS Studio. Hence, instantiating templates containing the related generated artefact type will hence trigger the execution of both the generator and the checker. The code excerpt below illustrates such a registration: In the initialization of the SisasStudio singleton class (org.sintef.no.sisas.transformation.SisasStudio), we add in the internal registry a new transformation point to the checker, the generator, and the artefact type.

```
private SisasStudio() {
                transfoRegistry = new Hashtable<ArtifactType, Transformation>();

                // ----
                // ADD BELOW ANY TRANFORMATION THAT MUST BE AVAILABLE BY DEFAULT OR ANY
                // NEW TRANSFORMATION THAT MUST BE SUPPORTED
                //
                List<Transformation> transformations = new LinkedList<Transformation>();
                transformations.add(new Transformation("UML TO POJO",
                                ArtifactType.POJO_JAVA_CODE,
                                "uml_to_pojo/plain_java_checker.m2t",
                                "uml_to_pojo/plain_java_generator.m2t"));
                transformations.add(new Transformation("UML to Foo",
```

```
                                ArtifactType.FOO_FILE,
                                "uml to foo/foo checker.m2t",
                                "uml to foo/foo generator.m2t"));

                // ----
                // ...

        }
```

5.2 Adding a new Project Template

Additionally, new project templates can be defined, and have to be defined to support new types of generated artefacts. The SiSaS studio includes a plugin which permit to build new project templates. To do so, just create a new Project template file, as describe the structure of the resulting source directory, as shown on Figure 4.2 below:

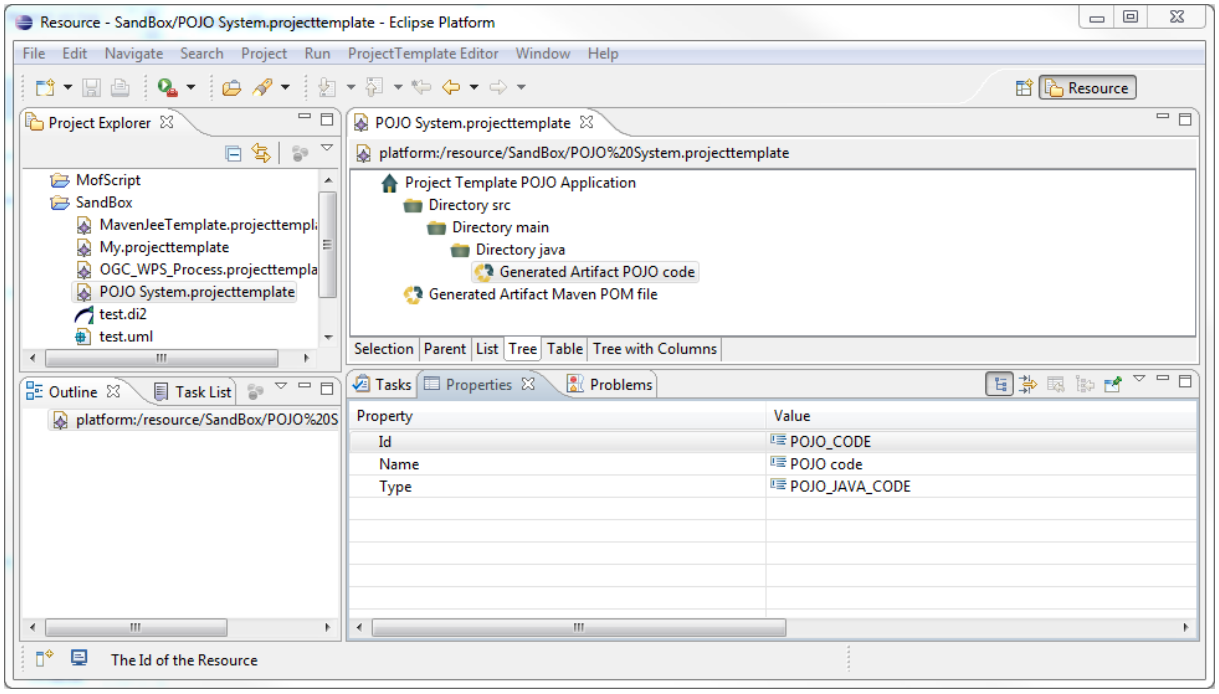


Figure 5.2 Defining new project template with the project template editor

Once the desired template has been designed and saved in a ".projecttemplate" file, it can be either registered during the SiSaS Studio session, using the SiSaS Popup menu, or copy into the SiSaS Studio installation, so as to pertain for later sessions. The transformation plugin has a specific directory named "template" which is automatically browsed when the SiSaS Studio starts, and each template placed there will be loaded and hence accessible in the SiSaS popup menu.

5.3 Creating a Bundle of the SiSaS Studio

Creating a bundle of Eclipse that contains an updated version of the SiSaS Studio is a simple process:

1. Go to the update site of the SiSaS Studio plugin (project org.sintef.no.sisas.transformations.update). Then rebuild the update site, build right clicking on PDE tool, and selecting "Build site".
2. Install a brand new eclipse distribution and install the needed plugins inside, MoFScript, Papyrus, etc.

3. Start the new eclipse and select File->Install New Software. Select a local repository, and provide the location of the update site directory (namely the place where you checkout the org.sintef.no.sisas.transformation.update). This will install the Sisas Studio in the new Eclipse distribution.
4. Exit Eclipse and create an archive containing the new eclipse distribution (just zip the eclipse directory). Your bundle is ready, enjoy ☺

6 TROUBLESHOOTING

This section summarizes common problems that may occur while developing the SiSaS Studio.

6.1.1 UNABLE TO START THE WORKBENCH

When running the SiSaS Studio as an Eclipse Workbench, you may get "*Could not reserve enough space for object heap. Could not create the Java virtual machine.*" This might be due to a mismatch between the JVM used by Eclipse when it creates a new workbench, and the one provided by your operating system. Make sure that the latter one is a 32 bit version. If you are using a 64 bits system, you may need to install an older version of the JVM, optimized for 32 bits systems.

7 REFERENCES

1. Eric Clayberg, Dan Rubel. *Eclipse Plug-ins.* 4th edition, Addison-Wesley, 2009.
2. Bruce Eckel, *Thinking in Java*. 4th edition, Prentice Hall, 2006.
3. Kito D. Mann. *Java Server Faces in Action*. Manning Publications, 2005.
4. Jon Oldevik. *MOFScript User Guide*. Unpublished. Version 1.0, February 2011. (available at http://eclipse.org/gmt/mofscript/)
5. Gøran K. Olsen. *SiSaS Studio – User Manual*. Volume1, 2 and 3. Unpublished.
6. Object Management Group (OMG). *Service oriented architecture Modeling Language (SoaML) - Specification for the UML Profile and Metamodel for Services (UPMS) SoaML*. ptc/2009-12-09. 2009. (see http://www.omg.org/spec/SoaML/)
7. Object Management Group (OMG). *Unified Modeling Language – Superstructure* (v2.4.1). formal/2011-08-06. 2011. (see http://www.uml.org/)
8. Open Modelica. (See http://www.openmodelica.org/)
9. Debu Panda, Reza Rahman and Derek Lane. *EJB 3 in Action*. Manning Publications, 2007.
10. Papyrus UML. (See http://www.eclipse.org/modeling/mdt/papyrus/)
11. Chris Richardson. *POJO in Action: Developing Enterprise Applications with Lightweight Frameworks*. Manning Publications, 2006
12. Wladimir Schamai. *Modelica Modeling Language (ModelicaML): A UML Profile for Modelica*. (available at http://www.openmodelica.org/index.php/developer/tools/134)
13. Sonatype. *Maven: The Definitive Guide*. O'Reilly Media. September 2008.
14. Eric Van der Vlist. *XML Schemas: The W3C's Object-Oriented Descriptions for XML*. O'Reilly Media. June 2002.