SINTEF

User's Manual

SiSaS Studio v 2.0

SINTEF ICT

6/12/2012

USER'S MANUAL

FOR SISAS STUDIO V2.0

Version: 0.1

Contributors:

- Franck Chauvel – SINTEF / ICT

Summary:

This document briefly list the features integrated in the SiSaS Studio (version 2.0) and explains how to use them. It also overview the development process underlying the use of the SiSaS Studio.

Change History:

Version	Date	Changes Description	Author
0.2	June 13, 2012	Main section filled out	F. Chauvel
0.1	June 12, 2012	Initial Outline	F. Chauvel

TABLE OF CONTENTS

1 INTRODUCTION

This document is about the SiSaS Studio, developed by SINTEF within the SiSaS Project. SiSaS provides tools and methods helping SINTEF researchers to develop and release software, especially scientific software. In this setting, the SiSaS Studio – as a development environment – provides users with the ability to model software systems and to generate part or the totality of the implementation.

This document illustrates how to use the SiSaS Studio to generate various sort of code (Java, XML, etc.) from UML models (mainly). This document mainly focuses on the code generation abilities of the SiSaS Studio, although the SiSaS Studio permits to build UML models as well. As all modelling features result from the integration of existing software (EMF plugins, Papyrus editor, etc.), the interested reader shall refer to the associated documentation and user guides.

It is worth to note that this document does not cover the internal architecture of the SiSaS Studio. It neither explains how to extend the SiSaS Studio so as to generate any new type of code that could be needed. Interested readers should refer to the companion "Developer's manual".

This document is organized as follows. Section 2 explains how to fetch and install the SiSaS Studio, and list third party software that are required to properly run and use the SiSaS Studio. Section 3 recalls the methodology that comes along with the SiSaS Studio. Section 4 dives into code generation from UML models, including Java application, Web services, JEE systems, etc. Finally, Section 5 provides tutorial explanation on specific issues, especially regarding customization of the SiSaS Studio.

2 INSTALLING THE SISAS STUDIO

2.1 INSTALLATION FROM A BUNDLE

The SiSaS Studio bundle can be downloaded as a zip file from the e-room location: https://project.sintef.no/eRoomReq/Files/ikt/SiSaS/0_44b57/SiSaS_Studio_0.3.zip

1. Unzip the bundle to your preferred directory
2. Locate the eclipse.exe file and start Eclipse
3. When starting Eclipse, choose your working directory
4. If desired, you can import the projects from the examples directory

2.2 INSTALLATION FROM THE ECLIPSE UPDATE SITE

Eclipse also includes a mechanism to automatically fetch and install additional plugins from the Internet. This alternate installation solution requires installing the SiSaS Studio using the following update site: http://www.modelbased/sisas/etc/sisas-studio.site

In this setting, you must have an Eclipse distribution matching the following criteria.

- Eclipse 3.5
- Eclipse Modelling Framework
- MoFScript Plugin 1.4
- Papyrus UML Editor, v1.12

3 DEVELOPING WITH SISAS STUDIO

The SiSaS Studio has been designed as a support for *Model Driven Engineering* (MDE). As a general paradigm, MDE advocates a special focus on models describing various aspects of a system, and from which part or the totality of the code can be automatically generated. To this ends, the SiSaS Studio embeds a basic modelling environment and a set of predefined models transformations to generate code. The modelling environment, named Papyrus allows users to graphically create UML models (it supports various types of diagram). The set of predefined model transformation allows users to obtain the implementation of the model, automatically.

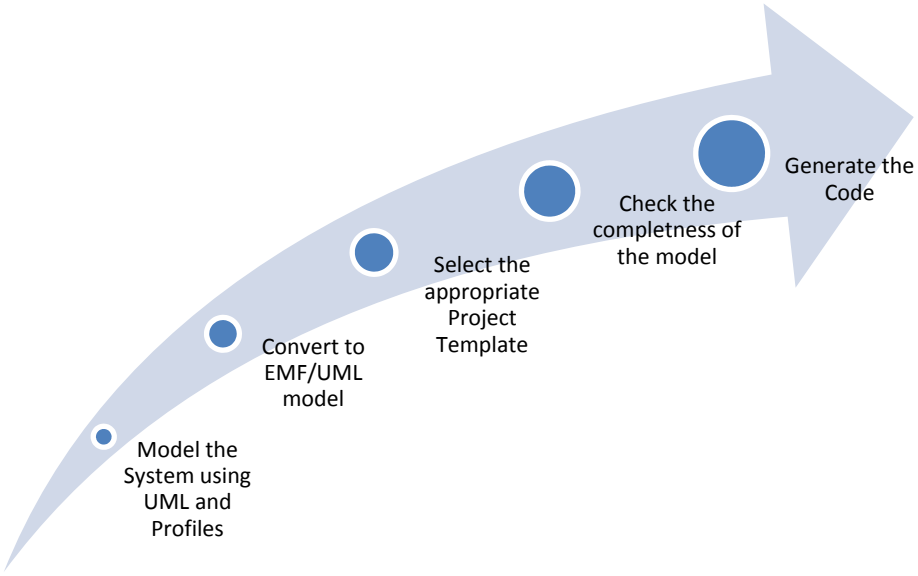

Figure 3.1 The process of developing software systems with SiSaS Studio

Figure 3.1 above details the process of building an application with the SiSaS Studio. Further examples are provided in Section 4, which illustrate the various types of applications that can be built with the Studio. This five steps process is structured as follows:

1. ***Model the System using UML and Profiles***. In this initial stage, the designer of the system is in charge of describing the system he needs, and annotating the model with specific profiles (migration, persistence, etc.)
2. ***Convert to EMF/UML model***. The transformations that are bundled in the SiSaS Studio exploit UML model stored in a specific format, namely EMF/UML model. It is hence necessary to convert models stored in other format, so as to enable model transformation.
3. ***Select the appropriate project template***. The SiSaS Studio provides a set of predefined templates, representing different code-level project structure reflecting specific applicative or development framework. Each template specifies a set of artefacts to be automatically generated. The user has merely to select the template he wants, or to defined a new one if none meets his requirements (*cf.* Section 5.1 for further detail about template creation).
4. ***Completeness Check***. When instantiating a given project template on a specific UML model, the SiSaS Studio will first check whether the given UML match the requirements of the select set of transformation. It is worth to note that model transformations operate on specific subsets of the UML notation and incomplete or inconsistent model would

leads to ill-formed code. To avoid such a situation, the SiSaS Studio embeds checkers that ensure that a given models matches the requirements of the selected model transformations. ***Such checks are automatically performed.***

5. ***Code Generation.*** Assuming that the given UML model is complete and consistent, the SiSaS Studio will generate all the artefacts specified in the project templates. ***This step is also performed automatically***.

3.1 Modelling with Enterprise Architect

Figure 3.2 below shows the look and feel of Enterprise Architect. Enterprise Architect is a commercial CASE tool, which supports the user in designing various UML models, but also supports other notations such as EA models, workflows, *etc*. Enterprise Architect, at least in its version 8.0, provides a relatively mature graphical interface that let the user build the UML diagram supported by the SiSaS transformations. Unfortunately, the format used by Enterprise Architect to store UML models in not natively supported by the transformation engine embedded in the SiSaS Studio. The resulting models hence have to be converted using the "Convert to EMF/UML" feature of the SiSaS Studio popup menu (as shown on Figure 4.1 below). The resulting models can be instantiated using any kind of project templates, but can also be visualized in the Papyrus editor.

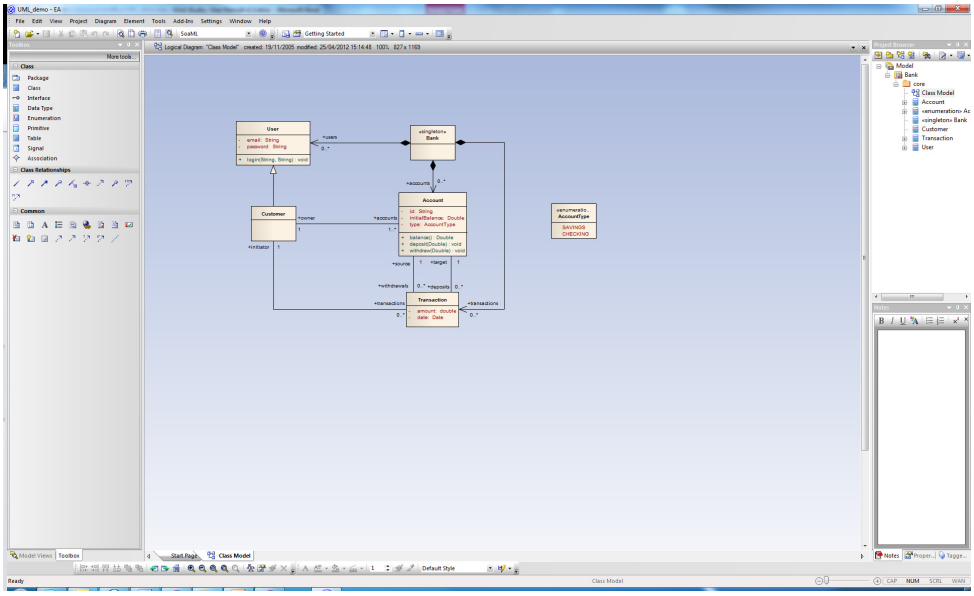

Figure 3.2 Modelling in Enterprise Architect

3.2 Modelling with Papyrus UML

An alternative modelling environment is embedded in the SiSaS Studio: the Papyrus UML editor. This editor has two advantages: it is integrated within the Eclipse framework upon which the SiSas Studio is built, and it stored UML models in a format that is natively supported by internal transformation engine. However, the graphical interface of the Papyrus UML editor is probably – of course this is a subjective matter – less mature than the one provided by Enterprise Architect. Unless the model under construction includes very advanced features that would be supported by the "Convert to EMF/UML" feature, we would advise to use the Enterprise Architect modeller.

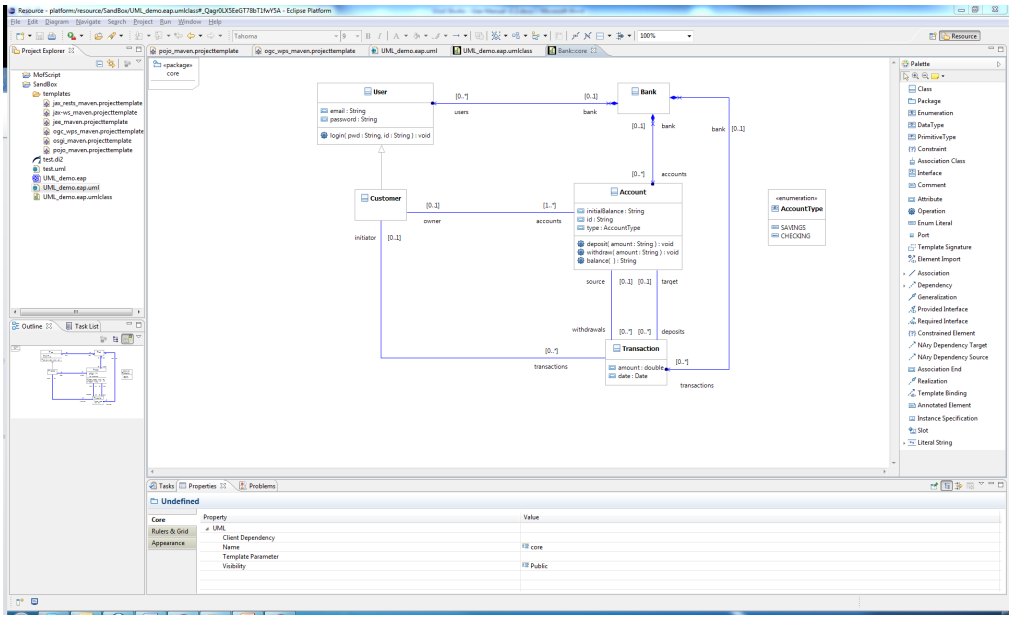

Figure 3.3 Modelling in Payprus

4 CODE GENERATION WITH THE SISAS STUDIO

Given a UML model, the SiSaS Studio permits to generate several types of application. Each type of application is described by a "project template" capturing the organization of the resulting code (in terms of directory structures and generated artefacts). Project templates may contain several generated artefacts (Java source code, XML configuration files, JSP templates, etc.) which will all be generated when instantiating a given template on a given UML model.

Internally, the SiSaS Studio encompasses many models transformations, which, given a UML model, produces a specific types of code. When instantiating a specific project template, the SiSaS Studio will browse the selected templates, looking for all artefacts that must generated, and trigger the proper transformations.

The instantiation of project templates can be trigger from the SiSaS Studio popup menu, as shown on Figure 4.1 below. For the record, the SiSaS popup menu appears, when users right-click on any type of file supported by the SiSaS Studio (Papyrus UML models whose name ends with".uml" in the current version). Each menu entry that starts with "Build" reflects one project template that can be instantiated. In Figure 4.1 for instance, there is only one project template available which permit to generate *Plain Old Java Objects* (POJO).

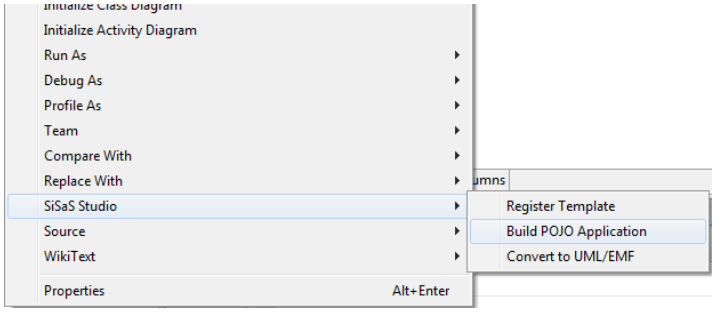

Figure 4.1 The SiSaS Studio popup menu and the instantiation of project templates

This section reviews the five main templates that are provided with the SiSaS Studio, and describes, for each of them, the organization the code that will be generated.

4.1 Building a "Plain Old" Java Application

The expression *"Plain Old Java Objects* (POJO) applications" stands for standard Java applications that do not require any specific framework or middleware to run. In that sense, Spring applications, JEE applications, OSGi bundles, *etc*. are not POJO applications.

The SiSaS Studio includes a specific project template that can used to generate such applications from UML models. This project template is illustrated by Figure 4.2 below.

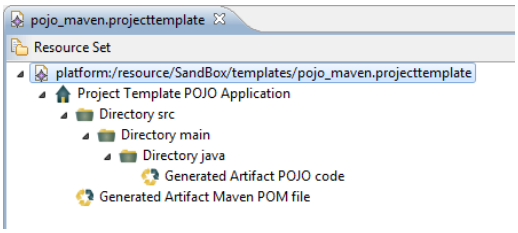

Figure 4.2 Structure of POJO project template

The code resulting from this template is actually a Maven [13] project. As any Maven project, the resulting project includes a directory named "`src`" containing the Java sources and a project descriptor named "`pom.xml`". All Java sources files will be place in the "`src/main/java`" and properly organized following the package structure described in the UML model.

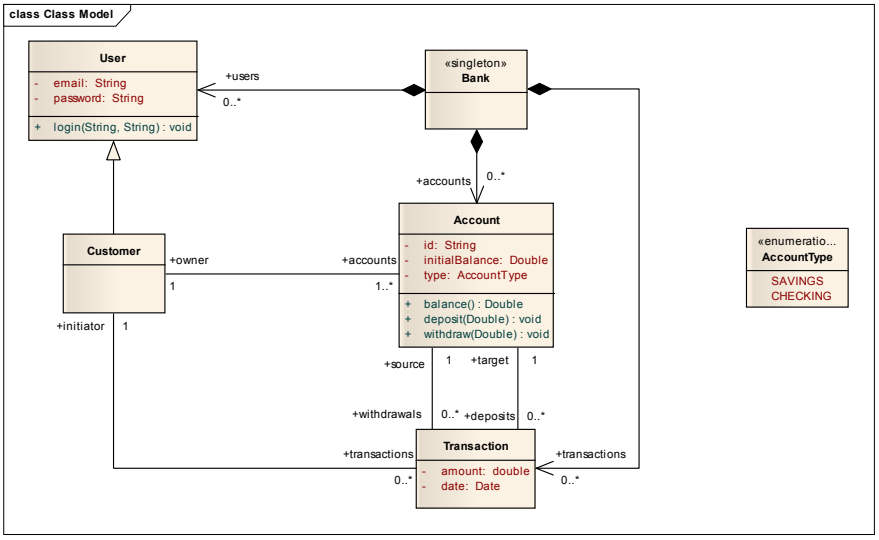

Figure 4.3 A sample Banking application, modelled as a UML diagram in Enterprise Architect

In this example we generate code from a UML model describing a simple banking application, which contains 6 classes: Account, AccountType, Bank, Customer, Transaction and User. The relationships between these classes are depicted by Figure 4.3 as a UML class diagram. Assuming that we need to generate the corresponding Java code in the "C:\temp\sisas\demo-java" directory, we obtain the following structure:

```
c:\temp\sisas\demo-java>tree /F
Folder PATH listing
Volume serial number is E215-6ABF
C:.
```



```
|   pom.xml
|
|___src
    |___main
        |___java
            |___bank
                |___core
                        Account.java
                        AccountType.java
                        Bank.java
                        Customer.java
                        Transaction.java
                        User.java
```

Once the code is generated, Maven permits to automatically compiles and package the application, as shown in the following example. The reader interested in using Maven may refer to [13] for exhaustive tutorial.

```
c:\temp\sisas\demo-java>mvn package
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] Building Bank 1.0-SNAPSHOT
[INFO] ------------------------------------------------------------------------
[INFO]
[INFO] --- maven-resources-plugin:2.4.3:resources (default-resources) @ Bank ---

[WARNING] Using platform encoding (Cp1252 actually) to copy filtered resources,
i.e. build is platform dependent!
[INFO] skip non existing resourceDirectory c:\temp\sisas\demo-java\src\main\reso
urces
[INFO]
[INFO] --- maven-compiler-plugin:2.3.2:compile (default-compile) @ Bank ---
[WARNING] File encoding has not been set, using platform encoding Cp1252, i.e. b
uild is platform dependent!
[INFO] Compiling 6 source files to c:\temp\sisas\demo-java\target\classes
[INFO]
[INFO] --- maven-resources-plugin:2.4.3:testResources (default-testResources) @
Bank ---
[WARNING] Using platform encoding (Cp1252 actually) to copy filtered resources,
i.e. build is platform dependent!
[INFO] skip non existing resourceDirectory c:\temp\sisas\demo-java\src\test\reso
urces
[INFO]
[INFO] --- maven-compiler-plugin:2.3.2:testCompile (default-testCompile) @ Bank
---
[INFO] No sources to compile
[INFO]
[INFO] --- maven-surefire-plugin:2.7.2:test (default-test) @ Bank ---
[INFO] No tests to run.
[INFO] Surefire report directory: c:\temp\sisas\demo-java\target\surefire-report
s

-------------------------------------------------------
 T E S T S
-------------------------------------------------------
There are no tests to run.

Results :

Tests run: 0, Failures: 0, Errors: 0, Skipped: 0

[INFO]
[INFO] --- maven-jar-plugin:2.3.1:jar (default-jar) @ Bank ---
[INFO] Building jar: c:\temp\sisas\demo-java\target\Bank-1.0-SNAPSHOT.jar
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 2.018s
[INFO] Finished at: Wed Jun 13 11:58:21 CEST 2012
[INFO] Final Memory: 10M/243M
[INFO] ------------------------------------------------------------------------
```

Once Maven has compiled and packaged the source, we obtain a new directory called "target", containing all the compiled class, plus a release of our project as a JAR file entitled "Bank-0.1-SNAPSHOT.jar", which can be distributed.

```
c:\temp\sisas\demo-java>tree /F
Folder PATH listing
Volume serial number is E215-6ABF
C:.
|   pom.xml
|
+---src
|   \---main
|       \---java
|           \---bank
|               \---core
|                       Account.java
|                       AccountType.java
|                       Bank.java
|                       Customer.java
|                       Transaction.java
|                       User.java
|
\---target
    |   Bank-1.0-SNAPSHOT.jar
    |
    +---classes
    |   \---bank
    |       \---core
    |               Account.class
    |               AccountType.class
    |               Bank.class
    |               Customer.class
    |               Transaction.class
    |               User.class
    |
    +---maven-archiver
    |       pom.properties
    |
    \---surefire
```

4.2 BUILDING A JEE 3 APPLICATION

4.3 BUILDING WEB SERVICES

4.4 BUILDING REST SERVICES

4.5 BUILDING OGC/WPS APPLICATIONS

5 TUTORIALS

5.1 CREATING CUSTOM PROJECT TEMPLATES

The SiSaS Studio let you define your own project templates so as to customize the way you want the generated code to be organized. The definition of a new template is a two-step process:

- Create a new Project Template (File → New → Others → Example EMF Wizards →Project Template)
  - The project template editor let you add new element by right-clicking and selecting the type of child element you want to append. Figure 5.1 below illustrate the use of the Project Template Editor.

- Project Templates are made of two types of elements: "Generated Artefacts" and "Directories". Generated Artefacts are artefacts that will be generated when the template will be instantiated. They ***must*** have an "Artefact type" indicating which transformation can be applied on them. Directories represent the directory structure containing the generated artefacts.

- To make the template available in the SiSaS Studio popup menu, you must register this new template, by using the "Register Template" entry of the SiSaS Studio popup menu. It is worth to note that templates are registered only during the current session.

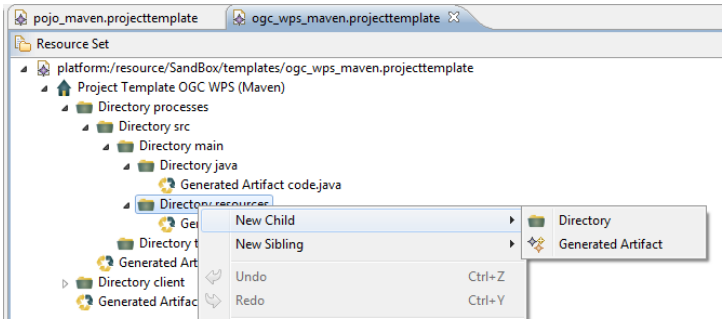

Figure 5.1 Editing Project templates

5.2 Developing New Transformations

The SiSaS Studio can be extended with new transformations if the provided one does not cover your needs. However, such extension is consider as further development of the Studio itself, and does not really fall within the scope of this document. Adding new transformations, debugging existing ones, or adding new templates are detailed in the developer manual.

6 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.

