

**UNIVERSITY OF INSUBRIA**

**DEPARTMENT OF THEORETICAL AND  
APPLIED SCIENCES**

**MASTER'S DEGREE IN COMPUTER SCIENCE**



**Design and development of applications related to  
system engineering activities of astronomical  
instrumentation with SysML**

**SUPERVISOR: Prof. Alberto Trombetta**

**GRADUATE: Alessandro Giovannacci**

**ACADEMIC YEAR: 2019-2020**

## Summary

In this experimental thesis, it is requested to develop a software system that simplifies the generation of a specific file in the “Cameo Systems Modeler” context: the report.

Cameo Systems Modeler (CSM) is a software, developed by No Magic, used by system engineers to model a system with SysML, a modeling language which is an extension of a subset of UML.

System engineers of INAF (Istituto Nazionale di Astrofisica) are currently working on the MAORY project, one of the first light instruments for the E-ELT, the European Extremely Large Telescope.

Engineers periodically have to generate a Requirement report of various subsystems of the entire project: this document contains a subset of all requirements, and must be formatted in a specific way, in order to be as concise and clear as possible.

The first task is the creation of a Velocity template, which is a specific file that contains a VTL script. Velocity is a Java-based template engine that provides a template language to reference objects defined in Java code. The language can be used in order to specify how a Cameo Systems Modeler report must be created, specifying the properties of the file by means of scripts.

In order to create the template, the SysML project structure has to be analyzed, in particular Requirements diagrams of each package, and the relations that these have with other diagrams.

Once the template file has been created, the configuration of the Report Wizard is fundamental, because the entire report generation process can be customized, specifying the scope from which the diagram extrapolation must begin, as well as setting the Report data, a collection of report which can be included in the generated report, and output options.

The last step is the implementation of a Cameo Systems Modeler plugin, using the Java language. CSM is mainly written in Java, and developers made available a specific Application Programming Interface (API) through which extensions can be written and included in the software.

The plugin simplifies and improve reports generation’s process, and once the configuration through Report wizard is complete, the plugin can generate the required reports as a background process, thus allowing users to perform other tasks in the meanwhile. The process can be a time-consuming task, in function of the scope and the number of requirements included, and by default it is set as a foreground task in the CSM context. Furthermore, the MAORY project is built as a Teamwork cloud project, which allows collaborative development and version model storage, so the only way to generate reports as a separate task might be outside the main Cameo Systems Modeler interface, using the command line interface; in this way, the process could be automated and users could perform other tasks in the meantime.

Unfortunately, this approach has disadvantages because of the cloud nature of the project, and for this reason this solution works only with local projects. The plugin represents the optimal solution, and it is implemented to run efficiently allowing the user to do other jobs during the report generation.

# Contents

Summary .....	2
1 Context.....	4
1.1 INAF.....	4
1.2 ESO .....	4
1.3 MAORY project.....	5
2 Software, languages and technologies involved.....	8
2.1 Cameo Systems Modeler .....	9
2.2 SysML, Systems Modeling Language .....	10
2.3 Report wizard.....	12
2.4 Velocity Template .....	12
2.5 Velocity Template Language .....	12
3 Implementation .....	13
3.1 Template's creation .....	13
3.1.1 Problem analysis .....	13
3.1.2 Creation of template file using VTL.....	18
3.1.3 Requirements.....	22
3.2 Report Wizard initial configuration.....	32
3.3 Reports Generation plugin.....	38
3.3.1 First approach .....	38
3.3.2 A more efficient and scalable solution .....	40
3.3.3 Plugin implementation.....	41
3.3.4 Plugin structure.....	46
3.3.5 Compilation of Java files .....	57
4 Bibliography .....	59

# 1 Context

## 1.1 INAF

The National Institute for Astrophysics (Istituto Nazionale di Astrofisica, INAF) is an Italian research institute in astronomy and astrophysics, founded in 1999.

The research performed covers most areas of astronomy, ranging from planetary science to cosmology.

INAF also maintains a close cooperation with other organisms who perform astronomic research in Italy and abroad, in particular with the National Institute of nuclear astrophysics (INFN), Italian Space Agency (ASI), European Space Agency (ESA), and National Aeronautics and Space Administration (NASA).

INAF researchers obtained great results in diverse fields within astronomy and astrophysics, and using various instruments. An example is the satellite AGILE, that has led to important results on the high energy Universe and the study of black holes.

The institute also claims international collaborations, one of the most important is related to the LBT (Large Binocular Telescope), the largest of its type, and amongst the most advanced in the world. VIMS and VIRTIS are two instruments designed and built by INAF; the first operates on the Cassini probe that has been revealing the mysteries of Saturn and its moons since 2004. VIRTIS is busy on ESA's Venus Express probe, designed particularly to study the greenhouse effect on this planet, something that is fundamental for our understanding of how climate change and non-reversible effects could result from careless behavior on the Earth. ESA's Planck satellite, thanks to its LFI instrument built by a team of researchers coordinated by INAF, has made the best map ever of the cosmic background radiation, emitted 13.7 billion years ago.

INAF is also working, with other nations, on the construction of large research infrastructures like the SRT (Sardinia Radio Telescope), and the LBT (Large Binocular Telescope) in Arizona. It maintains Galileo telescope in the Canaries and has completed the construction of the VST (VLT Survey Telescope), and contributes to the upgrade of scientific instrumentation at the Very Large Telescope.

## 1.2 ESO

The European Organization for Astronomical Research in the Southern Hemisphere is a 16-nation intergovernmental research organization for ground-based astronomy. Its observatories are located in northern Chile.

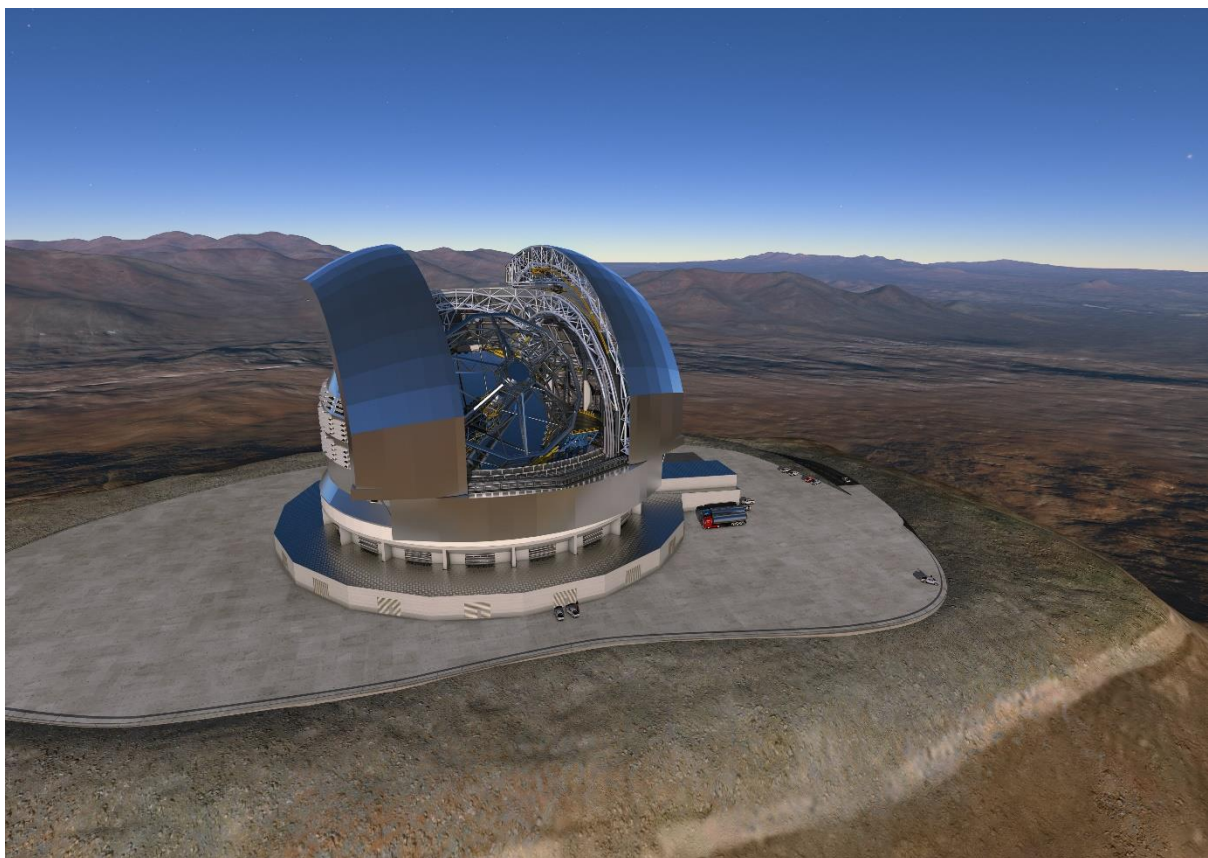
ESO's main mission, laid down in the 1962 Convention, is to provide state-of-the-art research facilities to astronomers and astrophysicists, allowing them to conduct front-line science in the best conditions.

ESO has built and operated some of the largest and most technologically advanced telescopes, including the 3.6 m New Technology Telescope, the Very Large Telescope, and the Atacama Large Millimeter Array.

ESO's observing facilities have made astronomical discoveries and produced several astronomical catalogues. Its findings include the discovery of the most distant gamma ray burst and evidence for a black hole at the center of the Milky Way. In 2004, the VLT allowed astronomers to obtain the first picture of an extrasolar planet orbiting a brown dwarf 173 light-years away. The High Accuracy Radial Velocity Planet Searcher instrument installed on the older ESO 3.6 m telescope led to the discovery of extrasolar planets.

### 1.3 MAORY project

The creation of a report, a template and the plugin has been carried on working with a specific project, MAORY, which will be part of the E-ELT (European Extremely Large Telescope), a telescope ESO (European Southern Observatory) is building. It is expected E-ELT will start its visual space exploration in 2024. It is a 39 m diameter optical-infrared telescope that ESO is building in collaboration with its community of member States. It is located in Chile, on Cerro Armazones, 3000 m above sea level.

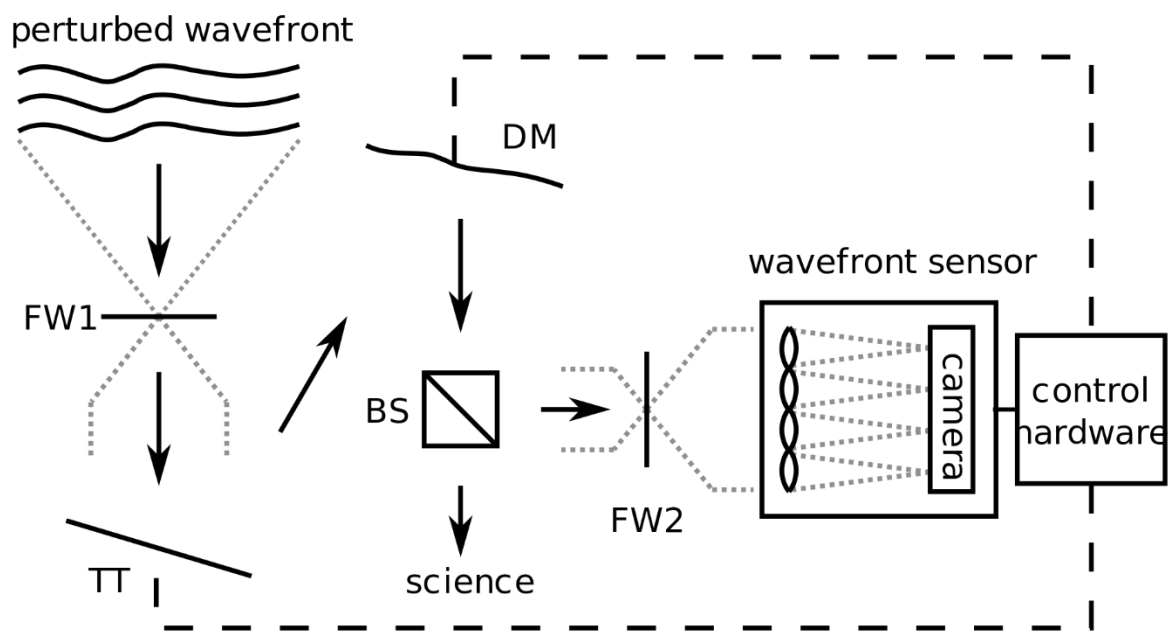


eso1617a - Cerro Armazones, Extremely Large Telescope, ELT Illustrations

URL: <https://www.eso.org/public/images/0919-e-elt-big-cc/>

MAORY stands for “Multi-conjugate Adaptive Optics RelaY”, and it is one of the first light instruments for the E-ELT.

Adaptive optics play a crucial role as part of a large telescope; it is a technology used to improve the performance of optical systems by reducing the effect of incoming wavefront distortions by deforming a mirror in order to compensate for the distortion. This technology is used in astronomical telescopes and laser communication systems to remove the effect of atmospheric distortion.

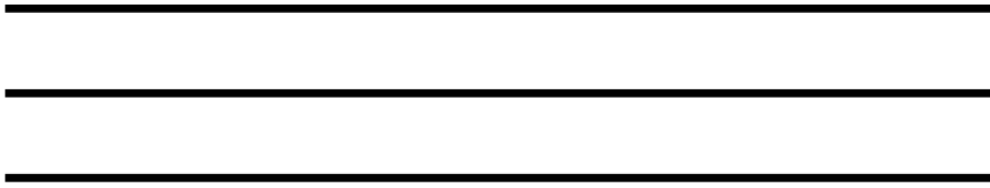


The light first hits a tip-tilt (TT) mirror and then a deformable mirror (DM) which corrects the wavefront. Part of the light is tapped off by a beamsplitter (BS) to the wavefront sensor and the control hardware which sends updated to the DM and TT mirrors.

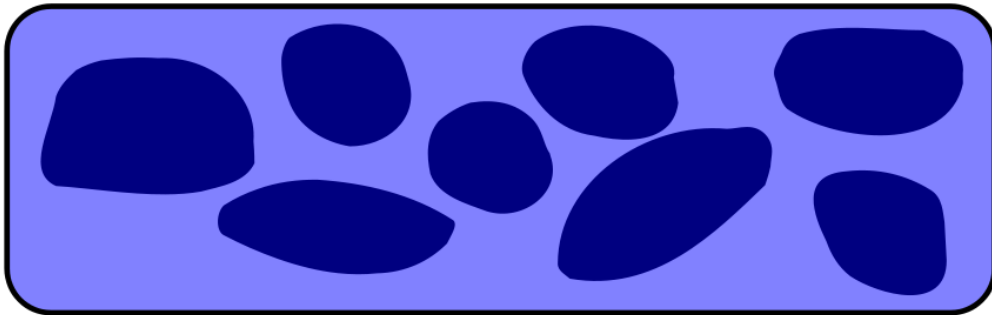
URL:[https://en.wikipedia.org/wiki/Adaptive\\_optics#/media/File:Adaptive\\_optics\\_system\\_full.svg](https://en.wikipedia.org/wiki/Adaptive_optics#/media/File:Adaptive_optics_system_full.svg)

It is mandatory to include an adaptive optics system in any telescope larger than approximately 20 cm, because visual images produced by these telescopes are blurred by distortions. When light from an astronomical object enters the Earth’s atmosphere, atmospheric turbulence can distort and move the image in various ways.

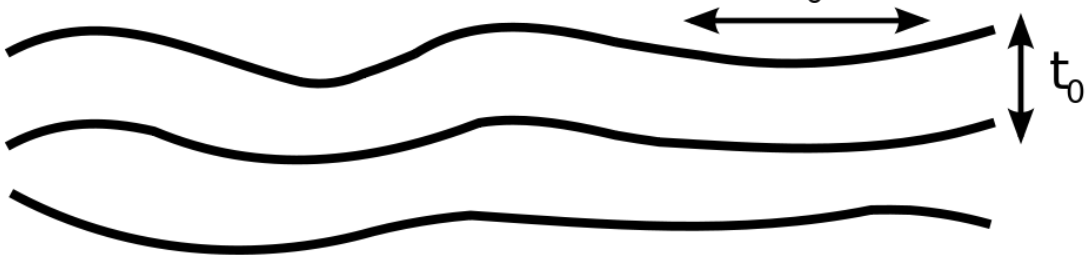
Plane wavefront



Inhomogeneous medium



Perturbed wavefront  $r_0$

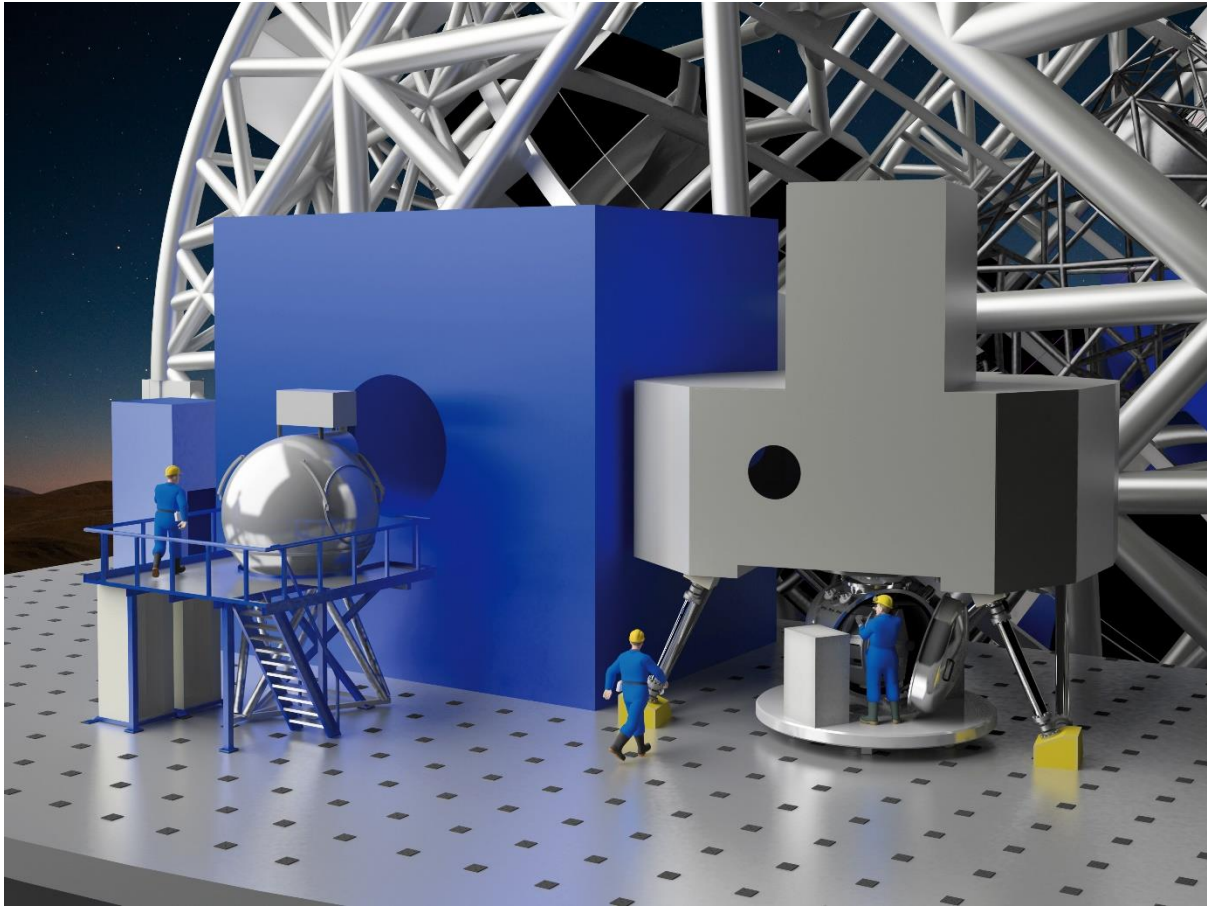


Light from a star passes through Earth's atmosphere, and the wavefront is perturbed.

URL: [https://en.wikipedia.org/wiki/Adaptive\\_optics#/media/File:Atmospheric\\_seeing\\_r0\\_t0.svg](https://en.wikipedia.org/wiki/Adaptive_optics#/media/File:Atmospheric_seeing_r0_t0.svg)

MAORY is a post-focal adaptive optics module. It supports the MICADO near-infrared camera by offering two adaptive optics modes: Multi-Conjugate Adaptive Optics (MCAO) and Single-Conjugate Adaptive Optics (SCAO).





ar-metis-maory-micado – Extremely Large Telescope, MAORY, METIS (Mid-infrared ELT Imager), MICADO  
 URL: <https://www.eso.org/public/images/ar-metis-maory-micado/>

In the MCAO mode, wavefront sensing is performed by a system based on up to six Laser Guide Stars (LGS) for high-order wavefront sensing and three Natural Guide Stars (NGS) for low-order wavefront sensing. The six LGS are produced by excitation of the atmospheric sodium layer (at an altitude of ~90 km) with six laser beacons (with  $\lambda = 589.2$  nm) propagated from E-ELT.

Wavefront compensation is achieved by up to two deformable mirrors in MAORY, which work together with the telescope adaptive and tip-tilt mirrors M4 and M5 respectively. In the SCAO mode, wavefront distortions are measured by a single NGS wavefront sensor and compensated by the telescope M4 and M5 mirrors, while the deformable mirrors inside MAORY are kept at their reference shape.

MAORY is supposed to operate at ambient temperature. The operating temperature range is 0°C-15°C (AD1).

## 2 Software, languages and technologies involved

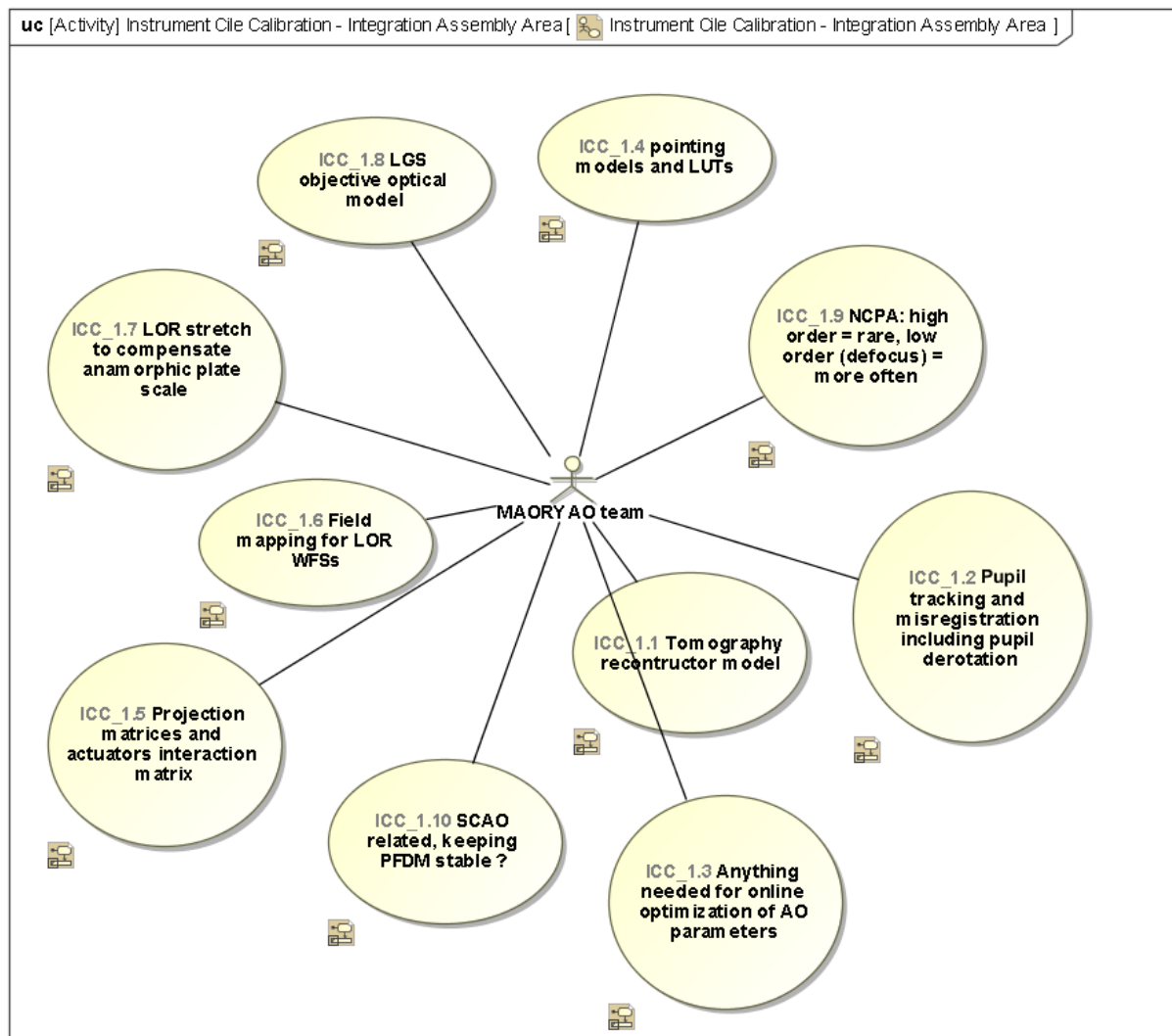


## 2.1 Cameo Systems Modeler

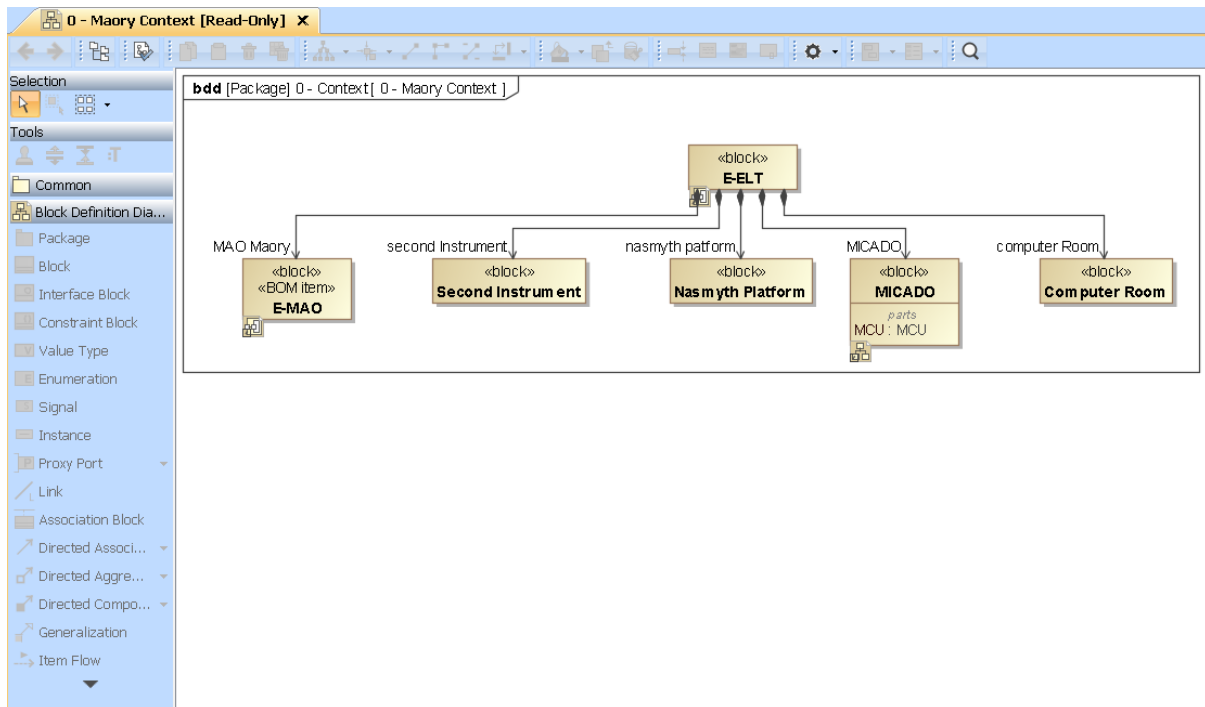
Cameo Systems Modeler is an industry leading cross-platform collaborative Model-Based Systems Engineering (MBSE) environment, which provides smart, robust, and intuitive tools to define, track, and visualize all aspects of systems in the most standard-compliant SysML models and diagrams. The environment enables systems engineers to:

- run engineering analysis for design decisions evaluation and requirements verification;
- continuously check model consistency;
- track design progress with metrics.

System models can be managed in remote repositories, stored as standard XMI files, or published to documents, images, and web views to address different stakeholder concerns.



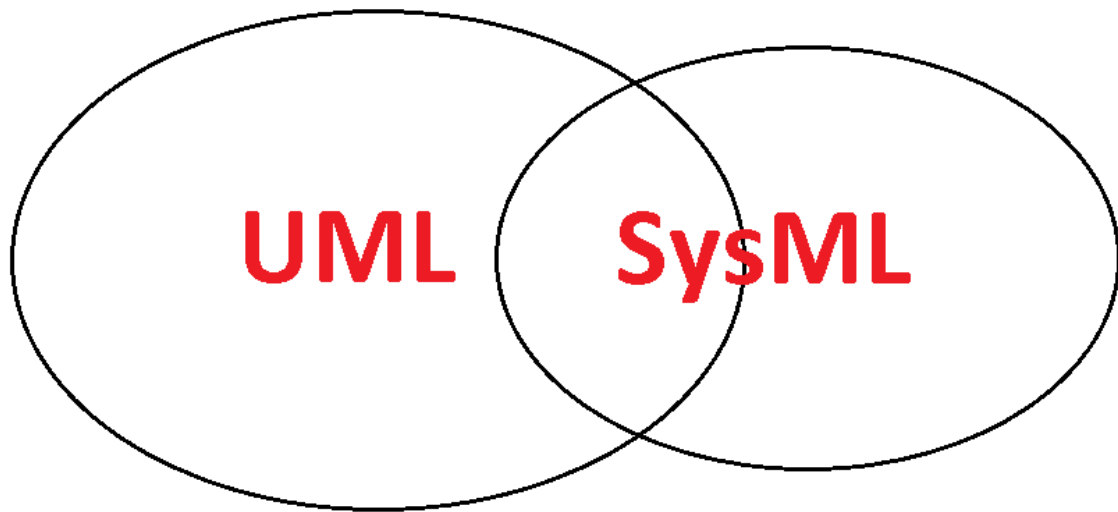
An example of Use Case diagram from Cameo Systems Modeler (MAORY project).



Another example taken from Cameo Systems Modeler. SysML Block Definition diagram (MAORY project).

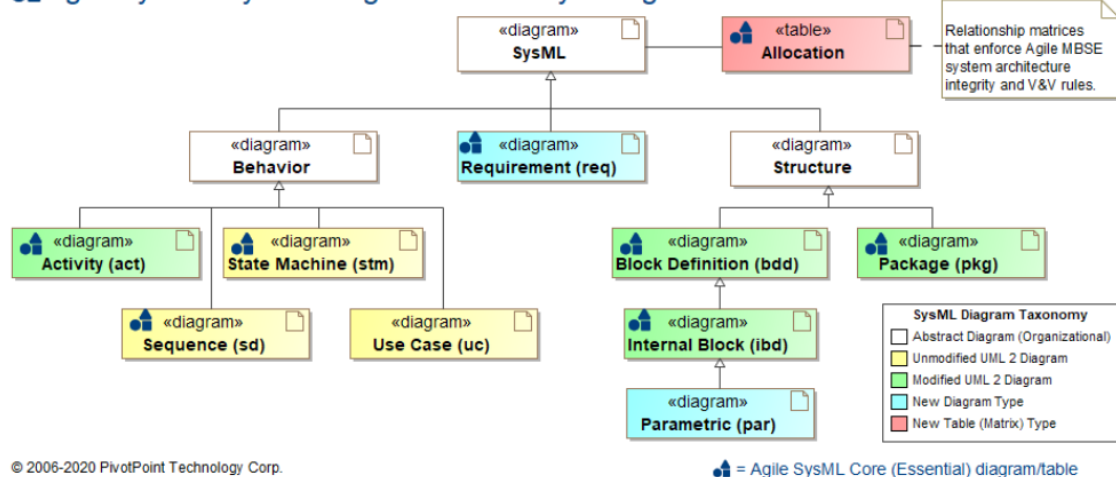
## 2.2 SysML, Systems Modeling Language

The Systems Modeling Language is a general-purpose modeling language for systems engineering applications. It supports the specification, analysis, design, verification and validation of a broad range of systems and systems-of-systems.



SysML is defined as an extension of a subset of the Unified Modeling Language (UML) using UML's profile mechanism. The language's extensions were designed to support systems engineering activities.

#### Agile SysML: SysML Diagram Taxonomy for Agile MBSE methods



URL: <https://sysml.org/>

There are several differences/improvements compared to UML, the main ones are the following:

- its semantics are more flexible and expressive. SysML reduces UML's software-centric restrictions and adds two new diagram types, requirement and parametric diagrams. The former can be used for requirements engineering; the latter can be used for performance analysis and quantitative analysis;
- it is a comparatively small language that is easier to learn and apply. Since SysML removes many of UML's software-centric constructs, the overall language is smaller both in diagram types and total constructs;
- SysML model management constructs support models, views, and viewpoints. These constructs extend UML's capabilities and are architecturally aligned with IEEE-Std-1471-2000.

## 2.3 Report wizard

Report wizard is a report engine for No Magic products like MagicDraw or Cameo Systems Modeler.

It is made on top of Velocity Engine, which is an open source templating engine, and it is integrated with No Magic application.

Report Wizard supports text-based templates to generate reports. Each report file format depends on the type of the templates. The type of template files that Report Wizard supports includes HTML, XML template and plain text.

## 2.4 Velocity Template

A template, in Velocity, is a text file that tells Velocity what the output should look like. It contains a document page style, layout, header, footer, and static text just like any other template that comes with most of the Word Processor programs. However, a Velocity template also contains specific placeholders for Java objects and Velocity Template Language scripts which will tell the Velocity Engine what and how to print the information in the output.

## 2.5 Velocity Template Language

Apache Velocity is a Java-based template engine that provides a template language to reference objects defined in Java code.

Velocity is an open source software project hosted by the Apache Software Foundation.

Velocity language can be used for web development, as well as for other purposes: it can be used to generate SQL, PostScript and XML from templates. It can be used either as a standalone

utility for generating source code and reports, or as an integrated component of other systems. For instance, Velocity provides template services for various web frameworks, enabling them with a view engine facilitating development of web applications according to a true MVC model.

As an example of Velocity script:

```
#foreach($instance in $InstanceSpecification)
    #foreach($slot in $instance.slot)
        #if($slot.definingFeature.name == 'Name')
            #if($slot.value.size() > 0)
                #set($v = $slot.value.get(0).text)
            #else
                #set($v = "")
            #end
            Slot value is $v
        #end
    #end
#end
```

## 3 Implementation

### 3.1 Template's creation

#### 3.1.1 Problem analysis

The first task to take into consideration is the creation of the template file.

Template file is the most important resource when customizing a report, because it is the file which contains all the information that the report engine will take in input to generate the report.

What is required is a specific requirement report, that is a specific document with a specific formatting style. This document must contain all requirements of a specific “element scope”.

An example of requirement tabular representation: INAF custom requirement report file must represent a requirement with this structure, table and diagram.

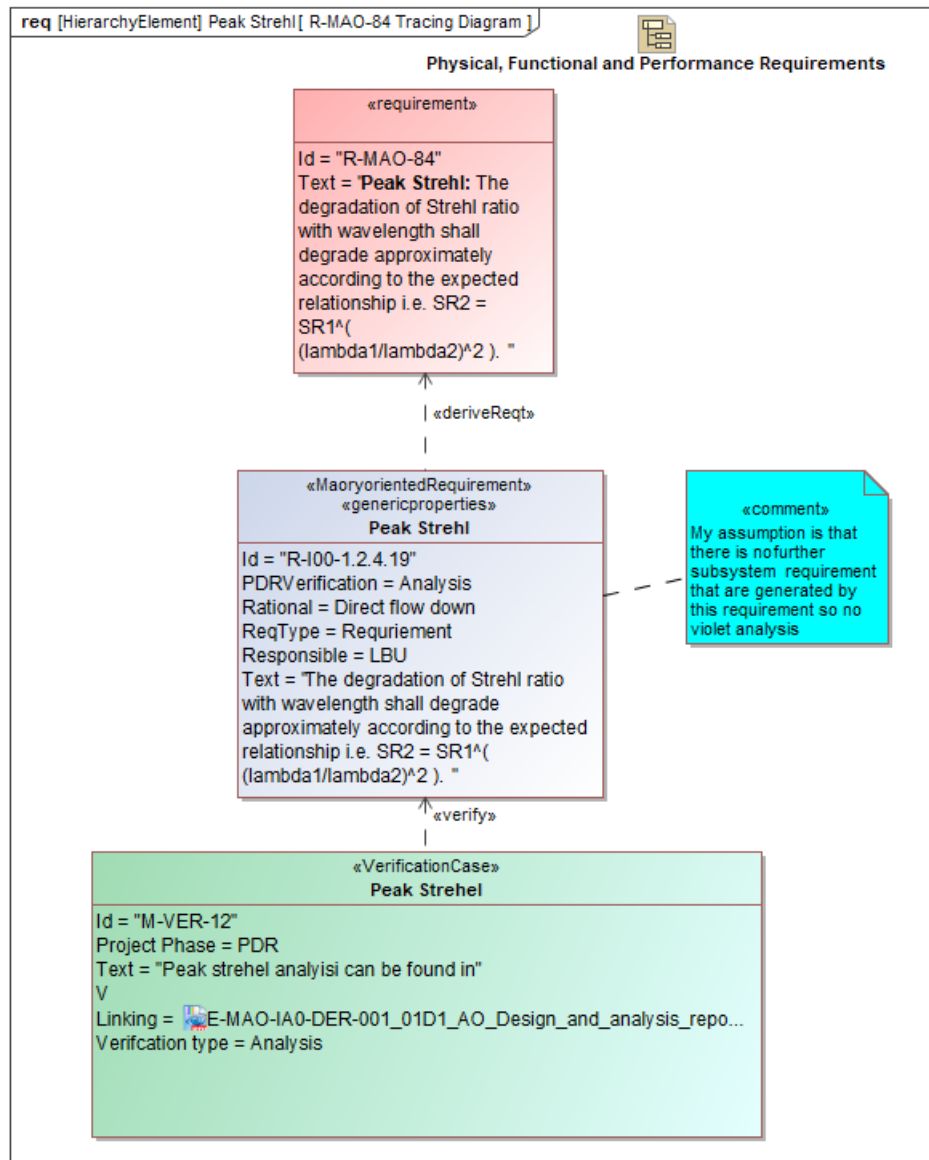
## 1.1 R-I00-1.1: Environmental conditions

Root for all I00-1 Environmental conditions requirements

<b><i>R-I00-1.1.1 : General applicable conditions</i></b>	
<b>Responsible</b>	FCO
<b>Requirement Type</b>	<i>Requirement</i>
<b>Description</b>	Unless otherwise specified all the requirements in this document and its applicable documents shall be met under the provisions specified in section 4 of ESO-254547.
<b>Rationale</b>	Direct flow down
<b>Parent Analysis</b>	
<b>Parent Requirement</b>	<b><i>R-MAO-51</i></b> - NA
<b>Refining analysis</b>	<b><i>M-ANA-1.1</i></b> - General applicable conditions
<b>Children requirements</b>	<b><i>MAO-ISO-1.1.1</i></b> - General applicable conditions <b><i>MAO-IH0-1.1.1</i></b> - General applicable conditions <b><i>MAO-IM0-1.1.1</i></b> - General applicable conditions <b><i>MAO-IT0-1.1.1</i></b> - General applicable conditions <b><i>MAO-IO0-1.1.1</i></b> - General applicable conditions <b><i>MAO-IP0-1.1.1</i></b> - General applicable conditions <b><i>MAO-IU0-1.1.1</i></b> - General applicable conditions <b><i>MAO-IN0-1.1.1</i></b> - General applicable conditions <b><i>MAO-IL0-1.1.1</i></b> - General applicable conditions <b><i>MAO-IR0-1.1.1</i></b> - General applicable conditions <b><i>MAO-ID0-1.1.1</i></b> - General applicable conditions
<b>Verification analysis</b>	
<b>Notes</b>	[]

R-I00-1.1.1 of Environmental conditions requirements.





Requirement diagram of R-I00-1.2.4.19.

Report will also have to show a first page with this specific formatting:



Programme: **E-ELT**

Project: **ELT MCAO Construction – MAORY**

## **MAORY REQUIREMENT FLOW DOWN**

**Document Number:** E-MAO-000-INA-TNO-007

**Document Version:** 1D1

**Document Type:** [TNO](#)

**Released On:** 2018-10-29

**Owner :** Marco Riva

**Approved by PI:**

**Released by PM:**



Document Classification: MAORY Consortium Internal [Confidential for Non-MAORY Staff]

Regarding the second page, it will contain two tables:

## Authors

Name	Affiliation

## Change Record from previous version

Date	Affected Section(s)	Changes / Reason / Remarks
	All	First issue

The third page of the document contains a “Table of contents”:



## Contents

1	<i>R-I00-1: tech spec requirements</i>	4
1.1	R-I00-1.1: Environmental conditions	4
1.2	R-I00-1.2: Physical, Functional and Performance Requirements	6
1.2.1	R-I00-1.2.2: Physical Characteristics	8
1.2.2	R-I00-1.2.3: General Requirements	10
1.2.3	R-I00-1.2.4: Performance Requirements	30
1.3	R-I00-1.3: Interfaces	91
1.3.1	R-I00-1.3.3: Mechanical Interfaces	93
1.3.2	R-I00-1.3.4: Services Interfaces	100
1.4	R-I00-1.4: Access and Handling	108
1.5	R-I00-1.5: Verification requirements	114
2	<i>R-I00-2: common requirement</i>	116

The structure described above represents the required output, this means that every time a user selects this specific report for generating requirements, the document created will have this structure.

### 3.1.2 Creation of template file using VTL

The version of Cameo used for implementing and testing the template is “Cameo Systems Modeler Demo”; the demo version is a sort of “Trial version” because it has some limitations, specifically concerning the number of “objects” that can be visualized and manipulated: in this context, object refers to specific diagrams. However, for the purpose of implementing and testing the template, this version is adequate, as for this task there will be no limitations.

The first task is to understand what variables have to be created with the “Report data”.

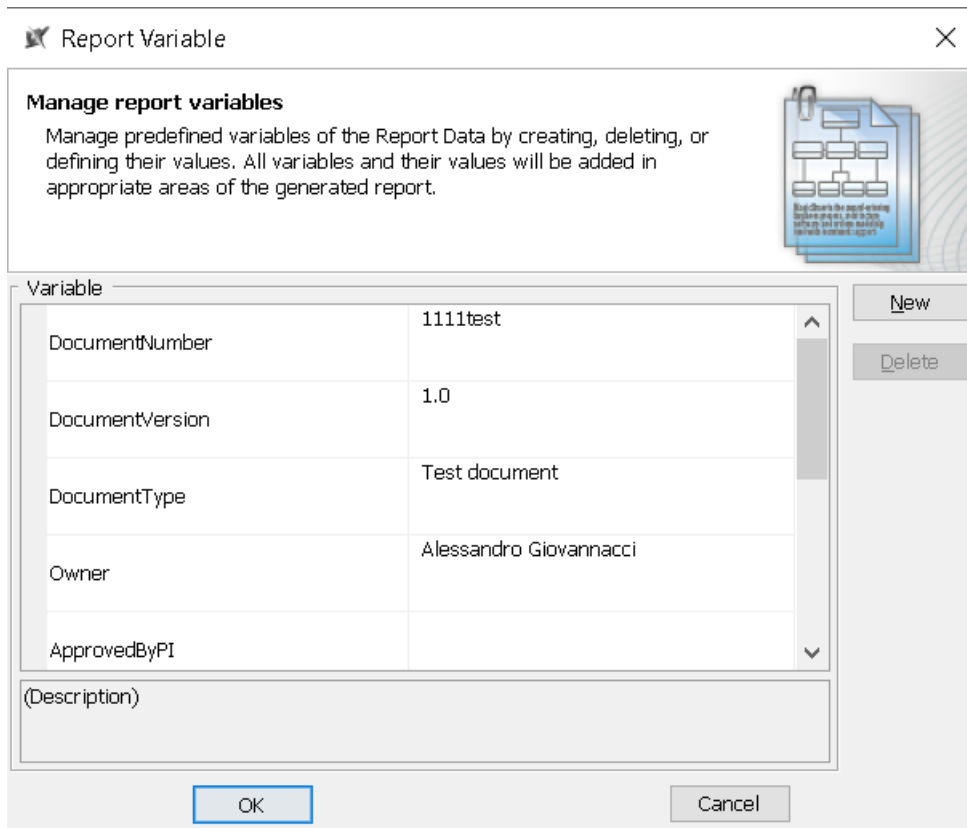
Analyzing the first and second page of the example document, it is evident that title of the project, as well as the other data like “Document number”, “Owner” or “Authors” must be “ad hoc” variables created in a specific report data (Report Wizard configuration).

Template file, as a VTL script file, can get value of the variables simply writing the variables’ name.

A variable in VTL has this form:

*\$var*

If the value of *\$var* is 3, by writing *\$var* in the VTL file the engine will generate the report and value 3 will be showed in the output.



The dialog box is titled "Report Variable" and contains a section "Manage report variables" with a descriptive text and a small icon of a document with a tree structure. Below this is a table with five rows of predefined variables. To the right of the table are "New" and "Delete" buttons. At the bottom are "OK" and "Cancel" buttons.

**Report Variable**

**Manage report variables**  
Manage predefined variables of the Report Data by creating, deleting, or defining their values. All variables and their values will be added in appropriate areas of the generated report.

Variable	
DocumentNumber	1111test
DocumentVersion	1.0
DocumentType	Test document
Owner	Alessandro Giovannacci
ApprovedByPI	

(Description)

New Delete

OK Cancel

Variables are case-sensitive, so the letters after the “\$” symbol must exactly match with the letters of the variable name in the report. Output is shown below:

Project logo

Programme: E-ELT Test

Project: test - E-ELT MCAO Construction - MAORY

## **Maory System model**

**Document Number:** 1111test

**Document Version:** 1.0

**Document Type:** Test document

**Released On:** September 30, 2020

**Owner:** Alessandro Giovannacci

**Approved by PI:**

**Released by PM:**

It can be noticed that there are also variables in the header of the document, these are:

*\$Programme*

*\$Project*

The header of the following pages of the document, precisely from page 2 to page n, where n is the maximum number of pages, has this structure (VTL script):

\$project.name	Doc. Number :	<u>\$DocumentNumber</u>
	Doc. Version :	<u>\$DocumentVersion</u>
	Released On :	<u>\$date.get("MMMMMM</u> <u>dd, yyyy")</u>
	Page :	Page 2 of 7



As for the table of the second page, this can be implemented with the use of arrays:

```
#set($s = $Authors.toString()) ##Authors variable from report data
#set($authors = [])
#set($authors = $s.split(","))
```

The `#set` directive is used for setting the value of a reference. A value can be assigned to either a variable reference or a property reference, and this occurs in brackets.

The left-hand side of the assignment must be a variable reference or a property reference. The right-hand side can be one of the following types:

- variable reference;
- string literal;
- property reference;
- method reference;
- number literal;
- ArrayList;
- Map.

The above script is written without a specific formatting, because the engine does not have to consider it when generating the document, but it only has to execute the instructions. In fact, the only elements that will be printed in the output document, concerning VTL, are VTL variables. The table can be formatted in this way:

**Authors**

Name	Affiliation
<code>#forrow(\$a in \$authors) \$a</code>	<code>#endrow</code>

The `#forrow` directive generates one row for each variable in `$authors` array. This means there will be generated n rows, where n is the number of authors set in the Report data.

The `SysMLReportHelper` is required in order to write not trivial VTL code, because it contains helper classes that can be used to work with SysML elements of a specific project.

```
#import('helper', 'com.nomagic.requirements.reporthelper.SysMLReportHelper')
```

Simple and complex macros can be defined. The `#macro` script element allows template designers to define a repeated segment of a VTL template. Velocimacros are very useful in a wide range of scenarios. In the template, an example of macro is the following:

```
##Macro for printing general message which indicates there is nothing to display
```

```
#macro(empty)
```

```
N/D
```

```
#end
```

This macro simply prints the text: “N/D”; it is useful because it permits changes of specific parts of text without replacing those one by one. If user decides to change a specific output, replacing “N/D” with “Nothing to display”, only the macro has to be modified.

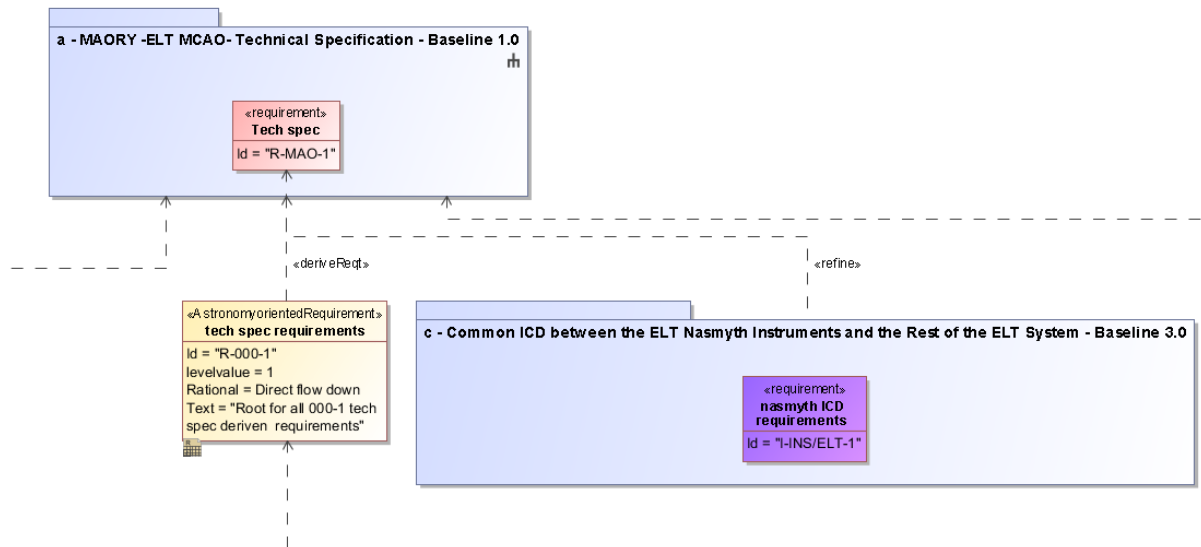
### 3.1.3 Requirements

The main problem is to extrapolate requirements from the project, from a specific “scope”.

A SysML Requirement diagram is a static structural diagram that shows the relationships among Requirement («requirement») constructs, model elements that Satisfy («satisfy» Dependency) them, and Test Cases that Verify («verify» Dependency) them.

The purpose of Requirement diagrams is to specify both functional and non-functional requirements within the model so that they can be traced to other model elements that Satisfy them and Test Cases that Verify them.

There are many requirements in the MAORY project.



The *import* directive showed in the previous paragraph enables to use object “helper”, with whom various methods can be called. VTL, in fact, is based on Java, and accesses the model which is represented by Java objects; “helper” object refers to:

*SysMLReportHelper*

In order to retrieve a requirement, all elements of a scope must be checked; the following is the method call used:

*helper.isRequirement(\$h)*

The parameter of the method contains an “element”, and it is initialized in a “for” cycle:

*#foreach(\$h in \$sorter.humanSort(\$elements, “id”))*

At every cycle, method *isRequirement* checks if the element is a requirement element, and if it is, variable *\$h* is used to get all the properties of the requirement.

```

#set($id = $h.id)
#set($name = $h.name)
#set($responsible = $h.responsible.text)
#set($reqType = $h.reqType.text)
  
```

```

#set($description = $h.text)
#set($rationale = $h.rational.text)
#set($parentAnalysis = $h.tracedTo)
#set($parentRequirement = $h.derivedFrom)
#set($refinedBy = $h.refinedBy)
#set($childrenReq = $h.derived)
##set($verifiedBy = $h.verifiedBy.get(0))
#set($fdrVerification = $h.FDRVerification.get(0).text)
#set($pdrVerification = $h.PDRVerification.get(0).text)
#set($notes = $h.notes)

```

The previous instructions create one variable for each property of the requirement. These properties are relevant because they have to be set in the requirement table.

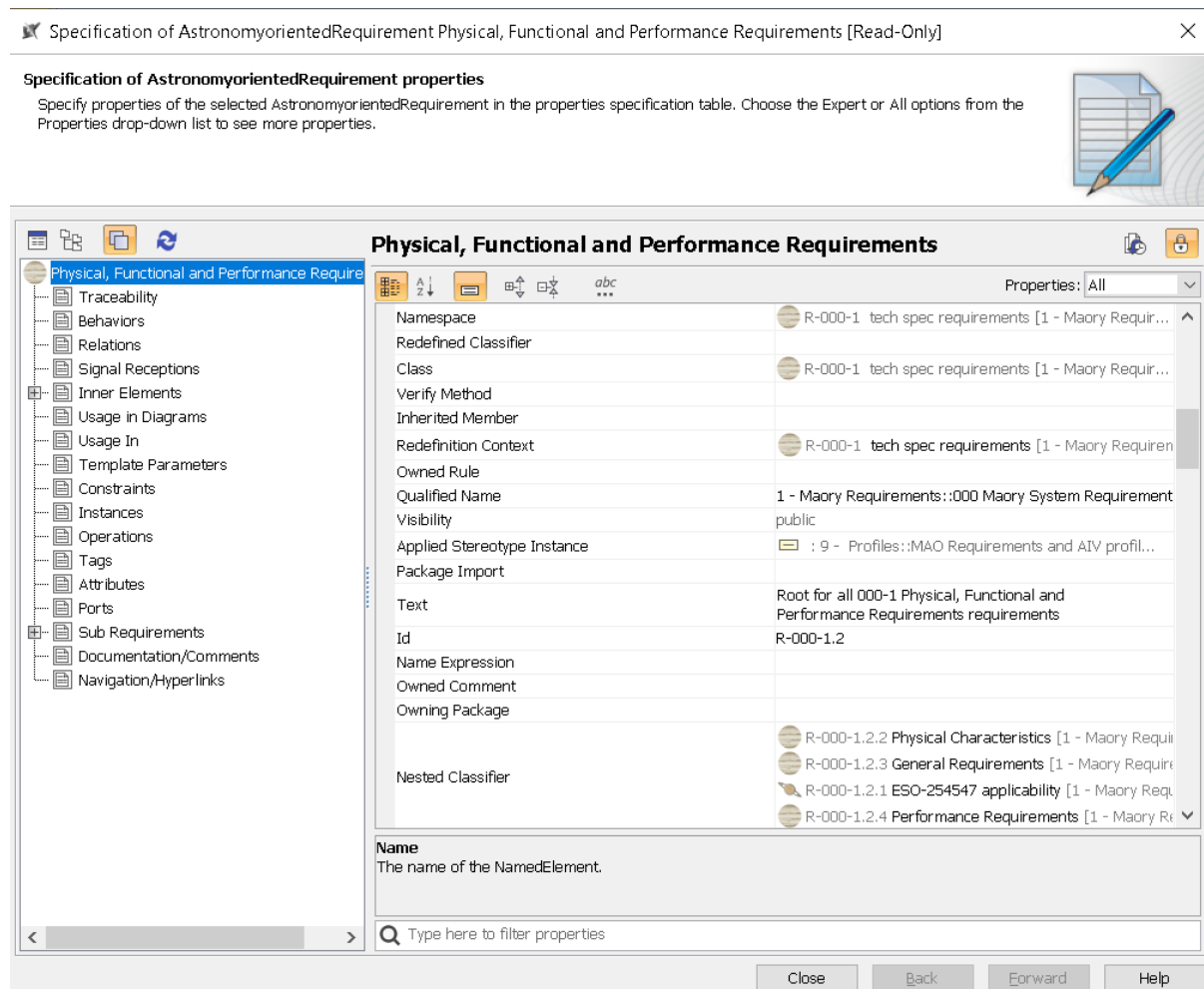
The following is the table with VTL script:

<b>\$id \$name</b>	
Responsible	\$responsible
Requirement Type	\$reqType
Description	\$description
Rationale	\$rationale
Parent Analysis	#foreach(\$el in \$parentAnalysis) \$el.id \$el.name #end
Parent Requirement	#foreach(\$el in \$parentRequirement) \$el.id \$el.name #end
Refining analysis	#foreach(\$el in \$refinedBy) \$el.id \$el.name #end
Children requirements	#foreach(\$el in \$childrenReq) \$el.id \$el.name #end
Verification analysis	##\$verifiedBy
FDR Verification	\$fdrVerification
PDR Verification	\$pdrVerification
Notes	#if(\$report.isEmpty(\$notes)) #empty() #else \$notes #end

Every property of a requirement is checked in CSM Requirement Specification window: every property in the VTL script has to be written with no spaces and with the first letter not capitalized. For example:

Verification analysis becomes verificationAnalysis

The specification window can be opened by right-clicking on a Requirement:



In this way, properties that must be included in the report can be checked. In particular, it should be noted that properties' type can be different in function of the specific property: for example, "Parent Requirement" variable's type is Java ArrayList, while "Description" variable's type is String. Dealing with ArrayList, a for-cycle is required in order to print all values:

```
#foreach($el in $parentRequirement)
```

```
$el.id $el.name
```

```
#end
```

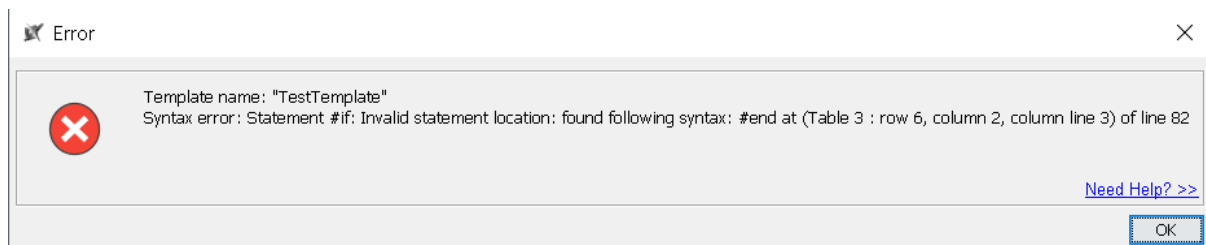
In the above example, both id and name are printed.

With VTL, variable declared in a for-cycle can be reused with the same name in another point of the script, without getting into conflict.

In general, writing VTL script directly in a specific document (that is the specific target report document type, in this case a .docx) can be tedious due to the absence of all the tools available in a modern IDE; particular attention must be paid at the syntax, even a blank space can lead to “compilation” errors, causing the interruption of the generation process.

The Velocity engine has got a notification error interface, despite not being as accurate as an interface included in a modern IDE.

In the following example, a syntax error has been reported: it is caused by a missing “#” before a “for”:



```
[2020.11.06::14:15:45] ERROR: [Report] Template name: "TestTemplate"
                        Syntax error: Statement #if: Invalid statement location: found following syntax: #end at (Table 3
                        row 6, column 2, column line 3) of line 82
```

Indeed, the notification does not report the specific syntax error, that is:

```
foreach($el in $parentAnalysis)
```

It notifies the presence of an error in an “#if” statement: in fact, the for-cycle is inside an else branch. With a complex code structure, errors identification can be tedious.

Last part of a Requirement extrapolation is the diagram, if present. Property “Member” can be checked in order to verify the presence of diagrams:

```
#foreach($el in $h.member)
#if($el.humanType.equals("Diagram"))
#set($reqDiagramString = "Requirement Diagram")
```



```
#if($report.getDiagramType($el) == $reqDiagramString)
```

The above lines of code check, for each element in variable *member*, if the element is a Diagram element, and if it is, it must be a Requirement diagram in order to be included.

If no Requirement diagrams have been found, Requirement extrapolation is completed, if instead the Requirement contains Diagram elements, these are included below Requirement table, with the name of the diagram and the image:

```
$el.name
```

```
$image.scale($el.image, 1, 0.35)
```

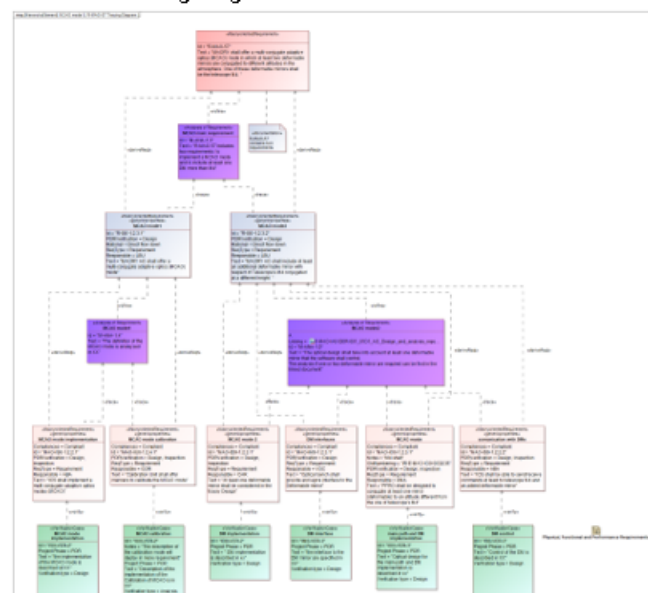
Here an example of a complete Requirement extrapolation:

### 1.2.2 R-100-1.2.3: General Requirements

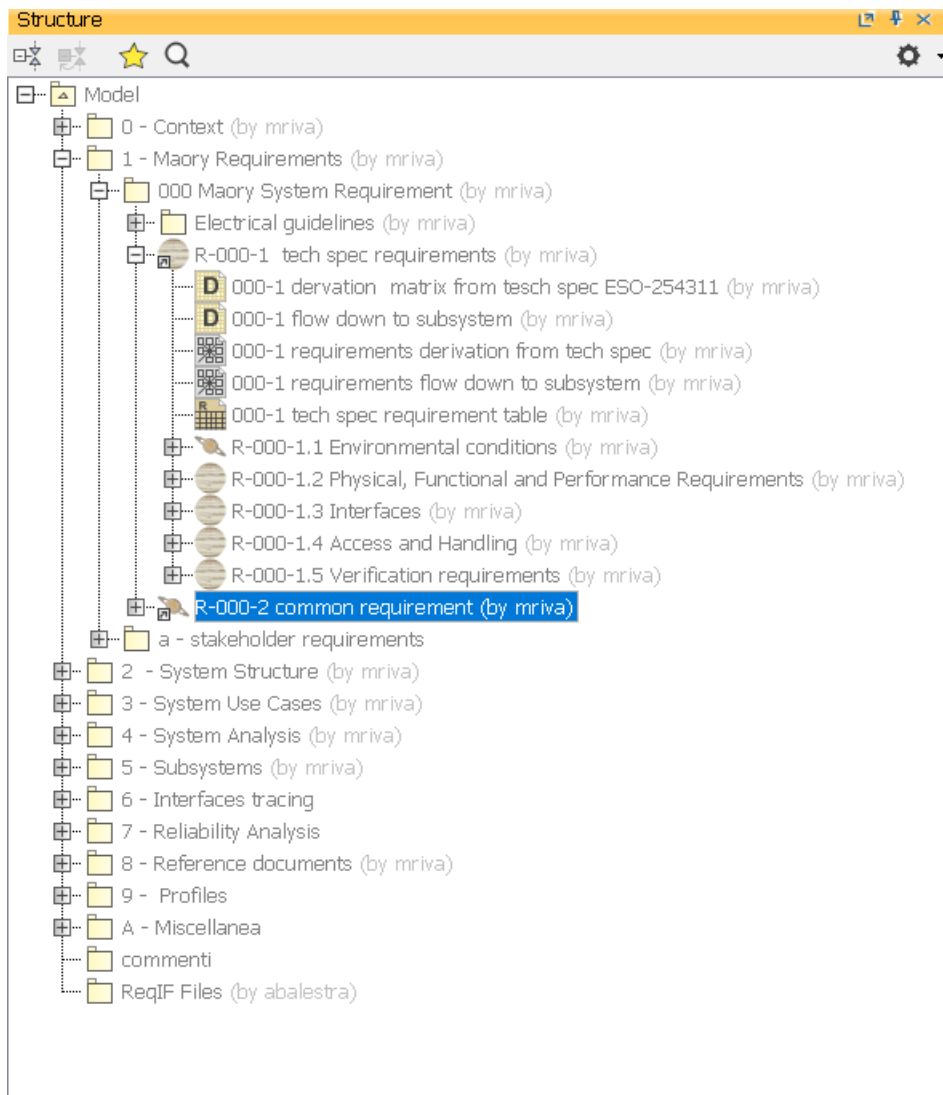
Root for all 100-1 Environmental General requirements

<b>R-100-1.2.3.1 : MCAO mode1</b>	
<b>Responsible</b>	LBU
<b>Requiemment Type</b>	Requiemment
<b>Description</b>	MAORYAO shall offer a multi-conjugate adaptive optics (MCAO) mode
<b>Rationale</b>	Direct flow down
<b>Parent Analysis</b>	<b>M-ANA-1.3</b> - MCAO main requirement
<b>Parent Requirement</b>	<b>R-MAO-57</b> - NA
<b>Refining analysis</b>	<b>M-ANA-1.4</b> - MCAO mode1
<b>Children requirements</b>	<b>MAO-ISO-1.2.2.1</b> - MCAO mode implementation <b>MAO-IUO-1.2.4.1</b> - MCAO mode calibration
<b>Verification analysis</b>	
<b>Notes</b>	[]

**R-MAO-57 Tracing Diagram**



The last step is the creation of a table of contents, by setting properly the various headings. The required structure contains all the information needed to write the script: specifically, the first thing that can be noted is the correspondence between the required table of contents and the roots of the requirements.



MAORY structure in Cameo Systems Modeler.

## •Contents¶

1 → R-100-1: tech spec requirements .....	4¶
1.1 → R-100-1.1: Environmental conditions .....	4¶
1.2 → R-100-1.2: Physical, Functional and Performance Requirements .....	6¶
1.2.1 → R-100-1.2.2: Physical Characteristics .....	8¶
1.2.2 → R-100-1.2.3: General Requirements .....	10¶
1.2.3 → R-100-1.2.4: Performance Requirements .....	30¶
1.3 → R-100-1.3: Interfaces .....	91¶
1.3.1 → R-100-1.3.3: Mechanical Interfaces .....	93¶
1.3.2 → R-100-1.3.4: Services Interfaces .....	100¶
1.4 → R-100-1.4: Access and Handling .....	108¶
1.5 → R-100-1.5: Verification requirements .....	114¶
2 → R-100-2: common requirement .....	116¶
-----Page Break-----¶	

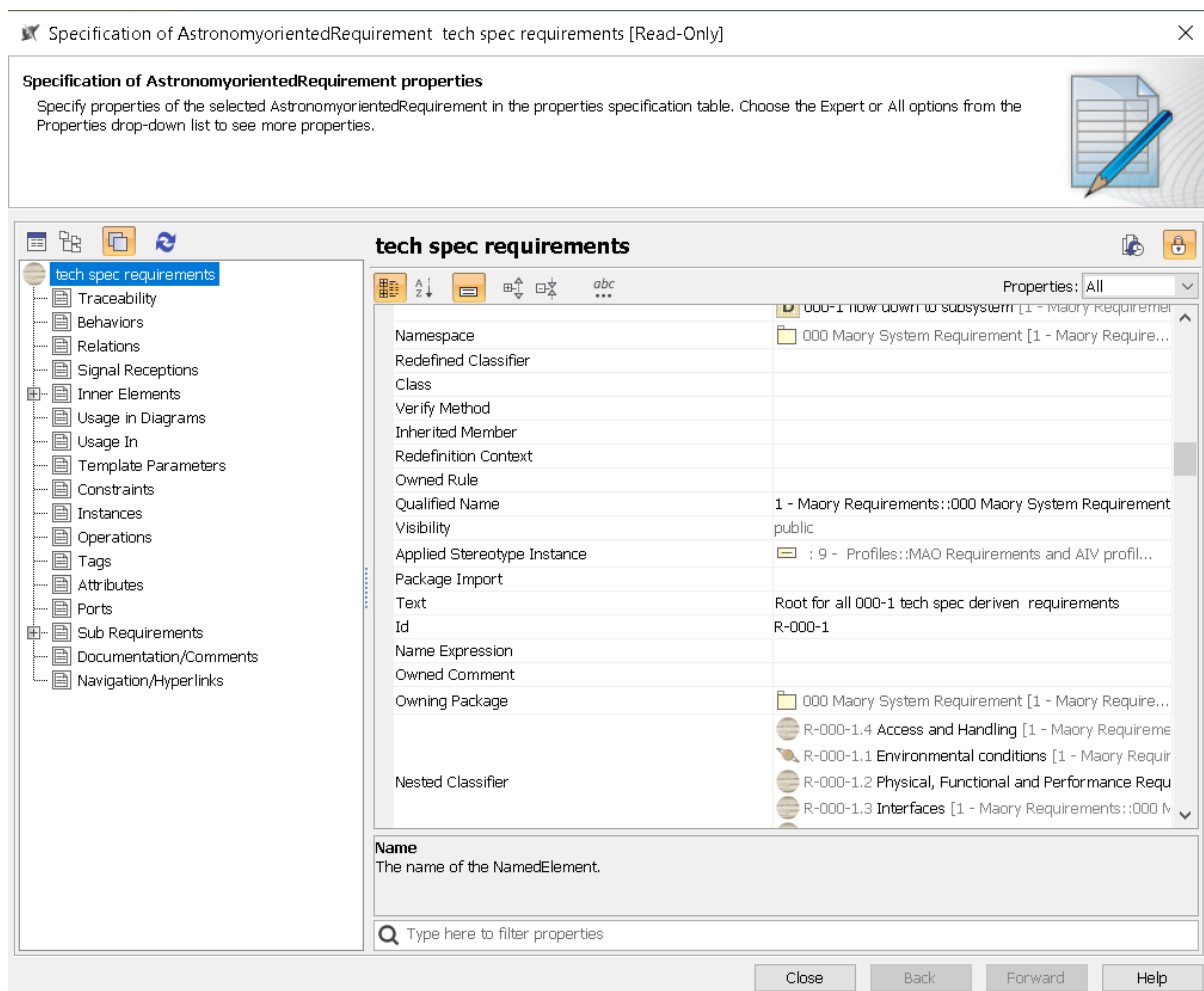
Required table of contents.

All string elements present in the table of contents are consequently headings, and they are all “Roots” for specific requirements.

For example, taking into consideration

*R-100-1 tech spec requirements*

and opening the Specification panel, it can be noted that indeed it is a “Root” for various requirements:



The property “Text” is identical for all roots in the project.

Starting from this fact, the VTL code that will format the heading is the following:

```

# if ($h.text.toString().contains("Root"))
# if (!$h.owner.text.toString().contains("Root"))

1 $id: $name
$description

# else

# set($par = $h.owner)
# if (!$par.owner.text.toString().contains("Root"))

1.1 $id: $name
$description

# else

1.1.1 $id: $name
$description

# end

# end

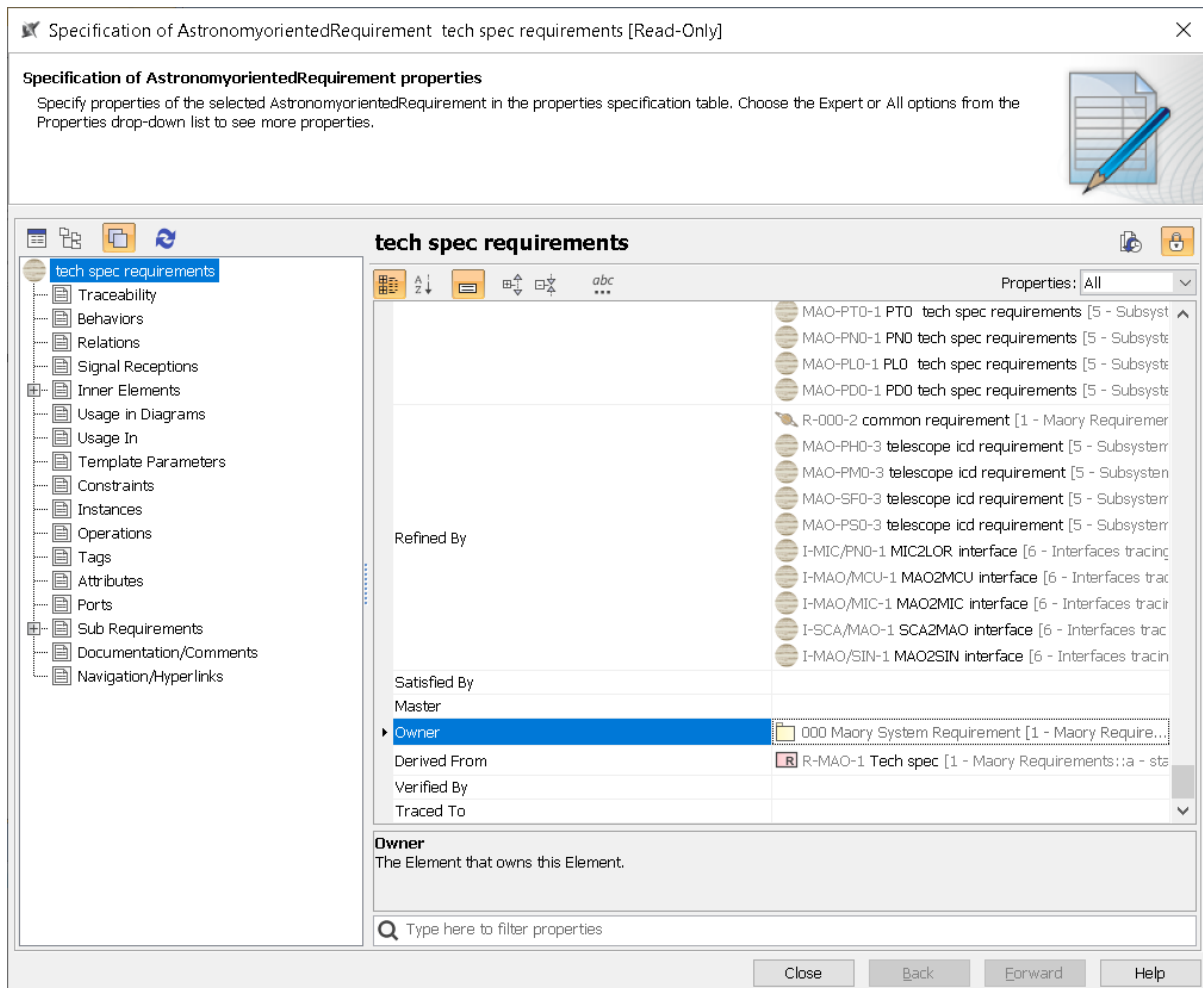
# else

```

The explanation of the script is the following: an element is checked in order to verify if it is a "Root". If it has no upper Root, it is the primary Root and it has to be formatted as Heading 1. If it has an upper Root, element

*Owner*

of the current element is checked; the Owner element contains information about other elements of which the current element belongs.



Now the Owner element can be checked, and if it contains an element which is Root, the current element will have Heading 2, otherwise Header 1.

In the script, Heading 1 starts with “1”, Heading 2 starts with “1.1” and Heading 3 starts with “1.1.1”: the Table of contents that will be generated after the report generation will properly number the various headers.

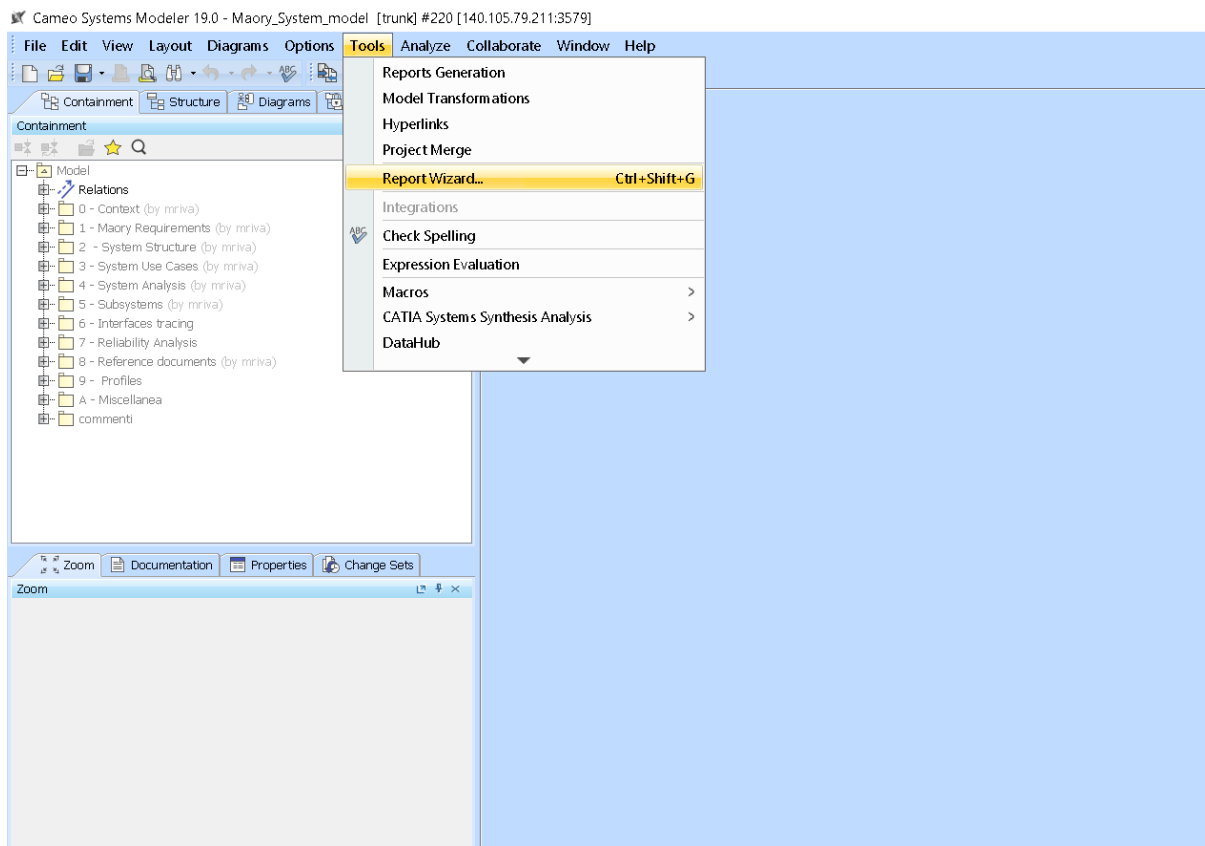
TABLE OF CONTENTS
Microsoft Word users please click here and press F9 to create Table of Contents.
OpenOffice.org users please remove this text and select Insert Table of Content from menu.

Table of contents can be generated after the creation of the report: once the report has been opened, by clicking F9 a Word process will create the table by checking the headers in the document.

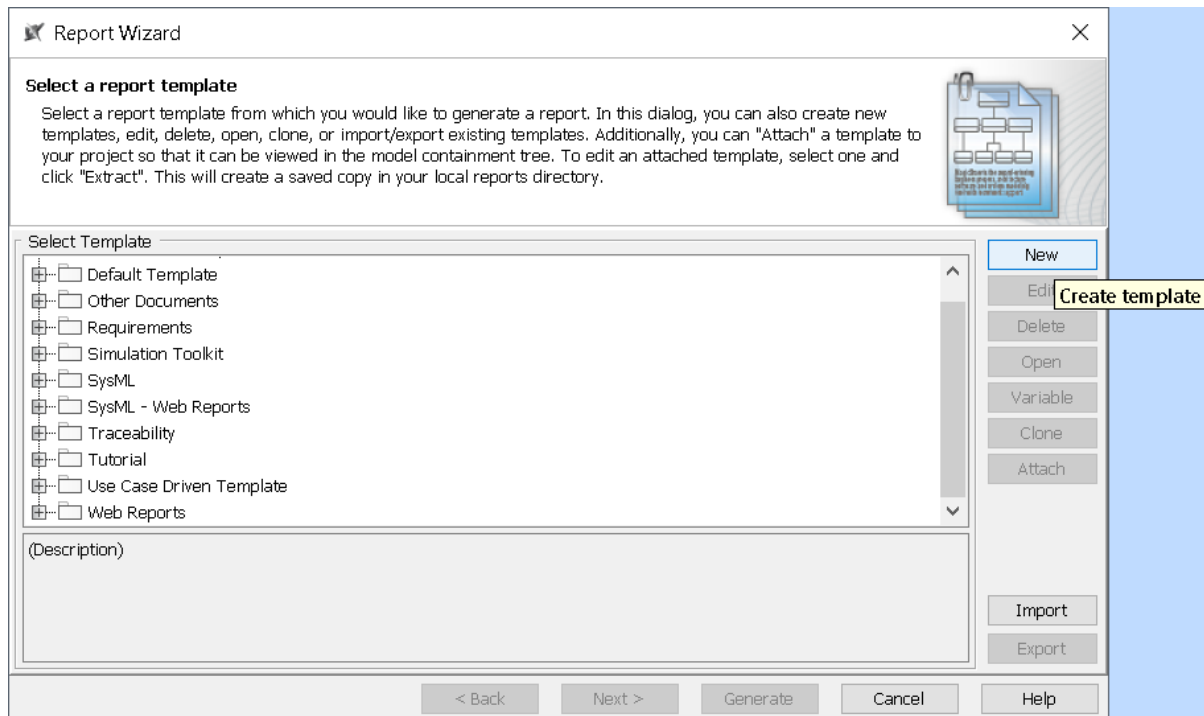
## 3.2 Report Wizard initial configuration



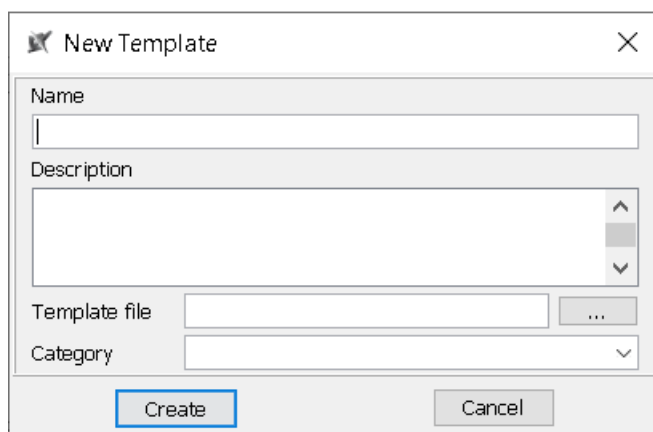
Initial step concerns the creation of a custom report template through Report wizard. User can open the engine by Tools menu.



The initial window lets the user select an already existent template or create a new one. By clicking on the “New” button, user can create a custom template.



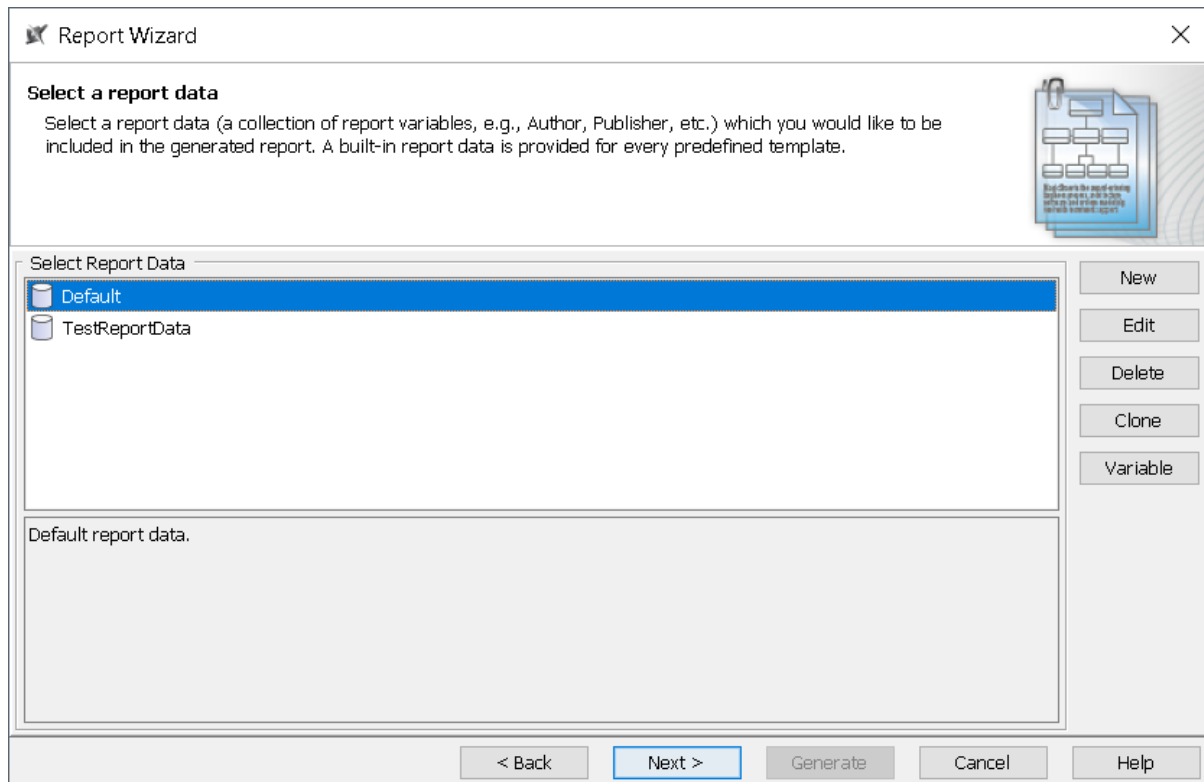
User has to choose a “Name” and a “Description”; after that, it is required to select the proper template file, which is a velocity script, and the category: in this scenario, “Category” is “Requirements” because the purpose of the project is to generate documents concerning requirements of MAORY System Model. Template file must be available, and user has to select it by indicating its location on the hard drive.



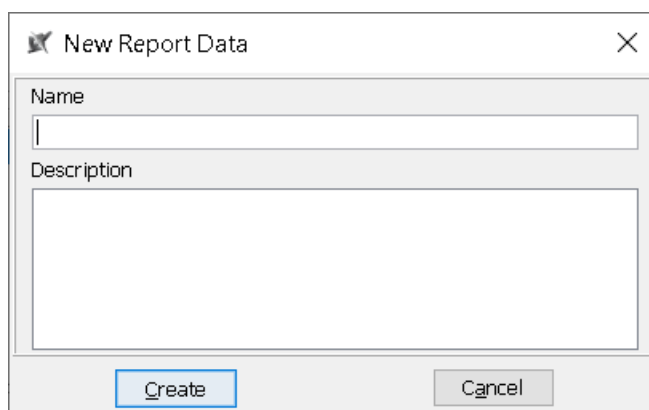
Once the template has been created, the next and important step is to create a “Report Data”. This is a collection of report variables, for example “Document classification”, “Authors”, etc. These variables will be included in the generated report, in fact they can be added in the Velocity script, in this way:

\$var

During report generation, \$var will assume its current value, that is the value that has been set in the “Report Data”.



By creating a new report data, user can create all variables that he/she will need in its report.




Once created, by clicking “Variable” button, a sub-menu will be showed, and user will be able to create variables by clicking “New” button.

Report Variable

Manage report variables

Manage predefined variables of the Report Data by creating, deleting, or defining their values. All variables and their values will be added in appropriate areas of the generated report.



Variable

DocumentNumber	1111test
DocumentVersion	1.0
DocumentType	Test document
Owner	Alessandro Giovannacci
ApprovedByPI	

(Description)

New

Delete

OK

Cancel

Next passage is to select the element scope. This scope will be the root from which the report generation will start. In the follow example, by selecting “Electrical guidelines” package, and by clicking on the “+” button, user will be able to generate all elements belonging to this package and sub-packages (in this case, that is MAORY, elements is equivalent to requirements).



Once all options have been set, user can close Report Wizard and save changes; in fact, there is no need to click “Generate” button in order to complete the process, as this part will be carried out by the plugin.

## 3.3 Reports Generation plugin

### 3.3.1 First approach

The key idea is to extend the Cameo Systems Modeler software, including a function that, when invoked, creates a specific number of files, that is reports, based on the selected packages the user wants to generate documentation for. In fact, Report Wizard does not permit to generate multiple files based on multiple packages but generates only one file if recalled by the graphic user interface. A way to solve the problem could be generating reports from the command line, using the “generate” command.

The Generate command creates a report document with a received set of information from arguments and parameters. The information will then be generated as a report document to the specified output file. By default, an argument is the specified data of the invoked parameters. If the -properties option is specified, the argument is the name of a properties file. A properties file contains other parameters, along with the specified data of each parameter.

A possible solution could be the following. The script has been created as a Windows batch file.

```
@echo off
```

```
setlocal EnableDelayedExpansion
```

```
::*****Set reportwizard folder variable*****
```

```
set /p programFolder= "Enter CSM ReportWizard path:"
```

```
cd %programFolder%
```

```
::*****Set project variable*****
```

```
set /p project= "Enter project path:"
```

```
::*****Set destination folder*****
```

```

set /p destination= "Enter destination folder:"

::*****Set packages names*****

set i=0

:new

set /a i=%i%+1

set /p packageName[%i%]= "Enter package name:"

set /p continue="Do you want to enter another package name?(y/n)"

set continue=%continue:="%

if %continue%==y (call goto :new)

::*****Generate reports*****

:loop

call generate -project "%project%" -output "%destination%\!packageName[%i%]!.docx" -
template "TestTemplate" -package "!packageName[%i%]!"

set /a i=%i%-1

if %i% NEQ 0 (goto :loop)

echo Files generation successfully completed

pause

```

The script sets Cameo Systems Modeler Report Wizard path (that is local path of the installation of the software), then asks the user to insert project path and packages' names, and destination path where files will be saved.

There are several problems with this script that makes it not suitable for the purpose. First problem is setting the local variable "project", which will contain the path of the project. MAORY has been created as a Teamwork Cloud Server project, and this means it could be difficult for a user to set the correct path. Second problem is portability, as this file has been written for Windows, but users work with different operating systems, so there could be problems in the distribution and the use of the file. Another problem concerns package names, because users should be careful typing them, as these must be precisely the same names of the project packages in Cameo Systems Modeler, e.g. case sensitive. A single error can not permit to generate the report. Furthermore, a marginal problem of this file could be its syntax, which is not particularly simple and intuitive, and this could lead to maintenance problems.

### 3.3.2 A more efficient and scalable solution

Cameo Systems Modeler, as other products developed by No Magic, is written in Java. It is based on the MagicDraw modeling platform, and developers provided the Open Java Application Programming Interface (API). This means custom plugins can be implemented, as well as adding actions to the menus or toolbars, changing UML model elements and creating new patterns.

The program core code and used libraries are packaged in jars files that are located in `CameoSystemsModeler\lib`, and its subdirectories.

Plugin libraries can be located in `CameoSystemsModeler \plugins`.

The code can be divided into three scope groups:

- OpenAPI;
- InternalAPI;
- NotAPI.

OpenAPI contains code for public usage, and typically plugins are written using this API, as it is stable through builds and versions.

InternalAPI contains code for internal NoMagic usage only and may change through builds and versions without any restrictions.

NotAPI contains code that may change in each build. This API is not meant to be used when developing a custom plugin.

API is well documented, allowing users to understand details of the classes, their attributes and methods.

The screenshot displays the No Magic API documentation interface. On the left, a sidebar lists 'All Classes' and 'Packages'. The 'Packages' section is expanded, showing a list of packages including `com.nomagic.actions`, `com.nomagic.annotation`, `com.nomagic.awt`, `com.nomagic.axis`, `com.nomagic.cameo.resources`, `com.nomagic.ci.persistence`, `com.nomagic.ci.persistence.decomposition`, `com.nomagic.ci.persistence.services`, `com.nomagic.ci.persistence.spi.decomposition`, `com.nomagic.ci.persistence.versioning`, `com.nomagic.diagramtable`, and `com.nomagic.generictable`. The main content area shows the 'Overview' tab selected, displaying a table of packages with their descriptions.

Package	Description
<code>com.nomagic.actions</code>	This package contains extensions to swing actions.
<code>com.nomagic.annotation</code>	
<code>com.nomagic.awt</code>	This package contains extensions to awt and swing components.
<code>com.nomagic.axis</code>	
<code>com.nomagic.cameo.resources</code>	
<code>com.nomagic.ci.persistence</code>	
<code>com.nomagic.ci.persistence.decomposition</code>	
<code>com.nomagic.ci.persistence.services</code>	
<code>com.nomagic.ci.persistence.spi.decomposition</code>	
<code>com.nomagic.ci.persistence.versioning</code>	
<code>com.nomagic.diagramtable</code>	
<code>com.nomagic.generictable</code>	

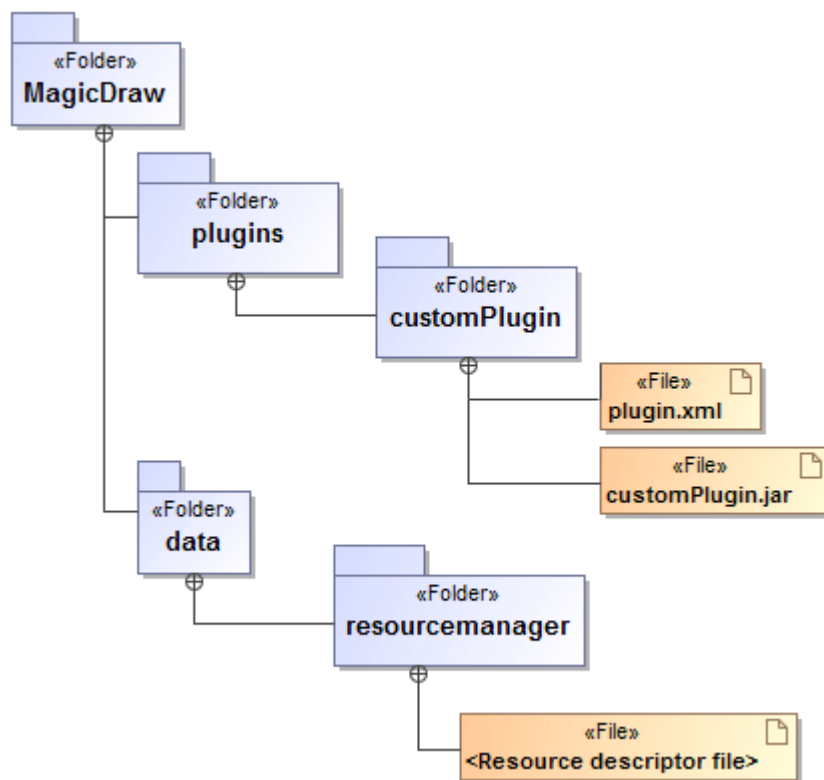


Plugins are the only one way to change the functionality of a modeling tool, in fact the main purpose of developing a plugin is adding new functionalities or extending already present functionalities.

### 3.3.3 Plugin implementation

Plugin must have a precise structure in the NoMagic's products context.

A plugin, that is a functionality extender, must contain a directory, compiled java files packaged into a jar file, and a plugin descriptor file, which is an xml file, plus optional files plugin could use or could generate during its execution. Plugin folder must be in the plugin directory of Cameo Systems Modeler.



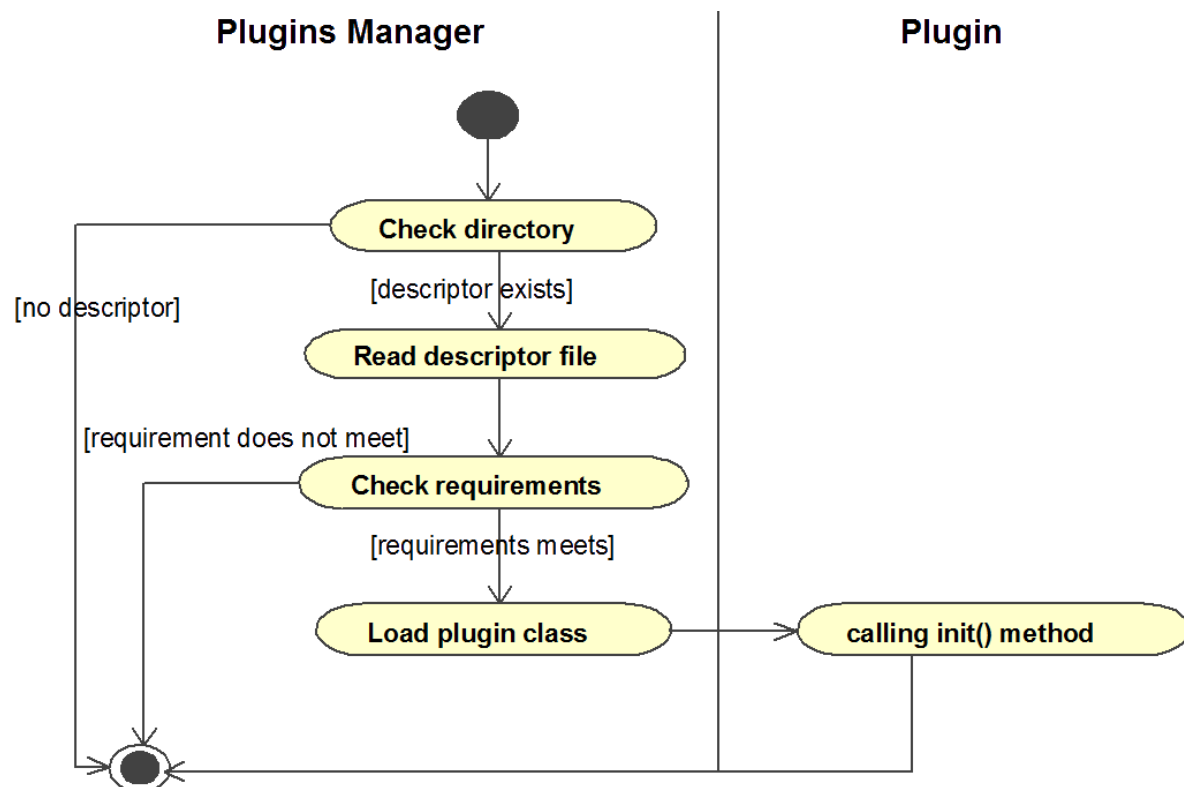
Cameo Systems Modeler, at every startup, scans the plugin directory, and searches for subdirectories: if a subdirectory contains the plugin descriptor file, the plugin's manager reads the descriptor file. If requirements specified in a descriptor file are fulfilled, the plugin's manager loads a specified class; the specified plugin class must be derived from the following class:

*com.nomagic.magicdraw.plugins.Plugin*

At this point, the method:

*init()*

of the loaded class is called. This method can perform various tasks, e.g. it can add GUI components using the actions architecture or do other activities and return from the method.



Plugin descriptor is a file written in XML which contains specific properties of the plugin; in fact, this file contains the definitions of elements of the plugin. Elements of this file are:

- plugin
- requires
- api
- required-plugin
- runtime
- library

Each element contains various attributes or nested elements, e.g. plugin element has eight properties:

- id

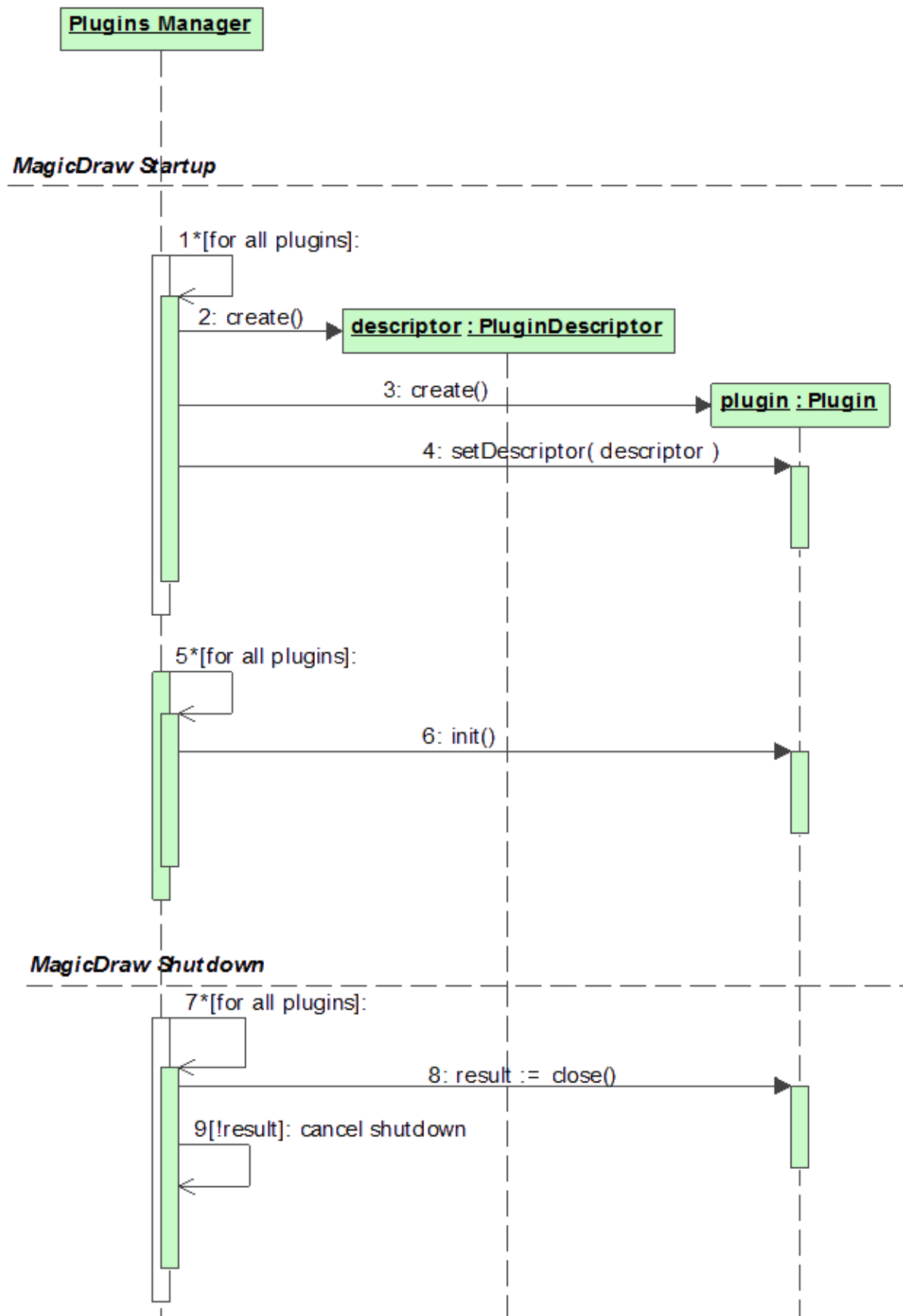
- name
- version
- internalVersion
- provider-name
- class
- ownClassLoader
- class-lookup

In order to write a plugin, class:

*com.nomagic.magicdraw.plugins.Plugin*

must be extended; it is the base abstract class for any modeling tool plugin. Every plugin has its own descriptor set by a plugin's manager. A plugin has three special methods:

- *init()* method is called on a program startup. The plugin must override this method and implement its own functionality there.
- *close()* method is called on a program exit. The plugin must override this method and return the value true if the plugin is ready to exit. In other case, it should return the value false. If the plugin returns false, the program exit is canceled.
- *isSupported()* method is called before the plugin initialization. If this method returns false, the plugin is not initialized. The method may be used to check if the plugin can be started, for example, on a specific operating system.



In order to get references to loaded plugins, class:

*com.nomagic.magicdraw.plugins.PluginUtils*


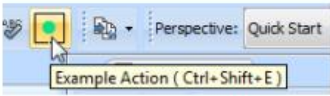
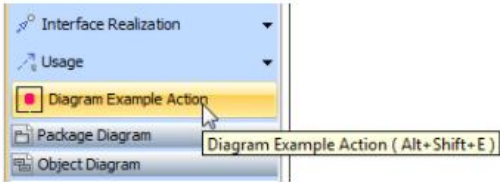
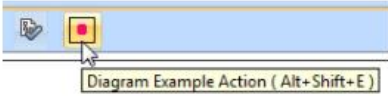
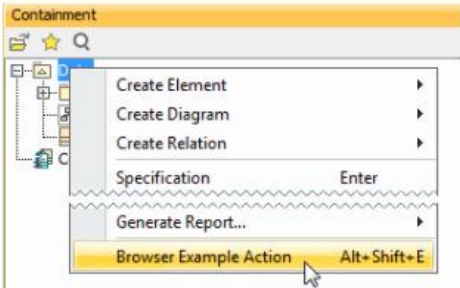
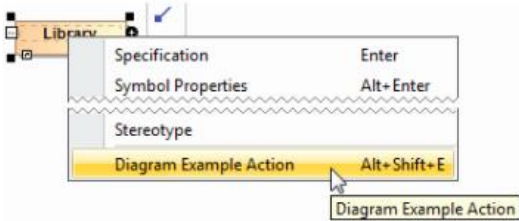
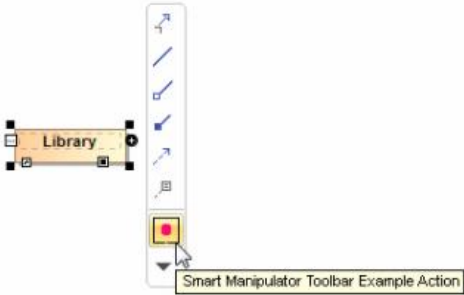
can be used, specifically if user wants to get a reference to other plugin objects.

Information loaded from the plugin XML file to the plugin can be retrieved by the class:

*com.nomagic.magicdraw.plugins.PluginDescriptor*

Cameo Systems Modeler is based on a program actions mechanism, which means that all menu commands and buttons are considered as actions.

The followings are all examples of actions in NoMagic's product:

Main menu	
Main toolbar	
Diagram pallet	
Diagram toolbar	
Browser shortcut menu	
Diagram shortcut menu	
Smart manipulator toolbar	

In this scenario, the way to add or extend a specific functionality is to create and add a new action.

### 3.3.4 Plugin structure

The main class of Reports Generation plugin is:

*ReportsGenerationPlugin*

which extends Plugin class of NoMagic.

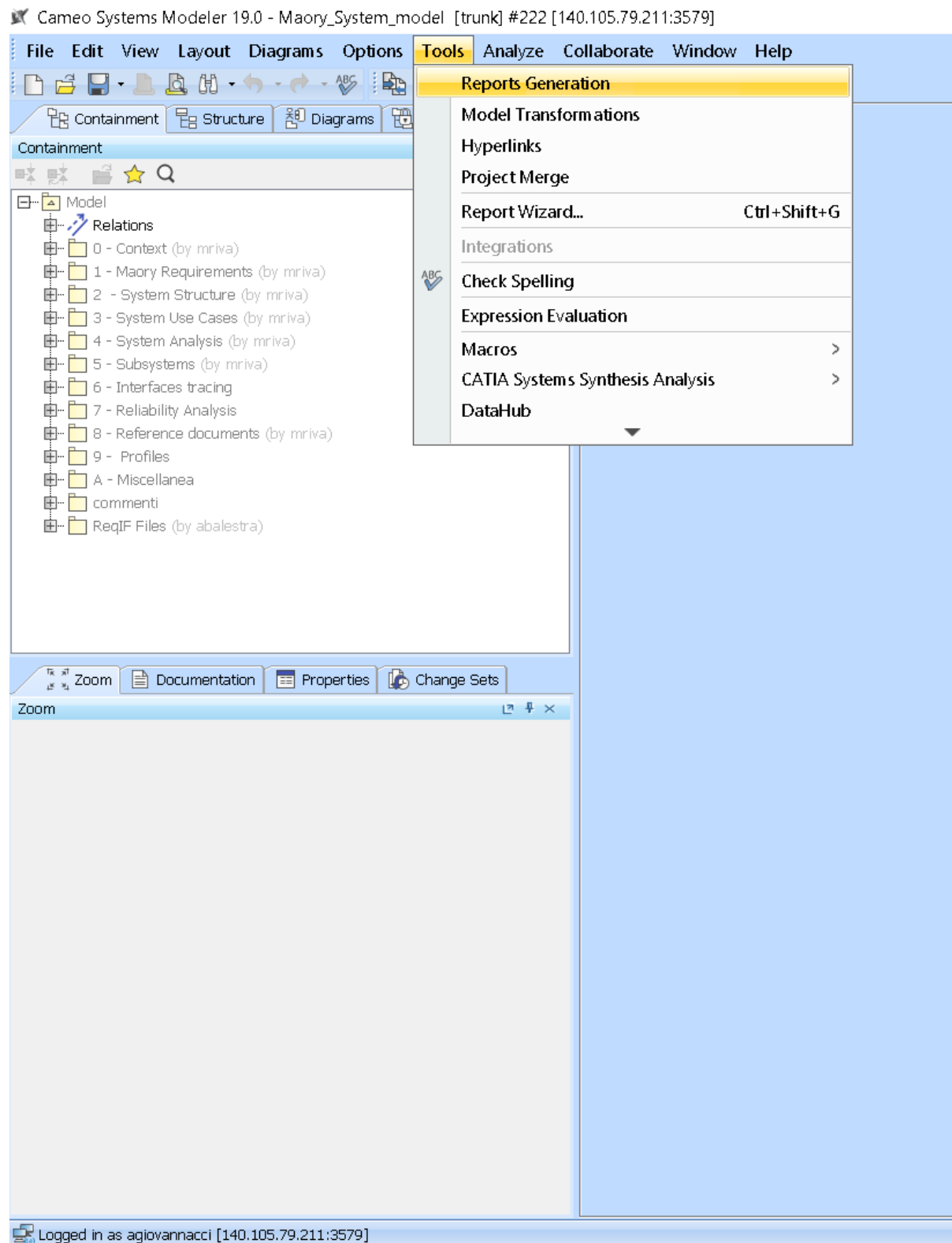
This class contains method:

*init()*

which sets a new action, specifically

*ReportsGenerationAction*

This will be the action through which Reports Generation plugin will be invoked, and this action will be set under the “Tools” menu of Cameo Systems Modeler.



All actions used in the modeling tool must be subclasses of:

*com.nomagic.magicdraw.actions.MDAAction*



Overriding the method:

```
actionPerformed(java.awt.event.ActionEvent)
```

in the custom action class will enable the action to perform its work.

```
public void actionPerformed(ActionEvent e){  
    //start generating report  
    BackgroundTaskRunner.runWithProgressStatus(new ReportsGenerator(), "My progress",  
false);  
}
```

In order to control the update of the action state, action can be added into a predefined actions group. All actions of the one group are disabled/enabled together. Conditions for groups enabling/disabling and status updating are predefined and cannot be changed.

Finally, every action must be added into a

```
com.nomagic.actions.ActionsCategory
```

class. *ActionsCategory* is a small group for actions. It can be represented as a separator or submenu (a nested category).

Categories are added into the

```
com.nomagic.actions.ActionsManager
```

class, which is an actions' container. One represents one GUI element, e.g. a menu bar, a shortcut menu, or a toolbar.

Actions in *ActionsManager* are configured by many configurators. A *Configurator* is responsible for adding or removing the action into a strictly defined place and position between other actions.

Private method implemented in class *ReportsGenerationPlugin*:

*setAMConfigurator(ReportsGenerationAction action)*

is responsible of adding the action into the “Tools” menu.

```
private void setAMConfigurator(ReportsGenerationAction action){  
    AMConfigurator conf = new AMConfigurator(){  
        public void configure(ActionsManager mngr){  
            // Searching for an action after which an insertion should be done.  
            NMAction found= mngr.getActionFor(ActionsID.TOOLS_PLUGINS);  
  
            // The action found, inserting  
            if(found != null){  
                ActionsCategory category = (ActionsCategory)mngr.getActionParent(found);  
                List actionsInCategory = category.getActions();  
  
                //Add the action  
                int indexOfFound = actionsInCategory.indexOf(found);  
                actionsInCategory.add(indexOfFound+1, action);  
  
                // Set all actions.  
                category.setActions(actionsInCategory);  
            }  
        }  
  
        public int getPriority(){  
            return AMConfigurator.MEDIUM_PRIORITY;  
        }  
    };
```

```
ActionsConfiguratorsManager.getInstance().addMainMenuConfigurator(conf);  
}
```

The last instruction of the method permits to register the configurator.

All configurators are registered in the

*com.nomagic.magicdraw.actions.ActionsConfiguratorsManager*

class. *ActionsConfiguratorsManager* enables to add or remove many configurators to every configuration predefined by a modeling tool.

The task of generating reports is performed by class

*ReportsGenerator*

which implements a specific interface:

*RunnableWithProgress*

The task of generating multiple reports in the MAORY context is computationally onerous. A single report generation of a package can take several minutes, in function of the number of requirements and diagrams involved in the exportation.

The choice of implementing the task as a background task is optimal, because it permits users to perform other tasks in the meantime.

The class that performs the generation task must override method

*run(ProgressStatus progressStatus)*

This is the structure of the class:

```
public class ReportsGenerator implements RunnableWithProgress{
```

```
    public void run(ProgressStatus progressStatus){  
        //...  
    }  
}
```

When user clicks on the Reports Generation action, method

*actionPerformed*

is called; it contains the following instruction:

```
BackgroundTaskRunner.runWithProgressStatus(new ReportsGenerator(), "My progress", false)
```

The first parameter of the method is the class that implements interface

*RunnableWithProgress*

while the other two parameters are secondary.

At this point, the report generation can start.

Report Wizard API provides various classes to help customizing the process. These are main packages:

# Report Wizard API Documentation

## Packages

Package	Description
<code>com.nomagic.magicdraw.magicreport</code>	
<code>com.nomagic.magicdraw.magicreport.helper</code>	
<code>com.nomagic.magicdraw.magicreport.tools</code>	
<code>com.nomagic.magicdraw.magicreport.ui.bean</code>	
<code>com.nomagic.reportwizard.tools</code>	
<code>com.nomagic.reportwizard.tools.dependencymatrix</code>	
<code>com.nomagic.reportwizard.tools.dialog</code>	
<code>com.nomagic.reportwizard.tools.doc</code>	
<code>com.nomagic.reportwizard.tools.doc.javadoc</code>	
<code>com.nomagic.reportwizard.tools.docbook</code>	
<code>com.nomagic.reportwizard.tools.generictable</code>	
<code>com.nomagic.reportwizard.tools.importer</code>	
<code>com.nomagic.reportwizard.tools.metrics</code>	
<code>com.nomagic.reportwizard.tools.query</code>	
<code>com.nomagic.reportwizard.tools.script</code>	
<code>com.nomagic.reportwizard.tools.template</code>	
<code>com.nomagic.reportwizard.tools.validation</code>	

## OVERVIEW

## PACKAGE

## CLASS

## USE

## TREE

## DEPRECATED

## INDEX

## HELP

Reports generation task involves various classes, more specifically:

- *TemplateBean*

com.nomagic.magicdraw.magicreport.ui.bean

## Class TemplateBean

java.lang.Object

com.nomagic.magicdraw.magicreport.ui.bean.TemplateBean

All Implemented Interfaces:

java.lang.Cloneable, java.lang.Comparable<TemplateBean>

---

```
public class TemplateBean
```

```
extends java.lang.Object
```

```
implements java.lang.Comparable<TemplateBean>, java.lang.Cloneable
```

Contains template information.

- *ReportBean*

com.nomagic.magicdraw.magicreport.ui.bean

## Class ReportBean

java.lang.Object

com.nomagic.magicdraw.magicreport.ui.bean.ReportBean

All Implemented Interfaces:

java.io.Serializable, java.lang.Cloneable, java.lang.Comparable<ReportBean>

---

```
public class ReportBean
```

```
extends java.lang.Object
```

```
implements java.io.Serializable, java.lang.Comparable<ReportBean>, java.lang.Cloneable
```

Contains report information. A report JavaBean.

- *GenerateTask*

com.nomagic.magicdraw.magicreport

## Class GenerateTask

```
java.lang.Object
  com.nomagic.task.SwingWorker<java.lang.Object>
    com.nomagic.task.Task
      com.nomagic.magicdraw.magicreport.ObserverTask
        com.nomagic.magicdraw.magicreport.GenerateTask
```

All Implemented Interfaces:

```
java.util.Observer
```

---

```
public class GenerateTask
  extends com.nomagic.magicdraw.magicreport.ObserverTask
  implements java.util.Observer
```

Generating task for Magic Report.

These three classes are mandatory and must be used in order to be able to start the generation process.

*TemplateBean* class, that is the object of the class, holds all the information of a specific template. In order to instantiate the object, a template must be selected:

*TemplateHelper*

class is responsible for listing all existing templates.

com.nomagic.magicdraw.magicreport.helper

## Class TemplateHelper

```
java.lang.Object
  com.nomagic.magicdraw.magicreport.helper.TemplateHelper
```

---

```
@OpenApiAll
public class TemplateHelper
  extends java.lang.Object
```

Provide helper methods for template such as create and save.

The following instruction permits to obtain a list of all templates:

```
List templateList = TemplateHelper.listTemplates();
```

At this point, the template created ad hoc for MAORY requirements can be selected, and all its information can be retrieved.

All template information is taken from a specific XML file, `template.xml`, which is stored in Cameo Systems Modeler folder.

From *templateBean* object, the default report can be retrieved:

```
reportBean = (ReportBean) templateBean.getDefaultReport().clone();
```

Object *reportBean* contains all predefined information set during the report wizard initial configuration.

All information can be retrieved by the getter methods of the class.

At this point, report can be set:

```
templateBean.setSelectedReport(reportBean);
```

There is a specific class, which is

*PackageSelectionBean*



com.nomagic.magicdraw.magicreport.ui.bean

## Class PackageSelectionBean

java.lang.Object

com.nomagic.magicdraw.magicreport.ui.bean.PackageSelectionBean

All Implemented Interfaces:

java.io.Serializable, java.lang.Cloneable

---

```
public class PackageSelectionBean
```

```
extends java.lang.Object
```

```
implements java.io.Serializable, java.lang.Cloneable
```

Contains selected packages information.

which can be used to get the selected packages set in the report initial configuration.

If the user wants to change the selection or the scope, e.g. including or excluding a specific package, he/she can open Report Wizard and apply changes.

Changes made in the Report Wizard are automatically saved by an internal daemon process of Cameo Systems Modeler, so at runtime the object *reportBean* contains the correct information.

### 3.3.5 Compilation of Java files

The compilation of the source code requires particular attention: in fact, several classes used in the implementation are part of the NoMagic's "InternalAPI"; this code is designed for internal NoMagic usage only, and it may change through builds and versions without any restrictions.

Despite not being supposed to be used by external developers, packages and classes of InternalAPI can still be used. In a modern Integrated development environment, deprecated signs appear when using this code, and normal compilation instructions are not enough in order to compile.

Several classes used are part of InternalAPI, such as:

- *ReportBean*
- *PackageSelectionBean*
- *TemplateBean*

Compilation can be done with the use of the non-standard option:

*Xlint*

followed by a name, where name is a warning name:

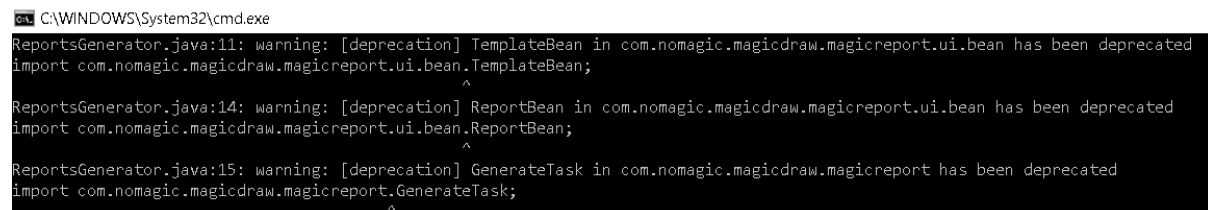
- *deprecation*
- *unchecked*

The following script is part of a batch file made for compilation task:

```
@echo off
cd C:\Program Files\Cameo Systems Modeler\plugins\reportsgeneration
javac -Xlint:deprecation ReportsGenerator.java
pause
cd..
javac -Xlint:deprecation reportsgeneration\ReportsGenerationAction.java
pause
javac -Xlint:unchecked reportsgeneration\ReportsGenerationPlugin.java
pause
jar -cf reportsgeneration.jar reportsgeneration/*.class
```

The last instruction calls the jar program which creates a jar file, and this jar is the plugin that will be loaded in Cameo Systems Modeler.

Compilation output still has warnings:



```
C:\WINDOWS\System32\cmd.exe
ReportsGenerator.java:11: warning: [deprecation] TemplateBean in com.nomagic.magicdraw.magicreport.ui.bean has been deprecated
import com.nomagic.magicdraw.magicreport.ui.bean.TemplateBean;
^
ReportsGenerator.java:14: warning: [deprecation] ReportBean in com.nomagic.magicdraw.magicreport.ui.bean has been deprecated
import com.nomagic.magicdraw.magicreport.ui.bean.ReportBean;
^
ReportsGenerator.java:15: warning: [deprecation] GenerateTask in com.nomagic.magicdraw.magicreport has been deprecated
import com.nomagic.magicdraw.magicreport.GenerateTask;
^
```

but the compilation process has been completed without errors, and class files have been generated.

## 4 Bibliography

- INAF: <https://en.wikipedia.org/wiki/INAF>  
<http://www.inaf.it/en/>
- No Magic, Cameo Systems Modeler and related images: <https://docs.nomagic.com>  
<https://www.nomagic.com/products/cameo-systems-modeler>
- Velocity Template Language: [https://en.wikipedia.org/wiki/Apache\\_Velocity](https://en.wikipedia.org/wiki/Apache_Velocity)  
<http://velocity.apache.org/>
- SysML: [https://en.wikipedia.org/wiki/Systems\\_Modeling\\_Language/](https://en.wikipedia.org/wiki/Systems_Modeling_Language/)  
<https://sysml.org/>
- MAORY: <http://www.maory.oabo.inaf.it/>
- Adaptive optics: [https://en.wikipedia.org/wiki/Adaptive\\_optics](https://en.wikipedia.org/wiki/Adaptive_optics)
- ESO: [https://en.wikipedia.org/wiki/European\\_Southern\\_Observatory](https://en.wikipedia.org/wiki/European_Southern_Observatory)  
<https://www.eso.org>