RobotML Documentation

Release 2.0alpha

RobotML development team

April 21, 2014

This page introducess the *RobotML* platform (see section below) as well as the the way to build the documentation you are reading (see *Documentation How-to*). There is, following, a section dedicated to indices and tables (see *Indices and tables*) and finally a list of websites linked to the platform (see *Related websites*).

RobotML is a language associated to a platform linked to PAPYRUS tool and build upon ECLIPSE framework. The goal of this language is to ease development of robotics application and to facilitate exchanges between roboticians and end-users.

RobotML documentation is decomposed in many parts. A presentation of its rationale is done in its introduction, then, instructions are provided in order to install the platform followed by some hints of how to use it through some examples. The following sections are providing in-depth information on the underlying language (the so-called meta-model), the different choices done in order to present the user modelling capabilities and finally the different generators add-ons with example of use. The last section is dedicated to the web *portal* and its interactions with the *RobotML* platform.

The last bit of important information is that each time it is necessary, the documentation will separate the *user* point of view from the *developper* one trying to limit the amount of knowledge to what is strictly deemed necessary.

CHAPTER

ONE

INTRODUCTION TO ROBOTML

RobotMl documentation covers many things. In order to understand the underlying logic it is necessary to understand what is *RobotML*, what are its different parts and what part the documentation is describing.

RobotML is composed of three main parts and some associated tools. These three parts are:

> 1. A modelling platform based upon *Papyrus* that allows to represent Robotics systems and associated environments 2. Generators based upon *Acceleo* and allowing to deploy the models composed thanks to the above parts towards a list of robotic middle-ware (namely as well as two environment simulator (namely *MORSE* and *Cycab-Tk*)

1.1 The rationale of RobotML

The goal of the RobotML development is to ease interactions between the different actors of the community. This goal is based upon the idea promoted in the following schematic.

Figure 1.1: *PROTEUS Rationale*

The sections that follow are detailing this schematic. The *Life Cycle* section details how the interactions are embodied in real life and the *RobotML* explains how the different developed tools and parts instantiate this life cycle.

1.1.1 Life cycle

Roles

As presented in picture *PROTEUS Rationale* there are several roles to be considered:

1. *Provider*: Here it means someone that is able to upload something on the *portal* either using the different tools developed in the *PROTEUS* project or directly knowing the protocols to be used. These actions ask for an authentication that will allow to limit access to the different parts of the portal.
2. *User*: Here it means someone that is able to download something from the portal in the same conditions as the *provider* with as much authentication knowing that some parts will be relayed to private website askinng perhaps for other authentication steps

Objects on the *portal*

*provider*s create and upload many different elements that can be downloaded by many *user*s. Elements that could be exchanged are numerous:

1. *Model*: It is an abstract view of a system as implemented using the *RobotML* representation
2. *Algorithm*: an Algorithm *provider* can provide to the community either a description of its solution or a software module implementing this description. In the scope of the *portal* this module will have to respect properties.
3. *Problem*: in the scope of the portal a *problem* is not only some documents lightly defining problems' condition. It means to provide specific data allowing to understand in detail what it is and to assess possible answers against it.
4. *Robot*: This portal is open to the community and thus also to Robots providers that will be able to promote their products on the portal. In the context of the life cycle it means also that
5. Data: It can be videos, photos, whatever that is of interest to robotic, e.g. elements to test video processing functions or EM images, etc.

Life Cycle

Presentation

Rationale, Roles and objects to be manipulated are not sufficient to implement a life cycle. In fact, the *portal* philosophy is explicitly build around the following life cycle:

1. In order to begin such a life cycle, it is necessary first to ask questions and in this scope it means for *Problem* to provide *problem*s
2. Then there is a need to provide answers, it means for *Solutions* to provide *Algorithms, including architectures* able to answer the question asked by the above *problem*.
3. Last but not least, there should be away for those asking questions to assess for themselves the so-called answers.

These steps are described in the following figure

As can be seen *Scenario* and *Problems* are in the hands mostly of the end-users. An end-user is often a company not knowing how to solve a question but knowing what this question is. It is in our context up to roboticians to provide answers to these questions providing *Solutions* and going to the extent of creating reusable *modules* for the community to consider at large.

Portal and life cycle

As can be seen in the following picture, the *RobotML portal* implements as it is this life cycle taking into account management of web account (authoring, reserved space on the portal, ...).

RobotML platform and life cycle

As can be seen on the following picture a specific customisation of the platform help system has been created that implements the life cycle. The user will be able to follow it and she / he will be guided through the different steps implementing each of the steps of the life cycle.

1.1.2 Supporting Life cycle through associated tools

Presentation

As it is presented in the previous page there is a need to implement the life cycle in the actual existing tools provided to the different types of users. The following picture details them.

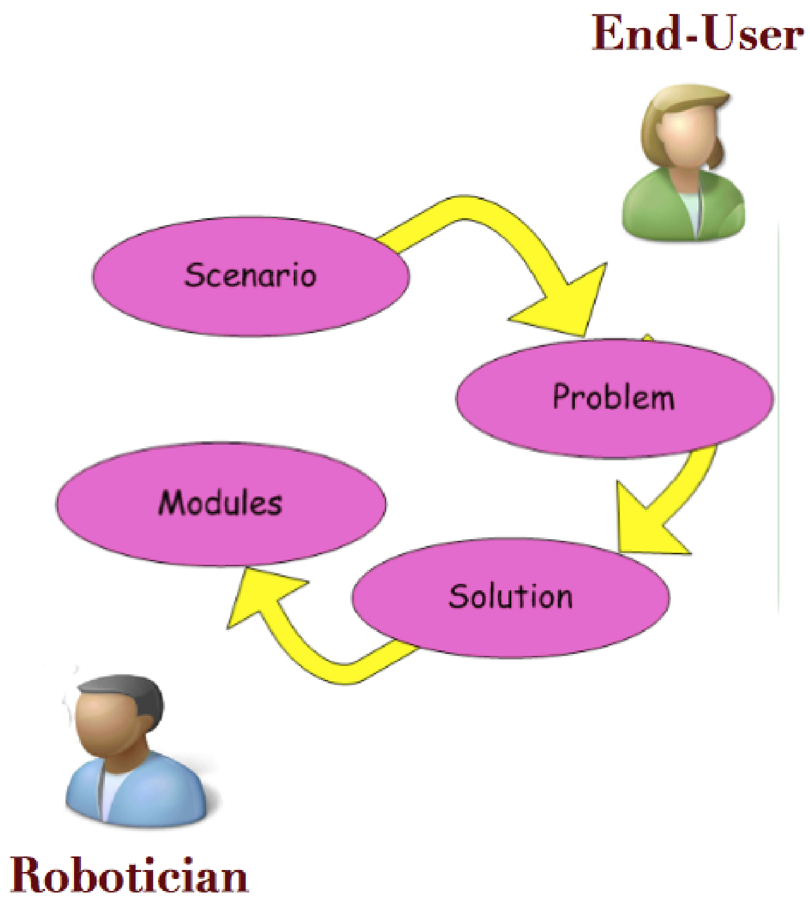
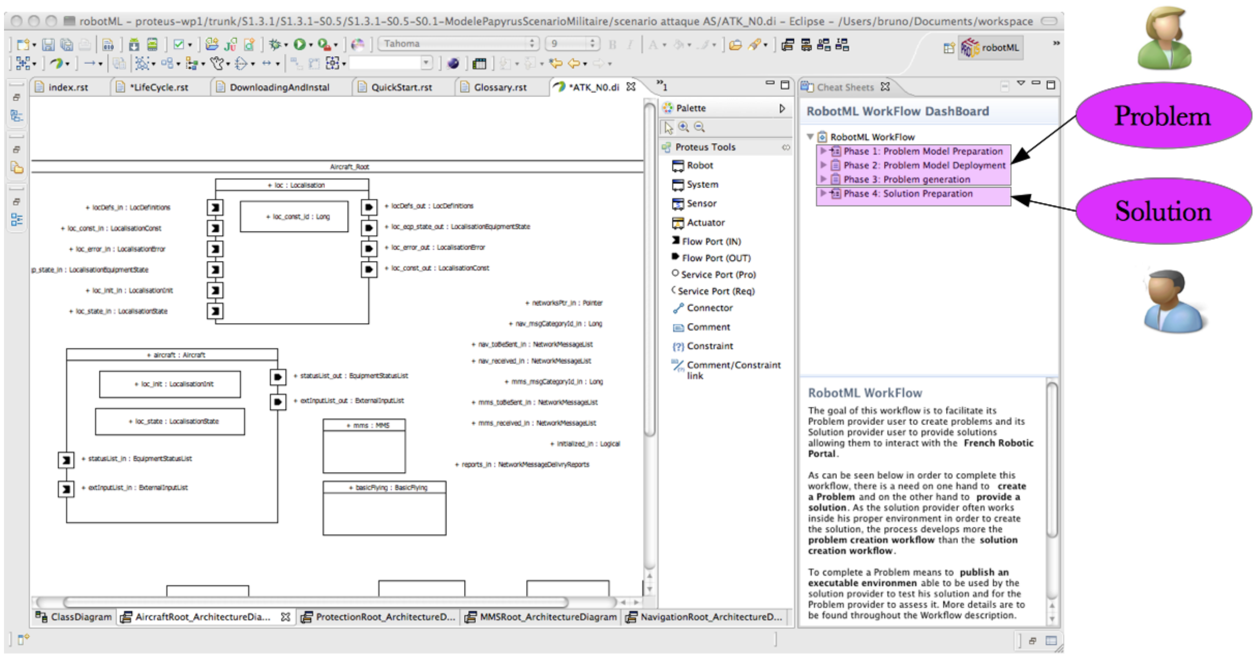


Figure 1.2: *PROTEUS Workflow*


robotML – proteus-wp1/trunk/S1.3.1/S1.3.1-S0.5/S1.3.1-S0.5-S0.1-ModelePapyrusScenarioMilitaire/scenario attaque AS/ATK_N0.di – Eclipse – /Users/bruno/Documents/workspace
index.rst
*LifeCycle.rst
DownloadingAndInstal
QuickStart.rst
Glossary.rst
*ATK_N0.di
Palette
Proteus Tools
Robot
System
Sensor
Actuator
Flow Port (IN)
Flow Port (OUT)
Service Port (Pro)
Service Port (Req)
Connector
Comment
Constraint
Comment/Constraint link
Cheat Sheets
ClassDiagram
AircraftRoot_ArchitectureDia...
ProtectionRoot_ArchitectureD...
MMSRoot_ArchitectureDiagram
NavigationRoot_ArchitectureD...
Problem
Solution

RobotML WorkFlow DashBoard

- RobotML WorkFlow
  - Phase 1: Problem Model Preparation
  - Phase 2: Problem Model Deployment
  - Phase 3: Problem generation
  - Phase 4: Solution Preparation

RobotML WorkFlow

The goal of this workflow is to facilitate its Problem provider user to create problems and its Solution provider user to provide solutions allowing them to interact with the **French Robotic Portal**.

As can be seen below in order to complete this workflow, there is a need on one hand to **create a Problem** and on the other hand to **provide a solution**. As the solution provider often works inside his proper environment in order to create the solution, the process develops more the **problem creation workflow** than the **solution creation workflow**.

To complete a Problem means to **publish an executable environmen** able to be used by the solution provider to test his solution and for the Problem provider to assess it. More details are to be found throughout the Workflow description.

Figure 1.3: *RobotML platform implementation of Life cycle*

Figure 1.4: *How life cycle is implemented*

short stories

Before entering in the niceties of the tools and life cycle, here follows two short stories to illustrate what is the meaning between all these beautiful words and wishes.

Betty and the navaid

We are introducing her Betty. She is working for a car company willing to provide to the market a new navigation aid based upon the interaction between the driver and a talking counterpart in the car. She discussed with the marketing department that explains why it would like to see such an aid and what would be the functionalities. She then discuss with the technical team that delivers constraints onto the system and the targeted system. Knowing the Robotic portal, she already has an account allowing her to ask for a new space in it where she will be able to create a problem. She characterises its access properties stating it is public for what concerns scenario and problem presentation. Those in charge of the portal, considering the information delivered agree on the creation of this space and then Betty is able to begin her job.

She then upload to the portal through the defined channel and a space she wanted to be public a document describing what she needs the system to do and a video to illustrate it. She opens now the *RobotML* platform in order to model the scenario, specifying her robot car, its associated system and the urban environment in which the scenario will take place. She precises in the model what should be measured through probes providing also a description of the criteria that will be derived of these metrics. Then, working with her IT department, she devised the generator and the environment models (libraries implementing walkers dynamic, car cinematic, lights, sensors physics, ...) allowing her to create a simulator in which the scenario can be shown. She verifies that she will be able generate directly from the simulator to her actual car robot system and uploads in a less public space this simulator and advertises video of it, presentation of the RobotML models and refined textual description of the metrics and criteuria used.

She waits for some time now seeing nonetheless that many people downloaded the scenario and videos. Some weeks later, she receives two requests from two academics and one SME willing to use the simulator to try some solutions on her problem. She discards one of the academic because the technology they are promoting is not inline with the main trend in her IT team and provides to the two other ones access to the simulator. Being able to issue such requests means implicitely that those Roboticians already have an account on the portal and thus request being accepted means for them to dispose of a specific webspace on the portal where they will be able to provide their solutions.

At this point many exchanges exist between Betty and these teams trying to understand what was the problem. As one of the solution needs an adaptation of the interfaces, Betty reissues an updated version of the simulator updating the model at the same time.

Some months later, the teams provides their solutions showing their results on the simulator using the criteria. Betty finds that the academic is better but nevertheless proposes to the two of them to show their solution on the actual robot. Meetings are held and test period is defined. Thanks to the possibility to generate directly to the Car navaid system, time to implement solutions is small enough.

One month later Betty can choose the academic solution and begins private discussions outside the portal scope in order to define IP and royalties with the final academic. This one not willing to work directly with a big company first transfer the knowledge to the SME that was competing with it.

And thus in a period of one year, Betty is able to tell her company that the new navaid system is ready to come out to the public as a release of the car navaid system.

Richard and his PhD

Richard is now in his second year of his PhD thesis which subject is an advanced servoing algorithm merging Radar and visual data to offer a proved and reliable localisation system in a urban environment. Unhappily, in his own testing environment he is unable to show flexibility of his methodology and associated implementation (urban car in some not very dense area). He goes to portal and shuffles through the proposed problems. Three are attracting his attention:

- A car manufacturer is proposing a problem in which the represented environment and associated sensors are very close to those he is using but more realistic;
- An aircraft manufacturer is having problems for its aircraft to manoeuvre on not well known airports;
- There are some new EM sensors that could be used onto humanoïd robots and perhaps their precision could be enough to allow the algorithm to wor. This problem is offered by a well known French Laboratory.

Richard won't be able to address each problem in his PhD time frame thus he needs more information than those proposed in the public part of these problems. Thus, as he already had an account on the portal, he send the needed forms as provided by the three problem's providers in order to access them completely. He is now able to download a complete description and most importantly, their models. Shuffling through the three of them, he identified very soon that unhappily it would be very difficult to adapt the aircraft system model to integrate his algorithm proposal. On the contrary, Work onto the car system seems really easy and with work, adaptation to humanoïd is possible.

Considering the first case, Richard now download the latest version of the *RobotML* platform and opens the car *problem*. He can directly identify the component that would host his algorithm. Interface is almost the same but not exactly. Deciding not to optimise code, he introduces in the model a type translation component that adapts inputs to his algorithm needs. Then as there is no need on the output, he generates thanks to the platform using generator provided by the car *problem provider* a new simulator that includes his algorithm. As it is he is now able to run tests and stores the data using the already defined probes and creating the metrics as defined in the problem description. Now using the criteria, he sees that he improves not so marginally localisation accuracy allowing the car to use less large roads. At this step he requests the *portal* responsible permission to open a *solution* page for him to record his results and do it. He publishes his achievements in th eproper review and goes on developing the theory associated with the algorithm using the insights gained thanks to the experimentation.

Considering the second case, due to his Thesis work, Richard postpones the integration to future times. Three months before delivering his text to reviewers and willing to ease his mind he goes back to his idea to test the algorithm onto humanoïd. He again asks for permission to use the problem as his lease was time suspended. Again he obtained this permission from the *problem provider* and using his existing version of the platform opens the model in order to better understand the *robot* system. It really has to be adapted to accept his algorithm as the EM *sensors* are not providing the same type of information as those he waited for; Moreover, synchronisation needed between the two types of sensors is not at all the same. Richard adapts the model many times discussing with the *problem provider* in order to understand what some parts of the architecture are meaning. After some efforts and adaptations at the limit of the system capability he is able to generate the complete robot application. Doing the same job as previously he is able to issue results to oppose to the criteria defined and there he sees the limits of the algorithm tested. Discussing with the *provider* he sees a complete new way to understand the theory leading to a major update of the software. There again Richard opens a :term:‘solution‘space and uploads his solution.

At this step, he considers that it could be a good idea to provide this code to the community and asks for opening a *module provider* space. When opened, he related this module to the two solutions developed and provide the model of the interface to be used inside the platform to be able to insert the algorithm in any other *problem*. Eventually he also provieds the source code through a packaged file and a C library on x86 architecture.

It's time now to present his final results in front of his PhD Jury. All goes smoothly but during the questions time, in the audience, someone requests some specific explanations onto the determinism of the solution and if it could be embedded in some limited memory architecture. Answering positively, Richard is surprised after the Jury felicited him to be proposed a job from the questioner in order to come in the car company that provided the *problem*.

Presentation of each step and suppporting tooling

These two stories are happy ones for sure and shows the type of impact that can be achieved if Robotic community agrees to use such a portal and its associated tools.

Formal Robotician Portal

This portal is the first supporting tool and a very important one. In fact it is the place where everything can be exchanged, worked between *providers* and *users*. It manages the access, provides the different needed spaces, *SVN* services, implements the life cycle.

RobotML

This tool is not usable in itself as it is the definition of the language that is implemented in the tool to come. Nevertheless, the definition of this language is not to be confused with its implementation through any tool. Anybody able to answer this definition can provide model (either *problem* or *solution*).

term

Note: It is of more importance to read the description of the *Life cycle* and the following pages.

1.2 Origin of RobotML

The *RobotML* platform has been created thanks to the French *ANR PROTEUS* project. It seems polite to tell some words about it.

1.2.1 PROTEUS as a project

PROTEUS is a French *ANR* project which goal is to close the existing gap between those working upon the robotic technologies (Research centres are often providing this type of contributions) and those using them (mainly companies willing to create products). In order to achieve such a goal this project is creating the different pieces of infrastructure needed:

1. **Robotic portal** which goals are to:
   (a) capitalise development, models, ...,
   (b) provide a download centre,
   (c) host challenges
2. **Modelling tool** which goal is to abstract robotic architecture representation in order to:
   (a) Use **Generation tools** which goal is to separate modelling problems with respect to implementation details and thus concrete robotic frameworks
   (b) Allow introduction of **verification tools** such as *DEVS* through its *VLE* embodiement

These two facets are to be verified through *OARP* allowing the project to debug and validate the complete infrastructure.

1.2.2 PROTEUS and the Robotic community

PROTEUS being a French Project was backed up and is directed towards the Robotic community as represented by its associated *GDR Robotique*. In this scope one of its study group (David Andreu from *LIRMM* and Cyril Novales from *PRISME*, Cyril Novales is participating to the project) is doing the link between the project and the French community.

Contacts were taken with the European Robotic community through the *EUROP / EURON* group and more specifically discussions exists between PROTEUS project and Herman Bruyninckx in order to correlate work doen on modelling tools in Europe and in PROTEUS. This action should lead at some point to an European initiative towards promotion of a standardised modelling language and its associated tooling.

CHAPTER

TWO

ROBOTML PLATFORM INSTALLATION

As explained previously, there will be throughout this documentation two points of view considered. The first one is the user point of view and the second one tries to provide enough information to those willing to participate to the platform development for him/her to install part or everything needed.

2.1 Modeller

You need to download the Eclipse Modelling tools. Then extract the modeller on your computer and launch it. Next, you need to install *Papyrus*, with *RobotML* extra feature, and *Subclipse*.

2.2 Papyrus

In **Help** menu, select "Install new software...". Clic on "Add" button, and add the right update site.

Eclipse version	Update site
Juno	http://download.eclipse.org/modeling/mdt/papyrus/updates/releases/juno
Kepler	http://download.eclipse.org/modeling/mdt/papyrus/updates/nightly/kepler

Install the "Papyrus UML" components.

2.3 RobotML

To install *RobotML*, *Papyrus'is required. See :ref:'Papyrus installation* to install it. In **Help** menu, select **Install new software...**, and choose the **Papyrus update site**. Uncheck the **group item** option. In the filter field, enter **RobotML** and select the binary element. You can now validate the installation.

2.4 RobotML extension

RobotML extension is not include as feature of *Papyrus*. You need to build the *RobotML extension* update site to install it.

2.4.1 RobotML extension building

Create a new workspace named as *RobotML_Extension*. Import from *git* the projet following project: - org.eclipse.papyrus.robotml.extension - org.eclipse.papyrus.robotml.extension.feature - org.eclipse.papyrus.robotml.extension.update

In the *org.eclipse.papyrus.robotml.extension.update* open the file *site.xml*. And clic and the **Build All** button.

The files *Artfact.jar* and *XXXXXX.jar* as been created in the update site project.

Now Got to the **Help** menu and choose **Install new software...**. Clic on the **Add** button and select to add a local update site.

Select the path of the update site project, and validate. In the update site viewer, check the *Dynamic validation* item.

Click next and accept the condition to finish the installation.

Ok the extension as been install on your platform.

Generator

2.5 Acceleo

To install *Acceleo* on your platform, use the Eclips's tool **Install Modeling Component**, from the ***Help** menu.

See also:

Acceleo presentation

2.6 XText

To install *Xtext* on your platform, use the Eclips's tool **Install Modeling Component**, from the ***Help** menu.

2.7 RTMaps

RTMaps has an extras feature of *Papyrus*. To use you need to have *Papyrus* and *RobotML* installed on yout platform.

See also:

Papyrus installation and *RobotML installation<RobotMLInstallation*

In **Help** menu, select **Install new software...**, and choose the **Papyrus update site**. Uncheck the **group item** option. In the filter field, enter **RTMaps** and select the binary element. You can now validate the installation.

See also:

RTMaps presentation

2.8 OROCOS

OROCOS has an extras feature of *Papyrus*. To use you need to have *Papyrus* and *RobotML* installed on yout platform.

See also:

Papyrus installation and *RobotML installation<RobotMLInstallation*

In **Help** menu, select **Install new software...**, and choose the **Papyrus update site**. Uncheck the **group item** option. In the filter field, enter **OROCOS** and select the binary element. You can now validate the installation.

See also:

Orrocos presentation

2.9 ARROCAM

ARROCAM has an extras feature of *Papyrus*. To use you need to have *Papyrus* and *RobotML* installed on your platform.

See also:

Papyrus installation and *RobotML installation<RobotMLInstallation*

In **Help** menu, select **Install new software...**, and choose the **Papyrus update site**. Uncheck the **group item** option. In the filter field, enter **Arrocam** and select the binary element. You can now validate the installation.

See also:

Arrocam presentation

2.10 Dynamic Validation

Dynamic validation is a RobotML extension include. It be installed, with the extension.

See also:

Dynamic validation presentation

Up

CHAPTER

THREE

HOW TO USE ROBOTML

3.1 Examples of Use

Contents:

3.1.1 How-to : MORSE and ROS

Pierrick Koch - GREYC - CNRS

anr-proteus.fr

Objectif: First step with ROS and MORSE.

ROS

Robot Operating System (ROS) can be seen as 2 main parts:

- A communication middleware, with related APIs (C++, Python, Java),
- A set of Robotic software (hardware abstraction, drivers, libs).

roscore: run a master node

```
roscore
```

TIP: to stop `roscore`: press Ctrl+C in the terminal

Basic commands:

roscd: change directory within ROS environment

```
Usage: roscd [package]
```

roslaunch: launching ROS nodes via `.launch` files

```
Usage: roslaunch [options] [package] <filename> [arg_name:=value...]
```

cf. Tutorials

---

rostopic

```
rostopic
rostopic is a command-line tool for printing information about ROS Topics.

Commands:
  rostopic bw    display bandwidth used by topic
  rostopic echo  print messages to screen
  rostopic find  find topics by type
  rostopic hz    display publishing rate of topic
  rostopic info  print information about active topic
  rostopic list  list active topics
  rostopic pub   publish data to topic
  rostopic type  print topic type

Type rostopic <command> -h for more detailed usage, e.g. 'rostopic echo -h'
```

---

rostopic: exemple

Publish & print a message on ROS bus:

```
rostopic pub /test std_msgs/String "Bonjour Proteus"
rostopic echo /test
```

|rostopic|

MORSE

Developped at LAAS/CNRS in Toulouse (France), MORSE is a generic robotic simulation platform based on Blender. Blender is a free and open source 3D modelisation platform. Which integrates differents methods for 3D rendering, animating, and a game engine used by MORSE for the simulation, as well as the physic engine Bullet. MORSE allow to build a robotic simulation by using Blender GUI / API.

Blender 2.5 introduced a new API allowing to interact with all scene-data through the Blender Python API, aka. `bpy`.

|blender-data-api|

|blender-roadmap|

MORSE

|morse|

MORSE GLSL

MORSE is meant to run with a graphic card compatible OpenGL Shading Language

ATI/AMD Radeon |AMD|

9x00, Xx00, X1x00, HD2x00, HD3x00 & + `sudo apt-get install 'xserver-xorg-video-radeon` <http://apt.ubuntu.com/p/xserver-xorg-video-radeon>'_

nVidia |nvidia|

Geforce FX, 6x00, 7x00, 8x00, 9x00, GTX 2x0 & + `sudo apt-get install 'nvidia-current` <http://apt.ubuntu.com/p/nvidia-current>'_

cf. Blender System Requirements Blender 2.48 Realtime GLSL Materials

Use the tool `/usr/bin/jockey-gtk` to install additional drivers. You can use `lspci|grep VGA` to know your card's name.

MORSE config

- Coordinates: main droite

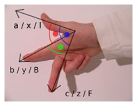
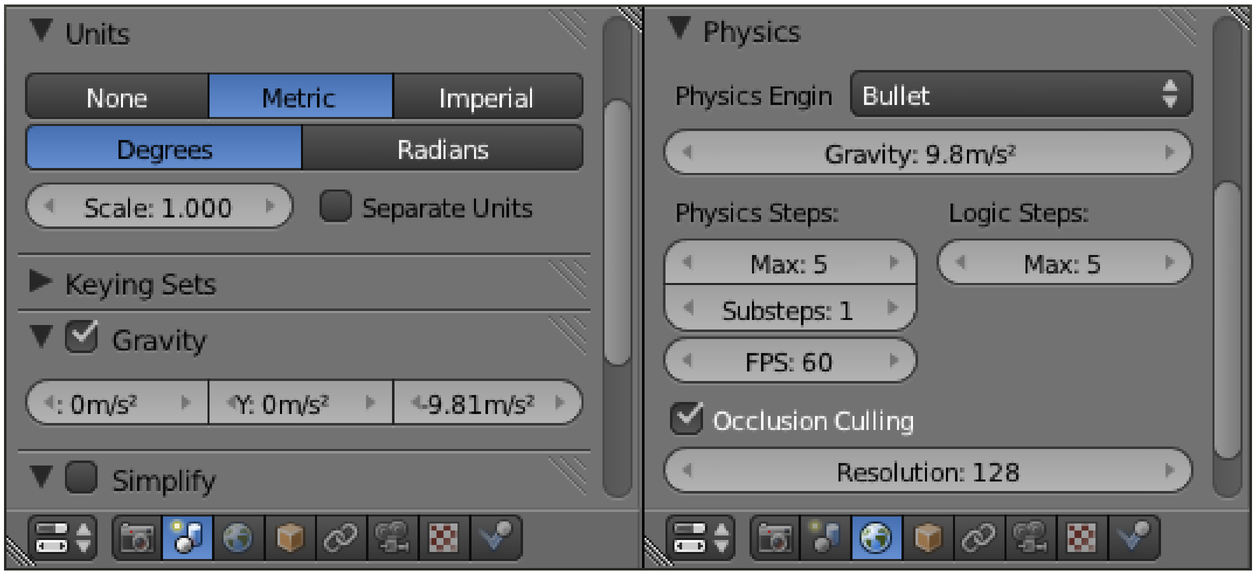


Figure 3.1: right

- Units: metric, angle: degree in UI, radian in simulation
- ROS: ROS Standard Units of Measure and Coordinate Conventions



Units

None [✓ Metric] Imperial

[✓ Degrees] Radians

Scale: 1.000 Separate Units

Keying Sets

[✓] Gravity

X: 0m/s² Y: 0m/s² Z: -9.81m/s²

Simplify

Physics

Physics Engin: Bullet

Gravity: 9.8m/s²

Physics Steps: Max: 5, Substeps: 1, FPS: 60

Logic Steps: Max: 5

[✓] Occlusion Culling

Resolution: 128

Figure 3.2: metric

MORSE & ROS

- Naming convention of topics

`/Robot/Composant` alphanumeric CamelCase

```
rostopic list -v

Published topics:
 * /ATRV/CameraMain [sensor_msgs/Image] 1 publisher
 * /rosout [rosgraph_msgs/Log] 1 publisher
 * /ATRV/Pose_sensor [nav_msgs/Odometry] 1 publisher
 * /rosout_agg [rosgraph_msgs/Log] 1 publisher
 * /ATRV/Odometry [geometry_msgs/Twist] 1 publisher
 * /ATRV/Sick [sensor_msgs/LaserScan] 1 publisher

Subscribed topics:
 * /rosout [rosgraph_msgs/Log] 1 subscriber
 * /ATRV/Motion_Controller [geometry_msgs/Twist] 1 subscriber
```

MORSE CLI

Once installed, MORSE is launch by the command `morse {check,edit,run} [scene]`, it accept differents parameters:

- .blend file (simulation scene)
- .py file (scene builder script)

ie:

```
morse edit my_builder_script.py

morse edit my_saved_scene.blend
```

... respectively result in:

1. edit a scene build through a script in Blender GUI.
2. edit a scene saved in a .blend file in Blender GUI.

MORSE: simulation loop

|morse-main-loop|

MORSE: scene builder API

|morse-builder|

cf. openrobots.org/morse/doc/latest/dev/builder.html

Demo: intro

robot made of one controler (v, ω)

```
!python
from morse.builder import *

# Add the robot
simplebot = Robot('atrv')

# Add a motion controler
motion = Actuator('v_omega')
motion.translate(z=0.3)
# Connect the controler to the robot
simplebot.append(motion)

# Configure the controler with the ROS middleware
motion.configure_mw('ros')

# Setup the environment
env = Environment('indoors-1/indoor-1')
env.aim_camera([1.0470, 0, 0.7854])
```

(code)

Demo: intro (2)

commands to run

start a ROS master node

```
roscore
```

start MORSE with a builder script (in another Terminal)

```
morse edit /usr/local/share/morse/examples/scenarii/ros_example.py
```

Press "`P`" in the 3D view to start the simulation (Game Engine)

Use "`CTRL`" + mouse, and "`Z, Q/A, S, D/W`" to move the camera

publish a message on the controler topic (in another Terminal)

```
rostopic pub -1 /ATRV/Motion_Controller geometry_msgs/Twist [1,0,0] [0,0,1]
```

Demo: intro (video)

(video)

Demo: advanced

obstacles avoidance w/ laser

```
mkdir -p ~/work/ros-addons
rosinstall ~/work/ros-addons /opt/ros/electric/ \
    http://anr-proteus.github.com/proteus.rosinstall
source ~/work/ros-addons/setup.bash
roscd proteus_demo && git pull && cd data
morse run wifibot.py
```

run the master node (in another Terminal)

```
roscore
```

launch our node to control the robot (in another Terminal)

```
source ~/work/ros-addons/setup.bash
roslaunch proteus_demo AvoidObstacleLaser.launch
```

TIP: if `rosinstall` is not recognized, install it as:

```
sudo pip install -U rosinstall
```

Demo: advanced (video)

(video)

Demo: advanced (code)

```
!python
# callback for each laser scan
def handle_sick(msg):
    mid = len(msg.ranges) // 2
    cmd = Twist()
    # stop if an object is less than 2m in a 30deg angle
    halt=False
    for distance_to_object in msg.ranges[mid-15:mid+15]:
        if distance_to_object < 2:
            halt=True
            break
    if halt:
        # rotate on the wider scanned side
        if sum(msg.ranges[:mid]) > sum(msg.ranges[mid:]):
            cmd.angular.z = -1
        else:
            cmd.angular.z = 1
    else:
        cmd.linear.x = 1
    # publie the command to the controler (Twist msg)
    topic.publish(cmd)

if __name__ == '__main__':
    rospy.init_node('wander')
    topic=rospy.Publisher('/ATRV/Motion_Controller', Twist)
    rospy.Subscriber('/ATRV/Sick', LaserScan, handle_sick)
    rospy.spin()

# http://ros.org/doc/api/sensor_msgs/html/msg/LaserScan.html
# http://ros.org/doc/api/geometry_msgs/html/msg/Twist.html
```

(code)

Demo: Orocos

The example's code is available on GitHub

```
roscd proteus_demo/data
morse run wifibot.py

rosmake proteus_orocos_obstaclelaser
roslaunch proteus_orocos_obstaclelaser AvoidObstacleLaser.launch
```

|morse-orocos|

RViz (video)

```
rosrun rviz rviz
```

launch rviz with a configuration file

```
roscd proteus_demo/data
rosrun rviz rviz -d rviz.vcg
```

(video)

Outdoor (video)

```
roscd proteus_demo/data
morse run wifibot.py
```

(video)

Exploration (Bosch demo w/ Stage)

```
sudo apt-get install ros-electric-bosch-common
roslaunch explore_stage explore.launch

roscd explore_stage
rosrun rviz rviz -d explore.vcg
```

|exploration|

Exploration (Bosch demo w/ MORSE)

RTMaps

|rtmaps|

TIPS

MORSE create cache files in the current directory,

```
rm scene.*.blend
```

allow you to delete those files.

Resources

Blender Cookie: cgcookie.com/blender

Blend Swap: blendswap.com (ie. Rover model)

Blender Wiki: wiki.blender.org

Build a robot

That's all Folks!

short link to this presentation: bit.ly/proteus2

1 page doc: bit.ly/proteus2md

sources on GitHub: bit.ly/proteus-src

made with landslide

3.1.2 Your First Scenario - attack an airport

Contents:

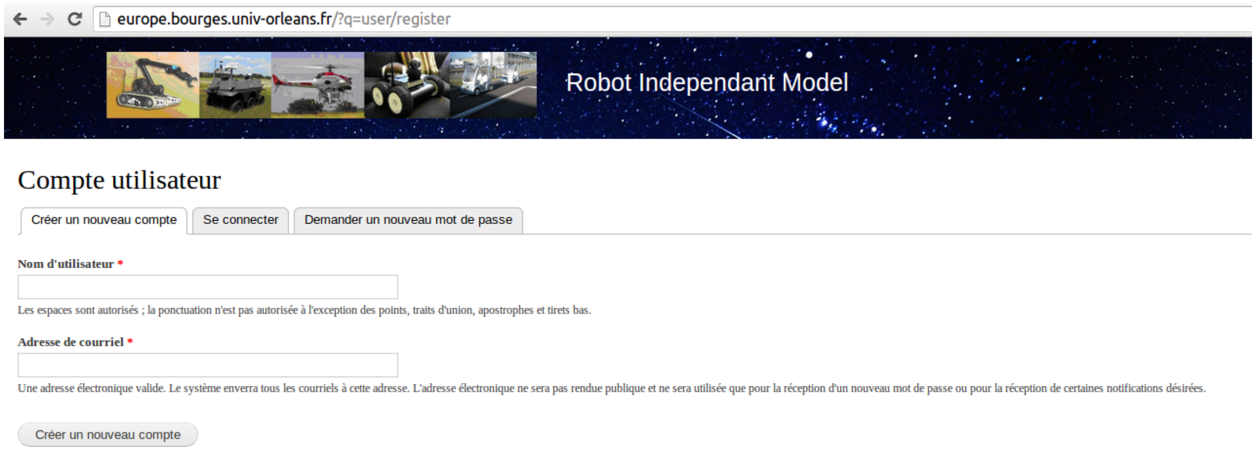
Your First Scenario - attack an airport - I initiate my project

RobotML portal

Open our web browser, and go to RobotML web site .

Login on the portal in using our account. if We don't have an account, then we go to register us on the portal to use it.

Account register On the portal's home page, clic on 'Regiter account' link.


europe.bourges.univ-orleans.fr/?q=user/register
Robot Independant Model

Compte utilisateur

Créer un nouveau compte
Se connecter
Demander un nouveau mot de passe

Nom d'utilisateur *

Les espaces sont autorisés ; la ponctuation n'est pas autorisée à l'exception des points, traits d'union, apostrophes et tirets bas.

Adresse de courriel *

Une adresse électronique valide. Le système enverra tous les courriels à cette adresse. L'adresse électronique ne sera pas rendue publique et ne sera utilisée que pour la réception d'un nouveau mot de passe ou pour la réception de certaines notifications désirées.

Complete the informations on the registering page. The minimal required information are :

- Name
- Surname
- Mail
- Password

When it finish clic on the 'OK' button. We go to receive a confirmation e-mail for the registration. This e-mail contain the login site identification, and the server url.

Login on portal On the portal's home page, enter our identification site in the login fields, and clic on the 'OK' button. The account's home page it be shown.

Now go to *descibe our scenary* .

Your First Scenario - attack an airport - i describe the scenario

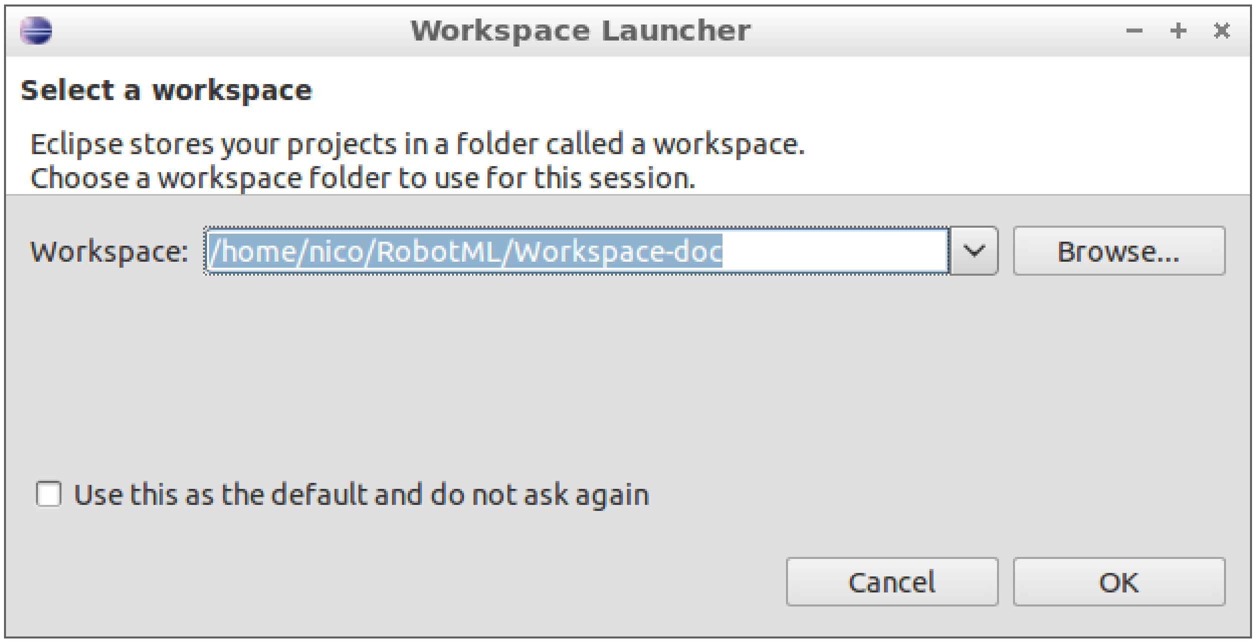
TBD

Your First Scenario - attack an airport - Launch RobotML tools

After describing your scenario, it's time to model using the **RobotML platform**.

Launch **RobotML platform** with the desktop icon .

We should to choose our workspace, and clic OK.


Workspace Launcher

Select a workspace

Eclipse stores your projects in a folder called a workspace.
Choose a workspace folder to use for this session.

Workspace: /home/nico/RobotML/Workspace-doc

☐ Use this as the default and do not ask again

Attention: Do not remember the workspace if we work often with another workspace. Else we should to use the **Switch workspace** function in the **File** menu.

Now the **RobotML platform** is launched, we go to modeling our problem.

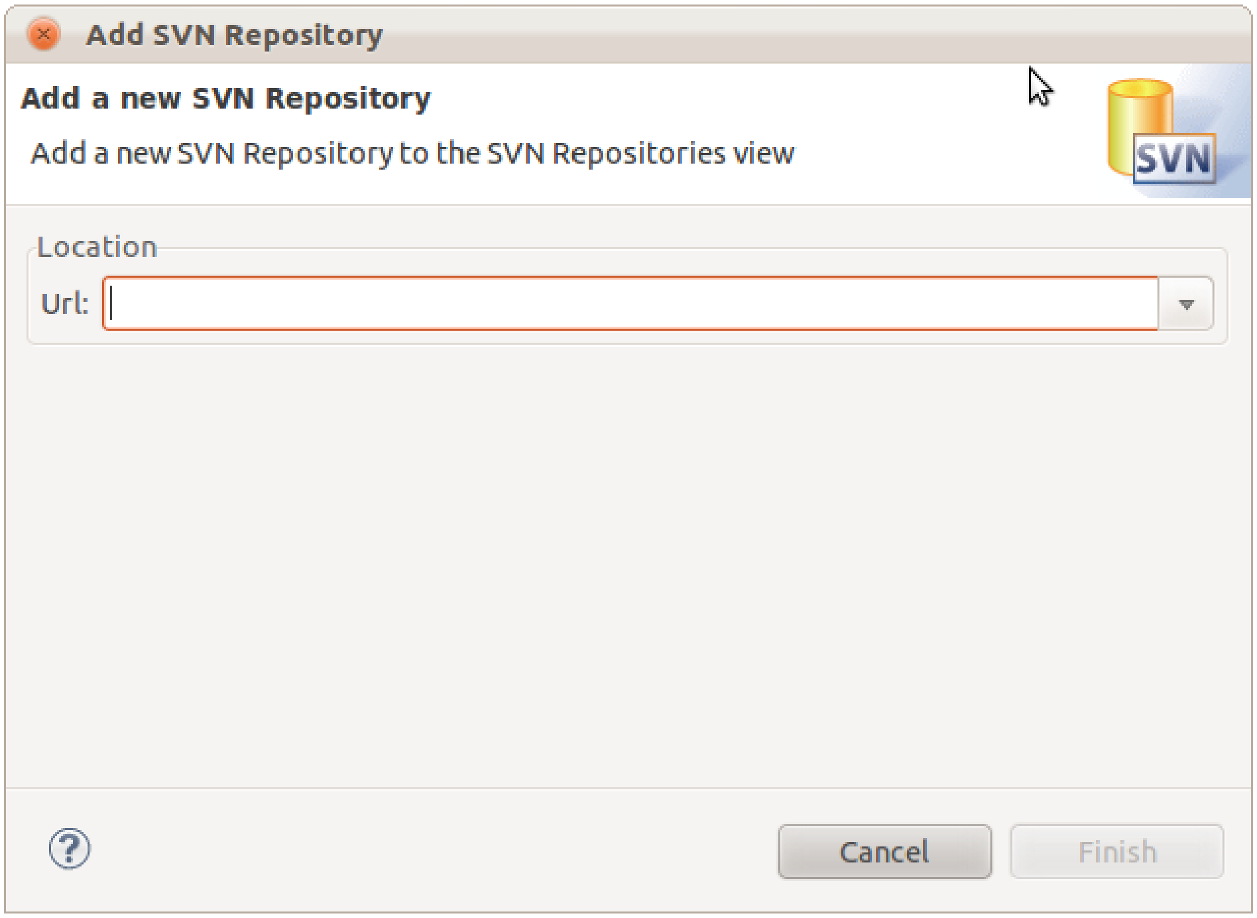
Your First Scenario - attack an airport - Problem model creation

Now we go to create a RobotML project and model our problem.

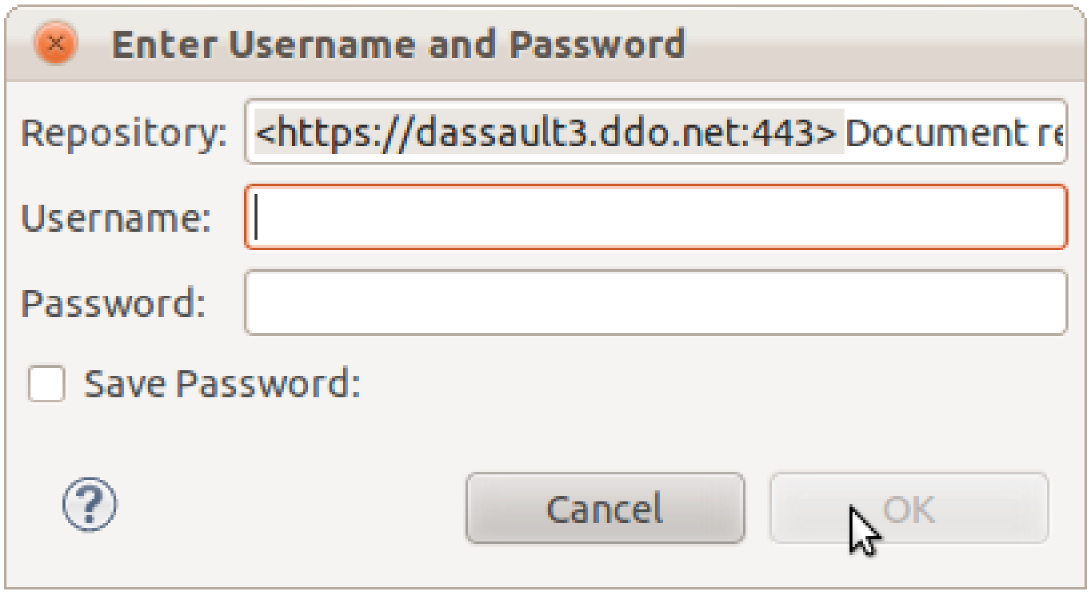
Create a RobotML project

Register our SVN reository With the **Window** menu, open the **SVN repository exploring** perspective. Press **Add a new SVN location** button in the **Project explorer** view.

Enter the url SVN repository given by the **RobotML portal** site in the **Add SVN repository** window.



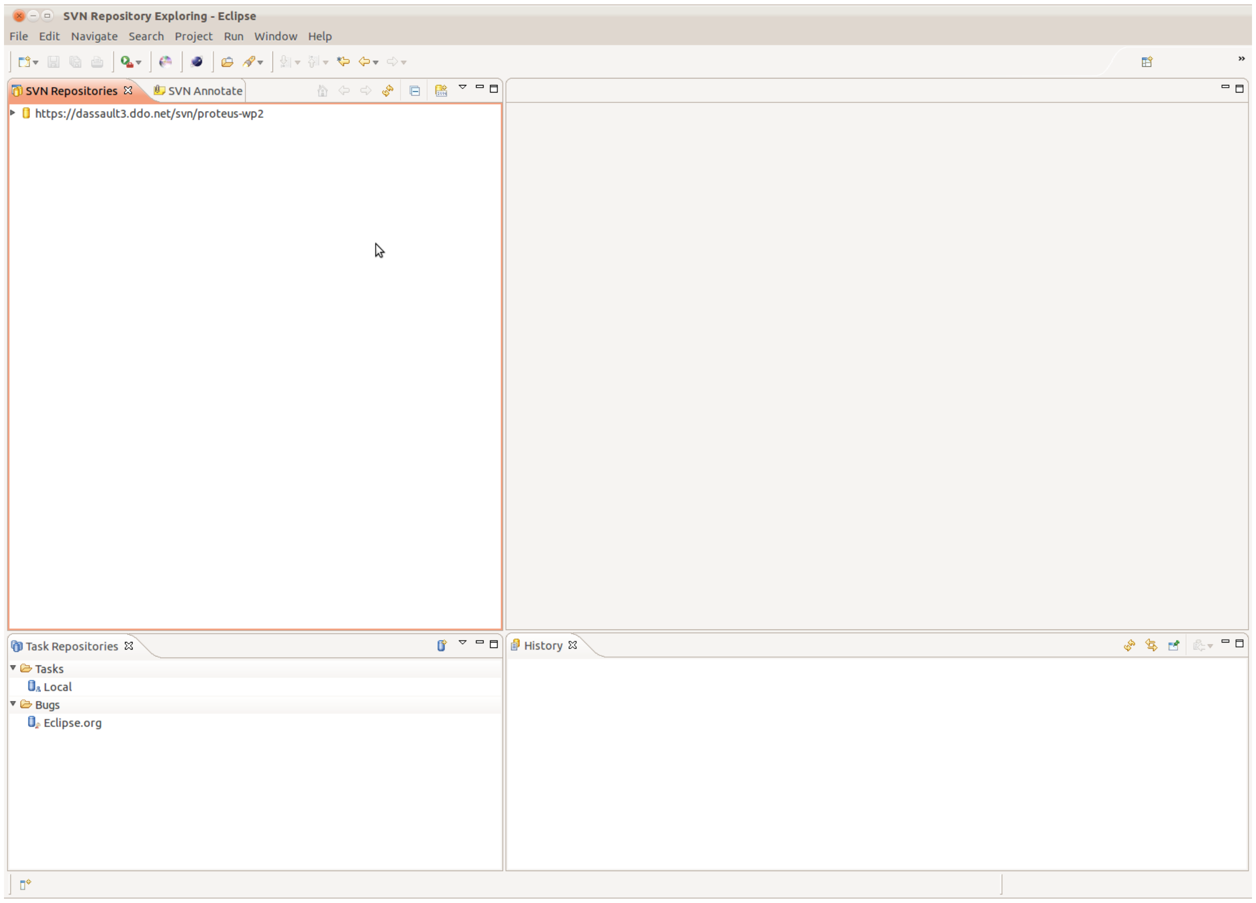
Enter the **RobotML portal** authentification.

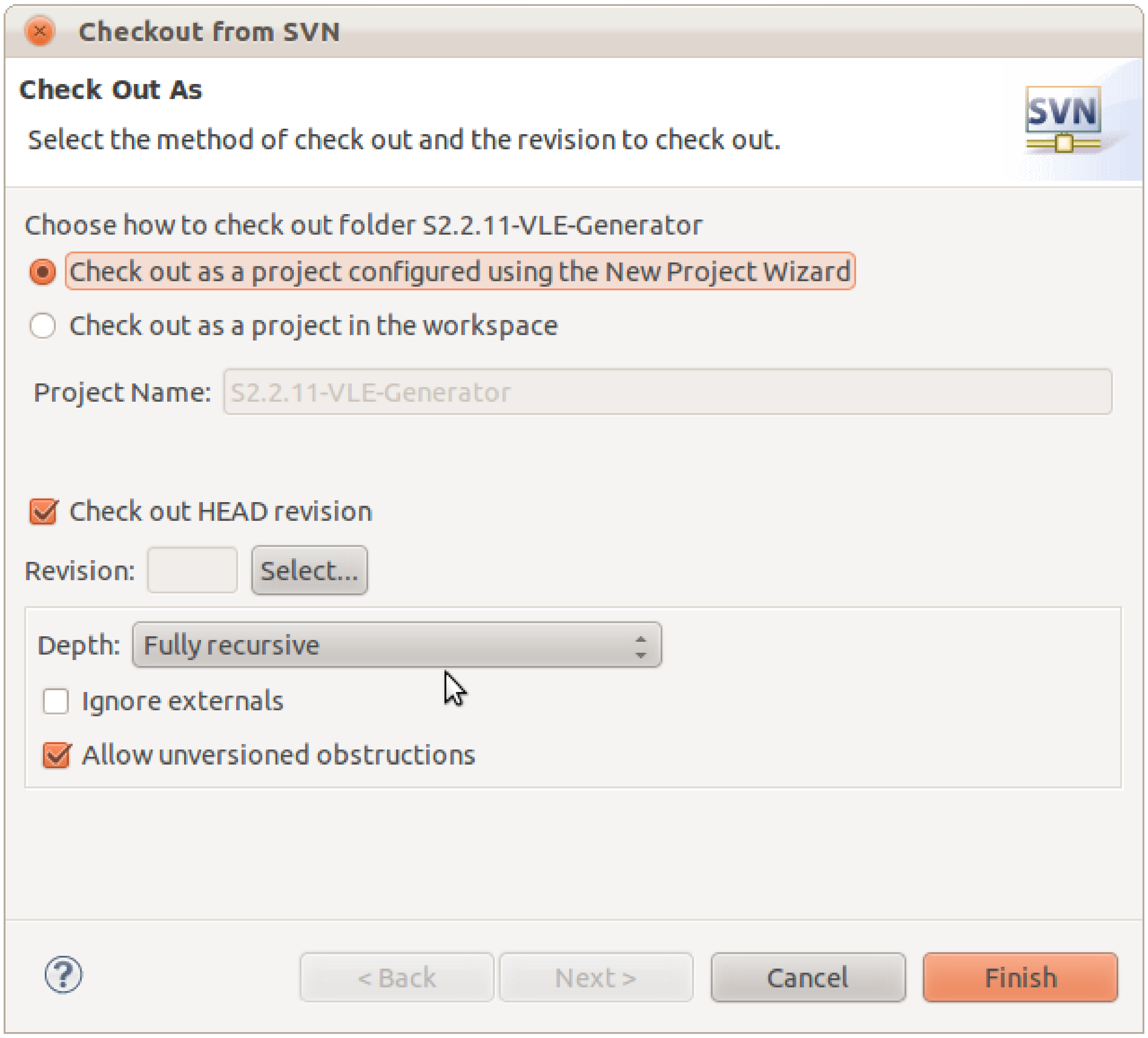


Repository: <https://dassault3.ddo.net:443> Document re

Note: Can save the authentification informations for next time.

Next clic on finish. The new SNV repository will be added in the **Project explorer** view.


SVN Repository Exploring - Eclipse
File Edit Navigate Search Project Run Window Help
SVN Repositories
SVN Annotate
https://dassault3.ddo.net/svn/proteus-wp2
Task Repositories
Tasks
Local
Bugs
Eclipse.org
History



Select the method of check out and the revision to check out.

Choose how to check out folder S2.2.11-VLE-Generator

[✓ Check out as a project configured using the New Project Wizard]

Check out as a project in the workspace

Project Name: S2.2.11-VLE-Generator

[✓ Check out HEAD revision]

Revision:

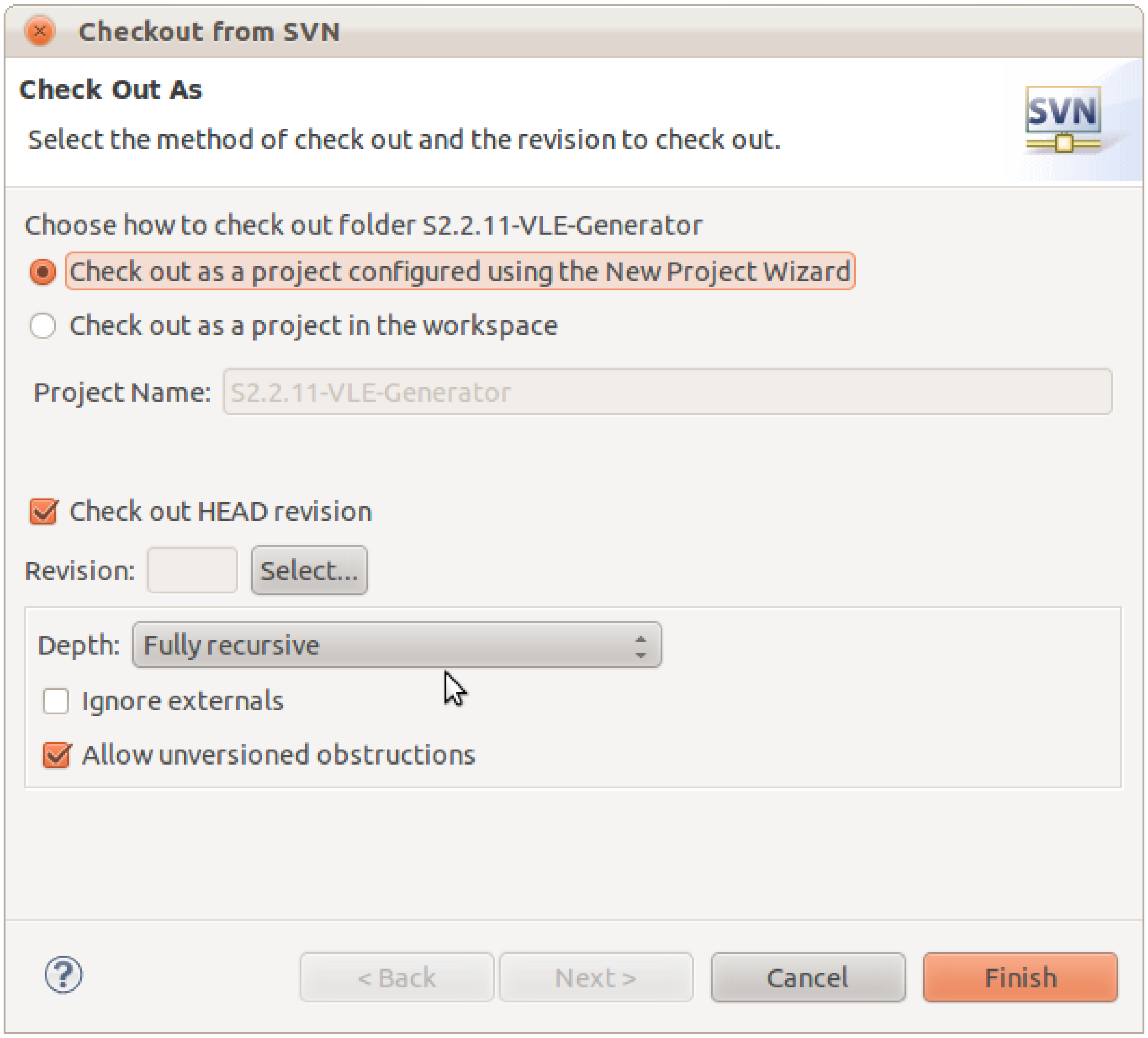
Depth: Fully recursive

Ignore externals

[✓ Allow unversioned obstructions]

Checking out our SVN repository and create the RobotML project

In the **SVN repository exploring** perspective, go in the **SVN repositories** view. Select the new location added and open it. Select the **trunk** node, and do right clic to select the **Checkout** command. Select *Check out as a project configured using new Project Wizard*.


Checkout from SVN

Check Out As

Select the method of check out and the revision to check out.

Choose how to check out folder S2.2.11-VLE-Generator

● Check out as a project configured using the New Project Wizard

○ Check out as a project in the workspace

Project Name: S2.2.11-VLE-Generator

✓ Check out HEAD revision

Revision: Select...

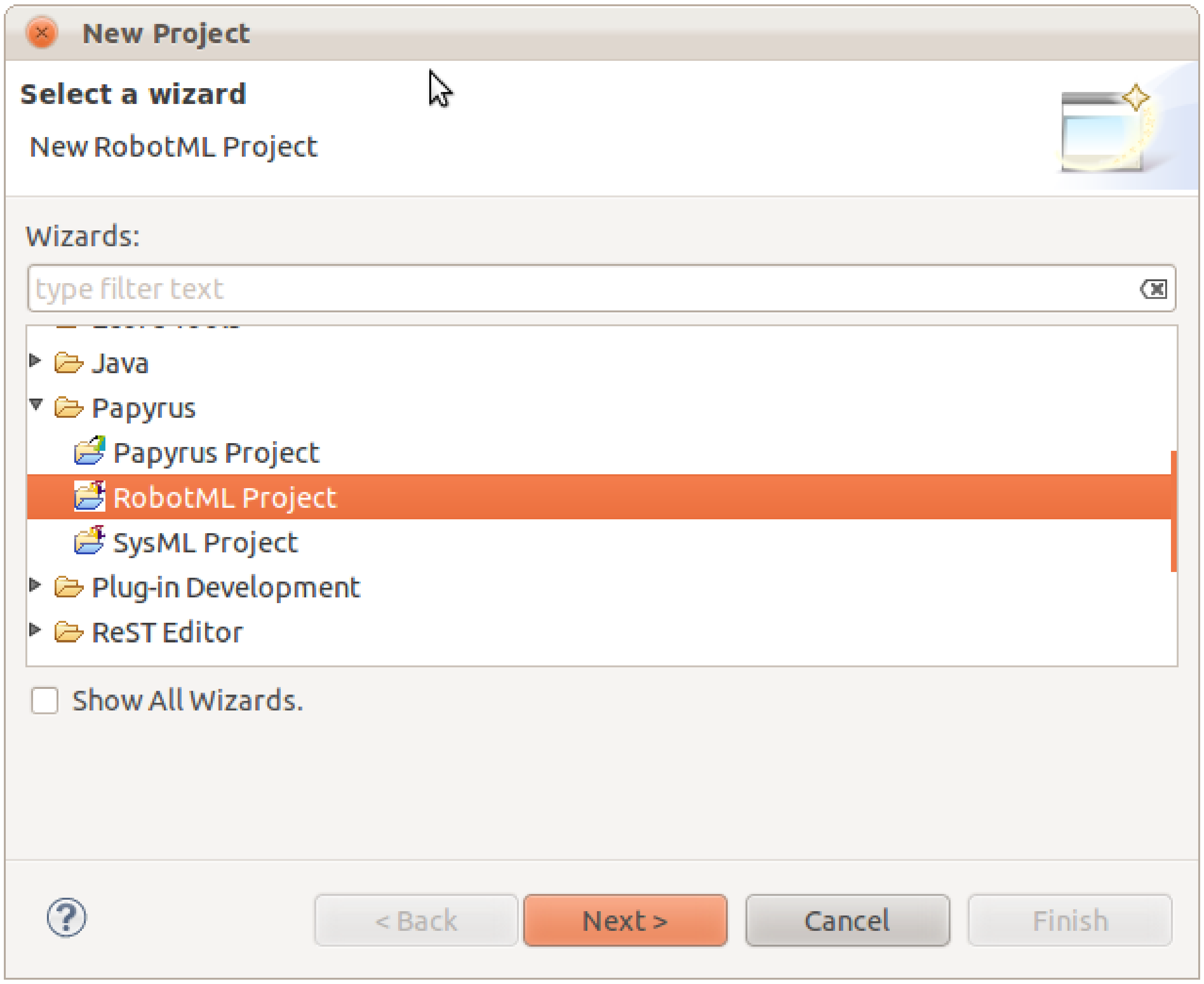
Depth: Fully recursive

☐ Ignore externals

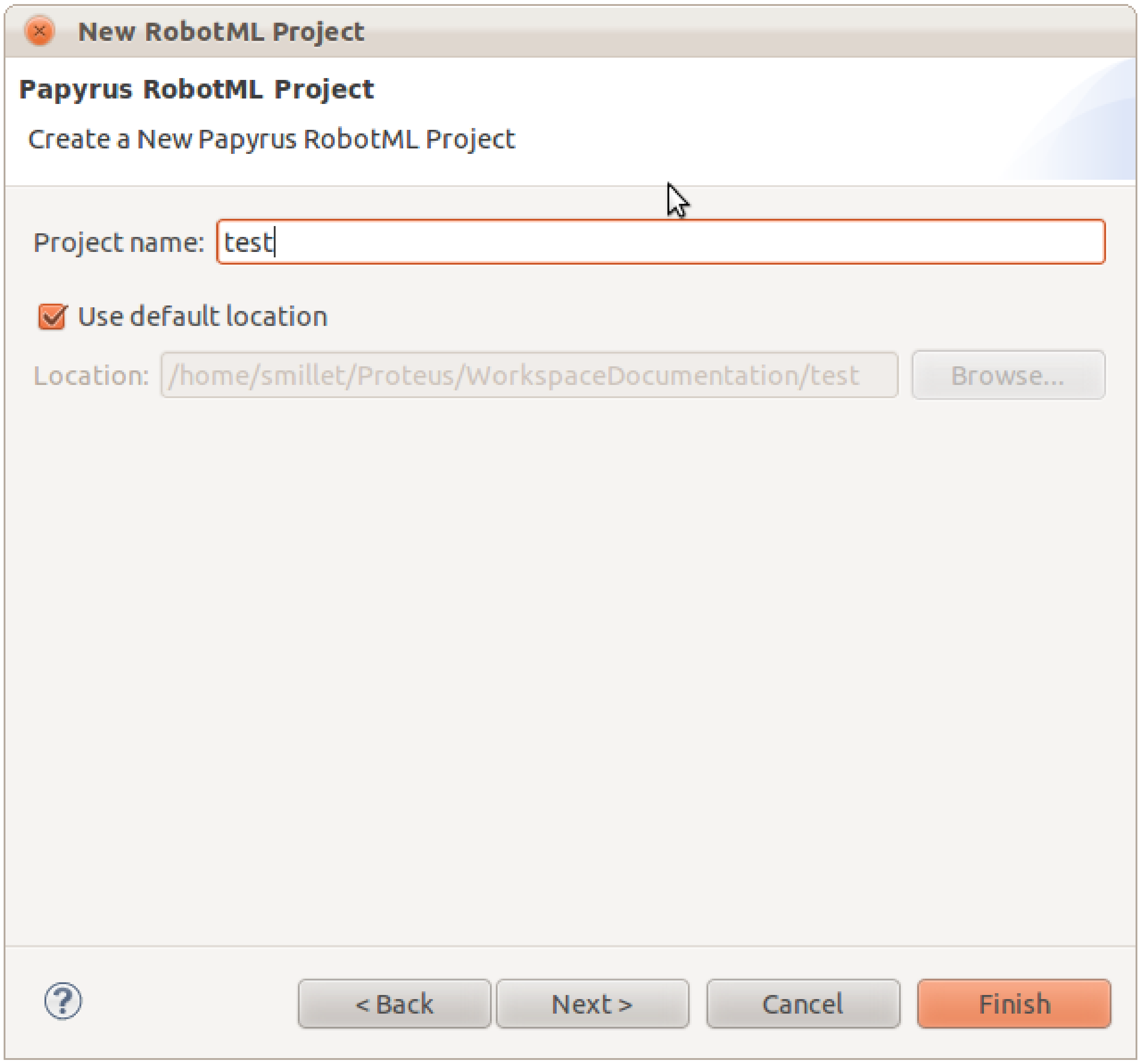
✓ Allow unversioned obstructions

Important: Should checking out the HEAD revision.

At next, select the **RobotML project** in the **Papyrus** category.



Name your RobotML project as “ATK_Scenario”, and clic **Finish**.


New RobotML Project

Papyrus RobotML Project

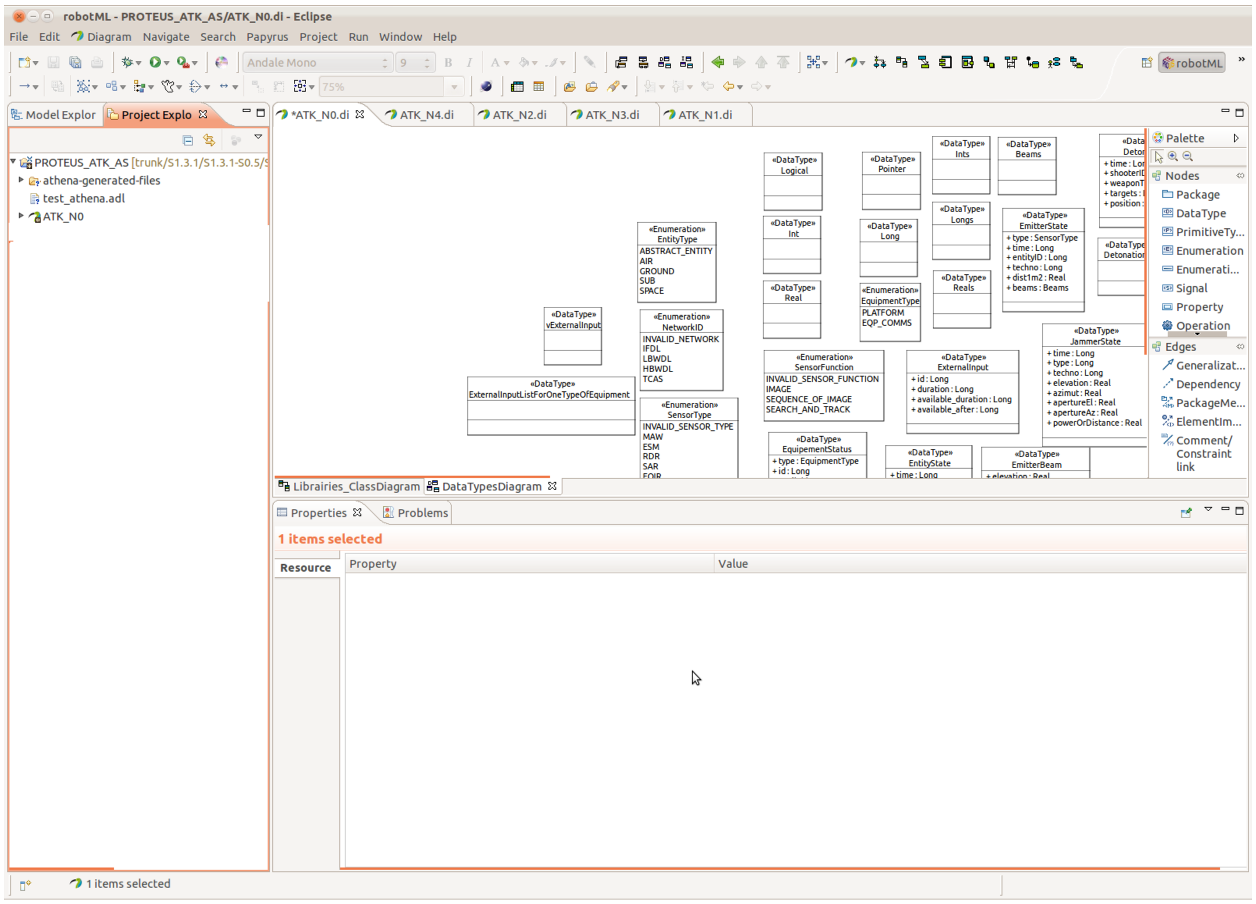
Create a New Papyrus RobotML Project

Project name: test

[✓] Use default location

Location: /home/smillet/Proteus/WorkspaceDocumentation/test

With the **Window** menu, select the **Java** perspective. We should to view the new **ATK_Scenario** project in the **Project explorer** view. This projct contains the new RobotML project created.



Modeling our problem Now we have to checkout our SVN repository. We go to model our problem, but now we have a problem:

what How to deasign our scenario?

how We should to separate on any part.

We go to deasign our model on 4 parts:

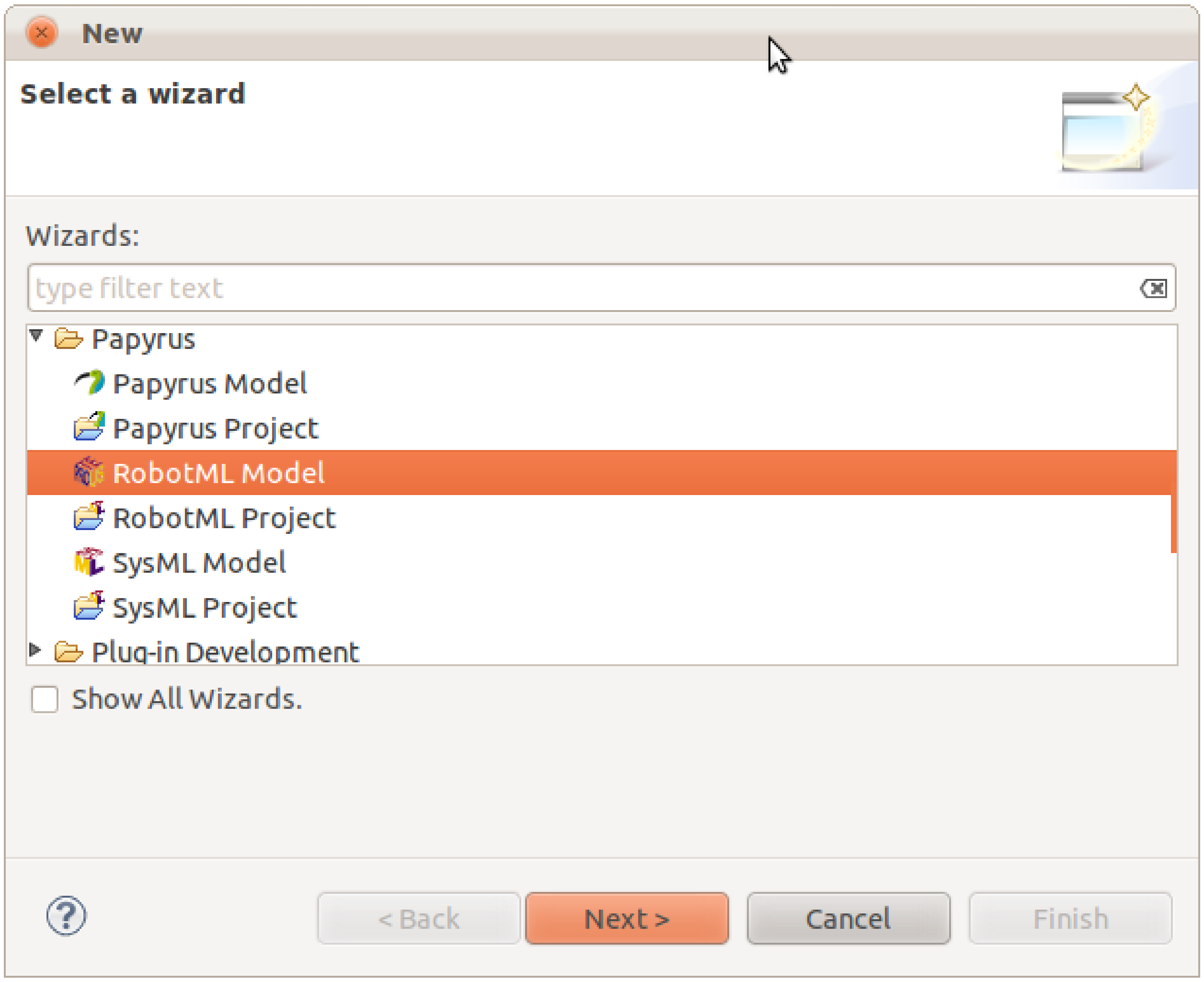
Project level	Model name	Model description
0	ATK_N0	Contain required model object (datatype, external library).
1	ATK_N1	Contain the basic model object
2	ATK_N2	Contain the system object model.
3	ATK_N3	Contain the robot object model
4	ATK_N4	Contain the scenary oject model.

what Why do this?

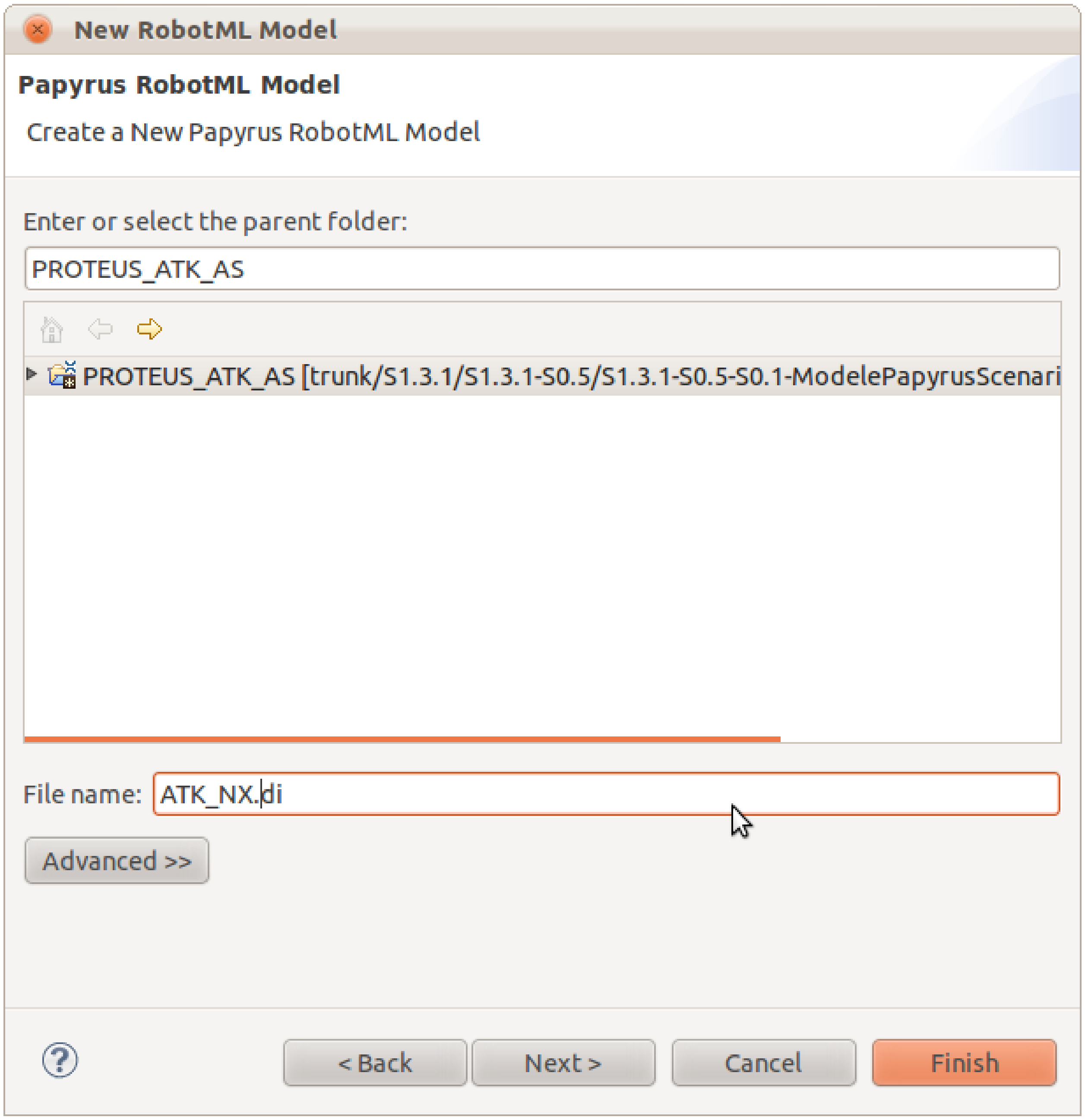
how Because if we want, **we could shared the model element with other model**, and it's **easiest to debug a small model**.

Model creation In first time we have created the model level 0. Now we go to create the other levels.

Adding a new model in the project In the **Project explorer** view, select the project node and do right clic to select the *New/Other...* submenu. Choose **RobotML Model** in the **Papyrus** category.



Next name the new model as **ATK_N1**, and clic on finish.



Papyrus RobotML Model

Create a New Papyrus RobotML Model

Enter or select the parent folder: PROTEUS_ATK_AS

PROTEUS_ATK_AS [trunk/S1.3.1/S1.3.1-S0.5/S1.3.1-S0.5-S0.1-ModelePapyrusScenari

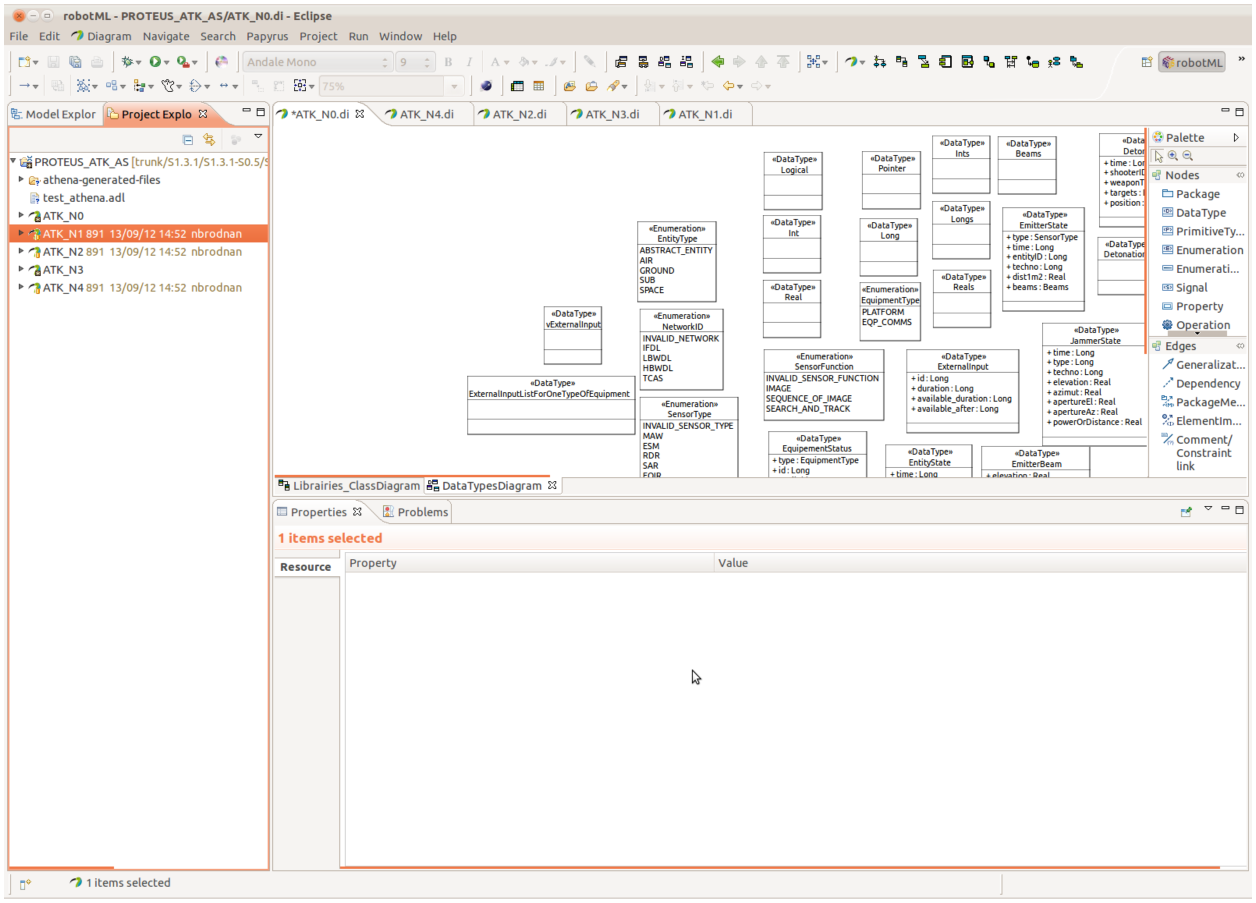
File name: ATK_NX.di

The new model is added to the workspace.

Note: We should repeat this operation to create the other model (ATK_N2, ATK_N3 and ATK_N4).

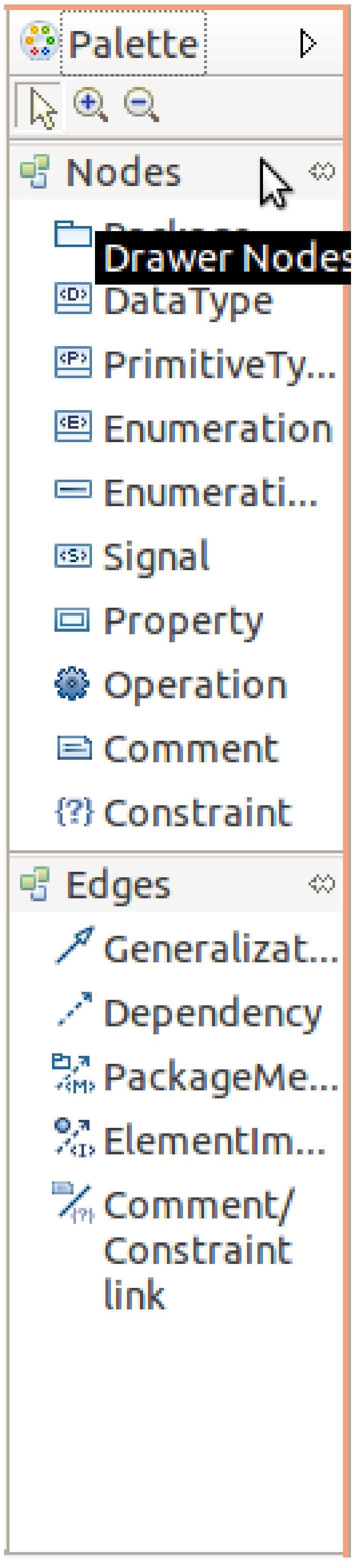
Adding required object in the model In the **Project explorer** view, open the **ATK_N0** RobotML model, and with the menu **Window/Open perspective...** open the **Model explorer** view. Select the root node model and rename it as **RequiredObjectModel**. Select root model node and do rigth clic. In the context menu, select **Add Child/new package**. Name it **DataTypes**.

Note: To deasign the model, it's more easy to work with diagram. Don't create element with menu if this is possible (except package and enumeration values).

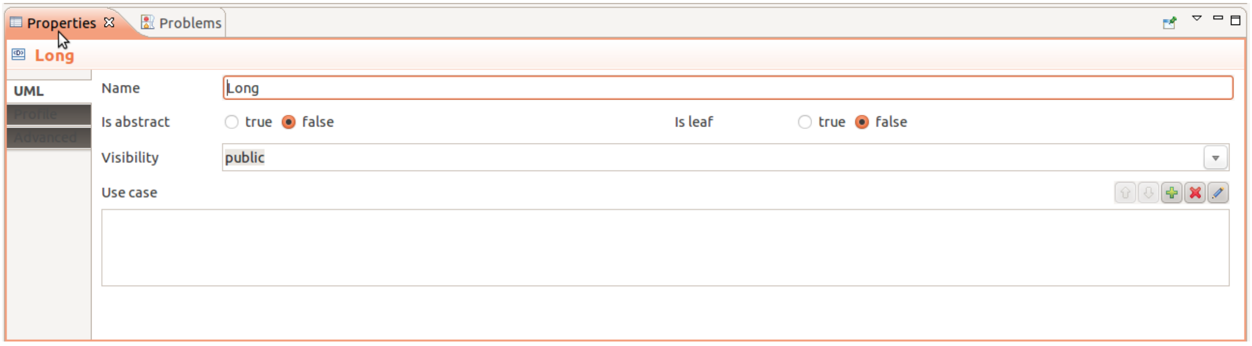

robotML - PROTEUS_ATK_AS/ATK_N0.di - Eclipse
File Edit Diagram Navigate Search Papyrus Project Run Window Help
Model Explor
Project Explo
PROTEUS_ATK_AS [trunk/S1.3.1/S1.3.1-S0.5/S
athena-generated-files
test_athena.adl
ATK_N0
ATK_N1 891 13/09/12 14:52 nbrodnan
ATK_N2 891 13/09/12 14:52 nbrodnan
ATK_N3
ATK_N4 891 13/09/12 14:52 nbrodnan
*ATK_N0.di
ATK_N4.di
ATK_N2.di
ATK_N3.di
ATK_N1.di
Palette
Nodes
Package
DataType
PrimitiveTy...
Enumeration
Enumerati...
Signal
Property
Operation
Edges
Generalizat...
Dependency
PackageMe...
ElementIm...
Comment/ Constraint link
Librairies_ClassDiagram
DataTypesDiagram
Properties
Problems
1 items selected
Resource
Property
Value
1 items selected

Create model's DataType Selet the **DataTypes** node and do right clic. In the context menu, **Add diagram/new DataType diagram**. name it **DataType_diagram**.

Now all action to add datatypes elements should to be done in this diagram. For this we should to use the **RobotML tool panel**, locate on the right side of the diagram.



Create basic datatype Select the **DataType component** in the **RobotML tool panel** and drag it on the diagram. Go it the **Properties** view, and rename the new datatype as **Long**. Add default value as **0** with a **Literal Integer**.


Properties ⊠ Problems

Long

UML

Name: Long

Is abstract: true [✓ false]   Is leaf: true [✓ false]

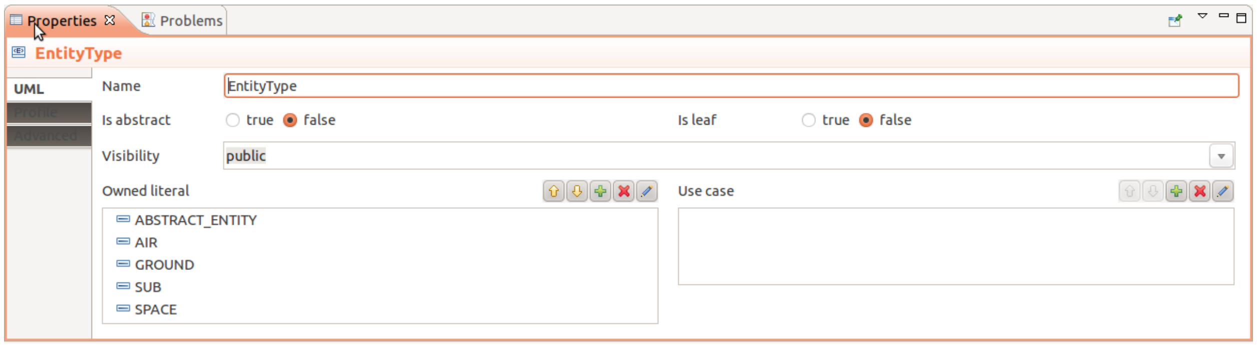
Visibility: public

Use case

Add the following basic datatype:

- Logical
- Pointer
- Real
- Int

Create enumeration In the **RobotML tool panel**, select the **Enumeration component** and drag it on the diagram. Go on the **Property** view and name the new enumeration **EntityType**. In th **RobotML tool panel**, select the **Enumeration literal** and drag it in the created enumeration. Select it and name it **ABSTRACT_ENTITY**.


Properties ⊠ Problems

EntityType

UML

Name: EntityType

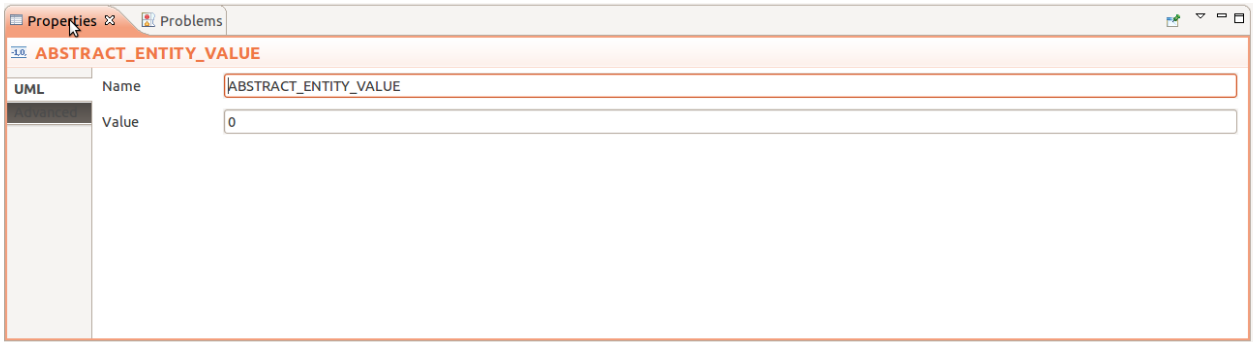
Is abstract: true [✓ false]   Is leaf: true [✓ false]

Visibility: public

Owned literal:
- ABSTRACT_ENTITY
- AIR
- GROUND
- SUB
- SPACE

Use case

In the **Model explorer** view, select the **EntityType** node and expand it. Select the **ABSTRACT_ENTITY** node and do right clic. Select **Add child / new LiteralValue** in the context menu. Name the lietral as **ABSTRACT_ENTITY_VALUE** and put his value at **0**.


Properties ⊠ Problems

ABSTRACT_ENTITY_VALUE

UML

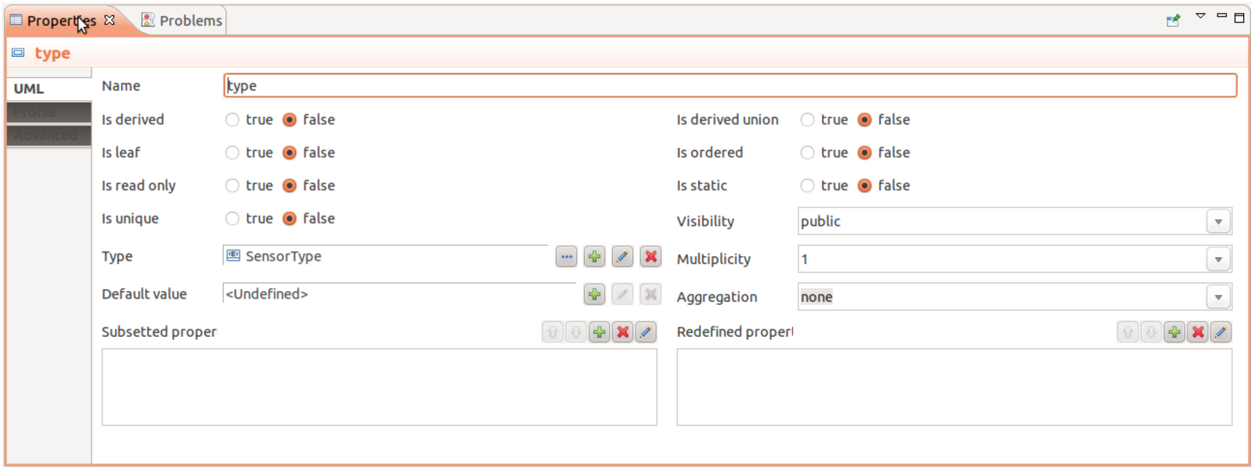
Name: ABSTRACT_ENTITY_VALUE

Value: 0

Repeat the operation to complete the **EntityType** enumeration as *AIR(0), GROUND(2), SEA(3), SUB(4), SPACE(5)*.

Create the enumeration referenced in the *Annex*.

Create composed datatype In the **RobotML tool panel**, select the **DatatType component** and drag it on the diagram. Go it the **Properties** view, and rename the new datatype as **SensorTechno**. In the **RobotML tool panel**, selcet the **Attribute component** and drag it in the new datatype created. Select the new attribute and edit their properties in the **Property** view. Name it as **id**, and select **Long** for the type value. Add a another attribute to the datatype. his name is **range** and is type is **Real**.

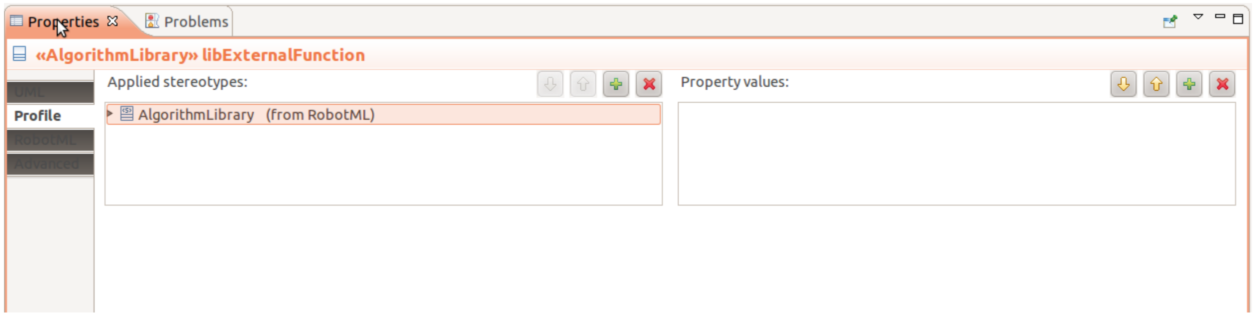


type

Name	type		
Is derived	true [✓ false]	Is derived union	true [✓ false]
Is leaf	true [✓ false]	Is ordered	true [✓ false]
Is read only	true [✓ false]	Is static	true [✓ false]
Is unique	true [✓ false]	Visibility	public
Type	SensorType	Multiplicity	1
Default value	<Undefined>	Aggregation	none
Subsetted proper		Redefined proper	

Create the composed datatypes referenced in the *Annex*.

Create our external librairy In the **Model explorer** view, select the root model node. Do right clic and select **Add child / new package...** to add the **Librairies** package. Select the **Librairies** package and do right clic and add a new component diagram selection **Add diagram / new RobotML component diagram...** in the context menu. Name the diagram as **ExternalLibrairies_ComponentDiagram**. In the new Diagram select **Component** element in the **RobotML tool panel** and drag it on the diagram. Name the component as **libExternalFunction**. In the **Property** view, select **Profil** and assign the **AlgorithmLibrary** profil.



«AlgorithmLibrary» libExternalFunction

Applied stereotypes:
AlgorithmLibrary (from RobotML)

Property values:

Create our external librairy's functions In our scenario we have two kinds of function:

- A simple function
- An interaction
- A processing

A simple function

In the **ExternalLibrairies_ComponentDiagram** view, select the **Operation** element in the **RobotML tool panel** and drag it on the **libExternalFunction** component. A new operation has added in the external librairy component, select it and rename as **manageSASystem**. Go on the profil tab and assign the **Algorithm** profil.

Edit the properties and add the function arguments on clicking on ✚ button *(See the Annex to know the parameters)*.

Create the external function referenced in the *Annex*.

An interaction The interaction is a simple function but it only called on the activation component's states. See *Machine State* to known how modelling it.

A processing

The processing is a pool of functions and it will be executed sequentially. It authorized adding action code in the modèle. A processing is called by an interaction. To add a processing select **libExternalFunction** component in the **model explorer**. Press right click and add a new **OpaqueBehavior**.

Edit the properties to change name, add an language definition, and link this processing to an interaction. To modeling processing, we choose the *Alf* language definition. This language is known by all plateform's generators and could be known by the custom generators (see *Alf* . *Generators API*). The action description code is write on the "body" property section. To link the processing with an interaction, clic on ✚ button in the **Specification** property section.

Create all the processing defined in the *Annex*.

Note: Ok, now we have define all datatypes and functions needed, we go to create the base components.

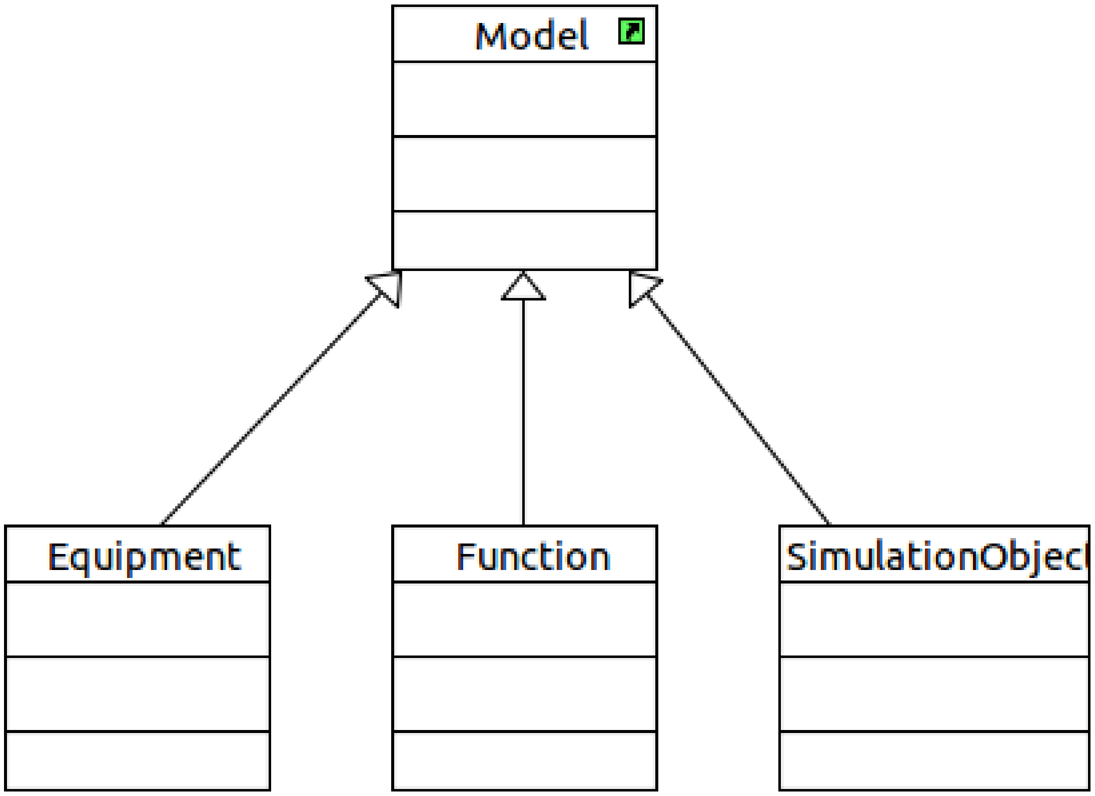
Adding the basic model objects As we see on upside, the basic model objects are define in the level 1 of the scenario model. So go to create the level 1 of the model, and named it ATK_N1.

Create the base components In the scenary we can found 4 base compenents:

- Automat : The element can be autonomous
- Function : The element is used as function (ex : management)
- Equipment : The element is used as equipement (ex : jammer)
- Simulation object : The element is a simulation publicobject

Show the **RobotML** perspective, and select the **BasicModelObject** node in the **Model explorer**. Do right clic and choose **new child/Add new package..** Names the new package as **Component base**, and do right clic on it. Choose **add new diagram/Add a class diagram**, named it **baseComponent_classDiagram**.

In the **RobotML tool panel** select the **Class component** and drag it on the diagram to create to **Automat** component. Edit the component properties and assign the **System** profile to **automat component** The automat has some action to do, go to *Adding a state machine* add some dynamic at the component; Create the other base component, as the following picture. Use **Generalization tool** on **RobotML tool panel** to assign the inheritance.



Add a state machine to a component The easiest method to add a state machine to a component is create a **State Machine diagram**. So select our **Automat** component and do right clic. Choose add a new **State machine diagram**. In the scenario, our automat has 3 states:

- Init, this state is only on the simulation startup.
- Running, only the component is initilazed and simulation is running.
- Killed, only in the simulation shutdown (never used).

With the **RobotML tool panel**:

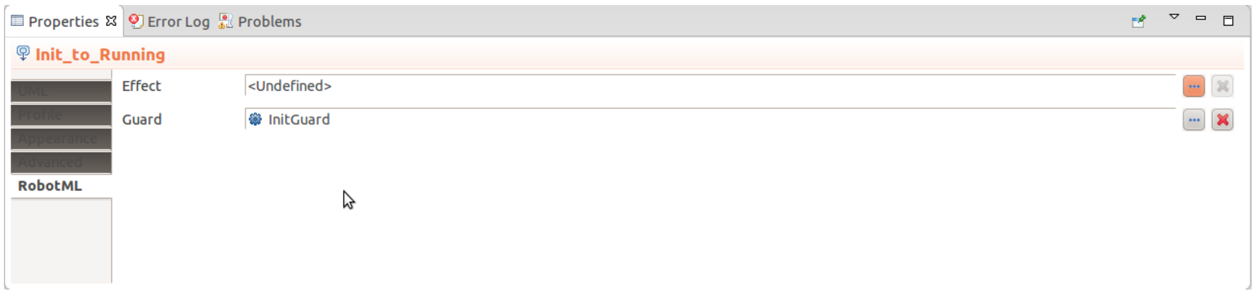
- Select the **Initial state** element and drag it on the diagram. Edit their properties and named it **INIT**.
- Select **State** element and drag it on the diagram. Edit their properties and named it **RUNNING**.
- Select the **Final state** element and drag it on the diagram. Edit their properties and named it **KILLED**.
- Select the **Transition** element and draw the transition between “INIT / RUNNING” and “RUNNING / KILLED” states.

Add transtion’s guard and effect Normally we have already added the usally function in the model. If it is not do, then

See also:

Create our external librairy’s functions

In the **State machine diagram** created, select the transition between this INIT and RUNNING states. Edit the properties, and show the **RobotML** tab. Assign the **InitGuard** operation to the transition guard. this operation define our condition to autorized the running state activation.



Effect <Undefined>

Guard InitGuard

Note: If you should to add an effect to a transition, then do the same action on the effect property.

Warning: RobotML not permit to a state machine to be inherited. The state machine must be clone in the child component. You can use the copy/paste function to do it.

Create the basics components Show the **RobotML** perspective, and select the **BasicModelObject** node in the **Model explorer**. Do right clic and choose **new child/Add new package..** Names the new package as **basic components**, and do right clic on it. Choose **add new diagram/Add a component diagram**, named it **basicComponents_componentDiagram**.

In the **RobotML tool panel** select the **Component** element, and drag it on the diagram. Named this component as **Esm**

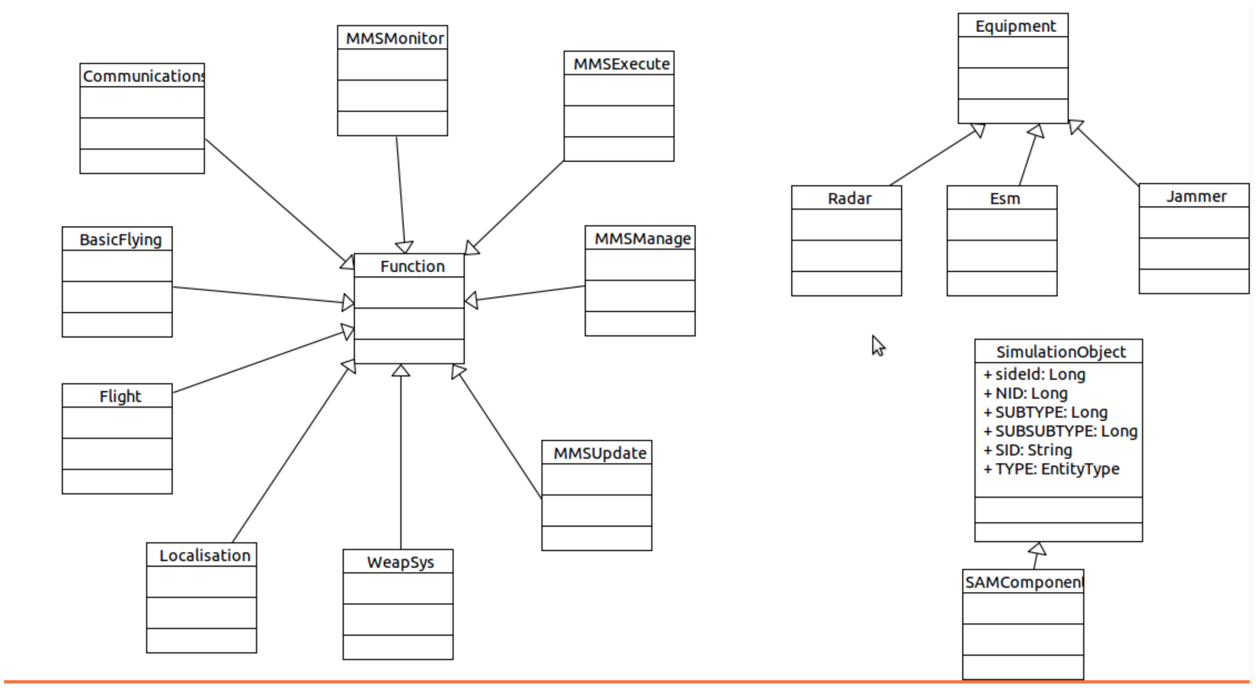
Adding port To add a port to the component, select the **Dataflow port** element in the **RobotML tool panel**, and drag it on the component. The new added port is symbolized by a black arrow. Named this port as **status_in** and edit their property. In the property view, choose **StatusList** for the port type. By default the dataflow port direction is **in**. In the **RobotML** tab, we can change it. The available values are:

•	In	Input port
•	Out	Output port
•	In / Out	Input and ouput port

Adding property To add a property to the component, select the **property** element in **RobotML tool panel**, and drag it on the component. The new property is symbilzed by a rectangle. Named this property as **esmDef**, and edit their property. In the property view,choose **esmDefinitions** for the property type.

Inheritance Now we had a **Esm** component but in the real life, this component is considerate as an equipement. We had also define the **Equipment** component base, we don't go to rewrite it! Then we go to use the inheritance method. We go to define **Esm** as child of the **Equipment** component base. So select the **BasicModelObject** node, and do right clic to chosse **add new diagram/add a class diagram**. Named the new diagram **BasicModelObject_Classdiagram**. In the **Model explorer** select the **Equipment** component base drag it on the diagram. Do the same action with the **Esm** component. In the **RobotMl tool panel** select the **Generalization** tool and link the **Esm** component to **Equipment**.

Now create all basics components referenced in the *Annex*, and assign the inheritance as the following class diagram.



Create the systems components Now we go to define the systems components present in our scenario:

•	MMS	Mission system management
•	Protection	Protection system management
•	Sensors	Sensors system management
•	Navigation	Navigation system management

So Create a new RobotMl model in the project view, and named it as **SystemObjectModel**, add a new **SystemComponents** package. Add a new **Component diagram** to the **SystemComponents** package, and create the fours system with the **Component** element of the **RobotML tool panel**.

Edit the properties's component and assign ports, inherit and property as it defined in the *Annex*.

Create the robots components In the scenario, we have two robots components entities:

•	SASystem	Ground / Air robot
•	Aircraft	Aircraft robot

Create the new **RobotObjectModel** RobotML model, and add the **RobotComponents** package. Add a new **Component diagram** to the **RobotMLComponents** package, and create the fours system with the **Component** element of the **RobotML tool panel**.

Edit the properties's component and assign ports, inherit and property as it defined in the *Annex*.

Create the scenary component The scenary component is the root element of the simulation. It define the simulation's envrionment, and contains the instances simulation.

Define the new **ScenaryObjectModel** RobotML model, and add the **ScenaryComponents** package. Create a **Component diagram** and create the **Root** component with the **Component** element of the **RobotML tool panel**.

Edit their properties and as assign ports, inherit and property as it define in the *Annex*.

Link the components Now all our components are created, but they need to communicate to have some dynamic. So with the **Link** element of the **RobotML tool panel**, go the draw the component communications in the different model level.

Static validation model When we finish the scenary modelling operation, we should to validate our model statically. The validation can to be run on all the model our just a sub tree section. To validate, select a node and do right clic. Choose **Validation / Validate model** or **Validation / Validate subtree**. When the validation been running, the error status is display on the **Model explorer**. Show the **Error log** view to known the model errors or warning.

Note: A model with not error and warning is strongest, and reduce the generation errors.

The deployment The deployment is use to generate the sources code files to destination of a middleware or a simulator. For our scenary, our destination is the *VLE* simulator.

Create a new deployment To create a new deployment, open the model level 4, and select the root level node. Do right clic, and select **add new diagram / new deployment diagram**.

Your First Scenario - attack an airport - Deployment

TBD

Your First Scenario - attack an airport - Problem ready

TBD

Athena generation specificities

Event definition on modelling

```
prototype test begin
   event initializing { time == 0 }

   stateset ModelState {INIT, RUNNING} = INIT begin
      transition from INIT to RUNNING on initializing
   end
end
```

To define an Athena event, you should to create an OpaqueBehavior "oInitializing". Select Athena as source code, and put time == 0 as body value. Create an Operation and name as Initializing, and apply the Algotrythm stereotype. Assign this opération as oInitializing behavior's sepcification.

To use this event, select a transition in your state machine, and go to the RobotML tab, in the property view. Assign the Initializing operation as transition's guard or effect.

Signal definition on modelling

```
prototype test begin
   signal sigEvent

   stateset ModelState {INIT, RUNNING) = INIT begin
      transition from INIT to RUNNING on initializing raise sigEvent
   end
end
```

To define an Athena signal, you should to create an Operation sigEvent, and apply the Algorythm stereotype. To use this signal, select a transition on your state machine, and assign the sigEveng operation as transition effect in the propertie's RobotML tab.

State action on modelling

```
prototype test begin
   when (ModelState::INIT) begin
      interaction init()
   end
end
```

To define a state action, you should create an OpaqueBehavioir "oInit". In the RotbotML tab of the proties view, selct Alf as source code, and complete the body's value on Alf language (you use Athena). Create an Operation and name as init. Apply the Alogorythm stereotype on this operation. In the RobotML tab of the OpaqueBahavior's properties view, assign the operation as specification of this behavior. To use this action, select youy state, and go to the RobotML tab of the properties view. Assign your operation and link the arguments.

Annex

Note: All code in this are writen on Alf.

List of enumeration for the scenario ATK

```
enum NetworkID {
   INVALID_NETWORK(0),
   IFDL(20001),
   LBWDL(20002),
   HBWDL(20003),
    TCAS(20004)
}

enum SensorType {
   INVALID_SENSOR_TYPE(0),
   MAW(1),
   ESM(2),
   RDR(3),
   SAR(4),
   EOIR(5)
}

enum SensorFunction {
   INVALID_SENSOR_FUNCTION(0),
   IMAGE(1),
   SEQUENCE_OF_IMAGE(2),
   SEARCH_AND_TRACK(3)
}

enum EquipementType {
   PLATFORM(0),
   EQP_COMMS(1)
}
```

List of composed datatypes for the scenario ATK

```
datatype SensorDefinition {
   public type : SensorType;
   public workWithEntities : Logical
   public workWithEntitiesTypes : longs
   public technos : SensorTechnos
   public useLineOfSight : logical
   public fieldOfRegard : SensorAngularBounds
   public fieldofView : SensorAngularDelta
   public intervisitype : Long
   public resolution : Vector3
   public toBeAcquireDuration : Long
   public tobeLostDuration : Long
   public canBeJammed : Logical
   public historyLenght : Long
   public isEmitter : Logica
   public id : Long
}
```

List of the external librairy's functions

```
public <<Algorithm>> function initRadarStep(in time : Long, in timestep : Long, in NID : long)
```

List of the processing

List of the components

```
public class Aircraft : Model
{
   public <<dataflowport>> port in time : Long
   private property SID : Long
}
```

Presentation of this example of use

The goal of this example is to guide you from scratch to the complete creation of a problem provided to the Robotic community through the portal. Each step will be explained in detail and you will be able mimeting it to build your own *problems*.

3.1.3 How-to : CycabTK Simulator

Amaury Nègre - e-Motion - CNRS / Inria

anr-proteus.fr

Objectives: First step with CycabTK.

CycabTK

CycabTK is a robotic toolkit developed at Inria Grenoble. It mainly offers a 3D simulator integrating physics engine for mobile robots. It has been designed to be interfaced with ROS. Prior version of the toolkit also offers a blackboard-based middleware, named Hugr, automatic mobile robot drivers and sensor drivers (sensors as Sick laser, GPS, motion tracker, mono or stereo camera).

Installation

CycabTK Simulator is a ROS package that provide a set of mg-Engine plugins to simulate robots and sensors.

Install from the proteus repository on Ubuntu 12.04 i386 :

If you use Ubuntu 12.04 i386, you just have to install the **CycabTKSimulator** package from the proteus repository (https://packages.greyc.fr/proteus) :

```
$ apt-get install CycabTKSimulator
```

Install from sources :

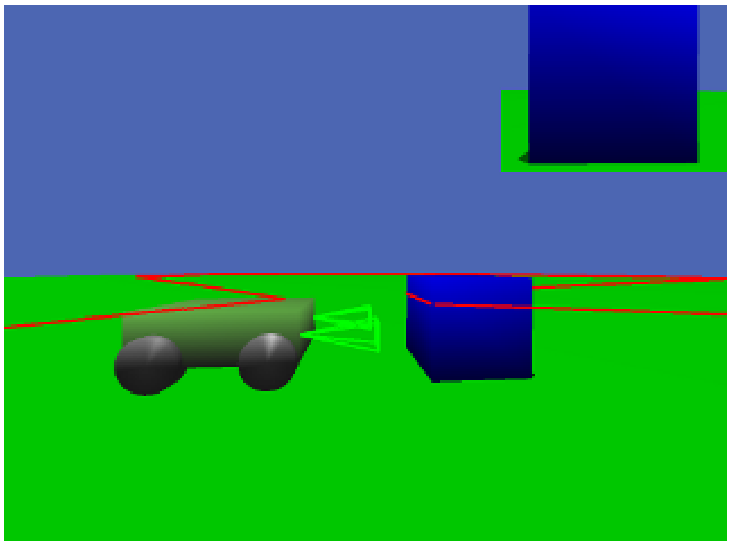
To install the simulator from the sources, read instructions on the CycabTK main site

Test installation :

To verify the installation, you can launch the tutorial launchfile :

```
$ roslaunch CycabTKSimulator cycabtk_tutorial.launch
```

After that, you should see a window like this:



You can move the robot by pressing the keyboard arrows.

You can check you have sensors / commands topics in ROS :

```
$ rostopic list
/clock
/robot/camera/camera_info
/robot/camera/image_raw
/robot/camera/image_raw/compressed
/robot/camera/image_raw/compressed/parameter_descriptions
/robot/camera/image_raw/compressed/parameter_updates
/robot/camera/image_raw/compressedDepth
/robot/camera/image_raw/compressedDepth/parameter_descriptions
/robot/camera/image_raw/compressedDepth/parameter_updates
/robot/camera/image_raw/theora
/robot/camera/image_raw/theora/parameter_descriptions
/robot/camera/image_raw/theora/parameter_updates
/robot/ground_vehicle_odometry
/robot/hokuyo/scan
/robot/ir1/range
/robot/ir2/range
/robot/twistCommand
/rosout
/rosout_agg
/tf
```

Create a new simulation

Instructions to create a new simulation are available on the CycabTK web site : http://cycabtk.gforge.inria.fr/wiki/doku.php?id=tuto:tutorial

3.2 Platform Installation

Contents:

3.2.1 Install *RobotML* Tooling

In order to ease the *user* work a specific customisation of the RobotML platform grouping its different components into a unique application has been done. The *ECLIPSE RCP* mechanism has been used to group:

1. *RobotML* editor
2. Generators towards different opensource tools,namely:
   (a) *Cycab-tk*:
   (b) *MORSE*:
   (c) *OROCOS*: it is a robotic middleware developed in Europe nowadays.
   (d) *ROS*:
   (e) *URBI*:
3. Generators towards different company-licensed robotic middleware, namely:
   (a) *ARROCAM*:
   (b) *RT-MAPS*:
4. Components needed for the platform to run correctly, namely:
   (a) *ATHENA*:
   (b) *SVN*:
   (c) *VLE*:

If the platform is able to provide generated code source, there is a need to also install the different associated frameworks. It is the goal of the *RobotML-sdk* (see *Install RobotML-sdk* to know how to install it).

For *users* willing to know more, it is possible to see how to create the platform in their classical *eclipse* using specific update sites one after the other following *RobotML platform installation from update site*.

3.2.2 Install RobotML-sdk

This presentation aims at helping you to install *RobotML-sdk*.

More information is available in different locations:

1. As this sdk was developed thanks to the *PROTEUS* project, information can be found on its website, anr-proteus.fr;
2. The Formal robotic portal is the place where all accessible information, data, tools, etc can be found;
3. This very document includes a *detailed description* of the different tools installed and how to test their correct installation using a specific installer.

Introduction

RobotML-sdk is a set of tools developed or selected within the PROTEUS project when specifying it. As already explained and shown again on the picture, in order to provide the whole power of RobotML, there is a need to provide the different possible targets on which to generate actual behaviour for the robot either in *simulators* or in actual robots. It is available as a set of Debian packages for the selected project's operating system Ubuntu Linux distribution.

Those tools are up-to-date on PROTEUS APT repository hosted by univ-orleans.fr.

Warning: The repository's access is secured, as some software are not free, every member has credentials to install and update his distribution.

RobotML-sdk constitution

Here follows the list of tools included in the sdk.

- Main tools
  - ROS Robot Operating System
  - Eclipse/Papyrus
  - Protege
- Simulation plateform
  - CycabTK
  - MORSE
  - VLE
- Robotic framework
  - Arrocam
  - RTMaps
  - Urbi
  - Orocos

Ubuntu installation

In order to harmonise software integration, Ubuntu Linux distribution was chosen.

- The latest LTS, the 12.04 (32 bits, x86 architecture), is the reference version. LTS stands for Long Term Support (5 years), meaning for the 12.04, until 2017!
- Download Ubuntu 12.04 (Precise Pangolin) (image CD ISO ~700 MB)
- Follow instruction provided by Ubuntu to install image on your own computer

Note: allow at least 10 GB for your installation (15 should be enough)

Menu installation

Authoring

Section author: Pierrick Koch <pierrick.koch@laas.fr>

3.3 Generation

Contents:

3.3.1 How to generate

Contents:

There is two kind of generator, middlewares and environment ... TODO à developper

TODO: Expliquer les plan de deploiements, la possibilité de générer une partie sur middleware une autre sur environement ...

3.3.2 Targets

Contents:

Middlewares

Contents:

OROCOS

Contents:

TBD

RTMaps

Contents:

TBD

Effibox

TODO mise en forme

Generation RobotML to Effibox

Effibox generated application

The generator creates a complete Effibox application containing: - subscriptions to the sensors used in the modeling - all model classes assigned to the middleware - standard datatypes and custom datatypes Below is an example of a generated application tree:

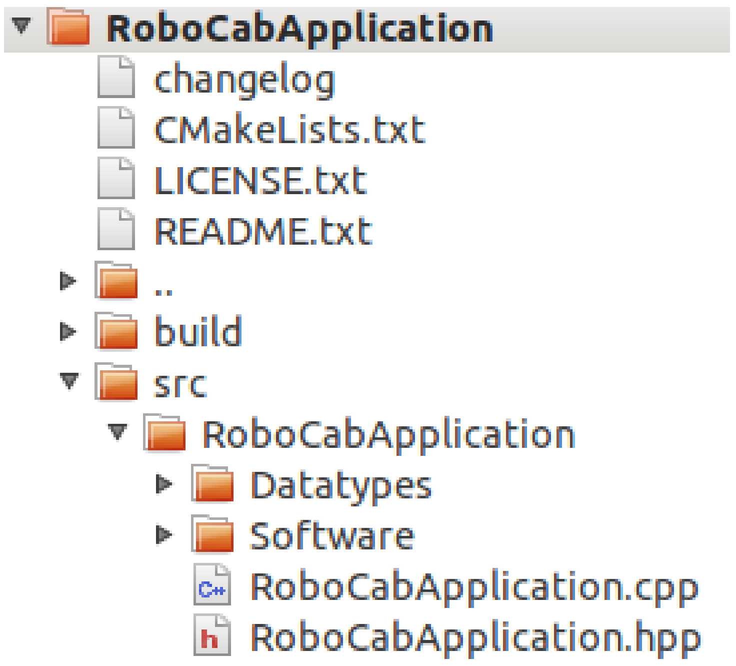
In the "src" directory are placed all the c + + source files: -. RoboCabApplication * pp is the main class - Data structures (standard and custom) are stored in a subdirectory Datatypes - The components of the model are stored in a subdirectory Software

Correspondence between the model and the generated code: Components: Component "RobotML" is translating as a C++ class. The elements of the component are declared as class attributes: - Instance of each sub-component - Input ports - Outputs ports - Internal and external connections between components - State machines and transitions

Ports:

Output Ports:

Each output port is translated by an Effibox communication object named TaskBuffer. In this object is assigned to one or more processing functions. Each time data is added to TaskBuffer (push method), all associated processing functions are executed in parallel.



Input port : An input port is translated as a method of the class corresponding to the component. This method can be associated with an output port as a processing function. Then, it's automatically called when an event is transmitted by the output port connected.

Connections: Internal: Within a macro-component, several sub-components may have their input/output ports / interconnected. The connection is simply an association with an input port and an output port, thanks to addFunctionToExecute method in TaskBuffer class (see output port).

Example : //inner components connexions pathPlanner.pathDef->addFunctionToExecute(boost::bind(&LocalisationModule::pathDefIn, &localisation, _1));

External: In a macro-component, an external connection is a connection between an input of an internal component and an input of the macro-component, or between an output of an internal component and an output of the macro-component. - An external connection "input-input" is translated by encoding the input port of the macro-component as a method that calls the method corresponding to the input port of the subcomponent. - An external connection "output-output" is translated by encoding the output port of the macro-component as a pointer to the input port of the internal component.

State Machines : State names are translate into a structure enum. Transitions are implemented as methods. If a method has already a c++ content in modeling, this content is copied.

Limits : - Only the contents of c++ transition functions are copied. If the content is written in another language, it will be ignored. - The change of state is currently not implemented correctly.

Datatypes : - RobotML datatypes: the RobotML datatypes are converted into C++ structures using standard C ++ language. Thus there may be differences between the datatypes and datatypes modeling after translation. - User datatypes: the user-defined datatypes are built the same way, based on the RobotML translate datatypes.

External algorithms: Not supported by the generator for the moment.

Port Service Not supported by the generator for the moment.

Deployment plan : - All components containing the following stereotypes are generated as a sensor subscription:

CameraSystem, GPSSystem, InertialMeasurementUnitSystem, InertialNavigationSystem, OdometrySystem, Robot (see next paragraph). - All components allocated on the platform are generated as an Effibox component, whatever their stereotype.

Sensor connections: The sensor connections are translated as Effibox subscriptions. They are stored in the xml file describing the Effibox application .awp file. Processing functions associated with the sensors are encoded in the main application class. In each function, Effibox types are converted to RoboML type. Here dataflows are connected to inputs of instantiated components.

To make the sensor data connections to your RobotML components works properly, you must respect some contraints into the model: - A RobotML IMU sensor must transmit the datatype Imu in an output dataflow. - A RobotML Gps sensor must transmit the datatype NavSatFix in an output dataflow. - A RobotML Camera sensor must transmit the datatype Image in an output dataflow. - A RobotML Lidar sensor must transmit the datatype LidarScan in an output dataflow. - A RobotML Odometry sensor must transmit the datatype CarLikeOdometry (Vipalab for example) or DifferentialOdometry (wifibot for example) in an output dataflow.

Environments

Contents:

CycabTK

Contents:

TBD

Morse

Contents:

TBD

CHAPTER

FOUR

ROBOTML META-MODEL

TBD

Meta-modeling means to work not on the domain but on the properties of this domain in order to extract generic features that could be used on many problems occurring in this domain. In this scope, it means to define the salient features of the robotic domain in order to create some specific language able to describe it easily.

CHAPTER

FIVE

ROBOTML MODELING PLATFORM

5.1 About Papyrus

Papyrus is aiming at providing an integrated and user-consumable environment for editing any kind of *EMF model* and particularly supporting *UML* and related modeling languages such as *SysML* and *MARTE*. Papyrus provides diagram editors for EMF-based modeling languages amongst them *UML* 2 and *SysML* and the glue required for integrating these editors (GMF-based or not) with other *MBD* and MDSD tools.

Papyrus provides a very advanced support for *UML* profiles enabling support for "pure" *DSL*. Every part of *Papyrus* may be customized: model explorer, diagram editors, property editors, etc.

See also:

Papyrus website

5.2 Papyrus tutorials

Go to Papyrus tutorials page.

5.3 DSL detailed

One of the objectives of the *PROTEUS* project is to provide domain specific languages (and related tools like editors, consistency checkers, etc) suitable to specify missions, environments and robot behaviours that have been specified by robotics experts involved in the project. The discussions within the *PROTEUS* project have led to the decision of defining three *DSL* s.

1 The *Architecture DSL* which will ease the definition of specific robotic architectures (reactive, deliberative, hybrid) and specific components that form those architectures (sensors, actuators, planners).

2 The "Control & Communication DSL" that will control the robotic components and will ease the definition of communication mechanisms between components (sending/receiving of events and data).

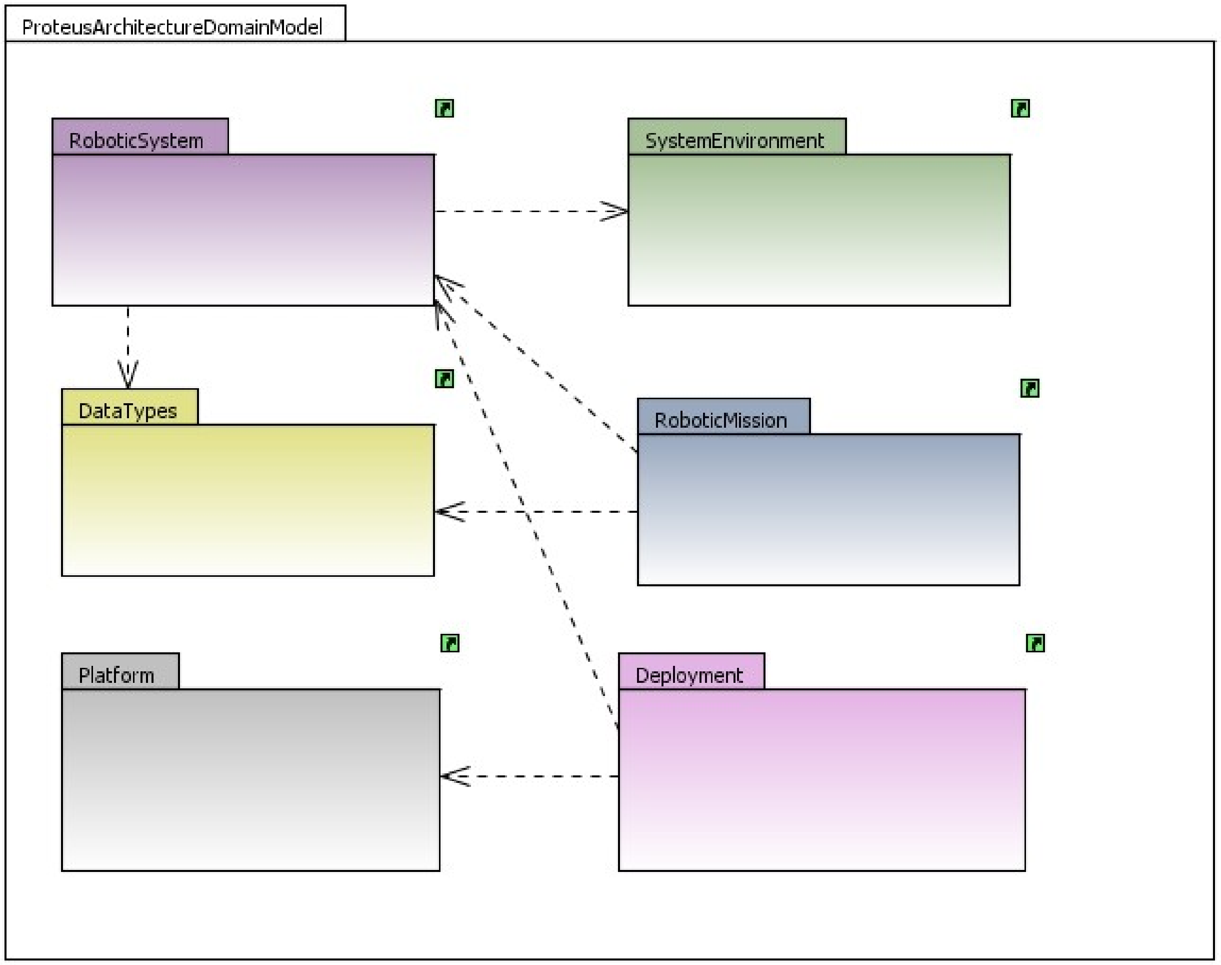
3 The "Algorithms DSL" that will ease the definition of algorithms which are to be used, triggered with the "Control & Communication DSL" for implementing behaviours in the different components of an architecture described with the *Architecture DSL*.

5.4 Domain model

The domain model of a given domain specific language defines the concepts and the terminology of the language domain. The relationships between the concepts are described as well. In this section, we present the domain model of the *RobotML DSL* s based on the requirements defined in the previous section and on the knowledge base provided by the ontology. We have used *UML* class diagrams to represent the domain model. Figure 1 show a generic view of the RobotML domain model which describe the architecture of the robotic system (ie. its internal structure), the behavior of components (through FSMs or algorithms) , the communications between them and the deployment of robotic systems to specific robotic platforms.

5.4.1 Architecture DSL

This section introduces a high-level "executive" summary of the subject matter of the relevant sub- package. It contains an informal and lightweight description that identifies the subject matter itself (key concepts and relationships), the general principles of operation (semantics), design rationale, relationship to other packages and their subject matter, etc. The idea is that, for a cursory reader, this should provide sufficient information of the subject without delving into technical details.



The Proteus architecture domain model package contains six sub-packages:

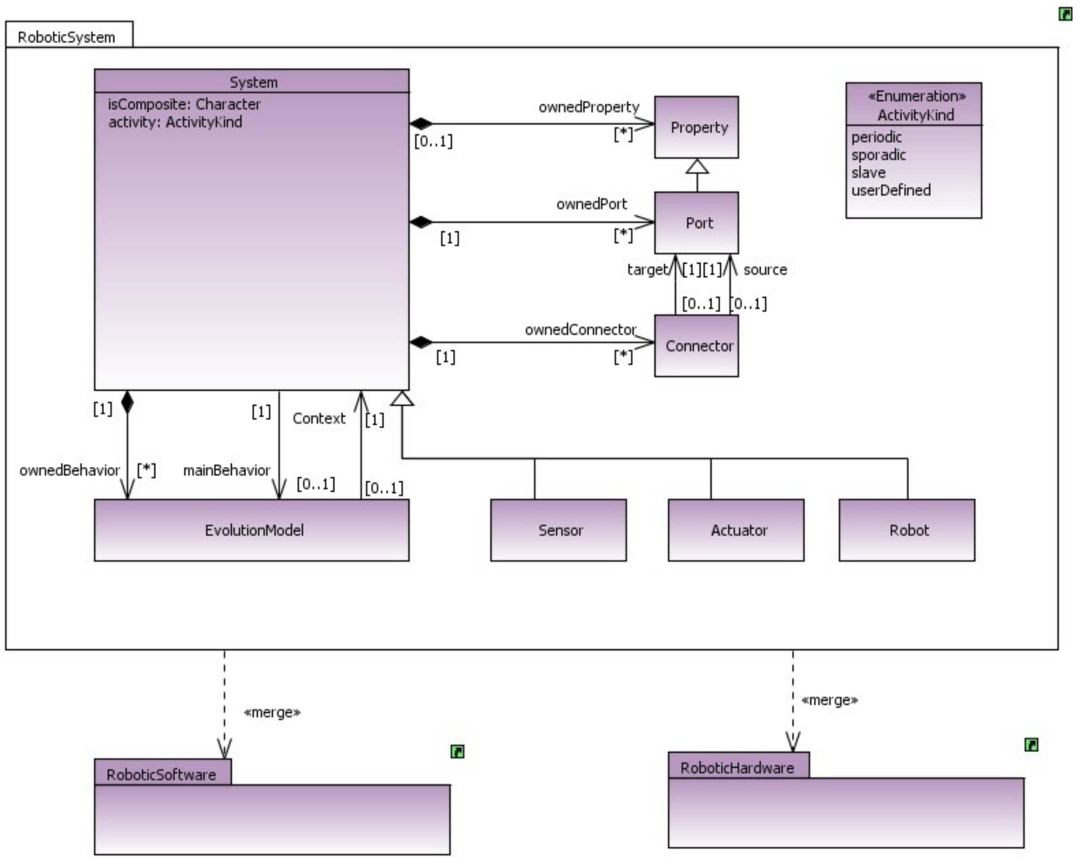
- The *robotic system package*: describes the concepts that help defining and composing a robotic system,

- The system environment package: since we not only model the robotic system but also its environment (for example for simulation purpose), this package defines the concepts composing the "real" environment where robots evolve,
- The data types package: defines the data items that will necessarily be exchanged between robotic components, between algorithms, etc,
- The robotic mission package: describes the concepts that are needed to define an operational mission and which are used by the components of an architecture performing it, The deployment package: specifies a set of constructs that can be used to define the execution architecture of robotic systems,
- The platform package: defines the concept of platform which represents a software,
- Execution environment that can be either a robotic simulator or a robotic middleware.

Domain model detailed

Robotic System Package

A robotic **System** is composed of **Systems** that communicate with other **Systems** through **Ports** and have specific **Evolution Models** (i.e. behaviours). **Systems** could be **Software** or **Hardware**. **Software** systems are detailed in the **RoboticSoftware** package (Figure 6), **Hardware Systems** are detailed in the RobThe environment is the highest possible "container" for the physical objects to exist, and it is considered as part of the architecture of a robotic system. Figure 8 shows the robotic environment package which enables the specification of different types of environments. oticHardware package (Figure 7).



The System concept corresponds to the component concept. The term system is more appropriate to describe a robotic component, this is due to the fact that the term system is more meaningful for a robotician than the term component. A System is composed of properties, ports and connectors.

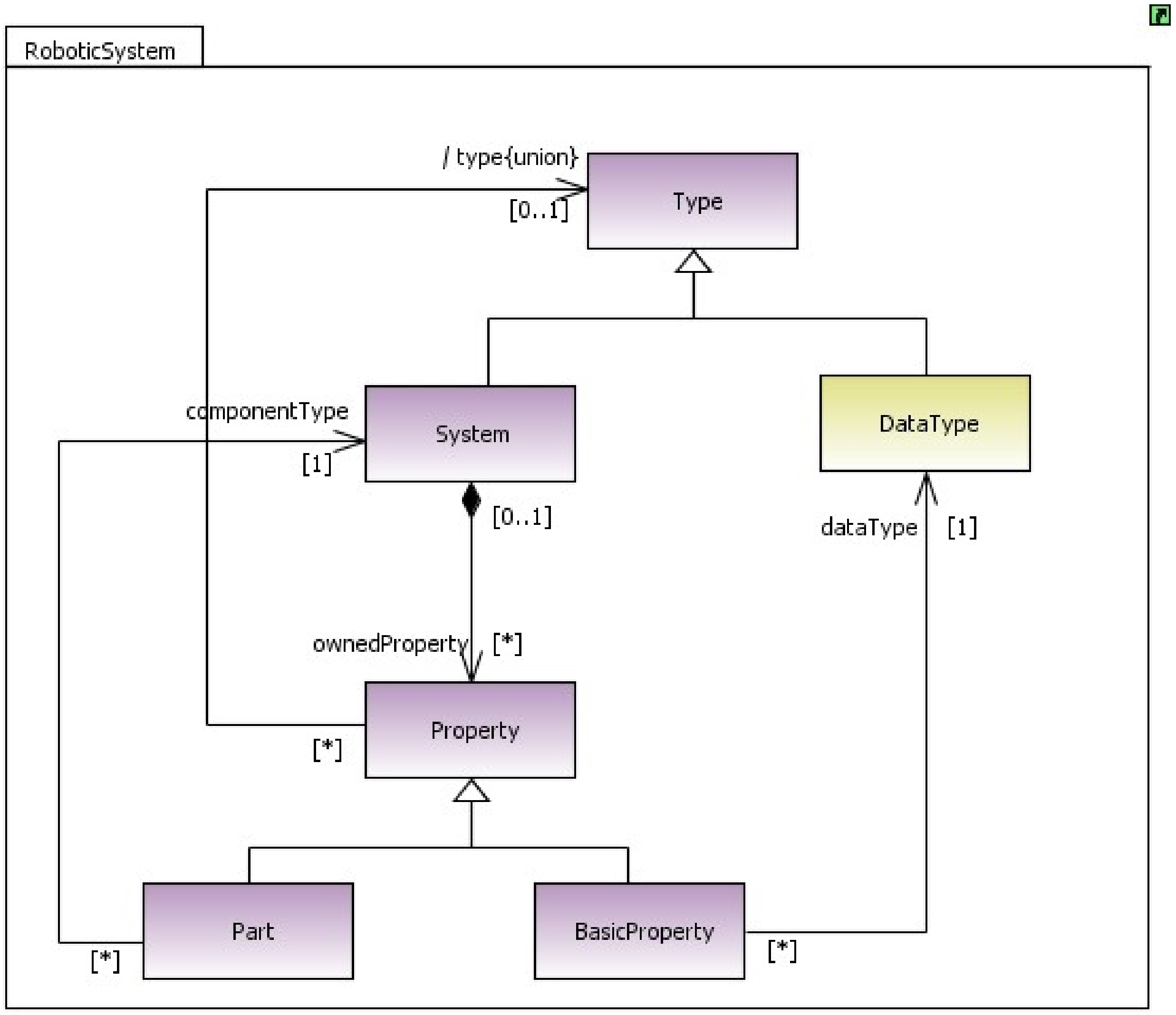


Figure 4 shows the details of the concept of **property**. A system is composed of **properties** which can be a subsystem or a variable. The **Part** property is typed by a **system** and thus corresponds to a subsystem. The **BasicProperty** is typed by a **data type** and thus corresponds to a variable.

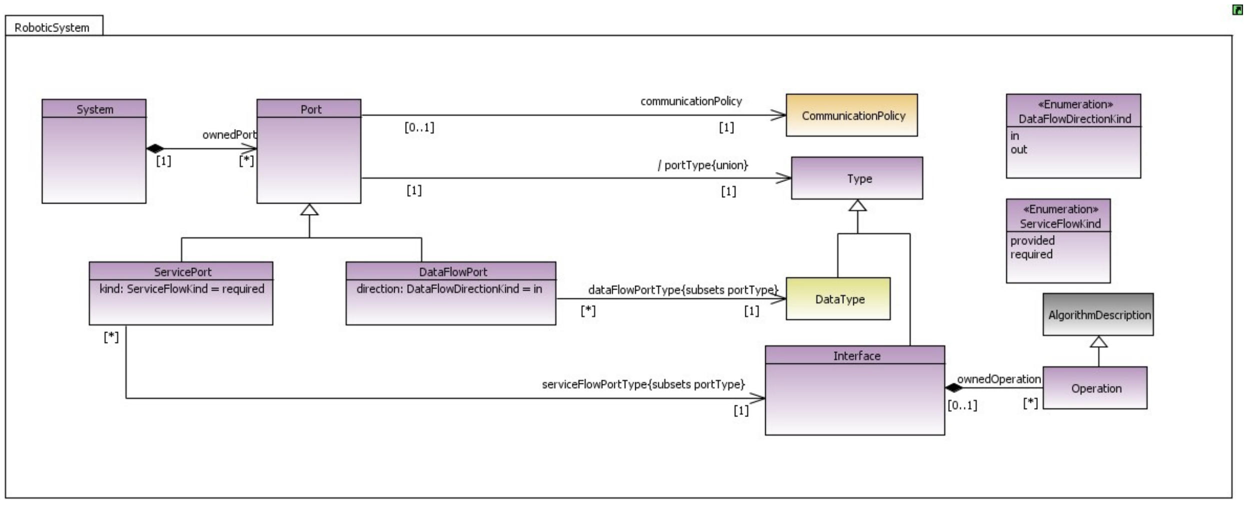
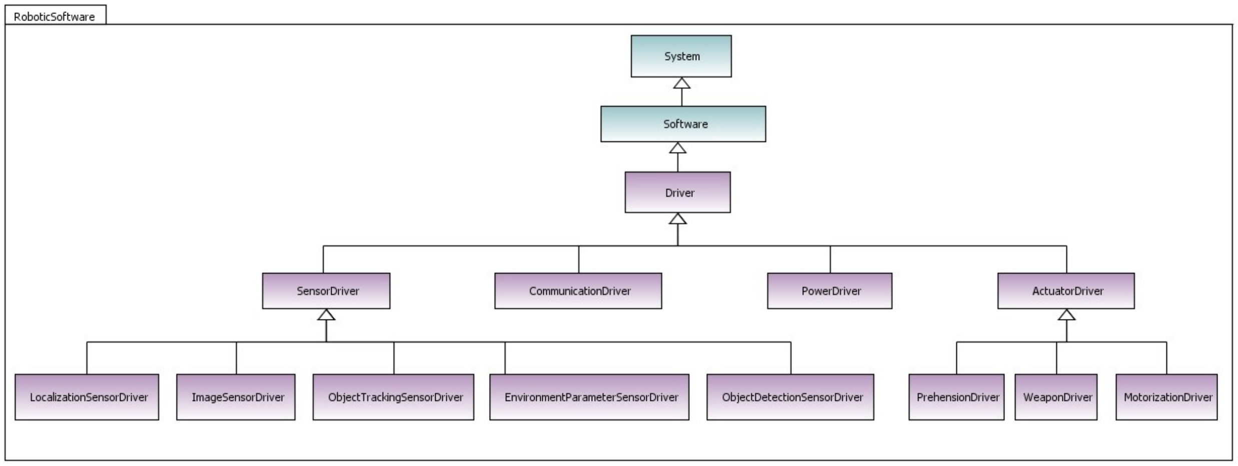


Figure 5 Shows the details of the concept of **port**. A **port** formalizes an interaction point of a **system**. It is a special kind of **properties** that has a type and a **communication policy**. A **ServicePort** is a **port** that is specific to client/server communications. **DataFlowPorts** enable data flow-oriented communication between systems, where messages that flow across ports represent data items. **ServicePorts** support a request/reply communication paradigm (also called client/server model of communication), where messages that flow across ports represent **operation** calls. **ServicePorts** are typed by an interface which specifies a number of operations without stating how exactly they are implemented.

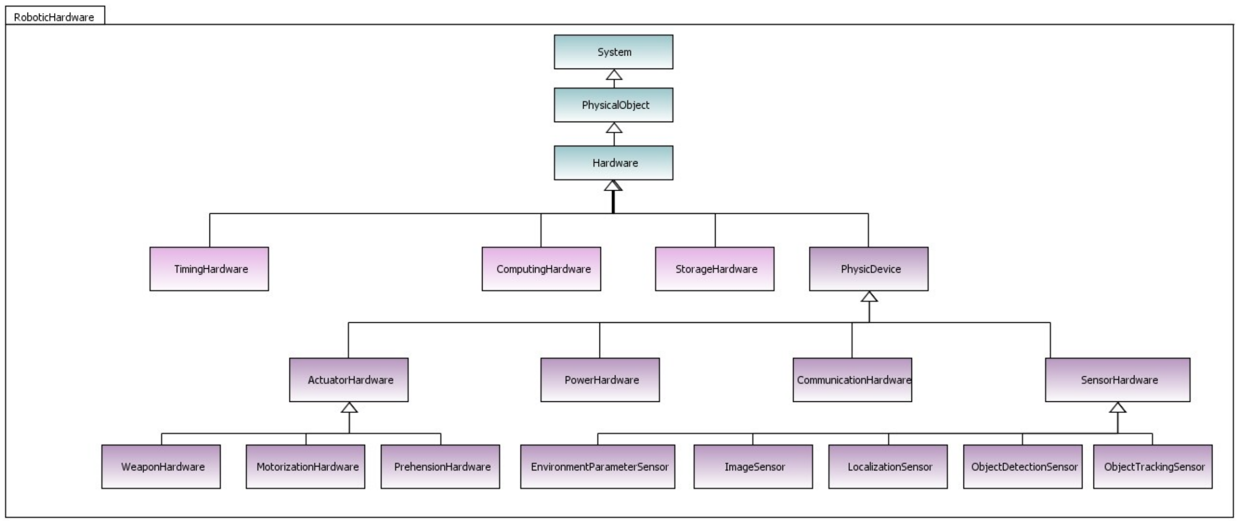
Detail of the robotic software package:

Figure 6 shows the details of the robotic software package. A software could be a driver which is the software system part of a DeviceSystem.



Details of the robotic hardware package:

Figure 7 shows the details of the **robotic hardware** package. A **hardware** could be: • a **physic device**: represents the hardware part of a device system, • a **storage hardware**: represents different forms of memory, • a **computing hardware**: represents physical processing devices capable of storing and executing program code, • a **timing hardware**: represents a hardware entity that is capable of following and evidencing the pace of time.



Robotic environment package

The **environment** is the highest possible "container" for the **physical** objects to exist, and it is considered as part of the architecture of a robotic system.

See RobotML API documentation.

5.5 Installation

You need to download the Eclipse Modelling tools. Then extract the modeller on your computer and launch it. You need to install *Papyrus*, with *RobotML* extra feature, *Acceleo*, and *Subclipse*.

- To install *Acceleo*, clic on **Help/install Modellign component**, and selection *Acceleo*.
- To install *Papyrus* and *Subclipse*, clic on **Help/Install new software...**, and add the following update site

5.6 Juno version

Plugin name	update site
Papyrus	http://download.eclipse.org/modeling/mdt/papyrus/updates/releases/juno
Subclipse	http://subclipse.tigris.org/update_1.10.x

Note: The juno version not support the dynamic validation.

5.7 Kepler version

Plugin name	update site
Papyrus (nigthly)	http://download.eclipse.org/modeling/mdt/papyrus/updates/nightly/kepler
Subclipse	http://subclipse.tigris.org/update_1.10.x

Up

CHAPTER

SIX

ROBOTML CODE GENERATORS

RobotML is using *Acceleo* as a code generator transforming models into code (*MDA* approach).

See also:

Acceleo website

Go to Acceleo tutorials page.

RobotML generators allowed to generate source code from the *RobotML* model to some middleware or simulator. They are include in the *RobotML* platform, and using *Acceleo* to browse the *RobotML* model.

List of the RobotML platform generators:

6.1 ARROCAM RobotML Generator

TBD

6.2 CycabTK RobotML Generator

TBD

6.3 MORSE RobotML Generator

6.3.1 MORSE Simulator

MORSE is a robotic simulation software developed by roboticists from several research laboratories (including GR-EYC). It is a framework to evaluate robotic algorithms and their integration in complex environments, modeled with the Blender 3D real-time engine which brings realistic rendering and physics simulation. The simulations can be specified at various levels of abstraction. This enables researchers to focus on their field of interest, that can range from processing low-level sensor data to the integration of a complete team of robots.

Building and configuration of scenarios are done with a set of Python classes provided by MORSE, known as the Builder API. The Builder API provides an internal domain-specific language (DSL) that completely hides the somewhat complex interface of Blender from the user, so that those unfamiliar with Blender can directly configure MORSE using Python scripts. Scripts can then be tracked on a version control system to follow changes and apply patches.

The API offers classes to add robots, sensors and actuators, to position, rename and configure any of the parameters used by these components, to include additional objects (furniture, obstacles, etc.) and to add middleware bindings, modifiers and services for each component.

6.3.2 MORSE Generator

The MORSE generator transforms RobotML models into a suitable Python script using the MORSE Builder API. The code is available here: https://github.com/RobotML/RobotML-SDK-Juno/tree/master/plugins/generators/MORSE The algorithm of the MORSE generator is basically a visitor on a RobotML model where RobotML stereotype that represent a sensors or actuators are mapped on a Python class. At the moment, only the AirOARP sensors are completely supported. This included the following prototypes: CameraSystem, GPSSystem, LIDARSystem.

6.4 OROCOS ROBOTML Generator

6.4.1 OROCOS middleware (Open Robot Control Software)

OROCOS is a real-time component-based framework, also called OROCOS-RTT (Real-Time Toolkit) for its real-time aspects. OROCOS components interact by exchanging data, events or services through ports. Data exchange is performed through input/output data flow ports. Services are implemented as Interface operations. OROCOS allows also the definition of components behaviors as state machines using the package rFSM.

6.4.2 OROCOS Code Generator

OROCOS code generator implements model to text transformations using Acceleo. Transformation rules link RobotML concepts (which gather the DSL architecture, the DSL control and the DSL communication) to OROCOS concepts. The code is available here https://github.com/RobotML/RobotML-SDK/tree/master/robotml-kepler/generators/OROCOS The OROCOS code generator takes input from a RobotML model and provides: * OROCOS components (hpp, cpp files). * Interfaces (hpp files) which define a set of abstract operations. * Data Structures (.hpp files) representing user defined data types. * a configuration file (OROCOS Program Script: OPS file) to configurate communication between components. * RTT-LUA components (lua files) which load the state machines describing the behavior of components * LUA state machines (lua files) implemented using the rFSM package where states, transitions, events, effects and guard are explicitly specified.

OROCOS generator aims at providing an executable application. To do so, a set of artifacts are also generated: - A makefile - A Cmakelists.txt where the components to be loaded are specified. - A manifest.xml file which specifies which libraries has to be imported.

6.5 RTMAPS RobotML Generator

TBD

6.6 VLE RobotML Generator

6.6.1 About

The Virtual Laboratory Environment *VLE* is a multi-modeling and simulation platform. It is a powerful modeler and simulator supporting the use of different formalisms for models specification and simulation. *VLE* is particularly well adapted for complex models where the coupling of different formalisms is required. In addition to the classical use of one single formalism for modeling and simulation, *VLE* can integrate, i.e. couple, heterogeneous formalisms in one coherent simulation model.

For instance, *VLE* supports, and is not limited to, the modeling and simulation of the following formalisms (stand alone or coupled together):

- Discrete Event Specifications
- Differential equations
- Difference equations
- Petri net
- Finite State Automata

VLE supports the following modeling and simulation paradigms:

- System Dynamics
- Multi-Agent Systems, Multi-Agent Simulation
- Decision support systems
- Learning systems

VLE is based on the theory of modeling and simulation initially developed by B.P. Zeigler in the 70's and continuously enriched until now by an active international community. *VLE* is based on the *DEVS* formalism (Discrete Event systems Specification). *VLE* provides a set of C++ libraries, the *VFL* and a lot of programs like a simulator, a graphical user interface to model and develop models and tools to analyze and visualize simulation outputs. The *VFL* are sufficiently well designed to allow the development of new simulators, models or new programs for modeling and analysis.

VLE's goals is to provide powerful tools for modeling, simulating and analysing complex dynamics systems. The development are complying with the :term'DEVS' specification DEVS and works made by the simulation community.

See also:

VLE official website

6.6.2 DEVS models

Name	Description
Passive	DEVS model which makes nothing, which reacts to no external event and which generates no ouput-event.
Counter	Very simple model which counts the number of events received on all the input-ports of the model. No output-event is produced.
Conditional	Model may require parameters that are expressed in condition form.
Generator	Generates an event every X time units (x can be a parameter).
Storage	Replicate the input to output with a lag. If an input is present before the replication to output, so this input is ignored
Binary Counter	Send "1" if two "1" is received within a period.
Sampling	Replicate with a frequency the last input.

See *RobotML VLE integration* to know the implementation specificities.

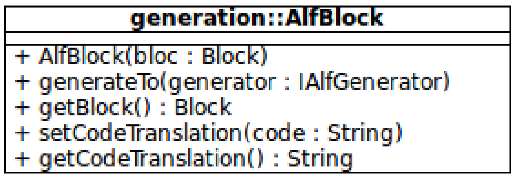
6.7 The Alf Generator API

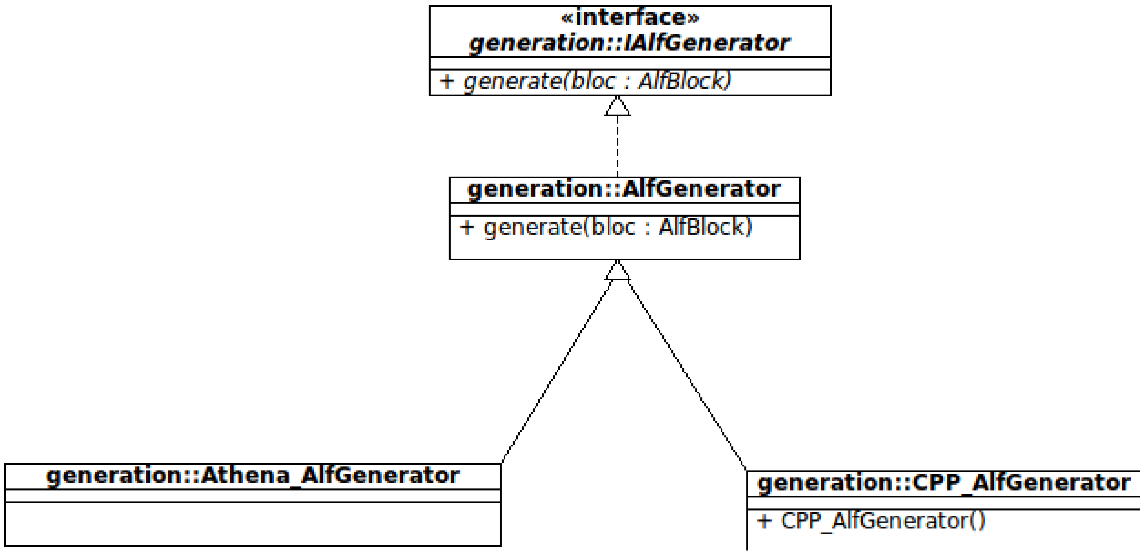
In RobotML, the *Alf* language is used to describe the operation contents (only OpaqueBehavior). This language permit to have a generic modeling and generate it easily in more other langauge.

For your needs, you can create a specific *Alf* generator language on using this *Alf Generator API*.

Warning: The *Alf* knowledge is clearly recommended. Go to see the Alf language site before start your work!

6.7.1 Overview


generation::AlfBlock
+ AlfBlock(bloc : Block)
+ generateTo(generator : IAlfGenerator)
+ getBlock() : Block
+ setCodeTranslation(code : String)
+ getCodeTranslation() : String


«interface»
generation::IAlfGenerator
+ generate(bloc : AlfBlock)
generation::AlfGenerator
+ generate(bloc : AlfBlock)
generation::Athena_AlfGenerator
generation::CPP_AlfGenerator
+ CPP_AlfGenerator()

AlfServices

The module **AlfServices** contains all the methods to translate RobotML element to functional code.

public

public static org.eclipse.papyrus.uml.alf.alf.Block createAlfBloc(org.eclipse.papyrus.uml2.NamedElement element)

element	UML element model.
return	Return an Alf Bloc corresponding to the UML element parameter.

Create an Alf bloc code from an Opaque behavior UML element.

public static String translateAlfBlocTo_Athena(org.eclipse.papyrus.uml.alf.alf.Block bloc)

bloc	Alf bloc to translate.
return	Return the alf bloc translation on Athena language.

Translate an Alf bloc to Athena code.

public static String translateAlfBlocTo_CPP(org.eclipse.papyrus.uml.alf.alf.Block bloc)

bloc	Alf bloc to translate.
return	Return the alf bloc translation on C++ language.

Translate an Alf bloc to C++ code.

AlfBlock

public

AlfBlock(org.eclipse.papyrus.uml.alf.alf.Block bloc)

bloc	Original Alf bloc.

Constructor, the alf bloc is mandatory, it will be transmited to the genrators.

void generateTo(IAlfGenerator generator)

generator	Translate generator

Call the Alf translation on using the generator parameter to work.

org.eclipse.papyrus.uml.alf.alf.Block getBlock()

return	Return the original Alf bloc

Return the Alf bloc to get the original information.

void setCodeTranslation(String code)

code	Code string to append on the result.

Append the code string parameter on the code string result.

String getCodeTranslation()

return	Return the code string result.

Return the code string translation result.

IAlfGenerator

Alf generator interface, contains the mandatory method to implement for the Alf generator.

public

abstract void generate(AlfBlock bloc)

bloc	Alf bloc to translate.

Abstract method to redefine in implementation. Translate the Alf bloc.

AlfGenerator

Abstract Alf base generator. Contains the common method for all Alf generators. It implement the IAlfGenerator interface.

public

abstract void generate(AlfBlock bloc)

bloc	Alf bloc to translate.

Translate the Alf bloc.

protected **String generateAlfBlock(org.eclipse.papyrus.uml.alf.alf.Block bloc)**

bloc	Alf bloc to translate.
return	Alf bloc translation.

Return the Alf bloc translation.

String generateStatementSequence(org.eclipse.papyrus.uml.alf.alf.StatementSequence aStatementSequence)

aStatementSequence	Alf statement sequence to translate.
return	Alf statement sequance translation.

Return the Alf statement sequence translation.

String generateDocumentedStatement(org.eclipse.papyrus.uml.alf.alf.DocumentedStatement aDocumented-Statement)

aDocumentedStatement	Alf documented statment to translate.
return	Alf documented statment translation.

Return the Alf documeneted statment translation.

String generateSequoclenceStatement(org.eclipse.papyrus.uml.alf.alf.Statement aStatement)

aStatement	Alf statment to translate.
return	Alf statment translation.

Return the Alf documeneted statment translation. **String generateState-ment(org.eclipse.papyrus.uml.alf.alf.Statement aStatement)**

aStatement	Alf statment to translate.
return	Alf statment translation.

Return the Alf documeneted statment translation. **String generateInlineState-ment(org.eclipse.papyrus.uml.alf.alf.InlineStatement aInlineStatement)**

aInLineStatement	Alf inline statment to translate.
return	Alf inline statment translation.

Return the Alf inline statment translation.

String generateAnnotatedStatement(org.eclipse.papyrus.uml.alf.alf.AnnotatedStatement aAnnotatedState-ment)*

aAnnotatedStatement	Alf annotated statment to translate.
return	Alf annotated statment translation.

Return the Alf annotated statment translation.

String generateBlockStatement(org.eclipse.papyrus.uml.alf.alf.BlockStatement aBlockStatement)

aBlockStatement	Alf block statment to translate.
return	Alf block statment translation.

Return the Alf block statment translation.

String generateEmptyStatement(org.eclipse.papyrus.uml.alf.alf.EmptyStatement aEmptyStatement)

aEmptyStatement	Alf empty statment to translate.
return	Alf empty statment translation.

Return the Alf empty statment translation. **String generateIfState-ment(org.eclipse.papyrus.uml.alf.alf.IfStatement aIfStatement)**

aIfStatement	Alf if statment to translate.
return	Alf if statment translation.

Return the Alf if statment translation. **String generateSequentialClausesTemplate(org.eclipse.papyrus.uml.alf.alf.SequentialClauses aSequentialClauses)**

aSequentialClauses	Alf sequential clauses to translate.
return	Alf sequential clauses translation.

Return the Alf sequential clauses translation. **String generateConcurrentClausesTemplate(org.eclipse.papyrus.uml.alf.alf.ConcurrentClauses aConcurrentClauses)**

aConcurrentClauses	Alf concurrent clauses to translate.
return	Alf concurrent clauses translation.

Return the Alf concurrent clauses translation. **String generateNonFinalClauseTemplate(org.eclipse.papyrus.uml.alf.alf.NonFinalClause aNonFinalClause)**

aNonFinalClause	Alf non final clause to translate.
return	Alf non final clause translation.

Return the Alf non final clause translation.

String generateFinalClauseTemplate(org.eclipse.papyrus.uml.alf.alf.FinalClause aFinalClause)

aFianalClause	Alf final clause to translate.
return	Alf final clause translation.

Return the Alf fianl clause translation. **String generateSwitchStatement(org.eclipse.papyrus.uml.alf.alf.SwitchStatement aSwitchStatement)**

aSwitchStatement	Alf switch statement to translate.
return	Alf switch statement translation.

Return the Alf switch statement translation. **String generateSwitchDefaultclause(org.eclipse.papyrus.uml.alf.alf.SwitchDefaultClause aSwitchDefaultClause)**

aSwitchDefaultClause	Alf switch default clause to translate.
return	Alf switch default clause translation.

Return the Alf switch default clause translation. **String generateSwitchClause(org.eclipse.papyrus.uml.alf.alf.SwitchClause aSwitchClause)**

aSwitchClause	Alf switch clause to translate.
return	Alf switch clause translation.

Return the Alf switch clause translation. **String generateSwitchCase(org.eclipse.papyrus.uml.alf.alf.SwitchCase aSwitchCase)**

aSwitchCase	Alf switch case to translate.
return	Alf switch case translation.

Return the Alf switch case translation.

String generateNonEmptyStatementSequence(org.eclipse.papyrus.uml.alf.alf.NonEmptyStatementSequence aNonEmptyStatementSequence)

anonEmptyStatementSequence	Alf non empty statement sequence to translate.
return	Alf non empty statement sequence translation.

Return the Alf non empty statement sequence translation. **String generateWhileStatement(org.eclipse.papyrus.uml.alf.alf.WhileStatement aWhileStatement)**

aWhileStatement	Alf while statement to translate.
return	Alf while statement translation.

Return the Alf while statement translation. **String generateDoStatement(org.eclipse.papyrus.uml.alf.alf.DoStatement aDoStatement)**

aDoStatement	Alf do statement to translate.
return	Alf do statement translation.

Return the Alf do statement translation.

String generateForStatement(org.eclipse.papyrus.uml.alf.alf.ForStatement aForStatement)

aForStatement	Alf for statement to translate.
return	Alf for statement translation.

Return the Alf for statement translation.

String generateBreakStatement(org.eclipse.papyrus.uml.alf.alf.BreakStatement aBreakStatement)

aBreakStatement	Alf break statement to translate.
return	Alf break statement translation.

Return the Alf break statement translation.

String generateReturnStatement(org.eclipse.papyrus.uml.alf.alf.ReturnStatement aReturnStatement)

aReturnStatement	Alf return statement to translate.
return	Alf return statement translation.

Return the Alf return statement translation.

String generateAcceptStatement(org.eclipse.papyrus.uml.alf.alf.AcceptStatement aAcceptStatement)

aAcceptStatement	Alf accept statement to translate.
return	Alf accepet statement translation.

Return the Alf accept statement translation.

String generateClassifyStatement(org.eclipse.papyrus.uml.alf.alf.ClassifyStatement aClassifyStatement)

aClassifyStatement	Alf classify statement to translate.
return	Alf classify statement translation.

Return the Alf classify statement translation. **String generateInvocationOrAssignementOrDeclarationStatement(org.eclipse.papyrus.uml.alf.alf.InvocationOrAssignementOrDeclarationStatement aInvocationOrAssignementOrDeclarationStatement)**

aInvocationorAssignementOrDeclarationStatement	Alf invocation or assignement or declaration statement to translate.
return	Alf invocation or assignement or declaration statement translation.

Return the Alf invocation, or assignement, or declaration, statement translation. **String generateVariableDeclarationCompletion(org.eclipse.papyrus.uml.alf.alf.VariableDeclarationCompletion aVariableDeclarationCompletion)**

aVaraibleDeclarationCompletion	Alf variable declaration completion to translate.
return	Alf variable declaration completion translation.

Return the Alf variable declaration completion translation.

String generateAssignmentCompletion(org.eclipse.papyrus.uml.alf.alf.AssignmentCompletion aAssignmentCompletion)

aAssignementCompletion	Alf assignement completion to translate.
return	Alf assignement completion translation.

Return the Alf assignement completion translation.

String generateAssignmentOperator(org.eclipse.papyrus.uml.alf.alf.AssignmentOperator aAssignmentOperator)

aAssignementOperator	Alf assignement operator to translate.
return	Alf assignement operator translation.

Return the Alf assignement operator translation.

String generateSuperInvocationStatement(org.eclipse.papyrus.uml.alf.alf.SuperInvocationStatement aSuperInvocationStatement)

aSuperInvocationStatement	Alf super invocation statement to translate.
return	Alf super invocation translation.

Return the Alf super invocation statement translation. **String generateSuperInvocationExpression(org.eclipse.papyrus.uml.alf.alf.SuperInvocationExpression aSuperInvocationExpression)**

aSuperInvocationExpression	Alf super invocation expression to translate.
return	Alf super invocation expression translation.

Return the Alf super invocation expression translation.

String generateOperationCallExpression(org.eclipse.papyrus.uml.alf.alf.OperationCallExpression aOperationCallExpression)

aOperationCallExpression	Alf operation call expression to translate.
return	Alf operation call expression translation.

Return the Alf operation call expression translation.

String generateTupleTemplate(org.eclipse.papyrus.uml.alf.alf.Tuple aTuple)

aTuple	Alf tuple to translate.
return	Alf tuple translation.

Return the Alf tuple translation.

String generateTupleElement(org.eclipse.papyrus.uml.alf.alf.TupleElement aTupleElement)

aTupleElement	Alf tuple element to translate.
return	Alf tuple element translation.

Return the Alf tuple element translation. **String generateOperationCallExpressionWithoutDot(org.eclipse.papyrus.uml.alf.alf.OperationCallExpressionWithoutDot aOperationCallExpressionWithoutDot)**

aOperationCallExpressionWithoutDot	Alf operation call expression without dot to translate.
return	Alf operation call expression without dot translation.

Return the Alf operation call expression without dot translation.

String generateThisInvocationStatement(org.eclipse.papyrus.uml.alf.alf.ThisInvocationStatement aThisInvocationStatement)

aThisInvocationstatement	Alf this invocation statement to translate.
return	Alf this invocation statement translation.

Return the Alf this invocation statement translation.

String generateInstanceCreationInvocationStatement(org.eclipse.papyrus.uml.alf.alf.InstanceCreationInvocationStatement aInstanceCreationInvocationStatement)

aInstanceCreationinvocationStatement	Alf instance creation invocation statement to translate.
return	Alf instance creation invocation statement translation.

Return the Alf instance creation invocation statement translation.

String generateAnnotation(org.eclipse.papyrus.uml.alf.alf.Annotation aAnnotation)

aAnnotation	Alf annotation to translate.
return	Alf annotation translation.

Return the Alf annotation translation.

String generateExpressionForm(org.eclipse.papyrus.uml.alf.alf.Expression aExpression)

aExpression	Alf expression to translate.
return	Alf expression translation.

Return the Alf expression translation.

String generateSequenceExpressionForm(org.eclipse.papyrus.uml.alf.alf.EList<Expression> list)

list | Alf expression list to translate.
return | Alf expression list translation.

Return the Alf expression list translation.

String generateConditionalTestExpression(org.eclipse.papyrus.uml.alf.alf.ConditionalTestExpression aConditionalTestExpression)

aConditionalTestExpression	Alf conditional test expression to translate.
return	Alf conditionnal test expression translation.

Return the Alf conditonal test expression translation.

String generateConditionalOrExpression(org.eclipse.papyrus.uml.alf.alf.ConditionalOrExpression aConditionalOrExpression)

aConditionalOrExpression	Alf conditional or expression to translate.
return	Alf conditionnal or expression translation.

Return the Alf conditonal or expression translation.

String generateConditionalAndExpression(org.eclipse.papyrus.uml.alf.alf.ConditionalAndExpression aConditionalAndExpression)

aConditionalAndExpression	Alf conditional and expression to translate.
return	Alf conditionnal and expression translation.

Return the Alf conditonal and expression translation.

String generateInclusiveOrExpression(org.eclipse.papyrus.uml.alf.alf.InclusiveOrExpression aInclusiveOrExpression)

aInclusiveOrExpression	Alf inclusive or expression to translate.
return	Alf inclusive or expression translation.

Return the Alf inclusive or expression translation.

String generateExclusiveOrExpression(org.eclipse.papyrus.uml.alf.alf.ExclusiveOrExpression aExclusiveOrExpression)

aExclusiveOrExpression	Alf exclusive or expression to translate.
return	Alf exclusive or expression translation.

Return the Alf exclusive or expression translation.

String generateAndExpression(org.eclipse.papyrus.uml.alf.alf.AndExpression aAndExpression)

aAndExpression	Alf and expression to translate.
return	Alf and expression translation.

Return the Alf conditonal test expression translation.

String generateEqualityExpression(org.eclipse.papyrus.uml.alf.alf.EqualityExpression aEqualityExpression)

aEqualityExpression	Alf equality expression to translate.
return	Alf equality expression translation.

Return the Alf equality expression translation.

String generateClassificationExpression(org.eclipse.papyrus.uml.alf.alf.ClassificationExpression aClassificationExpression)

aClassificationExpression	Alf classification expression to translate.
return	Alf classification expression translation.

Return the Alf classifcation expression translation.

String generateSequenceConstructionOrAccessCompletion(org.eclipse.papyrus.uml.alf.alf.SequenceConstructionOrAccessCom aSequenceConstructionOrAccessCompletion)

aSequenceContructionOrAccessCompletion	Alf sequence construction, or access completion to translate.
return	Alf sequence construction, or access completion translation.

Return the Alf sequence construction, or access completion translation.

String generateQualifiedNamePath(org.eclipse.papyrus.uml.alf.alf.QualifiedNamePath aQualifiedNamePath)

aQualifiedNamePath	Alf qualified name path to translate.
return	Alf qualified name path translation.

Return the Alf qualified name path translation.

String generateUnqualifiedName(org.eclipse.papyrus.uml.alf.alf.UnqualifiedName aUnqualifiedName)

aUnqualifiedName	Alf unqualified name to translate.
return	Alf unqualified name translation.

Return the Alf unqualified name translation.

String generateTemplateBinding(org.eclipse.papyrus.uml.alf.alf.TemplateBinding aTemplateBinding)

aTemplateBinding	Alf template binding to translate.
return	Alf template binding translation.

Return the Alf template binding translation.

String generateRelationalExpression(org.eclipse.papyrus.uml.alf.alf.RelationalExpression aRelationalExpression)

aRelationnalExpression	Alf relational expression to translate.
return	Alf relationnal expression translation.

Return the Alf relational expression translation.

String generateShiftExpression(org.eclipse.papyrus.uml.alf.alf.ShiftExpression aShiftExpression)

aShiftExpression	Alf shift expression to translate.
return	Alf shift expression translation.

Return the Alf shift expression translation.

String generateAdditiveExpression(org.eclipse.papyrus.uml.alf.alf.AdditiveExpression aAdditiveExpression)

aAdditiveExpression	Alf additive expression to translate.
return	Alf additive expression translation.

Return the Alf additive expression translation.

String generateMultiplicativeExpression(org.eclipse.papyrus.uml.alf.alf.MultiplicativeExpression aMultiplicativeExpression)

aMultiplicativeExpression	Alf multiplicative expression to translate.
return	Alf multiplicative expression translation.

Return the Alf multiplicative expression translation.

String generateUnaryExpression(org.eclipse.papyrus.uml.alf.alf.UnaryExpression aUnaryExpression)

aUnaryExpression	Alf unary expression to translate.
return	Alf unary expression translation.

Return the Alf unary expression translation.

String generatePrimaryExpression(org.eclipse.papyrus.uml.alf.alf.PrimaryExpression aPrimaryExpression)

aPrimaryExpression	Alf primary expression to translate.
return	Alf primary expression translation.

Return the Alf primary expression translation.

String generateValueSpecification(org.eclipse.papyrus.uml.alf.alf.ValueSpecification aValueSpecification)

aValueSpecification	Alf value specification to translate.
return	Alf value secification translation.

Return the Alf value specification translation. **String generateThisExpression(org.eclipse.papyrus.uml.alf.alf.ThisExpression aThisExpression)**

aThisExpression	Alf this expression to translate.
return	Alf this expression translation.

Return the Alf this expression translation.

String generateNullExpression(org.eclipse.papyrus.uml.alf.alf.NullExpression aNullExpression)

aNullExpression	Alf null expression to translate.
return	Alf null expression translation.

Return the Alf null expression translation.

String generateParenthesizedExpression(org.eclipse.papyrus.uml.alf.alf.ParenthesizedExpression aParenthesizedExpression)

aParenthesizedExpression	Alf parenthesizedExpression to translate.
return	Alf parenthsized expression translation.

Return the Alf parenthesized expression translation.

String generateNonLiteralValueSpecification(org.eclipse.papyrus.uml.alf.alf.NonLiteralValueSpecification aNonLiteralValueSpecification)

aNonLiteralValueSpecification	Alf non literal value specification to translate.
return	Alf non literal value specification translation.

Return the Alf non literal value specification translation. **String generateInstanceCreationExpression(org.eclipse.papyrus.uml.alf.alf.InstanceCreationExpression aInstanceCreationExpression)**

aInstanceCreationExpressionn	Alf instance creation expression to translate.
return	Alf instance creation expression translation.

Return the Alf instance creation expression translation. **String generateQualifiedNameWithBinding(org.eclipse.papyrus.uml.alf.alf.QualifiedNameWithBinding aQualifiedNameWithBinding)**

aQualifiedNameWithBinding	Alf qualified name with binding to translate.
return	Alf qualified name with binding translation.

Return the Alf qualified name with binding translation.

String generateSequenceConstructionCompletion(org.eclipse.papyrus.uml.alf.alf.SequenceConstructionCompletion aSequenceConstructionCompletion)

aSequenceContructionCompletion	Alf sequence construction completion to translate.
return	Alf sequence construction completion translation.

Return the Alf sequence construction completion translation.

String generateLiteral(org.eclipse.papyrus.uml.alf.alf.LITERAL aLITERAL)

aLITERAL	Alf literal to translate.
return	Alf literal translation.

Return the Alf literal translation.

String generateIntegerLiteral(org.eclipse.papyrus.uml.alf.alf.INTEGER_LITERAL aINTEGER_LITERAL)

aINTEGER_LITERAL	Alf integer literal to translate.
return	Alf integer literal translation.

Return the Alf integer literal translation.

String generateStringLiteral(org.eclipse.papyrus.uml.alf.alf.STRING_LITERAL aSTRING_LITERAL)

aSTRING_LITERAL	Alf string literal to translate.
return	Alf string literal translation.

Return the Alf string literal translation.

String generateBooleanLiteral(org.eclipse.papyrus.uml.alf.alf.BOOLEAN_LITERAL aBOOLEAN_LITERAL)

aBOOLEAN_LITERAL	Alf boolean literal to translate.
return	Alf boolean literal translation.

Return the Alf boolean literal translation.

String generateNumberLiteral(org.eclipse.papyrus.uml.alf.alf.NUMBER_LITERAL aNUMBER_LITERAL)

aNUMBER_LITERAL	Alf number literal to translate.
return	Alf number literal translation.

Return the Alf number literal translation.

String generateUnlimitedLiteral(org.eclipse.papyrus.uml.alf.alf.UNLIMITED_LITERAL aUNLIMITED_LITERAL)

aUNLIMITED_LITERAL	Alf unlimited literal to translate.
return	Alf unlimited literal translation.

Return the Alf unlimited literal translation.

String generateSuffixExpression(org.eclipse.papyrus.uml.alf.alf.SuffixExpression aSuffixExpression, String dot_or_arrow)

aSuffixExpression	Alf suffix expression to translate
dot_or_arrow	Suffix.
return	Alf suffix expression translation.

Return the Alf suffix expression translation.

String generatePropertyCallExpression(org.eclipse.papyrus.uml.alf.alf.PropertyCallExpression aPropertyCallExpression)

aPropertyCallExpression	Alf property call expression to translate.
return	Alf property call expression translation.

Return the Alf property call expression translation.

String generateLinkOperationExpression(org.eclipse.papyrus.uml.alf.alf.LinkOperationExpression aLinkOperationExpression)

aLinkOperationExpression	Alf link operation expression to translate.
return	Alf link operation expression translation.

Return the Alf link operation expression translation.

String generateLinkOperationTuple(org.eclipse.papyrus.uml.alf.alf.LinkOperationTuple aLinkOperationTuple)

aLinkOperationTuple	Alf link operation tuple to translate.
return	Alf link operation tuple translation.

Return the Alf link operation tuple translation.

String generateLinkOperationTupleElement(org.eclipse.papyrus.uml.alf.alf.LinkOperationTupleElement aLinkOperationTupleElement)

aLinkOperationTupleElement	Alf link operation tuple element to translate.
return	Alf link operation tuple element translation.

Return the Alf link operation tuple element translation.

String generateLinkOperationKind(org.eclipse.papyrus.uml.alf.alf.LinkOperationKind aLinkOperationKind)

aLinkOperationKind	Alf link operation kind to translate.
return	Alf link operation kind translation.

Return the Alf link operation kind translation.

String generateSequenceOperationExpression(org.eclipse.papyrus.uml.alf.alf.SequenceOperationExpression aSequenceOperationExpression)

aSequenceOperationExpression	Alf sequence operation expression to translate.
return	Alf sequence operation expression translation.

Return the Alf sequence operation expression translation.

String generateSequenceReductionExpression(org.eclipse.papyrus.uml.alf.alf.SequenceReductionExpression aSequenceReductionExpression)

aSequenceReductionExpression	Alf sequence reduction expression to translate.
return	Alf sequence reduction expression translation.

Return the Alf sequence reduction expression translation.

String generateSequenceExpansionExpression(org.eclipse.papyrus.uml.alf.alf.SequenceExpansionExpression aSequenceExpansionExpression)

aSequenceExpansionExpression	Alf sequence expansion expression to translate.
return	Alf sequence expansion expression translation.

Return the Alf sequence expansion epxression translation.

String generateClassExtentExpression(org.eclipse.papyrus.uml.alf.alf.ClassExtentExpression aClassExtentExpression)

aClassExtentExpression	Alf class extent expression to translate.
return	Alf class extent expression translation.

Return the Alf class extent expression translation. **String generateLocalNameDeclarationStatement(org.eclipse.papyrus.uml.alf.alf.LocalNameDeclarationStatement aLocalNameDeclarationStatement)**

aLocalNameDeclarartionStatement	Alf local name declaration statement to translate.
return	Alf local name declaration statement translation.

Return the Alf local name declaration statement translation. **String generateNameExpression(org.eclipse.papyrus.uml.alf.alf.NameExpression aNameExpression, Boolean isNotNew)**

aNameExpression	Alf name expression to translate.
isNotNew	is a new name expression ?
return	Alf name expression translation.

Return the Alf name expression translation.

Athena_AlfGenerator

Specialized Alf generator to translate Alf bloc to Athena code. It inherit from AlfGenerator.

protected

String generateAlfBlock(org.eclipse.papyrus.uml.alf.alf.Block bloc)

see AlfGenerator.generateAlfBlock

String generateStatement(org.eclipse.papyrus.uml.alf.alf.Statement aStatement)

see AlfGenerator.generateStatement

String generateIfStatement(org.eclipse.papyrus.uml.alf.alf.IfStatement aIfStatement)

see AlfGenerator.generateifStatement

String generateSequentialClausesTemplate(org.eclipse.papyrus.uml.alf.alf.SequentialClauses aSequentialClauses)

see AlfGenerator.generateSequentialClausesTemplate

String generateConcurrentClausesTemplate(org.eclipse.papyrus.uml.alf.alf.ConcurrentClauses aConcurrentClauses)

see AlfGenerator.generateConcurrentClausesTemplate

String generateNonFinalClauseTemplate(org.eclipse.papyrus.uml.alf.alf.NonFinalClause aNonFinalClause)

see AlfGenerator.generateFinalClauseTemplate

String generateWhileStatement(org.eclipse.papyrus.uml.alf.alf.WhileStatement aWhileStatement)

see AlfGenerator.generateWhileStatement

String generateInvocationOrAssignementOrDeclarationStatement(org.eclipse.papyrus.uml.alf.alf.InvocationOrAssignementOr aInvocationOrAssignementOrDeclarationStatement)

see AlfGenerator.generateInvocationOrAssignementOrDeclarationStatement

String generateVariableDeclarationCompletion(org.eclipse.papyrus.uml.alf.alf.VariableDeclarationCompletion aVariableDeclarationCompletion)

see AlfGenerator.generateVariableDeclarationCompletion

String generateLocalNameDeclarationStatement(org.eclipse.papyrus.uml.alf.alf.LocalNameDeclarationStatement aLocalNameDeclarationStatement)

see AlfGenerator.generateLocalNameDeclarationStatement

String generateNameExpression(org.eclipse.papyrus.uml.alf.alf.NameExpression aNameExpression, Boolean isNotNew)

see AlfGenerator.generateNameExpression

CPP_AlfGenerator

Specialized Alf generator to translate Alfbloc to C++ code. It inherit from AlfGenerator.

public

CPP_AlfGenerator()

Constrcutor

protected

String generateAlfBlock(org.eclipse.papyrus.uml.alf.alf.Block bloc)

see AlfGenerator.generateAlfBlock

String generateStatement(org.eclipse.papyrus.uml.alf.alf.Statement aStatement)

see AlfGenerator.generateStatement

String generateInlineStatement(org.eclipse.papyrus.uml.alf.alf.InlineStatement aInLineStatement)

see AlfGenerator.generateInlineStatement

String generateIfStatement(org.eclipse.papyrus.uml.alf.alf.IfStatement aIfStatement)

see AlfGenerator.generateIfStatement

String generateSequentialClausesTemplate(org.eclipse.papyrus.uml.alf.alf.SequentialClauses aSequential-Clauses)

see AlfGenerator.generateSequentialClausesTemplate

String generateConcurrentClausesTemplate(org.eclipse.papyrus.uml.alf.alf.ConcurrentClauses aConcurrent-Clauses)

see AlfGenerator.generateConcurrentClausesTemplate

String generateNonFinalClauseTemplate(org.eclipse.papyrus.uml.alf.alf.NonFinalClause aNonFinalClause)

see AlfGenerator.generateNonFinalClauseTemplate

String generateSwitchStatement(org.eclipse.papyrus.uml.alf.alf.SwitchStatement aSwitchStatement)

see AlfGenerator.generateSwitchStatement

String generateSwitchDefaultclause(org.eclipse.papyrus.uml.alf.alf.SwitchDefaultClause aSwitchDefault-Clause)

see AlfGenerator.generateSwitchDefaultclause

String generateSwitchClause(org.eclipse.papyrus.uml.alf.alf.SwitchClause aSwitchClause)

see AlfGenerator.generateSwitchClause

String generateSwitchCase(org.eclipse.papyrus.uml.alf.alf.SwitchCase aSwitchCase)

see AlfGenerator.generateSwitchCase

String generateWhileStatement(org.eclipse.papyrus.uml.alf.alf.WhileStatement aWhileStatement)

see AlfGenerator.generateWhileStatement

String generateDoStatement(org.eclipse.papyrus.uml.alf.alf.DoStatement aDoStatement)

see AlfGenerator.generateDoStatement

String generateInvocationOrAssignementOrDeclarationStatement(org.eclipse.papyrus.uml.alf.alf.InvocationOrAssignementOrI aInvocationOrAssignementorDeclarationStatement)

see AlfGenerator.generateInvocationOrAssignementOrDeclarationStatement

String generateVariableDeclarationCompletion(org.eclipse.papyrus.uml.alf.alf.VariableDeclarationCompletion aVariableDeclarationCompletion)

see AlfGenerator.generateVariableDeclarationCompletion

String generateSuperInvocationExpression(org.eclipse.papyrus.uml.alf.alf.SuperInvocationExpression aSuperInvocationExpression)

see AlfGenerator.generateSuperInvocationExpression

String generateOperationCallExpressionWithoutDot(org.eclipse.papyrus.uml.alf.alf.OperationCallExpressionWithoutDot aOperationCallExpressionWithoutDot)

see AlfGenerator.generateOperationCallExpressionWithoutDot

String generateThisExpression(org.eclipse.papyrus.uml.alf.alf.ThisExpression aThisExpression)

see AlfGenerator.generateThisExpression

String generateInstanceCreationExpression(org.eclipse.papyrus.uml.alf.alf.InstanceCreationExpression aInstanceCreationExpression)

see AlfGenerator.generateInstanceCreationExpression

String generateQualifiedNameWithBinding(org.eclipse.papyrus.uml.alf.alf.QualifiedNameWithBinding aQualifiedNameWithBinding)

see AlfGenerator.generateQualifiedNameWithBinding

String generateLocalNameDeclarationStatement(org.eclipse.papyrus.uml.alf.alf.LocalNameDeclarationStatement aLocalNameDeclarationStatement)

see AlfGenerator.generateLocalNameDeclarationStatement

String generateNameExpression(org.eclipse.papyrus.uml.alf.alf.NameExpression aNameExpression, Boolean isNotNew)

see AlfGenerator.generateNameExpression

6.7.2 How to use an existing Alf generator

You can use an Alf generator in your *RobotML* generator development. You have two solution to call an existing Alf generator. But that depend how you develop your *RobotML* generator.

- If you use *Acceleo* template, you should to import the AlfServices into your template.Then call *createAlfBloc* to get an Alf bloc corresponding to your operation body, and call the right operation translation you need.

```
[let behavior : OpaqueBehavior = element.oclAsType(OpaqueBahevior)]
   [let bloc : alf::Block = createAlfBloc(behavior)]
      [if((bloc = null) =(false))]
         [translateAlfBlocTo_Athena(bloc)/]
       [/if]
   [/let]
[/let]
```

- If you use JAVA class, you should to refrence the Alf generator API, and call the static methods *createAlfBloc* and the translation operation.

```
OpaqueBahevior behavior = (OpaqueBehavior)element;
org.eclipse.papyrus.uml.alf.alf.Block bloc = AlfServices.createAlfBloc(behavior);
if(bloc != null)
{
   String result = AlfServices.translateAlfBlocTo_Athena(behavior);
}
```

6.7.3 How to create a new Alf generator

If you use a specific language (not using in the robotml platform), you can create a new Alf generator with this API. You should do the following steps:

1. Create a new class generator *XXX_AlfGenerator*, inherited from AlfGenerator.

Note: By convention your generator shouk be suffixed *AlfGenerator*.

2. Override the needing methods to specify your result language generation.
3. Modify the AlfServices class to lauch your generator.

Note: By convention your calling methods should be prefixed by *translateAlfBlocTo_*.

```
public static String translateAlfBlocTo_XXX(org.eclipse.papyrus.uml.alf.alf.Block bloc)
{
   return AlfServices.translateAlfBloc(bloc, new XXX_AlfGenerator());
}
```

4. Call your Alf generator on your source code

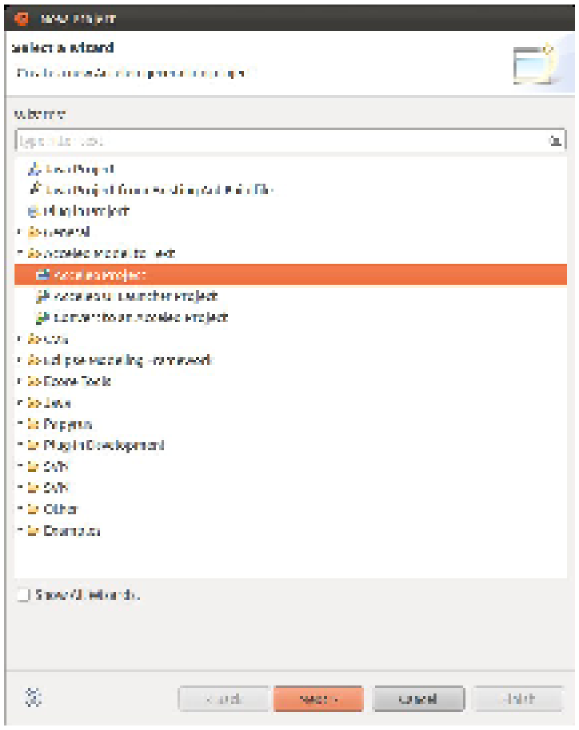
See also:

how to use an existing Alf generator

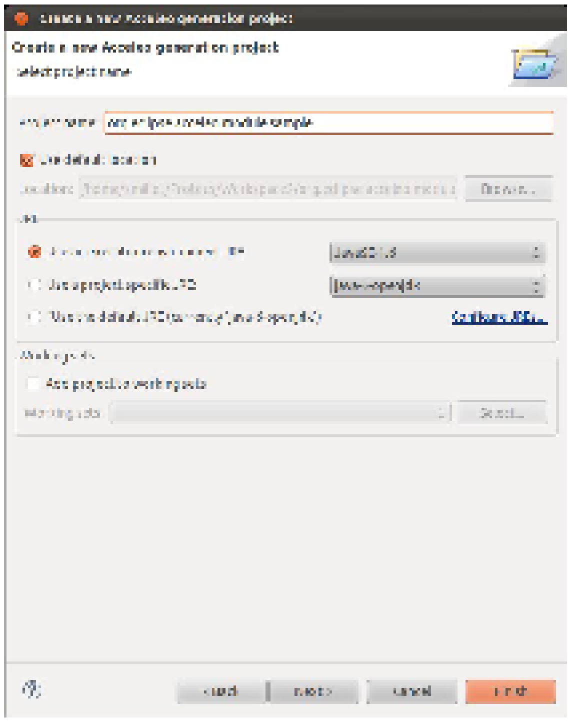
RobotML generator <CodeGenerators/RobotMLGenerators/RobotMLGenerators> Naming conventions <NamingConventions> Standard sensors structures <StandardSensorsStructures>

6.8 How to create a new RobotML generator

In your **Project explorer** view, do right clic and select **New/Project..**. In the **New project** window, choose **Acceleo prject** in the **Acceleo model to text** category, and clic on **Next**.



In the window **Create a new Acceleo generator project**, name your project, and select the 1.5 JAVA version (J2SE-1.5). Clic on **Next**.


Create a new Acceleo generator project
Create a new Acceleo generation project
select project name
Use default location
Browse...
Configure JREs...
Cancel
Finish

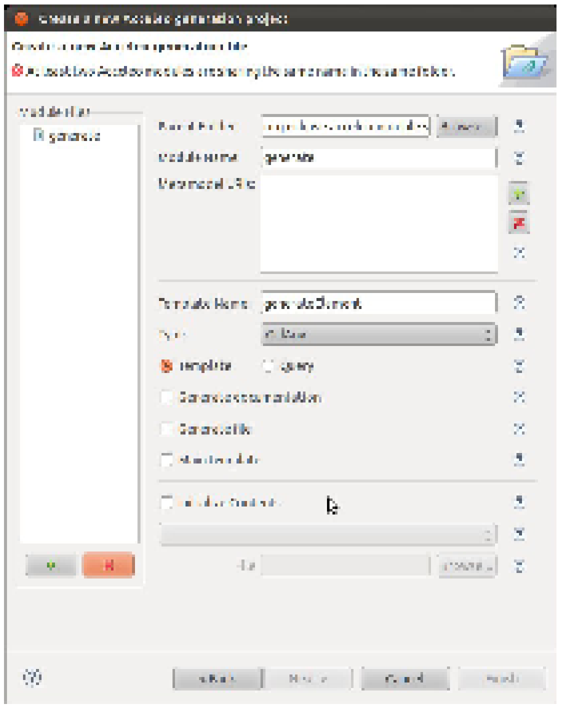
Now named your new module, and on the **add** button in the metamodel section. A metamodel list is shown. Select your metamodel should to use, and clic on OK. For example if you should use UML, select **http://www.eclipse.org/uml2/3.0.0/UML** in the **developpement time version** tab.

Note: If you should use a custom domain specific language, select your metamodel in the **Runtime version** tab.

Select your module type:

- **Template**, if you should to realize some generation with the metamodel.
- **Query**, if you should to do a functions library.

Clic on **Finish**



Your new RobotMLGenerators will added in your **Project explorer**. Now

6.9 Adding a new module to an existing RobotML generator

On your **Acceleo generator project**, do right clic, and select **new/Other...**. Chosse **Acceleo module file** in the category **Acceleo model to text**, and clic **Next**.

Name your module, select the metamodel should to use, and select the module type. Clic on **Finsih**.

See also:

How to create a new RobotML generator

6.10 Adding your new generator in the RobotML generator interface

Import the **RobotML generator project** in your workspace (org.eclipse.robotml.generators.generator.ui). Open the file *MANIFEST.MF* and slect the **Dependencies** tab. In the **Required plug-in** section, add your generator plug-in. In the package **org.eclipse.robotml.generators.generator.ui.xml**, open the file **config.xml**, and add you generator declaration int the **generator_list** node.

The following example show how to declare generators:

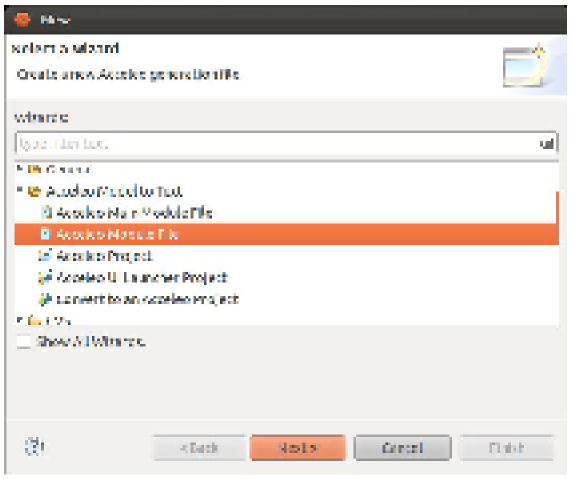
```
<robotml>
   <domain_list>
      <domain name="athenaDSL">
         <implementation class="org.xtext.athenaDSL.impl.AthenaDSLFactoryImpl" method="init"/>
         <implementation class="org.xtext.athenaDSL.impl.AthenaDSLPackageImpl" method="init"/>
      </domain>
   </domain_list>
   <generator_list>
      <generator name="athena" id="org.eclipse.robotml.generators.acceleo.athena" class="org.eclipse
      <generator name="vle" id="org.eclipse.robotml.generators.xtext.athena.vle" class="org.eclipse.r
      <generator name="athena-simu" id="org.eclipse.robotml.generators.xtext.athena.simu" class="org
   </generator_list>
</robotml>
```

Node	Attribute	Description
robotml		Root node.
domain_list		Declaration node for a domain list.
domain	• name	Domain language declaration. - Language name.
implementation	• class • method	Implementation declaration. - Class containing the method to call. - Method name to call.
generator_list		Declaration node for a generator list.
generator	• name • id • class • src-type • src-ext • target-folder • src-generator	Generator declaration. - Generator name. - Generator ID (org.eclipse.robotml.generators.XXX). - Generator class. - Source type (model or file). - Source file extension (Only if src="file"). - Target folder name. - Name of the source generator used to generate the source file (only if src="file").

Note: If your should to define a new Domain Specific Language, you shoukd to declare it on the file **config.xml**. Add a **domain** node, and specify the implementation to initialize your domain language (see the example).

6.11 Create a user interface for a RobotMl generator

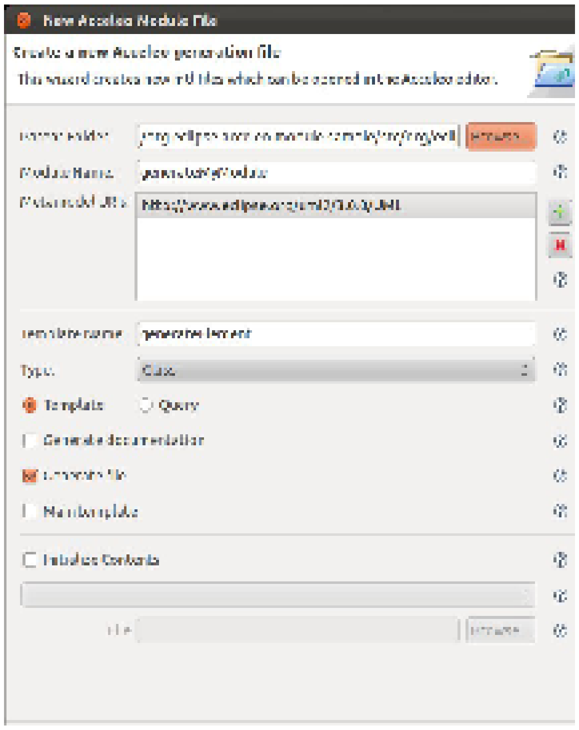
In your **Acceleo generator project**,, do right clic, and select **Acceleo/Create Acceleo UI launcher**. Name you UI project, clic on **Next**. Select the project who will be called by your interface.



Give a label to your generator.

> **Warning:** This label will be visible in the environment context menu.

Choose your working file filter. Your generator interface will be visible only if the selection match this file filter. Clic on **Finish**.


New Acceleo Module File
Create a new Acceleo generation file
This wizard creates new MTL files which can be opened in the Acceleo editor.
Browse...

Parent Folder: [illegible]
Module Name: [illegible]
Metamodel URIs: http://www.eclipse.org/uml2/3.0.0/UML
Template name: generateElement
Type: Class
[✓ Template] Query
Generate documentation
[✓ Generate file]
Main template
Initialize Contents
File:

6.12 Implement a RobotML generator in your user interface

In your Acceleo User interface project, open the file *GenerateAll.java*, and modifiy the method doGenerate() as following:

```
public void doGenerate(IProgressMonitor monitor) throws IOException {
   if (!targetFolder.getLocation().toFile().exists()) {
      targetFolder.getLocation().toFile().mkdirs();
   }

   monitor.subTask("Loading...");

   //GenerateXXX is the main class of the RobotML generator. By convention, <XXX> is the name of the
   //Example for 'Athena' langauge, the name of the main class is GenerateAthena.
   GenerateXXX generator = new GenerateXXX(modelURI, targetFolder.getLocation().toFile(), arguments);
   monitor.worked(1);
   String generationID =
      org.eclipse.acceleo.engine.utils.AcceleoLaunchingUtil.computeUIProjectID("org.eclipse.robotml.g
      "org.eclipse.robotml.generators.acceleo.athena.files.GenerateXXX",
      modelURI.toString(),
      targetFolder.getFullPath().toString(),
      new ArrayList<String>());
   generator.setGenerationID(generationID);
   generator.doGenerate(BasicMonitor.toMonitor(monitor));
}
```

6.13 Validate a new RobotML generator

In your **project explorer** view, select your *RobotML* generator, and do right clic. Choose, **Run as.../Eclipse application**. A new eclipse applciation start. In this new envioronment, import the needed files to execute your generator, then select a imported file, do right clic. The contextual menu, should contain your gnerator label. Select it to execute your generator.

6.14 Exception

In exception, if you using particular metamodel, it's necessary to register it befor using. For example : if you use the *DSL* Athena, you should to initialized it with the following methods in your user interface code.

```
AthenaDSLFactoryImpl.init();
AthenaDSLPackageImpl.init();
```

You need also to modifiy the *RobotML* generator code registerPackage method as following:

```
public void registerPackages(ResourceSet resourceSet) {
   super.registerPackages(resourceSet);
   if(!isInWorkspace(org.xtext.athenaDSL.AthenaDSLPackage.class)) {
   resourceSet.getPackageRegistry().put(org.xtext.athenaDSL.AthenaDSLPackage.eINSTANCE.getNsURI(),
   org.xtext.athenaDSL.AthenaDSLPackage.eINSTANCE);
   }
}
```

Warning: Do not forget replace the tag @generated by @generated not to not loosing your modification, when you modifiy the template file module.

CHAPTER

SEVEN

ROBOTML WEB PORTAL

CHAPTER

EIGHT

GLOSSARY

Note: Glossary is closely linked to the ontology developed by the consortium in order to support the definition of the *RobotML DSL*. The reader willing to know more about this subject is to follow http://www.anr-proteus.fr/?q=node/111

Acceleo **Acceleo** is the tool used in the *RobotML* platform to implement generators towards simulator and robotic middle-ware frameworks. More information can be found on Acceleo website.

Algorithm An algorithm is a recette describing elementary behaviour of module that can be found in robotic systems (see wikipedia for more information)

Application It is a computing entity able to work upon a hardware that provides computing capabilities. It is also a computing entity capable of providing life to another software. At the bottom level, it is the OS of the computer.

ANR **ANR** stands for **A**gence **N**ationale de la **R**echerche. More information can be found on ANR website

ATHENA **ATHENA** is a language associated with generators to go towards specific simulator code. It was developed thanks to an open consortium during its infancy but in its later flavour is totally owned by Dassault Aviation and used in collaborative project to help integrate development of diverse contributors into a unique application (as an example do consider http://www.pegase-project.eu).

ARROCAM ARROCAM is one of the existing Robotic middle-ware target that exists for the *RobotML* JUNO flavour. This middle-ware is developped by EFFIDENCE company. More information is available following www.effidence.com

BRICS **BRICS** stands for **B**est p**R**actice in robot **ICS***. It is an European project which goals are overlapping PROTEUS project. More information can be found followin http://www.best-of-robotics.org/)

Cycab-Tk Cycab-Tk is one of the existing Environment simulator target that exists for the *RobotML* JUNO flavour. This middle-ware is developped by a community under *INRIA* direction. More information is available following gforge.inria.fr/projects/cycabtk/

developper The **developper** is someone intervening on the *RobotML* platfrom development and thus that interacts with the development's repositories.

DEVS **DEVS** stands for **D**iscrete **EV**vent system **S**pecification. It is a mathematical approach of how to model system and to simulate them. More information can be found on DEVS wikipedia webpage

DSL **DSL** stands for **D**omain **S**pecific **L**anguage. Quotation from wikipedia: *It is a programming language or specification language dedicated to a particular problem domain, a particular problem representation technique, and/or a particular solution technique.* More information can be found following http://en.wikipedia.org/wiki/Domain-specific_language

ECLIPSE **ECLIPSE** is the platform on which the *RobotML* tooling was built. More information on it can be found on eclipse project website.

EMF **EMF** (**E**clipse **M**odeling **F**ramework) is an Eclipse-based modeling framework and code generation facility for building tools and other applications based on a structured data model.

EPL **EPL** (**E**clipse **P**ublic **L**icense) is an open source software license used by the **'Eclipse Foundation<http://www.eclipse.org>'_** for its software.

EUROP / EURON EUROP / EURON stands for EUROpean Platform / EUropean RObotics Network.

git **git** is a distributed revision control and source code management (SCM) system.

GDR Robotique **GDR** stands for **G**roupement **D**e **R**echerche. It is a community backed up by the French research group CNRS that groups the different laboratories concerned by Robotic research topics. It is associated to a club grouping interested Industrial partners. A Portal exists that allows interested readers to access this community.

LAAS LAAS stands for **L**aboratoire d'**A**nalyse et d'**A**rchitecture des **S**ystèmes. It is the biggest French CNRS team and in the scope of *RobotML* is supporting the *MORSE* environment simulator.

LIRMM LIRMM stands for **L**aboratoire d'**I**nformatique, de **R**obotique et de **M**icroelectronique de **M**ontpellier

MARTE **MARTE** (**M**odeling and **A**nalysis of **R**eal **T**ime and **E**mbedded systems) is the *OMG* standard for modeling real-time and embedded applications with UML2.

MBD **MBD** (**M**odel-**B**ased **D**esign) is a mathematical and visual method of addressing problems associated with designing complex control, signal processing and communication systems. It is used in many motion control, industrial equipment, aerospace, and automotive applications. Model-based design is a methodology applied in designing embedded software.

MDA **MDA** (**M**odel-**D**riven **A**rchitecture) is a software design approach for the development of software systems. It provides a set of guidelines for the structuring of specifications, which are expressed as models. Model-driven architecture is a kind of domain engineering, and supports model-driven engineering of software systems. It was launched by the Object Management Group (*OMG*) in 2001.

model A model provides an abstract view of the situation someone wants to convey to another person. This view if embodied using a Domain Specific Language can be used for other purposes such as documentation, generation of elements able to insert in a software application and so on and so forth

module A **module** in our context is exclusively, as delivered from a portal, software.

MORSE It is an add-on of the *BLENDER* 3D modelling environment that allows to simulate Physical environment and physical objects deplyed into it. As an example, It is what create sensible information to the different sensors.

OMG **OMG** (**O**bject **M**anagement **G**roup) is an international, open membership, not-for-profit computer industry standards consortium. *OMG* Task Forces develop enterprise integration standards for a wide range of technologies and an even wider range of industries. *OMG*'s modeling standards enable powerful visual design, execution and maintenance of software and other processes. Originally aimed at standardizing distributed object-oriented systems, the company now focuses on modeling (programs, systems and business processes) and model-based standards.

OROCOS It stands for **O**pen **RO**bot **CO**ntrol **S**oftware. It is a component based architecture and its goal is to allow clear separation of behaviour implementation and architecture concerns. Another goal is to provide standard behaviour components for decision issues. More information can be found on OROCOS website.

OS **OS** stands for **O**perating **S**ystem. It is the software that makes everything run on a computer

Papyrus **Papyrus** is the toolset on which the *RobotML* language has been based. More information can be found on papyrus website.

portal The portal is the commonplace where the robotic community will be able to upload and download data

PRISME PRISME stands for institut **P**luridisciplaire de **R**echerche en **I**ngenierie des **S**ystemes **M**ecanique, **E**nergetique de Bourges

problem A *problem* is

1. the definition of the architecture of one or more robots and their environment (in our case, using the proteus main tool);
2. the definition inside the robot(s) architecture of one or multiple components in which it is necessary to develop *solutions*;
3. the definition of *probes* that allow *simulation* exploitation;
4. the definition of the *metrics* and their associated validity domain in order to measure the *solution* quality and compare it to other *solutions*.

provider A provider is someone able to connect to the *portal* in order to upload to it the elements he/she can provide

PROTEUS It is the project that allowed the creation of all the data, tools, resources described by this documentation. **PROTEUS** stands for **P**latform for **RO**botic modelling and **T**ransformations for **E**nd-**U**sers and **S**cientific communities (There is also a french reading of this acronym: **P**lateforme pour la **R**obotique **O**rganisant les **T**ransferts **E**ntre **U**tilisateurs et **S**cientifiques). More information are to be found on PROTEUS website.

RCP it stands for *R*ich *C*lient *P*latform. It is the tool used in order to create the *RobotML* platform.

robot **Robot** definition is of ambiguous nature. Considering its wikipedia embodiement: *A robot is usually an electro-mechanical machine that is guided by a program or circuitry. Robots can be autonomous, semi-autonomous or remotely controlled and range from humanoids such as ASIMO and TOPIO to Nano robots, 'swarm' robots, and industrial robots*. It is a definition large enough to include aircraft, cars, mobile aspirators, etc. Considering the context, we will stick to this definition not emphasising humanoïd robots.

RobotML it stands for **Robot Modelling Language**. It is the Domain Specific Language that was developped during the PROTEUS project to allow Robotic community to create abstract views of robotic problems, solutions, etc.

RobotML-sdk it stands for *RobotML* **s**oftware **d**evelopment **t**oolkit. It is the set of tools associated with the RobotML platform in order to allow generation towards different simulators / robotic middlewares targets. This sdk is supported for the time being only on UBUNTU12.04. For more detail consult *the page dedicated to the sdk installation*.

ROS ROS stands for Robot Operation System. its goal is to provide a standard framework allowing to make modules into application. In the :term'RobotML' platform context it has been chosen as the default communication bus.

RTMaps N/A

simulator It is an application that in RobotML-sdk context must be generated by the RobotML platform and that after a configuration phase is executed

solution

1. In a *problem* context, A *solution* is an *algorithm* itself possibly integrating an *architecture* that allows during the execution of a *simulator* to extract values from the defined *metrics* that remains in their validity domain.
2. Subpart of interest in the software system of a robot. Its implementation is provided by a *Solution Provider*.

scenario A **Scenario** is a context defined loosely where its reader will understand what are the elements she / he will find in the *problems* that will be derived from it. In order to provide a scenario, there are no obligations to provide formalised elements such as required by a *problem*. Video, documents, pictures, etc that are sufficient to let those willing to use them to understand this context. It can be detailed in many details as well as only explaining the different actors or only goals.

sensor Equipment that detects, measures, and/or records physical phenomena, and indicates objects and activities by means of energy or particles emitted, reflected, or modified by the objects and activities.

SVN **SVN** stands for SubVersioN. It is a Version Control software that allows developers to synchronise source code in a distributed fashion. In the context of *RobotML* platform, there is on the *portal* a SVN erver allowing *user*s to share their experiences

Subclipse **Subclipse** is an Eclipse Team Provider plug-in providing support for Subversion within the Eclipse IDE. The software is released under the Eclipse Public License (*EPL*) 1.0 open source license.

SysML **SysML** (**S**ystems **M**odeling **L**anguage) 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.

user A user is someone able to connect to the *portal* in order to download from it the elements he/she would like to use namely the *RobotML* platform

VLE **VLE** stands for **V**irtual **L**aboratory **E**nvironment. It is an implementation of the *DEVS* theory allowing its users to have a safe approach of their system models. More information can be found on VLE website

UML **UML** (**U**nified **M**odeling **L**anguage) is a standardized (ISO/IEC 19501:2005), general-purpose modeling language in the field of software engineering. The Unified Modeling Language includes a set of graphic notation techniques to create visual models of object-oriented software-intensive systems

Xtext **Xtext** allows to define syntax, grammar. It is the basis of source coloured editor (in the platform case it is useful for the *ATHENA* code edition)

More information on RobotML are available through its dedicated portal that is accessible directly following http://rim.bourges.univ-orleans.fr/ .

Warning: Documentation is written solely in English due to lack of resources that would have allowed at least a French version. The reason for this choice is the importance of dissemination and cooperation at the European Level (see projects such as *BRICS*).

CHAPTER

NINE

DOCUMENTATION HOW-TO

The documentation you are reading has been generated using the sphinx tool. It is a part of the development in itself and here follows tho sections that explicits how to install the sphinx tool, where to find its associated documentation and provides a list of known bugs and frequently asked questions.

9.1 Sphinx and RobotML documentation

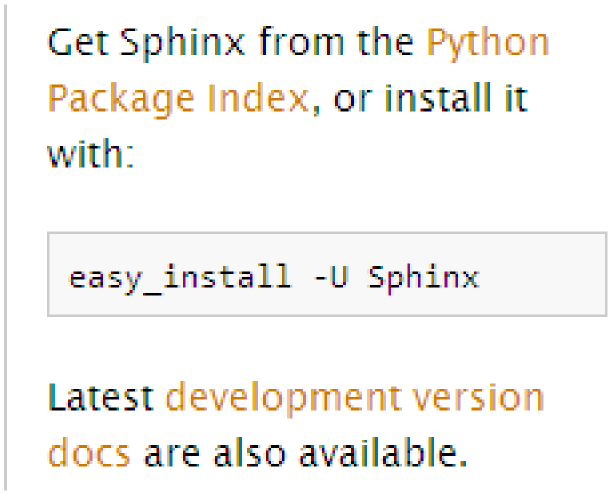
Documentation of this project is created thanks to the Sphinx tool. This tooling is using Python as its underlying framework. It means that it is indepedant of the platform on which you will work (It is more or less true, if you have any problems do send a mail to Serge Stinckwhich in order to ask questions or provide bugs).

It is possible to download the complete Sphinx documentation in acrobat format following http://sphinx.pocoo.org/sphinx.pdf.

9.1.1 Installation of Sphinx

In order to install this tool, you have to:

1. Install *Python*: version used by the consortium is 2.7.3 considering documentation generation. You can download it from here choosing your own flavour. This version is verified but it does not mean the others are not working (we are speaaking here of the version 3 branch). 2. Install *Sphinx*: version used for generating documentation is 1.1.3. Hereafter a screenshot of the website indicating how to install.



Get Sphinx from the Python Package Index, or install it with:

```
easy_install -U Sphinx
```

Latest development version docs are also available.

Figure 9.1: *Install Sphinx*

9.1.2 How to generate with Sphinx

There is no graphical console allowing to push any buttons. It is needed to

1. open a terminal (process is different with respect to the different OS) 2. verify you have access to sphinx command (PATH to complete and it depends also of the OS used) 3. change location in the folder tree in order to position to the root of the documentation (in PROTEUS distribution case on Linux, it means folder XXX) 4. type hereafter command (if -b option is forgotten, default is html, read documentation for more information)

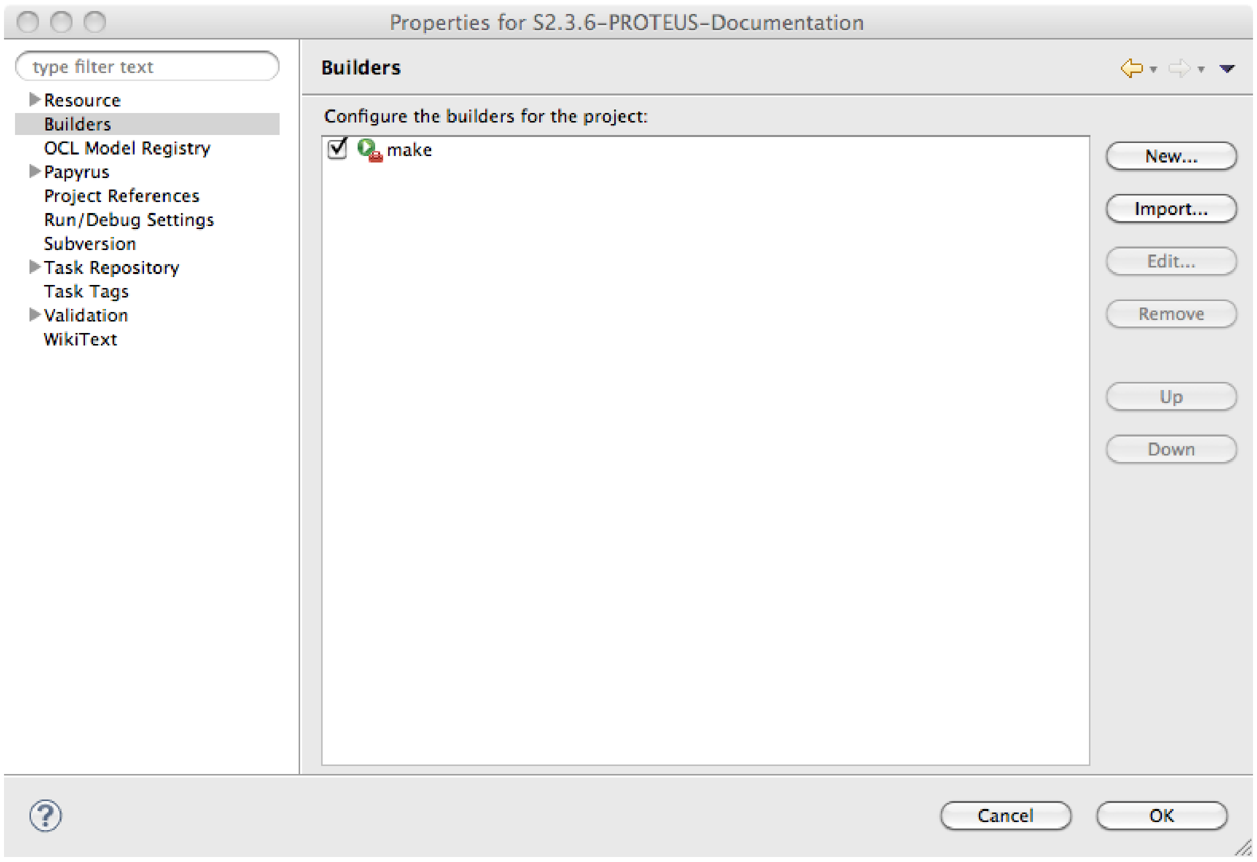
```
sphinx-build -b html . destination_directory
```

5. you are now able to consider the result in the *destination_directory*

9.1.3 ECLIPSE RsT plugin

It is possible to install an extension in the eclipse platform in order to manage Sphinx projects. You have to use the editor update site *http://resteditor.sourceforge.net/eclipse* and then a RsT editor will be accessible. This plugin is not perfect but allows a management of the documentation projects in the same interface where everything else is done.

It is necessary to create a generic project and then create the build command (menu *project/properties*, then consider the *builders* choice in the left pane).


Properties for S2.3.6-PROTEUS-Documentation

Builders

Configure the builders for the project:

☑ make

Figure 9.2: *how to choose builders index*

It is necessary then to create the project customised builder (there is no possibility to do otherwise). Either you create the builder from scratch (use the New buider button) or you edit existing buider (use the edit button after selecting the builder to edit).

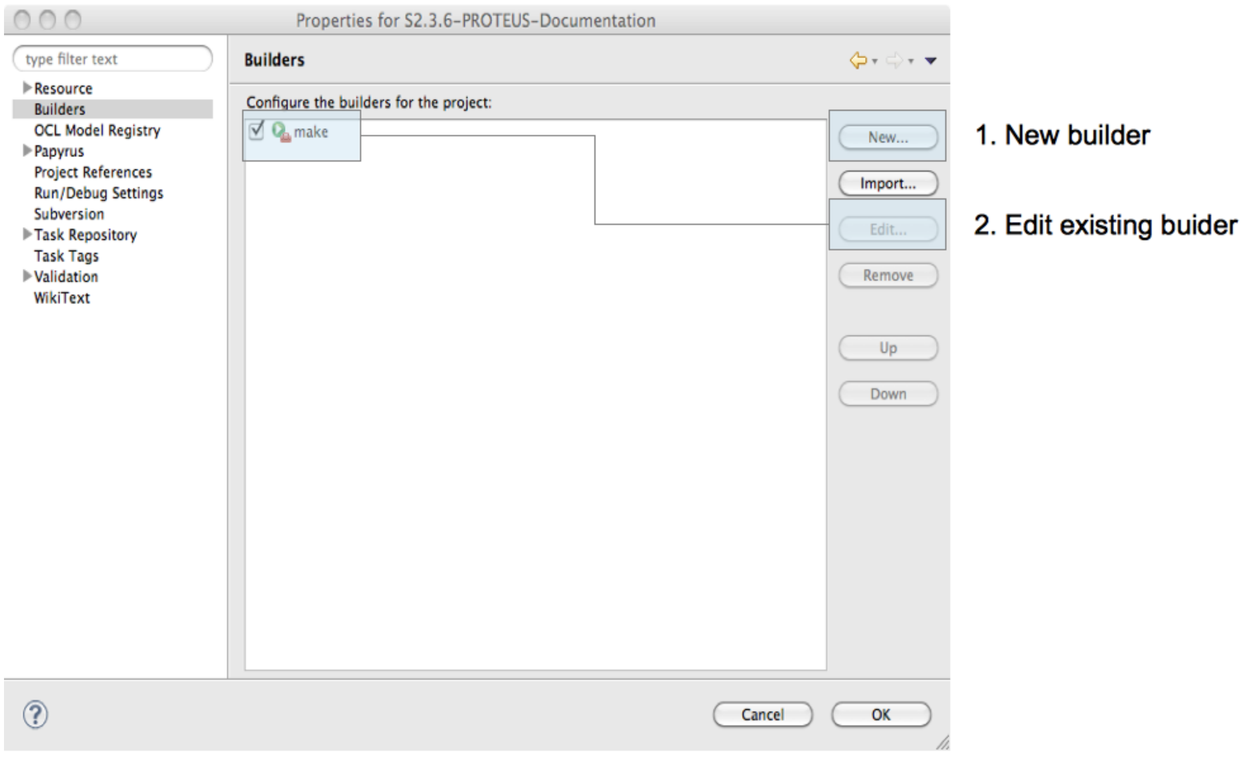
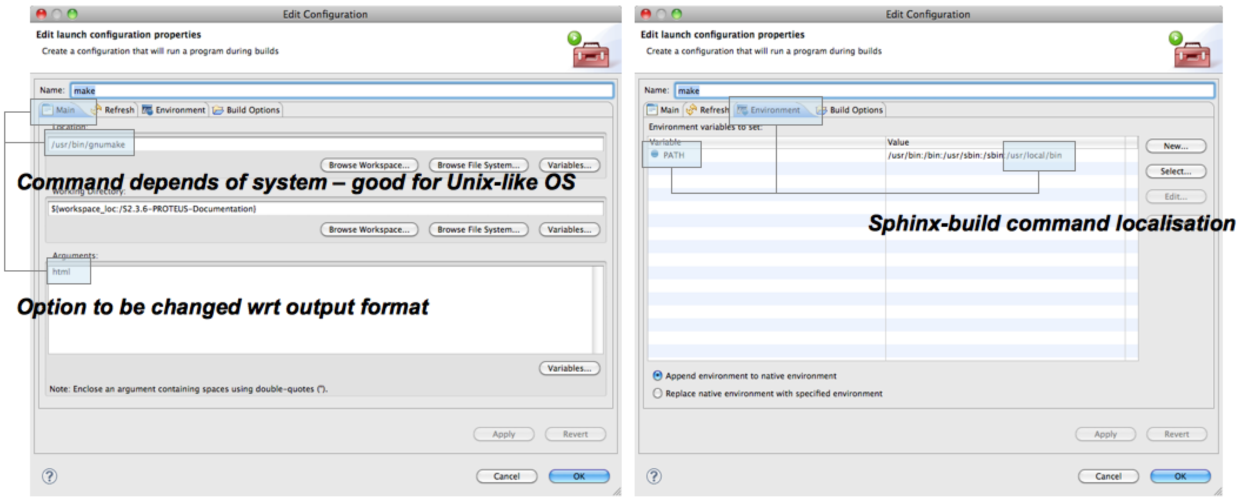


Figure 9.3: *how to create customised builder*

At this step, if you installed Sphinx, you are able to specify the command to launch when building the project. There are subtleties with respect to the different OS and what is shown concerns MacOS and Linux but is easily adaptable to Windows OS.

9.2 Bugs and questions

Bugs and questions on the documentation should be sent to the author of the page when indicated and if not to the repsonnsible of the documentation, Serge Stinckwhich.


Edit Configuration
Edit launch configuration properties
Create a configuration that will run a program during builds
Main
Refresh
Environment
Build Options
Browse Workspace...
Browse File System...
Variables...
Command depends of system – good for Unix-like OS
Option to be changed wrt output format
Sphinx-build command localisation
New...
Select...
Edit...
Apply
Revert
Cancel
OK

Name: make

Location: /usr/bin/gnumake

Working Directory: ${workspace_loc:/S2.3.6-PROTEUS-Documentation}

Arguments: html

Note: Enclose an argument containing spaces using double-quotes (").

Name: make

Environment variables to set:

Variable	Value
PATH	/usr/bin:/bin:/usr/sbin:/sbin:/usr/local/bin

[✓ Append environment to native environment]
Replace native environment with specified environment

Figure 9.4: *how to customise builder*

CHAPTER

TEN

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*

CHAPTER ELEVEN

RELATED WEBSITES

- eclipse
- papyrus
- BRICS
- Christian Schlegel
- OROCOS