Document title
Eclipse Papyrus Plugins for the Arrowhead Framework
Date
2022-11-10
Author
Cristina Paniagua
Contact
cristina.paniagua@ltu.se

Document type
Tool
Version
1.0
Status
Beta version
Page
1 (15)

Eclipse Papyrus Plugins for the Arrowhead Framework

Tool Documentation

Abstract

This document describes the Arrowhead plugins developed in the Eclipse IDE and Papyrus to lower the engineering effort and entrance barrier to the Arrowhead framework. The three plugins described are the Core Systems Deployment plugin, the Application System Skeletons Deployment plugin, and the Rule Deployment plugin. The document includes a detailed description of each plugin followed by a tutorial on how to install and use them, and future actions for planned releases.

ARTEMIS Innovation Pilot Project: Arrowhead
THEME [SP1-JTI-ARTEMIS-2012-AIPP4 SP1-JTI-ARTEMIS-2012-AIPP6]
[Production and Energy System Automation Intelligent-Built environment and urban infrastructure for sustainable and friendly cities]

Contents

1 Overview

The integration of cyber-physical systems (CPS) in big system of systems (SoS) means a shift the value chain towards a software-driven paradigm. This situation results in an increase in the demand for computer scientists and developers. However, this tendency is not easily covered, the number of resources is not enough to fulfill the requirements. In this scenario to improve software engineering methods and tools to obtain a more efficient work process is key to match and maintaining the current industrial demands.

One measure to overcome the increasing number of needed resources is the automatization of the engineering process. The steps defined by the standard IEC 81346 in the engineering process together with the arrowhead entities are presented in the following matrix, Fig. 1. The development of the plugins has been done to facilitate the transition between the functional design and the procurement and engineering steps. Paying special attention to the description and implementation of local clouds and their systems.

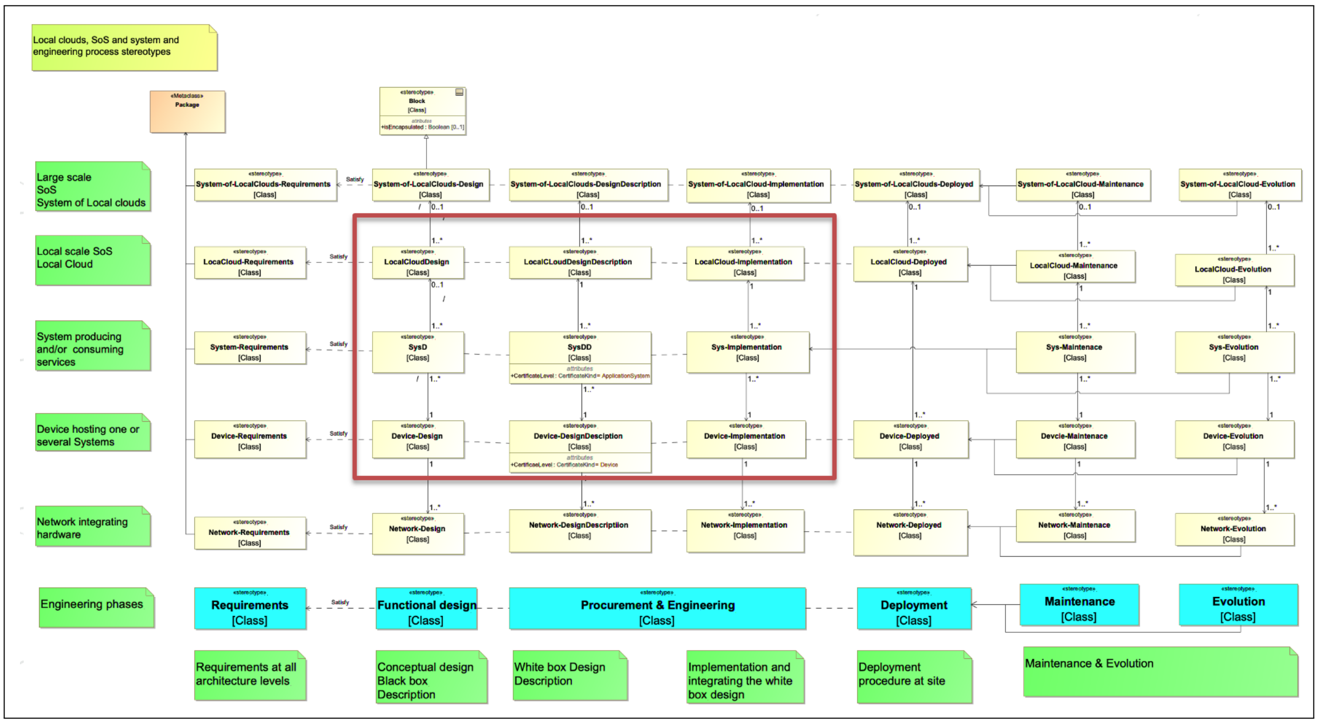

Local clouds, SoS and system and engineering process stereotypes
«Metaclass» Package
«stereotype» Block [Class]
Large scale SoS System of Local clouds
Local scale SoS Local Cloud
System producing and/or consuming services
Device hosting one or several Systems
Network integrating hardware
Engineering phases
System-of-LocalClouds-Requirements
System-of-LocalClouds-Design
System-of-LocalClouds-DesignDescription
System-of-LocalCloud-Implementation
System-of-LocalClouds-Deployed
System-of-LocalCloud-Maintenance
System-of-LocalCloud-Evolution
LocaCloud-Requirements
LocalCloudDesign
LocalCLoudDesignDescription
LocalCloud-Implementation
LocalCloud-Deployed
LocalCloud-Maintenance
LocalCloud-Evolution
System-Requirements
SysD
SysDD
Sys-Implementation
Sys-Maintenace
Sys-Evolution
Device-Requirements
Device-Design
Device-DesignDescription
Device-Implementation
Device-Deployed
Device-Maintenace
Device-Evolution
Network-Requirements
Network-Design
Network-DesignDescription
Network-Implementation
Network-Deployed
Network-Maintenace
Network-Evolution
Satisfy
Requirements [Class]
Functional design [Class]
Procurement & Engineering [Class]
Deployment [Class]
Maintenance [Class]
Evolution [Class]
Requirements at all architecture levels
Conceptual design Black box Description
White box Design Description
Implementation and integrating the white box design
Deployment procedure at site
Maintenance & Evolution

Figure 1: Engineering process matrix. The red square point out the phases where the plugins are used.

The tools have been implemented in the form of plugins to improve the integration between the design and implementation tools. In the majority of the cases due to a lack of compatibility between tools and poor interoperability solutions, the transition between one engineering step and the other is handled by hand. The interoperability between process phases, hence, is also a key factor to reduce the engineering effort and time.

The plugins automatize the implementation of Arrowhead Local clouds. A general definition of a plugin is a Software component that adds a specific feature to an existing computer program. In this specific case, the based program is the Eclipse IDE, they are integrated with the Papyrus plugin and based on the use of SysML models. The main objectives of this work are reducing the engineering time and decreasing the entrance barrier of the framework. Only open-source tools have been considered to increase its adoption.

This document outlines the main functionalities of the three plugins, their installation, use, and further considerations.

2 System Description

This section describes the functionalities and objectives of each of the plugins available. The plugins have been implemented and tested in the most prominent operating systems: Windows, Mac OS, and Ubuntu.

The implementation of the plugins has been done with the common objective of automatizing parts of the engineering process and lowering the entrance barrier of the Arrowhead framework. The description of the system using models is a practice largely used and accepted by the industry. The use of open-source tools such as Papyrus and the Eclipse IDE in conjunction with the Arrowhead profile enables the detailed description of local clouds. The local cloud models not only are meaningful for the design phase but can be used to autonomously generate software artifacts to rapid implement and configure a local cloud. A loca cloud is composed by at least the mandatory core systems, application systems, and orchestration and authorization rules. Therefore plugins have been designed to assist in the generation, configuration and/or execution of the aforementioned three parts of the local clouds.

Once installed, see Section 4, the plugins can be accessed by selecting the menu Arrowhead in the bar menu, the icons on the icon bar, or using key shortcuts. Figures 2 and 3 show the menu and icons in the MAC environment. Menus and icons are the same for all operating systems, as well as functionality and window menus.

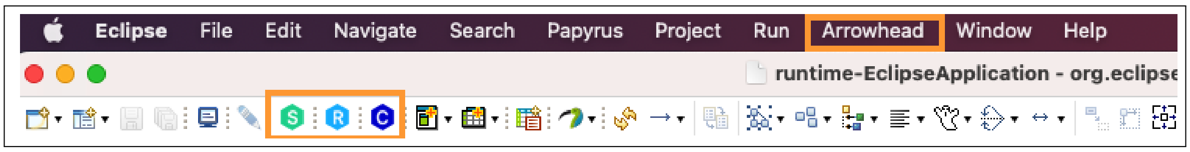


Figure 2: Eclipse IDE menu bar. Arrowhead menu and icons highlighted.

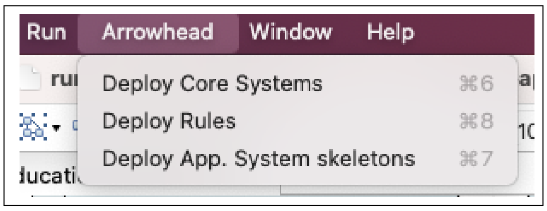


Figure 3: Options available in the Arrowhead menu.

2.1 Core Systems Deployment Plugin

The main objective of this plugin is to provide the last version of the core systems available in the official repository and prepare them for their execution and use. The plugin clone the available system code, builds it for the selected options, and generates the executable and the script to facilitate the execution.

The plugin can be executed by selecting *Arrowhead* >> *Deploy Core Systems* or clicking on the "C" dark blue icon. A window menu will be then opened as shown in Figure 4.

Options The following options need to be configured for the use of the plugin:

- Directory. The directory box needs to be filled with the output directory where the files will be generated. By default, the selected directory is the Eclipse workspace where the plugins are installed.
- Systems. There are two options (1) Mandatory Core Systems, which clone and compile only the three mandatory core systems (Service Registry, Authorization, and Orchestration), and (2) Mandatory and Support Core Systems, which clone and compile all available core systems at that moment of time in the official Eclipse Arrowhead GitHub repository. The list of systems includes the three mandatory systems and all support systems released.

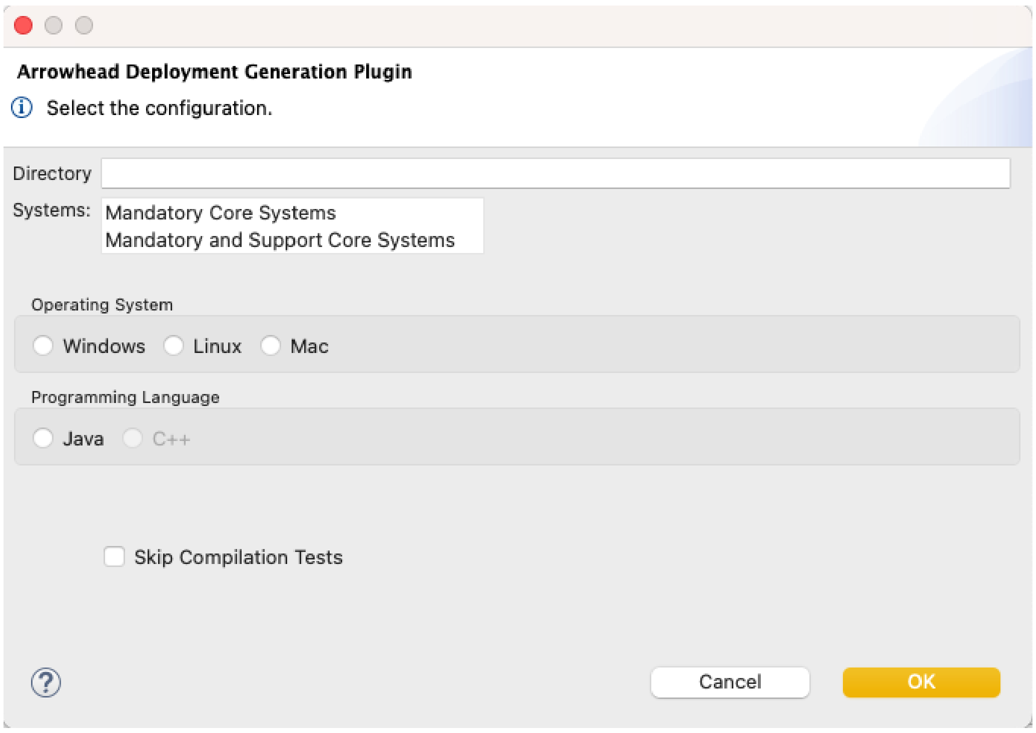
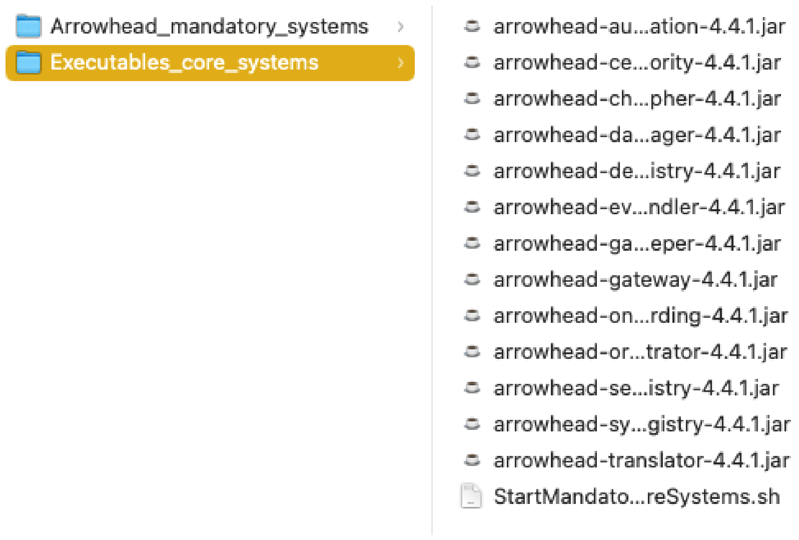

Arrowhead Deployment Generation Plugin
Select the configuration.
Directory
Systems: Mandatory Core Systems
Mandatory and Support Core Systems
Operating System
Windows Linux Mac
Programming Language
Java C++
Skip Compilation Tests

Figure 4: The configuration window of the core systems deployment plugin.

- Operating System. The user can select between three operating systems (Windows, Linux and Mac). The scripts and instructions generated will depend on the operating system selected.
- Programming Language. The core systems may be available in different implementations making use of different programming languages. Only the versions realized are active for selection.
- Skip Compilation Tests. This option can be selected to skip the compilation test and save compilation time. It is only recommended in case of demonstration purposes or rapid tests.

Generated Files The plugin creates two folders in the output directory. In the folder Arrowhead_mandatory_systems, the core-java-spring repository is cloned from the official Eclipse Arrowhead GitHub. In the second folder, Executables_core_systems the executable (JAR file in this case) of each core system is ready for its use. A script to launch the three mandatory core systems is also generated to facilitate their use.


Arrowhead_mandatory_systems
Executables_core_systems

arrowhead-au...ation-4.4.1.jar
arrowhead-ce...ority-4.4.1.jar
arrowhead-ch...pher-4.4.1.jar
arrowhead-da...ager-4.4.1.jar
arrowhead-de...istry-4.4.1.jar
arrowhead-ev...ndler-4.4.1.jar
arrowhead-ga...eper-4.4.1.jar
arrowhead-gateway-4.4.1.jar
arrowhead-on...rding-4.4.1.jar
arrowhead-or...trator-4.4.1.jar
arrowhead-se...istry-4.4.1.jar
arrowhead-sy...gistry-4.4.1.jar
arrowhead-translator-4.4.1.jar
StartMandato...reSystems.sh

Figure 5: Generated folders by the Core Systems plugin including the executable JARs of all core systems available at that moment.

2.2 Application System Skeletons Deployment Plugin

The main goal of this plugin is to generate the code for the application systems modeled in the local cloud. From the model, the information about the interfaces (Interface Design Description and the System Design Description) is captured and used to generate the maximum code possible. The business logic is not captured in the model, therefore, the generated project will be a skeleton that requires further manual implementation. The plugin helps to set the project structure, pom, and Arrowhead compliant interfaces.

The plugin can be executed by selecting *Arrowhead >> Deploy App. System Skeletons* or clicking on the "S" turquoise icon. After selecting the model, a window menu will be then opened as shown in Figure 6.

Options The following options need to be configured for the use of the plugin:

- Directory. The directory box needs to be filled with the output directory where the files will be generated. By default, the selected directory is the Eclipse workspace where the plugins are installed.
- Name. Name given to the generated output. This name allows to generate and differentiate various projects for the same model.
- Local Cloud. List of local clouds available for deployment in the project.
- Systems. When selecting a local cloud a list of application systems available in the cloud appears. A project folder for each system selected is generated. The text box allows multiple selection.
- Programming Language. The core systems may be available in different implementations making use of different programming languages. Only the versions realized are active for selection.
- Operating System. The user can select between three operating systems (Windows, Linux and Mac). The scripts and instructions generated will depend on the operating system selected.

Generated Files A new folder with the selected name is created each time that the plugin is used. The folder follows a project structure, including a folder for each application system and a parent *pom* file with the common dependencies, Fig. 7. In the case of using the systems on different devices, the common dependencies need to be copied to the individual *pom* files.

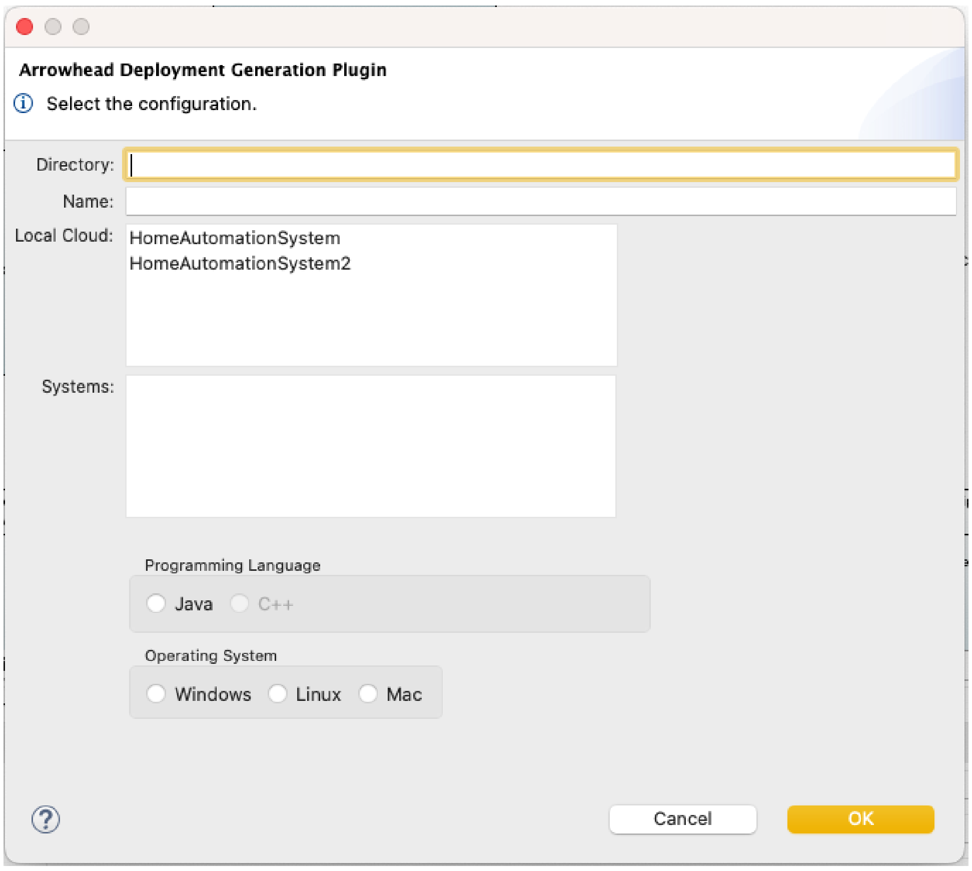


Figure 6: Configuration window of the Application System Skeletons Deployment Plugin.

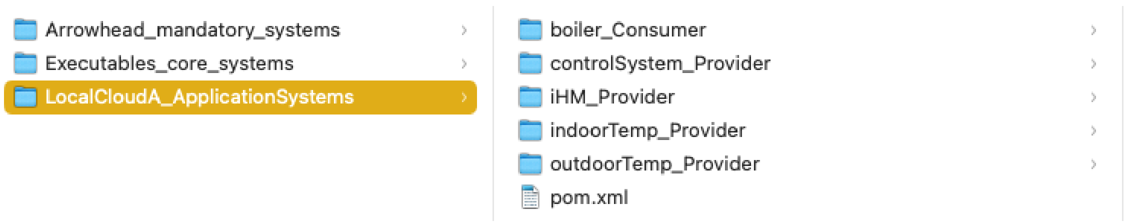


Figure 7: Example of a Local cloud folder generated by the plugin.

Each system folder also follows the java project structure similar to the skeletons and examples available on the official Arrowhead GitHub repository. The folder name is formed by the system name and the type of system (consumer or provider). For both types of systems, the folder includes a *pom* file that links the project to the parent *pom* and adds specific dependencies required by the system. The folder structure is common for both types (src/main/java and resources). The resources folder, Fig. 8, includes the properties file already configured and the certificates folder. Certificates are not generated in the current version, therefore, examples of certificates are provided.

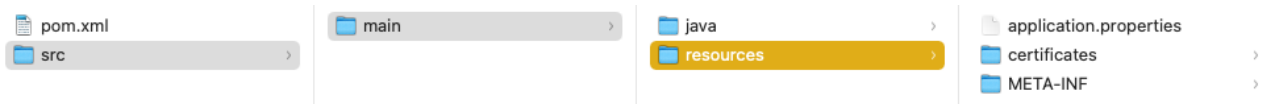


Figure 8: Resources folder generated by the plugin.

One of the differences between consumer and provider projects is the security folder included in the provider project, which contains all the arrowhead compliant classes to manage tokens and certificates, Fig. 9.

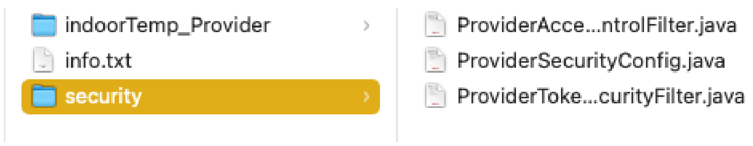

indoorTemp_Provider
info.txt
security
ProviderAcce...ntrolFilter.java
ProviderSecurityConfig.java
ProviderToke...curityFilter.java

Figure 9: Security folder generated by the plugin.

There are four types of java classes generated as explained in Table 1. All the classes are generated based on the IDD[1] properties described in the modeled local cloud. The code is tailored to the model to facilitate the use of the framework and speed up the implementation phase. Examples of generated classes can be seen in Fig 10 for the consumer and Fig 11 for the provider.

Type	Definition	Generation
Main	Main class of the project	(Consumer) Includes the interfaces to communicate with the orchestrator and each service that consumes. The protocol, encoding, operation type, names, and data types described in the model are used to generate specific interfaces.
Application Listener	Java Spring configuration class	(Provider) Includes the methods (already configured) to register the services in the Service Registry.
DTO Data Transfer Object	Simple java object class with the payload elements	For each payload, a java object (POJO) is created for data exchange and facilitating further use.
Controller	Class that includes the services offered	(Exclusive for Providers) For each provided service, a method is generated to provide a service according to the IDD described in the model.

Table 1: Types classes generated for the plugin

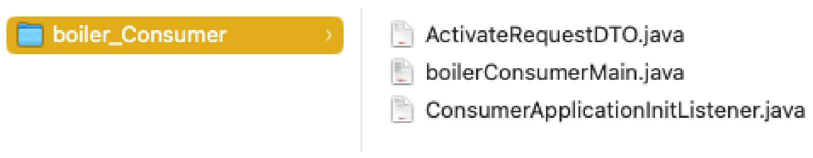

boiler_Consumer
ActivateRequestDTO.java
boilerConsumerMain.java
ConsumerApplicationInitListener.java

Figure 10: Example of the classes generated for a given consumer.

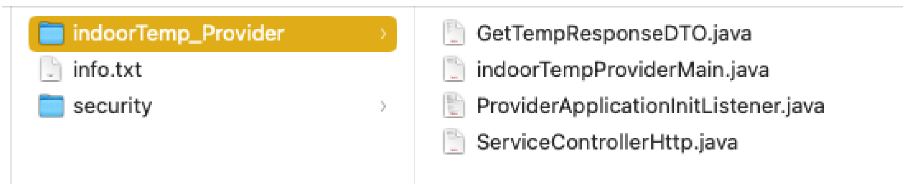

indoorTemp_Provider
info.txt
security
GetTempResponseDTO.java
indoorTempProviderMain.java
ProviderApplicationInitListener.java
ServiceControllerHttp.java

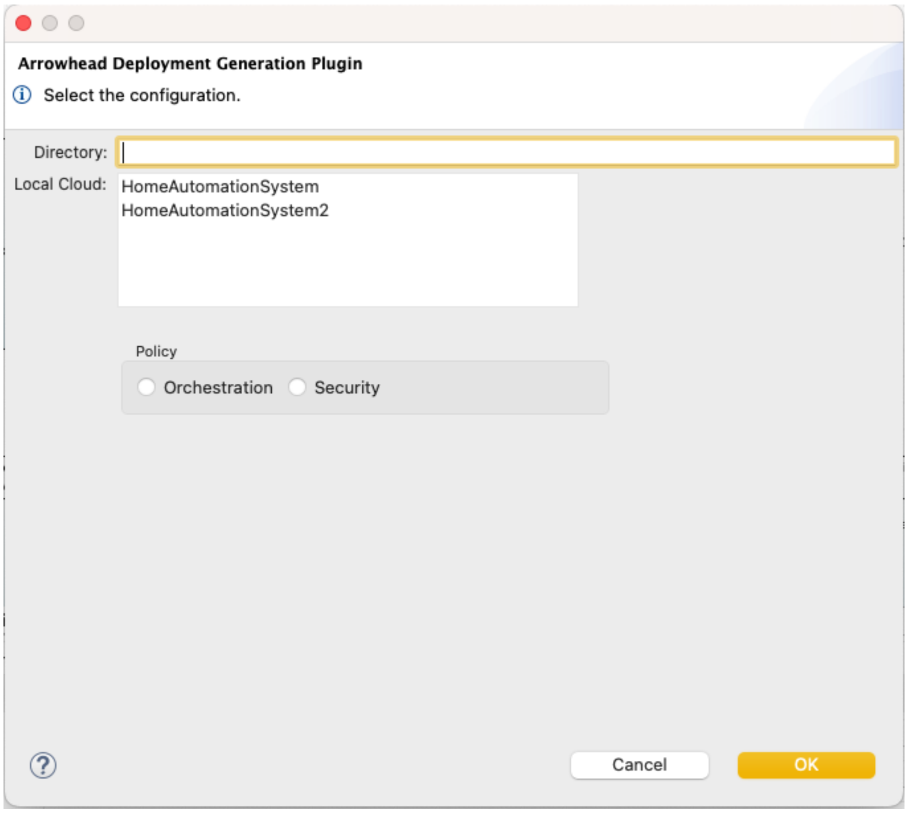
Figure 11: Example of the classes generated for a given provider.

2.3 Rules Deployment Plugin

The main objective of the plugin is to generate the database scripts necessary to populate the database with the orchestration and authorization rules based on the connections modeled between systems.

The plugin can be executed by selecting *Arrowhead* >> *Deploy Rules* or clicking on the "R" blue icon. After selecting the model, a window menu will be then opened as shown in Figure 12.

[1] Interface Design Description, see Arrowhead documentation for more information


Arrowhead Deployment Generation Plugin
Select the configuration.

Directory:
Local Cloud: HomeAutomationSystem
HomeAutomationSystem2

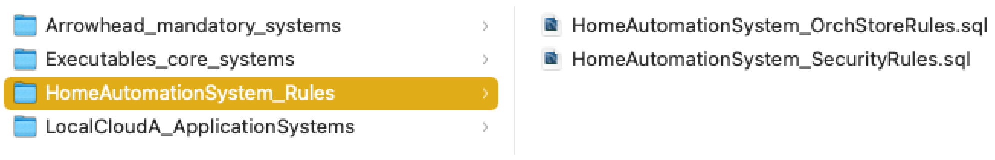
Policy
Orchestration Security

Figure 12: The configuration window of Rule Deployment Plugin

Options The following options need to be configured for the use of the plugin:

- Directory. The directory box needs to be filled with the output directory where the files will be generated. By default, the selected directory is the Eclipse workspace where the plugins are installed.
- Local Cloud. List of local clouds available for deployment in the project.
- Policy. Two types of rules can be generated, orchestration and security rules. Only one type can be selected each time.

Generated Files A new folder is generated, its name is based on the local cloud identifier. A file for each type of policy can be created. All connections in the local cloud are converted into rules. The output files are SQL scripts, compatibles with SQL-based databases. In the case that only some rules want to the added to the database, the selection needs to be done manually and it cannot be generated separately.


Arrowhead_mandatory_systems
Executables_core_systems
HomeAutomationSystem_Rules
LocalCloudA_ApplicationSystems

HomeAutomationSystem_OrchStoreRules.sql
HomeAutomationSystem_SecurityRules.sql

Figure 13: The output folder and orchestration and security rule files generated by the plugin.

3 System Requirements

In order to install the plugins, the following system requirements need to fulfilled.

- JDK 11 or superior on the computer
- Eclipse IDE for Eclipse committers. Version: 2021-12 (4.22.0) or superior[2]
- JRE 17 installed on the Eclipse IDE enviroment.
- Papyrus SysML 1.6 2.20 plugin installed on the Eclipse IDE enviroment (see marketplace in the menu *help >> Eclipse Marketplace*)

4 Installation

The following sub-sequence of steps is required to use and test the plugins. Any variation can lead to an error.

STEP 1. Eclipse IDE configuration The first step is to configure the workspace folder for the Eclipse IDE. You can choose the one by default or select a new folder when first you open the program. The second option is recommended. The selected folder will be used for the plugin installation and use.

STEP 2. Copy the plugins files and the Arrowhead profile Open the folder that you are using as a workspace and copy the zip file a config folder from the https://github.com/CristinaPaniagua/ModelstoCode repository. **Do not unzip the file.**

- models2codeUpdateSite.zip
- config

In addition, clone the repository https://github.com/eclipse-arrowhead/profile-library-sysml in the workspace folder. The workspace folder should look as shown in Fig. 5.

STEP 3. Plugin Installation from Local File To install the plugins select the menu *Help >> Install New Software ...*, Fig. 14.

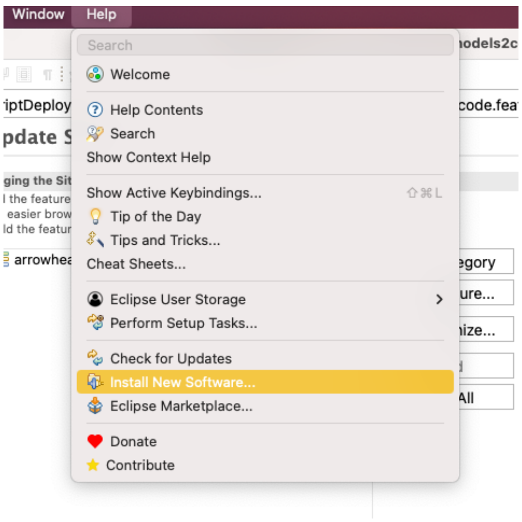


Figure 14

Select Add and Archive. Select the zip file available in the eclipse workspace, cloned from the repository in Step 2. Write a name for the plugin installation and continue with the installation.

[2]The plugins have not been tested in other versions of the IDE. Plugins may work in other versions; in case you try in other versions report the results of the trial to cristina.paniagua@ltu.se

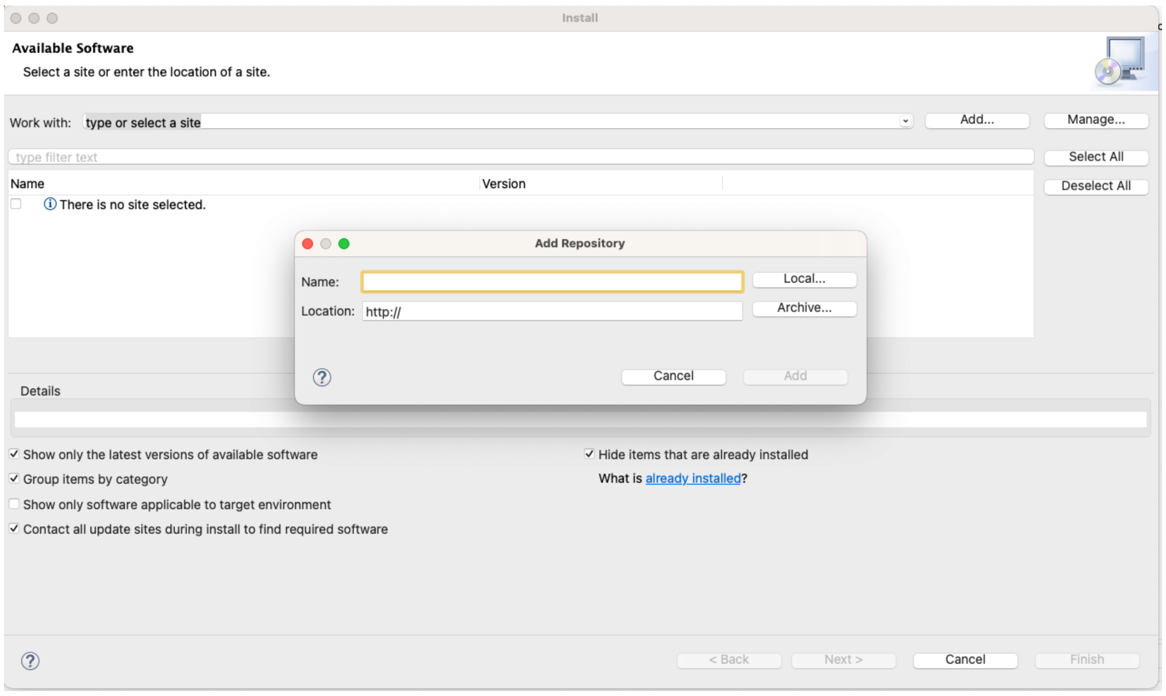
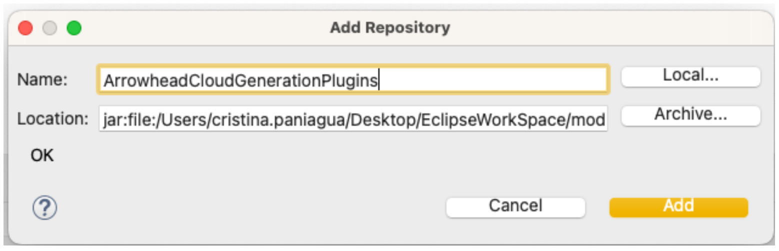

Install
Available Software
Select a site or enter the location of a site.
Work with: type or select a site
Add...
Manage...
type filter text
Select All
Deselect All
Name
Version
There is no site selected.
Add Repository
Name:
Local...
Location: http://
Archive...
Cancel
Add
Details
Show only the latest versions of available software
Hide items that are already installed
Group items by category
What is already installed?
Show only software applicable to target environment
Contact all update sites during install to find required software
< Back
Next >
Cancel
Finish

Figure 15



Name: ArrowheadCloudGenerationPlugins

Location: jar:file:/Users/cristina.paniagua/Desktop/EclipseWorkSpace/mod

Figure 16

STEP 4. Open or create your own model and start using the plugins The plugins are already ready to use, but in order to try them you need to have a local cloud modeled. If you want to try the plugins you can open the folder *[yourWorkspace]/profile-library-sysml/distribution/papyrusPlugins/examples/-org.eclipse.papyrus.arrowhead.example* and follow the instructions in the following section.

5 How to use

The following section shows an example of how to use the plugins. The model used in the example is available in the profile-library-sysml repository.

In order to use the plugins, the configured Eclipse application window must be opened. If the new window is not open the icons and Arrowhead menu will not be available. STEP 5 in the previous section shows how to open the Eclipse application window.

The first step to use the plugins is to have a model of a local cloud completely configured. The next section describes what are the elements need to have a compliant model with the plugins. The local cloud can be modeled using the arrowhead profile. In this example the selected model is the studio4education project that can be found in the folder profile-library-sysml/distribution/papyrusPlugins/examples/org.eclipse.papyrus.arrowhead.example /resources/studio4education.

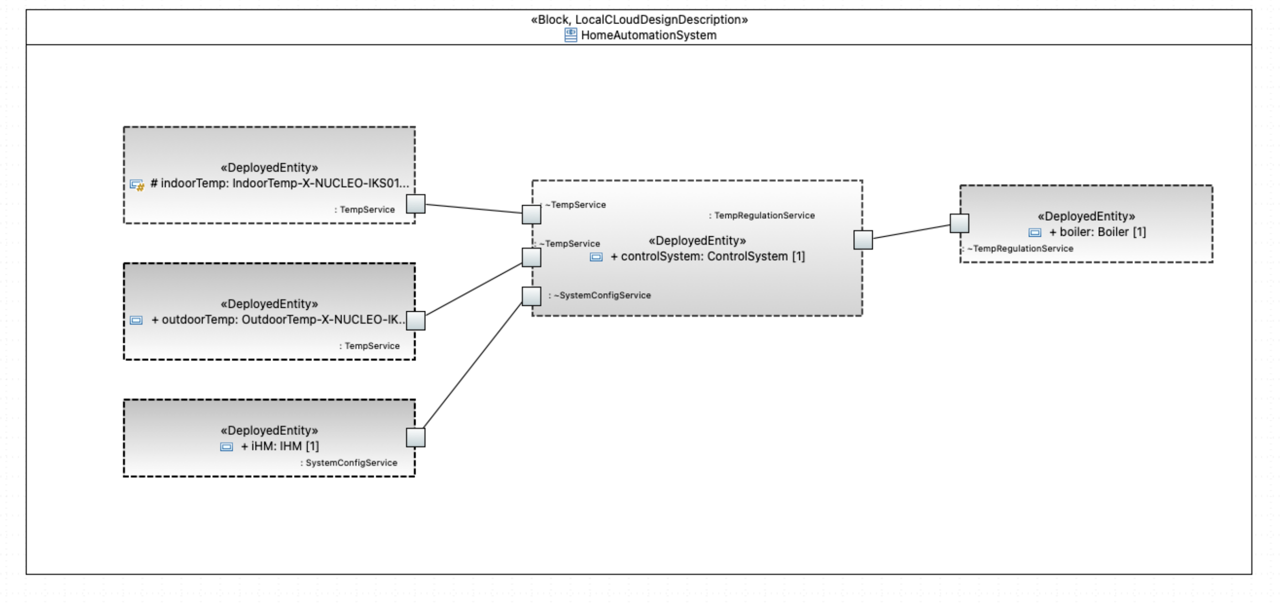


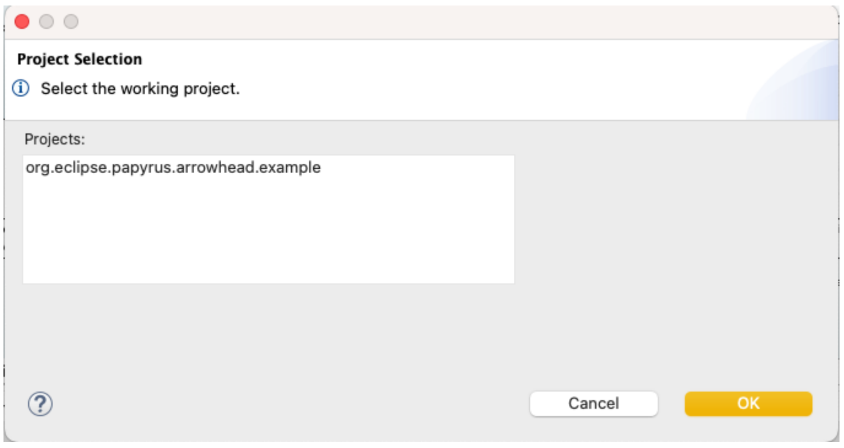
Figure 17: Local cloud model.

Once open, it is possible to visualize the model using the papyrus perspective (window>>perspective>>open perspective>>other...>>papyrus)

The plugins can be used in any order and as many times as wanted for the same model. They only require to open the Arrowhead menu in the upper bar and select the plugin or use the icons in the icon bar. Changing the options will result in different outputs.

Application System Skeletons Deployment Plugin does not require any model to be run. It only requires selecting an output directory, the type of core systems generated, and the operating system. Application System Skeletons Deployment Plugin and Rules Deployment Plugin require opening a model in the eclipse application window.

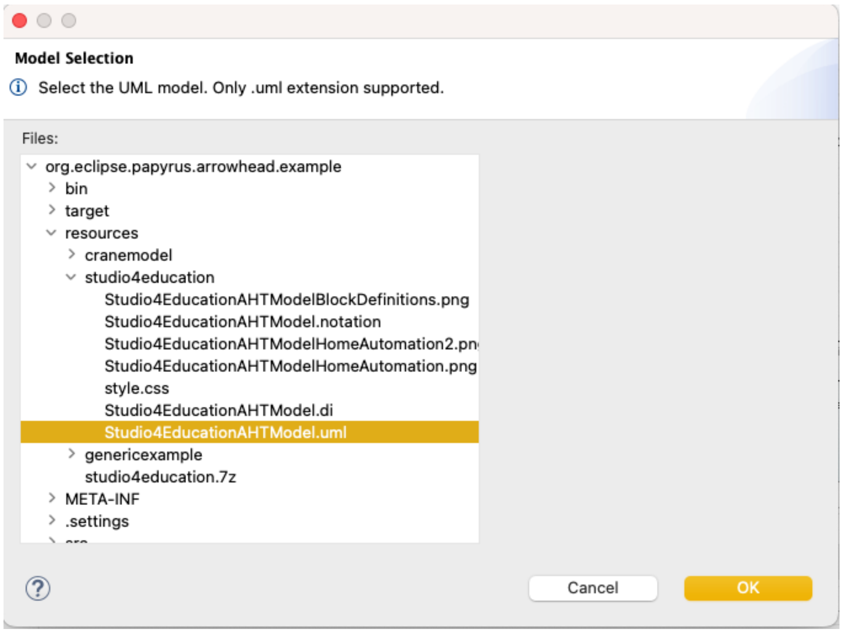
Once open the folder containing the project, when selecting any of the two aforementioned plugins a window to select the project appears, Fig. 18. In that window, all open projects are listed, independently if they contain SysML models or not.


Project Selection
Select the working project.

Projects:
org.eclipse.papyrus.arrowhead.example

Figure 18: Window to select the project.

Once selected the project, the next window allows selecting the specific model, Fig. 20. The text box shows a folder tree that can be navigated until finding the model. The selected file extension must be .uml. If not, the file is not accepted and an error message will appear.


Model Selection

Select the UML model. Only .uml extension supported.

Files:

```
org.eclipse.papyrus.arrowhead.example
  bin
  target
  resources
    cranemodel
    studio4education
      Studio4EducationAHTModelBlockDefinitions.png
      Studio4EducationAHTModel.notation
      Studio4EducationAHTModelHomeAutomation2.pn
      Studio4EducationAHTModelHomeAutomation.png
      style.css
      Studio4EducationAHTModel.di
      Studio4EducationAHTModel.uml
    genericexample
    studio4education.7z
  META-INF
  .settings
```

Figure 19: Window to select the model. Only .uml files are accepted.

Once selected the model, the configuration window appears which the options explained in Section 7. Once selected the desired options, press ok. The code is generated in the output directory. In case the output directory does not exist an error message will appear and the plugin will be closed. It can be selected again and without re-start the eclipse application window.

6 Limitations

This section describes the technical limitations of the plugins for the stated version. The limitations will be reviewed and considered for future versions.

Programming language The only available programming languages on the current version is Java. The future developments of the framework are intented to expand the number of programming languages. The plugins will be adapted in future releases to mirrot the supported options by the framework.

Protocol and encoding generation The generation of application system skeletons is currently limited to HTTP and COAP as communication protocols and XML, JSON and CBOR as encodings.

Business logic The behabiour and bussines logic is not pictured in the model, therefore, the application system skeleton plugin only generate interfaces. The bussines logic is not included in the generation.

7 Profile use and model considerations

The use of the Arrowhead profile is imperative to use the plugins. The profile includes several stereotypes and options. The following list of profile stereotypes is critical for the correct functionality of the plugins. Consequently, the wrong configuration or lack of them will lead to a fatal error.

- Blocks with the SysDD stereotype applied and configured. The block also must include the UML type equal to the service described in the Interface block.

- Interface blocks with the InterfaceDesignDescription stereotype applied and configured. That block need to include the operations for the described service.
- Operations with the operation stereotype applied and configured (e.g httpOperation).
- Connection between the services (ports) to provide the orchestration and authorization rules.

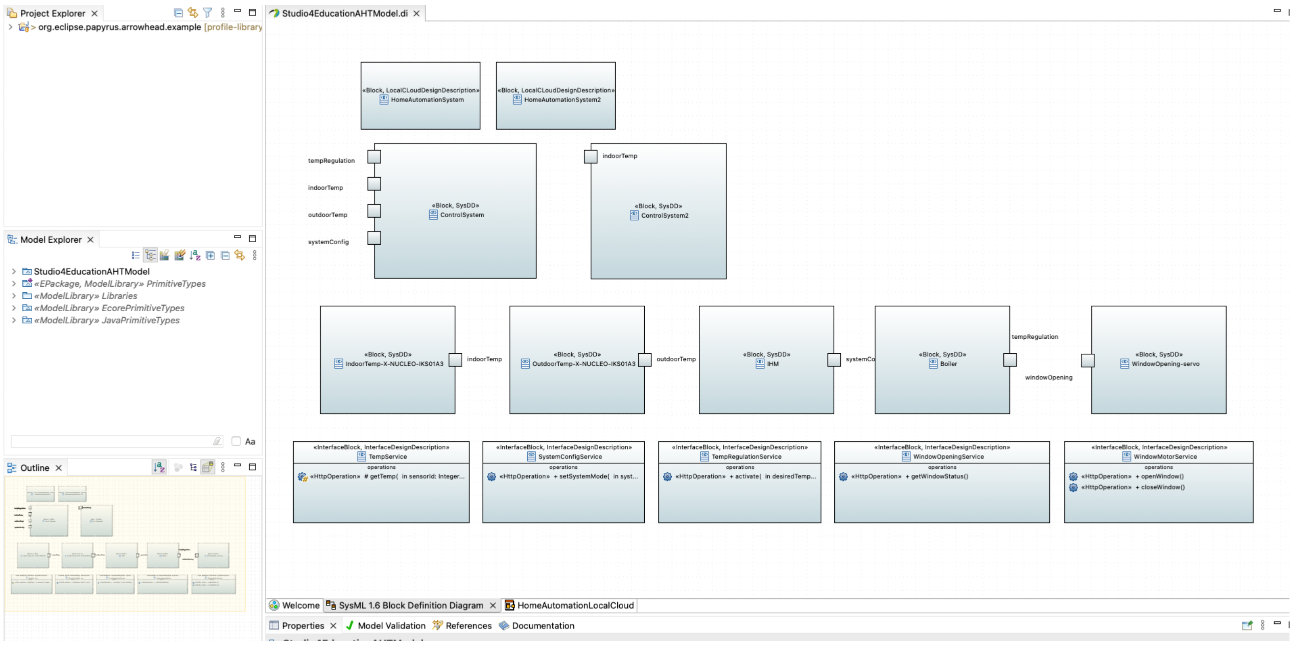

Project Explorer
org.eclipse.papyrus.arrowhead.example [profile-library
Model Explorer
Studio4EducationAHTModel
«EPackage, ModelLibrary» PrimitiveTypes
«ModelLibrary» Libraries
«ModelLibrary» EcorePrimitiveTypes
«ModelLibrary» JavaPrimitiveTypes
Outline
Studio4EducationAHTModel.di
«Block, LocalCLoudDesignDescription» HomeAutomationSystem
«Block, LocalCLoudDesignDescription» HomeAutomationSystem2
tempRegulation
indoorTemp
outdoorTemp
systemConfig
«Block, SysDD» ControlSystem
indoorTemp
«Block, SysDD» ControlSystem2
«Block, SysDD» IndoorTemp-X-NUCLEO-IKS01A3
indoorTemp
«Block, SysDD» OutdoorTemp-X-NUCLEO-IKS01A3
outdoorTemp
«Block, SysDD» IHM
systemCo
«Block, SysDD» Boiler
tempRegulation
windowOpening
«Block, SysDD» WindowOpening-servo
«InterfaceBlock, InterfaceDesignDescription» TempService
operations
«HttpOperation» # getTemp(in sensorId: Integer...
«InterfaceBlock, InterfaceDesignDescription» SystemConfigService
operations
«HttpOperation» + setSystemMode(in syst...
«InterfaceBlock, InterfaceDesignDescription» TempRegulationService
operations
«HttpOperation» + activate(in desiredTemp...
«InterfaceBlock, InterfaceDesignDescription» WindowOpeningService
operations
«HttpOperation» + getWindowStatus()
«InterfaceBlock, InterfaceDesignDescription» WindowMotorService
operations
«HttpOperation» + openWindow()
«HttpOperation» + closeWindow()
Welcome
SysML 1.6 Block Definition Diagram
HomeAutomationLocalCloud
Properties
Model Validation
References
Documentation

Figure 20: SysML blocks with the Arrowhead stereotypes used to describe the local cloud

NOTE: The listed elements are critical for the plugins but not sufficient. The model of the local cloud needs to be complete and accurate. The use of the profile is outside of the scope of this document.

8 Future work and future releases

This section summarizes the planned activities with regard to the described plugins. The current version of the plugins is not available on the official repository of Eclipse Arrowhead, but it will be available in an official release soon. All files will be moved then. In the meantime, the beta version of the plugins is available for trial and bug reports.

At the moment to write his document the version beta 1.0 of the plugins does not consider the device and hardware characteristics. Future versions of the plugins will optimize and configure the generated code according to the device selected in the model for the deployment phase and their hardware constraints. They will also include other programming languages, encoding and protocols.

In future versions, the use of an Eclipse configuration will not be necessary. We are working to make the plugins ready to install directly in the Eclipse IDE.

Future plans also include the development of a verification plugin to check the validity of the model before the use of the plugins. As well as, new plugins to generate interface description documents or other type of arrowhead documentation.

9 Revision History

9.1 Amendments

No.	Date	Version	Subject of Amendments	Author
1				

9.2 Quality Assurance

No.	Date	Version	Approved by
1			