Internship Report

Ang LI

ang.li@ensta-bretagne.org

Tel: +33664885681

Abstract

My internship project achieved the modeling tool named SETool which is focus on helping the students realize the small projects quickly. This is a tool about system engineering. It is helpful in the initial phase of system engineering process.

The tool must be lighter and easier to use in the domain of small projects than other modeling tools. It includes less meta-model classes than SysML and EFFBD. And It is easy to learn well for the students. It can also realize requirement analysis and functional architecture.

The project includes four main parts: define our own Domain Specific Language by meta modeling, design SETool by Sirius, create a case by using SETool and generate the source code and documents of this case.

Keyword : System Engineering, Domain Specific Language, Meta-model, Sirius, Acceleo

Acknowledgements

I would like to extend my sincere gratitude to Mr Champeau for his understanding, encouragement and help during my internship.

Table of contents

Introduction

This project achieved the modeling tool named SETool which is focus on helping the students realize the small projects quickly. I start this project in early May, and it will end in mid September.

The project includes four main parts: define our own Domain Specific Language by meta modeling, design SETool by Sirius, create a case by using SETool and generate the source code of this case.

1. Context

1.1 System engineering

1.1.1 Definition and classical uses

Systems engineering is an interdisciplinary field of engineering and engineering management that focuses on how to design and manage complex systems over their life cycles. The systems engineering process is a discovery process that is quite unlike a manufacturing process. A manufacturing process is focused on repetitive activities that achieve high quality outputs with minimum cost and time. The systems engineering process must begin by discovering the real problems that need to be resolved, and identify the most probable or highest impact failures that can occur-systems engineering involves finding elegant solutions to these problems. At its core systems engineering utilizes systems thinking principles to organize this body of knowledge. Issues such as requirements engineering, reliability, logistics, coordination of different teams, testing and evaluation, maintainability and many other disciplines necessary for successful system development, design, implementation, and ultimate decommission become more difficult when dealing with large or complex projects. Systems engineering deals with work-processes, optimization methods, and risk management tools in such projects. It overlaps technical and human-centered disciplines such as industrial engineering, mechanical engineering, manufacturing engineering, control engineering, software engineering, electrical engineering, cybernetics, organizational studies, engineering management and project management. Systems engineering ensures that all likely aspects of a project or system are considered, and integrated into a whole.

1.1.2 SE for small projects

Some references on SE for SMEs

System engineering includes a lot of useful modeling formalisms and graphical representations. Initially, when the primary purpose of a systems engineer is to comprehend a complex problem, graphic representations of a system are used to communicate a system's functional and data requirements. The common graphical representations include Functional flow block diagram (FFBD), Model-based design, Data Flow Diagram (DFD), N2 Chart, IDEF0 Diagram, Use case diagram, Sequence diagram, Block diagram, Signal-flow graph, USL Function Maps and Type Maps, Enterprise Architecture frameworks, Model-based systems engineering,etc.

A graphical representation relates the various subsystems or parts of a system through functions, data, or interfaces. Any or each of the above methods are used in an industry based on its requirements. For instance, the N2 chart may be used where interfaces between systems is important. Part of the design phase is to create structural and behavioral models of the system.

There are some popular modeling languages. UML(Unified modeling language) is the most common in the world. Others like Capella which is programmed by Java and DOORS are also used in specific domains. Systems Modeling Language (SysML) is a modeling language used for systems engineering applications, supports the specification, analysis, design, verification and validation of a broad range of complex systems. SysML includes about 250 classes in meta-model. Lifecycle Modeling Language (LML), is an open-standard modeling language designed for systems engineering that supports the full lifecycle: conceptual, utilization, support and retirement stages.

In our case students projects

In the students projects, the most frequently used modeling language is UML. UML includes some structure diagrams, behavior diagrams and interaction diagrams. These diagrams all focus on functional architecture. UML can not support a diagram which can display the relation between requirement and function. SysML achieves this, but it’s too heavy for a small student project.

Moreover, students need to use some complex and expensive software like Visio and Rhapsody to finish the requirement analysis and functional architecture. The two software are not easy to learn well for the beginners, especially Rhapsody. So the students need a simple and small tool to finish the initial phase of their projects.

The project’s focus

My Setool project is focus on create a light and useful tool to help the students finish quickly the requirement analysis and functional architecture of their projects. And they can get automatically the source code after they finish these models.

I define these steps to develop my project:

- Create my own Domain Specific Language by meta-modeling
- Design a modeling tool by Sirius
- Create a example model to test the tool
- Generate the source code for the model

2. Current approaches and technologies

2.1 SysML

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

SysML was originally developed by an open source specification project, and includes an open source license for distribution and use. SysML is defined as an extension of a subset of the Unified Modeling Language (UML) using UML's profile mechanism. SysML includes 9 types of diagram, some of which are taken from UML. They are Block definition diagram, Internal block diagram, Package diagram, Use case diagram, Requirements diagram, Activity diagram, Sequence diagram, State machine diagram, Parametric diagram.

SysML has some limitations and criticisms. SysML takes time and effort to learn. It is intended to be a conventional language for conducting model-based systems engineering but it has some important limitations that reduce its efficiency and limit its acceptance. SysML has been criticized for being incomplete although there is little to stop a practitioner from adding their own content. In SysML, there is no provision for several diagrams and graphically-oriented tools that are commonly used in system engineering. Missing elements include functional block diagram, N2 chart , House of Quality, Ishikawa diagram (fishbone), parameter diagram and others. The diagrams generated by SysML are complicated and some are difficult to understand by people that are unfamiliar with the language. System-engineering diagrams are primarily intended for other members of an embedded team; people outside the team, more often than not, are not system engineers and are less likely to know SysML. This can be overcome by including explanatory notes and legends in SysML diagrams to ease their interpretation. When drawn in a software tool, the diagrams that respect the rules of SysML often include redundant pieces of model information that can impair their interpretation.

2.2 Capella

Capella is an Open Source solution hosted at polarsys.org. The solution provides a process and tooling for graphical modeling of systems, hardware or software architectures, in accordance with the principles and recommendations defined by the Arcadia method. Capella is an initiative of PolarSys, one of several Eclipse Foundation working groups. Capella is mainly used for modeling complex and safety-critical systems in embedded systems development for industries such as aerospace, avionics, transportation, space, communications and security and automotive.

2.3 EFFBD

An Enhanced Functional Flow Block Diagram (EFFBD) is a multi-tier, time-sequenced, step-by-step flow diagram of a system’s functional flow. The term "functional" in this context is different from its use in functional programming or in mathematics, where pairing "functional" with "flow" would be ambiguous. Here, "functional flow" pertains to the sequencing of operations, with "flow" arrows expressing dependence on the success of prior operations. EFFBDs may also express input and output data dependencies between functional blocks, as shown in figures below, but EFFBDs primarily focus on sequencing. EFFBDs can be developed in a series of levels. EFFBDs show the same tasks identified through functional decomposition and display them in their logical, sequential relationship.

3. SETool

Approach and Solution

In the project, i chose DSL(Domain Specific Language) to realize the basic definition of my modeling tool on the level of meta model. DSL is a computer programming language of limited expressiveness focused on a particular domain. Domain-specific languages allow solutions to be expressed in the idiom and at the level of abstraction of the problem domain. The idea is that domain experts themselves may understand, validate, modify, and often even develop domain-specific language programs. Domain-specific languages allow validation at the domain level. As long as the language constructs are safe any sentence written with them can be considered safe. Domain-specific languages can help to shift the development of business information systems from traditional software developers to the typically larger group of domain-experts who (despite having less technical expertise) have deeper knowledge of the domain. DSL is a means of improving productivity to the developers. And more importantly, DSL establishes a bridge between the developer and domain experts, so that developers can quickly build a software to meet customer needs.

My choices of development technologies are Sirius and Acceleo.

Sirius is an open-source software project of the Eclipse Foundation. It allows to create custom graphical modeling workbenches by leveraging the Eclipse Modeling technologies, including EMF and GMF. Sirius is compatible with any tool that can produce EMF compatible models. The modeling workbench created is composed of a set of Eclipse editors (diagrams, tables and trees) which allow users to create, edit and visualize EMF models. Sirius is mainly used to design complex industrial systems or IT applications. The first use case was Capella, which is a Systems Engineering workbench contributed to the Eclipse Working Group PolarSys in 2014 by Thales.

Acceleo is an open-source code generator from the Eclipse Foundation that allows people to use a model-driven approach to building applications. It is an implementation of the "MOFM2T" standard, from the Object Management Group (OMG), for performing model-to-text transformation. Acceleo is written in Java and is deployed as a plugin in the Eclipse IDE. Acceleo is supported on Java 5+ based environments. Acceleo provides tools for code generation from EMF based models. Thanks to those tools, Acceleo allows, for example, incremental generation. Incremental generation gives people the ability to generate a piece of code and then modify the generated code and finally regenerating the code once again without losing the previous modifications. Acceleo also allows code generation from any kind of meta-model compatible with EMF like UML and custom meta-models(DSLs), customization of the generation with user defined templates and generation of any textual language (C, Java, Python, etc.).

4.Description developments

4.1 DSL definition

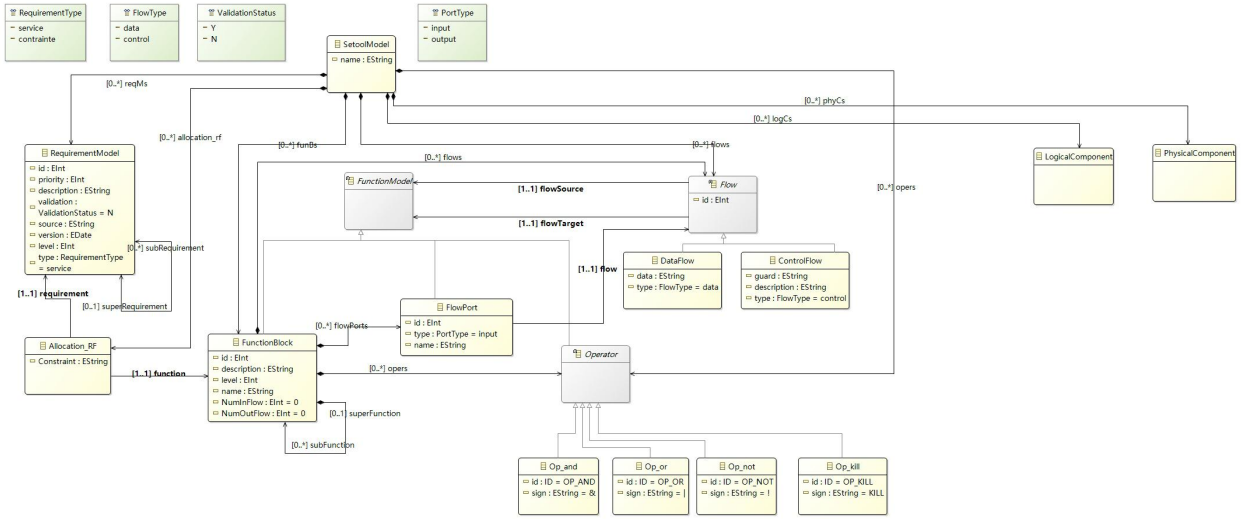


Figure 4.1

The first step in my project is to achieve the definition of our own domain-specific language by meta-modeling. According to the previous analysis of the requirements and the figure 4.1, our DSL named “SetoolModel” includes the necessary elements: RequirementModel, FunctionBlock, Flow, Operator, logicalComponent, PysicalComponent and other sub components which serve these components.

4.1.1 Requirement meta-model

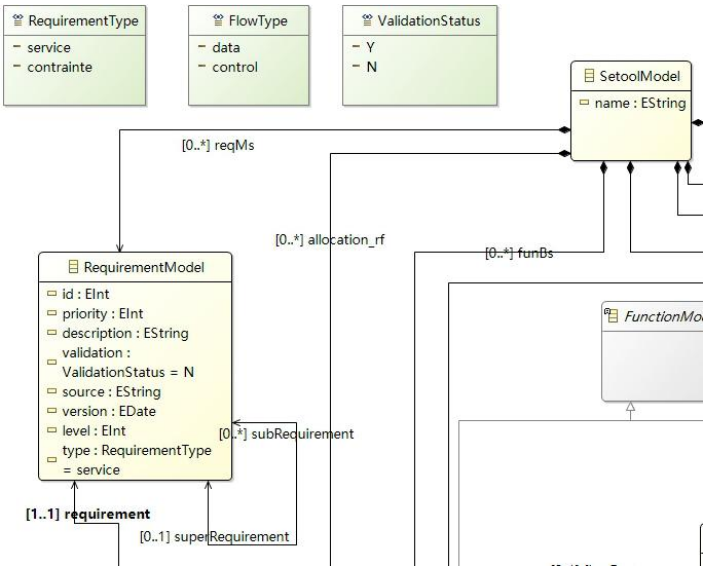


Figure 4.1.1

RequirementModel on the left side of Figure 4.1 represent the definitions of the attributes and associations in requirement module. In a SetoolModel module, it can contain zero or more requirement blocks which are named “reqMs”. There is able to be an association which is similar to single inheritance between two requirements. In my definition, one superRequirement has zero or more subRequirements and one subRequirement has zero or one SuperRequirement. According to many projects’ experience, this definition is correct. For example, a requirement “Send message” can includes two requirements “send message by e-mail” and ”send message by SMS”, but “send message by e-mail” is just included in “Send message”.

In a requirement model, a requirement has a unique integer variable “id” to identify, a integer variable “priority” to define the priority, a string variable “description” to express the requirement content, a custom variable “validation” to determine whether the requirement is realized by function, a string variable “source” to express the source of requirement, a Date variable “version” to express the modified date,a integer variable “level” to express which level of sub requirement the requirement is in, a custom variable “type” to determine that the requirement is a service or a constraint.

4.1.2 Allocation meta-model

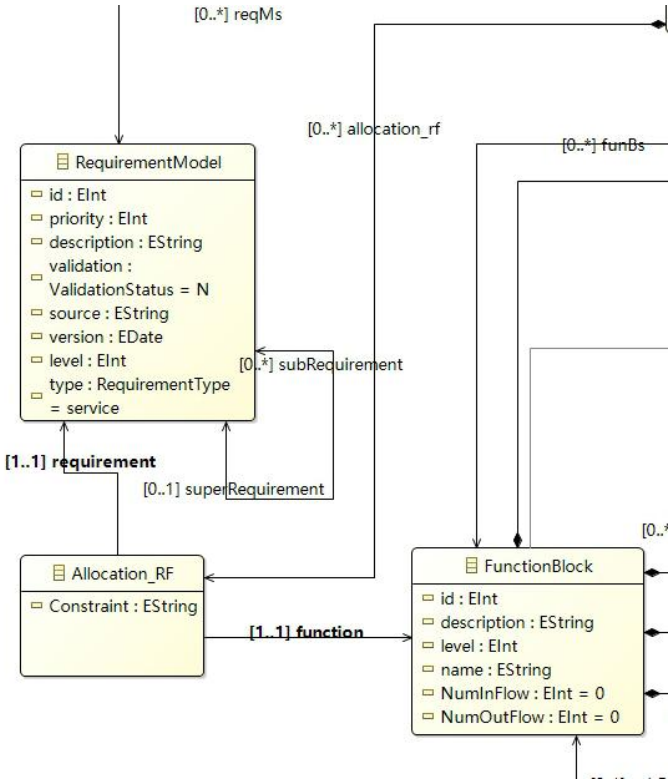


Figure 4.1.2

Allocation_RF define the special association between requirement module and function module. In a Setoolmodel module, it can contain zero or more Allocation_RF associations which are named “association_rf”. In my definition, one Allocation_RF association connect a requirement block and a function block together to express

that a function realize a requirement.

In a Allocation_RF association, it has a a string variable “constraint” to express the content of constraint when a function realize a requirement.

4.1.3 Function meta-model

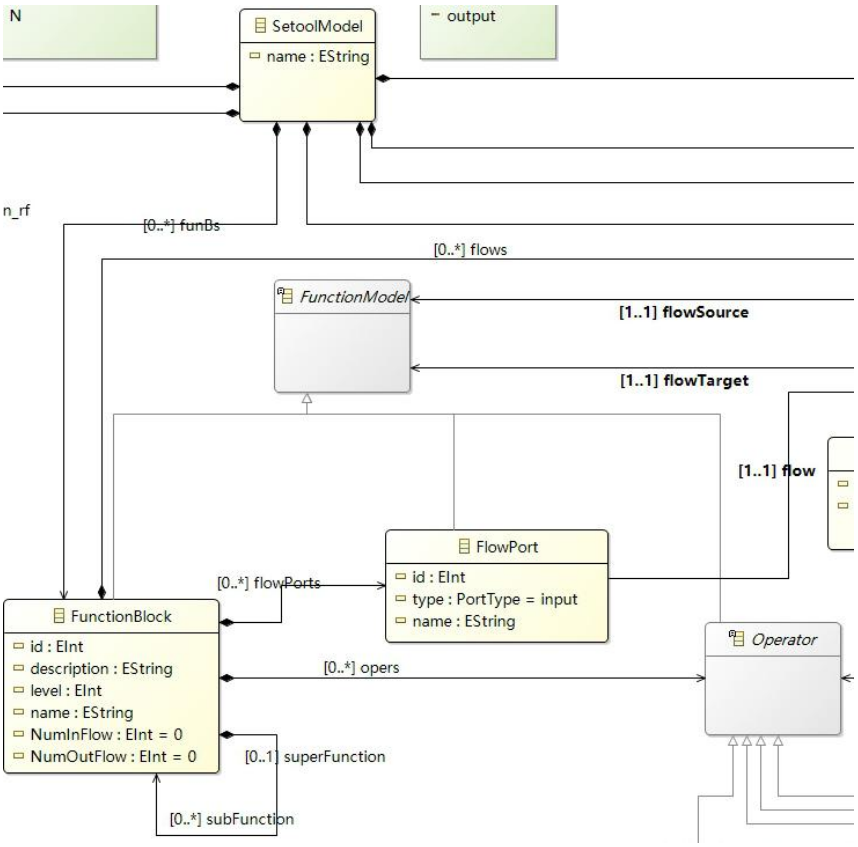


Figure 4.1.3

FunctionBlock and FunctionModel represent the definitions of the basic attributes and associations in Function module. FunctionModel is a virtual container that include more function blocks, flows, operators and ports. FunctionModel is abstract, it don’t have any instance. An instance of functionBlock is a special FunctionModel, it is able to include some sub function blocks. In a Setoolmodel module, it can contain zero or more function blocks which are named “funBs”. There is able to be an association which is similar to single inheritance between two functions. In my definition, one superFunction has zero or more subFunctions and one subFunction has zero or one SuperFunction. That is to say, in a superFunction container, there are more subFuction blocks.

In a function model, a function has a unique integer variable “id” to identify, a string variable “description” to express the function content, a integer variable “level” to express which level of sub function the function is in, a string variable “name” to express the name of function, a integer variable “NumInFlow” to show the quantity of in flows and a integer variable “NumOutFlow” to show the quantity of out flows.

“FlowPort” is a special FunctionModel. The purpose of using it is to identify flows’ targets clearly, when the user switches between different levels of function models. Each flow will generate a flow port in lower function level and point to a sub function target by FlowPort.

A FunctionModel has one or more inflow ports and one outflow port. For a FlowPort, there are a unique integer variable “id” to identify, a string variable “name” to express the name and a custom variable “type” to determine its port type.

4.1.4 Flow meta-model

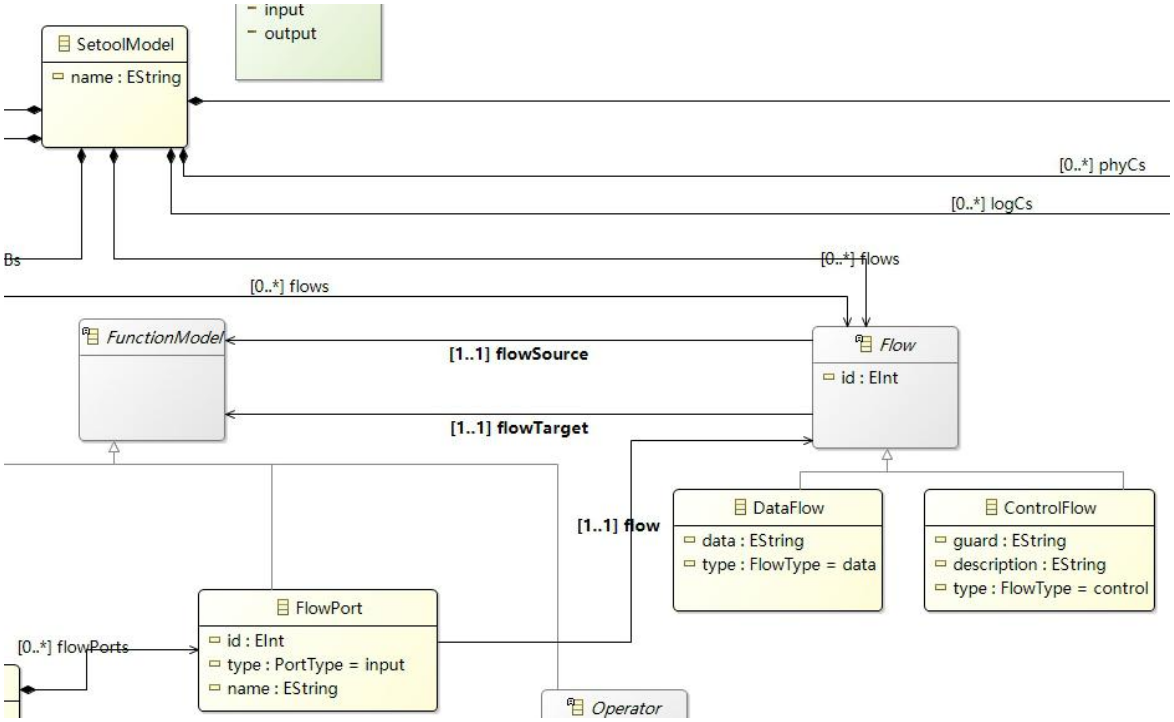


Figure 4.1.4

“Flow” represent the definitions of the attributes about flows in function module. Flow is abstract and it don’t have any instance. Flow has two sub-classes, “DataFlow” which represent association of data information and “ControlFlow” which represent association of control information between two different functions. In a Setoolmodel module or a function module, it can contain zero or more flows which are named “flows”.

For each Flow in a model, there are a unique integer variable “id” to identify, a variable “flowSource” to specify the source of flow and a variable “flowTarget” to specify the target of flow. Both source and target are instances of FunctionModel. In particular, a DataFlow has a string variable “data” to describe information about data content and a custom immutable constant “type” whose value is “data” to determine its flow type. Relatively, a ControlFlow has a string variable “guard” to describe expression of guard conditions, a string variable “description” to express information about control contents and a custom immutable constant “type” whose value is “control” to determine its flow type.

4.1.5 Operator meta-model

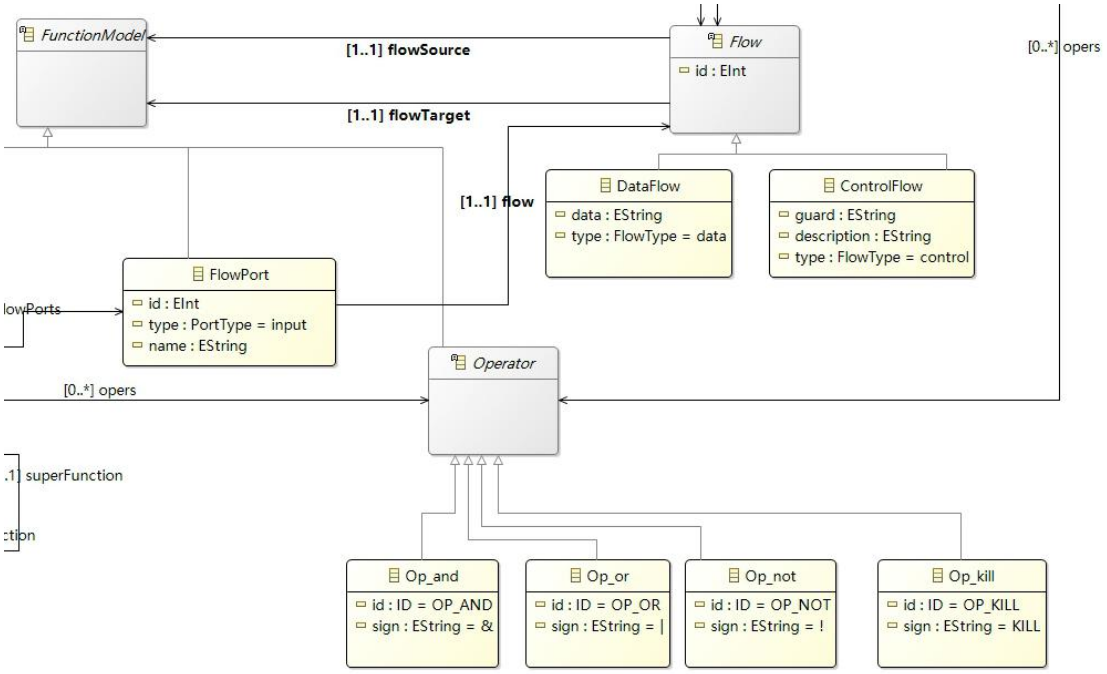


Figure 4.1.5

"Operator" represent the definitions of the attributes about operators in function module. Operator is abstract and it don't have any instance. In order to clearly show the effect of operator on flow in models, Operator is considered a special function model in the definition. Operator has four sub-classes, "Op_and" which represent logical AND operation, "Op_or" which represent logical OR operation, "Op_not" which represent logical NOT operation and "Op_kill" which represent an end of flow. In a Setoolmodel module or a function module, it can contain zero or more operators which are named "opers".

For each operator in a model, there are a unique immutable constant "id" to identify and a string immutable constant "sign" to show its sign expression.

4.1.6 Quick Tutorial

In order to complete the above meta-model, i used Obeo Designer which guarantees the set up and the deployment of industrial-strength modeling workbenches created with Sirius. It provides professional support and collaborative features.

Ecore Tools is already installed in Obeo Designer. If we use Eclipse, we need to download and install Ecore Tools in "Help/Install New Software" above all.

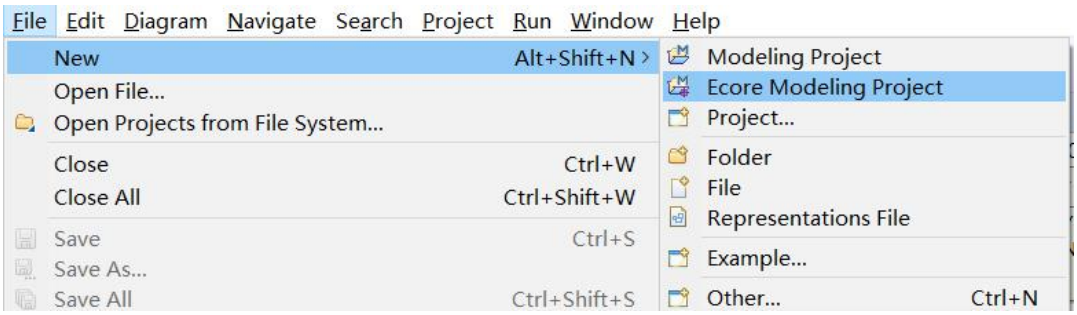


Figure 4.1.6

Then we create an Ecore Modeling Project as shown in figure 4.1.6.

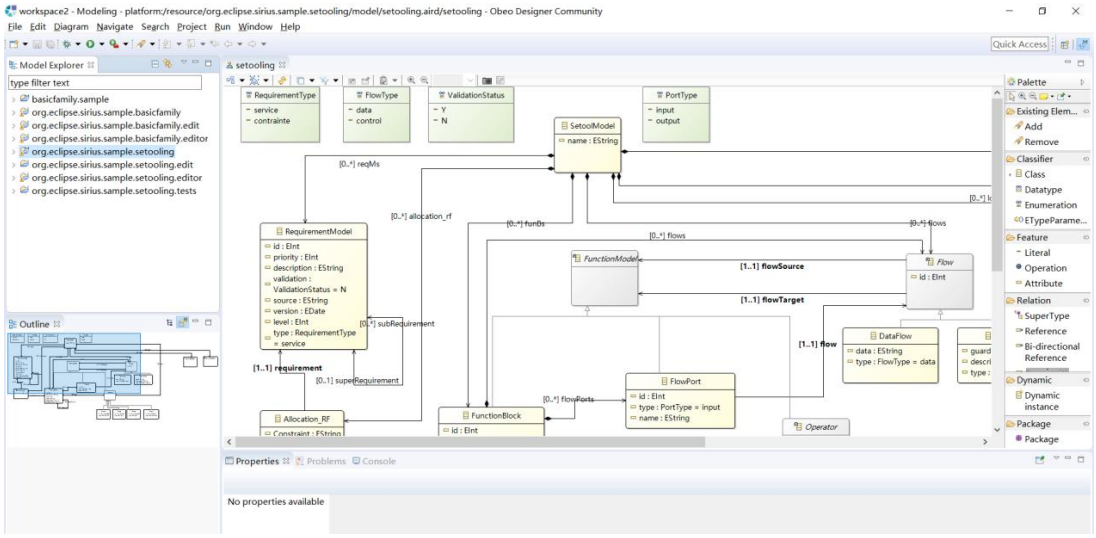


Figure 4.1.7

We use the palette to create the elements of the meta-model described previously.

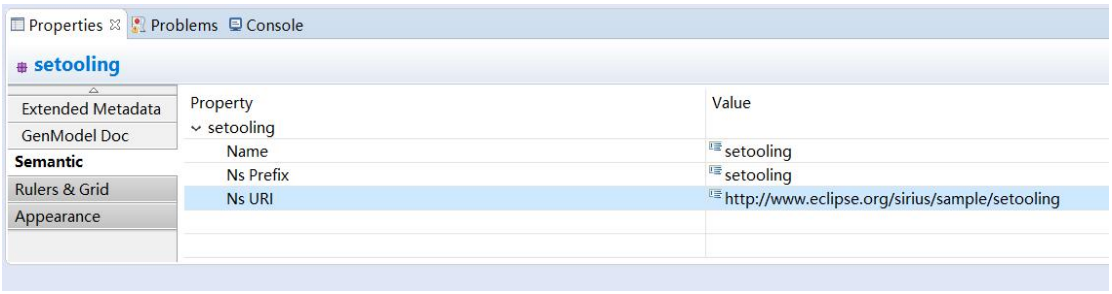


Figure 4.1.8

After completing the model, we set the Ns URI for generating the source code of the Meta-model.

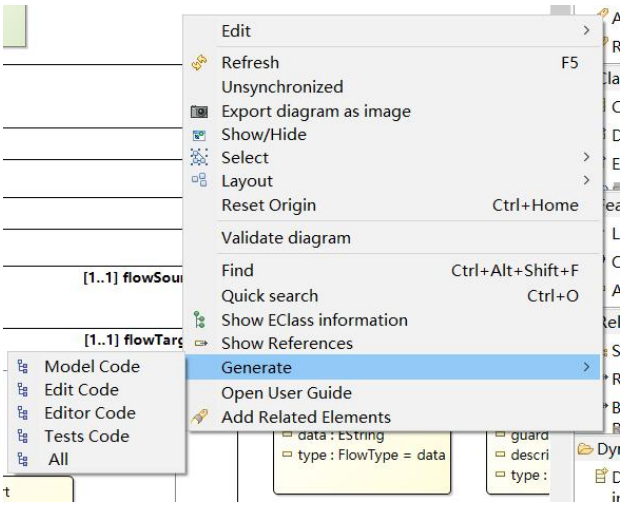


Figure 4.1.9

Finally, we right-click on the class diagram and select “Generate > All” to generate the source code of the meta model.

4.2 Create Tools (Sirius)

4.2.1 Necessary Configurations

After finishing the work of meta-modeling, i launched a runtime to use the Setooling concepts.

To launch a new eclipse application click on Run/Run Configurations and double click on Eclipse Application to get a New_configuration. If Eclipse is not running with java 8, in order to comfortably run Sirius in the new runtime, it should add the option “-XX:MaxPermSize=256m” in VM arguments.

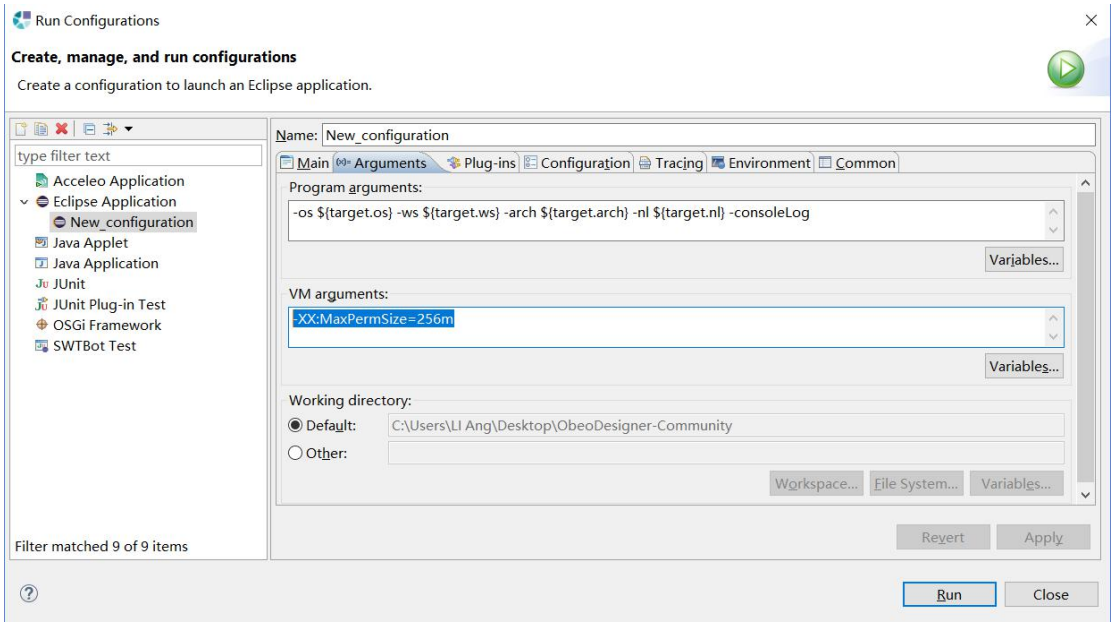


Figure 4.2.1

In the runtime window, i create a Viewpoint Specification Project. The Viewpoint

Specification Project creation wizard creates a new project containing a .odesign file. This file describes the modeling workbench. It will be interpreted by the Sirius runtime. Sirius automatically opens this file with a specific editor.

In this file the wizard has created a first viewpoint named MyViewpoint which i renamed “SetoolModel”. A viewpoint provides a set of representations (diagrams, tables or trees) that the end user will be able to instantiate.

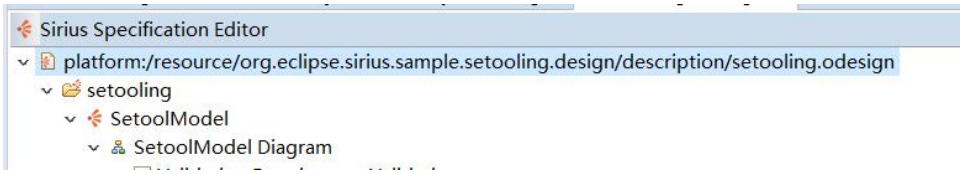


Figure 4.2.2

Before going forward, it has to declare the required meta-models that declare the types used by the modeling tool. It has to open the MANIFEST.MF file of the project and add the plug-in that defines the meta-models, in my project “org.eclipse.sirius.sample.setooling” in the list of the Required Plug-ins in the Dependencies tab.

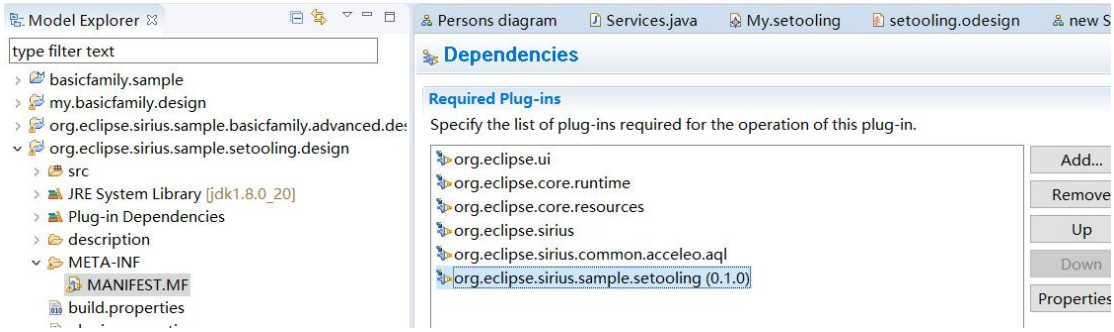


Figure 4.2.3

As shown in Figure 4.2.4, i define four main representations to realize the design parts of my project: SetoolModel diagram, FunctionStructure Diagram, Requirement table and Validation_RF table, as shown in Figure 4.2.4.

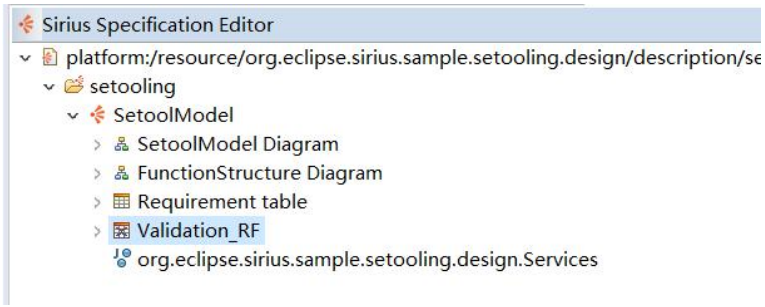


Figure 4.2.4

In SetoolModel diagram, i configure this diagram to graphically represent instances of SetoolModel which was defined in my meta-models.

In FunctionStructure Diagram, i configure this diagram as a special SetoolModel diagram that just focus on representing the associations of sub functions.

Requirement table is a table to represent instances of requirements in form of table.

Validation_RF is a table to represent the validation association between requirements and functions.

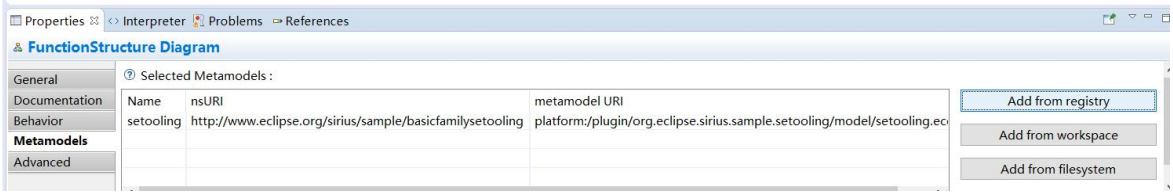


Figure 4.2.5

It's important to start create these representations by associating the meta-model that defines the types used by this diagram. We can select setooling meta-model from the registry in the Metamodel tab of Properties as shown in Figure 4.2.5.

4.2.2 SetoolModel Diagram

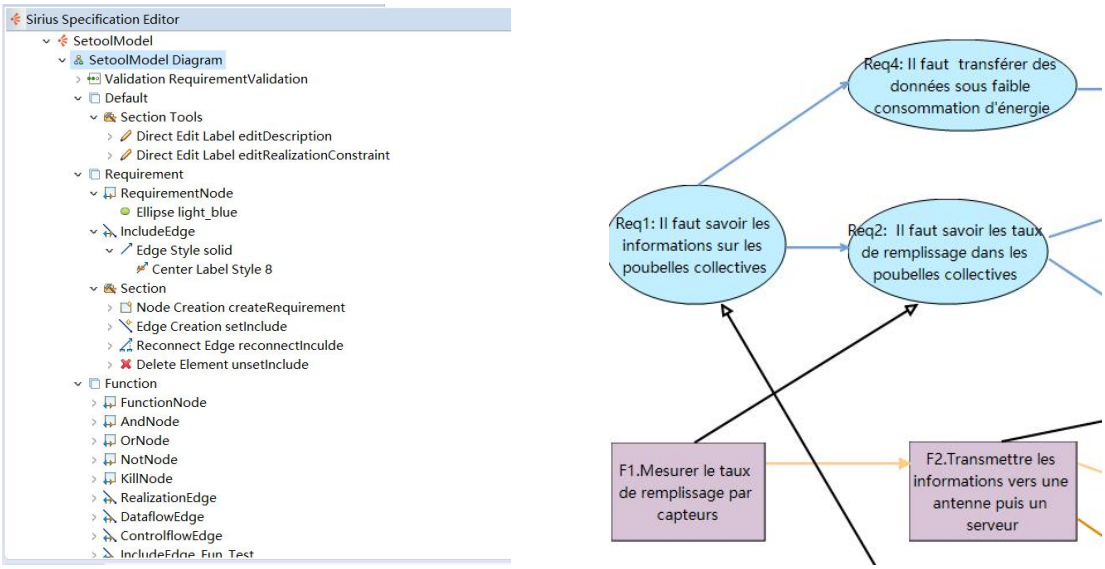


Figure 4.2.6

As shown in Figure 4.2.6, SetoolModel Diagram is the basic diagram about requirements and The highest level functions. This diagram shows macroscopically the associations between requirements and between requirement and zero level functions. For example, it's in order to achieve the contents shown on the right of Figure 4.2.6 which. The diagram includes three layers: Default layer for representing all elements, Requirement layer for representing all requirement elements and Function layer for representing all function elements.

The Diagram shows Nodes which are elements of the model. To create a new Node to the diagram, right-click on the Layer and select the menu New Diagram Element... / Node. And all created nodes must have an Id and must be associated to a Domain class.

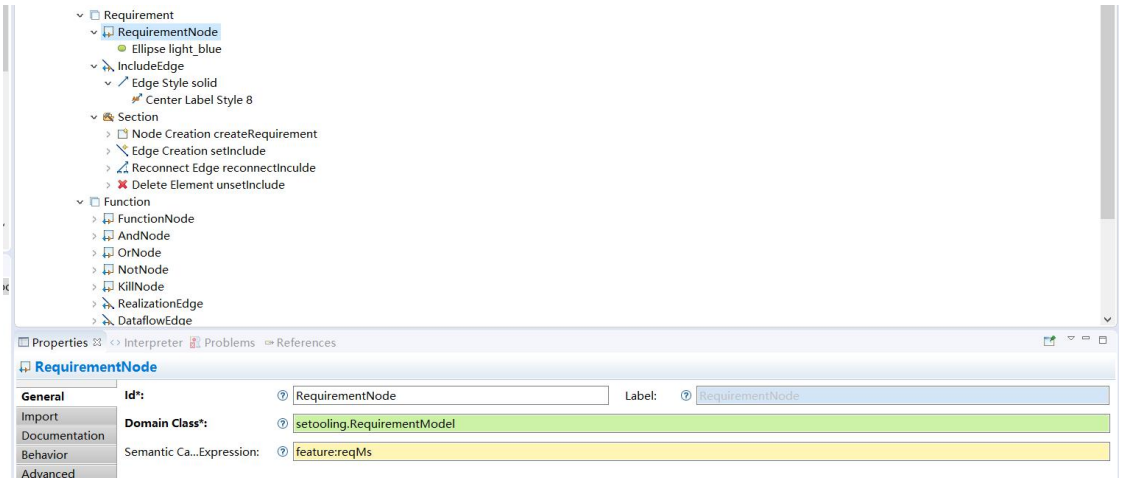


Figure 4.2.7

The Domain class must be consistent with the meta-model class. Semantic Candidates Expression is also important. It restrict the list of elements to consider before creating the graphical elements. If it is not set, then all semantic models in session will be browsed and any element of the given type validating the precondition expression will cause the creation of a graphical element. If this attribute is set then only the elements returned by the expression evaluation will be considered. As shown in Figure 4.2.7, “reqMs” is a feature of SetoolModel according to the definition in meta-model.

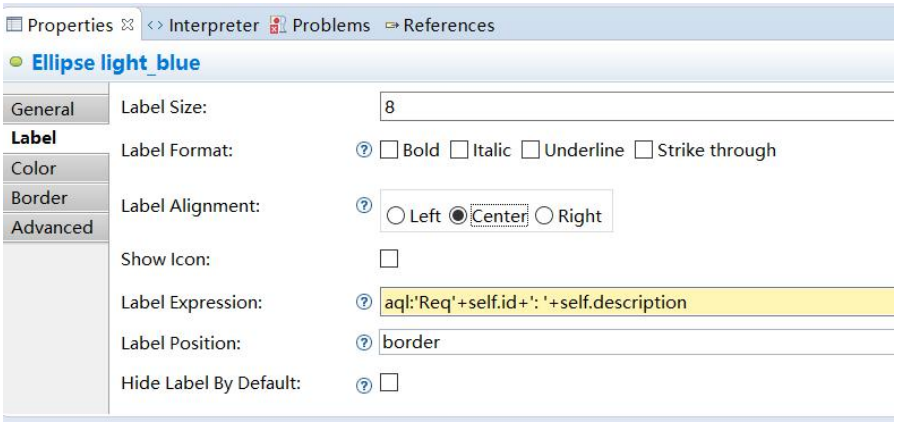


Figure 4.2.8

The Style must be created for a node then it could be graphically represented. I configure requirement nodes as light blue ellipse, function nodes as light purple square and four operator nodes as green diamond. The Style also defines the label of the Node. By default the label is calculated with the expression feature:name. By using AQL syntax,it allows to enter our own expressions. For example, i want to customize automatically the requirement name by specifying the Label Expression to “aql:'Req'+self.id+': '+self.description”. It also allows to specify the default size of the node.

To display relations between Nodes, we can create a Relation Based Edge or a Element Based Edge. According to the definition in the meta-model, the relations between requirement nodes are Relation Based Edge that named “IncludeEdge”. The

relations between function node and requirement node are Elements Based Edge that named “RealizationEdge”. The relations between function nodes and operator nodes are also Elements Based Edges that named “DataflowEdge” and “ControlflowEdge” in my design.

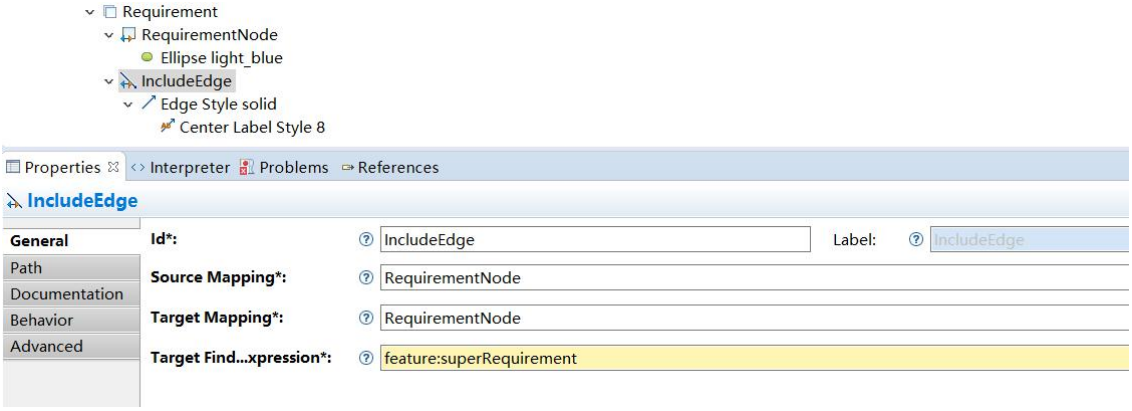


Figure 4.2.9

To create Relation Based Edge, enter the Source and Target Mappings of the relation. Sirius will select each instance of the concerned nodes which are visible on the current diagram. Then, Sirius will evaluate the Target Finder Expression on each Source Mapping to detect the corresponding Target Mapping. In my project, “IncludeEdge” represent the relation between super and sub requirements. Therefore, we set the configuration as shown in the Figure 4.2.9. the Target Finder Expression is the expression which should return, starting from the target semantic element.

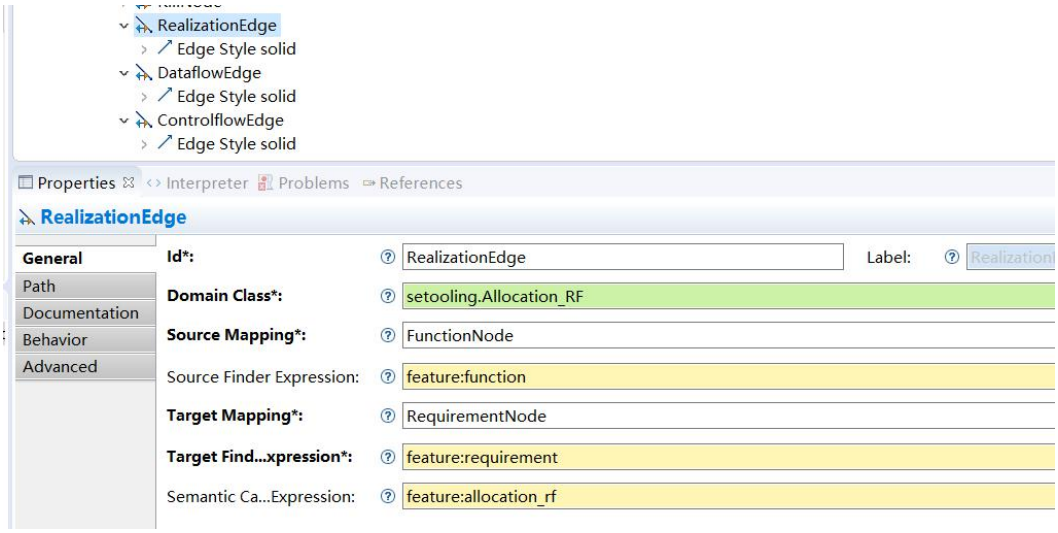


Figure 4.2.10

Creating Element Based Edge is more complex than Relation Based Edge. Element Based Edge is associated with a Domain Class which must be defined in meta-model and it has its own instances. In my project, “RealizationEdge” represent the relation that a function achieves a requirement. Therefore, we set the configuration as shown in the Figure 4.2.10.

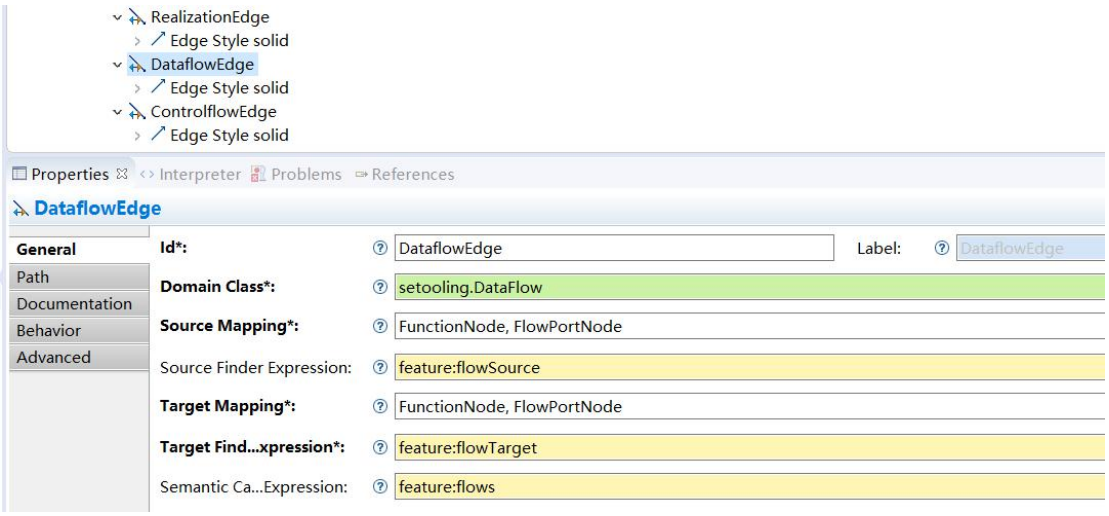


Figure 4.2.11

Similarly, “DataflowEdge” represent the data flow between functions. In order to facilitate the development of multilevel function structure diagram, i add the “FlowPort” nodes. We set the configuration as shown in the Figure 4.2.11.

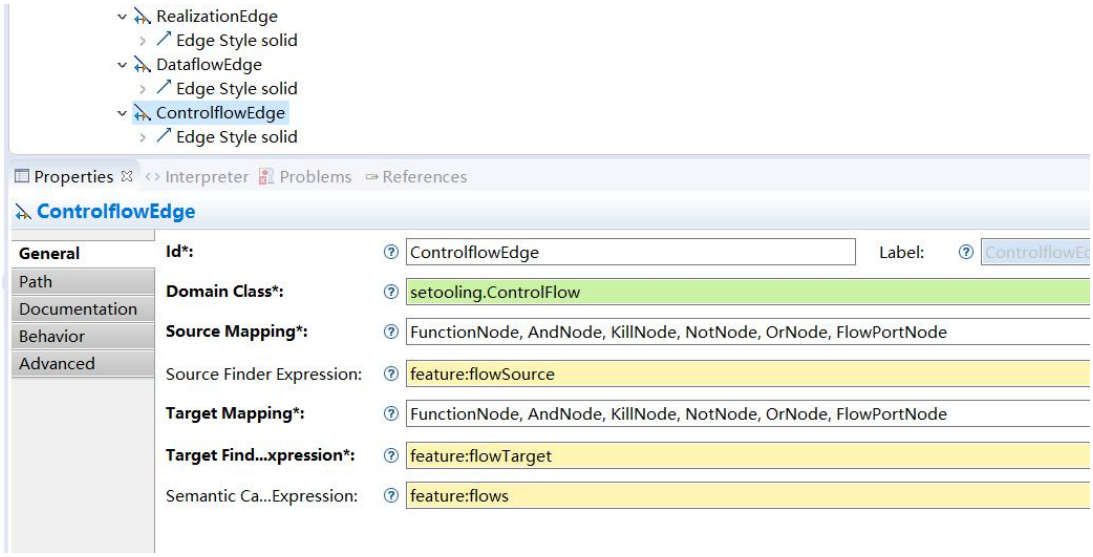


Figure 4.2.12

Similarly, “ControlflowEdge” represent the control flow between functions. Considering the logic operation between functions, i add four operator nodes as the additional source and target. Therefore, we set the configuration as shown in the Figure 4.2.12.

Edges are initialized with a default style which defines its graphical appearance. We can also edit the Style of the Relation like editing nodes.

In the previous steps, we have created a modeling workbench which can display an existing model. And then we need to complete this designer with a palette containing tools to allow users to create new model elements. We can add a new Section to the Layer (menu New tool... / Section) to achieve the goal.

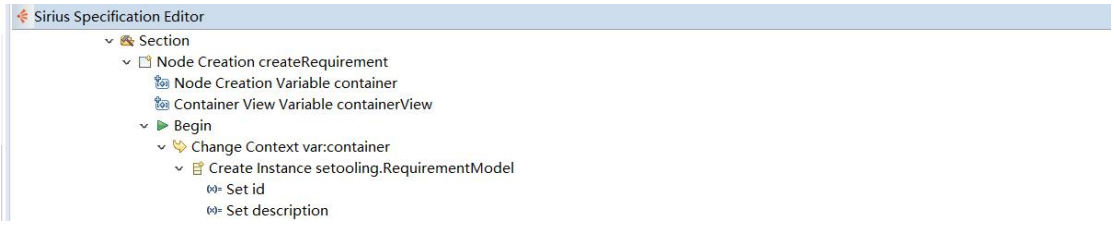


Figure 4.2.13

The palette is composed of tools which will allow the user to create new objects. To create new instances of Domain Classes, add a Node Creation element to the Section (menu New Element Creation... / Node Creation Description). To define which kind of nodes will be created by the tool, the Node Creation Description element must be associated to an existing node.

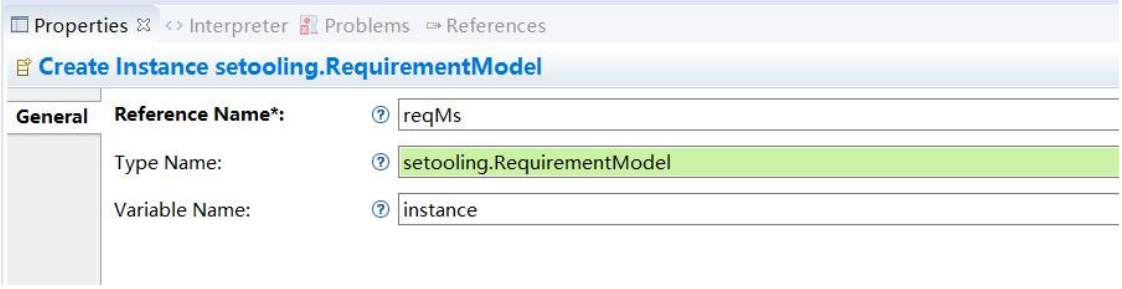


Figure 4.2.14

When the user will use the tool, actions will be executed. We have to define them. The first step consists in adding a Change Context element to define the context of the actions. The Change Context element contains an expression which allows Sirius to find the model element which will be the context. To write the expression, we can use the variable container which references the object holding the current diagram. Then, add a Create Instance which will create a new instance of RequirementModel, as shown in Figure .

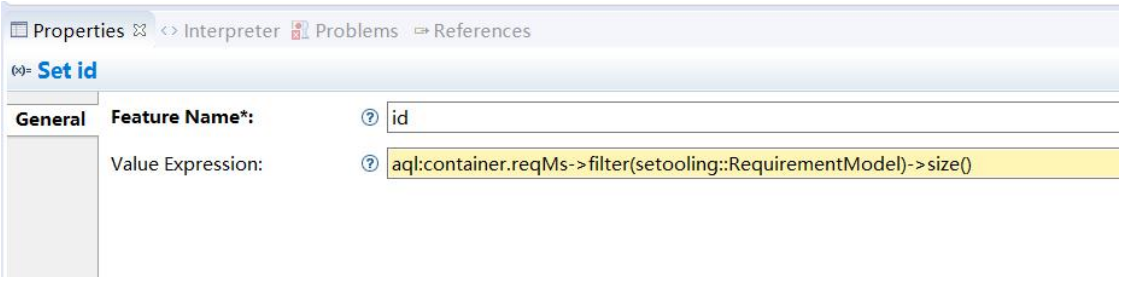


Figure 4.2.15

To specify the name of the created element, we can also add a Set Value operation. In this part of my project, i set id and description of requirement. For example in Figure 4.2.15, the AQL expression “aql:container.reqMs -> filter (setooling:: RequirementModel)->size()” will compute a default id depending on the number of requirement already created.

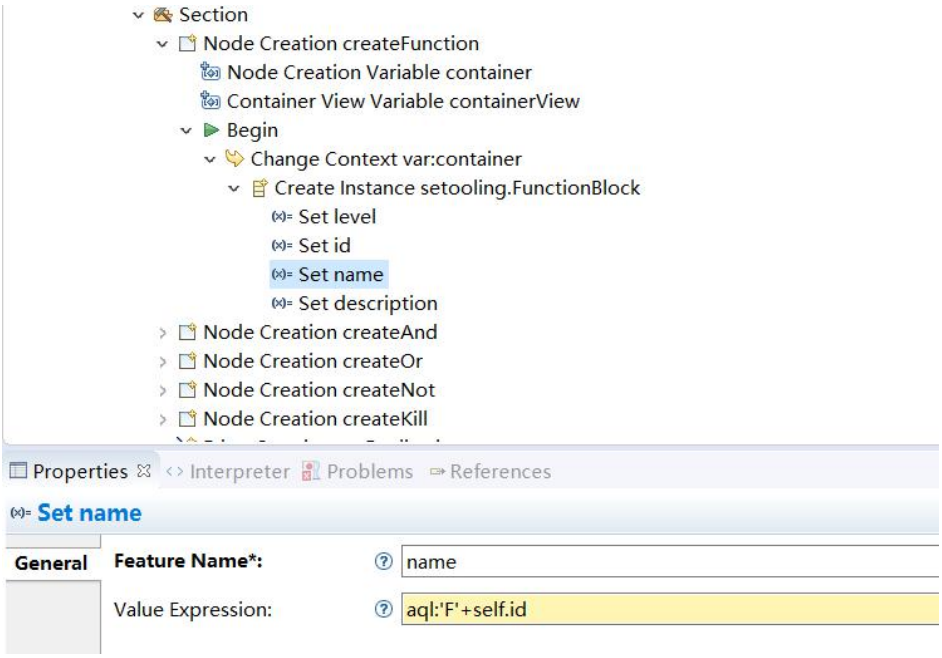


Figure 4.2.16

Similarly, we add other Node Creation elements about function and operators, as shown in Figure 4.2.16. Then depending on the definition in meta-model, we modify the attributes that need to be set and write the corresponding AQL expression.

The last important part in SetoolModel Diagram is the operations about Edge. Firstly, an Edge Creation tool allows the user to create relationships directly from the diagram, by using the palette. To create a edge, right click on the Section and select the menu New Element Creation > Edge Creation.

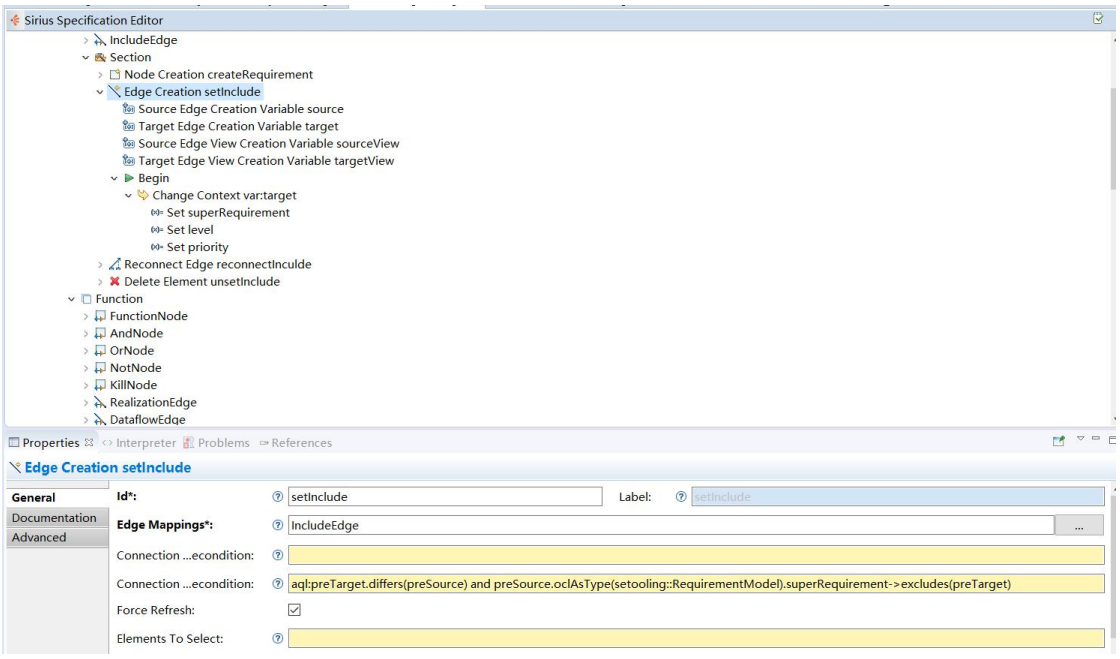


Figure 4.2.17

As a example in my project, give an Id to this tool “setinclude” and associate it to

“IncludeEdge”, the Edge Mapping which defines the graphical relation between two requirements. To prevent the user to create a include relationship from a super requirement to itself, or to one of its sub requirements, add the precondition expression “aql:preTarget.differs(preSource) and preSource.oclAsType (setooling::RequirementModel).superRequirement->excludes(preTarget)”.

Then under the Begin object, create a Change Context. Set its Browse Expression to var:target in order to define the execution context of the next operations. Then under the Change Context create a Set superRequirement. It will set the super one of the first requirement clicked (target) to the second requirement clicked (source).

Secondly, a Reconnect Edge tool allows the user to change the end of a relationship by moving it directly from the diagram. To create a reconnect tool, right click on the Section and select the menu New Element Edition > Reconnect Edge.

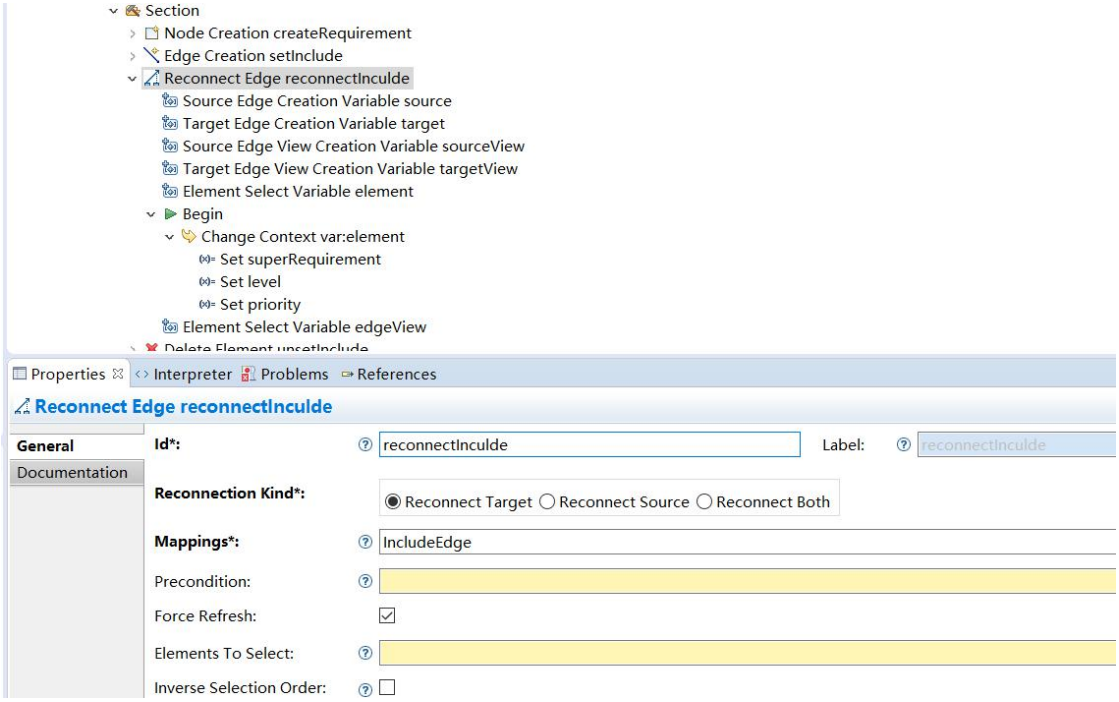


Figure 4.2.18

In this part of my project, i associate the IncludeEdge to this reconnect tool. This tool comes with six variables: source is the object currently attached to the moved end; target is the object going to be attached to the moved end; sourceView is the graphical object representing source; targetView is the graphical object representing target; element is the object attached to the other end; elementView is the graphical object representing element.

Then create a Change Context and set its expression to var:element that means the requirement who will change its super requirement. Then create a Set to assign the new selected superRequirement (var:target) as super one of this requirement.

Finally, a Delete Element tool specifies which actions have to be performed when the user hits the delete key on an diagram element.

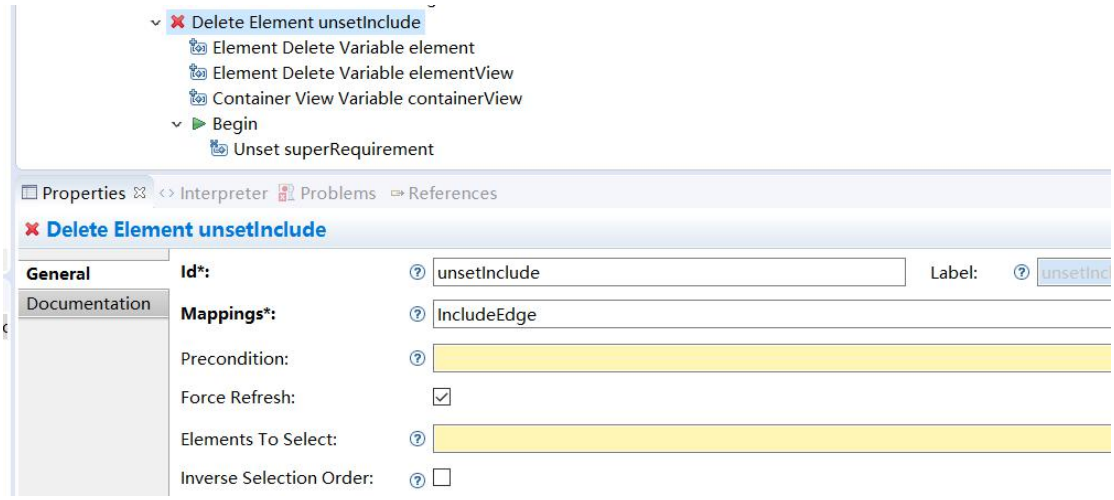


Figure 4.2.19

It is necessary on elements which deletion can't be interpreted by Sirius (for example edges) or if it need to perform specific actions. Right click on the Section and select the menu New Element Edition > Delete Element. In this part, i associate the IncludeEdge to this Delete tool. When deleting an edge, the variable element refers to the source of the relationship. So, create a Change Context and set its expression to var:element (the requirement which is the sub one). Then create a Unset on the feature superRequirement to remove this relation.

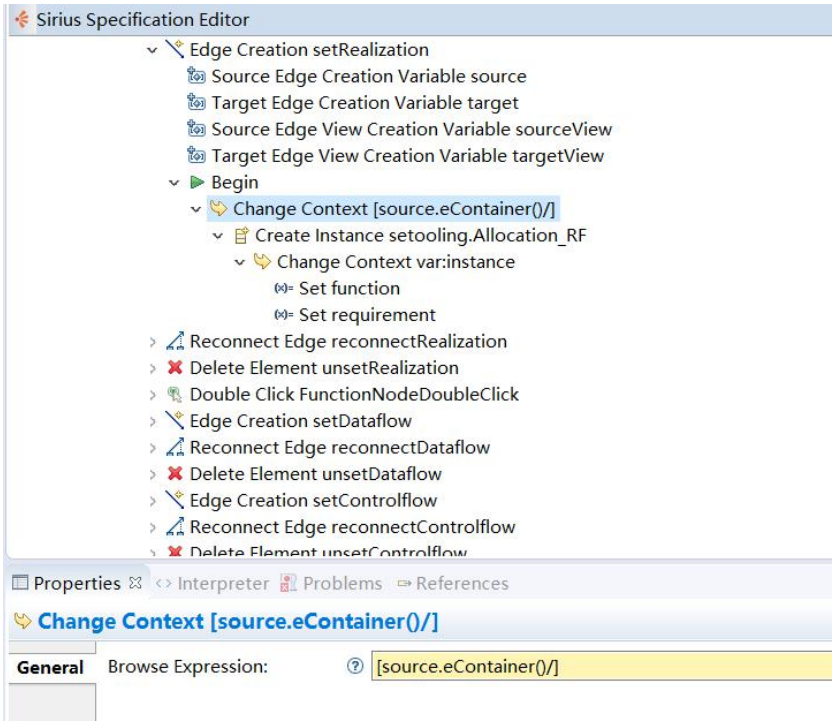


Figure 4.2.20

We can add other Edge Creation, Reconnect and Delete tooling about function and operators. Then depending on the definition in meta-model, we modify the attributes that need to be set and write the corresponding AQL expression. However, there are some differences between Relation Based Edge and Element Based Edge.

Because of based on elements, Element Based Edge can not change the value of its source or target by using “var:source” or “var:target”. To solve this problem, eContainer() is used as a virtual container. As shown in Figure 4.2.20, i use “[source.eContainer()/]” to create a virtual container between a function block and a requirement block for displaying graphically the realization relation. Then create an edge instance in this virtual container. At last, create a Set function as the source of the edge and a Set requirement as the target.

At the end of this part, there are other tools created in SetoolModel Diagram.

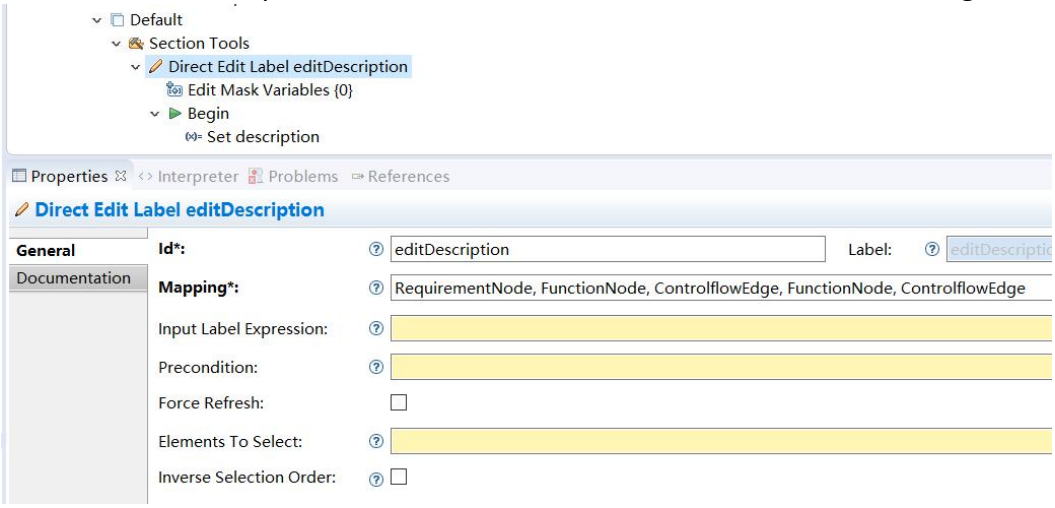


Figure 4.2.21

As shown in Figure 4.2.21, a Direct Edit Label specifies how to interpret the modification of graphical object's label in order to modify the model. The Direct Edit Label “editDescription” is created to change the descriptions of requirement and function by clicking the blocks. This tool comes with a mask that creates variables depending on the label's value. By default, the mask is set to {0}, which means that the variable named 0 will contain the full value of the label.

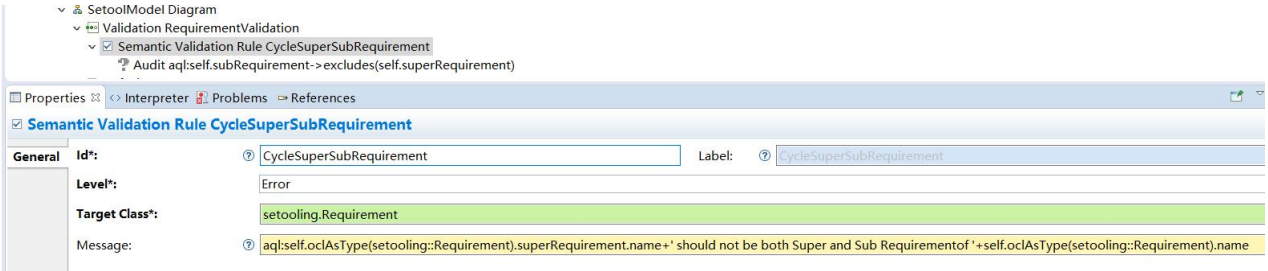


Figure 4.2.22

As shown in Figure 4.2.22, validation rules allow the user to evaluate the quality of a model. In my project, i add a rule named “CycleSuperSubRequirement” to prohibit the cycle between requirements by creating an Audit (a condition corresponding to this rule) with the expression “ aql: self.subRequirement-> excludes (self.superRequirement)”.

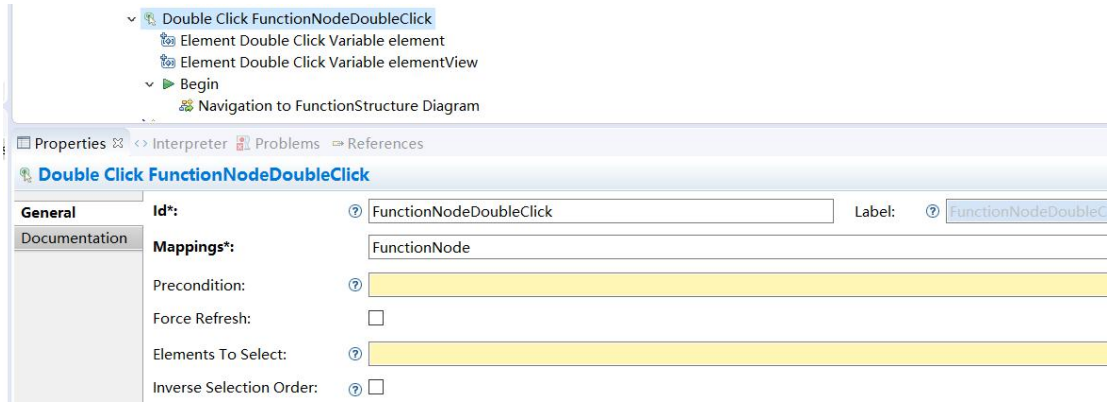


Figure 4.2.23

As shown in Figure 4.2.23, to facilitate the navigation between the different level function structure diagrams, i create a Double Click tool that allows the user to navigate from any super level function structure diagram to its sub level diagram.

4.2.3 FunctionStructure Diagram

The FunctionStructure Diagram is a special SetoolModel Diagram which focus on displaying the internal structure of a super function. The FunctionStructure Diagram includes all the elements and tools on the function layer of SetoolModel Diagram.

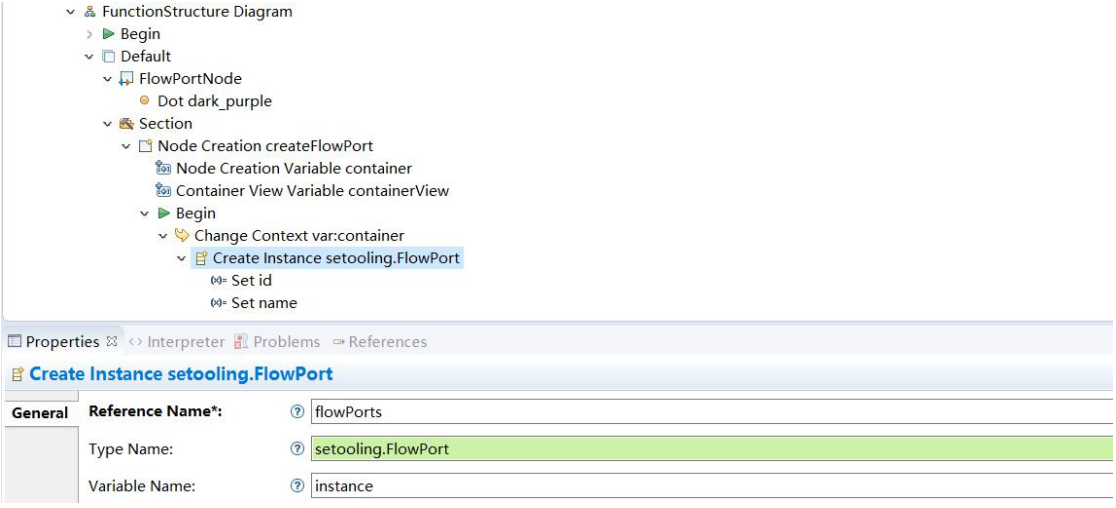


Figure 4.2.24

Moreover, as shown in Figure 4.2.24, there is the FlowPort Node and its tools in the FunctionStructure Diagram. According to the quantity of flows associated with the super function, the FlowPort nodes will be created automatically in the super function's structure diagram.

4.2.4 Requirement Table

With Sirius it is also possible to represent model elements with a Edition Table. Right-click on the Viewpoint and select the menu New representation > Edition Table

Description.

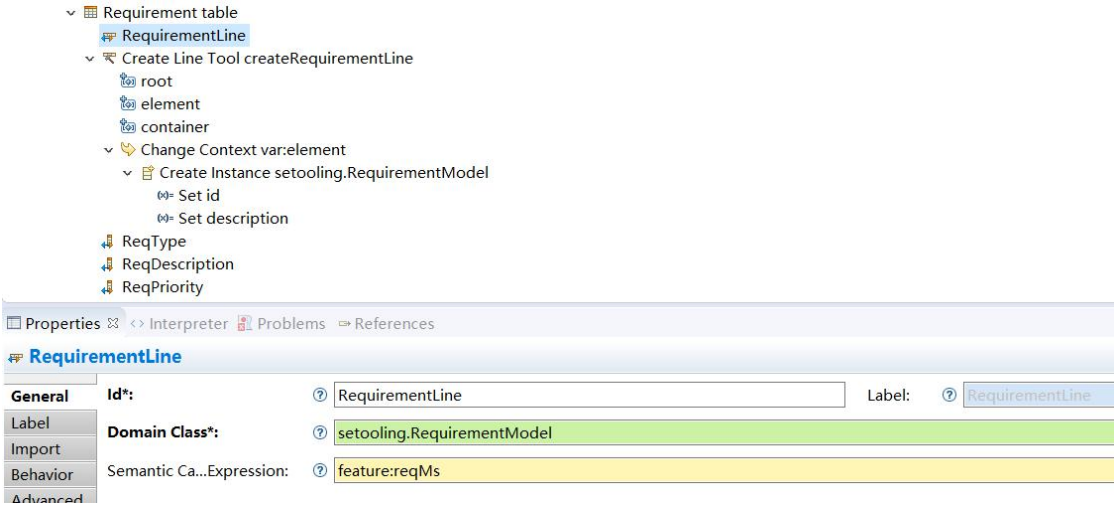


Figure 4.2.25

In my project, i need lines to represent the requirements. So, right-click on the Table and select the menu New Table Element > Line. Then i set Id, Domain class and Semantic Candidates Expression to the line as shown in Figure 4.2.25. ReqType, ReqDescription and ReqPriority are three feature columns about the attributes of type, description and priority that defined in meta-model. Finally, i add a Create Line Tool to update automatically this table depending the modifications in SetoolModel diagram.

4.2.5 Validation_RF Table

The validation_RF table is a Cross Table Description which is different with the Requirement table.

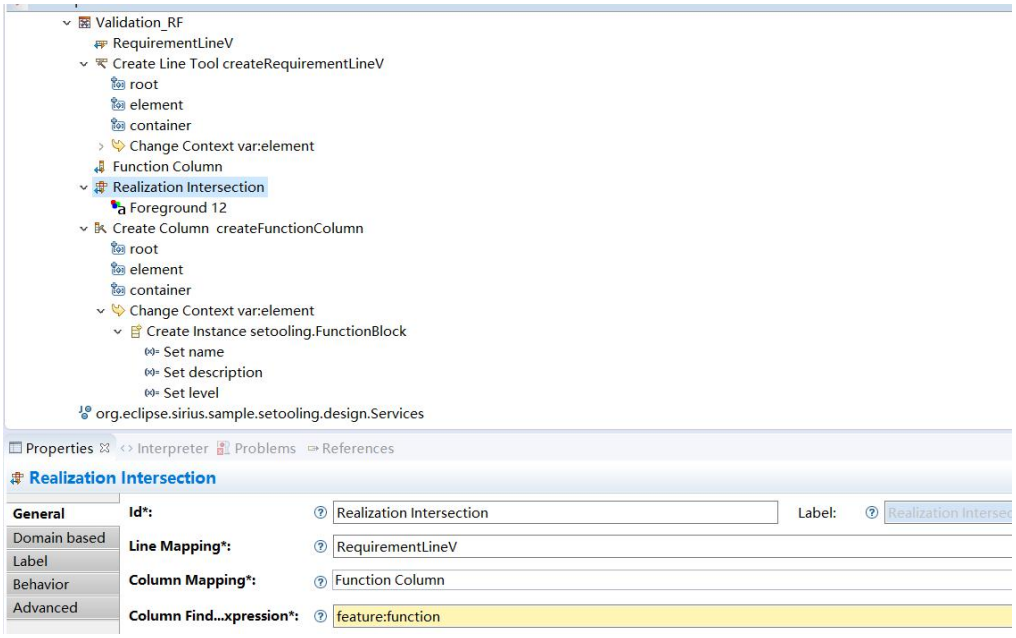


Figure 4.2.26

As shown in Figure 4.2.26. The lines still represent requirements, but the columns are changed into functions. A Create Line Tool and a Create Column Tool are created to update automatically this table depending the modifications in SetoolModel diagram. An intersection tool is created to configure the feature of intersection.

4.3 Generate (Acceleo)

. In this part of project, we want to generate the source code and the documents from the case which is created above by SETool.

Using the Acceleo perspective will make generations easier when working with Acceleo as it provides dedicated views and actions. Furthermore, being on that perspective will promote some menu items so that they are directly accessible instead of being nested within others. We can switch to the Acceleo perspective by selecting the Window > Open Perspective > Other... menu and selecting Acceleo in the popup that appears then.

```
[comment encoding = UTF-8 /]
[module generate('http://www.eclipse.org/uml2/3.0.0/UML'),

[template public generate(aClass : Class)]
[file (aClass.name.concat('.java'), false)]
  public class [aClass.name.toUpperFirst()/] {
  [for (p: Property | aClass.attribute) separator('\n')]
    private [p.type.name/] [p.name/];
  [/for]

  [for (p: Property | aClass.attribute) separator('\n')]
    public [p.type.name/] get[p.name.toUpperFirst()/]() {
      return this.[p.name/];
    }
  [/for]

  [for (o: Operation | aClass.ownedOperation) separator('
    public [o.type.name/] [o.name/]() {
      // TODO should be implemented
    }
  [/for]
  }
[/file]
[/template]
```

Figure 4.3.1

After creating a Acceleo project, we can generate the the documents, for example, create a bean for each of the classes defined in our sample model, as shown in Figure 4.3.1.

4.4 Feedbacks on the realization

During my internship , i used some software to realize my project, such as Eclipse, Sirius and Acceleo. The basic tutorials of three software are easy to find on Internet. However, in my opinion, the meta- modeling is not a easy domain for certain learners. It have to spend a lot of time to learn and understand. Then in Sirius, the AQL expression is a important part to realize a modeling tool, but there are little references on Internet.

5. Test

5.1 How to make a model

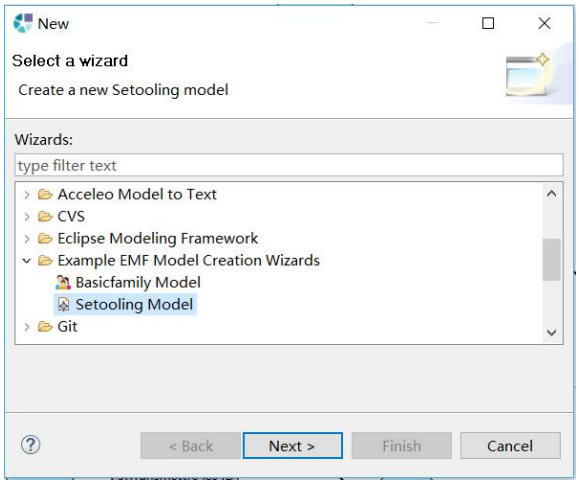


Figure 5.1.1

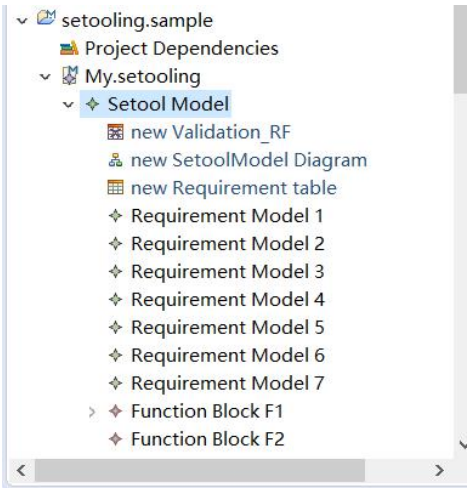


Figure 5.1.2

Firstly, we create a new modeling project, and On the new project created, right click on New > Other...Then in Example EMF Model Creation Wizards, select Setooling Model. Click on Next> button to create a Setooling model. Select SetoolModel as the type of the root object in the new model and click on Finish. Then new a SetoolingModel diagram, as shown in Figure 5.1.2. Similarly, new Validation_RF table and Requirement table can also be created.

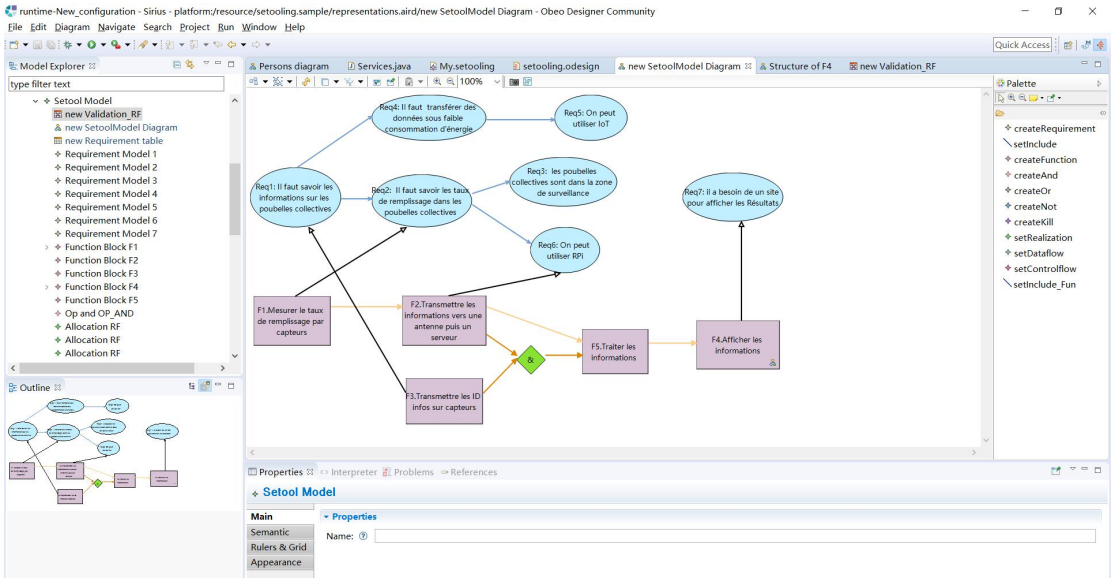


Figure 5.1.3

Secondly, in the new SetoolingModel diagram, we can use the Palette on the right side of Figure 5.1.3 to add new elements that have been designed.

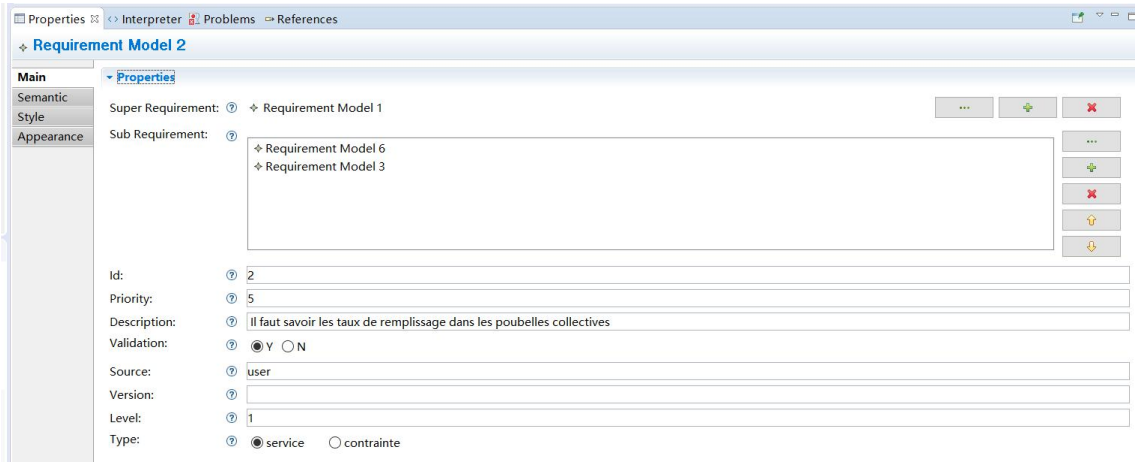


Figure 5.1.4

Thirdly, we can change the properties of all the elements like the example in figure 5.1.4.

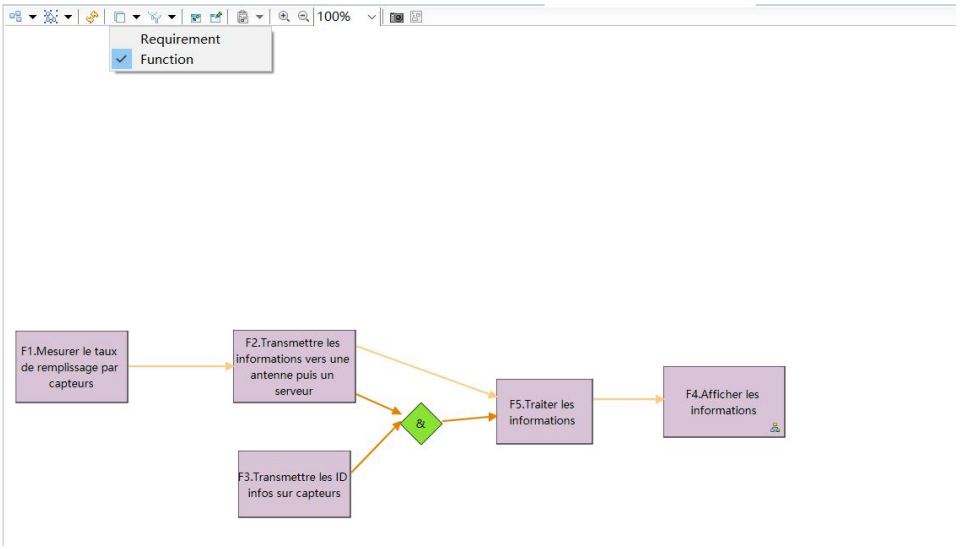


Figure 5.1.5

Fourthly, we can observe a finished diagram through different layers. As shown in Figure 5.1.5, it just display the elements on function layer.

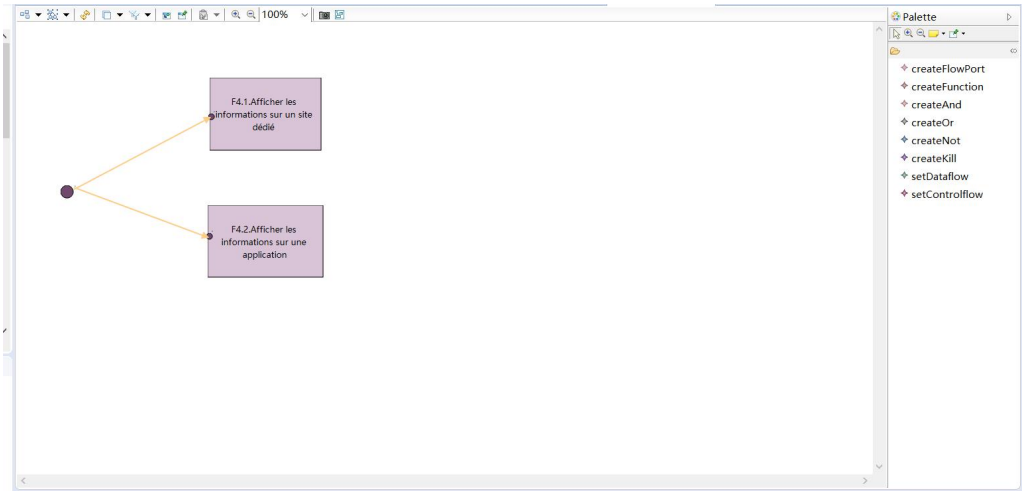


Figure 5.1.6

Finally, we can double click a function, for example “F4.Afficher les informations” in Figure 5.1.5, and then it will redirect to the “structure of F4” diagram. If a function has not yet a structure diagram, a double click will create a new one for it.

5.2 Test Case : Project SmartBrest

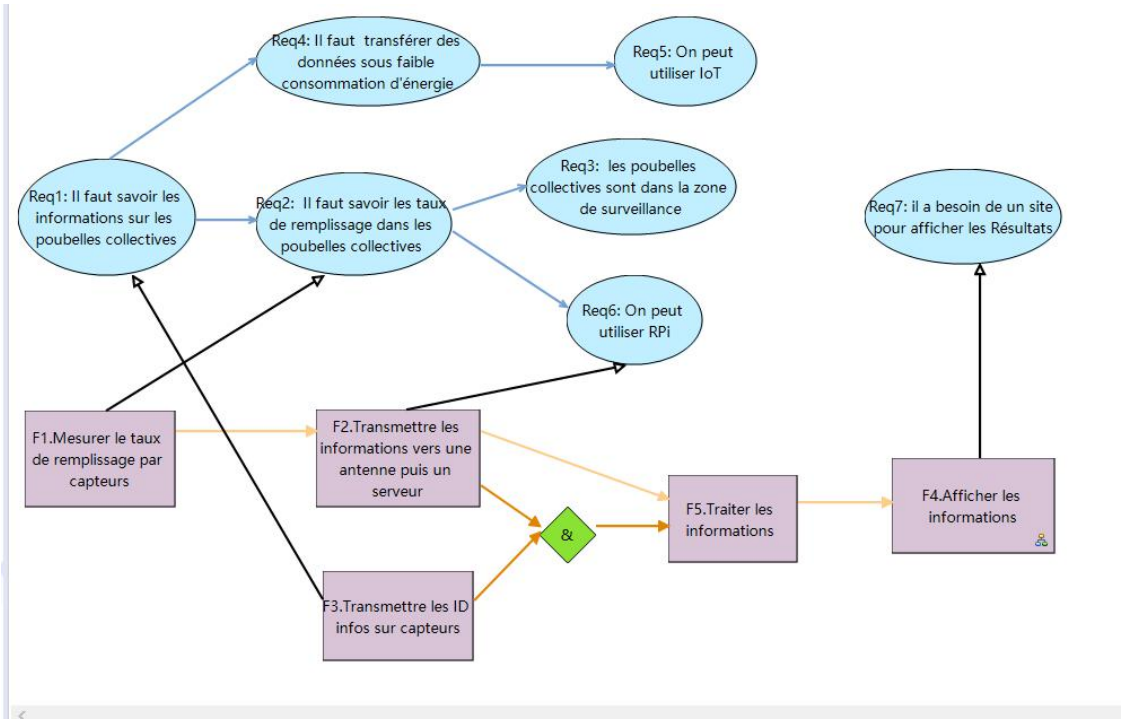
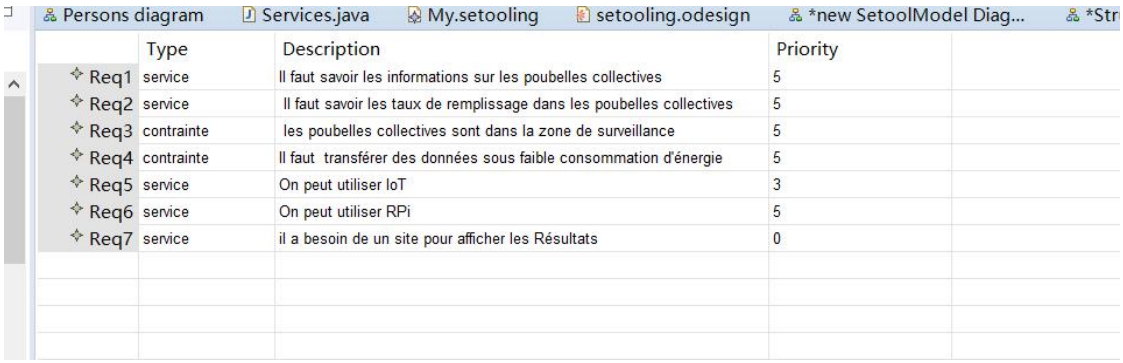


Figure 5.1.6

The project SmartBrest is a student project of ENSTA Bretagne. The purpose of this project is to develop an autonomous system of energy that would be present in certain waste containers and would allow cleaners to have information on the state of filling of these containers. This information would optimize the tours carried out and thus save time and money. I used it as a example in my SETool project to verify the availability of the tool.



	Type	Description	Priority
Req1	service	Il faut savoir les informations sur les poubelles collectives	5
Req2	service	Il faut savoir les taux de remplissage dans les poubelles collectives	5
Req3	contrainte	les poubelles collectives sont dans la zone de surveillance	5
Req4	contrainte	Il faut transférer des données sous faible consommation d'énergie	5
Req5	service	On peut utiliser IoT	3
Req6	service	On peut utiliser RPi	5
Req7	service	il a besoin de un site pour afficher les Résultats	0

Figure 5.1.7

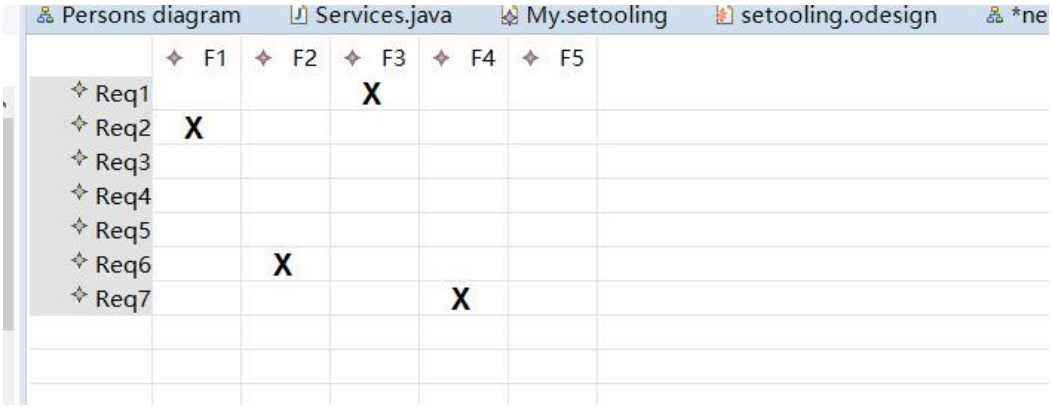


Figure 5.1.7

Moreover, SEtool generated automatically the requirement table(As shown in Figure 5.1.7) and the validation table(as shown in Figure 5.1.8) of SmartBrest. The requirement table is a table for requirement analysis during the system engineering process. The validation table shows the requirements and the functions which achieve them.

Conclusion

The project SETool achieved a modeling tool which is focus on realizing quickly the requirement analysis and functional architecture of the small projects for the students. It is helpful tool in the initial phase of system engineering process.

I have finished three main parts of this project: define our own Domain Specific Language by meta modeling, design SETool by Sirius and create a case by using SETool. Before the end of the internship,i will continue doing the generation of the source code and documents of this case. The current tool is not perfect. New functions and more friendly interfaces are worth being created.

From this project, i learned a lot about DSL, meta-model, designing a modeling tool. These knowledge and project experience must help me during my career period. Moreover, i find the gap between outstanding engineer and me. I sincerely admire those who have changed the world through technology.

At last, thank all my professors and classmates for helping me during my study period in France.

References

[1] Richard C. Gronback. <Eclipse Modeling Project: A Domain-Specific Language (DSL) Toolkit>

[2] AndrewFallOpens and Joseph Fall. <A domain-specific language for models of landscape dynamics>. Ecological Modelling. Volume 141. Issues 1–3. 1 July 2001. Pages 1-18 URL: https://doi.org/10.1016/S0304-3800(01)00334-9

[3] ALEXANDER FELFERNIG, GERHARD E. FRIEDRICH, and DIETMAR JANNACH, Int. J. Soft. Eng. Knowl. Eng. 10, 449 (2000). URL: https://doi.org/10.1142/S0218194000000249

[4] Sisi Xuanyuan, Yan Li, Lalit Patil, Zhaoliang Jiang. (2016) Configuration semantics representation: A rule-based ontology for product configuration. 2016 SAI Computing Conference (SAI), 734-741.

[5] Anders Haug. (2010) Managing diagrammatic models with different perspectives on product information. Journal of Intelligent Manufacturing 21:6, 811-822. Online publication date: 1-Dec-2010.

[6] Małgorzata Sadowska, Zbigniew Huzar. 2017. Semantic Validation of UML Class Diagrams with the Use of Domain Ontologies Expressed in OWL 2. Software Engineering: Challenges and Solutions, 47-59.

[7] https://wiki.eclipse.org/Sirius/Tutorials
[8] https://wiki.eclipse.org/Acceleo
[9] https://en.wikipedia.org/
[10] https://polarsys.org/capella/
[11] https://ecsoya.github.io/eclipse.tutorial/wiki/EMF-Tutorial