

Auteur : Geoffrey Copin, Yassine Ben Jemia, Zakaria Hamidi

12/10/2018

Type : STR

Equipe : Geoffrey Copin, Yassine Ben Jemia, Zakaria Hamidi

Statut : Initial

Destinataire(s) : CEA List

Page(s) 26

Définition d'un éditeur d'argument de sûreté de fonctionnement

STR

Version	Date	Modifications	Rédacteur(s)
1.0	15-11-2018	Création du plan de STR	Yassine Ben Jemia
2.0	17-12-2018	Rédaction	Yassine BEN JEMIA Zakaria HAMIDI Geoffrey Copin
3.0	20-01-2019	Ajout de diagrammes et captures Rédaction	Yassine BEN JEMIA Zakaria HAMIDI Geoffrey Copin
4.0	23-01-2019	Correction et mise en page	Yassine BEN JEMIA Zakaria HAMIDI Geoffrey Copin
5.0	06-02-2019	Corrections et ajouts	Yassine BEN JEMIA Zakaria HAMIDI Geoffrey Copin

Table des matières

Usage de ce document	3
Objectifs	3
Objets métiers	3
Workflow	4
Choix technologiques	5
Découpe en composants	5
WBS de la réalisation	7
Diagramme de WBS	7
Description des composants et tâches de la WBS	8
Définition des stéréotypes dans un profil GSN	8
Définition des ElementTypes	10
Menu New Child	11
Menu Propriétés	11
Palette	11
Vues graphiques en CSS	12
Fichier architecture	12
Vérification des contraintes	12
Bibliothèque de patterns	14
Plan d'intégration	20
Dépendance entre composants	20
Type de dépendance	20
Compromis de réalisation liés à la technologie	21
Nomenclature de l'IHM	22
Plan de déploiement et mise en exploitation	25
Plugin et configuration	25
Le plugin de documentation	26
Kit de livraison	27
Conclusion	27
Ontologie	28

Usage de ce document

Le but de la STR est de définir l'environnement techniques afin de formaliser les fonctionnalités de notre projet. Ce document décrit les choix techniques, l'architecture technique, les plans de test et intégration ainsi qu'une description détaillée sur les objectifs du projet.

Objectifs

L'objectif de ce projet sera d'élaborer un éditeur graphique d'argumentation de sûreté de fonctionnement, se basant sur le langage GSN et la norme SACM2.

L'éditeur devra permettre la création, la modification et la suppression de diagrammes à travers une palette qui contiendra les différents éléments et associations existantes dans le langage GSN, et également une validation dynamique entre ces derniers en fonction des contraintes existantes dans le langage.

Il devra également disposer d'une bibliothèque de patterns qui contient des modèles déjà créés, et ceci afin de faciliter à l'utilisateur la création de modèles.

Objets métiers

- Permettre à l'utilisateur de créer un diagramme et ajouter de nouveaux éléments, et cela via la palette ou le menu contextuelle
- Pouvoir modifier à tout moment les propriétés d'un élément par l'utilisateur
- Définir une relation entre deux éléments
- Contrôler la validité des relations à travers des contraintes
- Mettre à disposition de l'utilisateur une bibliothèque de patterns réutilisable

Workflow

Le plugin réalisé permet l'édition de modèles utilisant la notation graphique GSN.

Lors de l’affichage de la boîte de dialogue “New Papyrus Project”, l’utilisateur doit sélectionner l’option “Goal Structuring Notation” dans la section “Architecture Contexts” et cocher la case “GSN Analysis” parmi les *viewpoints* proposés. Pour finaliser la création du projet, il est nécessaire de sélectionner l’option “Goal Structuring Notation diagram” parmi les *représentations* et de charger le profil GSN.

Une fois le projet créé et ouvert, l’utilisateur peut ajouter des éléments à son modèle via la palette. Les propriétés des éléments sont éditables dans la vue *propriétés*. Il est également possible d’ajouter un élément au modèle via le menu contextuel de la vue *Model Explorer*. Les associations peuvent être ajoutés et modifiées de la même manière que des associations UML classiques. Enfin, la vue *Model Explorer* permet l’importation et l’instanciation des patterns définis dans la librairie de patterns.

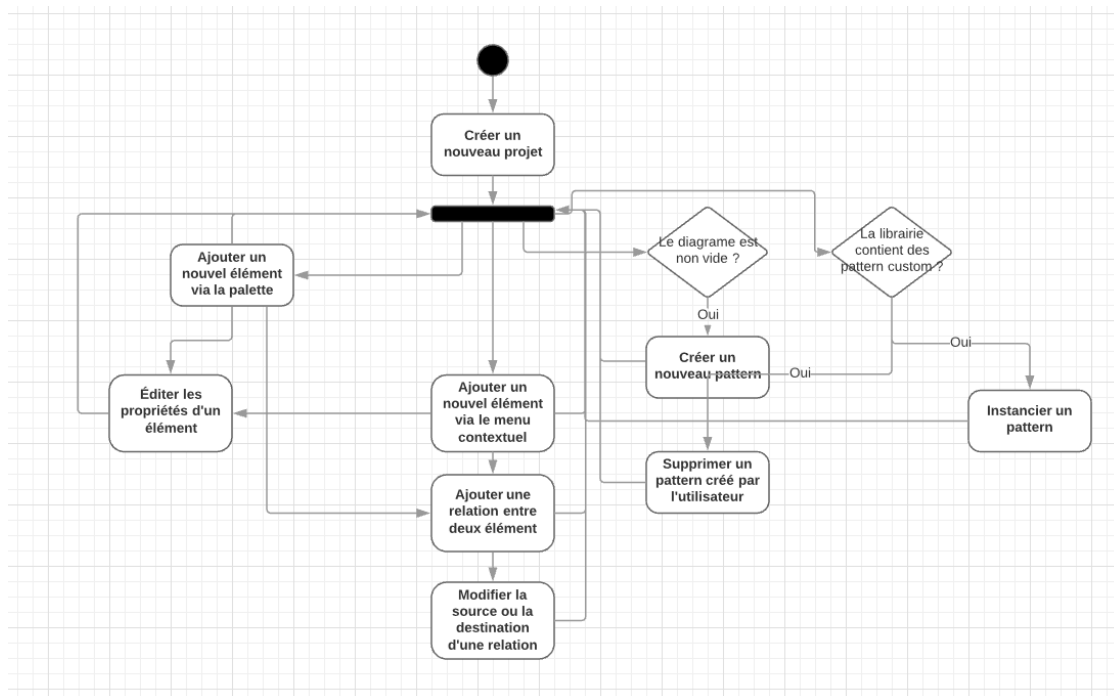


Figure 1: Diagramme de Gantt

Choix technologiques

Ce projet est réalisé sur l’environnement de développement Eclipse en utilisant l’outil de développement “Papyrus”. Le développement de notre projet exige l’utilisation de Eclipse dans sa dernière version “Photon”.

Découpe en composants

Le diagramme ci-dessous comporte 8 packages définis sous forme de plugins Eclipse.

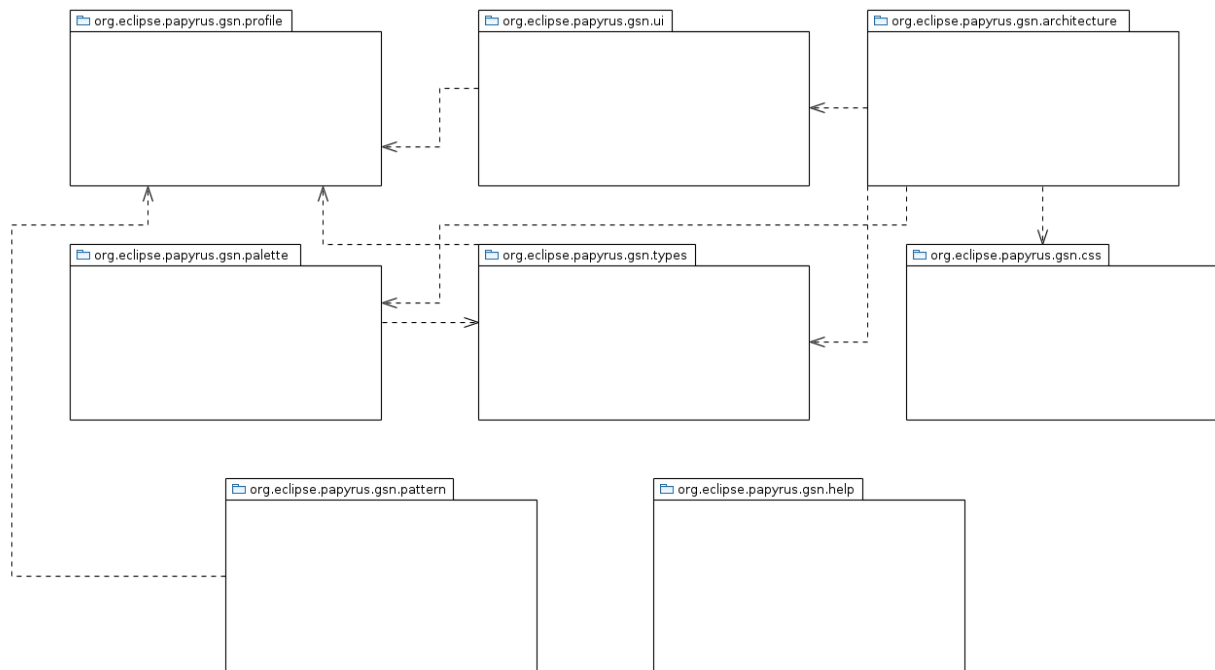


Figure 2: les packages sous forme de plugin

org.papyrus.gsn.architecture : Ce plugin permet de lier les différents composants sous une architecture contenant un point de vue, un contexte, la palette, les types, le CSS et l’UI

org.papyrus.gsn.profile : Ce plugin contient le métamodèle qui définit les stéréotypes du langage GSN et le code généré à partir du modèle qui est utilisé pour la vérification des contraintes.

org.papyrus.gsn.types : Ce composant contient les “ElementTypes sémantiques” et “ElementTypes graphiques”.

org.papyrus.gsn.ui : Ce package implémente la partie nécessaire à l’utilisation de la vue *Propriétés*.

org.papyrus.gsn.pattern: Ce plugin consiste à produire une librairie de pattern réutilisable par l'utilisateur.

org.papyrus.gsn.palette: Ce plugin définit la palette des stéréotypes du langage GSN.

org.papyrus.gsn.css: Ce plugin contient les formes qui représentent les stéréotypes ainsi qu'un template css qui applique des styles aux stéréotypes.

org.papyrus.gsn.help: Ce plugin contient la partie documentation du projet.

WBS de la réalisation

Diagramme de WBS

Le diagramme ci-dessous représente les différentes étapes de définition de l'éditeur graphique.

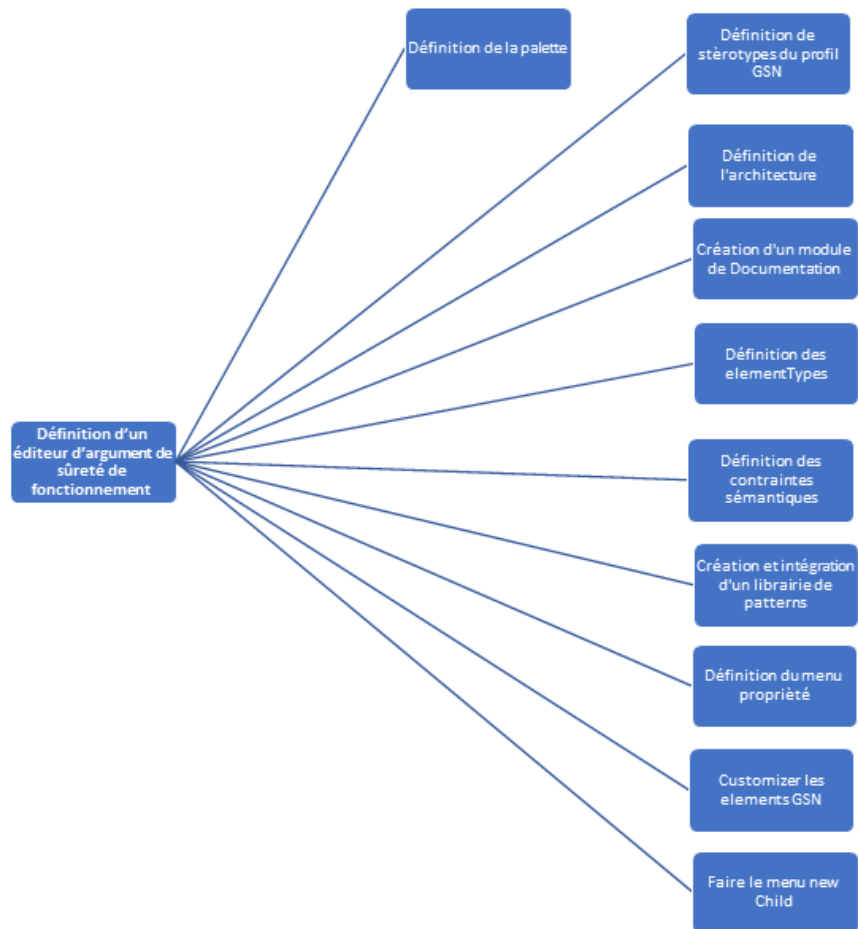


Figure 3: WBS

Description des composants et tâches de la WBS

Définition des stéréotypes dans un profil GSN

Le projet se base essentiellement sur un profil UML permettant de représenter les éléments du langage GSN dans un diagramme UML. Les stéréotypes sont répartis en trois packages décrits ci-dessous:

Core: *Goal, Solution, Context, Strategy, Justification, Assumption.*

Chacun de ces éléments est doté des propriétés suivantes:

- Identifier (String): identifiant de l'élément
- Statement (String): déclaration explicitant le rôle de l'élément dans l'argument
- Uninstanciated (Boolean): indique que l'élément concerné est un élément "générique" destiné à être remplacé par un élément concret.

Les éléments *Goal* et *Strategy* sont par ailleurs dotés de la propriété suivante:

- Undevelopped (Boolean): indique que l'élément concerné ne sera délibérément pas développé.

Tous ces éléments héritent de la métaclasse UML *Class*.

Associations: *InContextOf, SupportedBy, OptionnalSupportedBy, N-arySupportedBy, OptionnalInContextOf, N-aryInContextOf.*

Tous ces éléments héritent de la méta classe UML *Association*.

ModularExtensions: *AwayGoal, AwaySolution, AwayContext, ModuleReference, ContractModuleReference, Module* et *ContractModule*.

Les éléments *AwayGoal, AwaySolution, AwayContext, ModuleReference* et *ContractModuleReference* héritent de la métaclasse UML *Class*, et les éléments *Module* et *ContractModule* héritent de la méta classe UML *Package*.

Le stéréotype *CoreElement* est un parent des éléments du package "Core", il étend la métaclasse UML *Class* et permet l'implémentation des propriétés communes à tous les éléments. Les stéréotypes *AwayGoal, AwaySolution, AwayContext, ContractModuleReference* et *ModuleReference* étendent aussi le stéréotype *CoreElement*.

Par ailleurs, le stéréotype *ContractModule* est une généralisation du stéréotype *Module*.

Représentation des éléments de base du langage GSN

Classes de base :



Figure 4: les différentes classes de base du langage GSN

Associations de base:

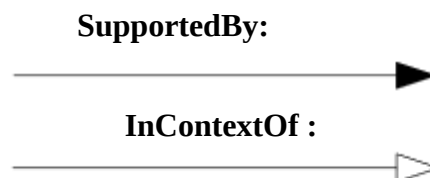


Figure 5: les associations du langage GSN

Définition des *ElementTypes*

Cette partie du projet permet de faire le binding entre les stéréotypes définis dans le modèle et les formes graphiques qui permettent la création des éléments. Il forme la partie sémantique qui sera utilisée par le menu “New Child” et la “Palette”.

La définition de l'elementType se base sur le langage XML et permet de regrouper des informations sur un élément.

Menu *New Child*

Afin d’avoir un menu qui permet de créer les éléments, nous avons défini les éléments du langage GSN en utilisant les *ElementTypes* afin de bien appliquer les stéréotypes du langage aux objets créés.

Menu *Propriétés*

Lors de la création du modèle, nous avons défini les propriétés qui seront par la suite utilisées. L’utilisation de ces propriétés nécessite la génération automatique d’un fichier .ctx qui regroupe l’ensemble des stéréotypes ainsi que leurs propriétés.

Les propriétés seront utilisées par l’utilisateur afin de décrire un élément GSN.

Palette

Cette partie du projet consiste à définir les éléments GSN dans une palette utilisable par l’utilisateur afin de modéliser son propre diagramme. La définition de cette fonctionnalité se décompose en deux parties :

- **La définition des “ *ElementTypes graphiques* ”:** Les *ElementTypes* graphiques servent à lier la partie graphique avec la partie sémantique. En se basant sur l’*ElementType* sémantique, nous avons créé pour chaque objet GSN des éléments qui permettent de faire le lien avec les formes utilisés pour l’éditeur graphique. Plusieurs spécialisations sont utilisées tels que “Class_Shape”, “Class_Shape_CN”, “Metaclass_Shape”, “Metaclass_Shape_CN”, “Class_NestedClassifierLabel”, “Edge” afin d’assurer l’affichage du stéréotype dans le diagramme.
- **La définition de la “ palette ”:** A l’issue de la première étape, une liste d’éléments créés sont définis afin de donner la possibilité à l’utilisateur de choisir un objet GSN et l’insérer dans le diagramme.

Vues graphiques en CSS

Afin de modéliser les éléments GSN graphiquement, nous avons utilisé le langage CSS, ce dernier sous Papyrus est assez particulier. Pour créer les formes décrites dans la norme, il faut utiliser des images vectorielles à l'aide des images SVG.

Fichier *architecture*

Cette partie du projet consiste à regrouper les modules dans un type précis de contexte afin de permettre à l'utilisateur l'utilisation du diagramme "Goal Structuring Notation" qui utilise le profil "GSN". L'utilisateur sera capable par la suite d'ajouter des éléments à travers la palette ou le menu contextuel "new child". Il peut également modifier les propriétés d'un élément.

Vérification des contraintes

Le langage GSN impose certaines restrictions concernant le types des entités qui peuvent être reliées par des associations. Il est donc nécessaire de vérifier dynamiquement la validité des associations lors de leur création ou de la modification d'une de leurs extrémités.

Dans le cas d'une association de type *InContextOf*, les combinaisons valides sont:

Goal -> Context

Goal -> Assumption

Goal -> Justification

Goal -> AwayGoal

Goal -> AwayContext

Goal -> Module

Strategy -> Context

Strategy -> Assumption

Strategy -> Justification

Strategy -> AwayGoal

Strategy -> AwayContext

Strategy -> Module

Dans le cas d'une association de type *SupportedBy*, les combinaisons valides sont:

Goal -> Goal

Goal -> Strategy

Goal -> Solution

Goal -> AwayGoal

Goal -> AwaySolution

Goal -> Module

Goal -> ContractModule

Strategy -> Goal

Strategy -> AwayGoal

Strategy -> AwaySolution

Strategy -> Module

Strategy -> ContractModule

L'optionalité et la multiplicité des associations n'ont aucune incidence sur les contraintes à respecter.

Les classes Java réalisant ces vérifications sont situées dans le package **validators** du plugin **org.eclipse.papyrus.gsn.profile**.

Bibliothèque de patterns

Cette bibliothèque servira à faciliter le travail de l'utilisateur. Elle comporte différents modèles de cas d'assurance déjà implémenter et connu. La bibliothèque est intégrée au projet sous forme de plugin.

L'utilisateur pourra charger un modèle à partir de cette bibliothèque et l'étendre, le modifier ou supprimer un élément en fonction du besoin sans modifier les patterns de base, pour cela il devra créer un projet, ensuite, clique droit et importer des packages.

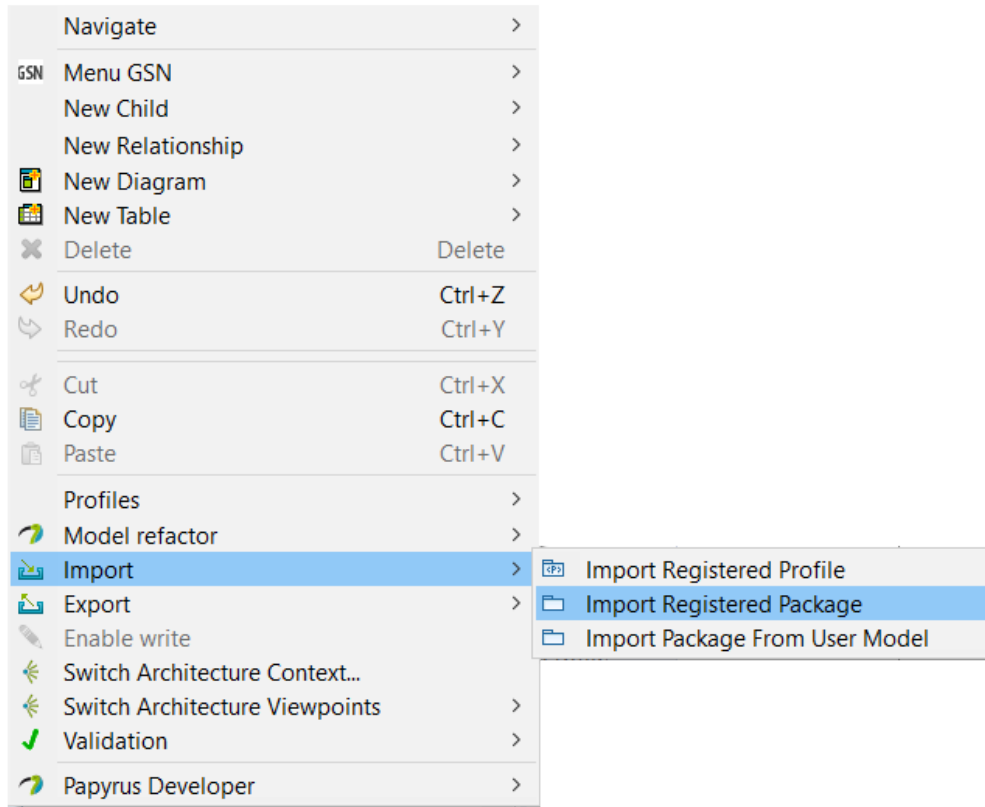


Figure 6: Importation des packages de patterns

Ensuite, il devra choisir les patterns de GSN “GSNPatterns” à importer

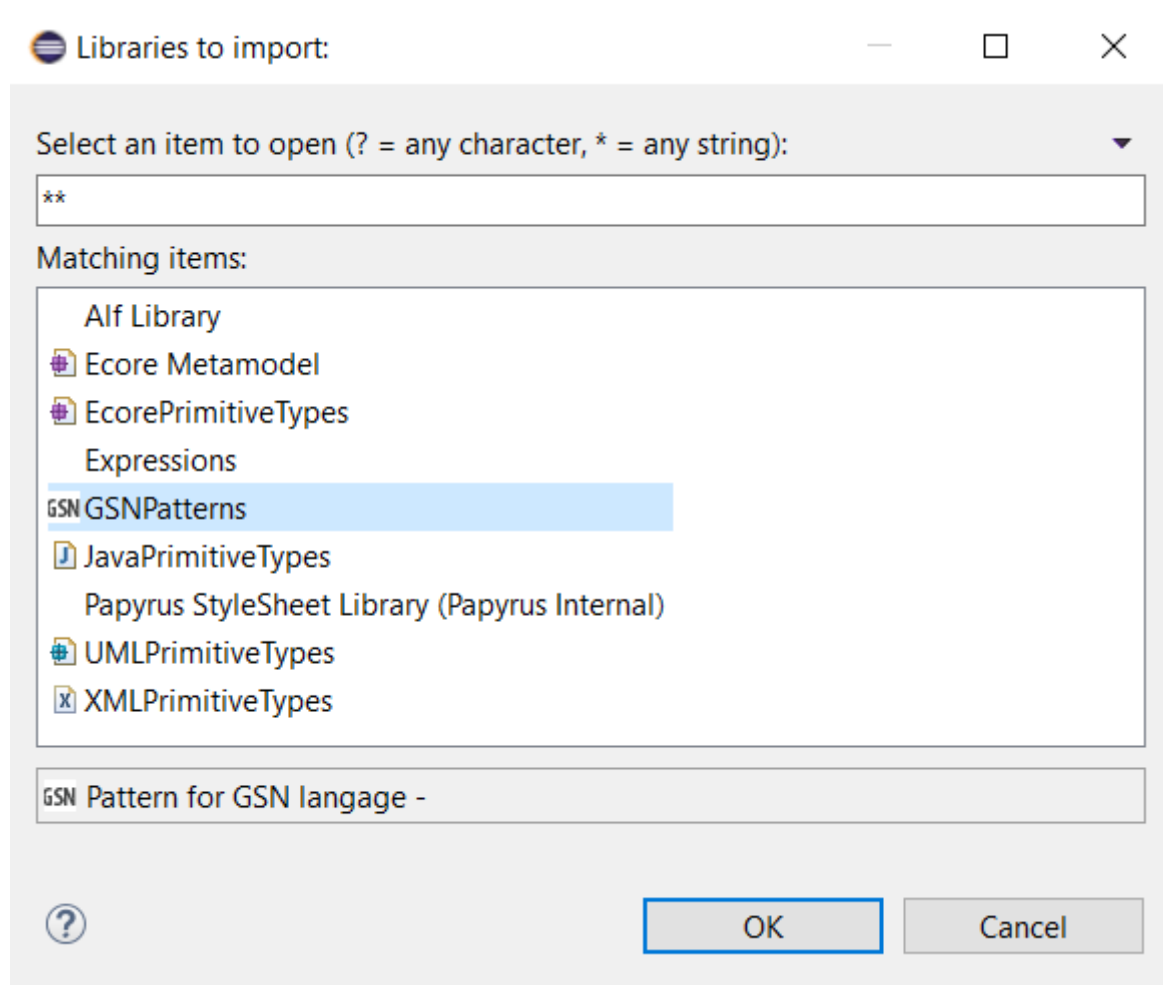


Figure 7: Choix des patterns GSN

Dans les patterns importés, plusieurs packages de patterns sont possibles, en fonction des besoins de l'utilisateur

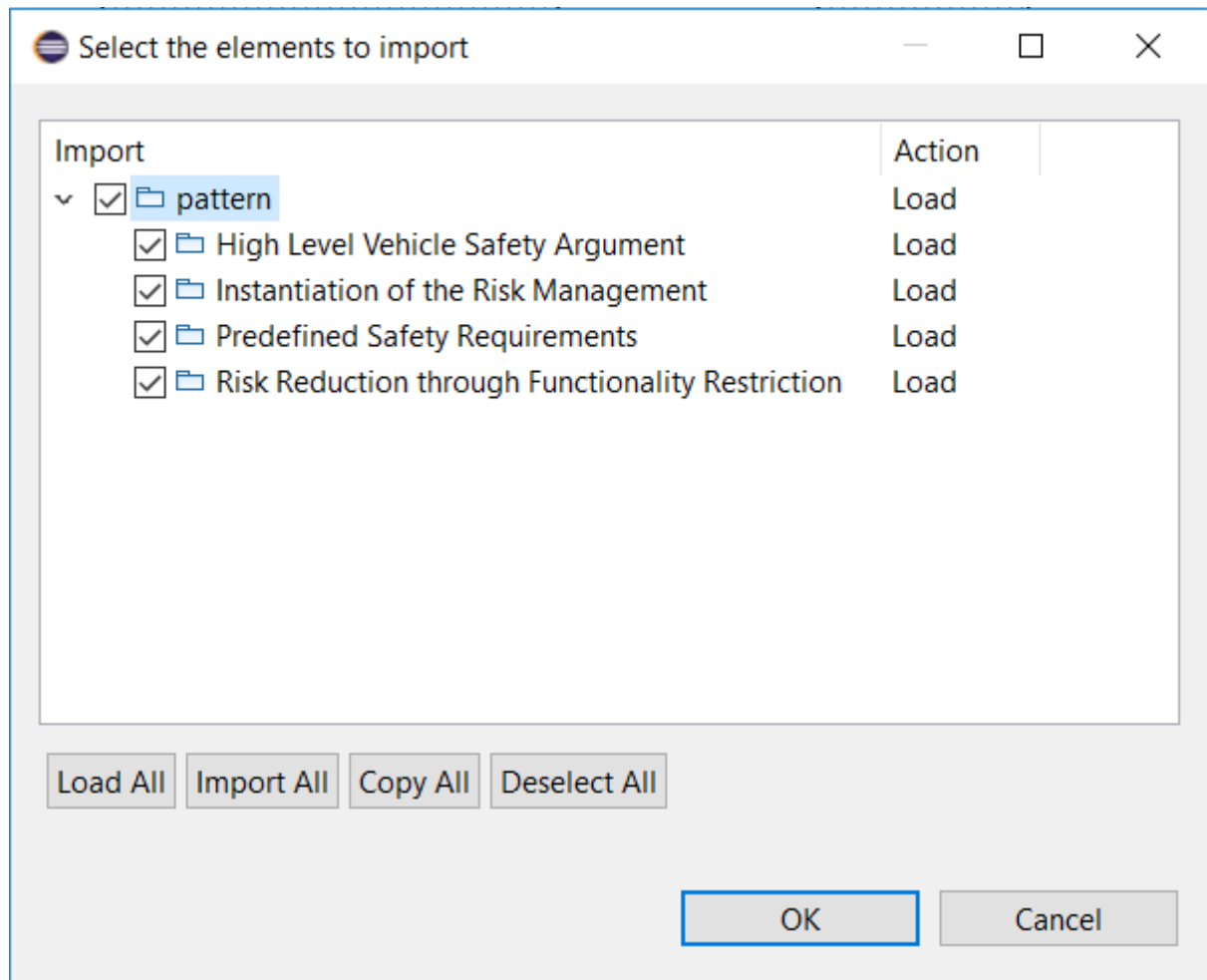


Figure 8: chargement des patterns GSN

On peut vérifier ensuite dans le “Model Explorer” que les packages ont bien été importés

- > test
- > «ModelLibrary» EcorePrimitiveTypes
- ▼ pattern
 - Class Diagram
 - > <Package Import> UML Primitive Types
 - > High Level Vehicle Safety Argument
 - > Instantiation of the Risk Management
 - > Predefined Safety Requirements
 - > Risk Reduction through Functionality Restriction
 - > «EPackage, ModelLibrary» PrimitiveTypes

Figure 9: Les patterns dans le “Model Explorer”

A cette étape, l'utilisateur doit copier le package du pattern voulu dans le "Model Explorer" du projet, ainsi il pourra accéder aux propriétés de modification et d'extension du diagramme, car les patterns ne sont pas modifiables directement.

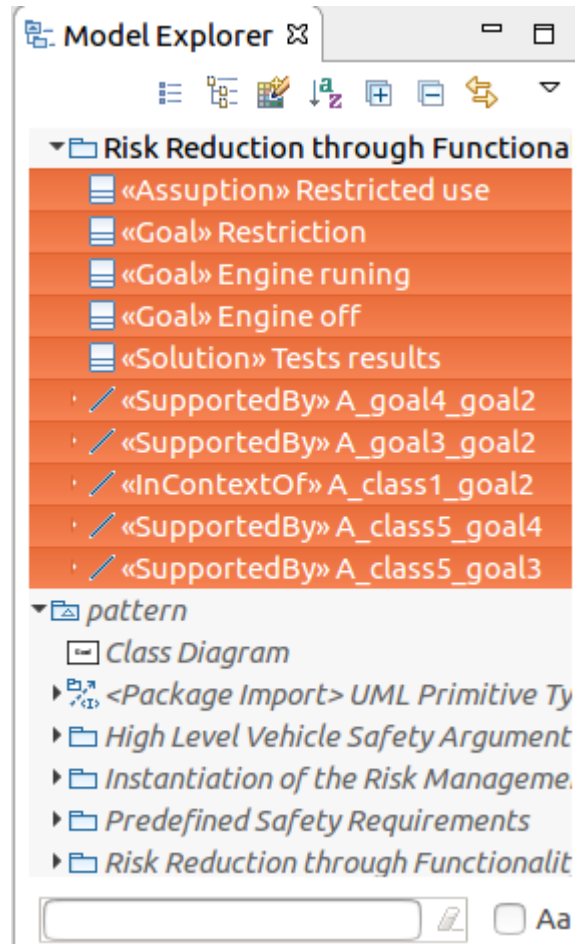


Figure 10: Copie des éléments du pattern dans le projet

En faisant un "drag and drop" des éléments du pattern dans l'instance du projet, le diagramme sera chargé

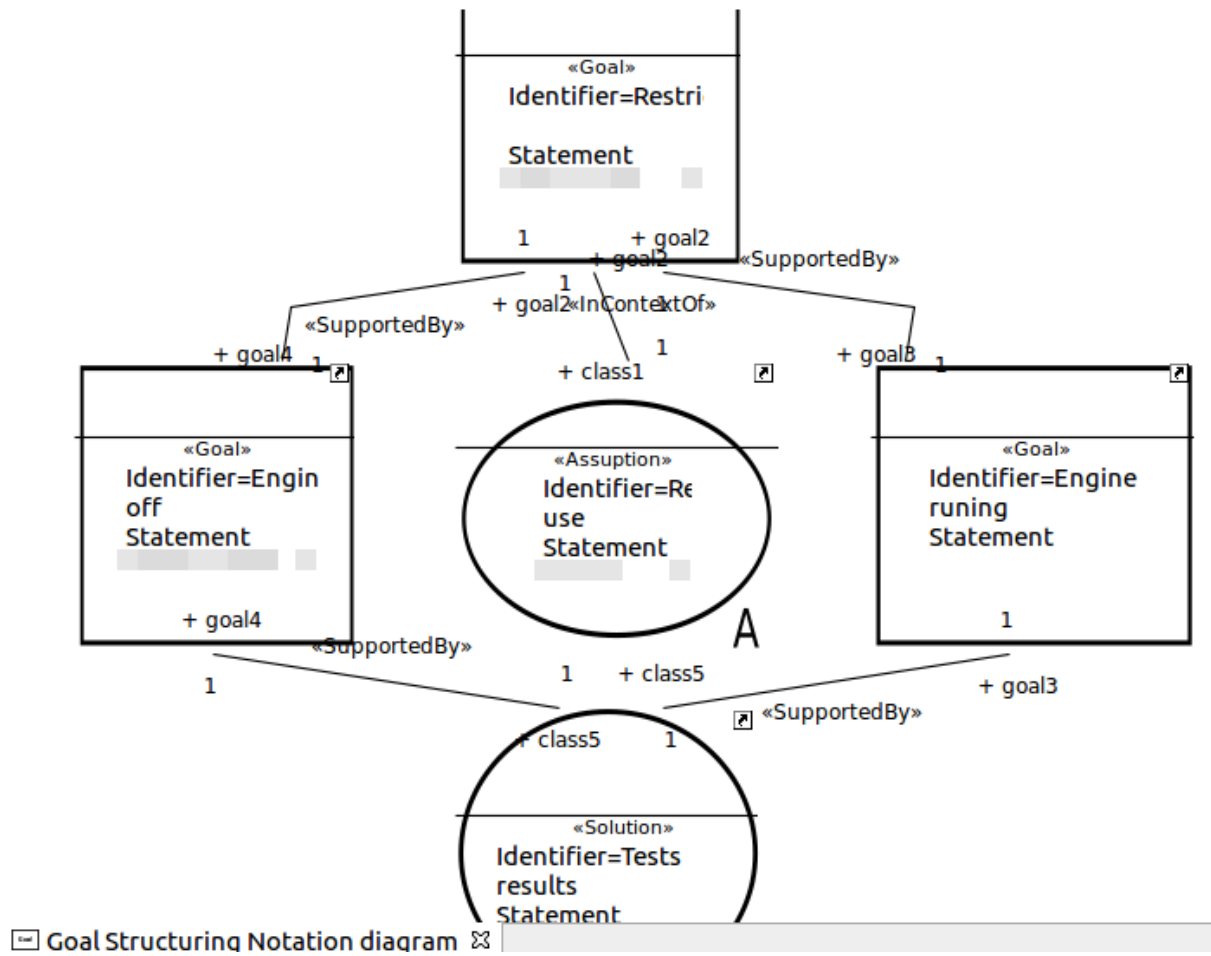


Figure 11: Pattern chargé dans l'instance du projet

Plan d'intégration

Dépendance entre composants

Tableau 1: Dépendance entre les composants

	Profile	UI	Types	Architecture	Pattern	CSS	Palette
Profile	-	-	-	-	-	-	-
UI	X	-	-	-	-	-	-
Types	X	-	-	-	-	-	-
Architecture	X	X	X	-	-	X	X
Pattern	X	-	-	-	-	-	-
CSS	-	-	-	-	-	-	-
Palette	-	-	X	-	-	-	-

Type de dépendance

UI > Profile: La partie “UI” qui correspond au menu des propriétés des éléments GSN nécessite la définition des attributs dans la classe du modèle, ainsi un fichier .ctx sera généré à partir du modèle pour appliquer les propriétés.

Types > Profile: La partie “Types” qui permet de définir les élémentTypes sémantiques et graphiques nécessite la partie profil qui implémente les stéréotypes et les classes java nécessaire pour le binding entre l'élémentType et le modèle.

Pattern > Profile: Le plugin pattern utilise le modèle définit dans le profile.

Palette > Types : La définition des *ElementTypes* graphiques et sémantiques sont nécessaires pour faire fonctionner la palette.

Architecture > Profil, UI, CSS, Types, Palette: L'architecture est l'élément de base de notre éditeur graphique. Elle regroupe les parties définit dans un diagramme ou l'utilisateur peut ajouter des éléments du langage à travers la palette. Afin d'avoir une sémantique graphique

qui correspond au langage GSN, La partie CSS permet de détecter les stéréotypes et de les customiser. Ainsi la manipulation d'un élément du langage nécessite la modification des propriétés.

Compromis de réalisation liés à la technologie

Papyrus ne propose pas de moyen simple de personnaliser l'affichage des associations. Nous n'avons donc pas pu respecter à la lettre le standard GSN. Voici une liste exhaustive des ajustements pour les différents types d'associations:

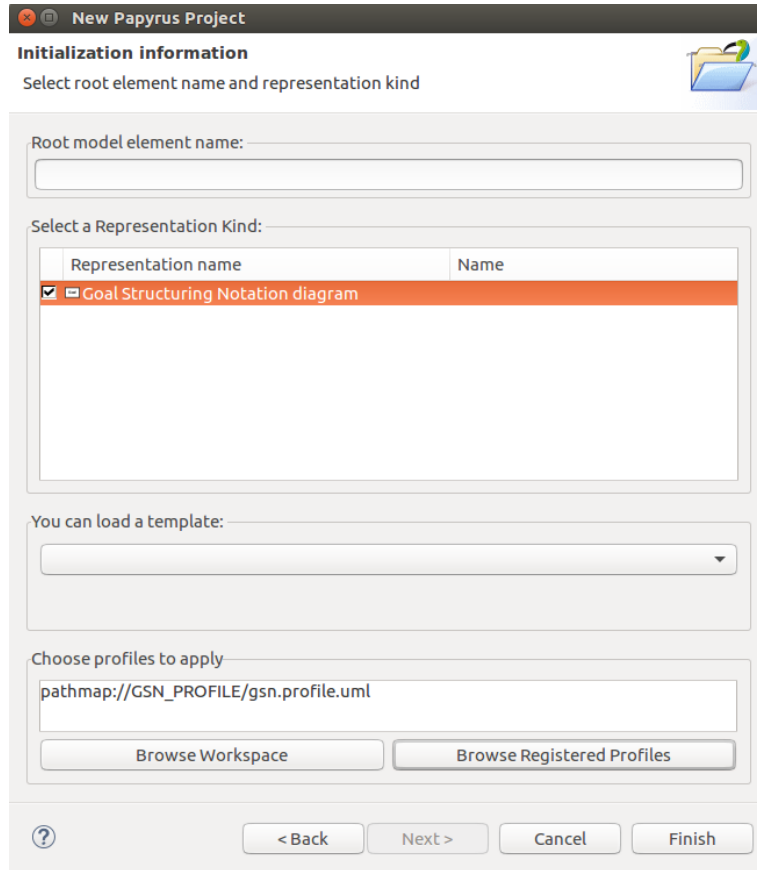
- Les associations sont affichées sous la forme d'associations UML classiques. Le nom du stéréotype servant alors à distinguer les différentes associations
- Du fait de la contrainte évoquée plus haut, l'arité des associations n-aires n'est pas affichée. Par ailleurs, le caractère optionnel des associations est indiqué en utilisant un stéréotype différent (*OptionnalInContextOf* ou *OptionnalSupportedBy*) plutôt que via une propriété booléenne.
- Le losange normalement présent à l'intersection des associations n-aires n'est pas affiché



Figure 12: Les types d'associations

Nomenclature de l'IHM

Notre interface IHM sera un "Goal Structuring Notation diagram".



New Papyrus Project

Initialization information

Select root element name and representation kind

Root model element name:

Select a Representation Kind:

Representation name	Name
<input checked="" type="checkbox"/> Goal Structuring Notation diagram	

You can load a template:

Choose profiles to apply

Figure 13: Interface IHM

Le diagramme doit permettre à l'utilisateur d'utiliser la palette afin de modéliser les diagrammes d'assurance et modéliser les arguments afin d'approuver ses systèmes.

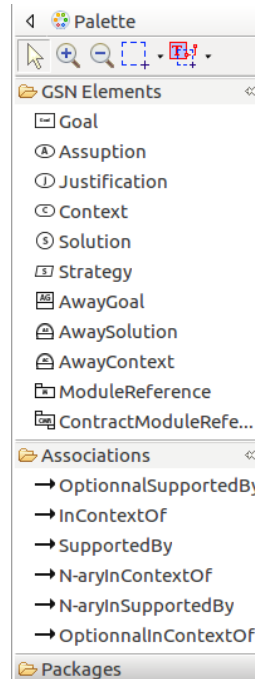


Figure 14: Palette

En faisant un Drag-and-Drop, l'utilisateur peut ajouter un ou plusieurs éléments et faire des associations entre eux.

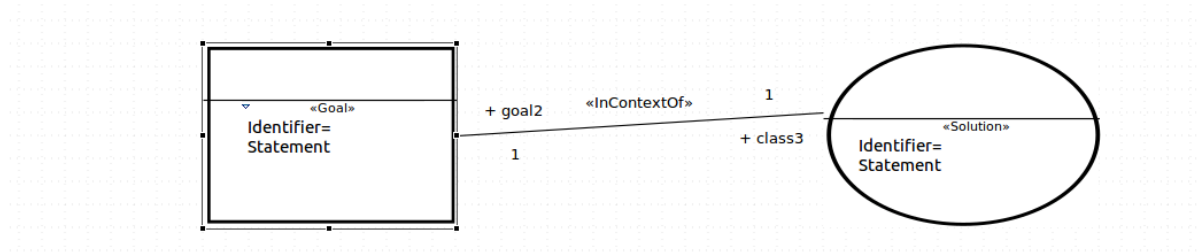


Figure 15: Exemple d'association

En sélectionnant un objet “GSN”, nous pouvons modifier les propriétés dans le menu GSN

UML	Identifier	<input type="text"/>
Comments	Statement	<input type="text"/>
Profile	Undeveloped	<input type="radio"/> true <input checked="" type="radio"/> false
Style	Uninstantiated	<input type="radio"/> true <input checked="" type="radio"/> false
Appearance		
Rulers And Grid		
Advanced		
GSN Properties		

Figure 16: Propriétés GSN

Enfin, l'utilisateur utilise le diagramme, qui ressemble à la figure ci-dessous

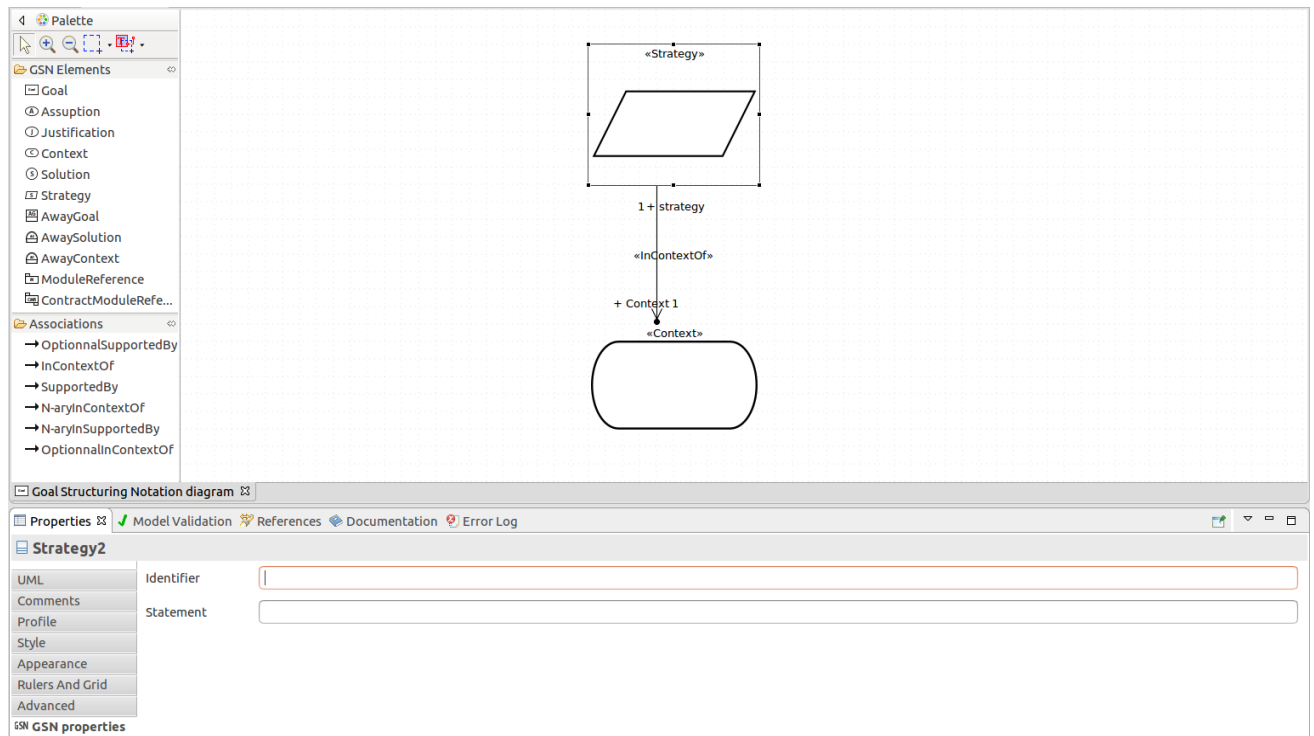
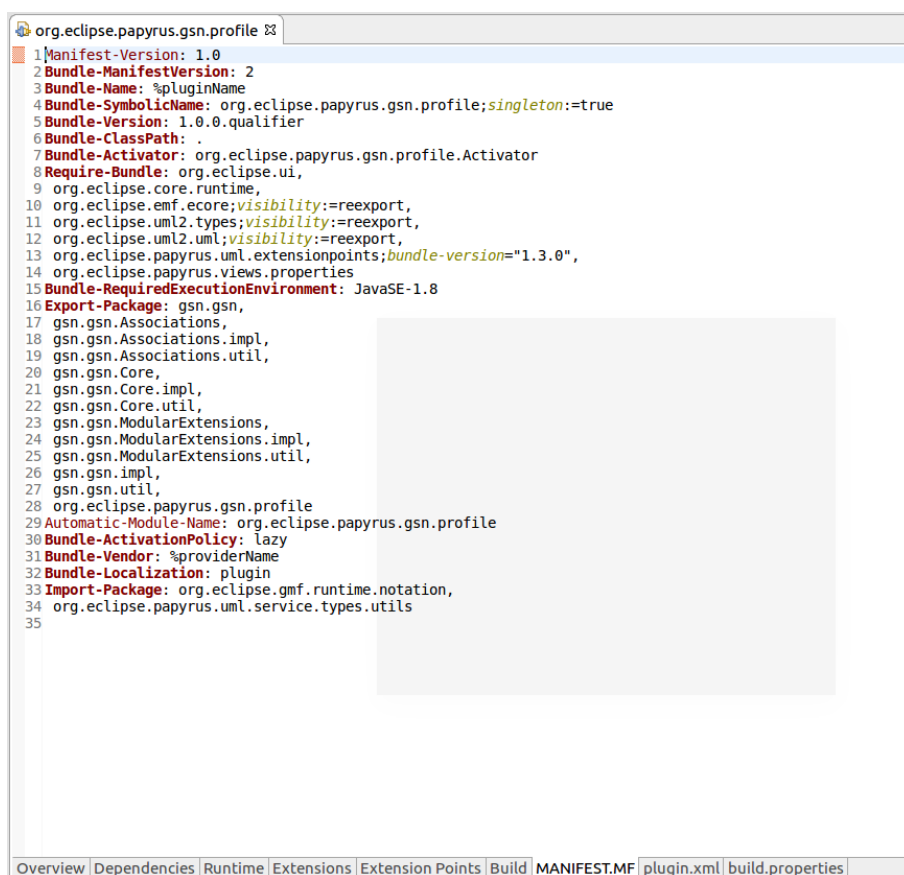


Figure 17: Exemple de diagramme

Plan de déploiement et mise en exploitation

Plugin et configuration

Les composants de notre système suivent une architecture modulaire. Chaque package est un plugin à part. Chaque projet se compose d'un fichier de configuration "plugin.xml" qui définit les extensions nécessaires pour le module ainsi qu'une classe "Activator.java" qui permet l'activation du projet.



```

1 Manifest-Version: 1.0
2 Bundle-ManifestVersion: 2
3 Bundle-Name: %pluginName
4 Bundle-SymbolicName: org.eclipse.papyrus.gsn.profile;singleton:=true
5 Bundle-Version: 1.0.0.qualifier
6 Bundle-ClassPath: .
7 Bundle-Activator: org.eclipse.papyrus.gsn.profile.Activator
8 Require-Bundle: org.eclipse.ui,
9 org.eclipse.core.runtime,
10 org.eclipse.emf.ecore;visibility:=reexport,
11 org.eclipse.uml2.types;visibility:=reexport,
12 org.eclipse.uml2.uml;visibility:=reexport,
13 org.eclipse.papyrus.uml.extensionpoints;bundle-version="1.3.0",
14 org.eclipse.papyrus.views.properties
15 Bundle-RequiredExecutionEnvironment: JavaSE-1.8
16 Export-Package: gsn.gsn,
17 gsn.gsn.Associations,
18 gsn.gsn.Associations.impl,
19 gsn.gsn.Associations.util,
20 gsn.gsn.Core,
21 gsn.gsn.Core.impl,
22 gsn.gsn.Core.util,
23 gsn.gsn.ModularExtensions,
24 gsn.gsn.ModularExtensions.impl,
25 gsn.gsn.ModularExtensions.util,
26 gsn.gsn.impl,
27 gsn.gsn.util,
28 org.eclipse.papyrus.gsn.profile
29 Automatic-Module-Name: org.eclipse.papyrus.gsn.profile
30 Bundle-ActivationPolicy: lazy
31 Bundle-Vendor: %providerName
32 Bundle-Localization: plugin
33 Import-Package: org.eclipse.gmf.runtime.notation,
34 org.eclipse.papyrus.uml.service.types.utils
35

```

Figure 18: classe "Activator"

Le plugin de documentation

Notre projet contiendra un plugin de documentation qui aidera les utilisateurs à suivre les étapes nécessaires pour définir un diagramme d'argumentation de sûreté. Le manuel utilisateur sera accessible depuis le menu *Help* de l'IDE Eclipse et comportera :

- Une description des éléments GSN
- Une liste exhaustive des contraintes relatives aux associations
- Un didacticiel expliquant l'usage de la vue propriété, du menu *New Child* et de la palette.
- Un didacticiel montrant les différentes étapes de la création d'un nouveau projet de modélisation utilisant le profil GSN.
- Un didacticiel expliquant l'utilisation de la librairie de patterns.

Kit de livraison

Le kit de livraison comportera :

- *Projet*
- *STBE*
- *STR*

Conclusion

Nous tenons avant tout à remercier notre client pour son encadrement, son assistance et le temps qu'il nous a consacré tout au long de ce projet, sachant répondre à toutes nos interrogations.

Cette STR résume l'ensemble des tâches à réaliser. Ce document présente les différents composants de notre solution, une description des plugins, des diagrammes, une présentation de l'IHM et une description du déploiement.

Ce projet a été une occasion de connaître les enjeux des projets professionnels. Cette expérience bénéfique pour nous a été un milieu d'autoévaluation et d'apprentissage de l'environnement "Papyrus".

Ontologie

GSN: « Goal Structuring Notation » est un langage qui permet d'argumenter graphiquement afin de créer une assurance sur des propriétés critiques d'un système.

SACM2 :« Structured Assurance Case Metamodel » est une spécification qui définit un métamodèle pour la représentation des cas d'assurance structurés.

Assurance case : Un cas d'assurance est un ensemble d'arguments et preuves créés pour appuyer l'affirmation selon laquelle un système défini répondra aux exigences particulières

Pattern : Un pattern est un modèle qui décrit une solution standard utilisable dans la conception de différents "assurance case"

Profile : Un profil permet de définir les stéréotypes d'un modèle donnée

ElementType : C'est la partie qui permet de définir les sémantiques de stéréotypes et qui assure un binding entre le modèle et la palette

New child : C'est un menu utilisé pour ajouter des éléments du langage