SysML par la pratique avec Papyrus

Jean-Michel Bruel, Sébastien Gérard

Version 1.6, 2019-12-12

Table des Matières

Commentaire

Si vous lisez ce commentaire, c'est que cette version est en cours de construction!

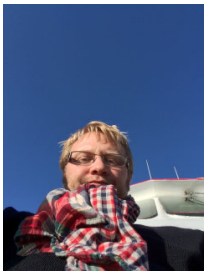
Reste à faire :

- ▢ Changer le site du livre par https://github.com/PapyrusSysMLinAction/SmartHomeUseCase
- ▢ Ajouter le lien vers le standalone de Papyrus-SysML si disponible
- ▢ Revoir les commentaires de Seb (Cadres, Outils, etc.)

À propos...

Des auteurs

Jean-Michel Bruel est Professeur des Universités à l'Université de Toulouse, en poste à l'IUT de Blagnac depuis 2008. Il a été Maître de Conférences à l'Université de Pau et des Pays de l'Adour de 1997 à 2008 après avoir obtenu son doctorat en informatique de l'Université Toulouse III - Paul Sabatier en 1996. Il est actuellement responsable du site Université Toulouse II - Jean Jaurès de l'IRIT et de l'équipe SM@RT. Il est l'un des co-fondateurs de l'association SysML-France (depuis intégrée à l'AFIS, membre du comité éditorial de la revue *SoSyM* depuis sa création en 2002, membre du *Steering Committee* de la conférence ACM/IEEE MODELS de 2008 à 2017, chef du département informatique de l'IUT de Blagnac de 2009 à 2012, co-responsable de l'axe Systèmes Ambiants de l'IRIT de 2009 à 2018 et co-animateur du comité technique MBSE de l'AFIS. Il enseigne la modélisation depuis 1995.



Sébastien Gérard est ... (Papa de Papyrus :-)

Historique

Ce document est la compilation de plusieurs années d'enseignement de SysML® ou de Papyrus-SysML par les auteurs depuis plus de dix ans, que ce soit :

- au Master TI, de l'Université de Pau et des Pays de l'Adour (avec Nicolas Belloir),
- au Master Recherche SAID, de l'Université Toulouse III - Paul Sabatier,
- au Master ICE de l'Université Toulouse II - Jean Jaurès (avec Pierre de Saqui-Sannes),
- au *Master of Science* de Göteborg, Suède (introduction réalisée par Nicolas Belloir),
- à Universitad Autónoma de Guadalajara, au Mexique (40h de formation professionnelle à des employés de Continental Mexique),

To be completed by Seb!!!

- ou plus récemment au Master DL de l'Université Toulouse III - Paul Sabatier (avec Christelle Chaudet) ou du master ICE de l'Université Toulouse II - Jean Jaurès (avec Benoît Combemale).

Vous trouverez en référence (cf. *Références*) les ouvrages et autres documents utilisés et affectionnés par les auteurs.

Remerciements

Commentaire

À faire au dernier moment !

Penser à remercier : Raphaël Faudou, Pascal Roques, Nicolas Belloir, Nicolas Hili, Jo, ...

Chapter 1. Avant-propos

1.1. Ce que n'est pas ce livre :

Ce livre **n'est pas** :

- un livre de référence sur SysML® (pour cela, même s'il est en anglais, nous conseillons [Friedenthal2016])
- un livre sur la meilleure approche méthodologique possible avec SysML®. Il n'en existe pas. Une méthode est bonne si elle vous est adaptée, en particulier si elle prend en compte les besoins et les contraintes de votre niveau d'experise, ou encore de vos préoccupations.
- un livre pour réviser l'examen de certification de l'OMG™ ou de l'INCOSE, comme [OCSMP] (mieux vaut vous plonger dans la spécification de référence [SysML])

1.2. Pourquoi lire ce livre ?

Il peut y avoir plusieurs raisons de lire ce livre, même partiellement. Vous pouvez vous reconnaître dans l'une des situations suivantes :

Je souhaite améliorer mon processus d'ingénierie système Ingénierie Système

Vous êtes un professionnel dans le domaine de l'ingénierie système et vous êtes intéressé par la modélisation. Nous espérons que les conseils pratiques, appliqués avec l'outil Papyrus-SysML, vous permettront de tester l'intérêt de SysML® pour votre domaine.

Je souhaite m'initier à SysML® par la pratique

Ce livre est entièrement réalisé dans cet objectif : apprendre les concepts principaux et la notation du langage en les mettant en oeuvre. Nous allons à l'essentiel des concepts et surtout nous vous permettons de les manipuler de manière concrète dans l'outil Papyrus-SysML au travers d'un exemple concret.

Je souhaite savoir utiliser Papyrus-SysML

Les outils logiciel prennent une part de plus en plus importante dans le développement de systèmes complexes. Si vous utilisez déjà Papyrus comme modeleur UML®, et que vous souhaitiez apprendre SysML® en conservant vos habitudes, ce livre, réalisé en collaboration avec l'équipe du CEA LIST en charge du développement de Papyrus-SysML, devrait vous y aider.

Je souhaite utiliser des solutions *open-source* pour l'Ingénierie Système

C'est un des avantages de Papyrus-SysML : c'est un logiciel ouvert, auquel de nombreux programmeurs contribuent, permettant ainsi la prise en compte rapide des remarques utilisateurs, la disponibilité d'un forum riche, etc.

Je souhaite utiliser des normes d'ingénierie systemeIngénierie Système

Si, comme il est généralement préconisé en Ingénierie Système, vos processus et vos outils s'appuient sur des normes et des standards de fait (cf. Figure 2), vous aurez avec SysML®, une notation standard pour vos modèles (apprise en filière technologique STI-2D des Lycées, en classes préparatoires et dans la plus part des d'écoles d'ingénieurs et des universités et cela a

l'échelle du mon entier). Et vous aurez, avec Papyrus-SysML, un outils 100% conforme à la spécification.

Je souhaite appliquer la démarche CESAM en utilisant la notation SysML®

CESAM est une démarche très répandue en France, et c'est une des raisons qui nous ont poussé à utiliser cette démarche comme exemple dans cet ouvrage.

Je souhaite préparé un cours ou une formation pratique à SysML®

Blala…

Je souhaite gagner du temps

Nous avons essayé de concevoir ce livre comme un manuel[1] pratique, en organisant les concepts et les pratiques. Nous fournissons également un site web d'accompagnement à cet ouvrage contenant les liens pratiques, modèles, exemples et références utilisés dans cet ouvrage (cf. https://github.com/jmbruel/sysmlpapyrusbook).

1.3. À qui s'adresse ce livre ?

En fonction de votre profil, vous pourrez avoir une lecture plus ou moins complète de cet ouvrage. Loin de nous l'idée de forcer la façon d'en aborder les chapitres, nous vous recommandons néanmoins les pistes de lectures suivantes selon la situation :

Table 1. Conseils de lecture en fonction de votre niveau en SysML (en ligne) et en Papyrus (en colonne)

SysML® / Papyrus-SysML	Découverte	Initié	Professionnel
Découverte	2,3,4,5,6,7,8,9,10,11	2,3,(4),5,6,(7),8,9,10,11	2,3,5,6,8,9,10,(11)
Initié	(2),3,4,(5),6,7,8,9,10,11	(2),3,(4,5),6,(7),8,9,10,11	(2),3,(5),6,8,9,10,(11)
Professionnel	(2),3,4,(6),7,(8,10),11	(2),3,(4,6,7,8,10),11	(2),3,(4,6,7,8,10,11)

Si vous êtes débutant en Ingénierie Système, nous vous conseillons en parallèle de vous initier. Le livre n'aborde pas directement les concepts systèmes.

Ce document a été réalisé de manière à être lu de préférence dans sa version électronique, ce qui permet de naviguer entre les références et les renvois interactivement, de consulter directement les documents référencés par une URL, etc.

Si vous lisez la version papier de ce document, ses hyperliens clickables ne vous servent à rien, mais n'hésitez pas à en consulter la version électronique!

Commentaire

⇒ Proposer des pistes de « formation » pour rentrer dans le cadre le cas échéant.

⇒ Description de la mise en place de l'environnement pour la mise en pratique.

⇒ Fournir un petit outillage Papyrus dédié à la mise en pratique du livre (Spécialisation particulière, projet type, exemple de modèle, cheat sheet, etc.) que l'on pourrait mettre à disposition via le site de Papyrus www.eclipse.org/papyrus/

Niveaux de concepts SysML

Les différents niveaux de concepts SysML® que nous utilisons dans ce livre sont alignés sur ceux du programme de certification de l'OMG™ [OCSMP].

1.4. Conventions typographiques et symboles récurrents

Nous avons défini et utilisé un certain nombre de conventions pour rendre ce document le plus agréable à lire et le plus utile possible (grâce notamment à la puissance d'AsciiDoc) :

- des mises en formes particulières (e.g., `NomDeBloc` pour un élément de modèle),
- des références bibliographiques (comme la spécification [SysML]), présentées en fin de document (cf. *Références*),
- tous les flottants (figures, tableaux) sont listés à la suite de la table des matière, mais les captures d'écran sont souvent sans numéro quand elles illustrent simplement des étapes d'un tutoriel par exemple.

Check this!!

- les termes anglais (souvent incontournables) sont repérés en *italique*, non pas pour indiquer qu'il s'agit d'un mot anglais, mais pour indiquer au lecteur que nous employons volontairement ces termes (e.g., *Requirements*),
- un certain nombre de symboles viennent identifier les notes :

Commentaire

Ce symbole permet de repérer rapidements des commentaires pour nous-mêmes. Il ne doit pas figurer dans la version finale en ligne, ni dans le PDF. De même XXX ces textes XXX devraient être éliminés au fur et à mesure…

Ceci est une simple note, un point remarquable.

Attention, piège ou erreur à éviter.

Ceci est un point important.

Convention : Ceci est une convention ou une bonne pratique

Dans ces notes, nous distillerons des conseils, des bonnes pratiques ou des conventions que nous recommandons d'adopter.

Définition : Exemple (OMG SysML® v1.6, p. 152)

Ces notes concernent des définitions tirées de la spécification SysML® et sont donc précisément référencées.

[1] Au sens latin du terme : "que l'on peut avoir toujours à portée de main".

Chapter 2. Introduction

SysML® est un langage de modélisation, souvent considéré comme une notation. La principale difficulté, avec SysML®, c'est que cette notation n'est associée à aucune méthode en particulier, ni aucun outil en particulier (cf. [Roques]). Il s'agit là d'une volonté délibérée des concepteurs, l'OMG™ et l'INCOSE, pour permettre la plus grande adoption de la notation en entreprise. Il vous est donc possible d'appliquer votre propre démarche, votre propre processus interne de développement. Contrairement à l'outil Capella par exemple, fortement couplé à la méthode Arcadia, l'outil Papyrus-SysML n'impose aucune démarche particulière. Nous allons aborder au Chapitre 6 les démarches connues et courantes en Ingénierie Système basé sur les Modèles généralement associés à SysML® et nous expliquerons celle que nous avons retenue pour cet ouvrage, et qui sera explicitée au Chapitre 8.

[SysML PRFC] | *SysML-PRFC.svg*

Figure 1. Langages, méthodes et outils : 3 piliers complémentaires (inspiré par Pascal Roques)

Comme le résume la figure précédente, nous allons aborder dans la pratique aussi bien les éléments de notation (SysML®), de méthodologie (CESAM) et d'outillage (Papyrus-SysML).

2.1. Pourquoi et quoi modéliser ?

2.1.1. Pourquoi modéliser ?

> Pour un observateur A, M est un **modèle** de l'objet O, si M aide A à répondre aux questions qu'il se pose sur O.
>
> — Marvin Minski, Matter, mind, and models (Semantic Information Processing 1968)

Il n'est pas dans les objectifs de ce livre d'explorer cette vaste question (voir [Jézéquel] ou [Cabot]), mais disons simplement que nous considérons la définition ci-dessus comme essentielle. Les modèles permettent de communiquer (entre personnes, entre outils) et d'appréhender la complexité des véritables systèmes en développement.

2.1.2. Différences entre modèle et dessin

Ne considérez pas SysML® comme une palette de dessins et d'éléments de base servant à faire des diagrammes. SysML® permet de définir des éléments conceptuels et d'en faire une représentation graphique. Cette dernière est importante car elle permet de communiquer visuellement sur le système en développement, mais du point de vue du concepteur, c'est **le modèle** qui importe le plus.

C'est pourquoi nous vous recommandons de ne jamais "dessiner" des diagrammes SysML® [2], mais d'utiliser des outils dédiés comme Papyrus-SysML. Ils respectent en général la norme SysML® 1.6.

Un des intérêts de la modélisation est de faciliter la communication, notamment au travers des diagrammes et leur aspect graphique et synthétique. Un dessin est donc un plus par rapport à du texte. Néanmoins, il ne faut pas se contenter d'un simple dessin pour au moins deux raisons

importantes :

- un dessin n'est pas assez formel (comment être sûr d'avoir correctement utilisé tel ou tel symbole) ;
- il est impossible d'assurer la cohérence globale des modèles dans le cas d'un dessin.

Un modèle est une sorte de base de donnée qui regroupe des éléments issues de différents points de vue (saisis le plus souvent au travers de diagrammes). Un diagramme est une vue partielle du modèle (donc incomplète). Le modèle est la vraie plus value car il va permettre de détecter les incohérences sur les exigences, les problèmes de complétude, lancer des analyses, faire des transformations vers d'autres langages ou formats, etc. Par exemple dans un outil de modélisation il y a une grande différence entre supprimer un élément d'un diagramme (on parlera alors de "masquer" un élément d'un diagramme) et supprimer un élément de modèle (ce qui aura pour effet de supprimer cet élément de tous les diagrammes où il était présent).

2.2. Démarches et méthodes de développement

Ce livre n'est pas un livre sur l'ingénierie système. Il ne s'agit donc pas pour nous de rappeler les grandes démarches de développement de système. Nous renvoyons pour cela le lecteur aux publications de l'INCOSE ou de l'AFIS.

Concernant les démarches de développement de systèmes en lien avec le MBSE, SysML® n'en impose aucune. C'est à la fois une faiblesse car cela pourrait servir de guide à l'utilisation de la notation elle-même, mais aussi une force car vous pouvez utiliser SysML® sans changer de méthode, y compris si vos développements et pratiques s'inscrivent dans le cadre d'utilisation de standards, nombreux en Ingénierie Système comme l'illustre la Figure 2.

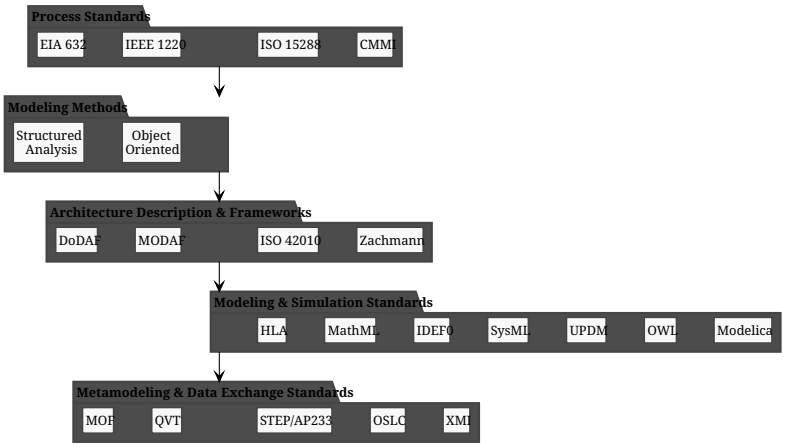


Figure 2. Taxonomie (partielle) des standards principaux en Ingénierie Système [Friedenthal2016]

Nous utiliserons dans ce livre une démarche minimaliste et très utilisée en France : CESAM. Cette méthode ainsi que les références aux démarches principales seront abordées au Chapitre 6 et l'explication de la démarche elle-même, illustrée avec des modèles SysML® réalisés avec Papyrus-SysML sera détaillée au Chapitre 8.

2.3. Matrice des concepts

Commentaire

À voir si on garde ou pas cette idée ☺

La matrice qui nous servira de "carte de base" pour placer les activités ou les modèles, sera celle-ci :

Table 2. Notre matrice des préoccupations

	Exigences	États	Structures	Interactions	Flux	Transversalités
Organisation						
Environnement						
Vision Opérationelle						
Vision Fonctionnelle						
Vision Organique						

Cette matrice permet de situer les différents éléments qui seront vus par la suite dans un cadre utile pour les comparer les uns aux autres. Nous vous conseillons de vous faire votre propre matrice. L'essentiel est de toujours bien se représenter les différents éléments qu'on aborde dans une carte mentale précise. Cela permet une meilleure mémorisation.

2.3.1. Points de vue

Dans un axe horizontal, nous avons différencié quatre grands points de vue :

Exigences

La définition des exigences et leur prise en compte en modélisation sont des éléments critiques pour le succès du développement de tout système. Sans couvrir l'ensemble des activités d'ingénierie système (ce qui nécessiterait tout un volume du type de [Sommerville1997]) nous insisterons beaucoup sur les exigences dans cet ouvrage.

États

Les systèmes complexes se décrivent souvent bien par les différents états dans lequel ils se trouvent et qui vont faire que le système réagit différemment à un même stimulus extérieur.

Structures

La description de l'architecture et des éléments constitutifs du système, avec les blocs, leurs

relations, organisations internes, etc. constituera un point de vue important. C'est souvent la partie de modélisation qui pose le moins de problème aux débutants.

Interactions

Le comportement d'un système est du point de vue de l'utilisateur final beaucoup plus important que la structure elle-même. L'utilisateur est la plus à même d'exprimer, de comprendre (via les modèles) et de valider. Nous distinguons dans ce livre le comportement global (du système) du comportement local (des éléments constitutifs).

Flux

Les flux (*Flows* en anglais) doivent être précisément identifiés en ingénierie système car ce sont eux que va consomer, contrôler, échanger, modifier, etc., le système en cours de développement. Il peut s'agir de matière, d'énergie, de données, etc. Ils seront plus ou moins concrets en fonction du niveau d'abstraction de la modélisation.

Transversalités

Un certain nombre de concepts sont transverses aux points de vue précédents. Il s'agira principalement de parler de cohérence ou de traçabilité entre les phases de développement ou entre les points de vue.

Ces différents points de vue ne doivent pas être confondus avec les différentes phases de développement (cf. paragraphe suivant). Ils sont plutôt à rapprocher de la notion de préoccupation.

2.3.2. Phase de développement

Dans un axe vertical, nous différencions cinq grandes phases du cycle de vie du développement :

Organisation

Une étape indépendante du type de cycle de développement envisagé (en V, agile, etc.) mais qui concerne la mise en place d'un cadre de travail qui permette un développement de qualité (outils, éditeurs, gestionnaire de version, de tâches, etc.).

Environnement

Aussi appelé le contexte du système. Souvent négligé (en informatique par exemple), il est primordial en Ingénierie Système. Les interactions entre le système et son environnement, les contraintes imposées par ce dernier sur le système, vont faire partie des premiers éléments à définir précisément.

De nombreux accidents se produisent parce que l'environnement du système a été mal modélisé car souvent considéré comme connu.

Vision Opérationelle

Cette phase vise plutôt à examiner le domaine du problème. Elle se focalise sur les cahiers des charges et les exigences. L'analyse débouche sur un dossier d'analyse qui décrit les grandes lignes (cas d'utilisation, architecture principale) du système.

Vision Fonctionnelle

Cette phase vise plutôt à examiner le domaine de la solution. Elle débouche sur un dossier de

conception qui décrit les détails conceptuels de la solution envisagée (structure détaillée, comportement, etc.).

Vision Organique

Cette phase traite des développements finaux (construction ou approvisionnement en matériel, développement de codes, etc.).

Il s'agit ici classiquement des grandes étapes de développement d'un système. On pourrait être surpris par l'étape que nous avons appelé "Organisation". C'est une étape que nous considérons importante, particulièrement pour pour le travail en équipe (ou pour l'enseignement). Avant toute activité de modélisation ou même de développement, il convient en effet de s'organiser en termes de choix d'outils, choix d'environnements, conventions, etc. Cette étape est souvent négligée par les étudiants. C'est pour cela que nous avons décidé de faire figurer cette étape de manière explicite.

Bien sûr dans une organisation existante cette étape sera contrainte par les habitudes "maison".

2.4. Organisation de ce livre

Suite à cette introduction, nous allons présenter l'étude de cas qui nous servira de "fil rouge" (Chapitre 3). Nous donnons ensuite les instructions pour installer l'environnement minimal pour réaliser les exercices de ce livre (Chapitre 4). Nous donnons ensuite un aperçu de tous les diagrammes SysML®, pour que le lecteur ait une bonne vue d'ensemble et puisse faire les connexions nécessaires entre tous les diagrammes (Chapitre 5). Après la définition de quelques éléments méthodologiques (chapitre Chapitre 6), nous attaquons la manipulation proprement dite de Papyrus-SysML (Chapitre 7). Nous rentrons ensuite dans le détail de SysML® en présentant les concepts avancés (Chapitre 9). Nous commençons par les exigences (Section 9.1), nous poursuivons par l'utilisation du système (cas d'utilisation et interfaces, Section 9.2), puis traitons des éléments structurels (Section 9.3) et comportementaux (Section 9.4), puis des interaction (Section 9.5). De part leur importance, nous consacrons un chapitre entier aux concepts avancés en lien avec les préoccupations transverses (Chapitre 10). Nous terminons par différents aspects complémentaires qui permettent d'aller encore plus loin (Chapitre 11). Un certain nombres d'annexes complètent le livre.

Rappelez-vous de vous référer à la grille de lecture que nous proposons en Table 1.

[2] Sauf bien sûr au brouillon ou sur un tableau, notamment quand on travaille en équipe.

Chapter 3. Étude de cas

Commentaire

Présenter le cas d'étude avec des illustrations SysML mais avec un look métier. Deux étapes : dans un premier temps, le cas d'étude est décrit globalement au travers de ses diagrammes sans s'attacher aux diagrammes eux-même. Dans un second temps, détail de chaque diagramme.

Nous allons modéliser dans cet ouvrage un système complexe, réel : la Maison Intelligente de Blagnac (MIB). Nous avons volontairement choisi un domaine qui prend de l'ampleur : l'Internet des Objets (IoT - *Internet of Thing*).

Figure 3. La "Maison Intelligente" de Blagnac

La Maison Intelligente de Blagnac est un appartement permettant l'assistance de personnes dépendantes à leur domicile. La spécification initiale de ce système est disponible sur le site http://mi.iut-blagnac.fr/. La modélisation dans Papyrus-SysML que nous allons faire ensemble tout au long de ce livre s'appuiera sur une spécification initiale issue de ce projet et que vous pourrez voir à la Section 3.3.

1. Nous ne nous intéresserons qu'à la partie intelligente de la maison (et non à ses caractéristiques architecturales par exemple).
2. Nous avons pris quelques libertés sur les caractéristiques de la maison réelle pour parfois les enrichir (comme l'exigence `SH_ACO_030` ci-dessous) ou parfois les simplifier.
3. Cette spécification initiale des besoins sert de base pour étudier la partie *elicitation* des exigences. Elles ne sont donc ni parfaites, ni complètes.
4. Enfin, pour des soucis de partage plus large avec la communauté internationale, les exemples de modèles réalisés dans le livre le seront en anglais et à partir de la version anglaise des exigences (disponible ici).

3.1. Description

La variété des équipements disponibles dans la Maison Intelligente de Blagnac permet à ses occupants de réaliser un ensemble de tâches comme :

- Surveiller (*Monitoring*) sa consommation énergétique (globale ou d'un équipement en particulier)
- Activer des dispositifs (*Actuators*) comme l'intensité d'une ampoule (*light bulb*), ou l'ouverture d'un rideau (*shutter*)
- Surveiller les données collectées par des capteurs (*Sensors*) comme la température,ou la luminosité.

L'objectif principal de la Maison Intelligente de Blagnac est d'être un support à l'assistance de personnes autonomes à leur domicile qui sont autonomes mais qui doivent être suivies et aidées en cas de problème. Les éléments de conforts mais surtout de sécurité sont donc primordiaux et doivent être fournis de manière continue, et peuvent potentiellement nécessiter l'intervention de moyens humains d'assistance.

L'intérêt de ce cas d'étude (que nous appèlerons *SmartHomeSystem*, ou SHS dans ce livre) est qu'elle nécessite de prendre en compte des préoccupations diverses, parfois contradictoire, comme sécurité (*security*), adaptabilité (*adaptability*), sociétaux (*societal*), légaux (*legal*), économiques (*economical*), et écologiques (*ecological*). Dans ce qui suit nous décrivons plus précisément quelques services attendus du système à concevoir pour chacune de ces préoccupations.

3.1.1. Sécurité

Comme n'importe quel autre logement, la Maison Intelligente de Blagnac doit être sécurisée. Notre système à concevoir doit donc fournir un certain niveau de protection contre les intrus et les voleurs. Ce niveau doit même être renforcé du fait de la vulnérabilité induite par l'ultra-connectivité du système. On profitera donc avantageusement du niveau d'équipement pour mettre en place les réponses adaptées (alarmes, alertes automatiques, etc.) sans pour autant oublier de donner un accès privilégié aux différents acteurs de la supervision (pompiers, médecins, etc.). L'accès aux données personnelles, notamment de santé, via les différents dispositifs connectés sera fortement contrôlé. Les notions d'identification, de certification ou de cryptage seront donc primordiales.

Cette description mélange des aspects différents de sécurité : la sureté, la sécurité et la cybersécurité. Nous avons conservé volontairement cette ambiguïté pour illustrer comment SysML® peut être utilisé pour exprimer la composition d'exigences.

3.1.2. Adaptation

Le système à concevoir doit également pouvoir s'adapter aux conditions physiques et mentales de l'occupant[3]. Par exemple, la hauteur des équipements de cuisine (meubles, éviers) s'abaisse automatiquement si l'occupant est en fauteuil roulant. Un autre exemple, qui concerne les personnes atteintes de troubles cognitifs, est la possibilité de guider l'occupant par la voix ou par

un signal lumineux au sol, d'un point à un autre de la maison, par exemple le matin pour guider l'habitant(e) de son lit à la salle de bain. Le système doit également permettre de configurer les nombreux équipements en fonction des préférences de l'occupant (par exemple, heure de fermeture des volets ou du réveil, etc.).

3.1.3. Environnement et économie

Une des caractéristiques des maisons intelligentes modernes est leur prise en compte de l'environnement et donc des consommations énergétiques. L'objectif peut être économique, légal (conformité à une norme ou un label), ou simplement social. Par exemple le système pourra abaisser la température de la maison à certains moments (programmés ou opportuns). Les dispositifs inutiles à certains moments de la journée pourront être mis en sommeil sommeil à certains moments de la journée, ou bien encore les débits Internet réduits. La maison pourra aussi être autonome en énergie, ou tout le moins, produire dans une certaine mesure sa propre énergie (éolienne, panneaux solaires, etc.).

3.2. Plateforme expérimentale

Un tel système de maison intelligente est disponible en réel sur le site expérimental de la Maison Intelligente de Blagnac, sur le campus de l'IUT de Blagnac, près de Toulouse. N'hésitez pas à nous contacter si vous êtes intéressé par des expérimentations réelles.

3.3. Liste des exigences

Nous fournissons ci-dessous une liste précise des exigences (*requirements*) que nous allons considérer pour notre système à développer. Cette liste est bien sûr non-exhaustive, mais elle nous permettra d'illustrer comment faire de l'ingénierie système avec Papyrus-SysML.

Table 3. Une liste initiale d'exigences pour le système de la maison intelligente (SmartHomesystem)

ID	Categorie	Description
SH_SEC_010	Sécurité	Le *SHS* doit empêcher les accès non autorisés.
SH_SEC_020		Le *SHS* doit protéger les systèmes d'informations utilisés.
SH_SEC_030		Le *SHS* doit fournir un contrôle d'accès par identification.
SH_SEC_040		Le *SHS* doit appeler la police en cas d'intrusion ou d'effraction.
SH_SEC_050		Le *SHS* doit être capable de détecter la fumée et d'appeler les pompiers en cas de suspicion d'incendie.
SH_SEC_060		Le *SHS* doit être capable de détecter la chute accidentelle et d'appeler le SAMU en conséquence.
SH_SEC_070		Le *SHS* doit permettre un accès spécifique aux secours (pompiers, ambulanciers, etc.).
SH_ACO_010	Adaptativité	Le *SHS* doit s'adapter aux conditions physiques et/ou mentales de l'occupant.

ID	Categorie	Description
SH_ACO_020		Le *SHS* doit s'adapter aux préférences spécifiques renseignées par l'occupant.
SH_ACO_030		Le *SHS* doit pouvoir apprendre du comportement de l'occupant.
SH_ACO_040		Le *SHS* doit aider l'occupant dans ses tâches quotidiennes en tenant compte le plus possible de ses difficultés.
SH_ACO_050		Le *SHS* doit pouvoir être supervisé par un dispositif central, dans la pièce principale de l'habitation.
SH_ACO_060		Le *SHS* doit permettre la connexion avec les équipements modernes de l'occupant (smartphone, montre, pèse-personne, etc.).
SH_ACO_070		Le *SHS* doit posséder un lit adapté aux personnes à mobilité réduite.
SH_ECO_010	Économique	Le *SHS* doit économiser l'énergie.
SH_ECO_020		Le domicile doit être économe en énergie et avoir une consommation globale inférieure aux normes constatées localement.
SH_ECO_030		Le *SHS* doit donner la priorité aux énergies renouvelables.

3.4. Quelques exemples de scénarios

Dans ce qui suit, nous imaginons que l'habitante qui occupe la maison s'appelle Alice et que son état de santé nécessite un suivi particulier.

- Chaque matin, quand Alice s'éveille, un signal lumineux au sol la guide pour mermettre d'atteindre la salle de bain pour sa toilette. Si elle a choisi l'option, ou qu'elle le demande, elle peut même bénéficier d'un guidage audio complémentaire.
- Alice bénéficie d'un "majordome intelligent"[4], capable :
  - d'afficher un indicateur de santé de la maison, qui combine les informations provenant de la mesure de plusieurs grandeurs physiques (e.g., Température, Humidité, CO2, ...)
  - d'afficher des indicateurs détaillés sur différents domaines (en instantané et avec des graphiques) en le ramenant à des indicateurs intelligibles pour l'humain (Qualité de l'air, Consommation électrique, ...)
  - d'afficher des prévisions pour la journée / semaine (Consommation électrique prévue, Production d'énergie prévue, "La voiture sera chargée pour faire votre trajet quotidien à 16h30", ...)
  - d'afficher des conseils / alertes pour bénéficier au mieux des capacités passives de la maison (Ouvrir la fenêtre pendant 10 minutes, Ne pas faire entrer de nouveaux visiteurs, ...)
- Si Alice chute sur le sol et qu'elle reste immobile plus de 30s, et qu'elle ne répond pas à une première série de messages audio diffusés dans la maison, le système à concevoir appelle les secours avec les informations rentrées au préalable dans le système.

3.5. Pour aller plus loin

Cette étude de cas a été utilisée dans le cadre d'un cours SysML® au Master DL ainsi que pour le workshop MDETOOLS'17. Ces deux sources peuvent fournir des modèles complémentaires à ceux que vous trouverez dans ce livre.

Pour une liste plus complète d'exigences, réalisée par les étudiants du Master DL, voir ce lien.

[3] Pour des raisons de simplicité, et pour coller à la réalité de la Maison Intelligente de Blagnac, nous considérons qu'il n'y a qu'un(e) seul(e) habitant(e).

[4] Scénario tiré du Projet ESIR3 NSOC donné par Benoît Combemale à l'ESIR.

Chapter 4. Avant de démarrer

Commentaire

Ce chapitre, technique, présente Papyrus, eclipse, etc. Il aborde également la question de l'organisation des modèles (packages, etc.).

Table 4. On s'organise...

	Exigences	États	Structures	Interactions	Flux	Transversalités
Organisation	**You are here!**					
Environnement						
Vision Opérationelle						
Vision Fonctionnelle						
Vision Organique						

4.1. Installation de Papyrus-SysML

4.1.1. Papyrus

Vérifiez que vous possédez bien sur votre machine : Papyrus-SysML `2019-09` (ou une version supérieure).

Figure 4. Papyrus au lancement [v.2019-09]

Si ce n'est pas le cas, installez-le ou mettez-le à jour. Nous vous conseillons d'installer Papyrus-SysML de cette façon plutôt que de le rajouter comme un plugin de votre installation eclipse.

Pour plus d'information, consultez guide d'installation du site de Papyrus.

Si vous voulez mettre à jour une ancienne version de Papyrus-SysML, il vous faut non seulement ajouter l'update site de Papyrus-SysML, mais aussi celui d'eclipse (par exemple `download.eclipse.org/releases/2019-06`).

4.1.2. SysML

Il vous faut ensuite installer le profil SysML® (version 1.6). Pour cela, deux possibilités :

1. Aller dans le menu **Help › Install New Software › ...** puis utilisez l'URL suivante pour retrouver le plugin officiel : https://download.eclipse.org/modeling/mdt/papyrus/components/sysml16/.
2. Récupérer directement depuis l'eclipse market place

1. Pour vérifier les éléments installés allez dans **About... › Installation Details** :

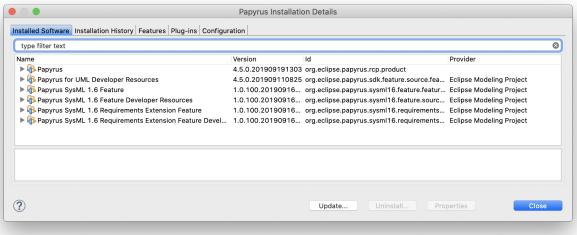


Figure 5. Détails d'installation [v.2019-09]

4.1.3. Compléments

Si vous souhaitez travailler avec Git pour versionner vos modèles ou travailler de manière collaborative, ou si vous souhaitez utiliser des éléments avancés de Papyrus-SysML, rendez-vous au Chapitre 11.

4.2. Matériel de formation en complément de ce livre

Commentaire

Information sur le matériel disponible autour du livre (site web, exemples, forums, etc.).

Le site qui répertorie tous les matériels utiles en complément de ce livre (tutoriels, modèles, compléments, corrections, etc.) se trouve ici : https://github.com/jmbruel/sysmlpapyrusbook.

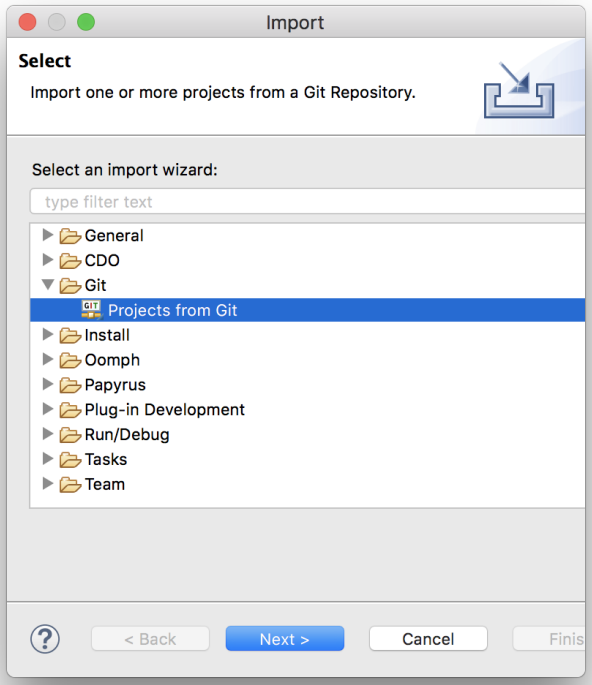
Si vous souhaitez vérifier que votre installation fonctionne en créant un premier projet Papyrus-SysML, vous pouvez consulter le Chapitre 7.

4.3. Pour ceux qui veulent aller plus vite

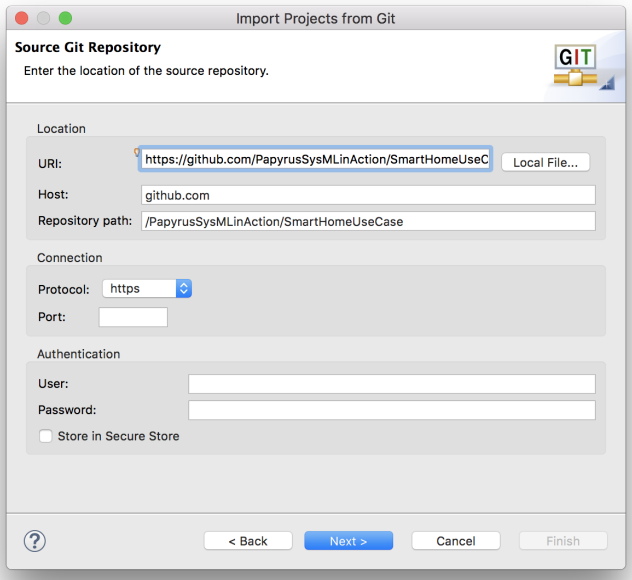
Les modèles de ce livre (cf. Chapitre 3) sont disponibles ici : https://github.com/PapyrusSysMLinAction/SmartHomeUseCase.

Pour créer un projet qui contienne ces modèles, suivez simplement les étapes suivantes :

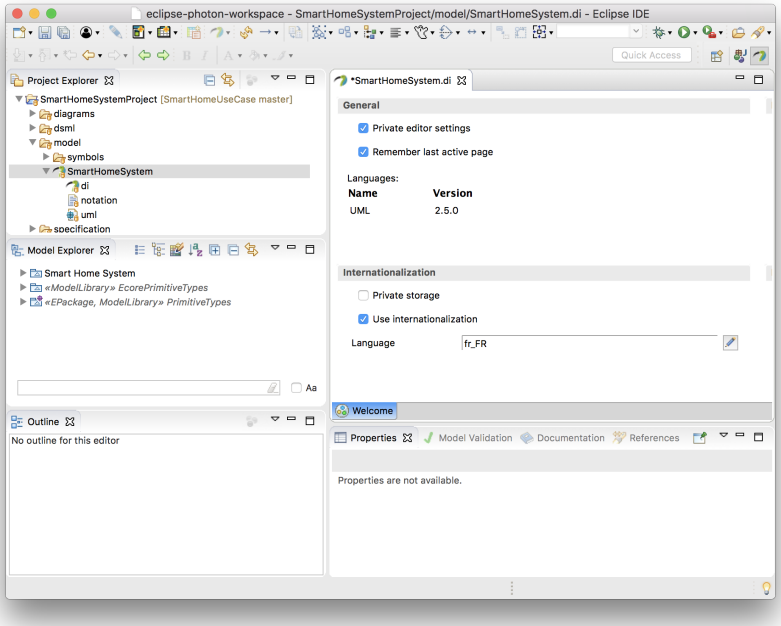
1. Importez le projet git existant
    - **File › Import... › Git › Projects from Git › Next**



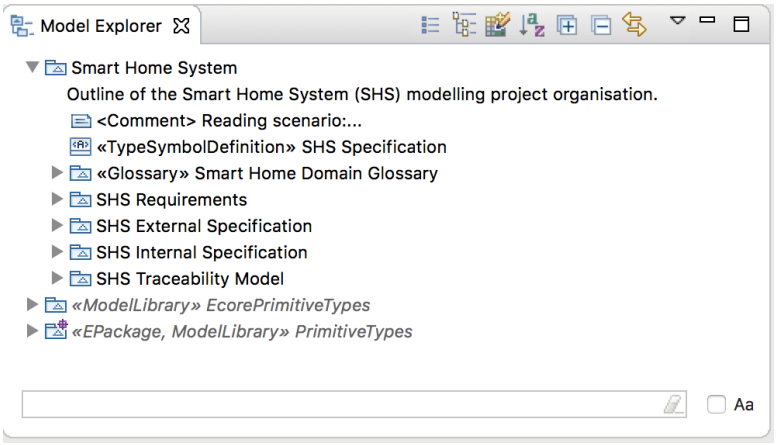
    - choisir **Clone URI › Next** et entrez l'URL du projet (https://github.com/PapyrusSysMLinAction/SmartHomeUseCase)



- choisir la branche `master` du dépôt et l'emplacement de votre copie locale
- Voilà!



2. Explorez le modèle via le `Model Explorer`



4.4. Pour ceux qui veulent aller plus loin

Nous vous conseillons d'installer également les plug-ins suivants qui peuvent s'avérer utiles :

- Eclipse Layout Kernel pour faire des diagrammes plus jolis sous eclipse

Chapter 5. Introduction à SysML

Commentaire

Introduire ici la méthodologie simplifiée adoptée pour le livre. Au travers de cette présentation, on introduira également les différents diagrammes de SysML.

Ce chapitre est une introduction au langage de modélisation pour l'ingénierie système, SysML®. L'objectif de ce chapitre est d'une part de vous donner les connaissances minimales nécessaires pour commencer la lecture de ce livre, et d'autre part de vous donner les clés principales de lecture du document de référence, la spécification elle-même [SysML]. Entre autre, nous allons vous présenter ici l'ensemble des diagrammes tels que proposés par le langage et nous discuterons également des questions auxquels ils peuvent permettre de répondre par rapport à des problématique d'Ingénierie Système.

5.1. Pourquoi SysML versus UML ?

> A good notation has subtlety and suggestiveness which at times makes it almost seem like a live teacher.
>
> — Bertrand Russell, The World of Mathematics (1956)

UML® (*Unified Modeling Language*, [UML]) est le premier langage de modélisation normalisé par l'OMG™ (*Object Management Group*). La première version du langage fut normalisée en 1997 par l'OMG™ et la spécification devient un standard ISO en 2005 [UML]. Initialement conçu pour le domaine du génie logiciel, le langage de modélisation unifiée a été néanmoins appliqué dès ses débuts au domaine de l'Ingénierie Système. Son vocabulaire spécifique à l'ingénierie logiciel – par exemple, les notions de classe et d'opération – et ses manques conceptuels ne permettant pas de couvrir l'ensemble des besoins pour une ingénierie système par les modèles performante – par exemple, pas de support dédié à la gestion des exigences – n'ont pas permis à UML® de devenir le langage unifié de l'Ingénierie Système. Fort de ce constat, des acteurs de l'OMG™ en partenariat avec l'INCOSE (*International Council on Systems Engineering*) décident de spécifier un nouveau langage de modélisation particulièrement adapté aux besoins et au vocabulaire de l'Ingénierie Système. C'est ainsi que né SysML® 1.0 en septembre 2007.

Dans ce contexte, les concepteurs de SysML® font toutefois le constat qu'UML® forme une bonne base de travail. En effet, il est déjà à l'époque un standard *de facto* en génie logiciel (une des branches du génie système), il fournit quand même beaucoup de concepts utiles pour décrire des systèmes, même complexes, il est stable et extensible (grâce notamment au mécanisme de profil) et il existe un grand nombre de ressources incluant des tutoriels et des outils, certains open-source comme Papyrus.

Fort de ce constat, la décision est prise d'implanter SysML® comme une extension d'UML® sous la forme d'un profil UML® pour l'Ingénierie Système.

5.2. SysML en quelques mots

5.2.1. Fiche d'identité

Voici à quoi pourrait ressembler la fiche d'identité de SysML® :

Commentaire

Choisir entre les deux mais mettre à jour le dessin (et avoir une version SVG) si option Seb adoptée.

Carte d'identité

- Date de naissance non officielle : 2001!
- Première spécification adoptée à l'OMG™ : 19 septembre 2007
- Version actuelle : `1.6 beta` (03/2019)
- ISO Standard ISO/IEC 19514:2017
- Paternité : OMG™ / UML® + INCOSE
- Auteurs principaux :
  - Conrad Bock
  - Cris Kobryn
  - Sanford Friedenthal
- Logo officiel :

Figure 6. Carte d'identité de SysML

5.2.2. Qui est "derrière"?

Comme toutes les spécifications issues de l'OMG™, SysML® est sujet à des évolutions incrémentales qui permettent d'améliorer constamment le langage au travers de ce que l'on appelle des RTF (*Revision Task Force*). Les évolutions de SysML® sont pilotées par deux consortiums : l'OMG™ d'une part pour les liens avec UML® et l'INCOSE d'autre part pour les liens avec l'ingénierie système.

Il est intéressant de rappeler que SysML® est un standard qui vient des utilisateurs, qui participent à sa définition et à ses évolutions. Voici une liste non exhaustive des entreprises et des acteurs qui participent à son évolution :

Industrie

American Systems, BAE Systems, Boeing, CEA, Deere & Company, EADS Astrium, Eurostep, Israel Aircraft Industries, Lockheed Martin, Motorola, NIST, Northrop Grumman, oose.de, Raytheon, Thales, ...

Vendeurs d'outils

Artisan, EmbeddedPlus, Gentleware, IBM, Mentor Graphics, PivotPoint Technology, Sparx Systems, Vitech, ...

Autres organisations

AP-233, Georgia Institute of Technology, AFIS, ...

La liste complète des membres de l'OMG™ est accessible à l'URL : http://www.omg.org/cgi-bin/apps/membersearch.pl

Sachez que vous aussi vous pouvez participer à l'effort de normalisation et améliorer ainsi la spécification. En effet, il vous suffit pour cela de lever une *issue* via ce service fournit par l'OMG™ : https://issues.omg.org/issues/create-new-issue. Et c'est l'objet des groupes de travail précédemment mentionnés (RTF) que de répondre à ces rapports de bugs en modifiant le cas échéant la spécification si elle est jugée pertinentes par l'équipe. Donc, si quelque chose ne va pas du côté de

SysML®, n'hésitez pas à remonter l'information, c'est la meilleure des façons de faire avant les choses. Enfin, si vous souhaitez faire avancer encore plus vite des changements, vous pouvez également participer à cette aventure de normalisation en devenant membre de l'OMG™.

5.2.3. SysML, c'est…

SysML® est un ensemble de 9 types de diagrammes classés en trois catégories (cf. Figure 8) :

- Les diagrammes structuraux représentés par le diagramme de définition de blocs (`bdd` en bref), le diagramme internes de blocs (`ibd` en bref), le diagramme paramétrique (`par` en bref) ou le diagramme de paquetages (`pkg` en bref).
- Les diagrammes comportementaux représentés par le diagramme de séquences (`sd` en bref), le diagramme d'activité (`act` en bref), le diagramme de cas d'utilisation (`uc` en bref) et le diagramme de machine à états[5] (`stm` en bref).
- Le diagramme des exigences (`req` en bref).

SysML® est un profil UML®

Comme expliqué précédemment, SysML® a été conçu comme une spécialisation d'UML® par le biais d'un profil UML®. En bref, un profil UML® permet d'une part de réduire la complexité d'UML® en enlevant les concepts qui ne sont pas nécessaire à usage donné. D'autre part, on peut aussi au travers d'un profil étendre des concepts UML® pour soit adapter le vocabulaire UML® à un domaine spécifique – par exemple, le concept de *Block* introduit pour remplacer le concept de *Class* ou encore adapter la sémantique d'un concept – par exemple l'introduction du concept de *Requirement* comme une extension du concept basique de *Class*. Un des intérêts majeurs de ce choix d'implantation est qu'ainsi tous les outils UML® peuvent devenir très simplement des outils SysML® et que l'enseignement de SysML® peut s'appuyer sur une connaissance d'UML® puisque SysML® est au trois quarts similaire à UML®[6]. Enfin, il faut aussi noter qu'il est de ce fait possible d'utiliser SysML® conjointement avec d'autres profils comme par exemple MARTE si l'on souhaite traiter des préoccupations de type temps-réel ou embarquée, ou encore SoaML si l'on souhaite aller vers des architectures de type SOA (*Sercice Oriented Architecure*).

SysML® est une notation, donc une notation et une sémantique

SysML® n'est pas une simple notation graphique. Les symboles qui forment sa notation ont une définition sémantique, ce qui forme un tout appelé langage. En conséquence de quoi, toutes les représentations SysML® ont un sens qui peut être compris de toute personne connaissant le langage.

SysML® est partiellement un langage formel de modélisation

D'aucun ont longtemps reproché à UML® et donc à SysML® de ne pas être suffisamment précisément décrit, ou pour être plus précis, de ne pas être formellement défini, laissant ainsi libre court à des interprétations potentiellement différentes selon les lecteurs. Ce n'est maintenant plus le cas grâce à une initiative de l'OMG™ lancée vers 2010 de formaliser UML® et SysML®. L'approche choisie pour atteindre cet objectif ambitieux a été d'adopter un processus itératif en commençant par la formalisation d'un noyau, appelé fUML (*foundational UML*), et de là, être capable de formaliser de façon incrémentale les autres parties des deux normes. On peut citer par exemple PSCS et PSSM (respectivement *Precise Semantics of Composite structure* et *Precise Semantics of StateMachine*).

5.2.4. SysML, ce n'est pas...

Nous l'avons déjà évoqué, mais il convient d'insister, SysML® n'est pas :

Une méthode

En effet, contrairement à ce que beaucoup pensent en l'abordant, SysML® ne vient pas avec une démarche méthodologique particulière. Il s'agit bel et bien d'un langage et pas d'une méthode.

Un outil

Nous verrons en effet que SysML® ne fait que ce qu'on veut bien en faire. Comme tout langage il est limité dans son pouvoir d'expression, mais surtout il est sujet, et parfois limité, par les capacités des outils qui permettent de manipuler ses concepts par le biais de ses éditeurs de diagrammes SysML®.

Ne dites pas "le SysML" mais tout simplement "SysML".

5.2.5. Différences avec UML

La Figure 7, extraite de la spécification officielle, résume bien les liens entre SysML® et UML®, à savoir que SysML® d'une part du point de vue sémantique reprend une partie seulement des concepts d'UML® (appelée `UML4SysML`), en exclue en certain nombre et en ajoute d'autre. D'autre part du point de vue syntaxique, SysML® reprend une partie des diagrammes, en propose des spécialisations et en ajoute un nouveau.

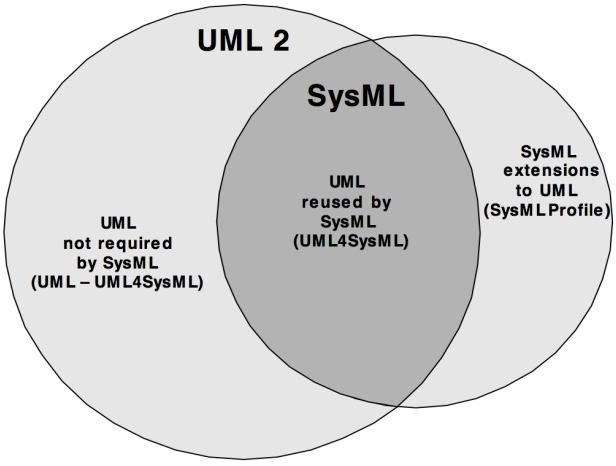


Figure 7. Liens entre UML et SysML (extrait de SysML 1.5, p. 9)

SysML® et UML® sont parfois utilisés de manière complémentaire, mais la plupart du temps, UML® est utilisé pour modéliser les aspects logiciels et SysML® pour modéliser les systèmes.

On retrouvera cette "paternité" entre UML® et SysML® dans Papyrus-SysML. Par exemple les propriétés d'un élément de modélisation se retrouveront dans des onglets séparés selon que ces propriétés sont "natives" d'UML® ou bien ajoutées par SysML®.

5.2.6. Outils SysML

Il va de soit que le meilleur outil SysML® à ce jour est Papyrus-SysML[7]. Ce livre vous en détaille l'utilisation, mais il existe un certain nombre d'autres outils permettant de réaliser des modèles SysML®. En voici une liste non exhaustive (ils sont pour la plupart payants) :

- Enterprise Architext (Sparx System)
- Cameo Systems Modeler (NoMagic[8])
- MagicDraw (NoMagic)
- Rhapsody (IBM)
- Visual Paradigm
- Modelio (Softeam)

5.2.7. Organisation des différents diagrammes

Les ingénieurs systèmes ont l'habitude d'utiliser des représentations graphiques comme des plans et des graphiques. SysML® propose de couvrir la modélisation d'un système au travers de 9 diagrammes permettant d'aborder les aspects structurels et comportementaux du système ainsi que les exigences. La Figure 8 présente cette organisation en faisant, en même temps, le lien avec ceux d'UML® :

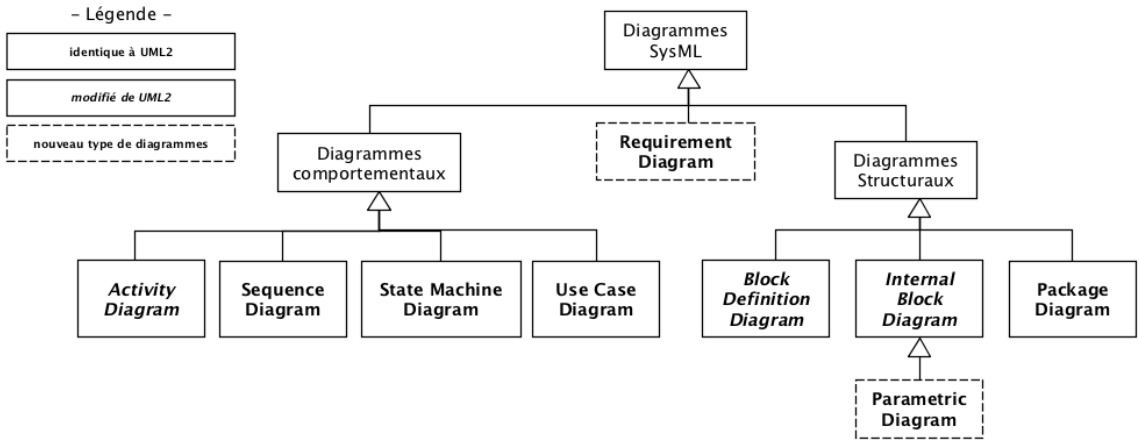


Figure 8. Les 9 diagrammes SysML et leur lien (historique) avec UML (extrait de la spécification)

Le nom de ces diagrammes revenant souvent dans ce document, nous utiliserons souvent leur version abrégée (uc pour "diagramme des cas d'utilisation" par exemple). Ces abréviations, sont définies dans la spécification.

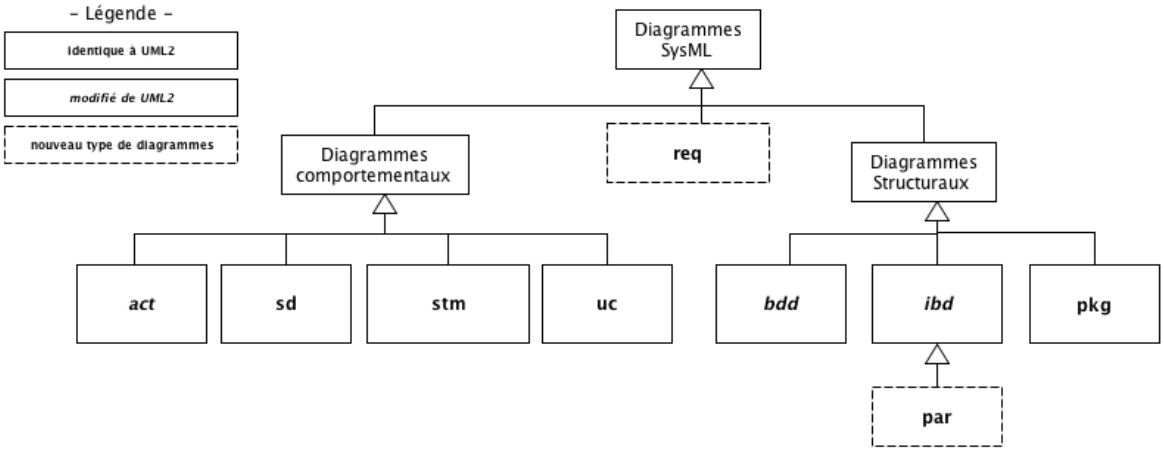


Figure 9. Version abrégée des diagrammes

Définition : Types de diagrammes (OMG SysML v1.5, p. 196)

SysML diagram kinds should have the following names (or abbreviations) as part of the heading…

5.2.8. Cadre pour les diagrammes

Abordons quelques principes généraux de SysML®, c'est-à-dire des éléments indépendants d'un diagramme en particulier :

- chaque diagramme SysML® décrit un élément précis (nommé) de modélisation,
- chaque diagramme SysML® doit être représenté à l'intérieur d'un cadre (*Diagram Frame*),

Malheureusement, à l'heure où nous écrivons cet ouvrage, cette fonctionalité n'est pas encore supportée par Papyrus-SysML.

- l'entête de ce cadre, appelé aussi **cartouche**, indique les informations sur le diagramme :
  - le **type** de diagramme (`req` , `act`, `bdd`, `ibd`, `stm`, etc. en gras) qui donne immédiatement une indication sur le point de vue porté à l'élément de modélisation (comportement, structure, etc.),
  - le type de l'élément (par exemple *package*, *block*, *activity*, etc.), optionnel,
  - le nom de l'élément modélisé (unique),
  - le nom du diagramme ou de la vue, optionnel.

Dans l'exemple illustré par la Figure 10, le diagramme nommé "*Context_Overview*" est un diagramme de type *Block Definition Diagram* (type `bdd`) et il représente un *package*, nommé "Context".

Commentaire

Changer la figure pour éviter le package et le blocl de même nom.

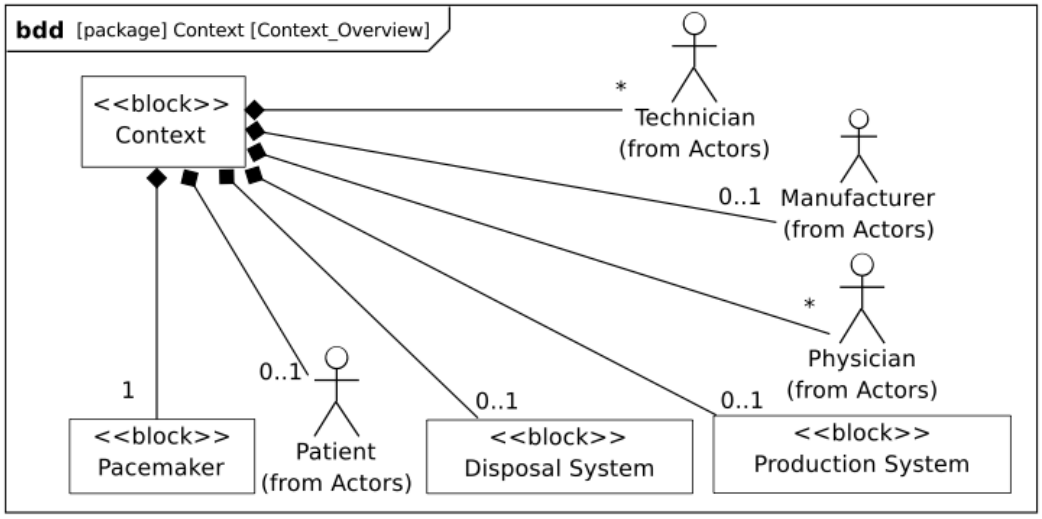


Figure 10. Exemple de diagramme SysML (source [Kordon])

Un cadre peut contenir dans ses "bords" des éléments importants en fonction du type de modèle qui est représenté dans le cadre (ports d'un bloc, points d'entrée/sortie d'une machine à état, paramètres d'une activité, etc.).

5.2.9. Notre présentation des diagrammes

Comme nous l'avons vu précédemment les deux grandes catégories de diagrammes SysML® sont les diagrammes structuraux et les diagrammes comportementaux. Mis à part les deux diagrammes des exigences (`req` , représentation visuelle des exigences) et de paquetages (`pkg`, représentation visuelle des *packages*), les autres diagrammes nécessitent de préciser l'utilisation que nous préconisons d'en faire.

Pour les diagrammes structuraux (souvent aussi qualifiés de statiques), il s'agira de les utiliser pour représenter des préoccupations de nature architecturale statique. Plus précisément, quand il s'agit d'illuster des principes architecturaux d'un système, nous utiliserons volontiers les diagrammes de définition de blocs (`bdd`), et pour représenter les détails de l'intérieur d'un sous-système, nous utiliserons alors plus naturellement les diagramme de blocs internes (`ibd`). Enfin, nous utiliserons le diagramme paramétrique (`par`) – qui au passage n'est qu'un diagramme de blocs internes dédié aux relations entre valeurs – pour représenter des contraintes entre les propriétés des différents éléments qui constitue un système.

Pour les diagrammes comportementaux (souvent aussi appelés dynamiques), nous faisons la distinction entre un comportement global et un comportement local, comme illustré dans la Figure 11. Ainsi les diagrammes des cas d'utilisation (`uc`) et de séquences (`sd`) seront plutôt utilisés pour représenter le comportement global d'un système tel qu'attendu par les parties prenantes du système. Tandis que les diagrammes d'activité (`act`) et les diagrammes de machines à états (`stm`) seront plutôt utilisés en phase de conception pour détailler le comportement d'un block système. On utilisera volontiers les diagrammes de machine à états pour décrire un comportement logique où l'historique de l'objet est important par rapport à son comportement et on utilisera les diagrammes d'activité pour décrire un comportement de type algorithmique.

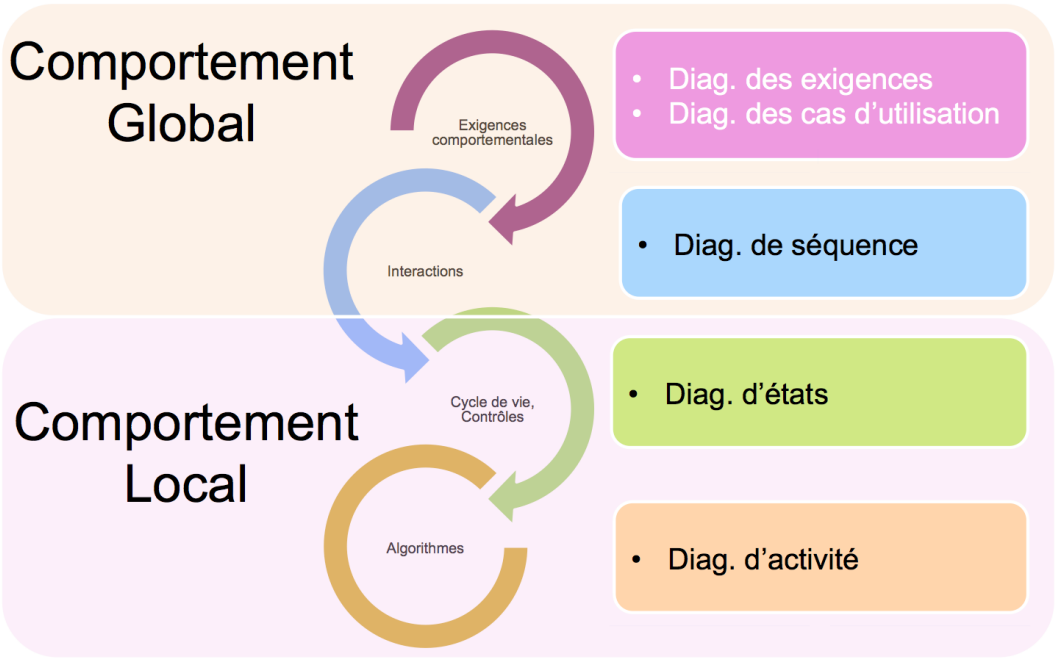


Figure 11. Différents diagrammes pour repésenter le comportement

Pour ceux qui cherchent à étudier un diagramme en particulier voici un plan des sections qui suivent (nous utilisons ici le "plan" vu lors de l'introduction de la Section 2.3) :

Table 5. Organisation

	Exigences	États	Structures	Interactions	Flux	Transversalités
Organisation	pkg	pkg	pkg, bdd, ibd	pkg	pkg	
Environnement			pkg, bdd, ibd			
Vision Opérationelle						
Vision Fonctionnelle						
Vision Organique	req, uc, sd	stm	bdd, ibd, par	uc, sd, stm, act	ibd	par, pkg

5.3. Diagramme et table des exigences (req)

La gestion des exigences est une activité cruciale pour la réussite de la conception et le développement de tout système. Dans SysML® les exigences peuvent être représentées sous deux formes complémentaires :

- le Diagramme des exigences (req), utile pour représenter les relations entre exigences;
- la Table des exigences, utile dès qu'il y a un grand nombre d'exigences.

5.3.1. Diagramme des exigences

Ce diagramme permet de représenter graphiquement les exigences et leurs relations. Il sera vu plus en détail dans la section Section 9.1, mais en voici un exemple appliqué à la Maison Intelligente de Blagnac : une exigence (`Security`) est complsée de plusieurs sous-exigences.

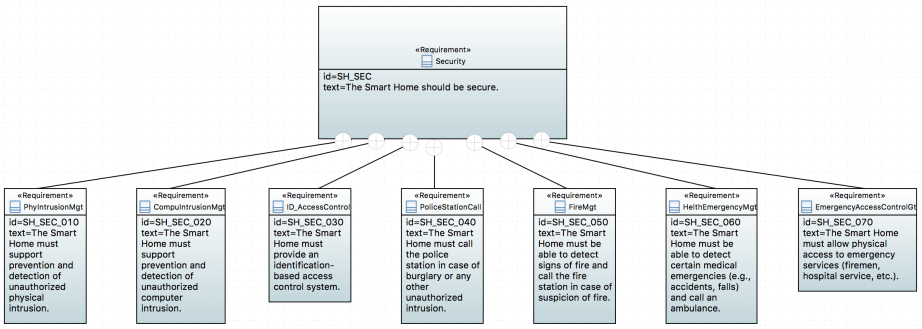


Figure 12. Exemple de diagramme des exigences

5.3.2. Table des exigences

Il est souvent plus pratique de lister les exigences sous forme de table.

SmartHomeSystem.di

	id	text
Safety&Security	SHS_SEC	The system must be safe and secure.
PhyIntrusionMgt	SHS_SEC_010	The system must support prevention and detection of unauthorized physical intrusion.
PhyIntrusionPreventionMgt	SHS_SEC_010_010	The system must support prevention unauthorized physical intrusion.
PhyIntrusionDetectionMgt	SHS_SEC_010_020	The system must support prevention unauthorized physical intrusion.
CompuIntrusionMgt	SHS_SEC_020	The system must support prevention and detection of unauthorized computer intrusion.
CompIntrusionPreventionMgt	SHS_SEC_020_010	The system must support prevention unauthorized computer intrusion.
CompIntrusionDetectionMgt	SHS_SEC_020_020	The system must support prevention unauthorized computer intrusion.
ID_AccessControl	SHS_SEC_030	The system must provide an identification-based access control system.
PoliceStationCall	SHS_SEC_040	The system must call the police station in case of burglary or any other unauthorized intrusion.
FireMgt	SHS_SEC_050	The system must be able to detect signs of fire and call the fire station in case of suspicion of fire.
HelthEmergencyMgt	SHS_SEC_060	The system must be able to detect certain medical emergencies (e.g., accidents, falls) and call an ambulance.
EmergencyAccessControlGt	SHS_SEC_070	The system must allow physical access to emergency services (firemen, hospital service, etc.).

Figure 13. Exemple de table des exigences pour le SmartHomeSystem

Tous ces éléments seront détaillés à la Section 9.1.

5.4. Diagramme des cas d'utilisation (uc)

Le Diagramme des Cas d'Utilisation est un modèle UML® permettant de représenter :

- les cas d'Utilisation (*Use Case*)
- les acteurs (*Actors_*)
- les interactions entre acteurs et cas d'Utilisation
- les relations entre cas d'Utilisation

Le travail sur les exigences (cf. Section 5.3 et Section 9.1) permet, en plus de lister précisément les exigences, d'identifier les éléments (humains ou non) qui intéragissent avec le système. En SysML®, on parlera de manière globale d'*Acteurs* (les "bonhommes" dans la Figure 14). De plus les grandes fonctionalités du système, du point de vue de ces acteurs, seront matérialisés sous la forme de *Cas d'utilisation* (les "patates" dans la Figure 14). Enfin le diagramme précise quel(s) acteur(s) intéragit avec quel cas d'utilisation (les traits enre acteurs et cas) et les relations éventuelles en les cas d'utilisation (extension, inclusion, en pointillé dans la Figure 14).

[System security use cases diagram] |

../../SmartHomeUseCase/SmartHomeSystemProject/diagrams/svg/System_security_use_cases_diagram.

SVG

Figure 14. Exemple de diagramme de cas d'utilisation

Tous ces éléments seront détaillés à la Section 9.2.3.

5.5. Diagramme de blocs (bdd)

Le diagramme de blocs (ou diagramme de définition de blocs) permet de principalement de représenter la structure en terme de composés/composants. Il permet de définir des relations (très souvent de composition) entre des éléments de base, appelés "blocs" et qui généralement représentent des éléments concrets. Ainsi ce diagramme permettra de préciser des éléments comme des propriétés, des associations, des multiplicités, etc.

La Figure 15 illustre un diagramme de bloc précisant ...

Figure 15. Exemple de diagramme de blocs

Pour les lecteurs familiers d'UML®, le concept SysML® de *block* est à rapprocher du concept UML® de *class* (cf. Section 9.3.3). D'ailleurs vous constaterez que les blocs sont des rectables (comme les classes en UML®) étiquetés avec le stéréotype <<Block>>.

Diagramme de contexte

Un diagramme de blocs un peu particulier consiste à représenter le système dans son environnement. Ce "diagramme de contexte" est pratique d'un point de vue modélisation car le système et les autres éléments en interaction avec lui, sont très tôt identifiés et "référençables" (des `connectableElement` en termes SysML®).

[Description of the SystemContext BDD view] |

*../../SmartHomeUseCase/SmartHomeSystemProject/diagrams/svg/Description_of_the_*

SystemContext_BDD_view.SVG

Figure 16. Exemple de diagramme de contexte

Tous ces éléments seront détaillés à la Section 9.3.3.

5.6. Diagramme de blocs internes (ibd)

Complémentaire au diagramme de blocs vu à l'instant, le diagramme de blocs internes permet de représenter comment les composants d'un bloc sont précisément reliés à ce bloc, voire reliés entre eux. Ainsi ce diagramme permettra de préciser des éléments comme des propriétés, des parties, des ports, des flux, etc.

Nous recommandons fortement de ne faire l'ibd d'un bloc qu'après avoir fait le bdd de ce bloc. En effet les *parts* et autres associations avec les éléments internes et externes de ce bloc seront ainsi déjà définis.

La Figure 17 illustre un diagramme de bloc précisant ...

Figure 17. Exemple de diagramme de blocs internes

Diagramme de contexte

Nous avons parlé du diagramme de contexte dans la section précédente sur le diagramme de bloc. Une autre façon de représenter le système dans son environnement est d'utiliser le diagramme de blocs internes :

[Structural description of the SHS Environment IBD view] |

../../SmartHomeUseCase/SmartHomeSystemProject/diagrams/svg/Structural_descript

ion_of_the_SHS_Environment_IBD_view.SVG

Figure 18. Exemple de diagramme de contexte (vue interne)

Tous ces éléments seront détaillés à la Section 9.2.2.

5.7. Diagrammes de séquences (sd)

Le diagramme de séquences (sd) est emprunté à UML® (*Sequence Diagram*) et est utilisé pour représenter les participants et leurs interactions.

Après que les grandes fonctionalités du système aient été identifiées et classifiées (cf. Section 5.4), il convient de préciser les cas d'utilisation en décrivant des scénarios concrets qui précisent, de manière chronologique, les interactions entre acteurs et les différents éléments du systèmes qui collaborent à la réalisation d'un service ou d'une fonctionalité donnée.

Il est souvent intéressant de procéder par étapes, en définissant un premier diagramme représentant les interactions entre le système (vue comme une boîte noire) et les différents acteurs impliqués dans ces interactions. Dans ce diagramme, appelé diagramme de séquences système (dss), le seul participant (en dehors des acteurs) est le système lui-même.

Il est ensuite possible de représenter le même cas d'utilisation, mais en considérant cette fois-ci le système comme une "boîte blanche", c'est-à-dire en précisant les différents participants impliqués par ce cas.

5.7.1. Diagramme de séquences

Le diagramme de séquences permet de représenter les différents participants (dont la ligne de vie représente une chronologie des événements) et les échanges de messages entre eux.

Figure 19. Exemple de diagramme de séquences

Les éléments graphiques utilisés dans ce diagramme sont principalement :

- les *participants* (les boîtes en haut qui représentent soit des acteurs soit des blocs)
- les *lignes de vie* (les traits verticaux pointillés sous les participants.

Le temps se déroule de haut en bas, sans notion particulière d'échelle de temps.

- les *messages* (les flèches horizontales) qui matérialisent les interactions et qui pourront être des événements, des appels de fonction, des signaux, etc.

Tous ces éléments seront détaillés à la [seq].

5.7.2. Diagrammes de séquences système

Les diagrammes de séquences système (`dss`) sont des diagrammes de séquences classiques où seul le système est représenté comme une boîte noire en interaction avec son environnement (les utilisateurs ou les périphériques généralement).

Il permet de décrire les scénarios des cas d'utilisation sans entrer dans les détails. Il convient donc mieux à l'ingénierie système qu'un diagramme de séquences classique.

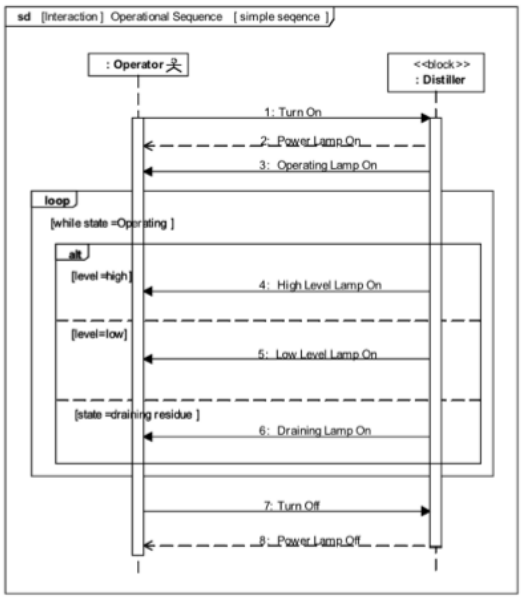


Figure 20. Exemples de dss

xxx à refaire à partir de SHS xxx

5.8. Diagramme d'états (stm)

Les diagrammes d'états-transitions (plus simplement diagramme d'états) sont empruntés à UML® et décrivent le comportement interne d'un objet à l'aide d'un automate à états finis.

Les notions importantes de ce diagramme sont :

- états
- actions
- événements déclencheurs

Tous ces éléments seront détaillés à la Section 9.4.2.

5.9. Diagramme d'activité (act)

Le diagramme d'activités, lui aussi empruntés à UML®, est un diagramme comportemental qui permet de représenter le déroulement d'un processus. Il combine donc des informations concernant la chronologie d'événements mais également le circuit et le traitement d'informations, objets ou flux en fonction de ces événements. Il sera par exemple utilisé pour préciser des cas d'utilisation.

Tous ces éléments seront détaillés à la Section 9.5.3.

5.10. Diagramme paramétrique (par)

Le diagramme paramétrique est un diagramme structurel qui permet d'exprimer des contraintes sur des éléments (des lois physiques par exemple). Ce diagramme est une extension du diagramme de blocs internes (ibd, cf. Section 9.2.2) mais permet de relier les blocs existants à des blocs particuliers (appelés "de contraintes").

Tous ces éléments seront détaillés à la Section 9.3.4.

5.11. Diagramme de *packages* (pkg)

Ce type de diagramme permet d'organiser les éléments de modèle. Un *package* peut contenir d'autres *package*, des blocs, des diagrammes, etc.

Un *package* n'est pas un composant mais une unité d'organisation du modèle. Un *package* constitue un espace de nommage (*namespace*) pour les éléments qu'il contient.

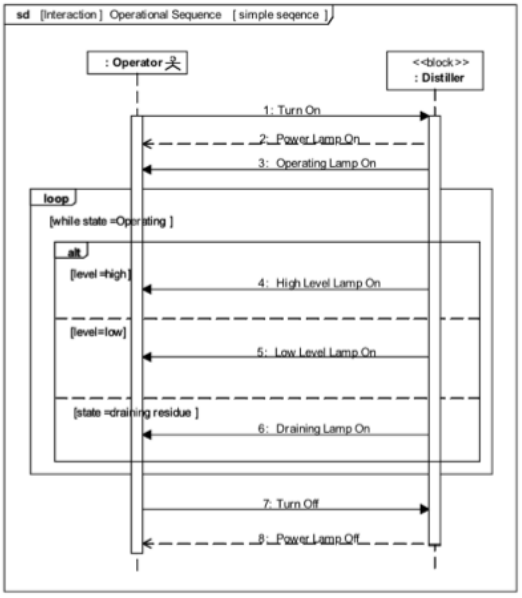


Figure 21. Exemples de diagramme de packages

Tous ces éléments seront détaillés à la Section 10.1.2.

5.12. Allocation, traçabilité et autres points de cohérences

Tous ces éléments seront détaillés au Chapitre 10.

5.13. En résumé

Nous avons survolé dans ce chapitre les différents diagrammes SysML® et ce qu'ils permettent de modéliser. Certains termes peuvent avoir paru flous ou complexe, mais nous allons revenir en profondeur sur chacun de ces diagrammes dans les chapitres suivants, pas d'inquiétude.

Vous n'utiliserez que rarement tous les diagrammes, mais il convient de bien les avoir tous en tête pour apréhender l'utilisation de SysML® dans son ensemble.

5.14. Questions de révision

1. Associez les diagrammes suivants avec leurs acronymes (`sd` , `bdd`, `uc`, `pkg`, `dss`) :
    - Diagramme de paquetages
    - Diagramme des cas d'utilisation
    - Diagramme de séquences système
    - Diagramme de blocs
    - Diagramme de séquences
2. Placez dans la matrice ci-dessous les différents diagrammes SysML® vus dans ce chapitre :

Table 6. Notre matrice des préoccupations

	Exigences	États	Structures	Interactions	Flux	Transversalités
Organisation						
Environnement						
Vision Opérationelle						
Vision Fonctionnelle						
Vision Organique						

[5] On les appelle également parfois diagrammes d'états ou encore automate.

[6] Il s'agit d'une estimation à la louche de notre part.

[7] La partialité des auteurs peut être questionnée, mais d'un point de vue complétude et respect de la norme, cette affirmation est régulièrement admise par la communauté.

[8] rachetée récemment par Dassault Systems

Chapter 6. Langages vs Méthode

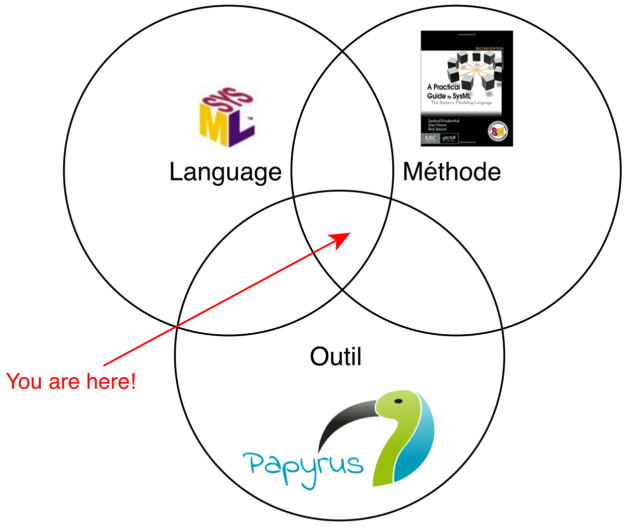


Figure 22. Langages, méthodes et outils : 3 piliers complémentaires (inspiré par [Roques])

Comme illustré par la Figure 22, comprendre SysML® c'est primordiale, savoir utiliser Papyrus-SysML c'est important, mais tout ceci n'est rien sans une démarche ou une méthode concrète pour mettre les deux en musique.

6.1. Une méthode MBSE simplifiée

La Figure 23 résume (dans un diagramme d'activité) une démarche simplifiée que vous pouvez suivre.

[simpleFridenthalmethod] | *simpleFridenthalmethod.svg*

Figure 23. Une démarche MBSE simplifiée (extrait de [Friedenthal2016])

Commentaire

Seb, peux-tu vérifier que la démarche que tu as utilisée pour les modèles est compatible avec cette démarche ?

Organiser les modèles

Comme présenté dans notre matrice des concepts, cette phase d'organisation est primordiale. Il s'agit de définir des *packages* qui vont premettre de retrouver facilement les éléments de modélisation.

Analyser les besoins des *stakeholders*

Il convient de démarrer par les besoins des parties prenantes. Comprendre le problème à résoudre est déjà un pas énorme vers la solution. Il s'agit dans cette phase de définir les acteurs, de cerner le domaine précis à modéliser, le contexte du système, les cas d'utilisation de haut niveau. Si possible, c'est aussi dans cette phase qu'il faut déterminer les éléments de mesure de satisfaction qui permettront de déterminer la pertinence des solutions proposées.

Spécifier les exigences système

Phase incontournable de toute approche d'[IS], l'expression des exigences, qu'elles soient textuelles ou graphiques est primordiale. Cette phase permettra de concrétiser et documenter les résultats de la phase précédente. Il s'agira de définir des diagrammes d'exigences, mais aussi des précisions (diagrammes d'activités ou de séquences système) pour les cas d'utilisation, et de préciser le diagramme de contexte.

Synthétiser des solutions alternatives

Il s'agit de la phase de conception à proprement parlé. On décompose le système en sous-systèmes (diagrammes de blocs). On définit les interactions entre les éléments (diagrammes d'activité) ainsi que leurs connexions (diagrammes de blocs internes).

Analyser les modèles

Il s'agit ici de modéliser les éléments utiles aux analyses et simulations (phases non traitées dans ce livre) au travers des détails dans les diagrammes de blocs ou encore au travers des diagrammes paramétriques.

Maintenir la traçabilité des exigences

Phase souvent négligée, il conviendra de bien veiller à systématiquement lier les modèles entre eux. Certains liens seront obtenus "par construction". Par exemple, réaliser un diagramme d'état à partir d'un click droit sur un bloc **New Diagram › SysML 1.4 State Machine Diagram** insèrera directement ce diagramme "dans" le bloc au niveau du *model explorer*. Mais pour tous les autres liens (une interaction qui `<<satisfy>>` une exigence, etc.) il vous faudra rigoureusement les ajouter manuellement.

Il s'agit d'une démarche simplifiée, qui fait abstraction d'étapes importantes comme la plannification, les analyses de risques, la gestion des configurations, etc.

Dans la suite nous présentons rapidement deux démarches plus complètes qui font référence en matière de MBSE utilisant SysML®, puis nous présenterons la méthodes que nous allons suivre dans cet ouvrage, en l'adaptant à SysML®.

6.2. OOSEM

OOSEM (*Object-Oriented Systems Engineering Method*) est certainement la méthode MBSE la plus utilisée en combinaison de SysML®, puisqu'elle est promue par Stanford Friedenthal, le principal leader de SysML®. Pour plus d'information sur cette méthode, cf. [Friedenthal2016].

6.3. SYSMOD

La méthode SYSMOD (*Systems Modeling Toolbox*) est une approche pragmatique pour modéliser les exigences et l'architecture d'un système. Elle fournit une boîte à outil et des étapes précises, ainsi que des guides et des "bonnes pratiques". Pour plus d'information sur cette méthode, cf. [Wielkins20XX].

6.4. CESAM

CESAM (CESAMES Architecture Method) est un framework d'architecture et de modélisation très connue en France. Nous allons la détailler au Chapitre 8 en l'adaptant à SysML® et en la plaçant dans l'esprit de la méthode simplifiée présentée en Section 6.1.

6.5. Autres méthodes et démarches

Pour plus de renseignements sur les différentes démarches de MBSE nous renvoyons le lecteur à l'étude menée par l'INCOSE par [Estefan]. Voir aussi [Ramos] ou [Dickerson].

Commentaire

To be completed!!

6.6. En résumé

...

Table 7. Organisation

	Exigences	États	Structures	Interactions	Flux	Transversalités
Organisation						
Environnement						
Vision Opérationelle						
Vision Fonctionnelle						
Vision Organique						

6.7. Questions de révision

1. ...

Chapter 7. Getting Started with Papyrus-SysML

Commentaire

Ce chapitre est un tuto Papyrus.

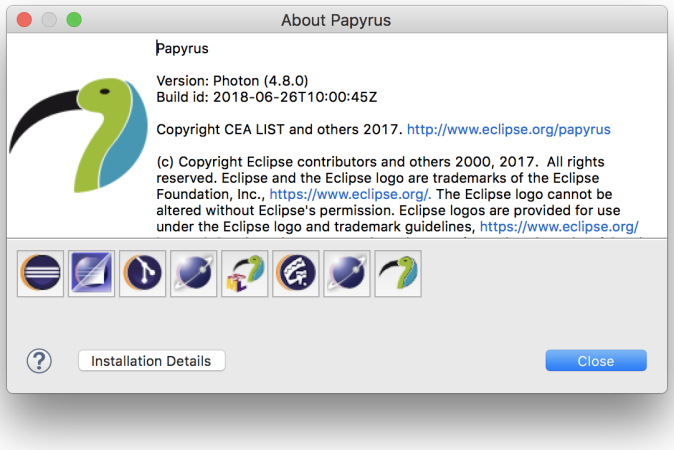
7.1. Fondements

	Exigences	Structures	Interactions	Transversalités
Organisation	You are here!	You are here!	You are here!	You are here!
Vision Opérationelle				
Vision Fonctionnelle				
Vision Organique				

Nous allons aborder dans ce chapitre l'utilisation de Papyrus-SysML. Le lecteur pourra trouver des informations complémentaires sur la page de documentation : https://wiki.eclipse.org/Papyrus_User_Guide.

7.2. Configuration

Comme indiqué au chapitre Chapitre 4, nous considérons que vous avez la version `4.0` (ou plus) de Papyrus-SysML, pour la version *Photon* (`4.8`) d'eclipse.



N'oubliez pas qu'un certain nombre de modèles "tous prêts" sont disponibles ici : https://github.com/PapyrusSysMLinAction/SmartHomeUseCase.

7.3. Préparation et organisation

Voici l'organisation que nous allons utiliser pour notre étude de cas.

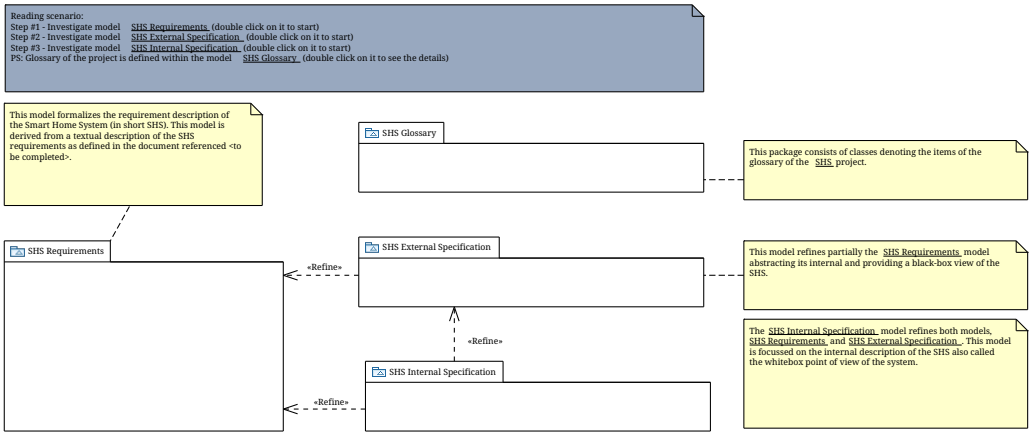
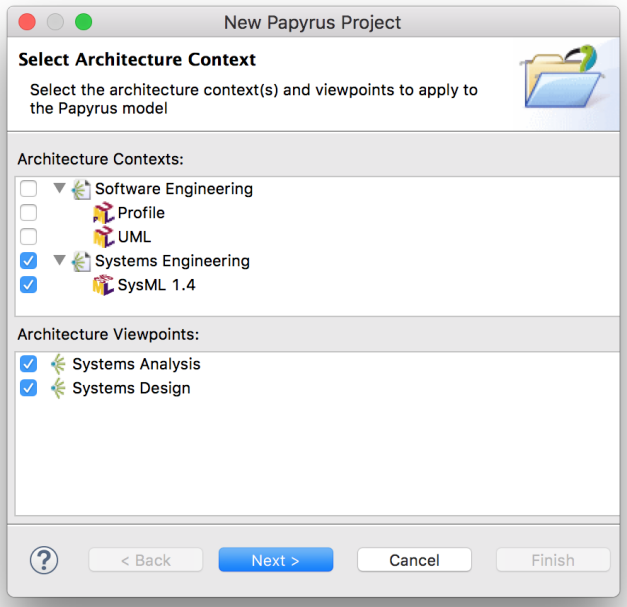


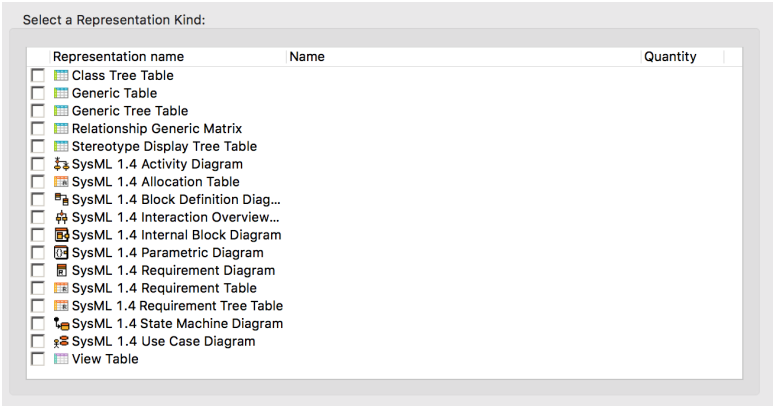
Figure 24. Organisation type d'un projet Papyrus

Créez votre premier projet :

1. Sélectionnez **File › New › Papyrus Project** et cochez `SysML 1.4` puis **[Next]**



2. Donnez un nom à votre projet
3. Cliquez sur **[Next]** (ou sur **[Finish]** si vous ne souhaitez pas configurer les éléments de départ)
4. Choisissez les éléments dont vous savez déjà que vous allez avoir besoin



5. Cliquez sur **[Finish]**. avez créé votre 1er projet .

Vous pouvez aussi démarrer d'un projet UML®. Par contre il faut changer en cliquant-droit sur le modèle dans le *Model Explorer* et en sélectionnant **Switch Architecture Context** puis en cochant *Systems Engineering*.

7.4. L'environnement de modélisation

Comme de nombreux outils basés sur eclipse, Papyrus-SysML propose une organisation type des fenêtres et panneaux, appelée *perspective*.

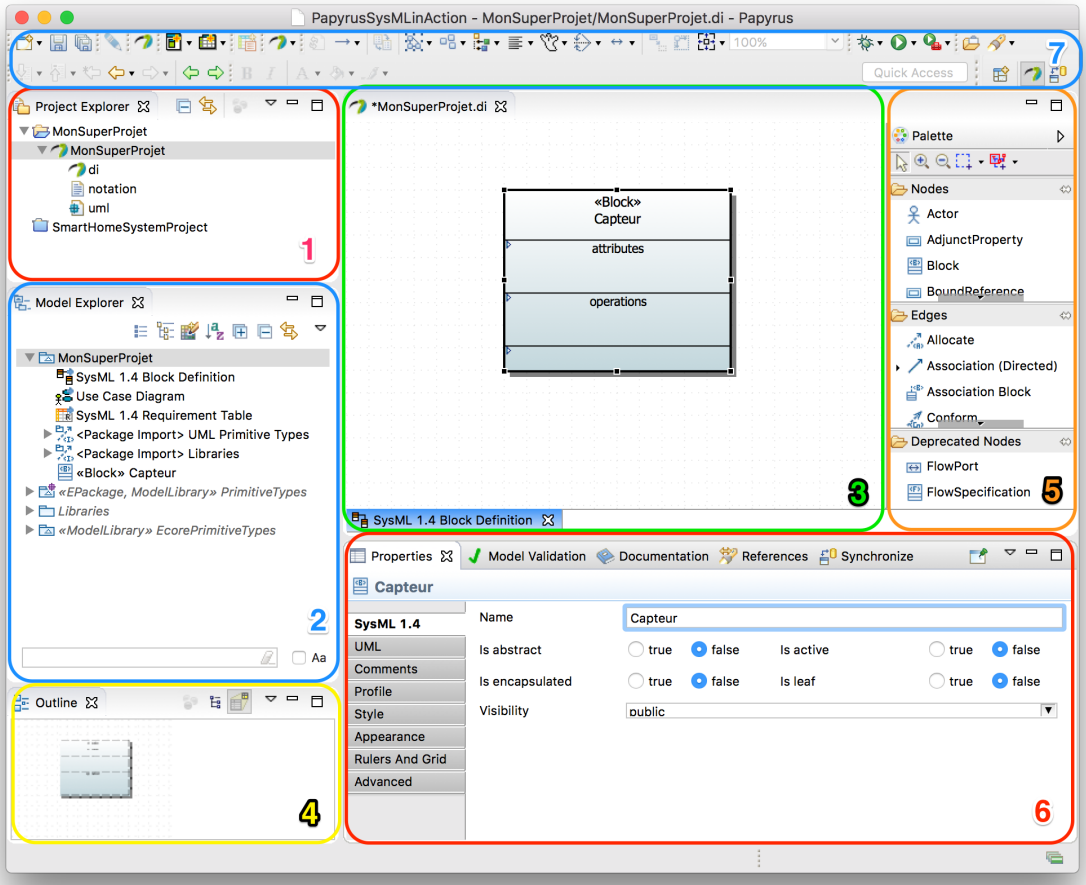


Figure 25. La perspective de base Papyrus-SysML

Les éléments principaux de cette organisation sont :

1. Le *Project Explorer*, élément eclipse qui vous permet d'accéder à vos projets (pas seulement Papyrus-SysML)
2. Le *Model Explorer*, qui permet d'accéder à vos modèles et à tous vos artefacts de modélisation (cf. section Section 7.6).
3. L'*Outline*, sorte de vue d'ensemble du diagramme ouvert (utile pour les très grands diagrammes)
4. L'éditeur de diagramme (*Multi diagram editor*), l'élément principal dans lequel vous ouvrirez et concevrez vos diagrammes.
5. La *Palette*, qui dépend du type de diagramme ouvert dans l'éditeur de diagramme et qui vous permet de "copier/coller" des éléments dans votre diagramme (cf. section Section 7.5)
6. La *Properties view*, qui permet d'accéder à des informations détaillées. Généralement, l'onglet le plus utilisé est l'onglet *Properties* qui permet d'accéder et de renseigner les détails de l'élément sélectionné (ici le bloc `Capteur`)
7. La barre de menu (*Toolbar*), qui contient certains raccourcis de manus sous forme d'icônes.

Pour retrouver la perspective Papyrus-SysML ou en changer : **Window › Perspective** (cf. Figure 26) ou cliquez sur l'icône Papyrus en haut à droite (Figure 27).

Figure 26. Menu Window > Perspective

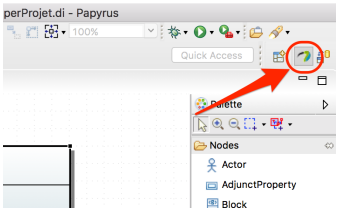


Figure 27. Pour activer la perspective Papyrus

7.5. Modélisation par les diagrammes

7.5.1. Exemple simple

Commençons à manipuler Papyrus-SysML avec un premier diagramme…

Nous conseillons d'éviter les caractères spéciaux ou les blancs dans les noms des éléments de modélisation. Cela peut en effet poser des problèmes plus tard (exécution de modèles, génération de documentation, etc.).

Pour notre modélisation du SmartHomeSystem, nous utilisons l'organisation suivante[9] :

[Outline of the Smart Home System SHS modelling project organisation] |

../../../SmartHomeUseCase/SmartHomeSystemProject/diagrams/Outline_of_the_Smart_Home_System_S

HS_modelling_project_organisation.PNG

Figure 28. Vue d'ensemble de notre projet de SmartHomeSystem

Comme vous pouvez le constater, nous annotons les éléments de modèles avec des commentaires, et pas seulement parce qu'il s'agit d'un exemple illustratif pour un livre, comme nous le détaillons dans la section suivante.

7.5.2. Annoter les éléments de modèles

Il est important de documenter les modèles. Si vous souhaitez ensuite pouvoir générer une documentation intéressante (cf. section [Gen2Doc]) ou analyser automatiquement les commentaires.

1. Pour ajouter ou supprimer des éléments de modèles sur lequel porte la note, contrôlez la partie *Annotated element* des propriétés (onglet UML).

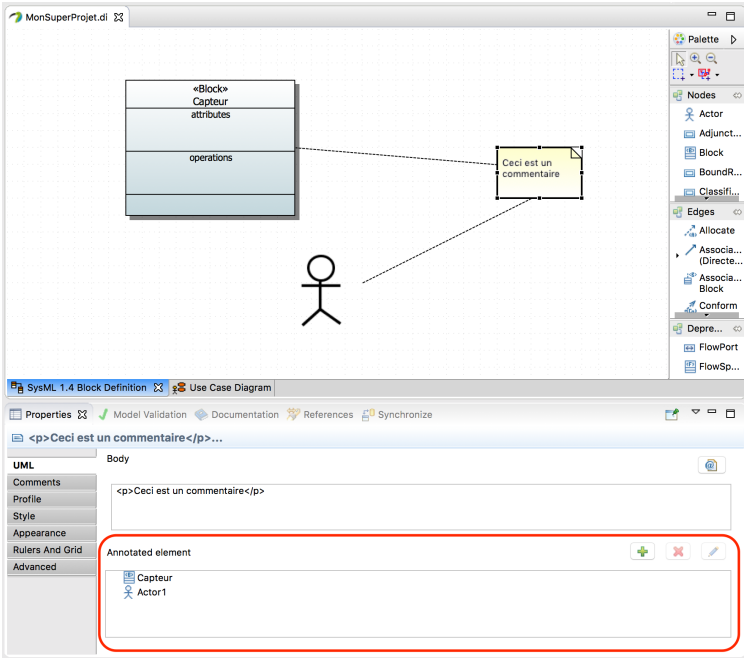


Figure 29. Lien entre une note et les éléments de modèles

2. Vous pouvez éditer des notes richement formatées. Il faut pour cela activer l'éditeur enrichi : **Preferences › Papyrus › Rich text** et cochez les cases.

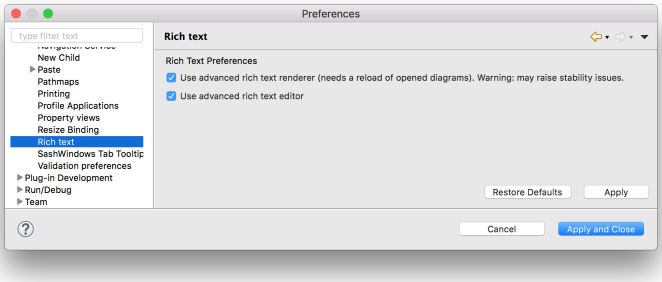


Figure 30. Activation de l'éditeur interne enrichi

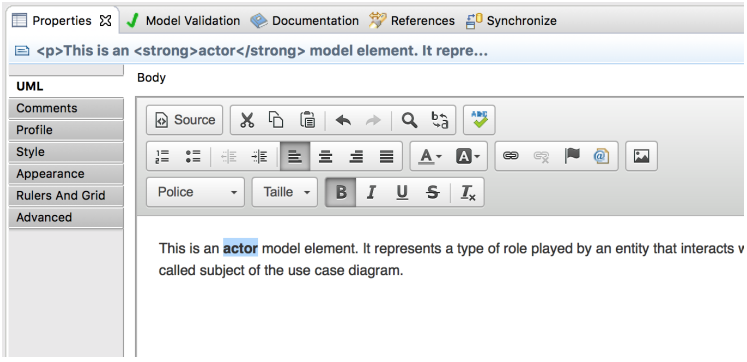


Figure 31. Nouveaux menus disponibles dans l'éditeur interne

7.6. Modélisation par les artefacts

7.6.1. Export des diagrammes

Si vous souhaitez simplement obtenir des figures à partir de vos diagrammes :

1. sélectionnez un modèle puis **File › Export › Export All Diagrams...**
2. choisissez le répertoire d'export et le format (PNG, GIF, SVG, PDF, ...)

Nous vous recommandons de sélectionner **Prefix with qualified names**.

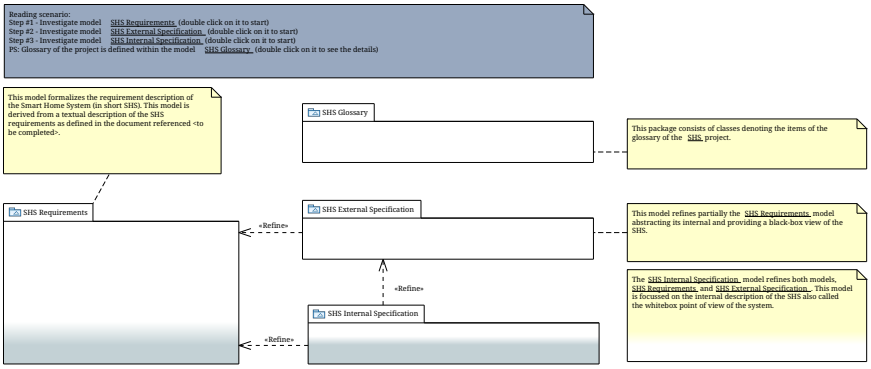


Figure 32. Exemple d'export de diagrammes

Vous constaterez que l'export ne conserve pas l'entête (cartouche). C'est une des fonctionnalités attendues des utilisateurs et en cours de prise en compte par les développeurs du CEA LIST. Nous vous invitons à soutenir cette évolution en allant voter sur : https://bugs.eclipse.org/bugs/show_bug.cgi?id=354296.

7.6.2. Model2Doc

Le meilleur moyen de générer une documentation à partir des modèles Papyrus-SysML est d'utiliser le complément Model2Doc. Pour cela installez le complément eclipse en utilisant le site suivant : https://download.eclipse.org/modeling/mdt/papyrus/components/model2doc/.

7.7. En résumé

La prise en main d'un outil comme Papyrus-SysML n'est pas quelque chose que l'on peut facilement résumer en quelques pages. Néanmoins, le fait qu'il soit basé sur eclipse permettra à ceux qui ont déjà l'habitude de cet IDE de ne pas être dépaysés.

Pour ceux qui veulent véritablement maîtriser l'outil, nous renvoyons le lecteur aux tutoriels disponibles sur le site : https://www.eclipse.org/papyrus/documentation.html.

Commentaire

Ce site n'est pas un ebonne vitrine, les tutos datent un peu. On met quoi ?

7.8. Questions de révision

1. Quelle est la version actuelle de Papyrus-SysML ?
2. Pourquoi y-a-t'il plusieurs façon de supprimer un élément dans Papyrus-SysML ?
3. Quel est l'intérêt d'avoir un outil de modélisation basé sur eclipse ?

[9] Issue du CEA LIST.

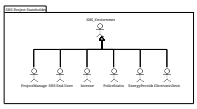
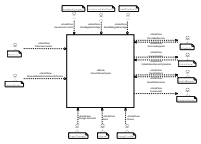
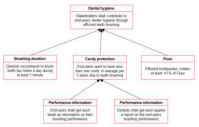
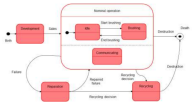
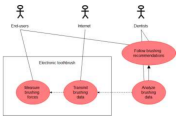
Chapter 8. Mise en œuvre de CESAM

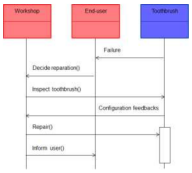
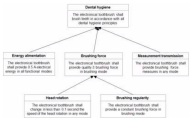
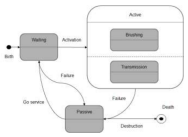
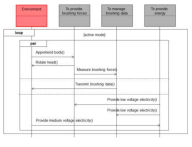
Commentaire

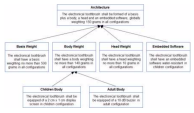
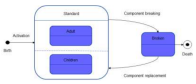
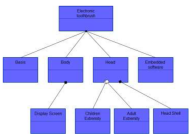
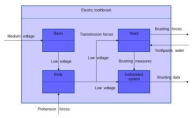
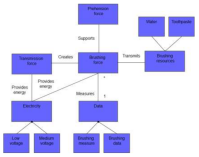
Déclinaison de la méthode CESAM avec SysML et Papyrus.

Pour ceux qui sont familiarisés avec CESAM et qui voudraient directement aller voir comment nous proposons de la mettre en œuvre avec SysML®, la Table 8 présente la liste des principaux produits (*deliverables*) de CESAM et les sous-sections correspondantes, où nous abordons comment les réaliser en SysML® :

Table 8. Les "produits" CESAM

Produits	Section	Illustration (from [CESAM17])
Stakeholder hierarchy diagram	8.3.2.1	
Environment diagram	8.3.2.2	
Need architecture diagram	8.4.2.2	
Life-cycle diagram	8.4.2.3	
Use case diagrams	8.4.2.4	

Produits	Section	Illustration (from [CESAM17])
Operational scenarios diagrams	8.4.2.5	
Operational flow diagrams	8.4.2.6	
Functional requirement architecture diagram	8.5.2.1	
Functional mode diagram	8.5.2.2	
Functional decomposition diagrams	8.5.2.3	
Functional interaction diagrams	8.5.2.3	
Functional scenario diagrams	8.5.2.4	

Produits	Section	Illustration (from [CESAM17])
Functional flow diagrams	8.5.2.5	
Constructional requirement architecture	8.6.2.1	
Constructional mode diagram	8.6.2.2	
Constructional decomposition	8.6.2.3	
Constructional interaction diagrams	8.6.2.3	
Constructional scenario diagrams	8.6.2.4	
Constructional flow diagrams	8.6.2.5	

8.1. Fondements

Comme abordé en Section 6.4, CESAM (*CESAMES Architecture Method*) est un *framework* d'architecture et de modélisation développé par CESAMES™ et dédié à la collaboration entre les arhitectes systèmes, les ingénieurs, pour les aider à maîtriser la complexité des systèmes intégrés.

Nous avons choisi dans ce livre de suivre cette démarche, en l'applicant sur notre cas d'étude et en utilisant la notation SysML® pour implémenter les produits et diagrammes préconisés.

Le principal intérêt de cette approche est qu'elle est conforme aux standards de l'INCOSE, qu'elle est suivie par de nombreux membres industriels français de l'AFIS et que nous avons rencontré une attente auprès des utilisteurs de SysML® de pouvoir suivre une démarche standard. Elle possède également l'avantage de proposer un cadre simple et complet en terme de points de vues abordés et de couverture des éléments à modéliser. Enfin, elle est tout à fait compatible avec la méthode simplifiée présentée à la Section 6.1.

Les principaux éléments de l'architecture sont :

- Une organisation hiérarchique des préoccupations (Quoi ? / Qui ? / Où ?), représentée par une "Pyramide d'Architecture Système" (cf. Figure 33).

La question du "Qui" porte ici sur le "*What*", c'est-à-dire "quelle partie du système". Pour la question du "qui est concerné par le système", elle est traitée en amont de ces éléments architecturaux, dans ce que CESAM appelle l'*Environment Architecture*, que nous avons appelé le Contexte du système (cf. Section 8.3).

- Une organisation en 9 vues du système, représentée par une matrice (cf. Figure 34).
- Une démarche guidant l'architecture du système, représentée par un processus (cf. Figure 35).

Commentaire

Figures à refaire pour les aspects © et pour plus de lisibilité!

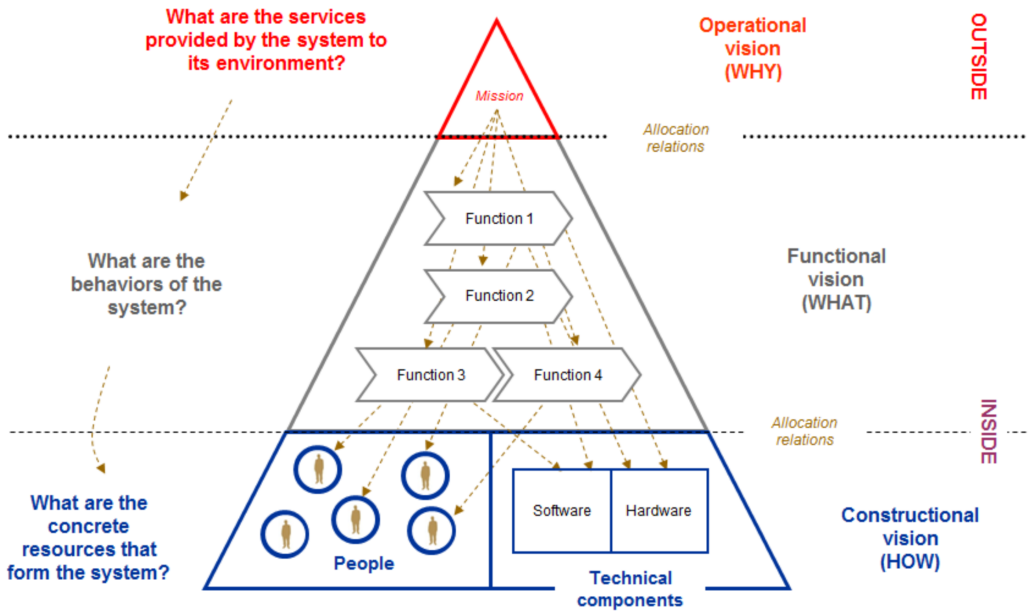


Figure 33. La Pyramide d'Architecture Systèmes de CESAM (extrait de [CESAM17])

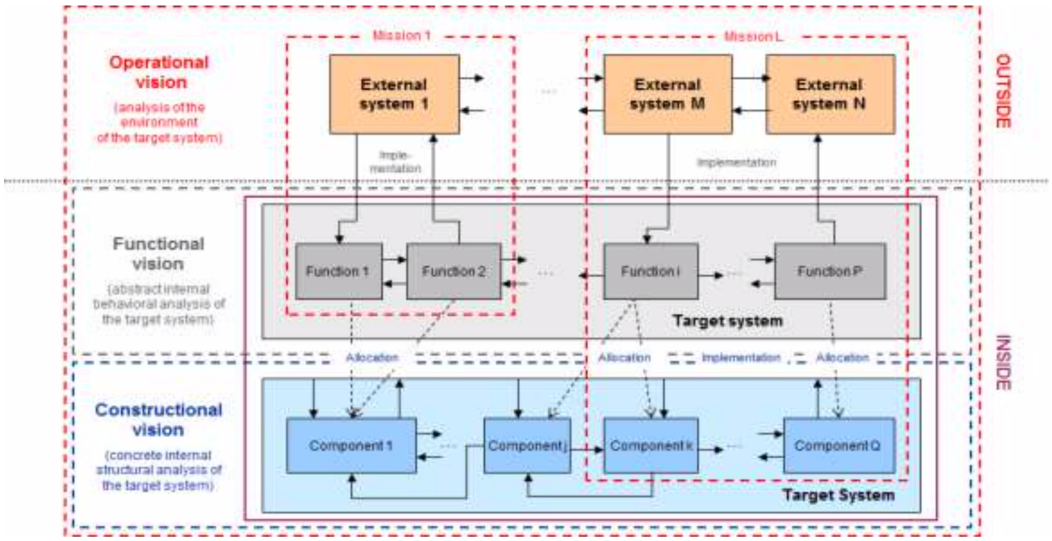


Figure 34. La Matrice d'Architecture Systèmes de CESAM (extrait de [CESAM17])

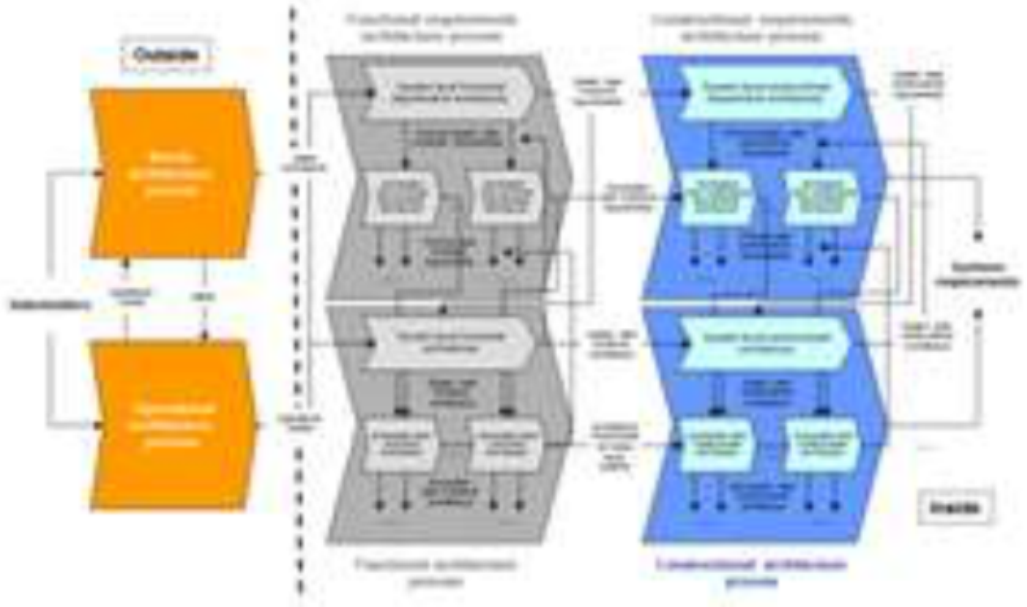


Figure 35. Le Processus d'Architecture des Systèmes de CESAM (extrait de [CESAM17])

Comme abordé en Section 2.3, nous avons choisi de synthétiser ces différents points de vue dans une matrice qui nous servira de "carte de base" pour placer les activités ou les modèles (cf. Table 9).

Table 9. Notre matrice des préoccupations

	Exigences	États	Structures	Interactions	Flux	Transversalités
Organisation						
Environnement						
Vision Opérationelle						
Vision Fonctionnelle						
Vision Organique						

Cette matrice permet de situer les différents éléments qui seront abordés dans un cadre utile pour comparer ces éléments les uns aux autres.

Les étapes Exigences, Structures, Interactions ouTransversalités ont des chapitres ou sections dédiés, tandisque les étapes États et Flux sont distilées comme des sous-section systématiques dans les sections qui suivent.

Voici un schéma qui présente le processus CESAM :

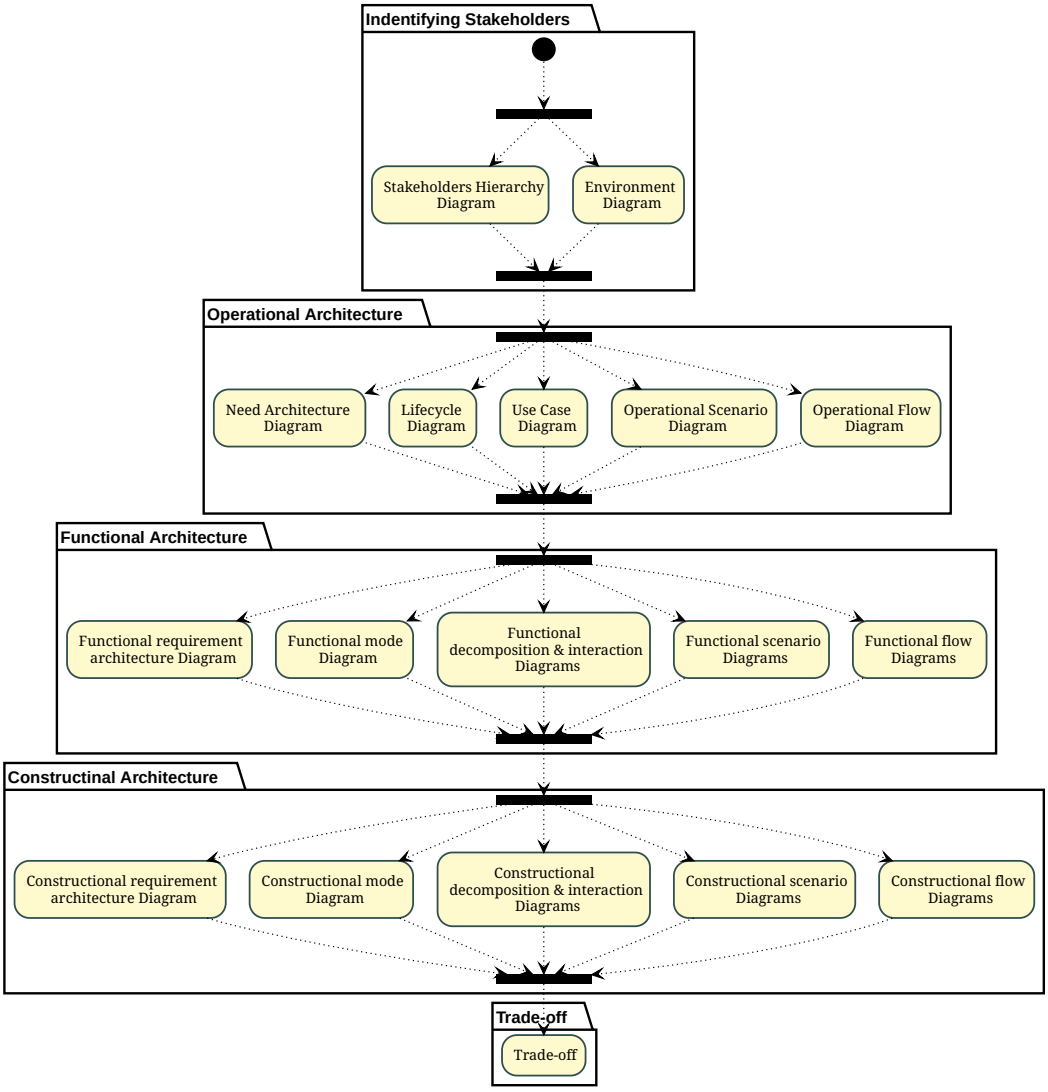


Figure 36. Les grandes étapes de CESAM

8.2. Organisation des modèles

Commentaire

Avant les visions, question du contexte et du WHO.

8.2.1. Principes

Blabla sur cette partie en générale.

	Exigences	États	Structures	Interactions	Flux	Transversalités
Organisation	You are here!					
Environnement						
Vision Opérationelle						
Vision Fonctionnelle						
Vision Organique						

8.2.2. Mise en œuvre en SysML

Concepts définis dans cette section

Nous aborderons dans cette partie les concepts du niveau initial (*Model User*) de la certification *Certified Systems Modeling Professional*™: package, model, dependency, comment.

Commentaire

Vérifier les termes (absents ou en trop)!

Nous proposons d'organiser le modèle en 2 *packages* séparant clairement les aspects développement (projet) et les aspects produit (système), conformément aux préconisations de CESAM.

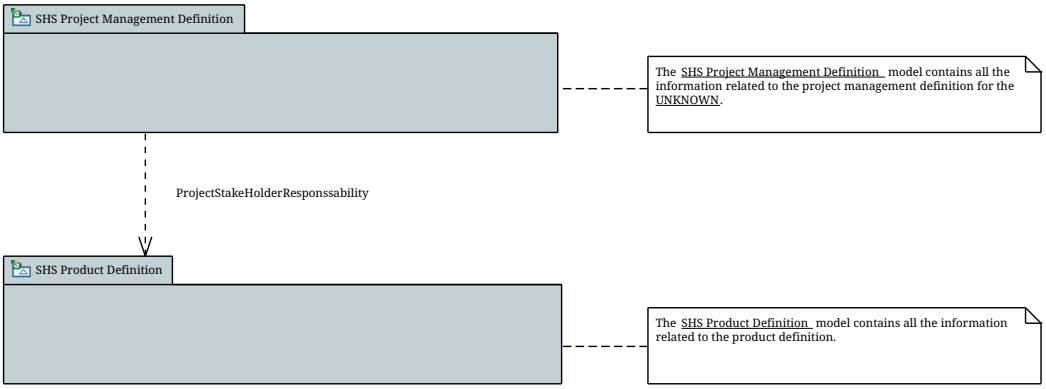


Figure 37. Exemple d'organisation principale

Pour réaliser ce diagramme en Papyrus-SysML :

1. Créez votre projet Papyrus-SysML si ce n'est déjà fait :
    - **File › New › Papyrus Project**

- Cochez la case **[SysML 1.6]** puis **[Next]**
- Donnez un nom à votre projet puis **[Next]**
- Choisissez votre premier diagramme : un **[SysML 1.6 Block Definition Diagram]** puis **[Finish]**

Parler du template `

2. Double-cliquez dans votre `Project Explorer` sur le modèle portant le même nom que le projet
3. Ouvrez le `Model Explorer` puis Double-cliquez sur le modèle
4. Glissez-Déposez deux `Model` depuis la palette

Notez la création des éléments de modèle correspondants dans le `Model Explorer`

5. Renommez les en conséquence
6. Reliez-les par une `Dependency` (dans l'onglet `General Annotaions` de la palette)
7. Etc.

Suite de l'organisation …

Parlez de l'organisation des stakeholders en *packages* (`Key blabla`). TIP: Parler des `ElementGroup` ⇒ organisation logique (adopté par SysML 2.0)

8.2.3. En résumé

8.2.4. Questions de révision

1. Quelle est la différence entre un *package* de type ***model*** et un *package* de type ***package*** ?

8.3. Contexte du système

Commentaire

Avant les visions, question du contexte et du WHO.

8.3.1. Principes

Blabla sur cette partie en générale.

	Exigences	États	Structures	Interactions	Flux	Transversalités
Organisation						

	Exigences	États	Structures	Interactions	Flux	Transversalités
Environnement	You are here!					
Vision Opérationelle						
Vision Fonctionnelle						
Vision Organique						

8.3.2. Mise en œuvre en SysML

Concepts définis dans cette section

Nous aborderons dans cette partie les concepts du niveau initial (*Model User*) de la certification *Certified Systems Modeling Professional*™: use case diagram, actor, generalization, package, block definition diagram, block, item flows, comment.

Commentaire

Vérifier les termes (absents ou en trop)!

Les acteurs sont bien connus de SysML® / UML®. Nous allons donc utilise un diagramme des cas d'utilisation pour lister les parties prenantes et leur hiérarchie éventuelle. Nous placerons ensuite dans un *Block Definition Diagrams* le système (le SmartHomeSystem dans notre exemple) et ses interactions avec les parties prenantes. Pour représenter les flux, SysML® possède le concept d'*Item Flow*, qui correspond parfaitement à ce besoin.

Diagramme des parties prenantes (*Stakeholder Hierarchy Diagram*)

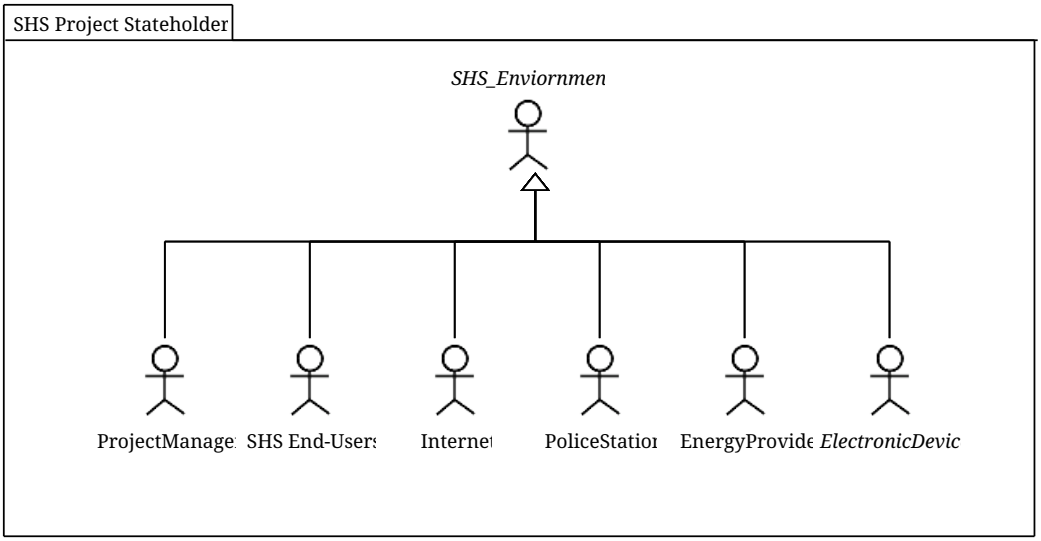


Figure 38. Exemple de Diagramme des parties prenantes

Il existe un stéréotype SysML® <<Stakeholder>> qui peut être utilisé pour clairement accentuer le rôle de ces acteurs.

Environment Diagram

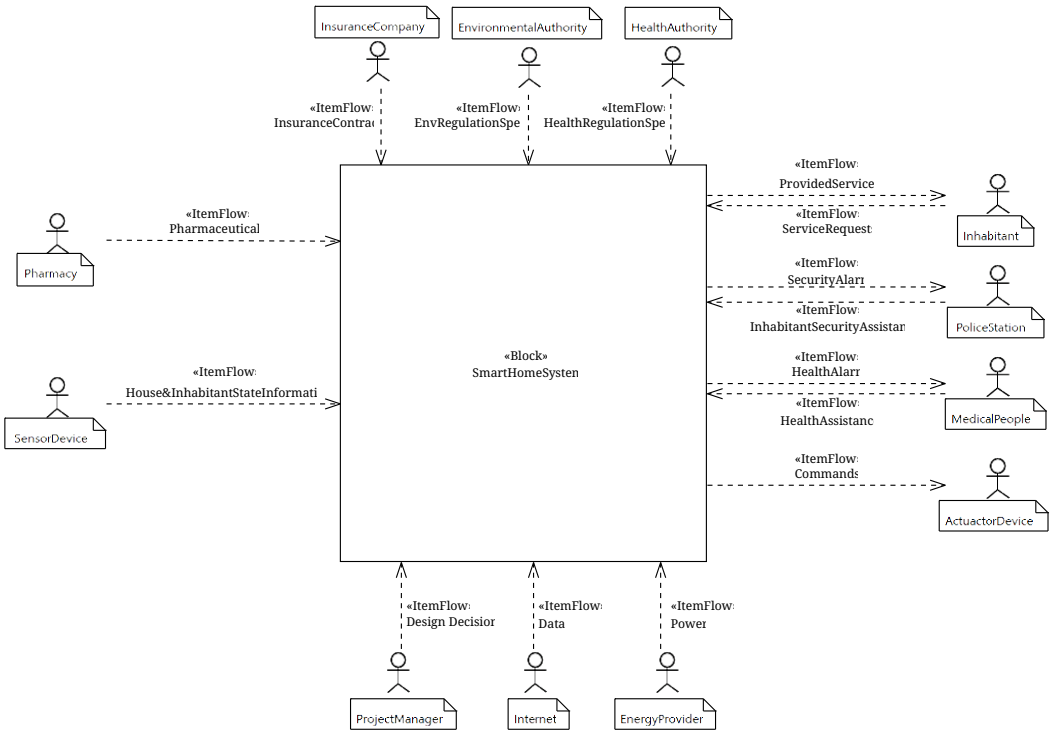


Figure 39. Exemple de Diagramme d'environnement projet

8.3.3. En résumé

8.3.4. Questions de révision

1. Quelles sont les différences entre **utilisateurs** et **parties prenantes** ?
2. Pourquoi regrouper les parties prenantes en "en lien avec le Projet" ou "en lien avec le Produit" ?
3. Quelle est la différence entre un *package* de type ***model*** et un *package* de type ***package*** ?

8.4. Vision Opérationnelle

Commentaire

Vision opérationelle.

8.4.1. Principes

Blabla sur cette partie en générale.

	Exigences	États	Structures	Interactions	Flux	Transversalités
Organisation						
Environnement						
Vision Opérationelle	You are here!					
Vision Fonctionnelle						
Vision Organique						

Concepts définis dans cette section

Nous aborderons les concepts du niveau initial (*Model User*) de la certification *Certified Systems Modeling Professional*™: requirement, requirement diagram, deriveRqt, verify, satisfy, refine, trace, containment.

8.4.2. Mise en œuvre en SysML

Organisation

Pour aborder la vision organisationnelle en Papyrus-SysML il nous faut tout d'abord récupérer les parties prenantes en lien direct avec le système parmi les celles définies à l'étape précédente (cf. Section 8.3.2.1) :

1. Copiez tout le package `SHS Project Stakeholders` dans `SHS_OperationalVision`
2. Renommez-le en `SHS_ProductStakeholders`
3. Supprimez toutes les parties prenantes qui ne sont liées qu'au projet et non au produit directement
4. Réalisez une matrice de dépendance (cf. Section 10.2) qui fera correspondre en colonne les parties prenantes de project avec en ligne les parties prenantes du produit.

Exigences

	Exigences	États	Structures	Interactions	Flux	Transversalités
Organisation						

	Exigences	États	Structures	Interactions	Flux	Transversalités
Environnement						
Vision Opérationelle	You are here!					
Vision Fonctionnelle						
Vision Organique						

À ce niveau d'abstraction il s'agit d'énumérer les besoins utilisateurs (`Needs`). Pour cela nous allons utiliser un diagramme d'exigences SysML®. Il s'agit d'organiser les besoins en en proposant une décomposition hiérarchique (cf. Figure 40). Nous en profiterons pour associer les besoins aux parties prenantes (d'où l'importance de les avoir préalablement "importés", cf. Section 8.4.2.1).

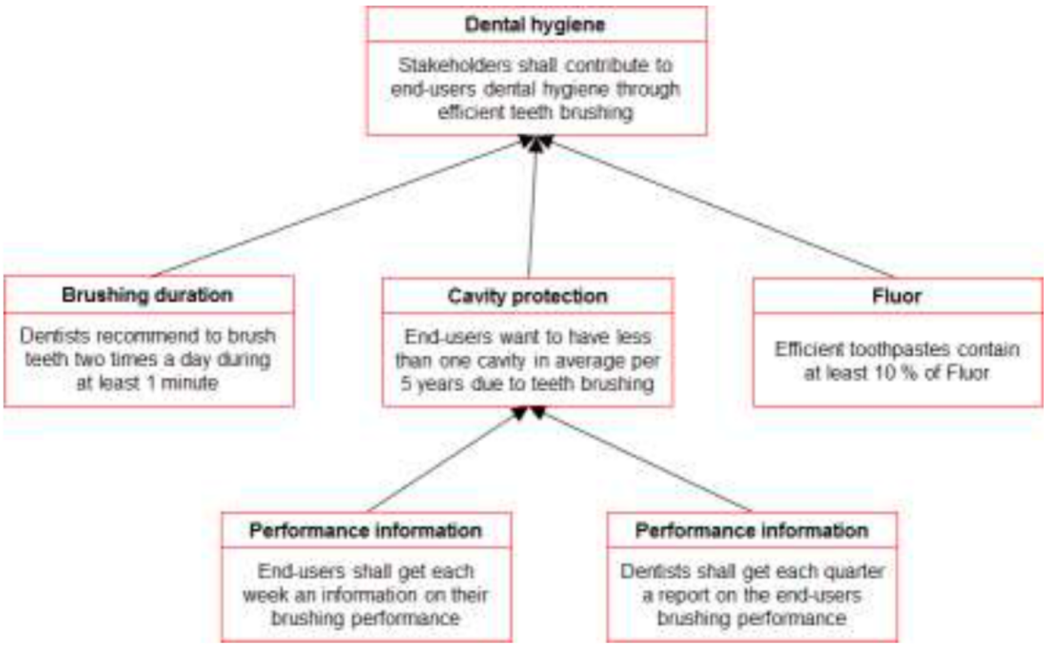


Figure 40. Exemple de Diagramme des besoins (extrait de [CESAM17])

Pour réaliser un tel diagramme en Papyrus-SysML, nous proposons la démarche suivante :

1. Placez-vous dans le *package* `SHS_Needs` dans `SHS_OperationalVision`
2. Créez un *package* par parties prenantes concrètes
3. Pour chaque *package*, créez un nouveau diagramme d'exigences
4. Organisez les besoins en les décompants (cf. discussions sur les exigences en Section 9.1)

Cette activité est bien plus facile si vous la faites en mode modèle plutôt qu'en mode diagramme (cf. Section 7.5).

5. Réalisez la traçabilité entre les besoins et les parties prenantes (cf. Section 10.2)

On obtient la Figure 41.

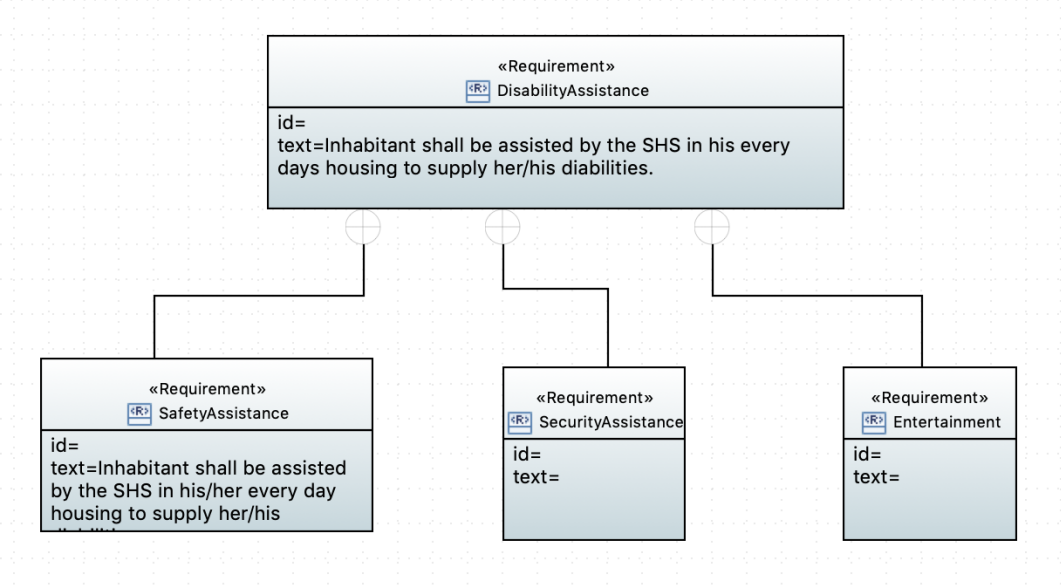


Figure 41. Exemple de Diagramme des besoins

États

	Exigences	États	Structures	Interactions	Flux	Transversalités
Organisation						
Environnement						
Vision Opérationelle		You are here!				
Vision Fonctionnelle						
Vision Organique						

Il s'agit dans cette partie de définir les grandes étapes du cycle de vie du système. Cela permet de définir :

- les contextes opérationels du système (parfois appelés mode dans certains domaines) ainsi que leur relations temporelles (de précédance, de concurrence, d'inclusion, ...
- les événements qui provoquent les différentes transitions entre ces contextes.

En voici un exemple :

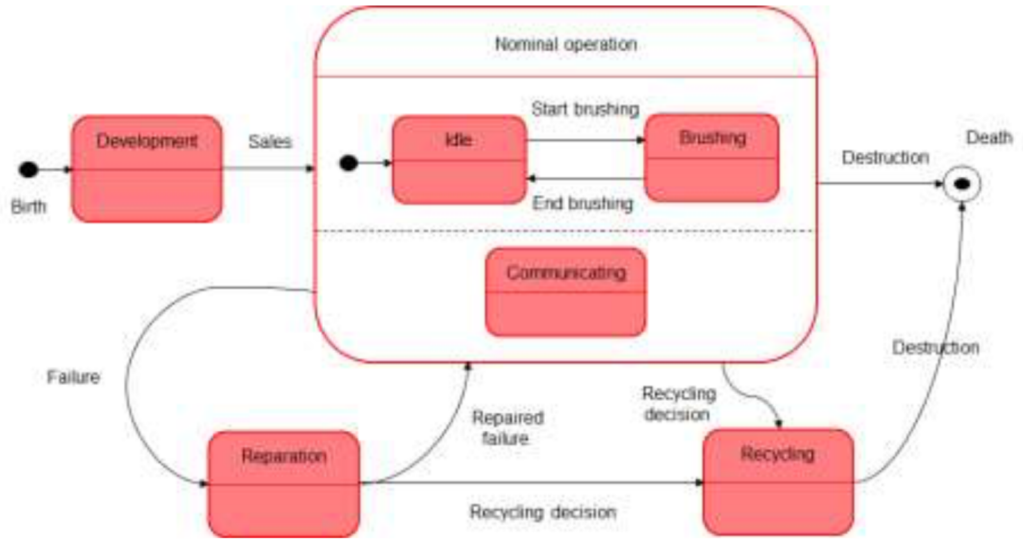


Figure 42. Exemple de Diagramme des cycles de vie (extrait de [CESAM17])

Pour réaliser un tel diagramme en Papyrus-SysML, nous proposons la démarche suivante :

1. Placez-vous dans le *package* `SHS_OperationalDescription` dans `SHS_OperationalVision`
2. Repérez le `SmartHomeSystem` et créez un diagramme d'état en cliquant-droit sur `SmartHomeSystem` : **New_Diagram › SysML 1.6 State Machine Diagram**

On obtient le diagramme des cycles de vie du système comme illustré à la Figure 43.

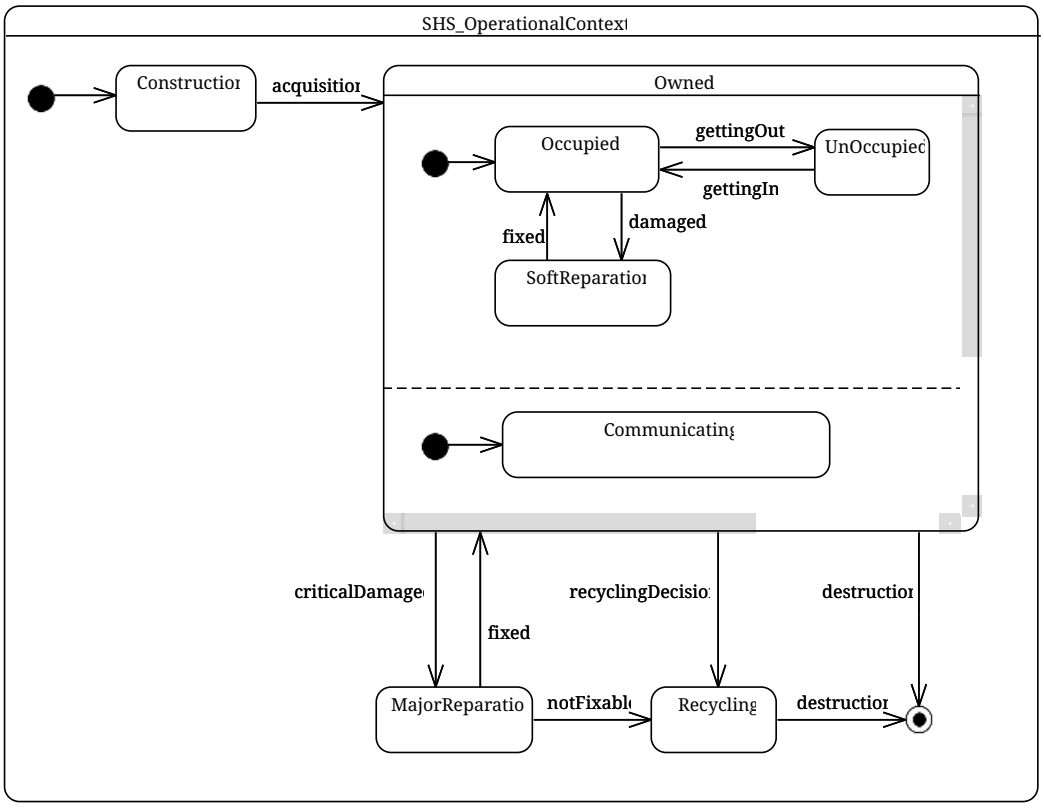


Figure 43. Exemple de Diagramme des cycles de vie

Structures

	Exigences	États	Structures	Interactions	Flux	Transversalités
Organisation						
Environnement						
Vision Opérationelle			You are here!			
Vision Fonctionnelle						
Vision Organique						

Dans cette étape, nous allons représenter de manière statique l'ensemble des missions du système à développer en précisant les collaborations entre le système et son environnement, ainsi que les relations entre ces missions (cf. Figure 44).

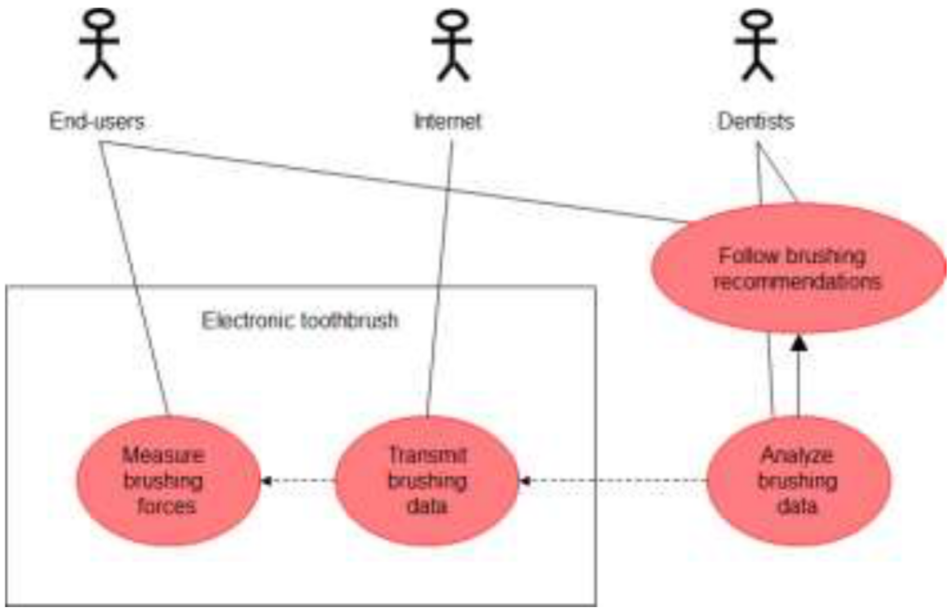


Figure 44. Exemple de Diagramme des cas d'utilisation (extrait de [CESAM17])

Pour recenser ces éléments de modélisation, nous allons réaliser un diagramme des cas d'utilisation en Papyrus-SysML, selon la démarche suivante :

1. Toujours dans le *package* `SHS_OperationalDescription` de `SHS_OperationalVision`, placez vous dans le *package* `SHS_Missions`
2. Créez éventuellement une structure en *packages* pour organiser les missions.
3. Créez un ou plusieurs diagrammes des cas d'utilisation dans ces *packages* : **New_Diagram ›**

SysML 1.6 Use Case Diagram

On obtient des diagrammes de cas d'utilisation comme illustré à la Figure 45.

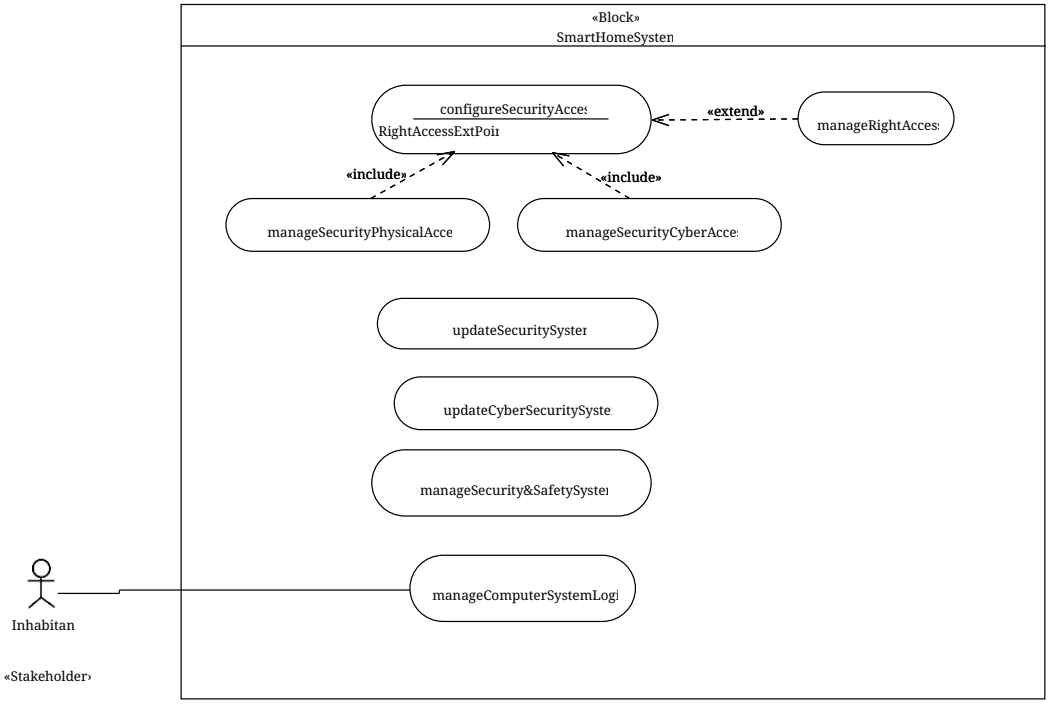


Figure 45. Exemple de Diagramme des cas d'utilisation

Interactions

	Exigences	États	Structures	Interactions	Flux	Transversalités
Organisation						
Environnement						
Vision Opérationelle				You are here!		
Vision Fonctionnelle						
Vision Organique						

Nous allons maintenant nous intéresser à la description de scénarios opérationnels qui vont représenter la dynamique des missions dans des contextes différents (cf. exemple de la Figure 46).

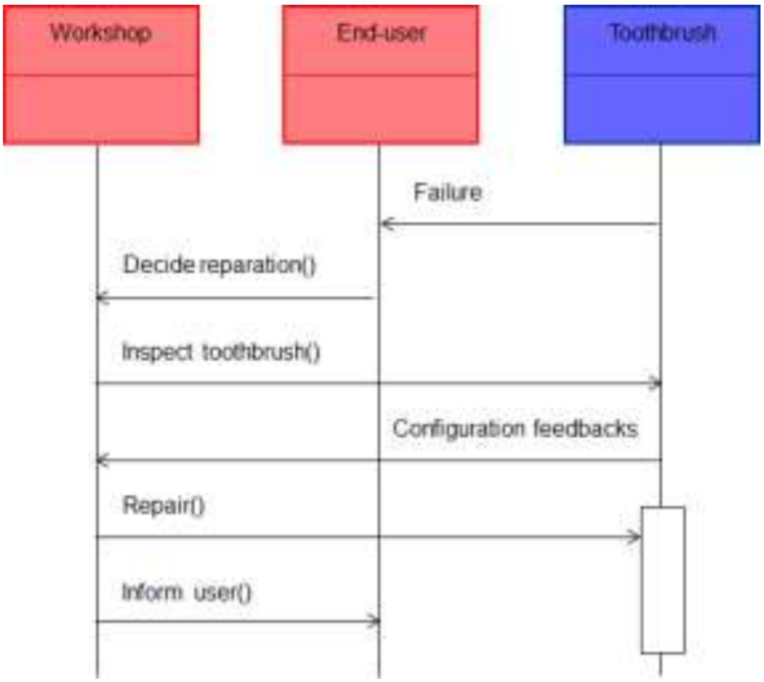


Figure 46. Exemple de Diagramme de scénario opérationnel (extrait de [CESAM17])

Pour recenser définir ces scénarios en Papyrus-SysML, nous allons réaliser des diagrammes de séquence système dans lequel le participant principal sera le système (en vue boîte noire donc). Ne serons représentées que ces interactions avec les parties prenantes.

1. Toujours dans le *package* `SHS_Missions`, repérez le cas d'utilisation dont vous voulez préciser le scénario
2. Créez une intéraction : **New_UML_for_SysML_1.6 › Interaction**
3. Créez le scénario correspondant : **New_Diagram › SysML 1.6 Sequence Diagram**
4. Placez-y les participants en allant chercher le système dans le diagramme de contexte et les acteurs dans les parties prenantes du niveau opérationel.
5. Ajoutez sus forme de message échangés les interactions entre les participants du diagramme de séquence.

Nous obtenons des diagrammes de séquence système comme illustré à la Figure 47.

Commentaire

Figure à changer pour du Papyrus-SysML!

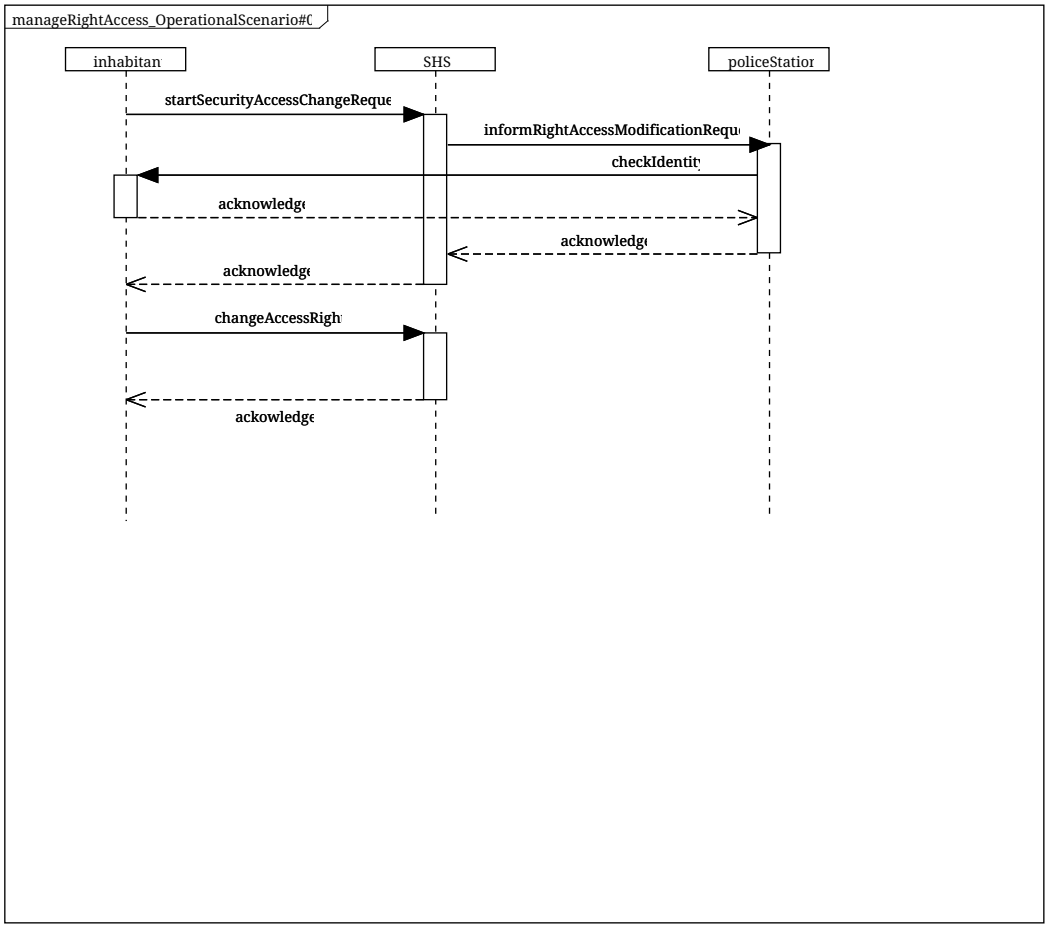


Figure 47. Exemple de Diagramme de scénario opérationnel (extrait de [CESAM17])

Flux

	Exigences	États	Structures	Interactions	Flux	Transversalités
Organisation						

	Exigences	États	Structures	Interactions	Flux	Transversalités
Environnement						
Vision Opérationelle					You are here!	
Vision Fonctionnelle						
Vision Organique						

Le diagramme des flux operationnels (cf. Figure 48) vise à répertorier tous les flux en lien avec le système en détaillant :

- leur relations logiques (fournir, mesurer, requérir, ...)
- leur relations d'abstraction

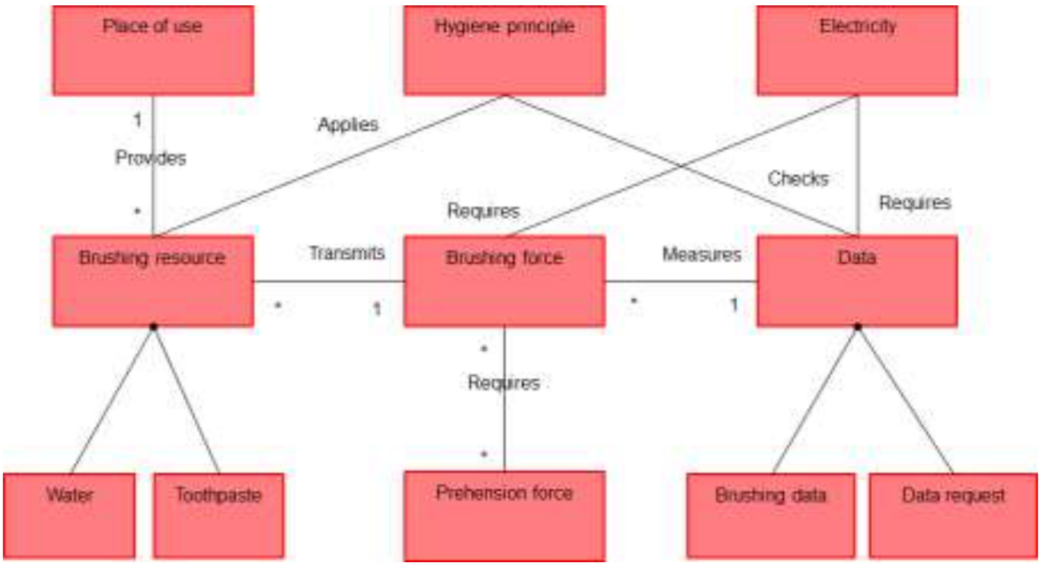


Figure 48. Exemple de Diagramme de flux opérationnels (extrait de [CESAM17])

CESAM insiste sur l'importance de cette phase pour l'élaboration du dictionnaire du système, c'est-à-dire la liste des objets qui seront manipulés par le système. Cela peut être en effet primordiale dans un contexte multi-domaines pour se coordonner sur les termes employés.

Pour réaliser ce dictionnaire en Papyrus-SysML, plutôt que de réaliser un diagramme de blocs avec des associations entre les blocs, nous avons choisis de réaliser une matrice de dépendance entre flux qui indique les relations précises (*from* et *to*). Pour cela :

1. bla
2. bli

Nous obtenons une table illustrée à la Figure 49.

Commentaire

Figure à changer pour du Papyrus-SysML!

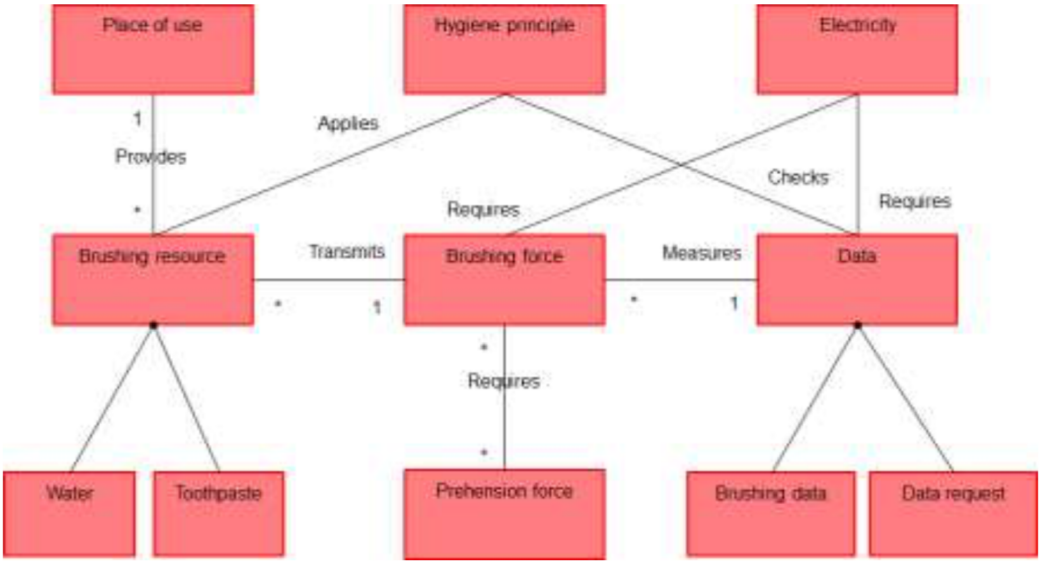


Figure 49. Exemple de Diagramme de flux opérationnels (extrait de [CESAM17])

8.4.3. En résumé

8.4.4. Questions de révision

1. Quelles sont les différences entre **besoins** et **exigences** ?
2. En quoi les cas d'utilisation sont-ils complémentaires des exigences?
3. Quelle est la différence entre un *package* de type ***model*** et un *package* de type ***package***?

8.5. Vision Fonctionnelle

Commentaire

Vision Fonctionnelle.

8.5.1. Principes

Blabla sur cette partie en générale.

	Exigences	États	Structures	Interactions	Flux	Transversalités
Organisation						

	Exigences	États	Structures	Interactions	Flux	Transversalités
Environnement						
Vision Opérationelle						
Vision Fonctionnelle			You are here!			
Vision Organique						

Concepts définis dans cette section

Nous aborderons les concepts du niveau initial (*Model User*) de la certification *Certified Systems Modeling Professional*™: ...

Commentaire

Lister les concepts abordés!

8.5.2. Mise en œuvre en SysML

Exigences

	Exigences	États	Structures	Interactions	Flux	Transversalités
Organisation						
Environnement						
Vision Opérationelle						
Vision Fonctionnelle	You are here!					
Vision Organique						

Commentaire

Figure à changer pour du Papyrus-SysML!

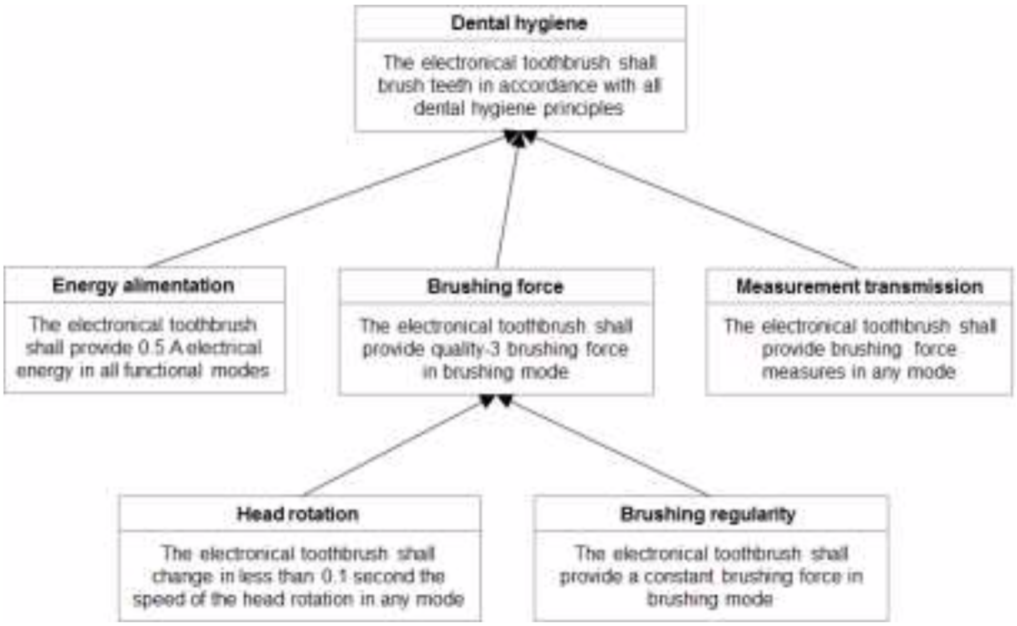


Figure 50. Exemple de Diagramme des exigences fonctionnelles (extrait de [CESAM17])

États

	Exigences	États	Structures	Interactions	Flux	Transversalités
Organisation						
Environnement						
Vision Opérationelle						
Vision Fonctionnelle		You are here!				
Vision Organique						

Commentaire

Figure à changer pour du Papyrus-SysML!

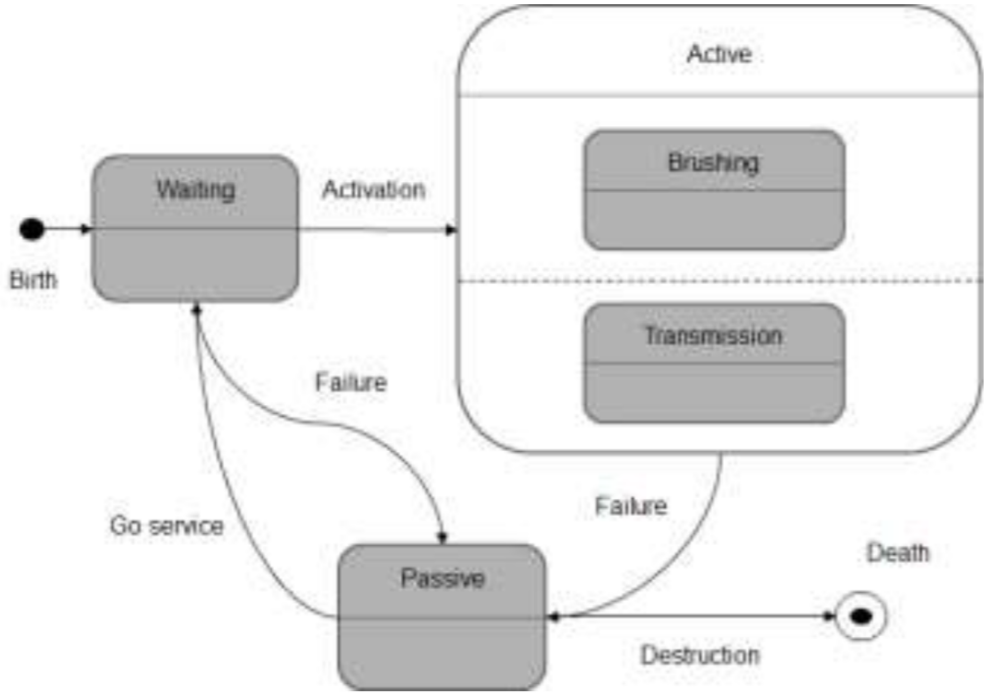


Figure 51. Exemple de Diagramme des Modes Fonctionnels (extrait de [CESAM17])

Structures

	Exigences	États	Structures	Interactions	Flux	Transversalités
Organisation						
Environnement						
Vision Opérationelle						
Vision Fonctionnelle			You are here!			
Vision Organique						

Commentaire

Figure à changer pour du Papyrus-SysML!

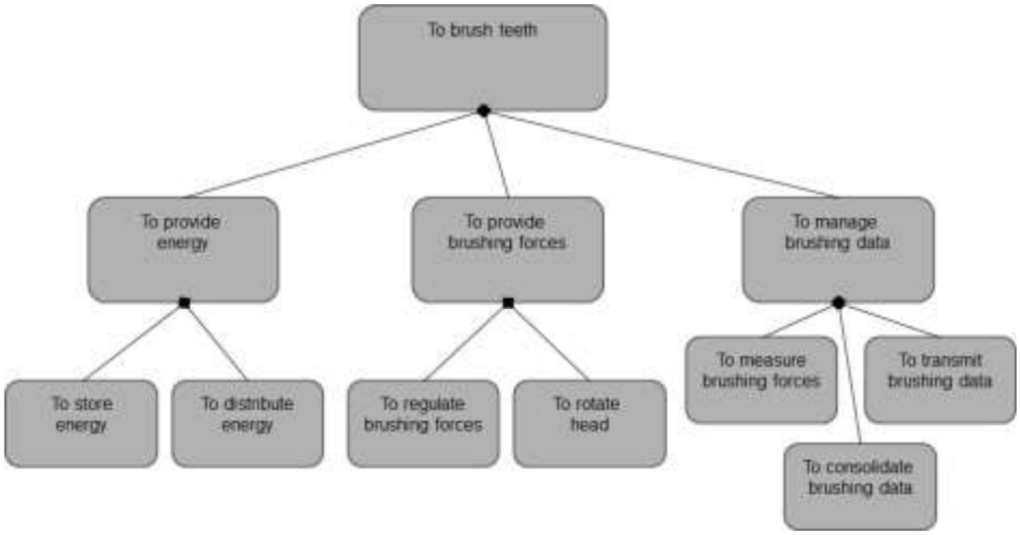


Figure 52. Exemple de Diagramme de Décomposition Fonctionnelle (extrait de [CESAM17])

Commentaire

Figure à changer pour du Papyrus-SysML!

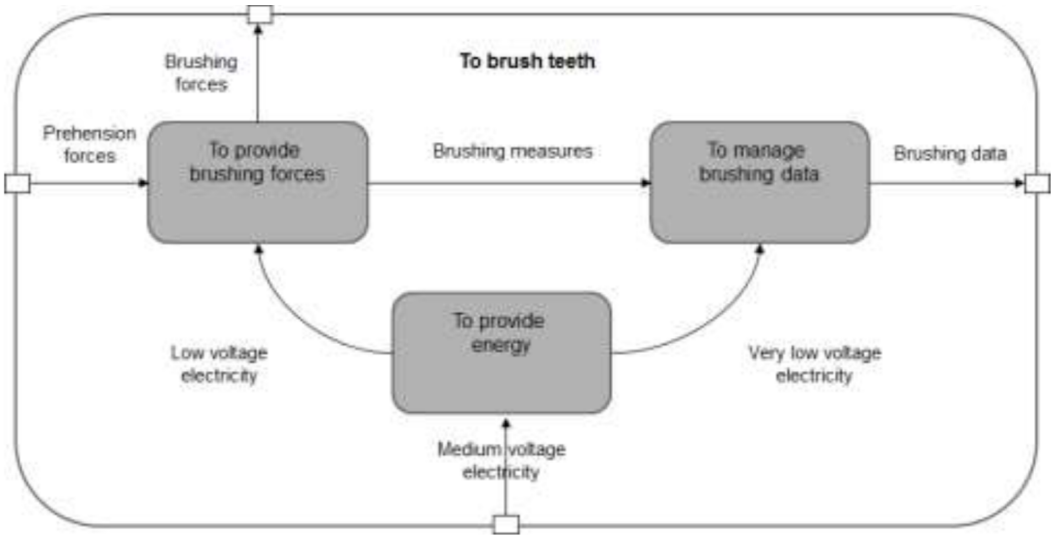


Figure 53. Exemple de Diagramme d'Interaction Fonctionnelle (extrait de [CESAM17])

Interactions

	Exigences	États	Structures	Interactions	Flux	Transversalités
Organisation						

	Exigences	États	Structures	Interactions	Flux	Transversalités
Environnement						
Vision Opérationelle						
Vision Fonctionnelle				You are here!		
Vision Organique						

Commentaire

Figure à changer pour du Papyrus-SysML!

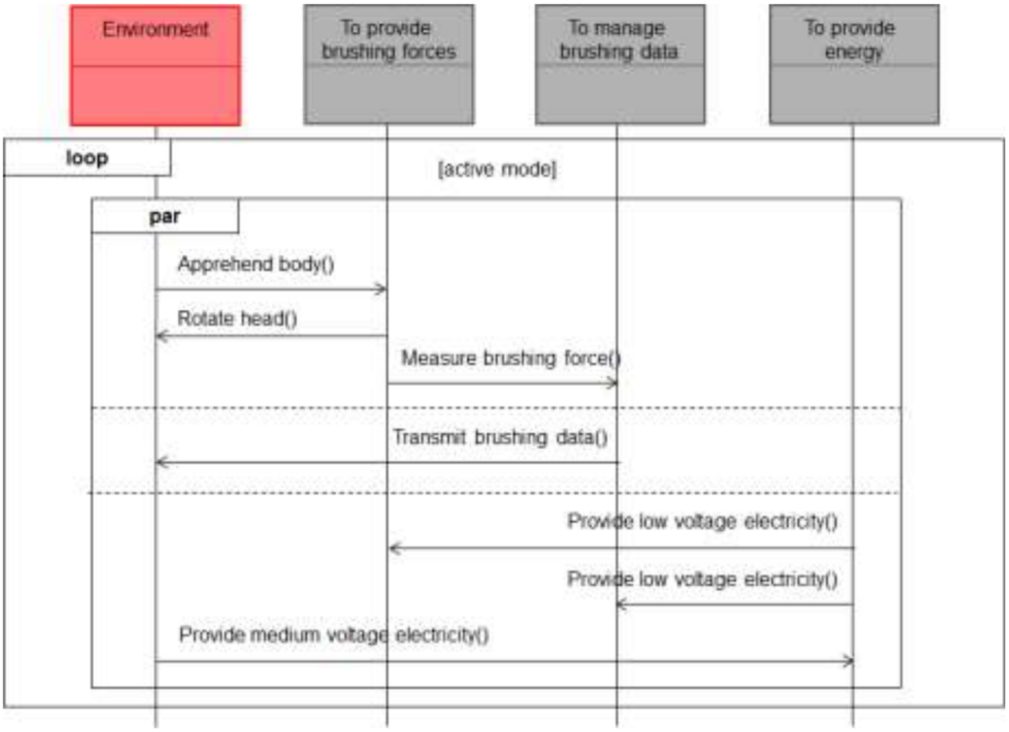


Figure 54. Exemple de Diagramme de Scénario Fonctionnel (extrait de [CESAM17])

Flux

	Exigences	États	Structures	Interactions	Flux	Transversalités
Organisation						

	Exigences	États	Structures	Interactions	Flux	Transversalités
Environnement						
Vision Opérationelle						
Vision Fonctionnelle					You are here!	
Vision Organique						

Commentaire

Figure à changer pour du Papyrus-SysML!

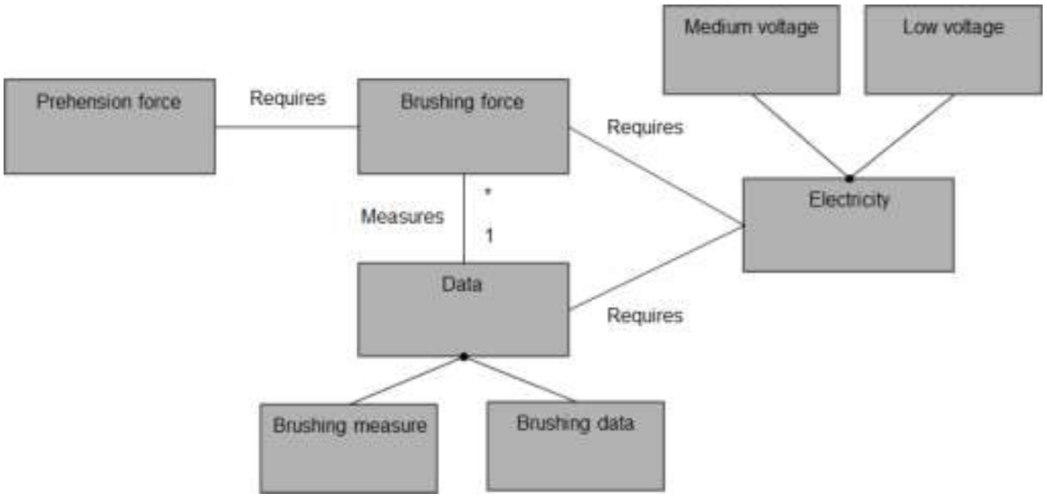


Figure 55. Exemple de Diagramme de Flux Fonctionnels (extrait de [CESAM17])

8.5.3. En résumé

8.5.4. Questions de révision

1. Quelles sont les différences entre ...

8.6. Vision Organique

Commentaire

Vision Organique.

8.6.1. Principes

Blabla sur cette partie en générale.

	Exigences	États	Structures	Interactions	Flux	Transversalités
Organisation						
Environnement						
Vision Opérationelle						
Vision Fonctionnelle						
Vision Organique			You are here!			

Concepts définis dans cette section

Nous aborderons les concepts du niveau initial (*Model User*) de la certification *Certified Systems Modeling Professional™*: ...

Commentaire

Lister les concepts abordés!

8.6.2. Mise en œuvre en SysML

Exigences

	Exigences	États	Structures	Interactions	Flux	Transversalités
Organisation						

	Exigences	États	Structures	Interactions	Flux	Transversalités
Environnement						
Vision Opérationelle						
Vision Fonctionnelle						
Vision Organique	You are here!					

Commentaire

Figure à changer pour du Papyrus-SysML!

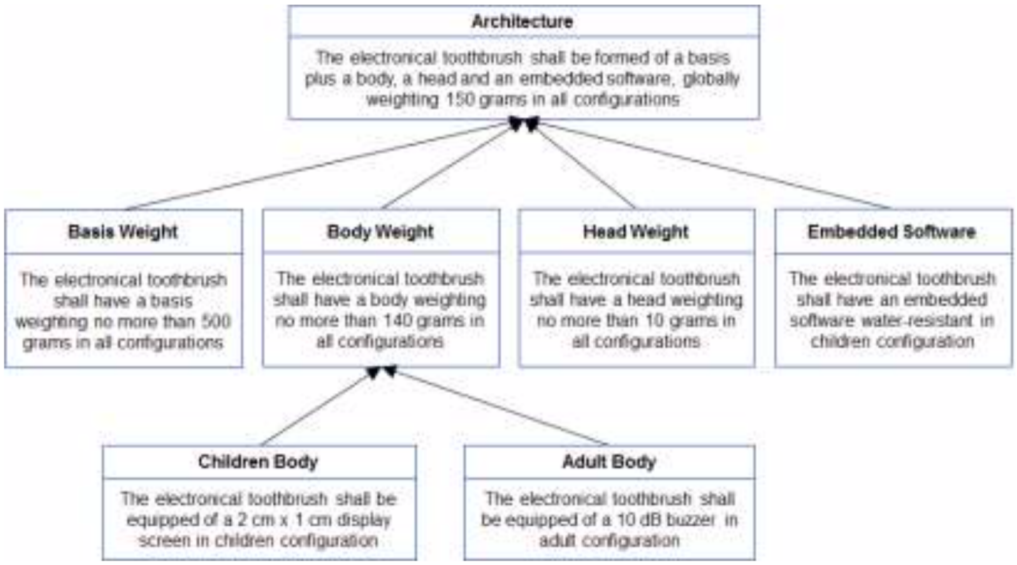


Figure 56. Exemple de Diagramme des Exigences Organiques (extrait de [CESAM17])

États

	Exigences	États	Structures	Interactions	Flux	Transversalités
Organisation						

	Exigences	États	Structures	Interactions	Flux	Transversalités
Environnement						
Vision Opérationelle						
Vision Fonctionnelle						
Vision Organique		You are here!				

Commentaire

Figure à changer pour du Papyrus-SysML!

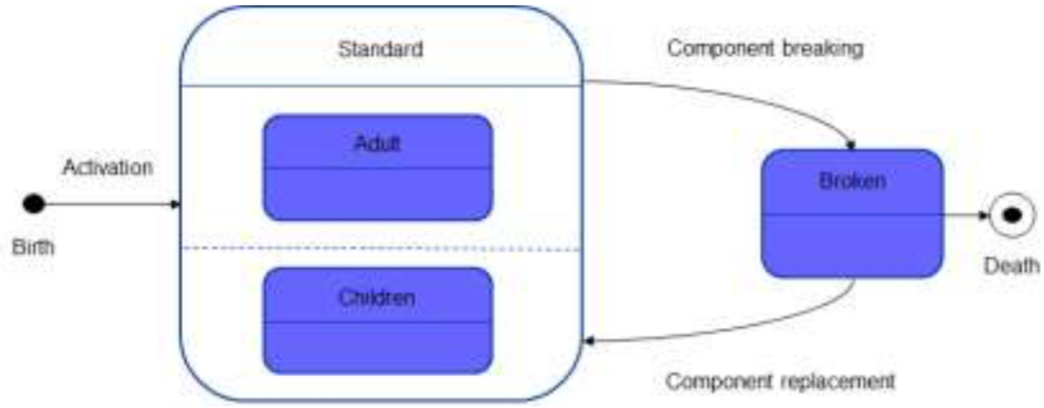


Figure 57. Exemple de Diagramme de Configuration (extrait de [CESAM17])

Structures

	Exigences	États	Structures	Interactions	Flux	Transversalités
Organisation						
Environnement						
Vision Opérationelle						
Vision Fonctionnelle						
Vision Organique			You are here!			

Commentaire

Figures à changer pour du Papyrus-SysML!

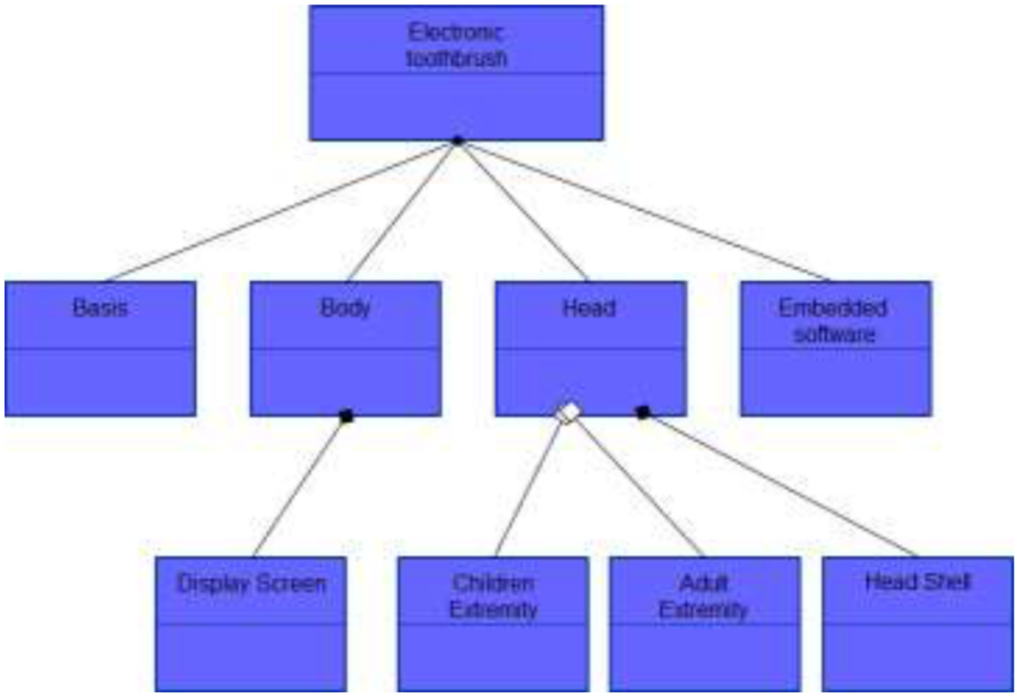


Figure 58. Exemple de Diagramme de Décomposition Organique (extrait de [CESAM17])

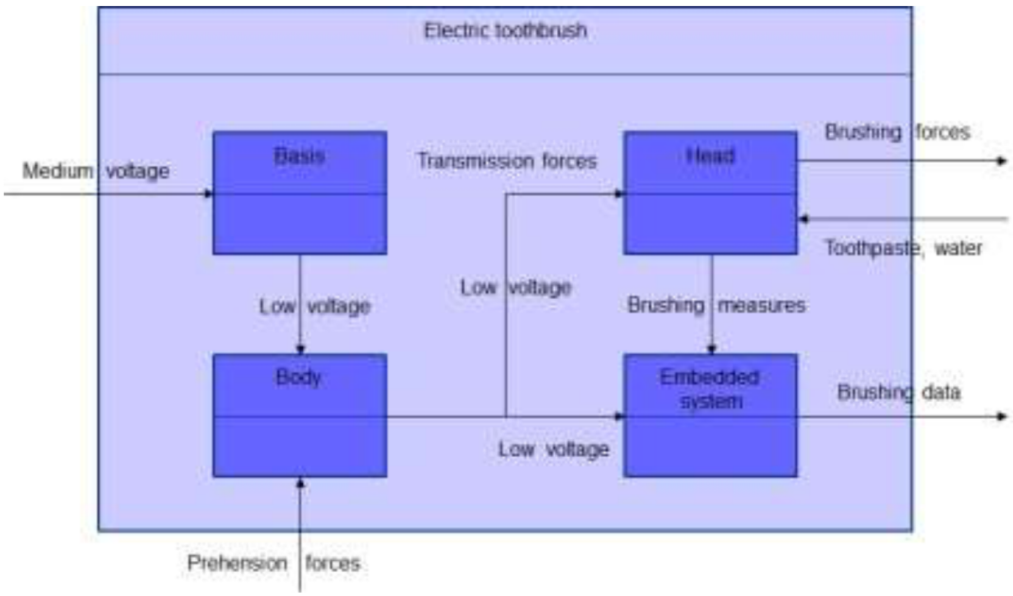


Figure 59. Exemple de Diagramme d'Interaction Organique (extrait de [CESAM17])

Interactions

	Exigences	États	Structures	Interactions	Flux	Transversalités
Organisation						

	Exigences	États	Structures	Interactions	Flux	Transversalités
Environnement						
Vision Opérationelle						
Vision Fonctionnelle						
Vision Organique				You are here!		

Commentaire

Figures à changer pour du Papyrus-SysML!

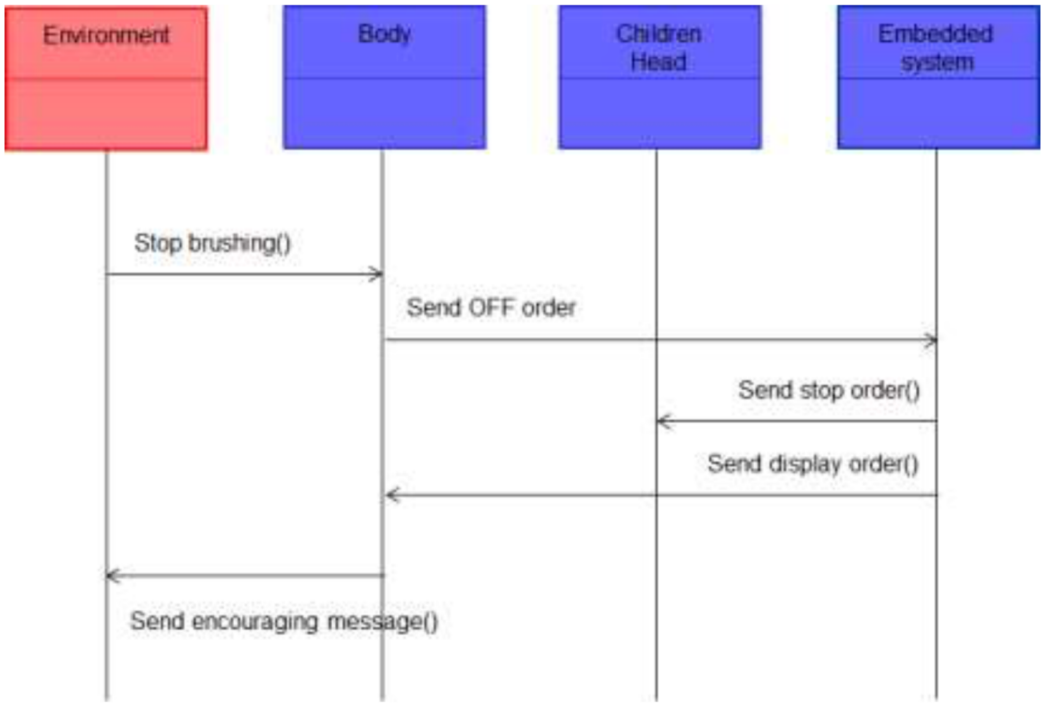


Figure 60. Exemple de Diagramme de Décomposition Organique (extrait de [CESAM17])

Flux

	Exigences	États	Structures	Interactions	Flux	Transversalités
Organisation						

	Exigences	États	Structures	Interactions	Flux	Transversalités
Environnement						
Vision Opérationelle						
Vision Fonctionnelle						
Vision Organique					You are here!	

Commentaire

Figures à changer pour du Papyrus-SysML!

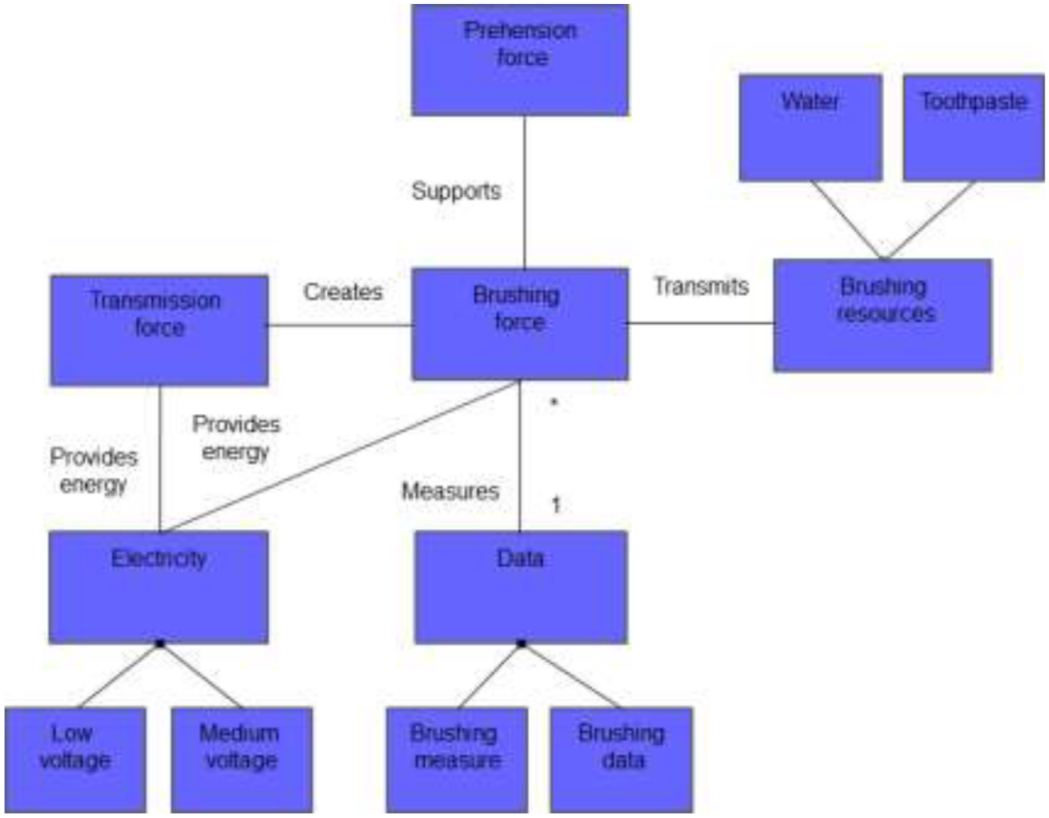


Figure 61. Exemple de Diagramme de Flux Organiques (extrait de [CESAM17])

8.6.3. En résumé

8.6.4. Questions de révision

1. Quelles sont les différences entre

8.7. Pour aller plus loin

Commentaire

Référencer la question du WHO, abordée en Section 8.3 et à aborder dans la partie transverse aussi! Mentionner aussi Papyrus-CESAM.

8.8. En résumé

Nous avons présenté dans ce chapitre une façon organisée et structurée de réaliser la modélisation d'un système en suivant la méthode CESAM, en utilisant la notation SysML®. Nous avons mis cela en pratique en modélisant notre étude de cas en utilisant Papyrus-SysML.

Ce chapitre est un véritable tutoriel concernant à la fois la méthode, la notation, et l'outil. Nous invitons le lecteur à utiliser le matériel d'accompagnement de cet ouvrage (vidéos, tutoriels, modèles à télécharger, etc.) sur le site d'accopagnement : https://bit.ly/sysmlbook.

8.9. Questions de révision

1. Qu'appelle-t'on la "Matrice d'Architecture Systèmes" de CESAM ?
2. Quelles sont les 3 principales "visions" de CESAM ?
3. Qu'est-ce qui les différencie principalement ?
4. Quels sont les principaux diagrammes utilisés dans la mise en œuvre en SysML® de CESAM ?

Chapter 9. Concepts SysML avancés

9.1. Besoins clients et exigences

Commentaire

Introduit SysML Req, les diagrammes/tables des exigences et les diagrammes de cas d'utilisation, les diagrammes de séquences « système » pour décrire les scénarios d'usage associés. Après avoir parlé des concepts SysML pour traiter de ce sujet, dire qu'il peut être nécessaire d'organiser les exigences ⇒ diagramme de paquetage.

9.1.1. Fondements

	Exigences	Structures	Interactions	Transversalités
Organisation	You are here!			
Vision Opérationelle	You are here!			
Vision Fonctionnelle	You are here!			
Vision Organique	You are here!			

Concepts définis dans cette section

Nous aborderons les concepts du niveau initial (*Model User*) de la certification *Certified Systems Modeling Professional*™: requirement, requirement diagram, deriveRqt, verify, satisfy, refine, trace, containment.

Les exigences sont prises en compte à différent niveaux en SysML®. Nous aborderons :

- L'organization des *Requirements*
- Les *Requirements properties*
- Les *Requirements links*
- Les *Requirements Diagrams*
- Les considérations sur la traçabilité
- Annotations des *Requirements*
- Les *Use Case Diagrams*

L'ingénierie des exigences est une discipline à part entière et nous n'abordons ici que les aspects en lien avec la modélisation système. Voir le livre de référence pour plus de détails ([Sommerville1997]) ou le guide de l'AFIS ([REQ2012]).

9.1.2. L'organisation des *Requirements*

Il ne s'agit pas ici de revenir sur les exigences elles-même, mais plutôt de voir comment SysML® permet de les exprimer, de les manipuler et surtout de les lier avec le reste du système.

Représentation de base

Un *Requirement* en SysML® permet de représenter une exigence, généralement référencée dans un document à part (ou dans un logiciel dédié). Une exigence est donc représentée par un bloc particulier (stéréotypé << requirement >>), avec deux attributs obligatoires : *Id* qui représente une référence unique (généralement reprise du document initiale d'où a été importée l'exigence), et *text* qui reprend le texte descriptif de l'exigence.

Définition : Requirements (OMG SysML v1.5, p. 161)

A requirement specifies a capability or condition that must (or should) be satisfied…
A requirement is defined as a stereotype of UML Class…

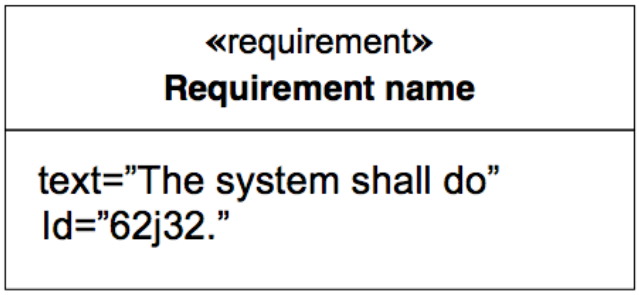


Figure 62. Un exemple de Requirement en SysML (source [SysML], p. 159)

Les **stéréotypes** sont très souvent utilisés en UML® / SysML®. Ils s'appliquent à d'autres éléments (comme des blocs ou des associations) pour en changer la signification par défaut.

Nouveauté de SysML 1.5

La dernière version de SysML® en date du 1er mai 2017 apporte des progrès significatifs en ce qui concerne les exigences.

http://model-based-systems-engineering.com/2017/05/09/whats-new-in-sysml-1-5-requirements-modeling/

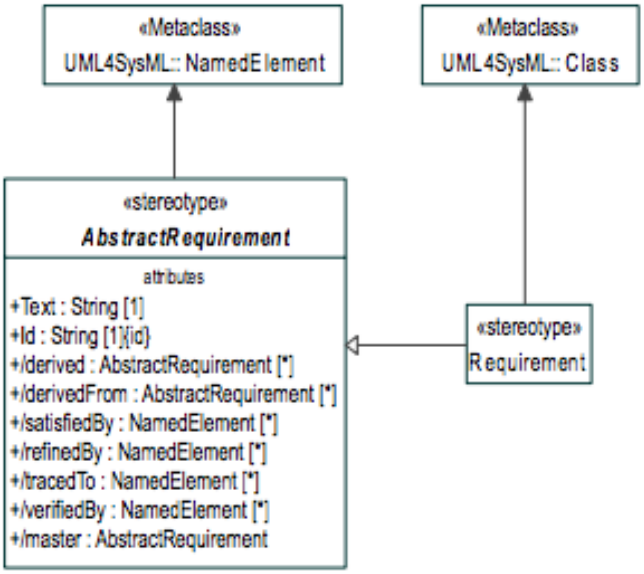


Figure 63. N'importe quel élément de modèle (qui a un nom) peut être une exigence ([SysML], p.168)

Différents types d'organisation

L'ingénierie des exigences aboutit généralement à une liste organisée d'exigences, que ce soit en terme de fonctionnelles/non fonctionnelles, de prioritaires/secondaires, etc. Le principal support de SysML® à cette organisation, outre la possibilité de les annoter (cf. section Stéréotyper les exigences), consiste à utiliser les *packages*.

Plusieurs types d'organisations sont possibles :

- Par niveau d'abstraction
  - Besoins généraux (en lien avec les *use cases* par exemple)
  - Besoins techniques (en lien avec les éléments de conception)
- Par point de vue
  - Besoins principaux (en lien avec les *use cases*)
  - Besoins spécifiques :
    - Fonctionnels
    - Marketing
    - Environnementaux
    - *Business*
    - ...
- etc.

Exemple industriel

Dans l'entrepris CS, les exigences sont organisées en calquant l'arborescence sur les documents textuels que les ingénieurs ont l'habitude d'utiliser. Ceci facilite le travail d'organisation des exigences pour les ingénieurs [Neptune17].

Dans le SmartHomeSystem, nous avons organisé les exigences en paquetages par grands types :

[Outlines of the system requirements model organisation] |

../../SmartHomeUseCase/SmartHomeSystemProject/diagrams/Outlines_of_the_system_requirements_m

odel_organisation.PNG

Figure 64. Notre organisation des exigences pour le cas d'étude

Une autre façon d'organiser les exigences consiste à utiliser la relation de *Containment* (cf. Section 9.1.10 pour plus de détails).

[SecurityRequirementDiagram] |

../../SmartHomeUseCase/SmartHomeSystemProject/diagrams/SecurityRequirementDi

agram.PNG

Figure 65. Composition entre exigences

Tableaux de *Requirements*

Les *requirements* sont habituellement stockés dans des tableaux (feuilles Excel le plus souvent!). Il est donc recommandé par la norme, et possible dans de nombreux outils, de représenter les exigences sous forme tabulaire.

Définition : Requirements Table (OMG SysML v1.5, p. 167)

The tabular format is used to represent the requirements, their properties and relationships…

table [requirement] Performance [Decomposition of Performance Requirement]

id	name	text
2	Performance	The Hybrid SUV shall have the braking, acceleration, and off-road capability of a typical SUV, but have dramatically better fuel economy.
2.1	Braking	The Hybrid SUV shall have the braking capability of a typical SUV.
2.2	FuelEconomy	The Hybrid SUV shall have dramatically better fuel economy than a typical SUV.
2.3	OffRoadCapability	The Hybrid SUV shall have the off-road capability of a typical SUV.
2.4	Acceleration	The Hybrid SUV shall have the acceleration of a typical SUV.

table [requirement] Performance [Tree of Performance Requirements]

id	name	relation	id	name	relation	id	name
2.1	Braking	deriveReqt	d.1	RegenerativeBraking			
2.2	FuelEconomy	deriveReqt	d.1	RegenerativeBraking			
2.2	FuelEconomy	deriveReqt	d.2	Range			
4.2	FuelCapacity	deriveReqt	d.2	Range			
2.3	OffRoadCapability	deriveReqt	d.4	Power	deriveReqt	d.2	PowerSourceManagement
2.4	Acceleration	deriveReqt	d.4	Power	deriveReqt	d.2	PowerSourceManagement
4.1	CargoCapacity	deriveReqt	d.4	Power	deriveReqt	d.2	PowerSourceManagement

Figure 66. Exemples de tableaux des exigences (OMG SysML v1.4, p. 163)

La plupart des outils modernes permettent le passage entre outils classiques de gestion des exigences (comme DOORS™) et outils de modélisation SysML®.

Figure 67. Import Papyrus de tableau des exigences

XXX Parler des imports/exports ReqIf, de l'import/export excel etc.

Papyrus for Requirements

Ce composant spécifique de Papyrus-SysML traite des exigences.

Pour l'installer, procéder comme l'installation de SysML® (cf. Chapitre 7) mais en utilisant l'url suivante : https://hudson.eclipse.org/papyrus/view/Requirements/job/Requirements-Master/lastSuccessfulBuild/artifact/releng/org.eclipse.papyrus.requirements.p2/target/repository/.

9.1.3. Les *Requirements properties*

Il est possible d'indiquer un certain nombre de propriétés sur un *requirement* :

- *priority* (`high`, `low`, ...)
- *source* (`stakeolder`, `law`, `technical`, ...)
- *risk* (`high`, `low`, ...)
- *status* (`proposed`, `aproved`, ...)
- *verification method* (`analysis`, `tests`, ...)

9.1.4. Les *Requirements links*

Les principales relations entre *requirement* sont :

Containment Pour décrire la décomposition d'une exigence en plusieurs sous-exigences (□–). Typiquement dès qu'une exigence est exprimée avec une conjonction "et" ("La voiture doit être rapide et économe.").

Refinement Pour décrire un ajout de précision (`<<refine>>`), comme par exemple une précision.

Derivation Pour indiquer une différence de niveau d'abstraction (`<<deriveReqt>>`), par exemple entre un système et un de ses sous-systèmes.

Lorsqu'une exigence possède plusieurs cas `<<refine>>` qui pointent vers lui, on considère que ces différents cas sont des options possibles de raffinement (cf. [conventions]).

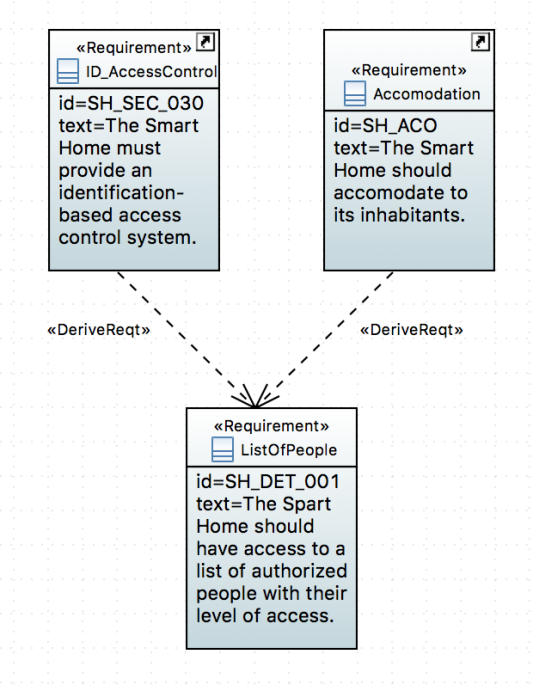


Figure 68. Exemples de relations entre exigences

Il existe ensuite les relations entre les besoins et les autres éléments de modélisation (les *block* principalement) comme `<<satisfy>>` ou `<<verify>>`, mais nous les aborderons dans la partie transverse.

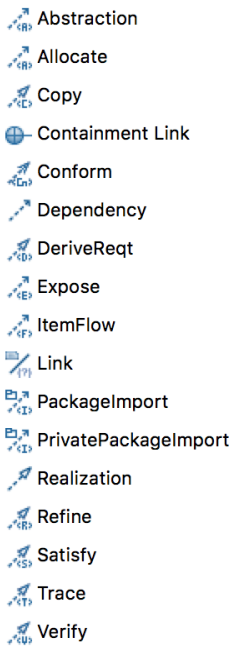


Figure 69. Relations liées au requirements dans Papyrus-SysML

9.1.5. Les *Requirements Diagrams*

Voici un exemple de `req` un peu plus étoffé, extrait de http://www.uml-sysml.org/sysml (voir aussi Figure 71) :

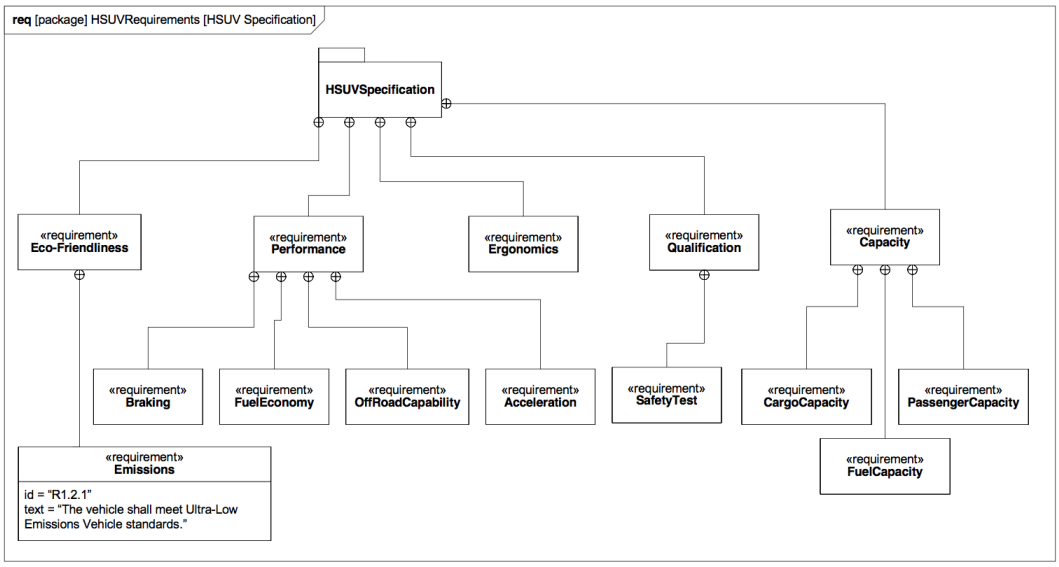


Figure 70. Exemples de composition d'exigences

9.1.6. Stéréotyper les *Requirements*

Tout comme pour n'importe quel bloc, il est possible de stéréotyper les *requirements*. Ceci permet de se définir ses propres priorités et classifications. Quelques exemples de stéréotypes utiles :

- `<<interfaceRequirement>>`, `<<physicalRequirement>>`, ...
- `<<FunctionalRequirement>>`, `<<nonFunctionalRequirement>>`

9.1.7. Annotations des *Requirements*

Il est possible d'annoter les éléments de modélisation en précisant les raisons (*rationale*) ou les éventuels problèmes anticipés (*problem*).

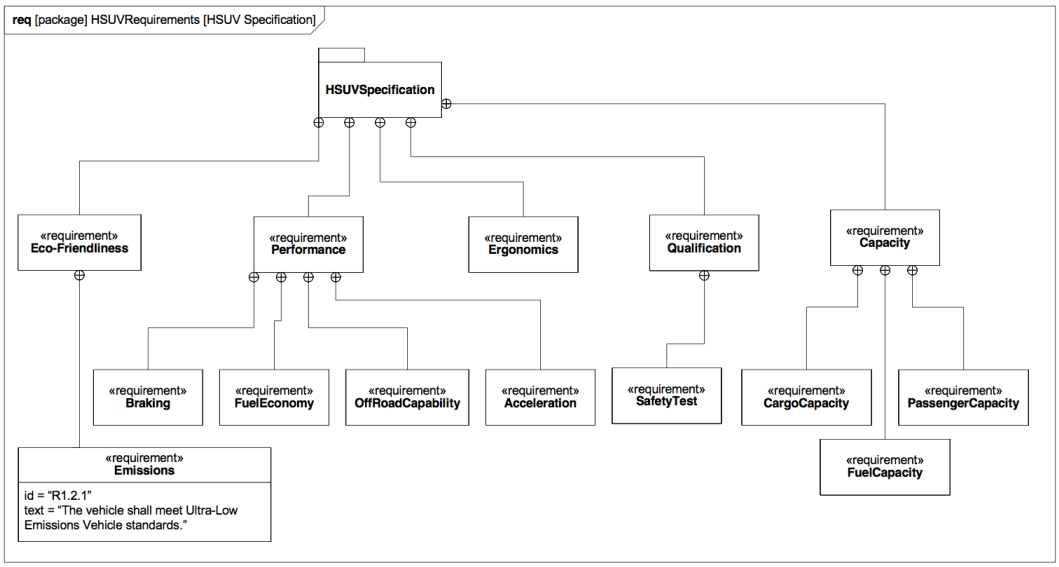


Figure 71. Exemples de rationale et problem (OMG SysML v1.5, p. 227)

9.1.8. Les considérations sur la traçabilité

Une fois que les *requirements* ont été définis et organisés, il est utile de les lier au moins aux *use cases* (en utilisant `<<refine>>` par exemple) et aux éléments structurels (en utilisant `<<satisfy>>` par

exemple), mais ceci sera abordé dans la partie sur les concepts transverses.

Par exemple, en général, chaque *requirement* devrait être relié à au moins un artefact de conception (ne serait-ce qu'un *use case*)[10].

9.1.9. Les *Use Case Diagrams*

Bien que nous traitions les cas d'utilisation dans la partie interface, nous les abordons ici du fait de leur proximité avec les *requirements*.

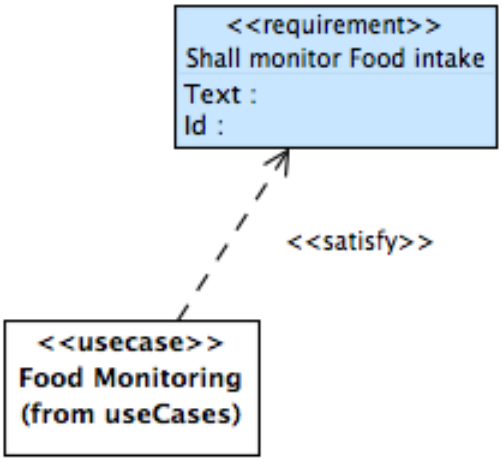


Figure 72. Exemple de lien entre use case et requirements

Ce diagramme est exactement identique à celui d'UML®.

Figure 73. Exemple de diagramme des cas d'utilisation

Un acteur représente un rôle joué par un utilisateur humain. Il faut donc plutôt raisonner sur les rôles que sur les personnes elles-mêmes pour identifier les acteurs.

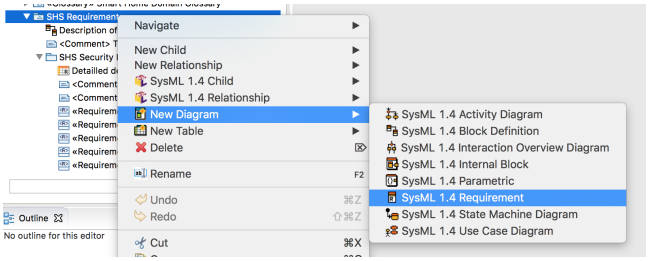
9.1.10. À vous de jouer…

Tables des exigences

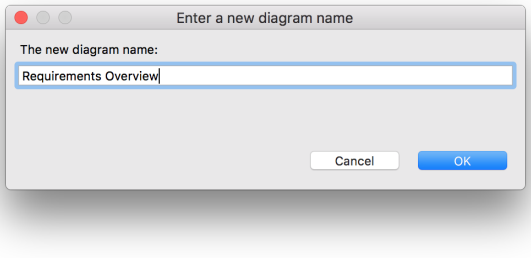
Diagramme des exigences

Réalisons maintenant un diagramme d'exigence.

1. **[ClickDroit]** sur le dossier `SHS Requirements`
2. Sélectionnez **New Diagram › SysML 1.4 Requirement**



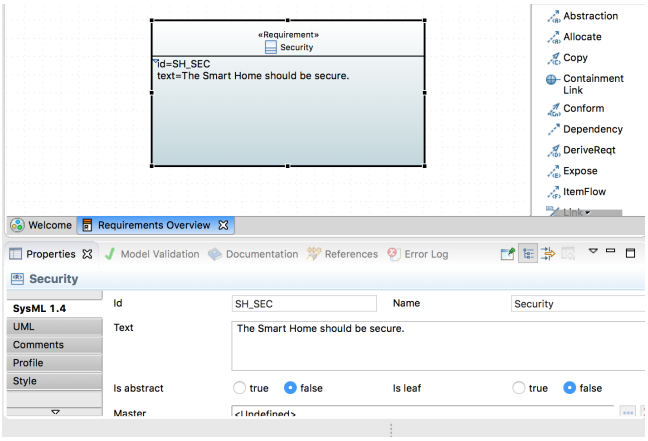
3. Donnez un nom à votre diagramme



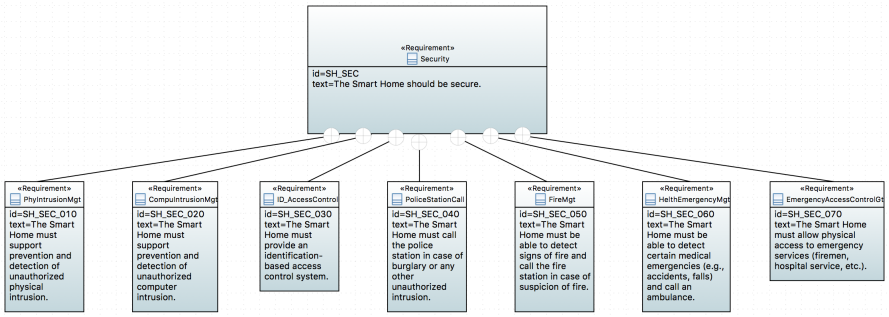
4. Sélectionnez un élément `Requirement` dans la palette en haut à droite et lachez-le sur la fenêtre principale correspondant à votre diagramme

Notez le petit `+` vert qui indique que vous êtes autorisé à lacher l'élément de la Palette à cet endroit du diagramme.

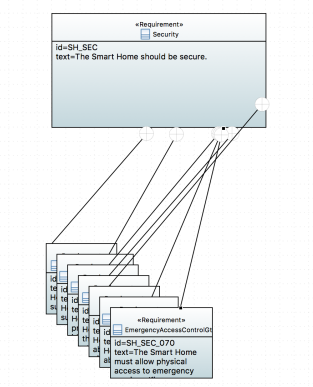
5. Renseignez les `Id` et `Text` à minima.



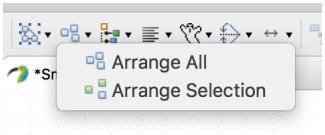
6. Sélectionnez toutes les exigences du *package* `SHS Security Requirements` et lachez-les sur votre diagramme, puis reliez les à votre première exigence par des liens de *Containment*.



Nous n'avons pas obtenu ce diagramme du premier coup. Voici la version initiale :



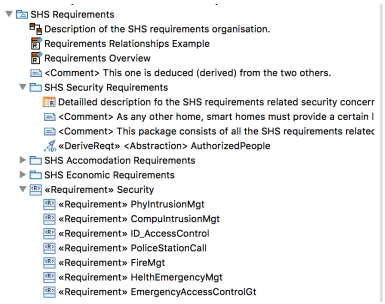
Nous avons sélectionner tous les élements à arranger puis utilisé les icônes dédiées dans la *Toolbar*.



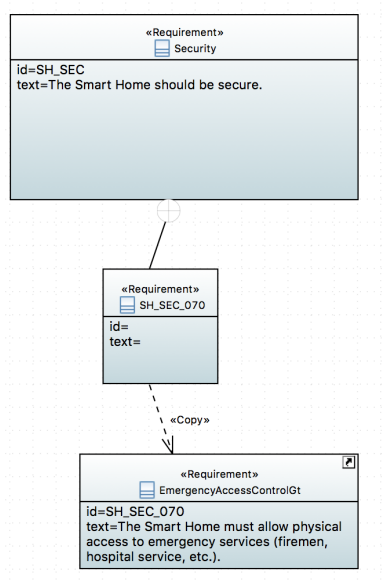
À ce stade il nous faut faire plusieurs observations :

- Notre exigence `Secutity`, se retrouve au même niveau que le diagramme lui-même (c'est-à-dire dans le modèle `SHS Requirements`).

- Les exigences qui composent notre exigence `Security` ont "disparues" de leur *package* initial et sont maintenant "dans" notre exigence.



On voit que la notion de *Containment* n'est pas juste une métaphore. Cela peut poser des problèmes d'organisation. C'est pour cela que nous vous conseillons plutôt de réaliser vos modèles à partir des artefacts. On pourra solutionner ce problème en utilisant le lien de << copy>> fait pour cela.



9.1.11. En résumé

Les exigences sont très importantes en ingénierie système, plus en tout cas qu'en ingénierie logiciel, du fait de la multiplication des sous-systèmes et donc des intermédiaires (fournisseurs, sous-traitants, etc.) avec qui les aspects contractuels seront souvent basés sur ces exigences. Il n'est donc pas étonnant qu'un diagramme et des mécanismes dédiés aient été prévus en SysML®.

Table 10. Déclinaison des Exigences

	Exigences	Structures	Interactions	Transversalités
Organisation	, `<<deriveReqt>>`			
Vision Opérationelle	`<<satisfy>>`, `<<refine>>`	`<<satisfy>>` entre reqs et UC	`<<refine>>`	
Vision Fonctionnelle	`<<allocate>>`			

	Exigences	Structures	Interactions	Transversalités
Vision Organique	`<<satisfy>>`, `<<verify>>`			

En terme de démarche, il est classique d'avoir de nombreux aller-retour entre la modélisation des exigences et la modélisation du système lui-même (cf. Figure 74).

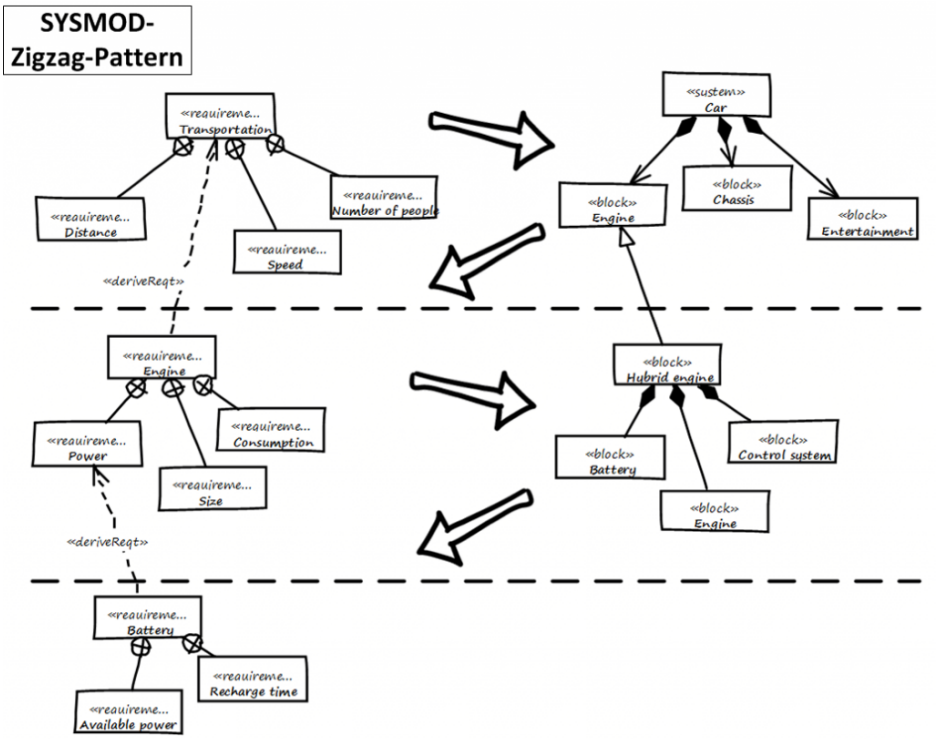


Figure 74. Exemple de démarche (SYSMOD Zigzag pattern)

9.1.12. Questions de révision

1. Quelles sont les différences entre **besoins** et **exigences** ?
2. En quoi les cas d'utilisation sont-ils complémentaires des exigences?
3. Quelle est la différence entre un *package* de type ***model*** et un *package* de type ***package***?

9.2. Usages et interfaces

Commentaire

Diagramme de contexte, BDD/IBD de haut

Avant de développer un système il faut définir le système ⇒ IBD

9.2.1. Diagramme de contexte

- Soit par IBD pour insister sur le système clos (tout ce qui est à l'extérieur n'est pas pris en compte). ⇒ SystemClos
- Soit par BDD ⇒ Context

9.2.2. *Internal Block Diagrams*

Un `ibd` décrit la structure interne d'un bloc sous forme de :

parts

Les parties qui constituent le système (ses sous-systèmes)

ports

Elément d'interaction avec un bloc

connecteurs

Liens entre ports

Parts

Les parties sont représentés par les éléments au bout d'une composition dans un `bdd` . Elles sont créés à la création du bloc qui les contient et sont détruites avec lui s'il est détruit (dépendance de vie).

Il ne s'agit pas de redessiner le BDD. Les *parts* sont des instances et non des classes (au sens objet).

On représente les *parts* comme des bloc en traits pleins et les *references* comme des blocs en trait pointillés.

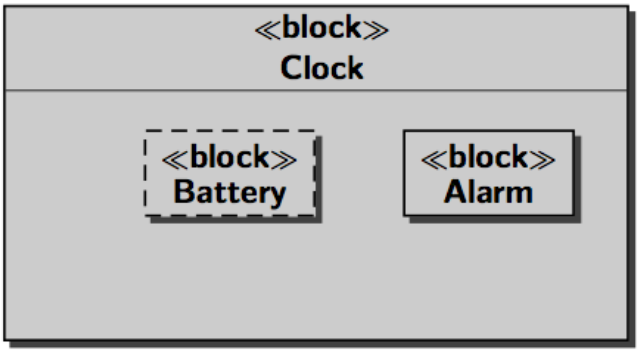


Figure 75. Exemple de Parts

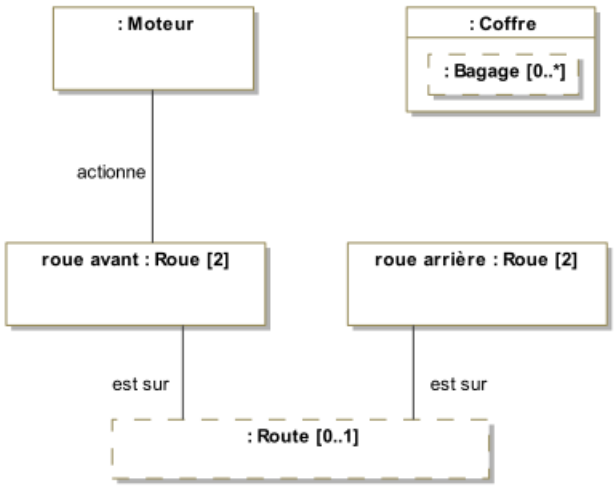


Figure 76. Autre exemple de Parts

Ports (SysML 1.2)

La dernière version de la spécification SysML® 1.6 préconise l'abandon des ports tels que définis dans la version 1.2. Nous présentons les nouvelles notions dans la section qui suit.

Néanmoins, de par l'importance des exemples qui utilisent les notions habituelles de ports, et vu que tous les outils ne supportent pas encore les nouveaux ports, nous indiquons ici leur définition et recommandons pour l'instant de les utiliser.

Les ports :

- préservent l'encapsulation du bloc
- matérialise le fait que les interactions avec l'extérieur (via un port) sont transmise à une partie (via un connecteur)
- les ports connectés doivent correspondre (*kind*, *type*, *direction*, etc.)

Les ports définissent les points d'interaction offerts (`«provided»`) et requis (`«required»`) entre les blocs. Les connecteurs peuvent traverser les "frontières" sans exiger de ports à chaque hiérarchie.

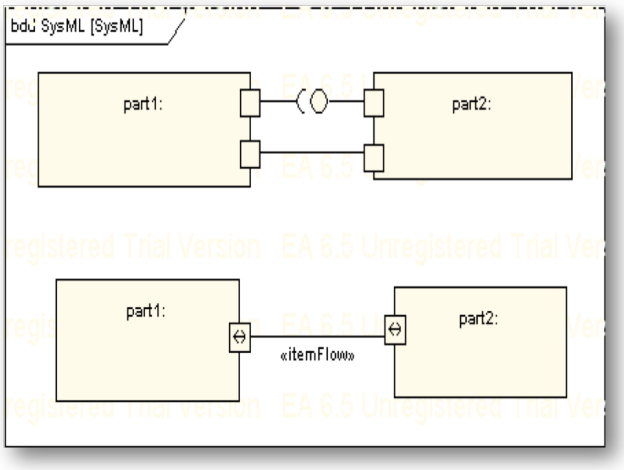


Figure 77. Exemples de flots

Définition : Ports (OMG SysML v1.5, p. 75)

Ports are points at which external entities can connect to and interact with a block in different or more limited ways than connecting directly to the block itself.

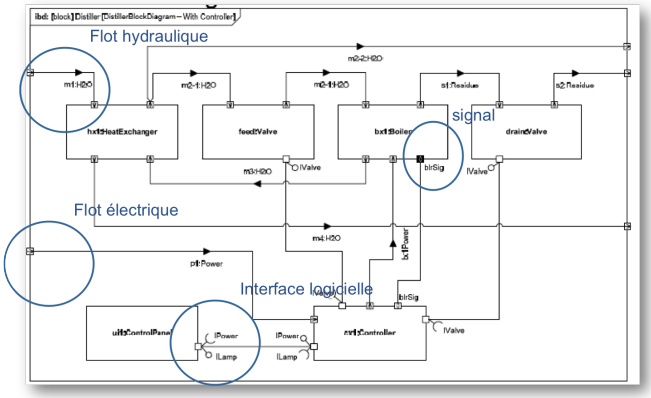


Figure 78. Exemples de flots multi-physique entre ports

Les ports peuvent être de nature classique (comme en UML®) et représenter la fourniture ou le besoin de services. On parle alors de **standard flows**.

Ils peuvent aussi être de nature "flux physique", on parle de **flow ports**.

Les `Flux` peuvent être :

- atomiques (un seul flux),
- composites (agrégation de flux de natures différentes).

Un *flow port* atomique ne spécifie qu'un seul type de flux en entrée ou en sortie (ou les deux), la direction étant simplement indiquée par une flèche à l'intérieur du carré représentant le port. Il peut être typé par un bloc ou un *Value Type* représentant le type d'élément pouvant circuler en entrée ou en sortie du port.

Ports (SysML 1.3)

La version 1.3 de la spécification SysML® introduit les concepts de :

proxy port

Ils doivent remplacer les ports `1.2` (ports de flots et ports standards) en en reprenant les caractéristiques et en ajoutant la possibilité d'imbrication et de spécification renforcée.

full port

En fait il s'agit du même concept qu'une partie qui serait exposée à l'extérieur.

Pour une discussion sur les différences entre les deux ports : http://model-based-systems-engineering.com/2013/09/23/sysml-full-ports-versus-proxy-ports/

Ports (SysML 1.4)

La version SysML® 1.6…

Commentaire

Check if this is still true in 1.4

9.2.3. Diagramme des Uses Cases

Nous aborderons les concepts du niveau initial (*Model User*) de la certification *Certified Systems Modeling Professional*™: use case, actor, subject, association, include, extend, et generalization.

9.2.4. En résumé

…

Table 11. Organisation

	Exigences	États	Structures	Interactions	Flux	Transversalités
Organisation						

	Exigences	États	Structures	Interactions	Flux	Transversalités
Environnement						
Vision Opérationelle						
Vision Fonctionnelle						
Vision Organique						

9.2.5. Questions de révision

1. ...

9.3. Structure et contraintes

Commentaire

IBD, BDD et Parametric. Parler de l'exemple du diagramme de contexte pour illustrer la différence de vue IBD/BDD.

9.3.1. Fondements

	Exigences	Structures	Interactions	Transversalités
Organisation	You are here!	You are here!	You are here!	You are here!
Vision Opérationelle		You are here!		
Vision Fonctionnelle		You are here!		
Vision Organique		You are here!		

Concepts définis dans cette section

Nous aborderons les concepts du niveau initial (*Model User*) de la certification *Certified Systems Modeling Professional*™: Package Diagram, ownership, namespace, containment, dependency, view, viewpoint, Block definition, Block usage, valuetype (with units), value properties, parts, references, operations, Block Definition Diagram, compartments, specialization, associations (including composite[11]), multiplicities, Internal Block Diagram, enclosing block, flow ports and standard ports, connectors and item flows, representation of parts, constraint blocks, Parametric Diagram, constraint properties, constraint parameters, constraint expressions

On abordera :

- l'organisation du système et des modèles
- les *Block Definition Diagrams*
- les *Internal Block Diagrams*
- les *Parametric Diagrams* (pour les contraintes physiques)
- les *Sequence Diagrams* (diagramme de séquences système)

9.3.2. Organisation du système et des modèles

En terme d'organisation, le mécanisme clef est celui de *package*. Celui-ci va permettre d'organiser les modèles, pas le système lui-même. Nous aborderons plus en détail cette organisation en étudiant le diagramme de *packages*.

Pour l'organisation du système, on trouve le plus souvent :

- un diagramme décrivant le contexte (le système dans son environnement), décrit dans un *block definition diagram* (cf. Figure 79)
- un diagramme décrivant les éléments internes principaux du système, décrit dans un *internal block diagram*

9.3.3. *Block Definition Diagrams*

Principes de base

Un `bdd` peut représenter :

- un *package*
- un bloc
- un bloc de contrainte (*constraint block*)

Un diagramme de bloc décrit les relations entre les blocs (compositions, généralisations, ...). Ce diagramme utilise les mêmes éléments que le diagramme de classe UML®.

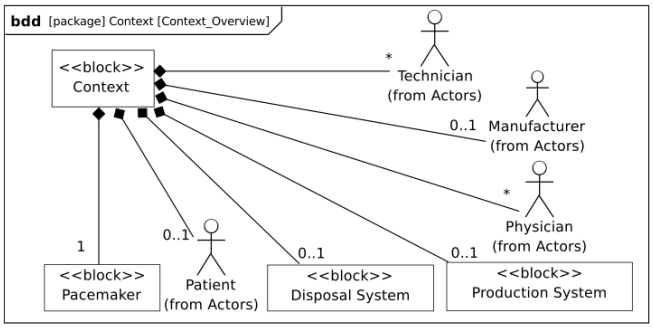


Figure 79. bdd du système dans son environnement

Un bloc est constitué d'un certain nombre de compartiments (*Compartments*) :

Properties

Equivalent UML® des propriétés (e.g., attributs).

Operations

Les méthodes supportées par les instances du bloc.

Constraints

Les contraintes (cf. Figure 80)

Allocations

Les allocations (cf. Chapitre 10)

Requirements

Les exigences liées à ce bloc.

User defined

On peut définir ses propres compartiments.

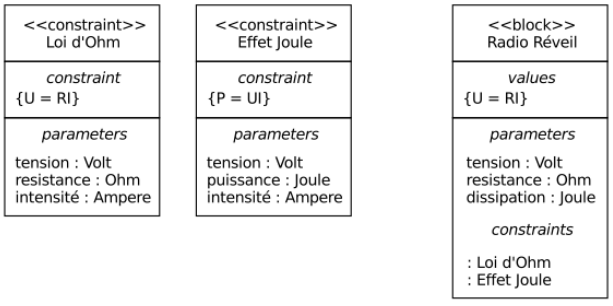


Figure 80. Exemple de définition de contraintes

9.3.4. Diagramme paramétrique (par)

Afin de capturer de manière précise les contraintes entre valeurs, ou encore les liens entre les sorties et les entrées d'un bloc, SysML® utilise trois concepts clefs :

- *Constraints* (un type de bloc)
- *Parametric diagram* (un type d'ibd)
- *Value binding*

Contraintes

C'est un bloc particulier :

- avec un stéréotype <<constraint>> (au lieu de bloc)
- des paramètres en guise d'attributs
- des relations liant (contraignant) ces paramètres

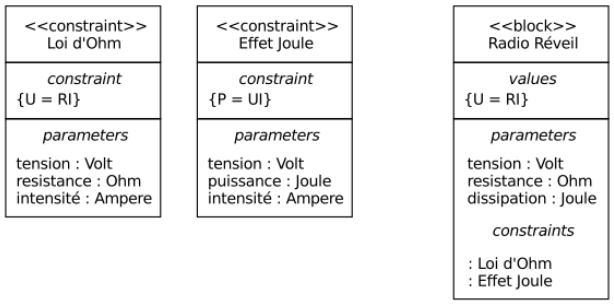


Figure 81. Exemple de contraintes

Définition : ConstraintBlock (OMG SysML v1.5, p. 105)

A constraint block is a block that packages the statement of a constraint so it may be applied in a reusable way to constrain properties of other blocks.

Diagramme paramétrique

C'est une forme particulière d'*Internal Block Definition* (cf. Section 9.2.2). On y retrouve les contraintes, vues à l'instant, mais cette fois-ci on a la représentation graphique des liens entre les données.

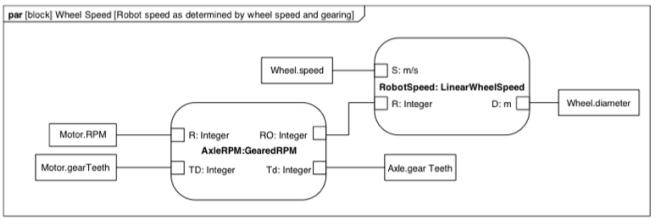


Figure 82. Exemple de diagramme paramétrique

Il est regrettable que ce diagramme soit le moins utilisé (cf. [OMG2009]).

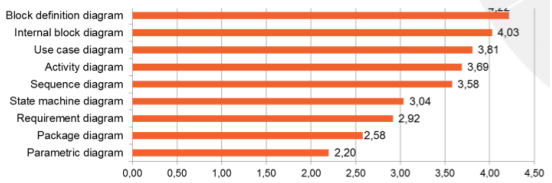


Figure 83. Diagrammes les plus utilisés (extrait de [OMG2009])

Certaines approches (cf. [MeDICIS]) utilisent des feuilles excel pour traduire les diagrammes paramétriques et contrôler l'impact des changements de valeurs de tel ou tel paramètre.

Value Binding

Une fois les contraintes exprimées, il faut lier les paramètres (formels) à des valeurs (paramètre réel). C'est l'objet des *Value Binding*.

Pour assigner des valeurs spécifiques, on utilise des *Block Configurations*;

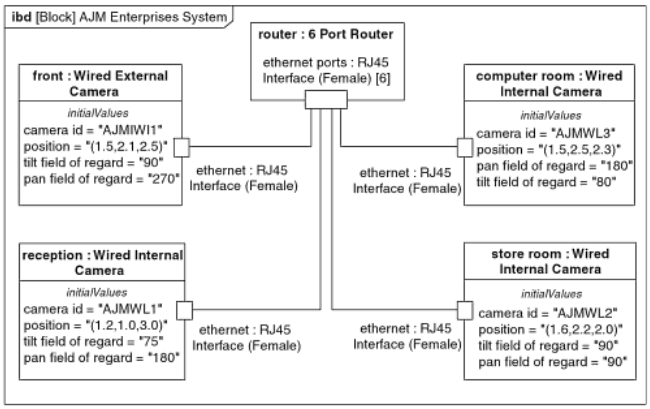


Figure 84. Exemple de bloc de configuration

9.3.5. En résumé

En résumé, il existe plusieurs diagrammes permettant d'exprimer la structure du système à concevoir. En fonction du niveau de détail nécessaire on peut voir les sous-systèmes comme des boîtes noires (des blocs) ou comme des boîtes blanches (grâce à l'`ibd` correspondant).

Table 12. Place des aspects structurels

	Exigences	Structures	Interactions	Transversalités
Organisation		`package`		
Vision Opérationelle		`bdd` `par`		
Vision Fonctionnelle		`bdd` `par` `ibd` `dss`		
Vision Organique		`bdd` `par` `ibd` `dss`		

9.3.6. Questions de révision

1. Quelles sont les différences entre une association dirigée (→), une composition (losange noir) et l'agrégation (losange blanc) ?
2. Puisqu'un `bdd` me donne souvent la liste des sous-systèmes (liens de composition), pourquoi ai-je besoin d'un `ibd` ?

9.4. Comportement local

Commentaire

(au sens atomique , local, block) Diagramme de machine à états (aspects contrôle) et diagramme d'activité (description algorithme ?) Utilisation OpaqueBehavior ?

9.4.1. Fondements

Table 13. Place des comportements

	Exigences	Structures	Interactions	Transversalités
Organisation			You are here!	
Vision Opérationelle			You are here!	
Vision Fonctionnelle			You are here!	
Vision Organique			You are here!	

9.4.2. Diagramme d'états

Machines à état

SysML® a repris le concept, déjà connu en UML®, de machine à états (*State Machines*). Ce diagramme représente les différents **états** possibles d'un élément particulier (généralement un bloc), et comment ce bloc réagit à des événements en fonction de son état courant (en passant éventuellement dans un nouvel état).

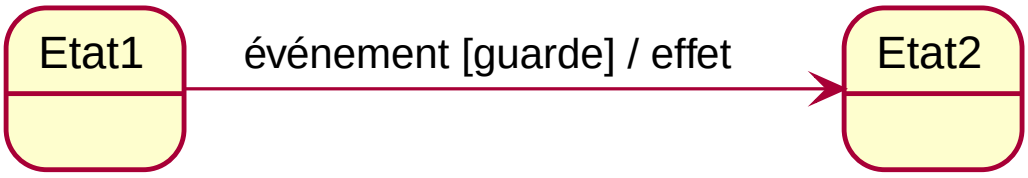
Quand utiliser une machine à états?

Tous les éléments d'un système ne nécessitent pas d'avoir un comportement décrit de manière précise par une machine à état. Quelques situations typiques peuvent faire penser qu'il peut être judicieux de le faire :

- quand un bloc réagit de manière différente à plusieurs événements successifs pourtant identiques
- quand la description du comportement attendu du bloc est lui-même décrit avec des phrases du type "si … et que … alors …".

Transitions

Cette réaction (nommée **transition**) possède un événement déclencheur, une condition (garde), un effet et un état cible, comme illustré dans la [transition], qui se lit : "Si le bloc est dans l'état 'Etat1' et que survient l'événement `événement` et que la condition `guarde` est vraie, alors exécuter `effet` et se plaver dans l'état `Etat2`."



Quand vous clickez sur une transition (ou **[F2]**), l'éditeur qui s'ouvre pour saisir le texte de la transition peut vous assister. En tapant **[Ctrl-Space]**, vous obtenez les concepts disponibles.

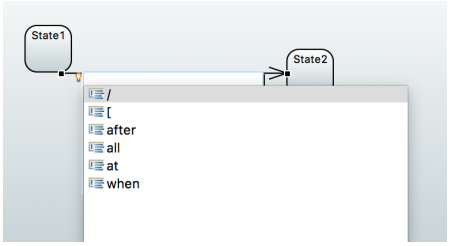


Figure 85. Editeur de transition "intelligent"

1. Notez que les transitions sont toujours **dirigées** (flèche) d'un état de départ vers un état d'arrivée.
2. Notez que les guardes sont entre [].
3. Il peut y avoir autant de transitions sortantes d'un état que l'on veut. La seule attention qu'il faudra porter à cette situation est que toutes les transitions sont "exclusives" les unes par rapport aux autres, sinon on se retrouvera avec une machine dite "**indéterministe**".

Premiers "pseudo-états"

Le diagramme d'états comprend également deux états particuliers (appelés pseudo-états) :

- l'état initial du diagramme d'états correspond à l'état dans lequel on "démarre" la machine a état (à la création d'une instance par exemple);

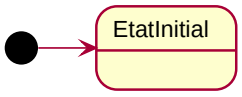


Figure 86. Etat initial

- l'état final du diagramme d'états correspond à la destruction de l'instance.

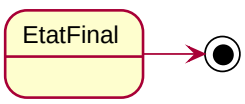


Figure 87. Etat final

Nous verrons d'autres pseudo-états un peu plus tard.

Voici un exemple complet de machine à état :

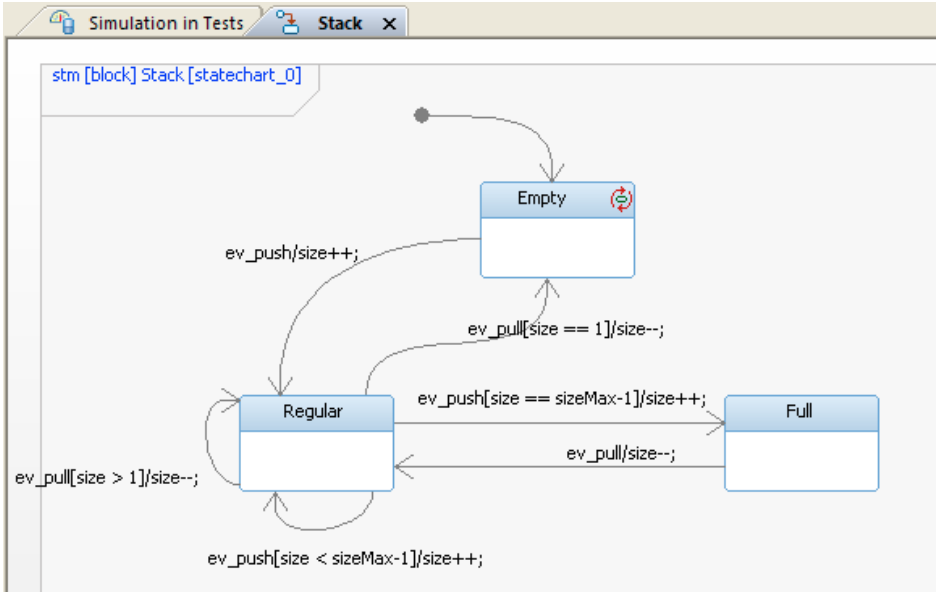


Figure 88. Un exemple de diagramme d'états

Notez que cette machine à état n'a pas d'état final, son exécution est infinie.

Activités d'un état

Un état, outre l'intérêt conceptuel de représenter un situation particulièrement intéressante de l'élément qu'elle décrit (comme l'état `Full` de la pile de la Stack de la figure précédente), peut avoir son propre comportement. Il est possible de préciser ce qui se passe (exécution de méthode, envoie de messages, etc.) :

entry en entrant dans l'état (par exemple initialiser une variable)

doActivity en cours d'état (et après toute `entry` spécifiée)

exit au moment de sortir de l'état (et juste avant la transition sortante elle-même)

États composites

Lorsque le comportement d'un état nécessite plus de détails, on créé un **état composite** (aussi appelé super-état) qui est lui-même une machine à état. On peut ainsi factoriser des transitions déclenchées par le même événement (et amenant vers le même état cible), tout en spécifiant des transitions particulières entre les sous-états. Il est également possible d'attacher un diagramme d'états à un état (composite du coup) pour garder une représentation hiérarchique.

Figure 89. Exemple d'état composite

Régions concurrentes

Un diagramme d'états peut représenter des régions concurrentes (dont les activités peuvent évoluer en parallèle), graphiquement représentées par des zones séparées par des traits pointillés. Chaque région contient ses propres états et transitions.

Figure 90. Exemple de régions concurrentes

Pseudo-états complémentaires

Il existe d'autres concepts utiles :

choice permet de représenter un branchement conditionnel (sur les guardes) d'une même transition.

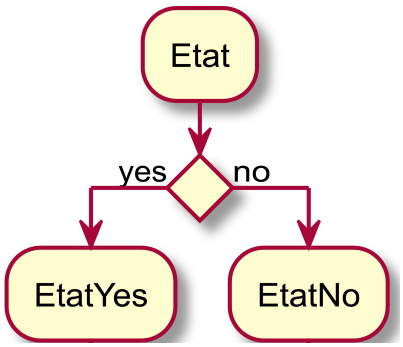


Figure 91. Exemple d'utilisation du choice

fork permet de paralléliser une transition vers deux (ou plus) états de sortie.

join permet de synchroniser deux (ou plus) transitions d'entrée. Par défaut il faut que toutes les transitions entrantes soit tirées (`ET`), mais il est possible de définir plus précisément le comportement du *join*.

shallowHistory permet de replacer l'état composite dans les états au moment de la précedente sortie, mais uniquement au niveau où est placé le pseudo-état.

deepHistory permet également de replacer l'état composite dans les états au moment de la précedente sortie, mais cette fois y compris dans tous les sous-états.

Déclenchement des transitions

Les événements déclencheurs d'une transition peuvent être de nature différente :

Call Event
: déclenché par l'invocation d'une méthode, c'est-à-dire d'une opération fournie par l'élément considéré (le nom de l'opération est utilisé)

Signal Event
: déclenché par l'arrivée d'un signal asynchrone (le nom du signal est utilisé)

Time Event
: déclenché à un certain moment, que ce soit absolu (mot-clef `at`) ou relativement à l'entrée dans l'état (mot-clef `after`). Très utile pour représenter la notion de *Time-Out*.

Change Event
: déclenché par le changement de valeur d'un élément (un attribut par exemple). On utilise le mot-clef `when` Exemple `when "t==10"`

À vous de jouer

Commentaire

Mise en pratique de Papyrus-SysML…

9.4.3. Diagramme d'activité

Le diagramme d'activité est étudié dans ce livre plus en détail au chapitre Section 9.5.3, dans la partie sur les interactions. Il s'agit d'un diagramme souvent utilisé pour représenter le comportement du système, notamment des scénarios des cas d'utilisation. Nous revoyons le lecteur au chapitre Section 9.5.3 pour plus de détails.

9.4.4. En résumé

…

Table 14. Organisation

	Exigences	États	Structures	Interactions	Flux	Transversalités
Organisation						
Environnement						
Vision Opérationelle						
Vision Fonctionnelle						
Vision Organique						

9.4.5. Questions de révision

1. ...

9.5. Interaction

Commentaire

Diagramme de séquence (y compris haut niveau = DS Système). en version description global de comportement (description de processus ⇒ BPMN ?).

9.5.1. Fondements

Table 15. Place des interactions

	Exigences	Structures	Interactions	Transversalités
Organisation			You are here!	
Vision Opérationelle			You are here!	
Vision Fonctionnelle			You are here!	
Vision Organique			You are here!	

On abordera :

- les *Sequence Diagrams*
- les *Activity Diagrams*

Les *State Machines* sont vues dans la partie sur les comportements.

9.5.2. Diagrammes de séquence

Généralités

Il permet de :

- modéliser les interactions
- représenter les échanges de messages
- séquencer ces échanges dans le temps
- spécifier les scénarios des cas d'Utilisation

Les éléments qui composent ce diagramme sont :

Lignes de vie
: des lignes verticales pointillées représentant un élément en interaction, et qui permettent d'indiquer un départ ou une arrivée d'interaction

Barres d'activation
: pour matérialiser sur une ligne de vie quand l'élément correspondant est actif

Messages
: ce qui "circule" d'un élément à l'autre (signal, appel de méthode, ...)

Par commodité nous parlerons aussi de :

Participants
: les éléments en interaction (des parties ou des références généralement), correspondant au nom et au type de la ligne de vie (cadre en haut des lignes de vie).

Les participants représentent des instances, souvent "anonymes". Ils ne sont pas à proprement parler des éléments

Exemple

ToDo

Notions avancées

On peut également représenter des instructions itératives ou conditionnelles au travers de **fragments combinés** (*combined fragments*). Un fragment combiné possède un opérateur (*interaction operator*) et un ou plusieurs opérandes (*operands*).

Les principaux opérateurs sont :

- `loop` (boucle)
- `alt` (alternative)

- `opt` (optionel)
- `par` (parallèle)

Chaque opérande est assortie d'une `guarde` qui permet de déterminer les conditions sous lesquelles cette partie du scénario s'exécute.

```
-- tiré de [Fowler2004]
procedure distribuer
  foreach (ligne)
    if (produit.valeur
                  > $10000
      spécial.distribuer
    else
      standard.distribuer
    endif
  end for
  if (nécessiteConfirmation)
    coursier.confirmer
  end procedure
```

Figure 92. Exemple d' algorithme... (source [Fowler2004])

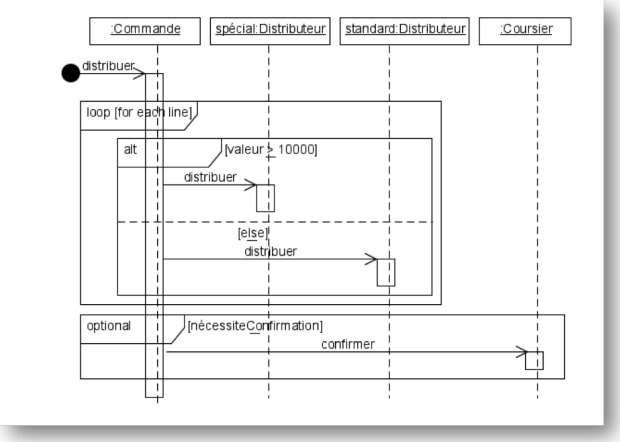


Figure 93. Et le diagramme correpondant (source [Fowler2004])

Exemple de conceptions

Le diagramme de séquence est un diagramme utile pour montrer les "responsabilités" de certains objets par rapport aux autres. Dans un code logiciel, on peut y déceler plus facilement que tel objet est plus chargé que d'autres. Les deux diagrammes suivants (tirés de [Fowler2004]) montrent deux conceptions différentes possibles pour l'implémentation d'une même fonctionnalité. On mesure visuellement assez bien la différence entre la version "centralisée" (Figure 94) et la version "objet" (Figure 95).

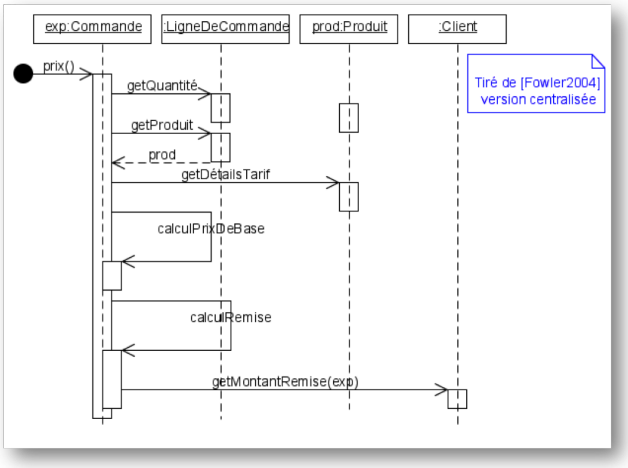


Figure 94. Conception "centralisée" (source [Fowler2004])

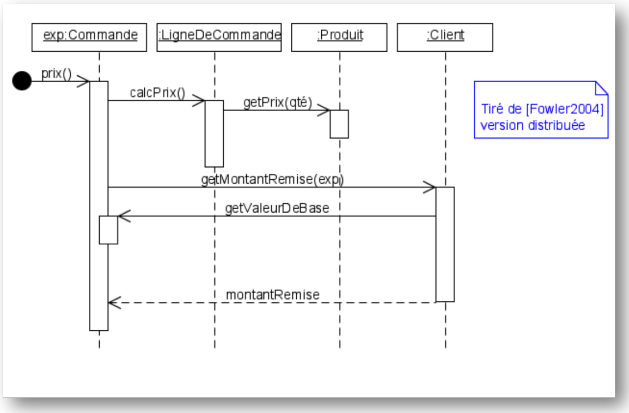


Figure 95. Conception "objet" (source [Fowler2004])

On utilise le diagramme de séquence pour représenter des algorithmes et des séquencements temporels. Lorsque le comportement se rapproche plus d'un flot, on utilise le diagramme d'activité (cf. section sur le Section 9.5.3).

Lien entre UC, DSS et DS

La décomposition hiérarchique permet une description "*TOP-DOWN*" du système à réaliser.

On fait un Diagramme de Séquence Système pour chaque cas d'utilisation (issu du Diagramme d'UC) pour déterminer les échanges d'informations entre l'acteur et le système.

Ensuite on fait un Diagramme de Séquence (DS) pour décrire comment les blocs composant le système (issus du `bdd`) collaborent pour réaliser le traitement demandé.

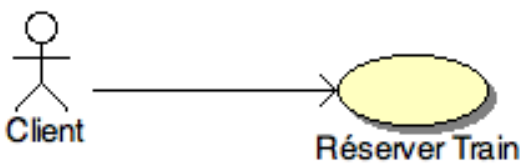


Figure 96. Diagramme d'UC

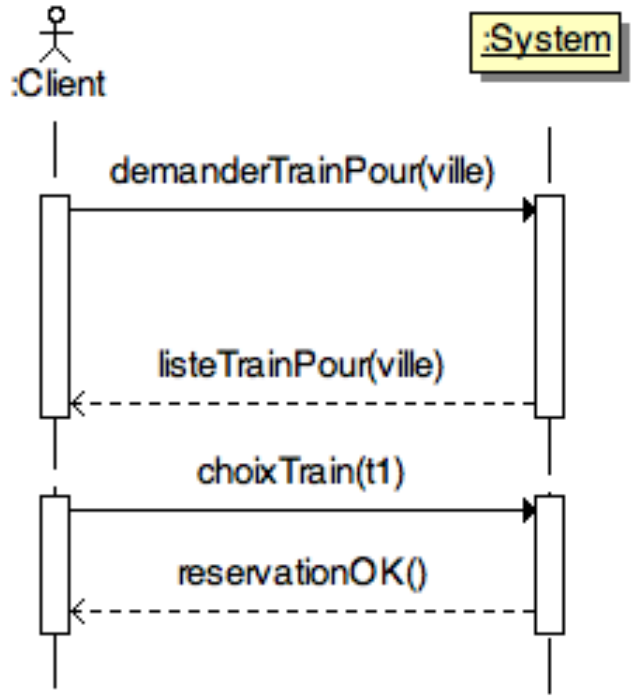


Figure 97. Le DSS correspondant

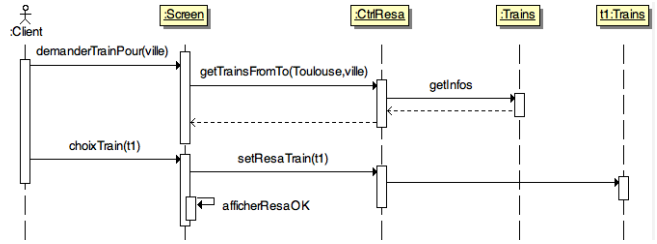


Figure 98. Le DS correspondant

Cadres référence

Les diagrammes de séquence représentent une interaction qui peut être utilisée dans une autre interaction (à la manière d'un appel de fonction en programmation). L'opérateur *ref* est alors utilisé.

Figure 99. Exemple de diagramme de séquence référençant un autre diagramme de séquence

Le temps dans les diagrammes de séquence

Il est possible d'ajouter des contraintes liées au temps dans un diagramme de séquence :

- des contraintes de durée entre 2 événements

- des contraintes de temps pour spécifier des instants dans un scénario

Figure 100. Exemple de contrainte de durée dans un diagramme de séquence

Pour aller plus loin…

Nous n'avons pas présenté dans ce livre un certain nombre de concepts complémentaires :

- certains opérateurs (`strict`, `break`, `critical`, …), relativement peu utilisés;
- les invariants d'état, qui permettent de contraindre les interactions en fonction de l'état dans lequel se trouve le participant;
- la possibilité de décomposer les lignes de vie, pour représenter les comportements internes.

À vous de jouer

Commentaire

Mise en pratique de Papyrus-SysML…

9.5.3. Diagrammes d'activité

Le diagramme d'activité (*Activity Diagrams*) permet de décrire les traitements. Il sert très souvent à décrire plus en détail les cas d'utilisation. Il est utilisé pour représenter les flots de données et de contrôle entre les actions. Il est utilisé en général pour détailler un cas d'utilisation. Il est utilisé pour l'expression de la logique de contrôle et d'entrées/sorties. Le diagramme d'activité sert non seulement à préciser la séquence d'actions à réaliser, mais aussi ce qui est produit, consommé, ou transformé, au cours de l'exécution de cette activité.

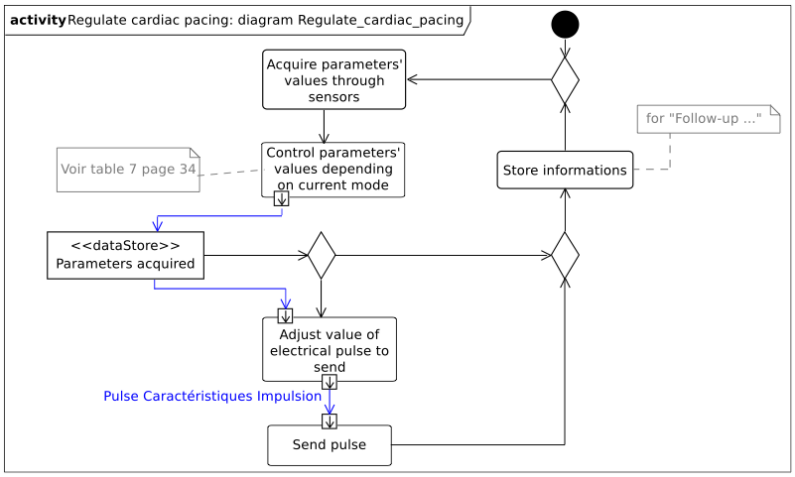


Figure 101. Exemple de diagramme d'activité (extrait de [SeeBook2012])

Les éléments de base du diagramme d'activité sont :

- les activités,
- les actions,
- les flots de contrôle entre actions,
- les décisions (branchements conditionnels),
- un début et une ou plusieurs fins possibles.

Activités

Une activité représente les aspects algorithmiques d'un comportement. Elle permet de modéliser un processus par exemple. Une activité possède :

- un ensemble de paramètres (*input*,*output*,*input/output*)
- un ensemble d'actions (appel de méthode, lecture, écriture, ...)
- un ensemble de flots entre actions (de contrôle ou de données)

Actions

Les actions sont les unités fondamentales pour spécifier les comportements en SysML®. Une action représente un traitement ou une transformation et ne peuvent pas être décomposées (vous êtes sûrement en présence d'une activité si vous ressentez le besoin de décomposer...). Les actions sont contenues dans les activités, qui leur servent alors de contexte.

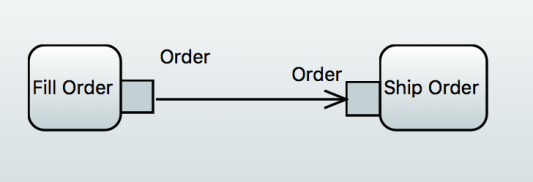


Figure 102. Transmission d'objet entre 2 actions

Il existe un certain nombre de types d'actions prédéfinies en SysML® (héritées d'UML®) :

Call operation	invocation d'une opération sur un objet. Utilisé pour appeler une méthode.
Call behavior	invocation d'une activité. Utilisé pour réutiliser une activité déjà décrite.
Send	création d'un message et transmission (asynchrone) à un objet cible. Utilisé pour envoyer des signaux.
Accept event	attente de la réception du type d'événement spécifié (un signal le plus souvent). Utilisée pour recevoir des signaux (asynchrones).
Accept call	idem que le précédent mais pour les appels synchrones.
Reply	transmission d'un message en réponse à un *accept call*.
Create	création d'une instance (de bloc ou, plus généralement, d'objet).
Destroy	destruction d'une instance.
Raise exception	pour lever une exception.

Flots

Un **flot de contrôle** permet le contrôle de l'exécution des noeuds d'activités. Les flots de contrôle sont des flèches reliant deux noeuds (actions, décisions, etc.).

Le diagramme d'activité permet également d'utiliser des **flots d'objets** (reliant une action et un objet consommé ou produit). Les *object flow*, associés aux broches d'entrée/sortie (*input/output pin*) permettent alors de décrire les transformations sur les objets manipulés.

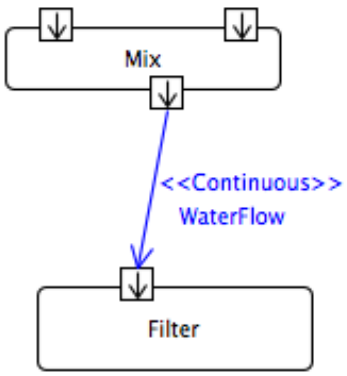


Figure 103. Un exemple de flot continu

Pour permettre la modélisation des **flots continus**, SysML® ajoute à UML® la possibilité de caractériser la nature du débit qui circule sur le flot : continu (par exemple, courant électrique, fluide, etc.) ou discret (par exemple, évenements, requêtes, etc.). On utilise pour cela des stéréotypes : `<<continuous>>` et `<<discrete>>`. Par défaut, un flot est supposé discret.

Définition : FlowProperty (OMG SysML v1.5, p. 82)

A FlowProperty signifies a single flow element to/from a block. A flow property has the same notation as a Property only with a direction prefix (in | out | inout). Flow properties are listed in a compartment labeled flow properties.

Décision

Une décision est un noeud de contrôle représentant un choix dynamique entre plusieurs conditions (mutuellement exclusives). Elle est représentée par un losange qui possède un arc entrant et plusieurs arcs sortants. Il existe plusieurs noeuds de contrôle (cf. Figure 104) :

fork

Un *fork* est un noeud de contrôle représentant un débranchement parallèle. Il est représenté par une barre (horizontale ou verticale) qui possède un arc entrant et plusieurs arcs sortants. Le *fork* duplique le "jeton" entrant sur chaque flot sortant. Les jetons sur les arcs sortants sont indépendants et concurrents.

join

Un *join* est un noeud de contrôle structuré représentant une synchronisation entre actions (rendez-vous). Il est représenté par une barre (horizontale ou verticale) qui possède un arc sortant et plusieurs arcs entrants. Le *join* ne produit son jeton de sortie que lorsqu'un jeton est disponible sur chaque flot entrant (d'où la synchronisation).

flow final

Contrairement à la fin d'activité qui est globale à l'activité, la fin de flot est locale au flot concerné et n'a pas d'effet sur l'activité englobante.

merge

La fusion est l'inverse de la décision : le même symbole du losange, mais cette fois-ci avec plusieurs flots entrants et un seul sortant.

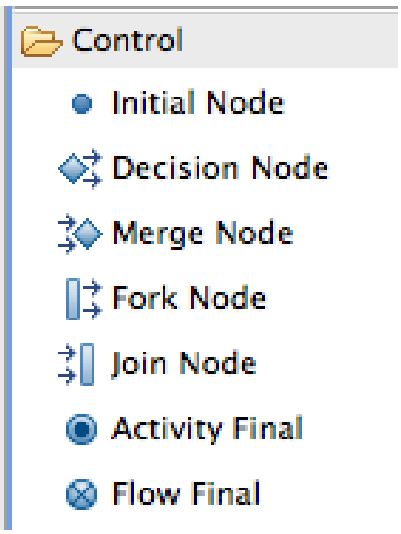


Figure 104. Les différents contrôles de flow SysML

Pour se rapprocher de SADT/SART, la norme prévoit la possibilité d'utiliser les pointillés pour les flux de contrôle.

Définition : ControlFlow (OMG SysML v1.5, p. 120)

Control flow may be notated with a dashed line and stick arrowhead…

9.5.4. Réutilisation

Les activités peuvent être réutilisées à travers des actions d'appel (*callBehaviorAction*). L'action d'appel est représentée graphiquement par une fourche à droite de la boîte d'action, ainsi que par la chaîne : `nom d'action : nom d'activité`. SysML® propose encore bien d'autres concepts et notations, comme la région interruptible, la région d'expansion ou encore les flots de type *stream* qui sortent du cadre de ce livre d'introduction.

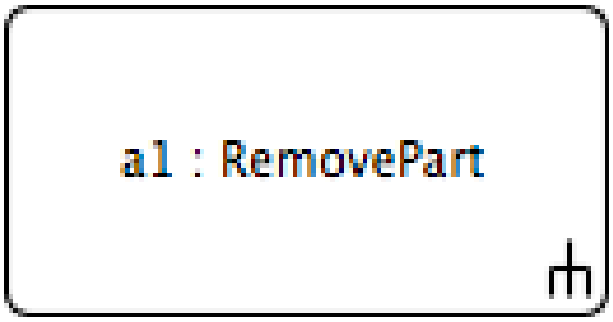


Figure 105. Exemple de callBehaviorAction

9.5.5. À vous de jouer

Commentaire

Mise en pratique de Papyrus-SysML…

9.5.6. En résumé

Il existe de nombreux diagrammes pour exprimer les comportements. Ces modèles sont importants dans la mesure où ils peuvent servir à valider le futur système vis-à-vis de ces comportements exprimés. Ils ne sont donc véritablement utiles que lorsqu'ils sont couplés à des outils de simulation ou d'analyse (cf. Section 11.2.8).

Table 16. Place du Comportement

	Exigences	Structures	Interactions	Transversalités
Organisation			`pkg`	
Vision Opérationelle			`uc sd`	

	Exigences	Structures	Interactions	Transversalités
Vision Fonctionnelle			`dss sd act`	
Vision Organique			`stm`	

9.5.7. Questions de révision

1. Comment, pour exprimer un comportement, savoir si j'ai besoin d'un diagramme de séquence plutôt qu'un diagramme d'activité ou encore d'une machine à état ?

9.6. En résumé

SysML® est une notation complexe et souvent la difficulté pour un débutant n'est pas tant la notation elle-même, que déterminer à quel endroit, avec quel diagramme et quels éléments la modélisation d'un concept particulier va poivoir être réalisé.

Table 17. Organisation

	Exigences	États	Structures	Interactions	Flux	Transversalités
Organisation						
Environnement						
Vision Opérationelle						
Vision Fonctionnelle						
Vision Organique						

9.7. Questions de révision

1. …

[10] et vice-versa!

[11] but not shared aggregation

Chapter 10. Préoccupations transverses de modélisation

Commentaire

Traçabilité, allocation etc.

On abordera :

- Les aspects organisationnels
- La traçabilité des exigences
- Les mécanismes d'allocation
- Le diagramme paramétrique (aspects transveres)

10.1. Organisation

	Exigences	Structures	Interactions	Transversalités
Organisation				You are here!
Vision Opérationelle				
Vision Fonctionnelle				
Vision Organique				

10.1.1. Fondements

On abordera :

- Le diagramme de paquetages (*Package Diagram*)
- Les différent types de *packages*
- Les organisations possibles
- La notion de *Namespaces*
- Les *Dependencies*
- Les adaptations graphiques

10.1.2. Le diagramme de paquetages (pkg)

Le diagramme de paquetages (*Package Diagram*) permet de représenter l'organisation des modèles en paquetages. Nous dirons plus volontiers 'diagramme de *package*' dans cet ouvrage. Le terme "paquetage" est en effet très peu utilisé dans les communautés de développement et de modélisation.

Concepts définis dans cette section

Nous aborderons les concepts du niveau initial (*Model User*) de la certification *Certified Systems Modeling Professional*™: Package Diagram, ownership, namespace, containment, dependency, view, viewpoint

Ce diagramme est identique à celui d'UML®, et le concept de paquetage (*package*) est classique pour les développeurs (java notamment). Il permet d'organiser les modèles en créant un espace de nommage (cf Section 10.1.5).

10.1.3. Les différent types de *packages*

Il existe plusieurs types de *packages* :

`models`	un *package* "top-level" dans une hiérarchie de *packages*
`packages`	le type le plus classique : un ensemble d'éléments de modèles
`model librairies`	un *package* prévu pour être réutilisé (importé) par d'autres éléments
`views`	un *package* spécial pour représenter les points de vue

Un point de vue (*viewpoint*) est utilisé pour matérialiser une perspective particulière de modélisation. Il possède des propriétés standardisés (*concerns*, *language*, *purpose*, etc.) et permet d'indiquer qu'une vue (un *packetage* particulier, stéréotypé <<view>>) est conforme (dépendance <<conform>>) à un point de vue.

10.1.4. Les organisations possibles

Les modèles peuvent être organisés selon toutes sortes de considération :

- par hiérarchie "système" (e.g., entreprise, système, composant, ...)
- par types de diagrammes (e.g., besoins, structure, comportements, ...)
- par cycle de vie (e.g., analyse, conception, ...)
- par équipes (e.g., architectes, [IPT], ...)
- par points de vue (e.g., sécurité, performance, ...)
- etc.

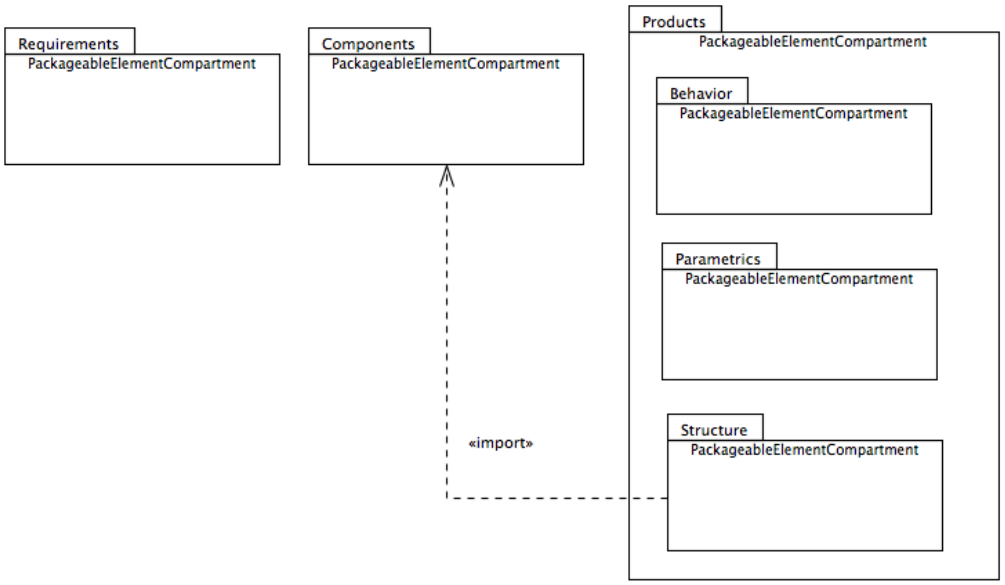


Figure 106. Exemple d'organisation simple

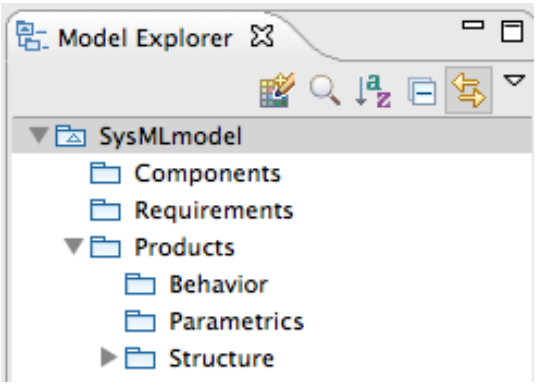


Figure 107. Représentation de cette organisation dans un outil

10.1.5. La notion de *Namespaces*

Un *package* permet de créer un espace de nommage pour tous les éléments qu'il contient. En effet une hiérarchie de *packages* est basée sur la notion d'appartenance (ou de contenance, *containment*), c'est-à-dire de possession (*ownership*). Ainsi dans un *package* on n'a pas à se soucier des noms des éléments. Même si d'autres utilisent les mêmes noms, il n'y aura pas ambiguïté.

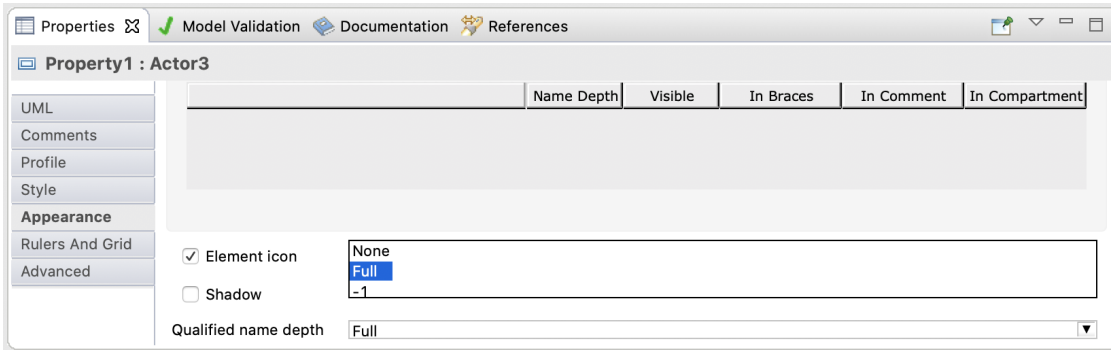
Définition : Namespace (OMG SysML v1.5, p. 23)

The package defines a namespace for the packageable elements.

Pour éviter toute ambiguïté dans les diagrammes, on peut utiliser pour les éléments de modèles leur nom complet (*Qualified name*), c'est-à-dire le nom de l'élément préfixé par son (ou ses) *package(s)* (e.g., `Structure::Products::Clock`).

Dans les outils SysML®, il faut souvent demander explicitement à voir les noms complets (*Qualified names*) des éléments (la plupart du temps dans les options graphiques).

Pour Papyrus-SysML, il suffit sur l'émément sélectionné d'aller dans l'onglet **Property › Appearance › Qualified name Depth** et de modifier la valeur par défaut, comme illustré ci-dessous.



10.1.6. Les dépendances

Un certain nombre de dépendances peuvent exister entre des éléments de *package* ou entre les *packages* eux-mêmes :

Dependency une dépendance "générale", non précisée, représentée par une simple flèche pointillée ----->

Use l'élément "utilise" celui à l'autre bout de la flèche (un type par exemple), représentée par le stéréotype <<use>>

Refine l'élément est un raffinage (plus détaillé) de celui à l'autre bout de la flèche, représentée par le stéréotype <<refine>>

Realization l'élément est une "réalisation" (implémentation) de celui à l'autre bout de la flèche, représentée par le stéréotype <<realize>>

Allocation l'élément (e.g., une activité ou un *requirement*) est "alloué" sur celui à l'autre bout de la flèche (un `bloc` la plupart du temps), représentée par le stéréotype <<allocate>>

10.2. Matrices de dépendances

Commentaire

Expliquer ici comment réaliser une matrice de dépendance

Nous vous conseillons de créer (si ce n'est déjà fait) un *package* pour rassembler vos modèles de traçabilité. Vous pouvez même avoir des *packages* dédiés (par exemple `Stakeholder Traceability Model`).

Pour créer une matrice de traçabilité avec {papurys} : . Placez-vous dans le *package* où vous souhaitez votre table . **ClickDroit › New Table › Relationship Generic Matrix** . Sélectionnez les données d'entrée des lignes et des colonnes dans l'onglet `Matrix` de la table (dans ses `Properties`). Ajoutez éventuellement un filtre. Par exemple pour n'avoir que les éléments possédant le stéréotype +[stakeholder] :

- Clickez sur le ▫ du champs `filter` des lignes (`row`) ou des colonnes (`columns`)
- Sélectionnez un filtre (`HasAppliedStereotypesExpression` pour notre exemple)

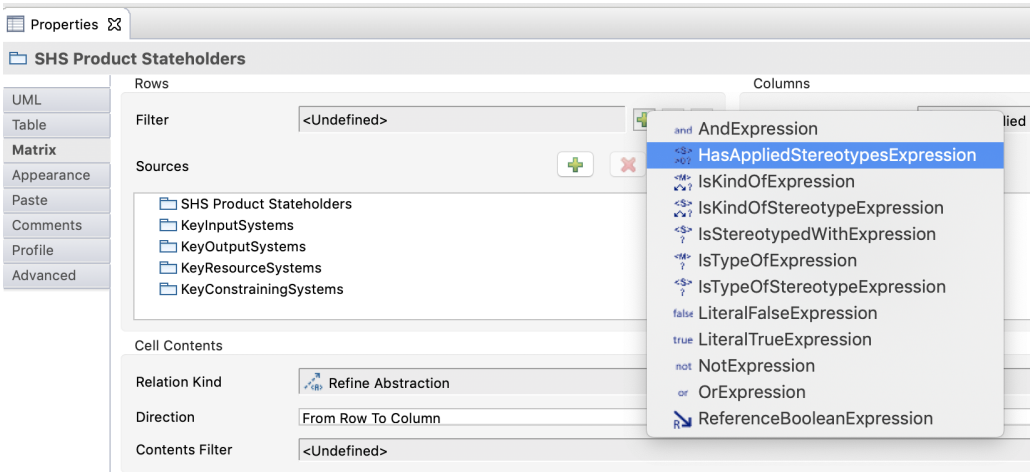


Figure 108. Sélection d'un filtre sur les entrées d'une table

- Donnez le nom du stéréotype recherché dans le champs `Name` (`stakeholder` pour notre exemple)

1. Sélectionnez le `Relation Owner` où seront générés les relations correspondantes aux cases cochées dans la matrice. Par exemple `Table Root Element` mettra les relations dans le même *package* que la table.
2. Vous n'avez plus qu'à cocher les cases et observer les relations nouvelles créées dans le *package*.

Vous puvez même réaliser un diagramme de bloc en glissant-déposant les relations pour obtenir directement un diagramme de traces.

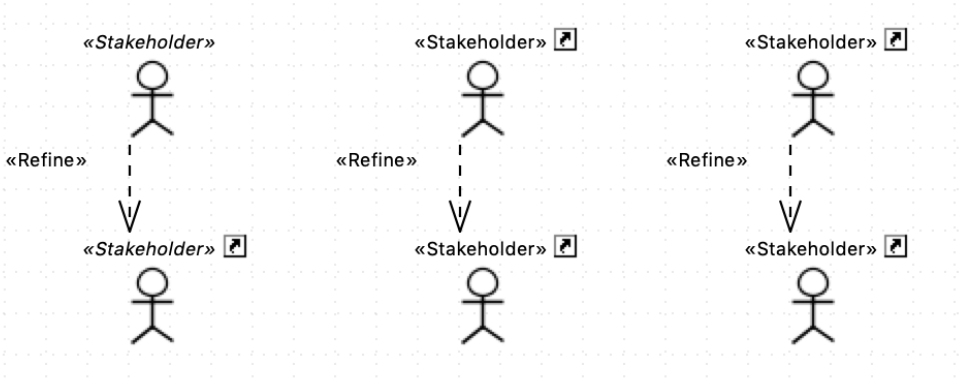


Figure 109. Exemple de diagramme de traces

10.3. La traçabilité des exigences

Nous avons vu déjà un certain nombre de mécanismes SysML® qui permettent de tracer les exigences. Nous les regroupons ici dans une matrice spécifique (qui se lit dans le sens des relations, par exemple un élément de structure comme un bloc `<<satisfy>>+ une exigence).

Table 18. Traçabilité

	Exigences	Structures	Interactions
Exigences	<<deriveReqt>>, <<refine>>, <<copy>>		
Structures	<<allocate>>, <<satisfy>>		<<allocate>>
Interactions	<<refine>>		

Comme indiqué dans le tableau ci-dessus, en général, le lien de raffinement est utilisé entre une exigence et un élément comportemental (état, activité, `uc` , etc.) tandis que l'allocation concerne principalement les éléments de structures.

XXX Mettre un exemple avec tous ces liens. XXX

10.4. Les mécanismes d'allocation

Un mécanisme nouveau en SysML® et important pour lIngénierie Système est le mécanisme d'**allocation**. Il permet de préciser quel élément conceptuel (comme un comportement ou une activité) est alloué sur quel élément physique. Il est possible d'exprimer cette allocation de plusieurs manières.

XXX ToDO XXX

- Parler du `<<AllocatedTo>>+, compartiments des blocs et autres annotations.
- Parler des zones d'allocation dans les machines à états où les diagrammes d'activité par exemple.
- Parler des `<<allocate>>+.

10.5. Les adaptations graphiques

Commentaire

Parler des feuilles de style. Associer les couleurs CESAM par exemple.

10.6. En résumé

SysML® propose un certain nombre de mécanismes pour organiser les différents modèles, tirés pour la plupart d'UML®. Ces mécanismes seront plus faciles à comprendre au travers de leur utilisation concrète dans la suite.

Table 19. Organisation

	Exigences	Structures	Interactions	Transversalités
Organisation	`package`	`package`	`package`	`dependencies`
…				

10.7. Questions de révision

1. Quels sont les 5 types de dépendances entre *packageable elements* ?
2. À quoi cela peut-il servir de définir les dépendances (donnez des exemples concrets) ?
3. Quelles sont les différences entre `<<satisfy>>+ et `<<allocate>>+ ?
4. Pourquoi est-il important de relier un *use case* à au moins un *requirement* ?
5. L'inverse est-il aussi important ?

Chapter 11. Modéliser oui, mais...

Commentaire

On aborde ici des éléments souvent absent des livres : l'utilisation des outils en milieu industriel! Par exemple et en vrac :

- Versionner et configurer, comparer et fusionner
- Simuler
- Analyser
- Comment utiliser les paramétrique pour faire le liens entre les outils de moélisation et les outils d'analyse)
- Tester
- Générer du code
- Documenter
- Personnalisation de l'outil à un contexte/domaine spécifique

11.1. Collaborer et gérer les version de modèles

Commentaire

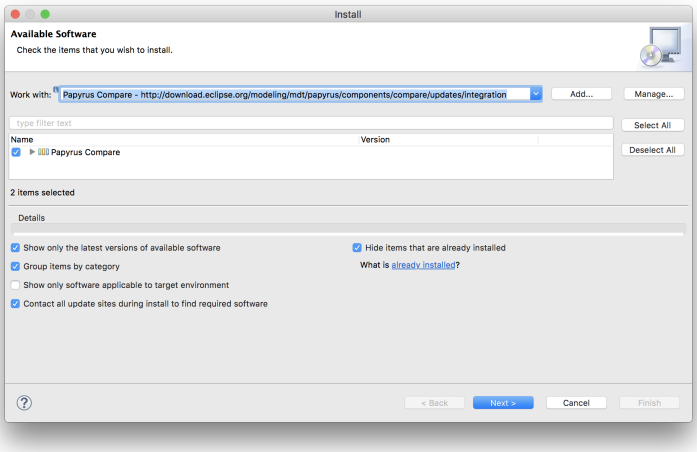
JMB ⇒ reprendre l'expérience du Master DL avec EMF Compare et Egit.

Pour pouvoir collaborer à plusieurs autour du même modèle, il est nécessaire d'utiliser des outils appropriés. Dans cette section nous illustrons l'utilisation des plugins eclipse : EGit (qui doit déjà être intégré à votre eclipse et Papyrus Compare.

11.1.1. Vade Mecum

Voici quelques instructions qui vous permettrons de gérer simplement les version de vos modèles :

- Installez également le plugin Papyrus Compare en allant sur **Help › Install New Software › ...** et en entrant l'URL : http://download.eclipse.org/modeling/mdt/papyrus/components/compare/updates/nightly/



eclipse vous demandera sûrement de redémarrer.

- Allez dans **Préférences › Team › Git › Synchronize** et sélectionner `Recursive Model Merge Strategy` dans `Preferred merge strategy`

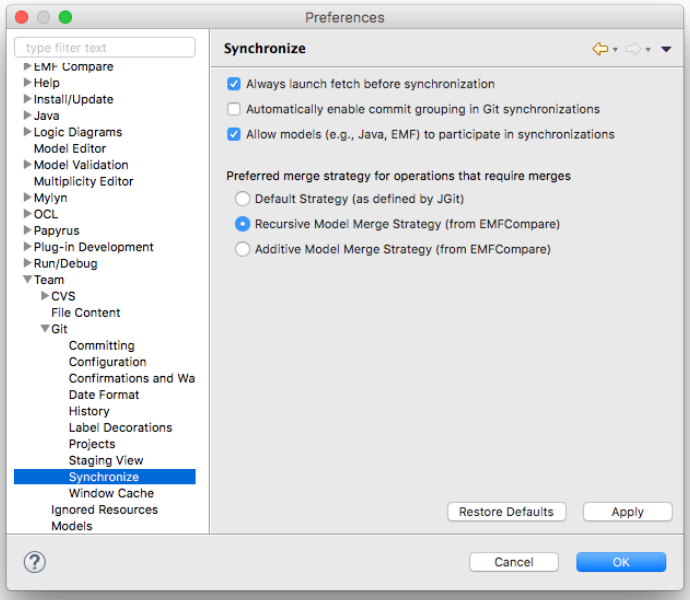


Figure 110. Pensez à sélectionner la bonne stratégie de Merge

Si vous n'avez pas cette option à cocher, vous n'avez pas la bonne version des plugins! ☺.

11.1.2. Exemple concret d'utilisation

Nous avons expérimenté avec succès[12], l'utilisation de cette solution avec 24 étudiants collaborant sur le même modèle Papyrus-SysML.

Voici quelques principes généraux qui permettent de modéliser de manière collaborative sans trop de difficulté :

- les utilisateurs doivent maîtriser les concepts de versionnement et les commandes Git en général
- la "racine" du modèle, à savoir les principaux éléments (par exemple un bloc `Capteur` dans notre étude de cas, duquel tous les capteurs spécifiques hériteront), les exigences initiales, le diagramme de contexte, etc. doivent être réalisé avant le démarrage de la partie collaborative pour que tout le monde parte du même système
- il est préférable que chaque contributeur travaille sur une partie relativement indépendante, pour éviter au maximum les conflits
- il est préférable que chaque contributeur travaille sur une branche dédiée et qu'il vérifie localement que l'intégration de sa branche dans la branche principale ne va pas poser de problème
- un seul contributeur est chargé de *merger* les branches individuelles sur la branche principale

11.2. Compléments Papyrus

Commentaire

Voir avec Seb ce qu'on garde, ce qu'on remonte dans la partie principale (section [gettingStarted]), ce qu'on jette…

11.2.1. Personnaliser les styles

11.2.2. Layer Support

https://wiki.eclipse.org/Papyrus/UserGuide/Layers

11.2.3. Exécution de modèles

Pour bien concevoir un modèle, particulièrement s'il est dynamique (s'il représente un comportement), il est important de pouvoir le manipuler, l'animer.

Les initiatives récentes de l'OMG™ pour développer une sémantique exécutable à UML®, appelée fUML, ont été implémentées dans plusieurs outils :

- Une implémentation récente de fUML est disponible ici : http://modeldriven.github.io/fUML-Reference-Implementation/
- Une autre implémentation est disponible dans l'outil Cameo Simulation Toolkit de MagicDraw.
- Pour aller plus loin avec Papyrus-SysML, qui dispose aussi de sa propre implémentation de fUML, appelée Moka, consultez la documentation spécifique : https://wiki.eclipse.org/Papyrus/UserGuide/fUML_ALF.

11.2.4. Reverse Engineering

11.2.5. Fragmenter un modèle

11.2.6. Papyrus for Requirements

11.2.7. Scripter Papyrus

Il est possible d'exécuter des scripts…

1. Ajouter le complément EASE à Papyrus-SysML en utilisant l'update-site suivant : https://download.eclipse.org/modeling/mdt/papyrus/components/ease/2019-03/.
2. Ouvrir l'exemple qui vient avec (**File › New › Example… › Papyrus EASE › Papyrus EASE Jupyter Example**). Cf. Figure 111.

Cela crée un projet dans votre workspace, lisez le fichier `Readme.MD` qui donne quelques instructions.

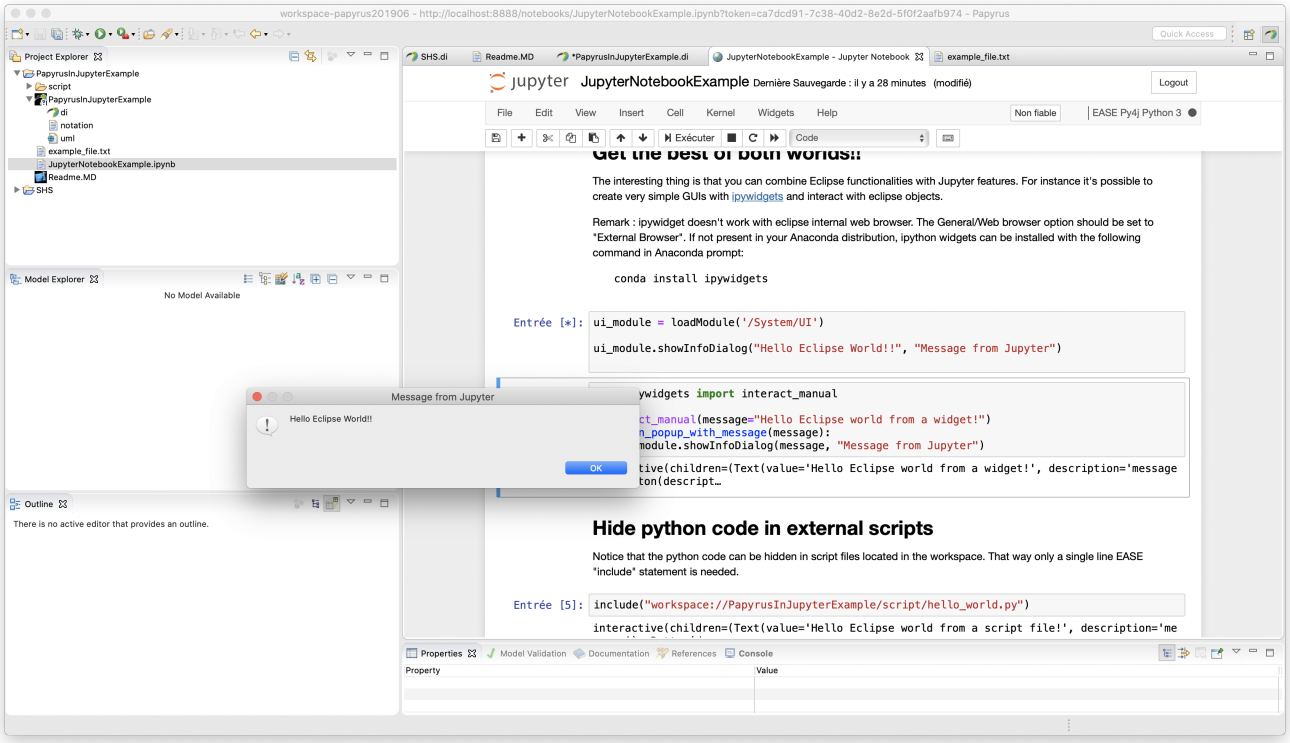


Figure 111. Exemple de scripting de Papyrus

11.2.8. Analyser les modèles

[12] dans le cadre du cours SysML® du Master DL

Références

Articles et livres

- [Dickerson] 2013
- [OCSMP] Certified Systems Modeling Professional. http://www.omg.org/ocsmp/
- [Estefan] 2007/2008 INCOSE
- [Ramos] 2012 IEEE
- [Friedenthal2016] Le livre de Friedenthal
- [SysML] OMG Systems Modeling Language (OMG SysML). Version 1.5. formal/2017-05-01. Available here.
- [UML] ISO/IEC 19501:2005 - Information technology - Open Distributed Processing - Unified Modeling Language (UML) Version 1.4.2". Iso.org. 1 April 2005. Retrieved 7 May 2015.
- [Kordon] Embedded Systems Analysis and Modeling with SysML, UML and AADL, F. Kordon, J. Hugues, A. Canals, A. Dohet, Wiley, 2013.
- [REQ2012] Guide Bonnes Pratiques en Ingénierie des Exigences, AFIS 2012.
- [Sommerville1997] Ian Sommerville, Pete Sawyer. Requirements Engineering: A Good Practice Guide. Wiley, 1997.
- [Fowler2004] Martin Fowler. UML 2.0 INFORMATIQUE PROFESSIONNELLE, 2004.
- [OMG2009] The Current State of Model Based Systems Engineering: Results from the OMG SysML Request for Information.
- [Neptune17] Conférence Neptune'2017. CNAM Paris. 01/06/2017.
- [Jézéquel] Le bouquin sur l'ingénierie des modèles de Benoit et al.
- [Cabot] Le bouquin de Jordi
- [Roques] Le bouquin de Pascal
- [CESAM17] CESAM: CESAMES Systems Architecting Method — A Pocket Guide. Daniel KROB, January 2017. Available here.

Websites

- Papyrus
- CESAM

Appendix A: Acronymes

Liste (non exhaustive) des acronymes utilisés dans ce livre :

ACT	***ACT**ivity Diagram*, le diagramme d'activité (cf. Section 9.5.3)
BDD	***B**lock **D**efinition **D**iagram*, le diagramme de définition de blocs (cf. Section 9.3.3)
CESAM	***CES**AMES **A**rchitecture **M**ethod*, le *framework* d'architecture et de modélisation de CESAMES™ (cf. Section 6.4)
DS	**D**iagramme de **S**équence (cf. [seq])
DSS	**D**iagramme de **S**équence **S**ystème (cf. [seq])
fUML	***f**oundational **UML***, un sous-ensemble du standard UML® pour lequel il existe une sémantique d'exécution standard et précise
IBD	***I**nternal **B**lock **D**iagram*, le diagramme de blocs internes (cf. Section 9.2.2)
IDE	***I**ntegrated **D**evelopment **E**nvironment*, environnement de développement intégré (comme eclipse ou IntelliJ)
IS	**I**ngénierie **S**ystème
MBSE	***M**odel-**B**ased **S**ystems **E**ngineering*, Ingénierie systèmes basée modèles
OOSEM	***O**bject-**O**riented **S**ystems **E**ngineering **M**ethod*)
PAR	***Par**ametric Diagram*, le diagramme paramétrique (cf. Section 9.3.4)
PKG	***Pac**k**ag**e Diagram*, le diagramme des paquetages (cf. Section 10.1.2)
REQ	***REQ**uirements Diagram*, le diagramme des exigences (cf. Section 9.1)
RTF	***R**evision **T**ask **F**orce*
SD	***S**equence **D**iagram*, le diagramme de séquences (cf. [seq])
STM	***ST**ate **M**achine Diagram*, le diagramme d'états (cf. Section 9.4.2)
SysML®	***Sys**tems **M**odeling **L**anguage*

UC ***U**se **C**ase Diagram*, le diagramme des cas d'utilisation (cf. Section 9.2.3)

UML® ***U**nified **M**odeling **L**anguage*

Appendix B: Traductions

Commentaire

Peut-être à merger avec l'index… Mais utile pour l'instant pour se mettre d'accord sur les termes français.

De nombreux utilisateur de SysML® sont habitués aux termes anglais de cette notation. Idem pour les utilisateurs de CESAM. Nous reprenons ici la liste des traductions françaises que nous avons utilisées (et qui ne sont pas nécessairement standards) pour permettre au lecteur de faire une correspondance.

Activity Diagram	Diagramme d'activité (cf. Section 9.5.3)
Actor	Acteur
Block Definition Diagram	Diagramme de définition de blocs (cf. Section 9.3.3)
Constructional flows or objects	Flux ou objets organiques
Constructional requirements	Exigences organiques
Constructional scenarios	Scénarios organiques
Constructional vision	Vision organique
Components	Composants
Composite State	État composite (ou super-état)
***Diagram Frame* (et *Header*)**	Cadre (et cartouche) (cf. Section 5.2.8)
Dynamics	Comportement
Environment Architecture	Contexte
Expected properties	Propriétés attendues
Flows	Flux
Functional flows or objects	Flux ou objets fonctionnels
Functional modes	Modes fonctionnels

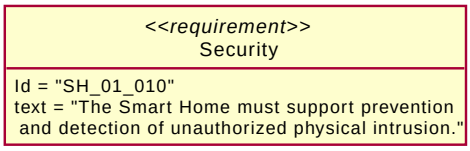
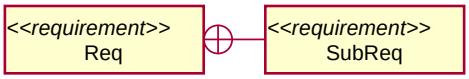
Functional requirements	Exigences fonctionnelles
Functional scenarios	Scénarios fonctionnels
Functional vision	Vision fonctionnelle
Functions	Fonctions
Internal Block Diagram	Diagramme de blocs internes (cf. Section 9.2.2)
Missions	Missions
Multiplicity	Cardinalité (ou Multiplicité)
Needs	Besoins
Operational contexts	Contextes opérationnels (cf. Section 8.4.2.3)
Operational vision	Vision opérationnelle (cf. Section 8.4)
Operational flows or objects	Flux ou objets opérationnels (cf. Section 8.4.2.6)
Operational scenarios	Scénarios opérationnels (cf. Section 8.4.2.5)
Parametric Diagram	Diagramme paramétrique (cf. Section 9.3.4)
Package	Paquetages, mais nous préférons garder Package, admis dans le vocabulaire courant, du moins en informatique.
Package Diagram	Diagramme des paquetages (cf. Section 10.1.2)
Requirements	Exigences
Requirements Diagram	Diagramme des exigences (cf. Section 9.1)
Sequence Diagram	Diagramme de séquences (cf. [seq])
Stakeholders	Parties prenantes (du système)
States	États
State Machine	Machine à état

State Machine Diagram	Diagramme d'états (cf. Section 9.4.2)
Static elements	Structure
Use Case Diagram	Diagramme des cas d'utilisation (cf. Section 9.2.3)

Appendix C: Notation

Comme indiqué en avant-propos de ce livre, nous n'abordons qu'une partie des concepts SysML®, les plus importants. Nous indiquons dans le tableau ci-dessous le différents concepts du niveau initial (*Model User*) de la certification *Certified Systems Modeling Professional*™ de l'OMG™ [OCSMP].

Diagramme des exigences

Exigence		Composition	

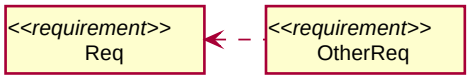
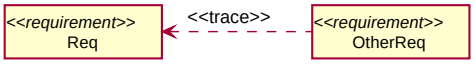
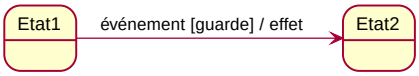
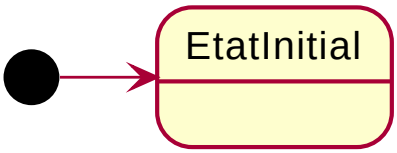
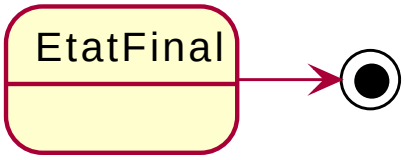
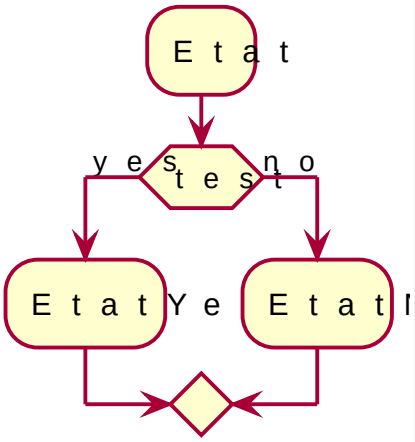
Dépendance		Traçabilité	
Dérive		Vérifie	

Diagramme d’état

États, transition		État initial	

État final		Choix	

Appendix D: Les histoires de SysML et de Papyrus

Histoire de SysML

SysML® est né… Après la définition, à la fin des années 90, du formalisme de modélisation unifié UML®, l'INCOSE a décidé en 2003 de faire d'UML® ce langage commun pour l'IS. UML® contenait en effet déjà à l'époque nombre de diagrammes indispensables, comme les diagrammes de séquence, d'états, de cas d'utilisation, etc. Le travail sur la nouvelle version UML 2.0, entamé à l'OMG™ à peu près à la même période, a abouti à la définition d'un langage de modélisation très proche du besoin des ingénieurs système, avec des améliorations notables sur les diagrammes d'activité et de séquence, ainsi que la mise au point du diagramme de structure composite. Cependant, il restait une barrière psychologique importante à l'adoption d'UML par la communauté de l'IS : sa teinture "logicielle" ! La possibilité d'extension d'UML®, grâce au concept de stéréotype, a permis d'adapter le vocabulaire pour les ingénieurs système. En éliminant les mots "objet" et "classe" au profit du terme plus neutre "bloc", c'est-à-dire en gommant les aspects les plus informatiques d'UML®, et en renommant ce langage de modélisation, l'OMG™ veut promouvoir SysML® comme un nouveau langage différent d'UML®, tout en profitant de sa filiation directe. L'OMG™ a annoncé l'adoption de SysML® en juillet 2006 et la disponibilité de la première version officielle (SysML v1.0) en septembre 2007. Une nouvelle spécification SysML v1.1 a été rendue publique en décembre 2008, puis la v1.2 en juin 2010, la v1.3 en juin 2012, la v1.4 en septembre 2015, et la révision courante SysML® v1.5 a été publiée en mai 2017.

```
=> Le futur et les changements techniques à venir, dans les cartons de l'omg.
```

Histoire de Papyrus

Papyrus-SysML est né…

Appendix E: Couverture des concepts

Comme indiqué en avant-propos de ce livre, nous n'abordons qu'une partie des concepts SysML®, les plus importants. Nous nous sommes basés sur la classification du système de certification de l'OMG™ [OCSMP].

Nous indiquons dans le tableau ci-dessous le différents concepts du niveau initial (*Model User*) de la certification *Certified Systems Modeling Professional*™ (c'est pour cela qu'ils sont listés ici en anglais) et où ils sont abordés dans le livre.

Modèles d'exigences

Concept	Lien
Requirements	5.3, 9.1
deriveRqt, verify, satisfy, refine, trace, containment	9.1.4
Requirements Diagram	5.3, 9.1.5
Use Case Diagram	5.4, 9.1.9, 9.2.3
use case, actor, and subject	9.2.3
association, include, extend, generalization	9.2.3

Modèles structurels

Concept	Lien
Package Diagram	5.11, 10.1.2
ownership, namespace, containment	10.1.5
dependency	10.1.6
view, viewpoint	10.1.3
Block definition, Block usage	
valuetype (with units)	
value properties, parts, references, and operations	
Block Definition Diagram	
compartments	
specialization, associations (including composite[13]), multiplicities	
Internal Block Diagram	
enclosing block	
flow ports and standard ports	
connectors and item flows	
representation of parts	

Concept	Lien
constraint blocks	
Parametric Diagram	
constraint properties, constraint parameters, and constraint expressions	
connecting constraint properties and value properties with binding connectors	

Modèles comportementaux

Concept	Lien
Activity Diagram	
I/O flow including object flow, parameters and parameter nodes, and pins	
control flow including control nodes	
activity partitions (swimlanes)	
actions	
send signal action	
accept event action	
Sequence Diagram	
lifelines, asynchronous and synchronous messages	
interaction references	
State Machine Diagram	
states and regions	
transitions	
trigger by time and signal events, guard, and action	
behaviors (entry, exit, and do)	

Eléments transverses

Concept	Lien
Allocation	10.4
AllocatedFrom and AllocatedTo	
representation (callouts, compartments, allocate activity partitions, and tables)	
special notations for comment, rationale, problem, and constraint	
diagram frames, ports, parameters, and anchors on diagram frames	
diagram header, and diagram description	5.2.8

Concept	Lien
Stereotype	

[13] but not shared aggregation

Appendix F: Nouveautés de SysML 1.6

Pour les lecteurs habitués à SysML® 1.4 nous résumons ici les principales nouveautés de la version 1.5.

Exigences

Nous avons traité en détail cet aspect dans la section Section 9.1.2.2.

Suite

http://model-based-systems-engineering.com/2017/05/17/whats-new-in-sysml-1-5-miscellaneous/

Nouveautés de SysML 1.6

SysML® 1.6 a été publié le 6 décembre 2019! Nous en résumons ici les principales nouveautés.

Block-typed Properties without Associations

Property-specific Type

===

Appendix G: Et le futur? SysML 2!

Quelques mots en provenance de l'OMG™.

Appendix H: Index (Reference guide)

Liste des concepts et renvoies vers leur description dans le livre.

Divers (notes et ToDoList)

Commentaire

- Au sujet de SysML
  - ☑ s'appuyer sur les niveaux tels que définit dans le certificat SysML de l'OMG cf. Niveaux de concepts SysML
- Sur le format :
  - ☐ Limite du nombre de pages ⇒ 250 pages
  - ☐ Liens avec le livre SysML de Sandy ?
  - ☐ Solutions possibles d'organisation globales :
    - si public débutant seulement alors un chapitre sur les possibilités avancés incluant les facilités de personnalisation, intégration avec d'autres formalismes/outils, etc.
    - si public débutant et avancé, alors :
      - Deux parties dédiés.
      - Pour chaque chapitre (quand cela fait du sens), avoir deux niveaux de présentation, débutant et avancé.
      - Des parties bien marquées « avancé » au fil de l'eau.