

Introduction à SysML

Table des matières

I Partie 1 : Introduction	1
1 Avant-propos	2
2 C'est quoi SysML?	6
3 A propos du Bac STI2D	11
4 Un exemple fil rouge	13
II Partie 2 : Ingénierie système	19
5 Introduction	20
6 Différence avec l'ingénierie logicielle	22
7 Normes et standards	27
8 Des documents aux modèles	28
9 Les exigences	29
10 L'architecture du système	31
11 Le comportement du système	32

12 Méthodes et démarches	33
III Partie 3 : La notation SysML	34
13 Pourquoi une nouvelle notation	35
14 Introduction à SysML	36
15 Outils SysML	41
16 Cadre pour les diagrammes	42
17 Organisation	43
18 Les exigences	49
19 L'architecture du système	58
20 Le comportement du système	73
21 Les aspects transversaux	87
IV Partie 4 : Modéliser un système en SysML	90
22 Une démarche parmi d'autres	91
23 Recettes et bonnes pratiques	97
V Partie 5 : Pour aller plus loin	99
24 Considérations méthodologiques	100
25 Analyses et simulation	101
26 Exercices de révision	102

VI Annexes	105
27 Liens utiles	106
28 Conventions	108
29 Le temps et sa prise en compte dans les modèles	109
30 FAQ	110
31 FABQ	115
Bibliographie	116
Bibliographie	116
Glossaire	117
Index	119

Table des figures

2.1	<i>Trends</i> : Twitter	8
2.2	<i>Trends</i> : Google (Carte)	9
2.3	Effet SysML-France?	9
2.4	Principaux domaines d'utilisation	10
2.5	Diagrammes les plus utilisés	10
4.1	Le système CMS	14
6.1	Un système complexe	23
6.2	Un système de système	23
6.3	Des exigences au système	24
6.4	Analyse Fonctionnelle et/ou Comportementale	24
6.5	Analyse Structurelle	25
6.6	Analyse de performance	25
6.7	Analyses spécifiques	26
6.8	Des exigences au système	26
9.1	300 corps de métiers sont parfois présents sur un chantier	29
9.2	Humour (taken from here)	30
14.1	Liens entre UML et SysML	37
14.2	Les 9 diagrammes SysML et leur lien avec UML	38
14.3	Version abrégée des diagrammes	39

16.1 Exemple de diagramme SysML	42
17.1 Exemple d'organisation simple	45
17.2 Représentation de cette organisation dans un outil	45
17.3 Un autre exemple d'organisation	46
18.1 Exemples tableau d'exigences	50
18.2 Import Modelio de tableau d'exigences	51
18.3 Exemples de relations entre exigences	52
18.4 Relations liées au <i>requirements</i> dans TOPCASED	52
18.5 Exemples de composition d'exigences	53
18.6 Exemples de <i>rationale</i> et <i>problem</i>	54
18.7 Exemple de lien entre <i>use case</i> et <i>requirements</i>	54
18.8 Exemple de diagramme des cas d'utilisation	55
18.9 Exemple de démarche (<i>SYSMOD Zigzag pattern</i>)	56
19.1 bdd du système dans son environnement	59
19.2 Exemple de définition de contraintes	60
19.3 Définition de <i>Value Types</i>	61
19.4 Deux façon de représenter une propriété de type B	63
19.5 Exemple de lien de généralisation/spécialisation	64
19.6 Exemple de <i>Parts</i>	65
19.7 Autre exemple de <i>Parts</i>	66
19.8 Exemples de flots	67
19.9 Exemples de flots multi-phérique entre ports	68
19.10 Exemple de contraintes	69
19.11 Exemple de diagramme paramétrique	70
19.12 Exemple de bloc de configuration	70
19.13 Exemples de DSS	71
20.1 Notation dans le diagramme d'UC	76
20.2 Exemple de diagramme de séquence	77

20.3 Exemple d' algorithme	78
20.4 Et le diagramme correpondant	78
20.5 Conception "centralisée"	79
20.6 Conception "objet"	80
20.7 Diagramme d'UC	81
20.8 Le DSS correspondant	81
20.9 Le DS correspondant	81
20.10Un exemple de diagramme d'état (R,UK)	82
20.11Exemple de diagramme d'activité (tiré de [SeeBook2012])	83
20.12Un exemple de flot continu (UK)	84
20.13Les différents contrôles de flow SysML	85
20.14Exemple de <i>callBehaviorAction</i> (UK)	85
21.1 Exemple de diagramme paramétrique	88
22.1 Outilage autour de SysML	92
22.2 Génération de documentation à partir de TOPCASED (1)	92
22.3 Génération de documentation à partir de TOPCASED (1)	93
22.4 Génération de documentation à partir de Rhapsody	93
22.5 Animation Artisan	95
23.1 Le contexte du Pacemaker ([SeeBook2012])	98
26.1 Exemple de QCM sur SysML	103
26.2 Mots-croisés sur SysML	104
29.1 Exemple de contrainte temporelle (tirée de [SysML])	109
30.1 Exemple de divergence modèle/diagramme (diagramme)	111
30.2 Exemple de divergence modèle/diagramme (modèle)	111
30.3 Exemple de partage de <i>requirement</i>	112

Liste des tableaux

5.1 La carte de base	20
14.1 Organisation	39
17.1 Organisation	47
18.1 Déclinaison des Exigences	56
19.1 Place des aspects structurels	72
20.1 Place du Comportement	86
21.1 Traçabilité	87

Remerciements

À faire au dernier moment...

Préface

Exemple de préface...

Première partie

Partie 1 : Introduction

Chapitre 1

Avant-propos

1.1 Sur ce document

A qui est destiné ce document?

Les étudiants qui découvrent le langage, mes collègues enseignants qui cherchent un document de support à leur cours et d'exercice accessible, et... moi-même (pour organiser mes notes diverses)!

A qui n'est-il pas destiné?

Si vous appartenez à l'une de ces catégories, ce livre **n'est pas pour vous** :

- vous cherchez un livre de référence (pour cela, même s'il est en anglais, je conseille [FMS])
- vous voulez vous perfectionner (ce livre n'est qu'une introduction)
- vous souhaitez préparer la certification de l'**OMG** (mieux vaut vous plonger dans la spécification [SysML])

Historique

Ce document est la compilation de plusieurs années d'enseignement de **SysML** depuis 2007, que ce soit :

- au **Master TI**, de l'**Université de Pau et des Pays de l'Adour** (cours d'introduction avec mon collègue et ami Nicolas Belloir),
- au **Master Recherche SAID**, de l'**UPS** (introduction),
- au **Master ICE** de l'**Université de Toulouse II - Le Mirail** (introduction avec mon collègue et ami Pierre de Saqui Sannes),

- au *Master of Science* de Göteborg, Suède (introduction réalisée par Nicolas Belloir),
- à l'[Universidad Autónoma de Guadalajara](#), au Mexique (40h de formation professionnelle aux employés de Continental),
- ou plus récemment au [Master DL-SI](#) de l'[UPS](#).

Vous trouverez en référence (cf. [Bibliographie](#)) les ouvrages et autres documents utilisés.

1.2 Sur l'auteur

- Professeur à l'[Université de Toulouse](#)
- Co-fondateur de l'association [SysML-France](#) en 2009
- Membre du comité éditorial de la revue [Software and System Modeling journal](#) depuis sa création en 2002
- Membre du *Steering Committee* de la conférence ACM/IEEE [MODELS](#) depuis 2008
- Enseignant en modélisation depuis 1995
- Marié, une (merveilleuse) fille

1.3 Comment lire ce document?

Version électronique

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.



Note

Si vous lisez la version papier de ce document, ces liens clickables ne vous servent à rien, mais n'hésitez pas à en consulter la version [électronique](#)!

Conventions typographiques

J'ai utilisé un certain nombre de conventions personnelles 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, présentées en fin de document (cf. [Bibliographie](#)),
- tous les *flottants* (figures, tableaux) sont listés à la suite de la table des matières,

- 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*).

Les figures, sauf mention contraire, ont été réalisées avec l'outil **TOPCASED** en français. Le titre des figures indique (entre parenthèses) un R pour les figures issues de **Rhapsody** et un UK pour les figures en anglais. Pour les conventions (de nommage notamment), cf. Chapitre 28.

**Note**

Les notes comme celles-ci sont utilisées pour indiquer des éléments intéressants pour la majorité des lecteurs.

**Attention**

Ces notes indiquent des points importants qui réclament votre attention.

**ASTUCE**

Celles-ci concernent en général des points de détail et permettent "d'aller plus loin".

**Définition : Exemple (OMG SysML v1.3, p. 152)**

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

Pourquoi parler de "document"?

Parce que j'ignore la version que vous êtes en train de lire. A partir de l'**original**, plusieurs versions ont été générées grâce à **AsciiDoc** :

- pour le web (nous utilisons à l'**IUT de Blagnac** l'excellent **Moodle**) au format **html**
- pour présentation (en amphi par exemple) au format **slidy** ou **deck.js**
- pour impression au format **pdf** (bien que bien sûr nous vous recommandons l'achat du livre)
- pour lire au format **Kindle** (bientôt!)

Utilisation et autres mentions légales

Dernière MAJ : 02/06/2013 - 22:43:14 CEST

Document généré par Jean-Michel Bruel via AsciiDoc (version 8 . 6 . 8) de Stuart Rackham.

La version présentation a été générée en utilisant W3C HTML Slidy © de Dave Raggett, amélioré par Jean-Michel Inglebert. Pour l'instant ce document est libre d'utilisation et géré par la Licence Creative Commons.  licence Creative Commons Paternité - Partage à l'Identique 3.0 non transposé.

N'hésitez pas à m'envoyer vos remarques en tout genre en m'écrivant à bruel@irit.fr.

1.4 Méthode pour cet ouvrage

C'est à l'heure du commencement qu'il faut tout particulièrement veiller à ce que les équilibres soient précis.

— Princesse Irulan *Extrait du Manuel de Muad'Dib*

Mon approche pédagogique repose sur quelques principes, que j'ai essayé de mettre en oeuvre dans cet ouvrage :

La répétition

Par exemple certains diagrammes sont abordés plusieurs fois (comme le diagramme paramétrique). Le lecteur pourra avoir une impression de redite par moment. Sauf erreur de ma part (toujours possible!), c'est volontaire. En général les répétitions vont en niveau de précision, de détails et de complexité croissant. Ces répétitions sont limitées dans la version livre de cet ouvrage (car toute longueur inutile a un coût dans ce cas).

L'illustration

Dans la mesure du possible, j'essaye de donner des exemples aux principes énoncés. Vous trouverez donc plus d'exemples que de définitions.

Le référencement

Les définitions ou autres affirmations sont tirées d'ouvrages de référence généralement citées.

La "carte de base"

J'aime réaliser une "carte"¹ qui sert à "placer" les différents concepts abordés. Il me semble que cela permet aux étudiants de raccrocher les nouveaux concepts aux précédents.

Aucune connaissance particulière d'**UML** n'est nécessaire, même si j'y fais référence à plusieurs endroits pour les étudiants qui connaissent cette notation (quasiment enseignée partout maintenant comme langage de modélisation). Il s'agit d'un parti pris prenant en compte plusieurs points :

- La plupart des ingénieurs systèmes ne connaissent pas **UML**.
- Les étudiants de STI2D ne connaissent pas **UML** et sont pourtant formés à **SysML**.
- Ceux qui connaissent **UML** auront sûrement plaisir à retrouver les bases.

1. voir aussi le concept de *Mind Maps*.

Chapitre 2

C'est quoi SysML?

Si vous ne deviez lire qu'un seul chapitre, voilà ce qu'il faudrait retenir.

2.1 Fiche d'identité

- Date de naissance non officielle : 2001!
- Première spécification adoptée à l'**OMG** : 19 septembre 2007
- Version actuelle : **1.3** (12/06/2012)
- Paternité : **OMG/UML + INCOSE**
- Auteurs principaux :
 - Conrad Bock
 - Cris Kobryn
 - Sanford Friedenthal

2.2 SysML, c'est...

Un ensemble de 9 types de diagrammes

- Diagrammes structuraux
- Diagrammes de définition de blocs (**bdd**)
- Diagrammes internes de blocs (**ibd**)
- Diagrammes paramétriques (**par**)

- Diagrammes de packages (pkg)
- Diagrammes comportementaux
 - Diagrammes de séquence (sd)
 - Diagrammes d'activité (act)
 - Diagrammes de cas d'utilisation (uc)
 - Diagrammes d'états (stm)
- Diagramme d'exigence (req)

Un profil UML

C'est à dire une **extension** de cette notation, un ensemble de nouveaux concepts et éléments qui sont définis à partir des éléments de base d'**UML**. Un exemple : le **bloc** qui n'est qu'une redéfinition de la **classe**.

Une notation

Une notation de plus en plus enseignée et connue et qui servira donc de plus en plus de **référence** à la modélisation des systèmes.

2.3 SysML, ce n'est pas...

Une méthode

En effet, contrairement à ce que beaucoup pensent en l'abordant, **SysML** ne propose pas de démarche particulière de développement de système. C'est à la fois sa force (votre méthode existante pourra continuer à être utilisée) comme sa faiblesse car cette absence de guide méthodologique fait souvent défaut à son utilisation.

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 reste une simple notation qu'il convient d'utiliser avec des outils et des démarches associées.

Un raton laveur

C'est juste pour voir ceux qui suivent.

2.4 Références et liens utiles

Vous trouverez en fin d'ouvrage un ensemble de liens utiles (cf. Chapitre 27) et de références (cf. **Bibliographie**). Sinon, vous pouvez aussi constatez les évolutions de l'intérêt pour **SysML** grâce aux "trends". Voici les dernières tendances au moment où nous écrivons ces lignes¹ :

1. Ou bien utilisez les URLs comme <http://www.google.fr/trends/explore#q=sysml>.

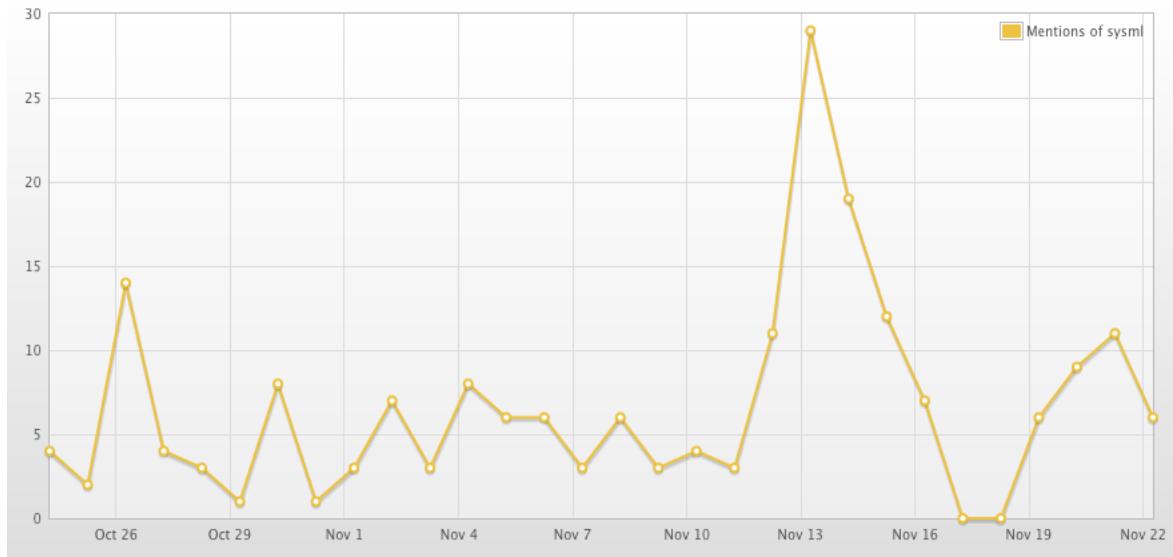


FIGURE 2.1 – Trends : Twitter

**Note**

On y voit les journées SysML 2012 (Toulouse et Mulhouse).

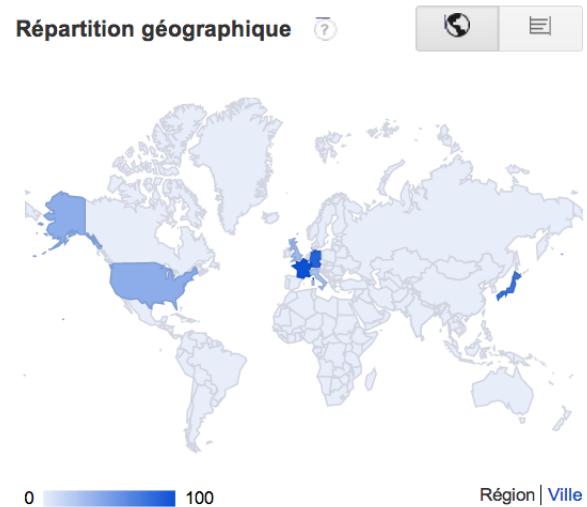
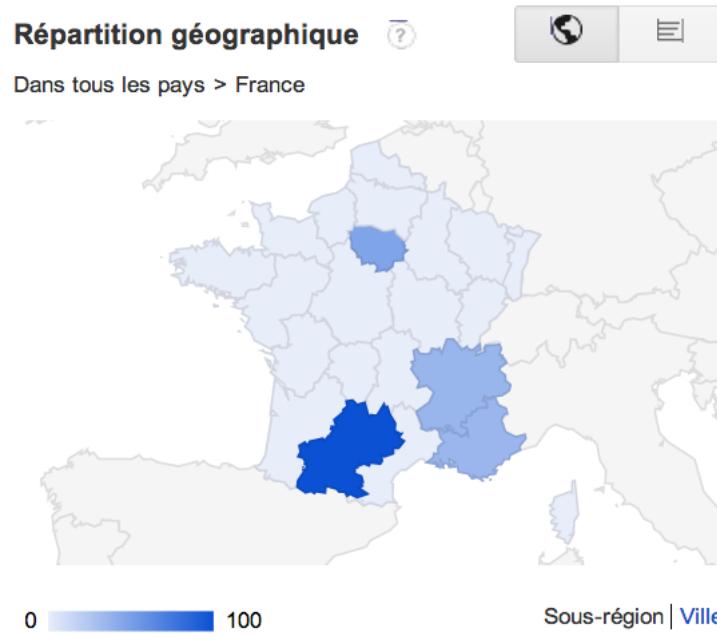
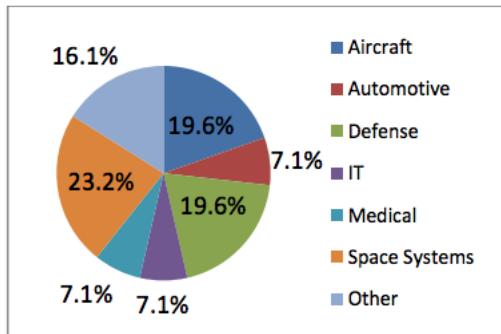
FIGURE 2.2 – *Trends* : Google (Carte)

FIGURE 2.3 – Effet SysML-France?

À noter également que l'**OMG** a réalisé en 2009 une enquête sur l'utilisation de **SysML**² dont voici deux extraits :



Application of SysML by System Type

FIGURE 2.4 – Principaux domaines d'utilisation

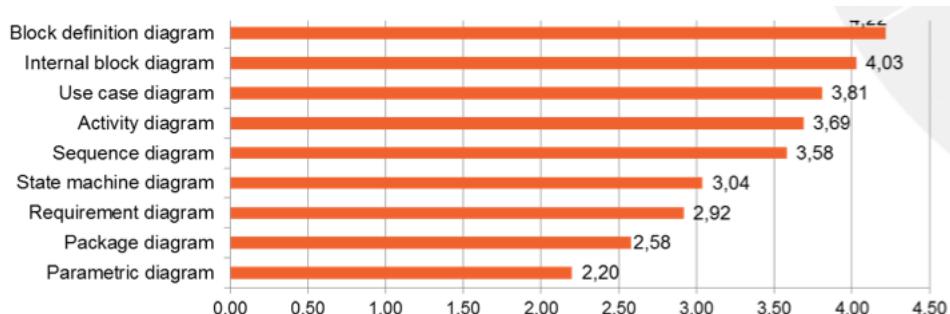


FIGURE 2.5 – Diagrammes les plus utilisés

2. http://www.omg.sysml.org/SysML_2009_RFI_Response_Summary-bone-cloutier.pdf

Chapitre 3

A propos du Bac STI2D

Si vous utilisez cet ouvrage dans le cadre du bac STI2D (Sciences et Technologies de l'Industrie et du Développement Durable) qui a introduit depuis 2011 la notation **SysML** au programme, nous donnons ici des conseils sur l'utilisation de ce cours¹.

L'objectif en STI2D n'est pas de former des spécialistes de **SysML** mais de permettre à tous d'apprendre une notation pour la modélisation de système qui se veut universelle. Il ne faut donc pas viser la complétude ou même demander trop de détails. La logique de la démarche de modélisation et l'importance de la communication devront primer.



Note

A l'heure où nous écrivons ces lignes, il est également prévu de l'enseigner en classe prépa dès 2013.

3.1 Utilisation pratique

Cet ouvrage est tout à fait utilisable dans le cadre des cours dispensés en STI2D. Seul un sous-ensemble des diagrammes de **SysML** a été retenu. Les élèves et les enseignants du bac STI2D pourront trouver dans ce document des éléments utiles sur ces diagrammes :

- diagramme des exigences (cf. Chapitre 18)
- diagramme des cas d'utilisation (cf. Section 20.2)
- diagramme de séquences (cf. Section 20.4)

1. Je remercie au passage les collègues de Lycée rencontrés dans le cadre de **SysML-France** pour nos fructueuses discussions à ce sujet.

- diagramme d'états (cf. Section 20.5)
- diagramme de définition de blocs (cf. Section 19.3)
- diagramme de blocs internes (cf. Section 19.4)

Ces 6 diagrammes sont tous traités dans cet ouvrage à un niveau qui devrait correspondre à l'utilisation qui en est faite en STI2D.

Ces 6 diagrammes servent trois objectifs principaux inscrits au programme et dont les élèves pourront également trouver des éléments de réflexion :

- Modélisation des exigences (cf. Chapitre 9)
- Modélisation structurelle (cf. Chapitre 10)
- Modélisation comportementale (cf. Chapitre 11)

3.2 Pour aller plus loin

Les questions et exercices de fin de chapitres de la partie notation seront peut-être d'un niveau plus avancé pour servir véritablement d'exercices, mais pourront amener à une réflexion encadrée par l'enseignant.

Un blog récent recense les supports en liens avec STI2D : <http://www.scoop.it/t/formation-sysml-sti2d>.

Chapitre 4

Un exemple fil rouge

L'exemple de système qui sera modélisé tout au long de ce livre en guise d'exemple est l'exemple d'un système de gestion et de supervision de crise. Les détails sont donnés en annexe (cf. [Annexes](#)).

Il existe un certain nombre d'autres exemples complets :

- Le radio-réveil de [Pascal Roques \[Roques2010\]](#), un exemple simpliste mais didactique qui a le mérite d'être déjà connu des modeleurs [UML](#) qui ont lu ses livres.
- Le distilleur de [\[FMS\]](#), un exemple très complet et lui aussi très connu car beaucoup utilisé dans les tutoriels issus de l'[OMG](#).
- Le pacemaker de [\[SeeBook2012\]¹](#), un exemple très récent et dont l'avantage est d'être traité selon plusieurs approches différentes et complémentaires ([SysML](#), [AADL](#) et [MARTE](#)).

Les exemples complets ont le mérite de donner une vue d'ensemble des liens qui peuvent exister entre les différents diagrammes. On peut y voir comment ces diagrammes sont complémentaires les uns des autres. Ils sont en général plus réalistes que les diagrammes utilisés pour illustrer tel ou tel concept de la notation.

4.1 Enoncé



ASTUCE

Cette étude de cas est tirée de [Kienzle2010](#)

1. Nous avons réalisé le chapitre d'introduction à [SysML](#) de cet ouvrage.

Le système **CMS** (*crash management system*) est un système distribué de gestion d'accidents qui est responsable de la coordination et de la communication entre un coordinateur présent dans une caserne de pompiers (appelé FSC pour *Fire Station Coordinator*) et un autre présent dans un poste de police (appelé PSC pour *Police Station Coordinator*) afin de gérer une crise dans un délai raisonnable.

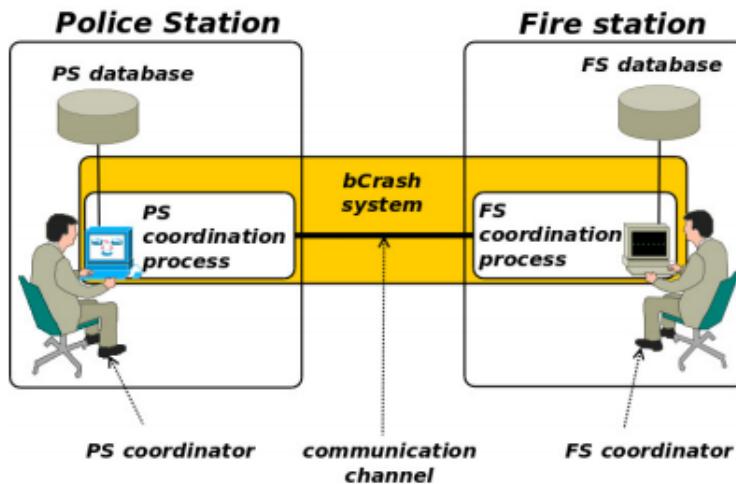


FIGURE 4.1 – Le système CMS

La communication interne entre les membres de la police (y compris le PSC) est en dehors du domaine qui nous intéresse ici (la gestion de crise). La même hypothèse s'applique aux communications internes ou avec des acteurs externes côté pompiers (y compris le FSC). Les informations concernant la crise ainsi que tout ce qui a trait aux tâches des coordinateurs sont mises à jour et maintenues pendant et après la crise.

Il n'existe pas de base de données centrale; caserne de pompiers et police ayant leur base de données respectives distinctes et seulement accessible aux autres à travers le système CMS. Chaque processus de coordination est donc en charge de l'ajout et la mise à jour des informations dans sa base de données respective.

CMS commence à fonctionner au moment où une crise donnée a été détectée et déclarée à la fois à la caserne de pompiers et au poste de police.

Toutes les caractéristiques des différents acteurs sont détaillées ci-dessous.

Coordinateur des pompiers (FSC)

Un FSC maintient le contrôle sur une situation de crise en communiquant avec le coordinateur du poste de police (PSC) ainsi que les pompiers.

Pour atteindre ses objectifs, les responsabilités d'un FSC sont les suivantes :

- de déterminer où, quand et combien de camions de pompiers à envoyer,
- de communiquer avec le PSC pour se présenter,
- de garder le PSC informé en ce qui concerne la crise,
- de proposer une stratégie pour traiter la crise,
- parvenir à un accord avec le PSC sur la façon de procéder,
- de recevoir des mises à jour concernant la crise côté pompiers, et
- de rassembler et de diffuser des informations actualisées aux pompiers.

Pompiers

Un pompier agit sur ordres reçus du FSC et des fait des rapports au FSC. Par ailleurs, un pompier communique avec d'autres pompiers, des victimes et des témoins.

Les responsabilités d'un pompier sont les suivantes :

- recevoir des demandes pour aller/revenir sur les lieux de la crise,
- signaler sa position au FSC,
- signaler les conditions de la crise au FSC et à tous les pompiers, et
- communiquer avec les victimes et les témoins.

Coordinateur du poste de police (PSC)

Pour atteindre ses objectifs, un PSC effectue les mêmes activités que le FSC.

Agents de police

Les agents de police agissent sur ordres reçus du PSC. En outre, un agent de police communique avec d'autres policiers, des victimes et des témoins. Pour atteindre ses objectifs, un agent de police exerce les mêmes activités qu'un pompier en termes de communication avec son coordinateur.

Victimes (de la crise)

Une victime a été touchée par la crise et peut communiquer avec les policiers et les pompiers. Les responsabilités d'une victime sont :

- de fournir des informations liées à la crise
- de suivre les instructions de pompiers et de policiers.

Témoins (de la crise)

Un témoin a observé la crise et communique avec les policiers et les pompiers. Les responsabilités d'un témoin sont les suivantes :

- de fournir des informations aux pompiers et les policiers, et
- de suivre les instructions de pompiers et de policiers.

Informations complémentaires

Pour enrichir vos modèles, vous pouvez incorporer certains des besoins non-fonctionnels décrits ci-dessous.

Le système de gestion de crise doit montrer les suivants propriétés non-fonctionnelles :

Disponibilité

- Le système doit être en opération 24 heures par jour, tous les jours, sans interruption, pendant toute l'année, sauf pour un temps d'arrêt maximal de 2 heures tous les 30 jours pour la maintenance.
- Le système doit récupérer dans un maximum de 30 secondes en cas d'échec.
- L'entretien doit être reportée ou interrompue quand une crise est imminente sans affecter les capacités des systèmes.

Fiabilité

- Le système ne doit pas dépasser un taux d'échec maximum de 0,001%.
- Les unités mobiles sont en mesure de communiquer avec d'autres unités sur le site crise et le centre de contrôle, indépendamment des conditions d'emplacement, le terrain et la météo.

Persistance

- Le système doit permettre le stockage, la mise à jour et l'accès à l'information suivante sur les crises : type de crise, l'emplacement de la crise, rapport d'un témoin, emplacement du témoin, les données concernant les témoins, la durée de la crise, les ressources déployées, les victimes civiles, les stratégies utilisées, l'emplacement des équipes de secours sur la crise, journal des communications, et des décisions.
- Le système doit permettre le stockage, la mise à jour et l'accès à l'information suivante sur les ressources disponibles et déployés (à la fois en interne et en externe) : type de ressources (humaines ou équipement), la capacité, l'équipe de sauvetage, l'emplacement, l'heure estimée d'arrivée (ETA) sur le site crise.

- Le système doit permettre le stockage, la mise à jour et l'accès à l'information suivante sur les stratégies de sortie de crise : type de crise, étapes pour résoudre la crise, la configuration des missions nécessaires, des liens vers d'autres stratégies, applications aux crises précédentes, et le taux de succès.

Sécurité

- Le système doit définir des politiques d'accès pour les différentes catégories d'utilisateurs. La politique d'accès doit décrire les éléments et informations de chaque acteur peut ajouter, accéder et mettre à jour les informations.
- Le système doit authentifier les utilisateurs sur la base des politiques d'accès lors de leur premier accès pour accéder aux éléments d'informations. Si un utilisateur reste inactif pendant 30 minutes ou plus, le système doit les obliger à se ré-authentifier.
- Toutes les communications dans le système doit utiliser des canaux sécurisés conformes avec le cryptage AES-128 standard.

Mobilité

- Les ressources de secours doivent pouvoir accéder à l'information sur les mouvements.
- Le système fournit des informations de localisation utiles pour économiser les ressources.
- Les ressources de secours doivent communiquer leur emplacement au centre de contrôle.
- Le système doit avoir accès à des cartes détaillées, des données de terrain et les conditions météorologiques pour l'emplacement de crise et les routes qui y mènent.

Sécurité

- Le système doit surveiller les émissions provenant du site crise pour déterminer les distances de fonctionnement sûres pour les ressources de sauvetage.
- Le système doit surveiller les conditions météorologiques et le terrain sur le site de crise pour assurer la sécurité et le retrait des moyens de secours, et l'évacuation de civils, et les victimes.
- Le système détermine un périmètre pour le site crise pour assurer la sécurité des civils et l'évacuation des victimes à une distance sûre.
- Le système surveille les activités criminelles pour assurer la sécurité des moyens de secours, des civils et des blessés.
- La sécurité du personnel de secours doit avoir la priorité absolue pour le système.

Adaptabilité

- Le système doit recommander ou solliciter des ressources alternatives en cas d'indisponibilité ou l'insuffisance de ressources appropriées.
- Le système doit être en mesure d'utiliser les canaux de communication de rechange en cas d'indisponibilité ou l'insuffisance des moyens existants.
- Le système doit être en mesure de maintenir une communication efficace dans les zones de perturbation ou de bruit élevé sur le site crise.

Précision

- Le système doit avoir accès aux données de la carte, le terrain et les conditions météorologiques avec une précision de 99%.
- Le système doit fournir des informations à jour pour sauver les ressources.
- Le système doit enregistrer des données sur la réception sans modifications.
- La communication entre le système et les ressources de sauvetage doit avoir un facteur de détérioration maximum de 0,0001 pour 1000 km

Deuxième partie

Partie 2 : Ingénierie système

Chapitre 5

Introduction

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

TABLE 5.1: La carte de base

	Requirements	Structure	Comportement	Transverse
Organisation				
Analyse				
Conception				
Implémentation				

5.1 Points de vue

Dans un axe horizontal, j'ai différencié quatre grands points de vue :

Requirements

Les exigences et leur prises en compte sont un éléments critique pour le succès du développement du système. Sans explorer l'ensemble des activités d'ingénierie système (ce qui nécessiterait tout un volume du type de Chapitre 18) nous insisterons beaucoup sur cet aspect qui est souvent à l'origine de l'intérêt de **SysML**.

Structure

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 **SysML** qui pose le moins de problème aux débutants.

Comportement

Le comportement d'un système est du point de vue de l'utilisateur final beaucoup plus important que la structure elle-même. C'est la partie qu'il est la plus à même d'exprimer, de comprendre (vos modèles) et de valider.

Transverse

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

5.2 Phase de développement

Dans un axe vertical, j'ai différencié quatre 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.).

Analyse

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.

Conception

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.).

Implémentation

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

Chapitre 6

Différence avec l'ingénierie logicielle

Enseignant en informatique, je me retrouve souvent à enseigner **SysML** à des informaticiens. D'où ce petit exposé sur mon opinion de la différence entre les deux "mondes".

6.1 Une ingénierie plus ancienne

Que ce soit généralement en terme de cycle de développement ou historiquement, l'Ingénierie Système arrive avant l'Ingénierie Logicielle. Les ingénieurs systèmes ont donc une longue expérience et des pratiques bien ancrées.

6.2 Des systèmes plus complexes

On parle de système complexe lorsque l'on a affaire à :

- un ensemble d'éléments humains et matériels en relation avec :
 - de nombreux éléments technologiques (Informatique, Hydraulique, Electronique, ...)
 - intégrés pour fournir des services (finalité du système) en fonction de leur environnement
 - interagissant entre eux et avec leur environnement



FIGURE 6.1 – Un système complexe

On parle aussi de **Système de systèmes** quand un système :

- doit gérer les interactions entre ses parties (ou composantes)
- assure un comportement prévu à l'avance
- gère les comportements (de l'environnement) inattendus

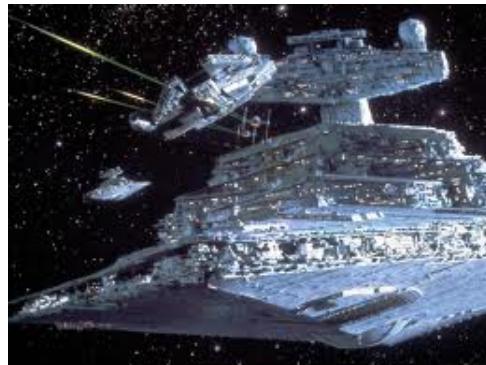


FIGURE 6.2 – Un système de système

6.3 Différents types d'analyse

Toute la question que l'Ingénierie Système cherche à résoudre est : comment passer des exigences au système de la façon la plus efficace possible.

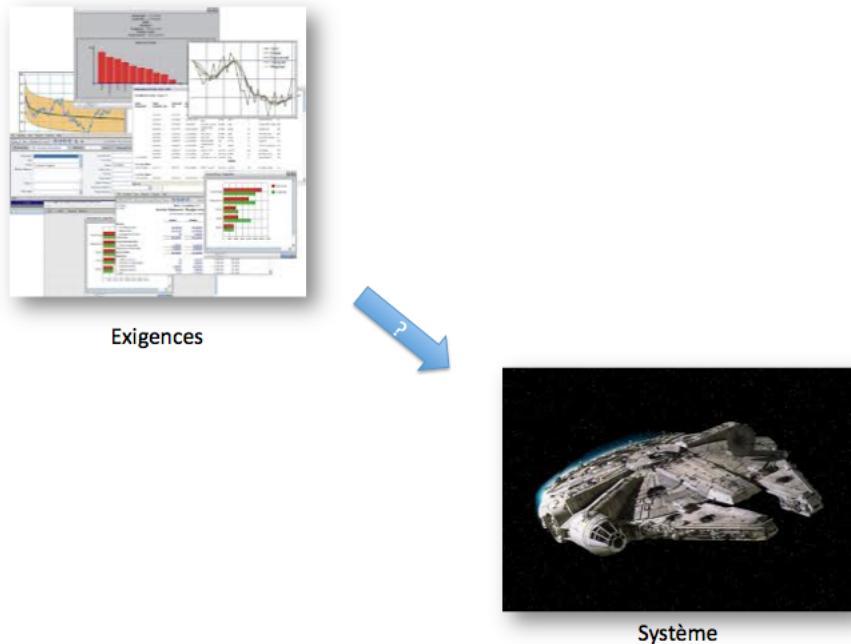


FIGURE 6.3 – Des exigences au système

Pour cela l'Ingénierie Système est découpée en plusieurs analyses, chacune avec un but bien particulier :

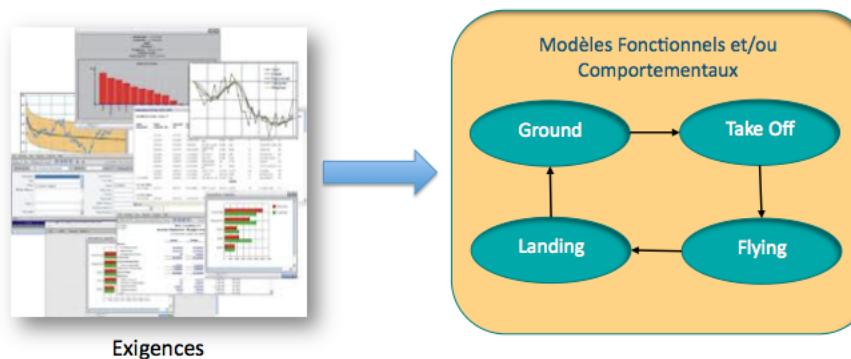


FIGURE 6.4 – Analyse Fonctionnelle et/ou Comportementale

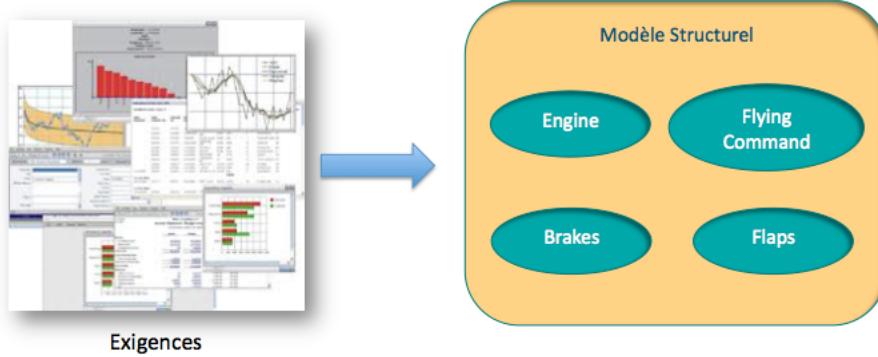


FIGURE 6.5 – Analyse Structurelle

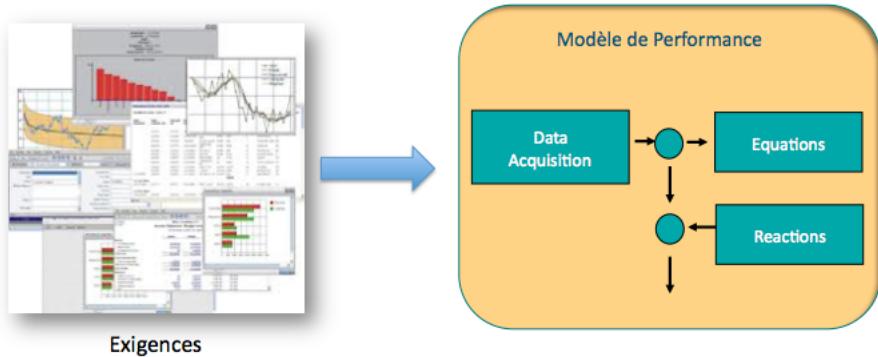


FIGURE 6.6 – Analyse de performance

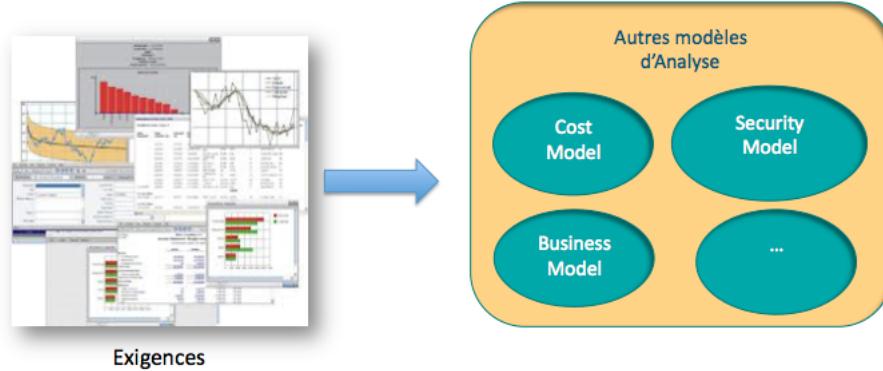


FIGURE 6.7 – Analyses spécifiques

Pour arriver à combler le gap entre le système à développer et ses spécifications.

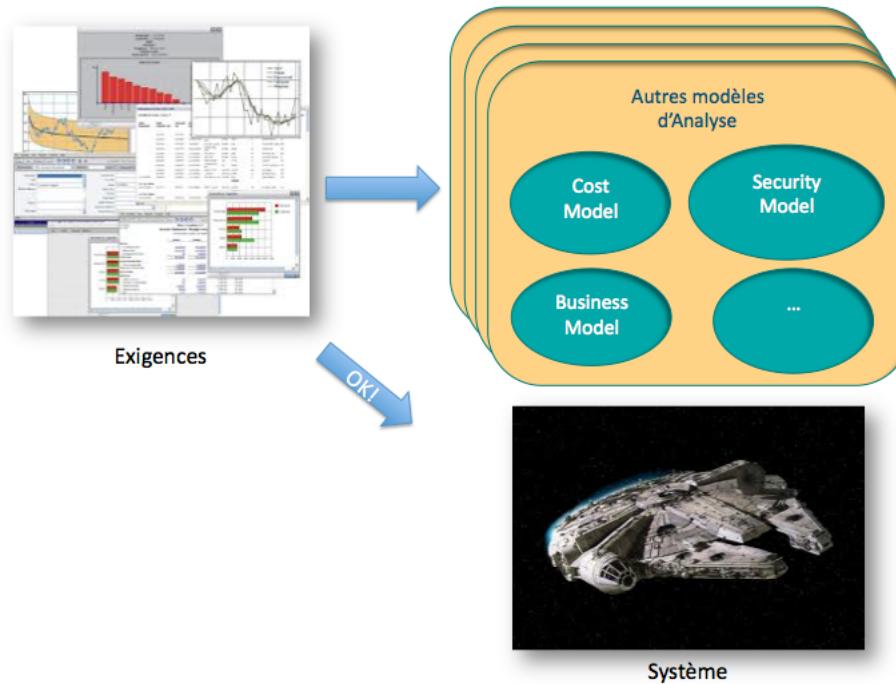


FIGURE 6.8 – Des exigences au système

Chapitre 7

Normes et standards

Il existe un grand nombre de standards en Ingénierie Système. Cette section fera (bientôt) une revue de ces différents standards et organismes et de leur utilisation (IEEE, EIA, ISO, certification, NASA, INCOSE, ...).

Enfin, citons un rapport de 2010, le [Rapport Potier](#), qui présente l'état des logiciels embarqués et qui sera utiles à ceux qui s'intéressent aux verrous technologiques liés à ce domaine.

L'Ingénierie Système génère beaucoup de documentation. Les processus de certification (par exemple dans l'aéronautique) sont encore basés sur des documents textuels.

Chapitre 8

Des documents aux modèles

Vue la complexité grandissante des systèmes, petit à petit cette ingénierie tente de passer d'une ingénierie **centrée documents** à une ingénierie **centrée modèles**. D'où l'importance de se poser la question des notations et langages pour réaliser et communiquer avec ces modèles (cf. Chapitre 13).

Chapitre 9

Les exigences

L'ingénierie des exigences est d'une importance capitale en Ingénierie Système. Nous renvoyons pour l'instant le lecteur au cours de Master qui précède ce cours.



FIGURE 9.1 – 300 corps de métiers sont parfois présents sur un chantier

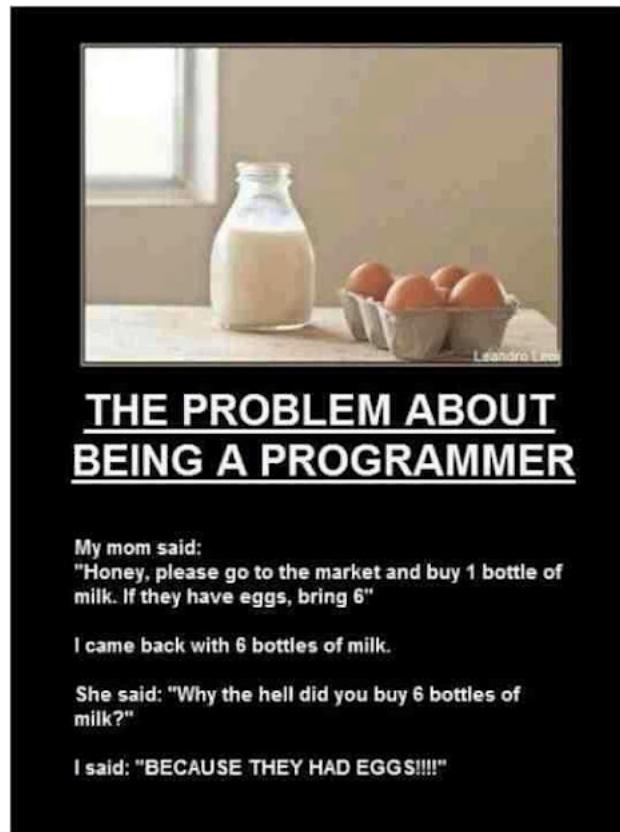


FIGURE 9.2 – Humour (taken from [here](#))

Chapitre 10

L'architecture du système

Liens avec AADL, ...

Chapitre 11

Le comportement du système

Liens avec la V&V

Chapitre 12

Méthodes et démarches

Everything should be made as simple as possible, but not simpler.

— Albert Einstein

SysML n'est pas une méthode. En effet aucune démarche n'est imposée pour l'utilisation des diagrammes, l'ordre logique dans lesquels il vaut mieux les réaliser, etc. La spécification ne porte que sur la notation elle-même. D'où le pluriel dans le titre de cette section : il existe presque autant de méthodes que d'entreprise développant des systèmes. Nous nous contenterons de donner ici quelques heuristiques (cf. Chapitre 24 pour la présentation de quelques méthodes bien identifiées) :

Exemple 12.1 Approche itérative

Un diagramme ne doit pas être considéré comme définitif. Il peut être complété alors que l'on traite un autre aspect de la modélisation (exemple classique : ajout d'un nouveau bloc lors de la réalisation d'un diagramme de séquence). Quelque soit la démarche adoptée elle doit être **itérative** et permettre de revenir sur les premières étapes.

Exemple 12.2 Niveau d'abstraction

Bien intégrer les niveaux d'abstraction dans votre démarche. SysML possède certaines constructions pour formaliser cet aspect (Packages par exemple). Nous matérialisons cet aspect par la partie verticale de la matrice (cf. Tableau 5.1).

Exemple 12.3 Tous les diagrammes ne sont pas utiles

N'essayez pas de réaliser tous les diagrammes possibles pour votre système. Réalisez uniquement ceux qui sont utiles à votre cas particulier.

Troisième partie

Partie 3 : La notation SysML

Chapitre 13

Pourquoi une nouvelle notation

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)*

Il existe une notation qui se veut "unifiée" pour les modèles : **UML**. Néanmoins cette notation est peu adaptée pour l'Ingénierie Système :

- UML 1.x était complètement inadaptée :
 - Principalement pour les systèmes d'information
 - Peu de liens entre les diagrammes
 - Peu de liens entre les modèles et les exigences
- UML 2.x n'est pas beaucoup mieux si ce n'est :
 - Implication des ingénieurs systèmes pour sa définition
 - Introduction du diagramme de structure composite

En conclusion **UML** est une bonne base :

- Standard *De facto* en génie logiciel
- Fournit beaucoup de concepts utiles pour décrire des systèmes (même complexes)
- Stable et extensible (grâce notamment au mécanisme de *profile*)
- Beaucoup d'outils disponibles

Mais...

- Manque de certains concepts clés d'Ingénierie Système
- Vocabulaire beaucoup trop « software » pour être utilisé par les ingénieurs systèmes (concept de classe ou d'héritage par exemple)
- Trop de diagrammes (13 sortes)

Chapitre 14

Introduction à SysML

14.1 Fiche d'identité

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

- Date de naissance non officielle : 2001!
- Première spécification adoptée à l'**OMG** : 19 septembre 2007
- Version actuelle : **1.3** (12/06/2012)
- Paternité : **OMG/UML + INCOSE**
- Auteurs principaux :
 - Conrad Bock
 - Cris Kobryn
 - Sanford Friedenthal

14.2 Différence avec UML

La figure [suivante](#), tirée de la spécification, résume bien les liens entre **SysML** et **UML**, à savoir que **SysML** reprend une partie seulement des concepts d'**UML** (appelée **UML4SysML**) en y ajoutant des concepts nouveaux.

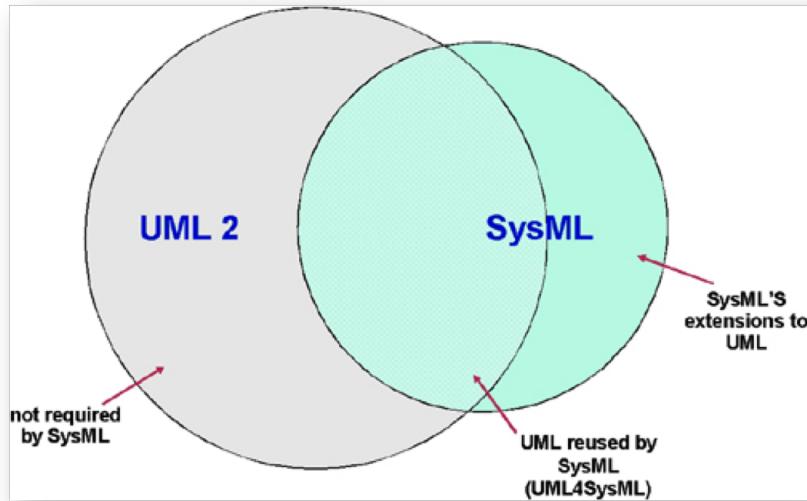


FIGURE 14.1 – Liens entre UML et SysML

14.3 Qui est "derrière"?

Industrie

American Systems, BAE Systems, Boeing, 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, INCOSE, Georgia Institute of Technology, AFIS, ...



ASTUCE

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

14.4 Organisation des différents diagrammes

SysML propose de couvrir la modélisation d'un système en 9 diagrammes. Ces diagrammes couvrent les aspects structurels et comportementaux du système ainsi que les exigences. Le diagramme suivant présente cette organisation en faisant au passage le lien avec ceux d'UML :

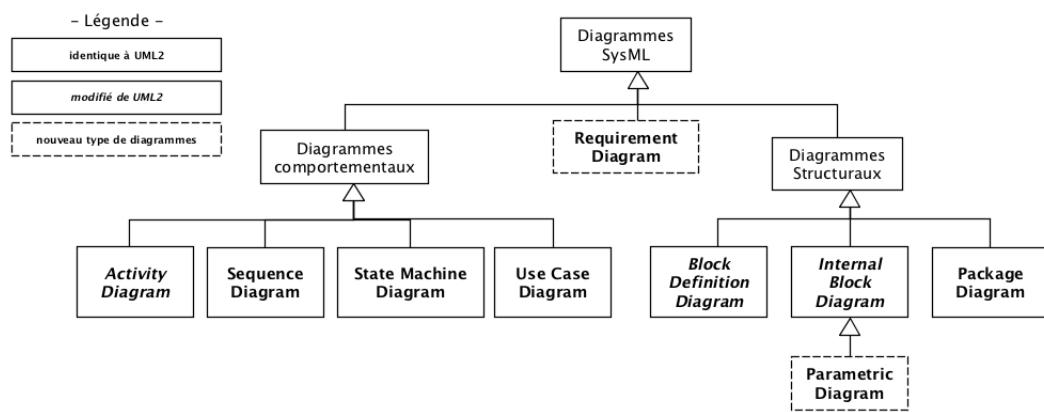


FIGURE 14.2 – Les 9 diagrammes SysML et leur lien avec UML

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

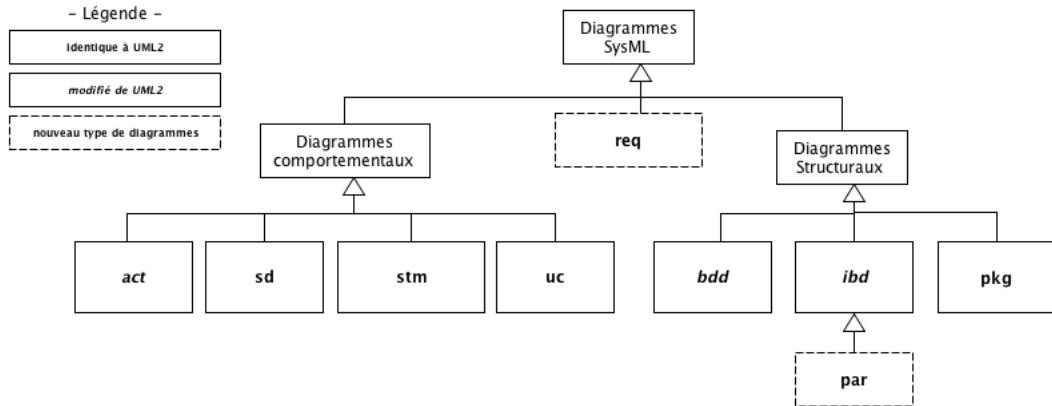


FIGURE 14.3 – Version abrégée des diagrammes

**Définition : Types de diagrammes (OMG SysML v1.3, p. 170)**

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

14.5 Différence entre modèle et dessin

SysML n'est pas une palette de dessins et d'éléments de base servant à faire des diagrammes. Il existe une représentation graphique des éléments modélisés en SysML. Elle 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 SysML¹, mais d'utiliser des outils dédiés (cf. Chapitre 15).

Pour ceux qui cherchent à étudier un diagramme en particulier voici un plan de cette section (nous utilisons ici le "plan" vu lors de l'introduction de la Tableau 5.1) :

TABLE 14.1: Organisation

	Requirements, cf. Chapitre 18	Structure, cf. Chapitre 19	Comportement, cf. Chapitre 20	Transverse, cf. Chapitre 21
Organisation	pkg	pkg, bdd	pkg	

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

TABLE 14.1: (continued)

	Requirements, cf. Chapitre 18	Structure, cf. Chapitre 19	Comportement, cf. Chapitre 20	Transverse, cf. Chapitre 21
Analyse, Conception, Implémentation²	req	bdd, ibd, sd, par	uc, sd, stm, act	par

2. En fonction du niveau de détail.

Chapitre 15

Outils SysML

Il existe un certain nombre d'outils permettant de réaliser des modèles SysML. Voici une liste non exhaustive :

- [TopCased](#)
- [Papyrus](#)
- [Artisan](#)
- [Rhapsody](#)
- [Modelio](#)
- ...

Vous trouverez sur Internet des comparatifs et des avis à jour sur les outils.

Ce que je voudrai souligner ici c'est l'importance du modèle comme "dépôt" (je préfère le terme anglais de *repository*) d'éléments de base en relation les uns avec les autres. C'est toute la différence entre le dessin et le modèle.



Attention

Attention toutefois à ne pas confondre ce que vous permet (ou pas) de faire l'outil et la notation elle-même. Les fabricants ont parfois pris des libertés ou bien n'ont pas complètement implémenté toutes les subtilités de la notation.

Dans le cadre de notre exemple fil rouge, nous utilisons l'outil [TOPCASED](#).

Chapitre 16

Cadre pour les diagrammes

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

- Chaque diagramme **SysML** représente un élément de modélisation
- Chaque diagramme **SysML** doit être inclus dans un cadre (*Diagram Frame*)
- L'entête du cadre, appelé **cartouche**, indique les informations sur le diagramme : le type de diagramme (req, act, bdd, ibd, stm, etc.); le type d'élément (*package*, *block*, *activity*, etc.); le nom de l'élément et le nom du diagramme ou de la vue.

Dans l'exemple ci-dessous, le diagramme "Context_Overview" est un *Block Definition Diagram* (type *bdd*) qui représente un *package*, nommé "Context".

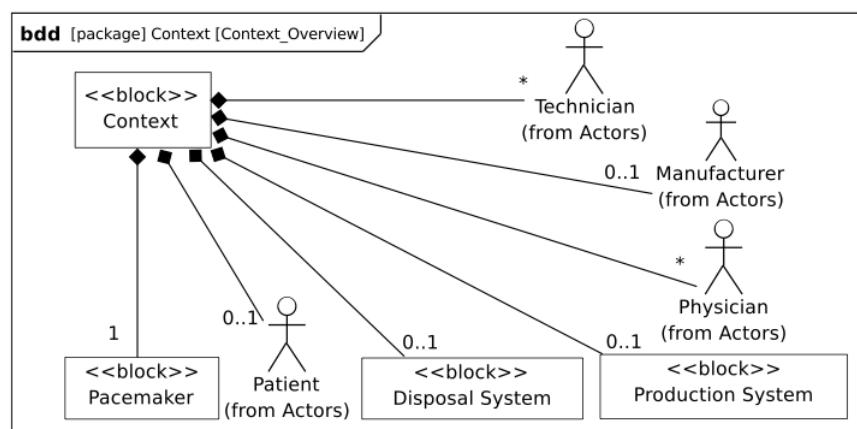


FIGURE 16.1 – Exemple de diagramme SysML

Chapitre 17

Organisation

17.1 Fondements

On abordera :

- Le *Package Diagram*
- Les différent types de *packages*
- Les organisations possibles
- La notion de *Namespaces*
- Les *Dependencies*

17.2 Le *Package Diagram*

Le diagramme de paquetage permet de représenter l'organisation des modèles en paquetages.

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 [17.5](#)).

Les modèles peuvent être organisés selon toutes sortes de considération (cf. Section [17.4](#)) :

- hiérarchie "système" (e.g., entreprise, système, composant)
- types de diagrammes (e.g., besoins, structure, comportements)
- par points de vue
- etc.

17.3 Les différents types de *packages*

Il existe plusieurs types de *package* :

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



ASTUCE

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

17.4 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.

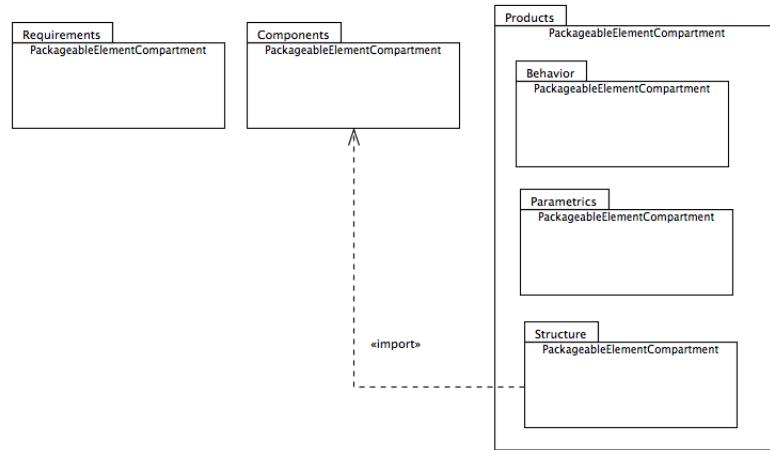


FIGURE 17.1 – Exemple d’organisation simple

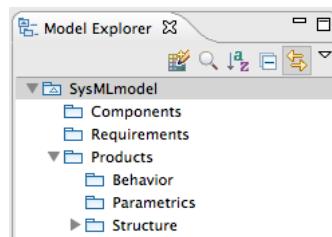


FIGURE 17.2 – Représentation de cette organisation dans un outil

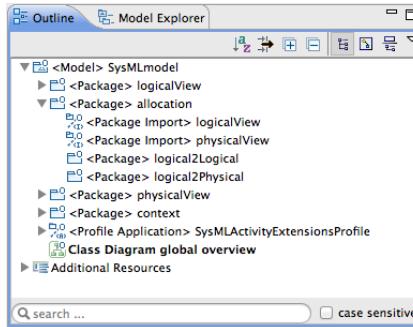
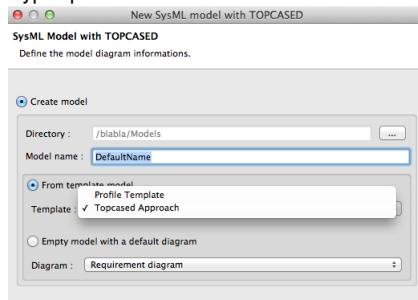


FIGURE 17.3 – Un autre exemple d’organisation

ASTUCE

L’outil **TOPCASED** propose, lors de la création d’un premier modèle, de créer une organisation “type” par défaut.



17.5 La notion de *Namespaces*

Un *package* permet de créer un espace de nommage pour tous les éléments qu’il contient. 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 v1.3, p. 23)**

The package defines a namespace for the packageable elements.

Pour éviter toute ambiguïté, 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`).

**ASTUCE**

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).

17.6 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 raffinement (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 *block* la plupart du temps),
représentée par le stéréotype <<allocate>>

17.7 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 17.1: Organisation

	Requirements	Structure	Comportement	Transverse
Organisation	package	package	package	dependences
...				

17.8 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) ?

Chapitre 18

Les exigences

18.1 Fondements

On abordera :

- L'organisation des *Requirements*
- Les *Requirements properties*
- Les *Requirements links*
- Les *Requirements Diagrams*
- Les considérations sur la traçabilité
- Annotations des *Requirements*
- Les *Use Case Diagrams* (scénarios)



ASTUCE

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 ([\[Somerville1997\]](#)) ou le guide de l'AFIS ([\[REQ2012\]](#)).

18.2 L'organisation des *Requirements*

Différents types d'organisation

Comme nous l'avons vu pour 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.

Tableaux de Requirements

Les *requirements* sont généralement stockés dans des tableaux (feuilles excel le plus souvent!).

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 18.1 – Exemples tableau d'exigences

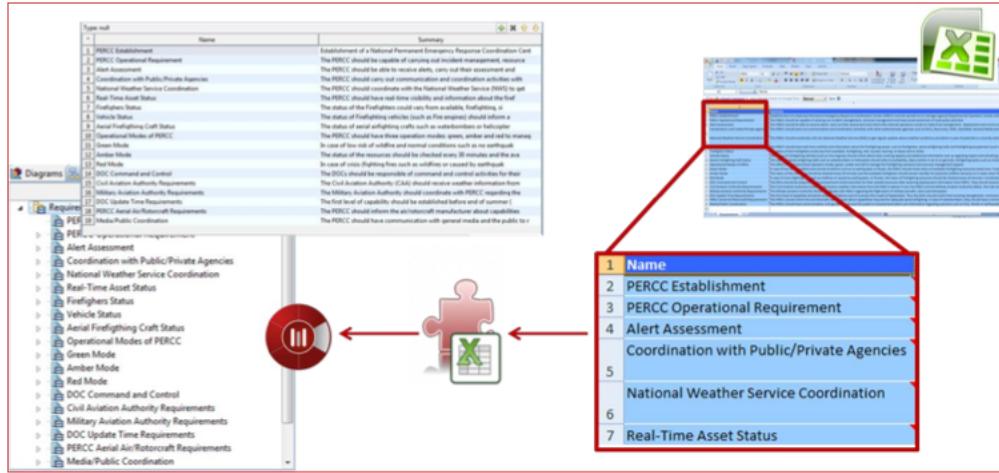


FIGURE 18.2 – Import Modelio de tableau d'exigences

18.3 Les *Requirements properties*

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

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

18.4 Les *Requirements links*

Les principales relations entre *requirement* sont :

Containment

pour décrire la décomposition d'une exigence en plusieurs sous-exigences ($\oplus -$)

Refinement

pour décrire un ajout de précision (<<refine>>)

Derivation

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

**ASTUCE**

Lorsqu'un cas d'utilisation possède plusieurs cas <<refine>> qui pointent vers lui, on considère que ces différents cas sont des options possibles de raffinement (cf. Chapitre 28).

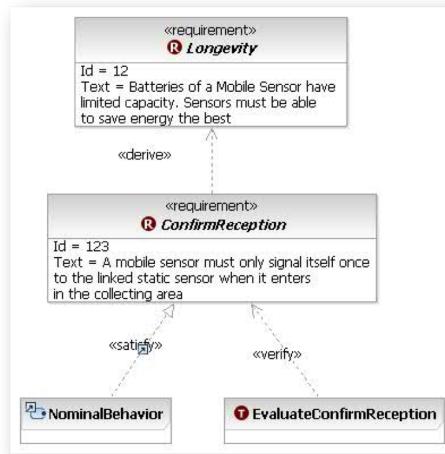


FIGURE 18.3 – Exemples de relations entre exigences

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](#).

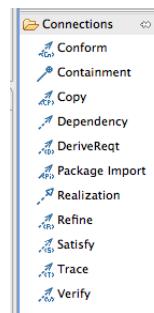


FIGURE 18.4 – Relations liées au *requirements* dans TOPCASED

18.5 Les *Requirements Diagrams*

Voici un exemple de `req` un peu plus étoffé, tiré de <http://www.uml-sysml.org/sysml> (voir aussi Figure 18.6) :

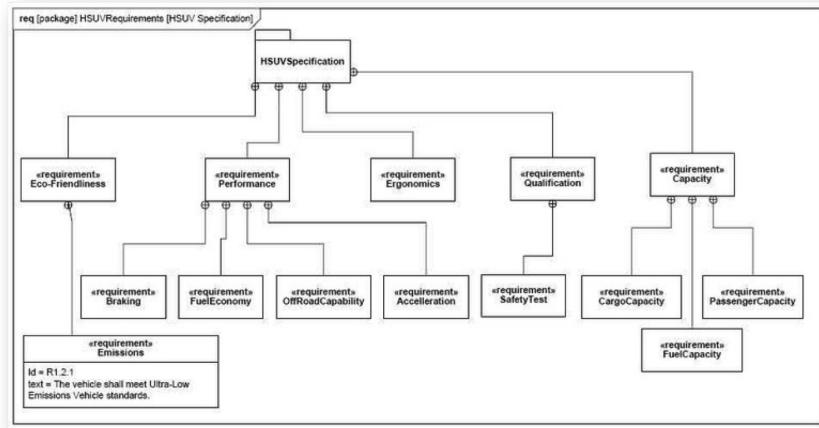


FIGURE 18.5 – Exemples de composition d'exigences

18.6 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 Chapitre 21.

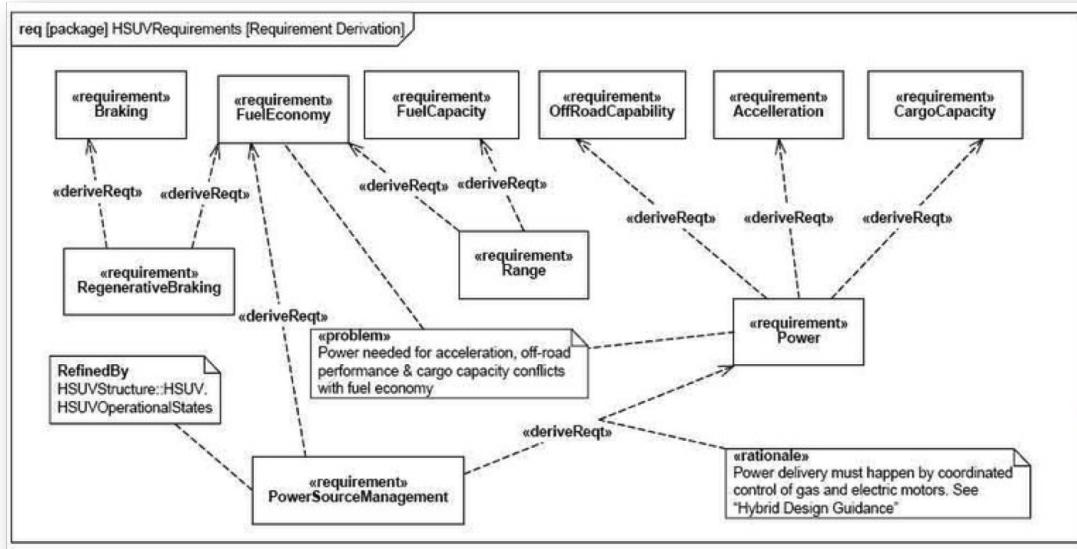


Note

En général chaque *requirement* devrait être relié à au moins un *use case* (et vice-versa!).

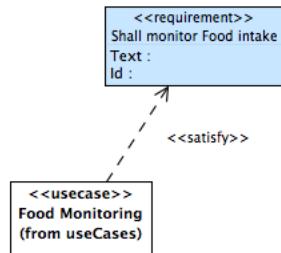
18.7 Annotations des *Requirements*

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

FIGURE 18.6 – Exemples de *rationale* et *problem*

18.8 Les *Use Case Diagrams* (scénarios)

Bien que nous traitions les cas d'utilisation dans la partie [comportement](#), nous les abordons ici du fait de leur proximité avec les *requirements*.

FIGURE 18.7 – Exemple de lien entre *use case* et *requirements*

Ce diagramme est exactement identique à celui d'[UML](#).

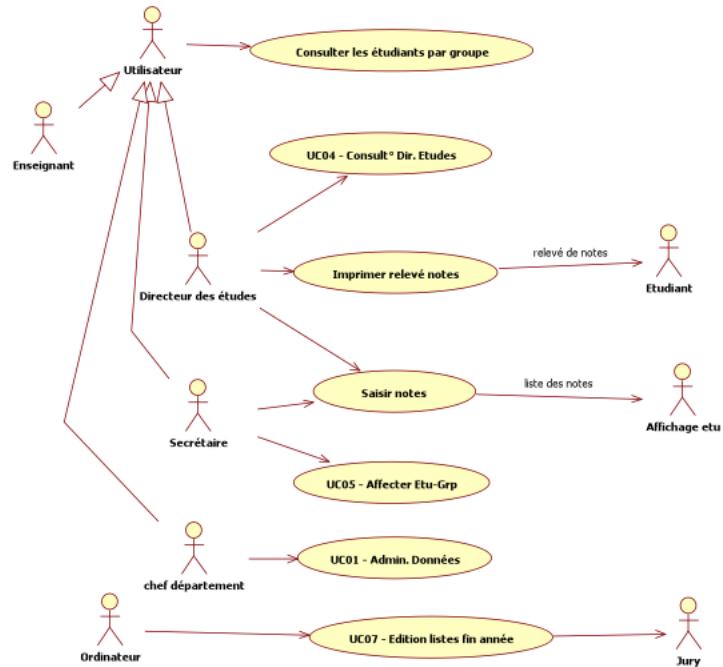


FIGURE 18.8 – Exemple de diagramme des cas d'utilisation

**ASTUCE**

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.

18.9 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 18.1: Déclinaison des Exigences

	Requirements	Structure	Comportement	Transverse
Organisation	$\oplus - <<\text{deriv eReqt}>>$			
Analyse	$<<\text{satisf y}>>, <<\text{refine}>>$	$<<\text{satisf y}>>$ entre reqs et UC	$<<\text{refine}>>$	
Conception	$<<\text{allocat e}>>$			
Implémentation	$<<\text{satisf y}>>, <<\text{verify}>>$			

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 18.9).

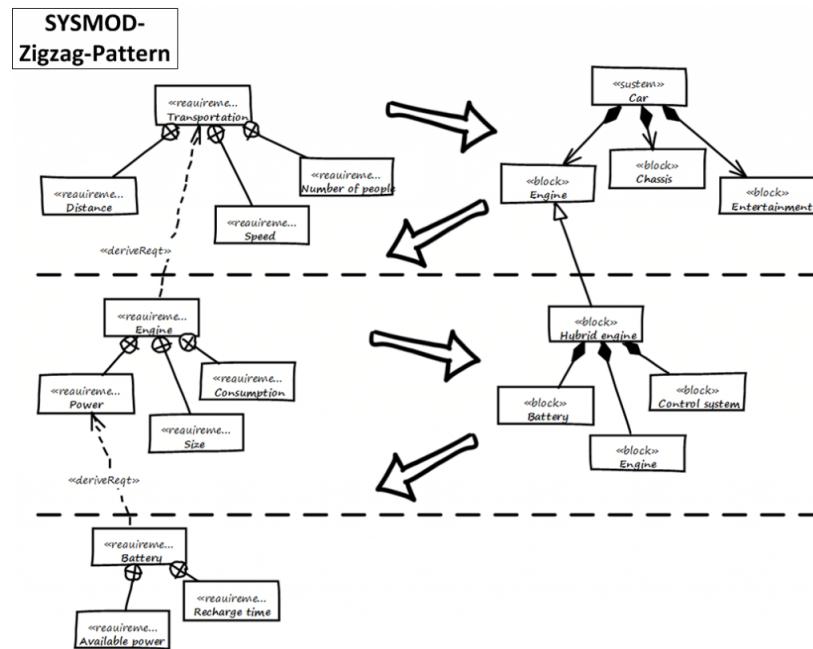


FIGURE 18.9 – Exemple de démarche (SYSMOD Zigzag pattern)

18.10 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**?

Chapitre 19

L'architecture du système

19.1 Fondements

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 séquence système)

19.2 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 avons abordé cette organisation (cf. Section 17.2).

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 19.1)
- un diagramme décrivant les éléments internes principaux du système, décrit dans un *internal block diagram*

19.3 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.

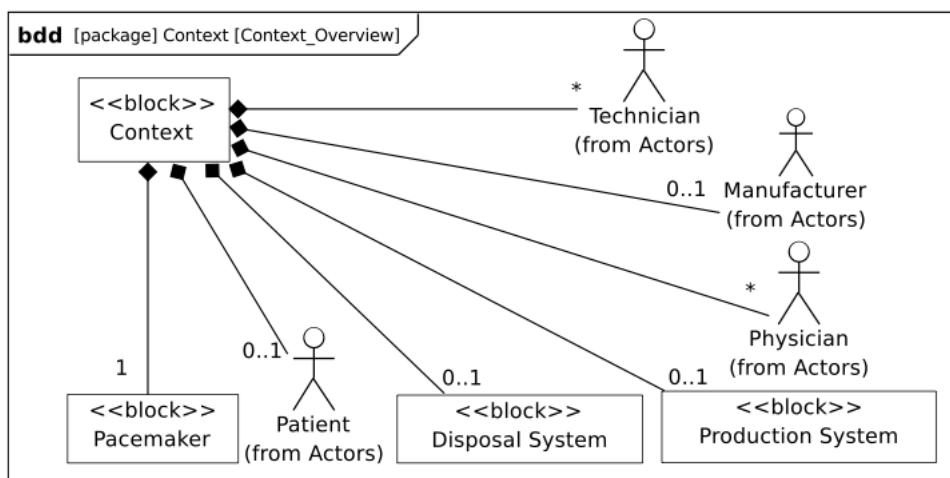


FIGURE 19.1 – bdd du système dans son environnement

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

Properties

Équivalent UML des propriétés (e.g., attributs).

Operations

Les méthodes supportées par les instances du bloc.

Constraints

Les contraintes (cf. Figure 19.2)

Allocations

Les allocations (cf. Chapitre 21)

Requirements

Les exigences liées à ce bloc.

User defined

On peut définir ses propres compartiments.

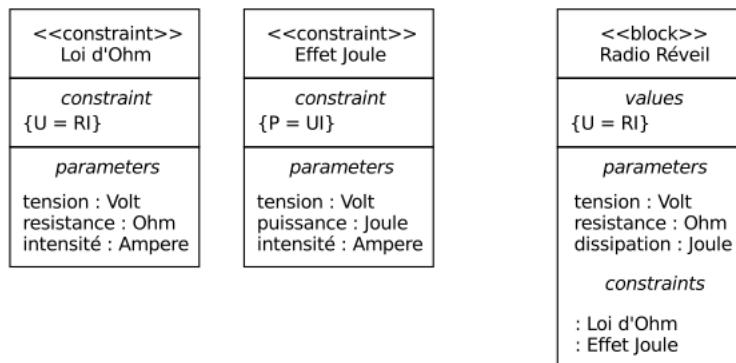


FIGURE 19.2 – Exemple de définition de contraintes

Propriétés

On peut différencier 4 types de propriétés d'un bloc :

value properties

Des caractéristiques (quantifiables), aussi appelées simplement *values*

parts

Les éléments qui composent le bloc (cf. Section 19.4)

references

Les éléments auquel le bloc a accès (via des associations ou des agrégations)

constraint properties

Les contraintes que doivent respecter les propriétés (nous les verrons plus en détail, cf. Section 19.5).



Note

Les *values* sont ce qui se rapproche le plus des attributs de classes UML.

Value Types

Pour associer un type aux valeurs, **SysML** propose de définir des *Value Types*.

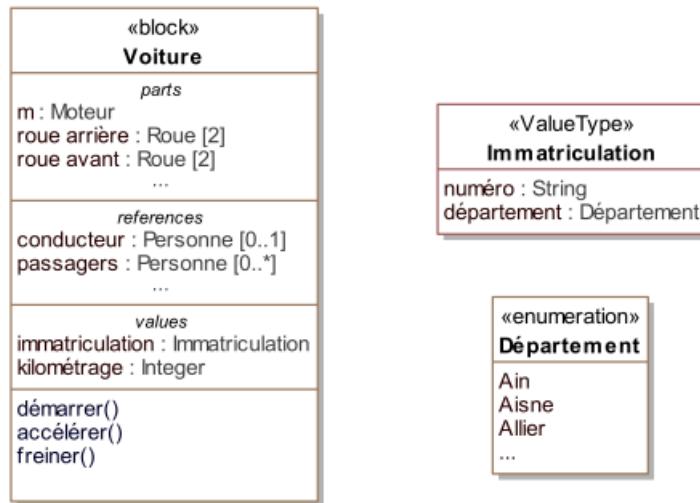


FIGURE 19.3 – Définition de *Value Types*.

Associations entre blocs

Il existe deux types de relations entre blocs :

- l’association (y compris l’agrégation et la composition)
- la généralisation/spécialisation

Ces deux types de relations, bien connues en **UML**, permettent de matérialiser les liens qui existent entre les éléments du système. Avant d’aborder les associations, il est important de différencier la description d’éléments structurels sous la forme d’un bloc (au travers d’un bdd par exemple) et ces éléments pris individuellement. Ces derniers sont des **instances** individuelles du même bloc. Cette notion, très présente dans les approches orientées objets est souvent plus ardue à appréhender pour les ingénieurs systèmes. Il faut bien comprendre que la modélisation d’un bloc consiste à représenter l’ensemble des éléments qui caractérisent tout une série d’objets (des moteurs, des pompes, des données, etc.). Il serait fastidieux de les représenter tous (individuellement), et c’est donc leur “signature” que l’on représente. C’est pour cela qu’un bloc n’est pas un élément physique, mais simplement sa représentation, tandis qu’une instance de ce bloc représentera elle cet élément physique. C’est le cas notamment des participants d’un diagramme de séquence ou encore des parties d’un composé, qui sont des instances et non des blocs.

Association

Une **association** est un ensemble de liens permanents existant entre les instances de deux ou plusieurs blocs. On dira qu'une association lie plusieurs blocs ou que les blocs **participent** à l'association.

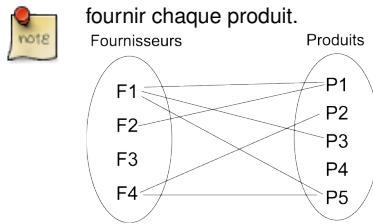
Une association possède plusieurs propriétés :

Dimension d'une association

Nombre de blocs mis en jeu par l'association
(binaire : 2, ternaire : 3, n-aire : n)

Exemple d'association binaire

Soient les bloc Fournisseurs et Produits. On veut indiquer quels sont les produits susceptibles d'être fournis par chaque fournisseur et quels sont les fournisseurs susceptibles de fournir chaque produit.



Nom d'une association

Afin de clarifier les informations, il est important de nommer les associations.

Il existe trois façons de nommer une association :

- un verbe à l'infinitif (e.g., Fournir)
- un verbe conjugué avec un sens de lecture : Fournit > ou < Est fourni par
- un rôle (placé à une extrémité de l'association)

Cardinalité

Indique à combien d'instances minimum et maximum du bloc d'en face est lié toute instance du bloc de départ. Elle est représentée par un couple (M..N).



Attention

Attention, dans une cardinalité M..N, M doit toujours être inférieur ou égal à N. Exemple : 3..10.

Vers le code : que signifie vraiment une association?

En terme de logiciel, une **association** représente une contrainte sur la suite du développement : que ce soit un **code** (en langage orienté objet la plupart du temps) ou une **base de donnée**.

Pour reprendre l'exemple précédent, cela signifie concrètement au niveau d'un code par exemple que depuis une variable `Produits` on doit être capable d'accéder à une variable (correspondante) de type tableau (ou liste, ou ...) de `Fournisseurs`.

Ce qui peut donner en java :

```
public class Produits
{
    //Produits Attributes
    private String idPro;
    private String designation;
    private float poids;

    //Produits Associations
    private List<Fournisseurs> fournisseurs;
    ...
}
```

En terme d'ingénierie système, on utilisera plutôt des associations spécifiques (l'agrégation et la composition).

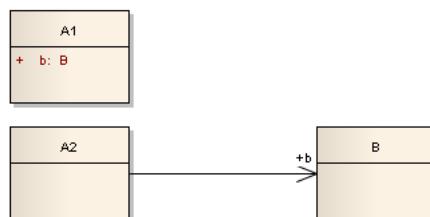


FIGURE 19.4 – Deux façons de représenter une propriété de type B

En terme d'Ingénierie Système, une composition indique que l'élément est une partie intégrante (on parle de *part*) du tout (un composant, comme le moteur d'une voiture par exemple) tandis qu'une agrégation indique que l'élément est une partie "externe" (on parle de *reference*) comme la batterie d'un portable.

Note

Un moyen simple en terme logiciel de déterminer si une association A→B est une association dirigée (navigable dans un sens), une agrégation ou une composition est de raisonner en terme d'implémentation :

- c'est une agrégation si b est initialisé dans le constructeur de A ;
- c'est une composition si il est aussi détruit dans le destructeur de A ;
- c'est une association dirigée simple si aucun des deux cas précédent ne s'applique.

Généralisation/spécialisation

Lorsque plusieurs blocs ont des caractéristiques en communs (propriétés, associations, comportement), il peut être utile de "factoriser" ces éléments en un bloc dont les autres vont "hériter". Quand on réalise ces liens hiérarchiques (on utilise souvent le terme "est un") en partant des blocs différents pour établir un nouveau bloc contenant les points communs on parle de **généralisation**. À l'inverse, quand on constate qu'un bloc possède réellement plusieurs déclinaisons différentes et que l'on crée alors des blocs spécifiques, on parle alors de **spécialisation**.

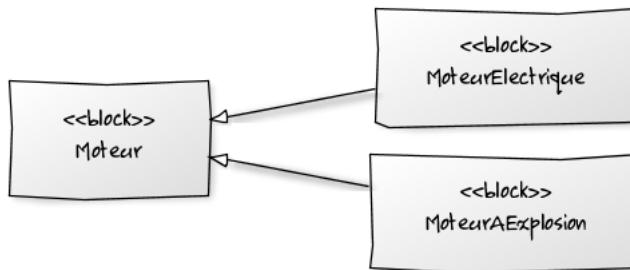


FIGURE 19.5 – Exemple de lien de généralisation/spécialisation

On retrouve cette association entre blocs, mais aussi entre acteurs, cas d'utilisation, etc.

19.4 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).

**Attention**

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 blocs en traits pleins et les *references* comme des blocs en trait pointillés.

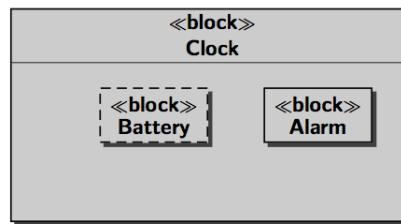
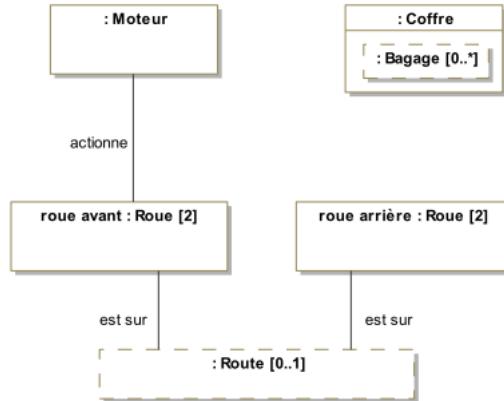


FIGURE 19.6 – Exemple de *Parts*

FIGURE 19.7 – Autre exemple de *Parts*

Ports

Les ports :

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



Note

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.

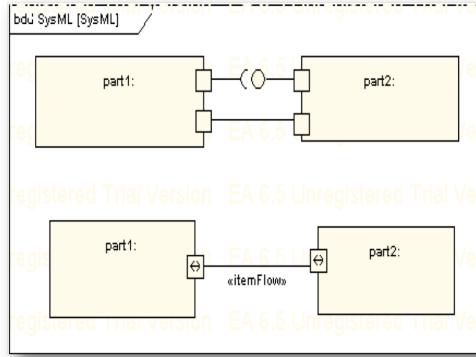


FIGURE 19.8 – Exemples de flots

**Définition : Réutilisation d'exigences (OMG SysML v1.3, p. 57)**

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.

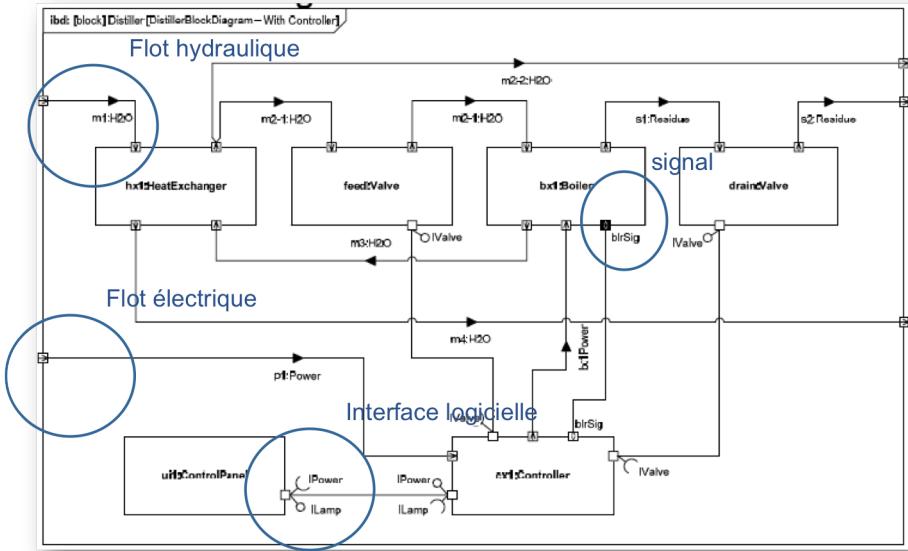


FIGURE 19.9 – Exemples de flots multi-phérique entre ports

Les ports peuvent être de nature classique (comme en UML) et représenter la fourniture ou le besoin de services. Ils peuvent aussi être de nature "flux physique".

Les Flux peuvent être :

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



Note

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.

19.5 Parametric Diagrams

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

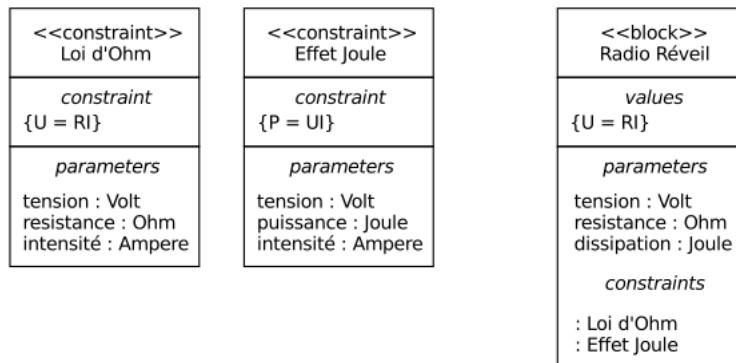


FIGURE 19.10 – Exemple de contraintes



Définition : Réutilisation d'exigences (OMG SysML v1.3, p. 86)

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 de *Internal Block Definition*

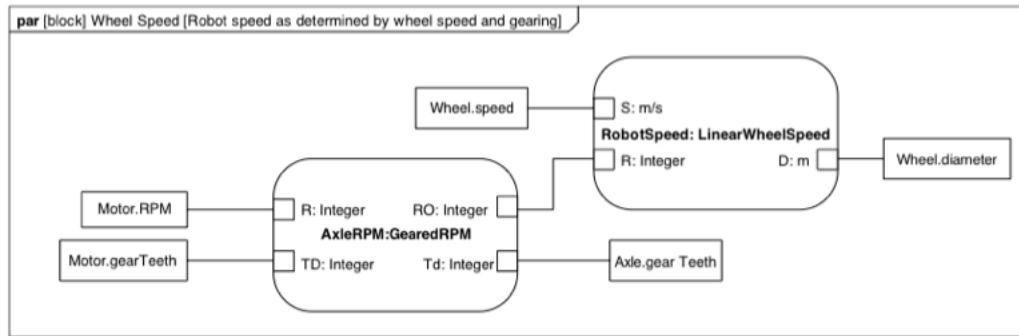


FIGURE 19.11 – Exemple de diagramme paramétrique

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*;

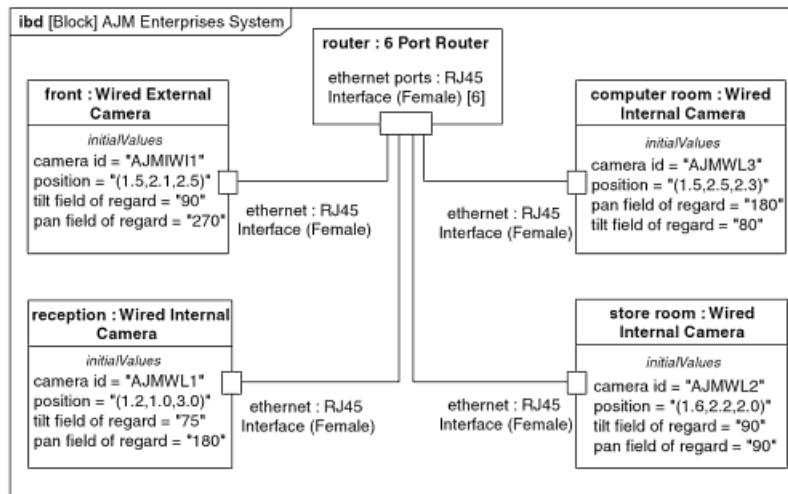


FIGURE 19.12 – Exemple de bloc de configuration

19.6 Diagrammes de séquence système

Les diagrammes de séquence système (DSS) sont des *Sequence Diagrams UML* classiques où seul le système est représenté comme une boîte noire en interaction avec son environnement (les utilisateurs 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 séquence classique (cf. section sur les Section 20.4).

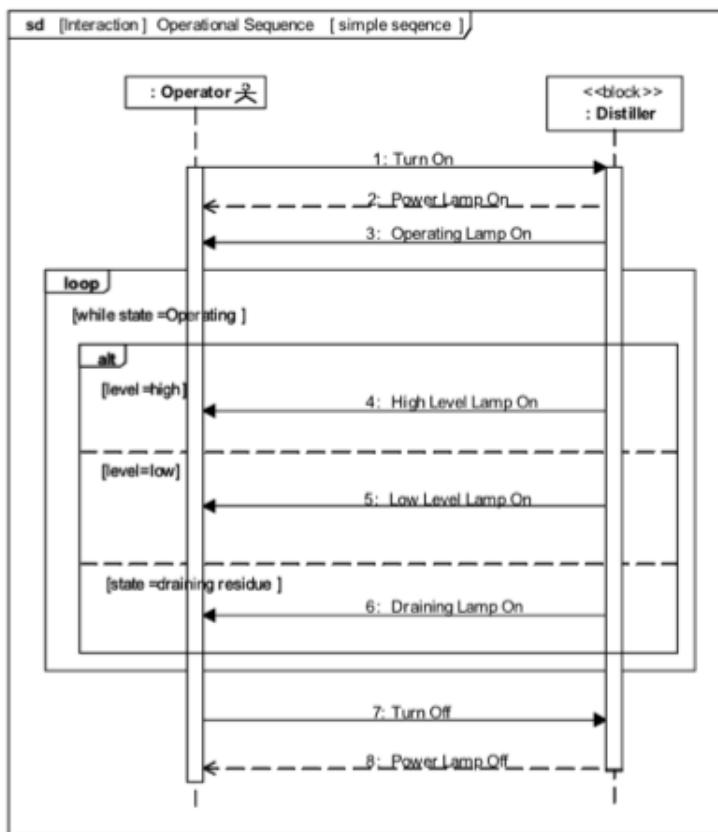


FIGURE 19.13 – Exemples de DSS

19.7 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 19.1: Place des aspects structurels

	Requirements	Structure	Comportement	Transverse
Organisation		package		
Analyse		bdd par		
Conception		bdd par ibd dss		
Implémentation		bdd par ibd dss		

19.8 Questions de révision

1. Quelles sont les différences entre une association dirigée (\rightarrow), 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` ?

Chapitre 20

Le comportement du système

20.1 Fondements

On abordera :

- les *Use Case Diagrams* (scénarios)
- les *Sequence Diagrams*
- les *State Machines*
- les *Activity Diagrams*

20.2 *Use Case Diagrams*

Les éléments de base :

Acteurs

les principaux acteurs (leur rôle) qui participent (on parle parfois d'acteurs principaux) ou qui bénéficient (on parle alors d'acteurs secondaires) du système.

Cas d'utilisation

représente un ensemble d'actions réalisées par le système intéressant pour au moins un acteur

Association

participation d'un acteur à un cas d'utilisation.

Sujet

le domaine étudié (qui peut être une partie seulement de tout le système, pas forcément modélisé dans son ensemble)

**ASTUCE**

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.

20.3 Le Diagramme des Cas d’Utilisation

Le **Diagramme des Cas d’Utilisation** est un diagramme **UML** permettant de représenter :

- les **UC** (*Use Case* ou Cas d’Utilisation)
- les **acteurs** (principaux et secondaires)
- les **relations**
 - entre acteurs et *Use Case*
 - entre *Use Cases*

Cas d’Utilisation (*Use Case*)

Un cas d’utilisation représente un ensemble de **scénarios** que le système doit exécuter pour produire un résultat observable par un **acteur**.

Exemple de cas d’utilisation (UML)

Retrait par carte bancaire

Scénario principal

L’UC démarre lorsque le Guichet Automatique Bancaire (GAB) demande au client son numéro confidentiel après l’introduction de sa CB. Le client entre son code et valide son entrée. Le GAB contrôle la validité du code. Si le code est valide, le GAB autorise le retrait et l’UC se termine.

Scénario alternatif n°1

Le client peut à tout instant annuler l’opération. La carte est éjectée et l’UC se termine.

Exemple de codification de l’UC

UC01 ou RetraitCB (pour Retrait par carte bleue)

Précisions

Un cas d’utilisation peut être précisé par :

- une description textuelle

-
- un ou des diagrammes **UML** (séquence, activité)
-

**Note**

Dans les outils, cette "précision" se manifeste par le fait que l'on "attache" généralement un diagramme de séquence à un cas d'utilisation (clic droit sur un *Use Case* → nouveau scd).

Acteur

Un acteur peut être une personne, un ensemble de personnes, un logiciel, un processus qui interagit avec un ou plusieurs **UC**.

On peut trouver plusieurs types d'acteurs :

- extérieurs au système (cf. actor Figure 20.1)
 - les acteurs principaux
 - les acteurs secondaires
 - exemples de types d'acteurs prédéfinis dans UML :
 - <<utility>>
 - <<process>>
 - <<thread>>
-

**Note**

On peut utiliser des liens de généralisation/spécialisation entre acteurs pour représenter les possibilités pour le spécialisé d'avoir les mêmes prérogatives (notamment en terme d'utilisation du système) que le généralisé.

Relations entre acteurs et *Use Case*

En général, une simple association relie acteurs et *Use Case*. On peut également orienter ces associations en plaçant une direction (flèche vide) au bout de l'association.

Relations entre *Use Case*

Après avoir lister les cas d'utilisation, il est utile de les organiser et de montrer les relations entre eux. Plusieurs relations sont possibles :

Extension (<<extend>>)

Indique que le *Use Case* source est **éventuellement** exécutée en complément du *Use Case* destination (cas particulier, erreur...). Le point précis où l'extension peut se produire est appelé *extension point* (surtout utile quand il existe plusieurs extensions pour un même cas)

Inclusion (<<include>>)

Indique que le *Use Case* est inclus **obligatoirement** dans un autre *Use Case* (notion de sous-fonction par exemple)

Généralisation

Relation entre un *Use Case* général et un autre plus spécialisé qui hérite de ses caractéristiques et en rajoute (différents modes d'utilisation d'un système par exemple, ou encore différents acteurs impliqués)

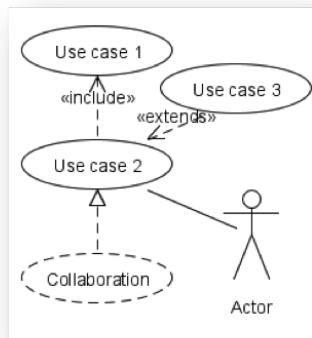


FIGURE 20.1 – Notation dans le diagramme d'UC

**ASTUCE**

On n'utilise généralement <<include>> que dans le cas où le sous-cas d'utilisation est inclus dans plusieurs UC. Si ce n'est pas le cas, il est généralement englobé dans l'UC.

Pour construire un UC (de manière générale)

1. identifier les acteurs
2. identifier les cas d'utilisation
3. structurer en *packages*
4. finaliser les diagrammes de cas d'utilisation (ajouter les relations)

**Note**

Certains méthodologistes (comme T. Wielkins) préconisent de ne pas utiliser les acteurs et les cas d'utilisation (cf. son blog)

20.4 Sequence Diagrams

Généralités

Il permet de :

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

Les éléments qui composent ce diagramme sont :

Participants

les éléments en interaction (des blocs généralement)

Lignes de vie

des lignes verticales qui permettent d'indiquer un départ ou une arrivée d'interaction

Barres d'activation

pour matérialiser quand l'élément est actif

Messages

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

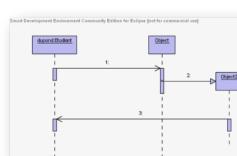


FIGURE 20.2 – Exemple de diagramme de séquence

**Attention**

Les participants (et leur ligne de vie) représentent des instances de blocs (souvent "anonymes").

Notions avancées

On peut également représenter des instructions itératives et conditionnelles au travers de **cadres d'interaction** :

- loop (boucle)
- alt (alternative)
- opt (optionnel)
- par (parallèle)
- region (région critique - un seul *thread* à la fois)

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

FIGURE 20.3 – Exemple d’ algorithme...

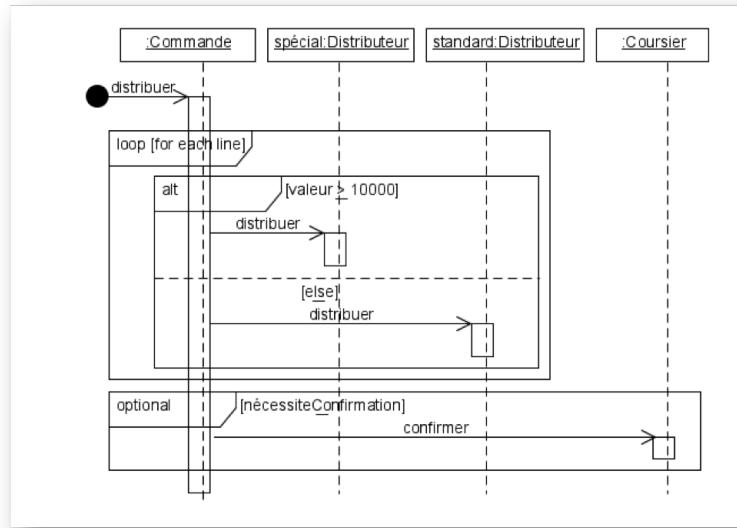


FIGURE 20.4 – Et le diagramme correpondant

Exemple de conceptions

Le diagramme de séquences 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 20.5) et la version "objet" (Figure 20.6).

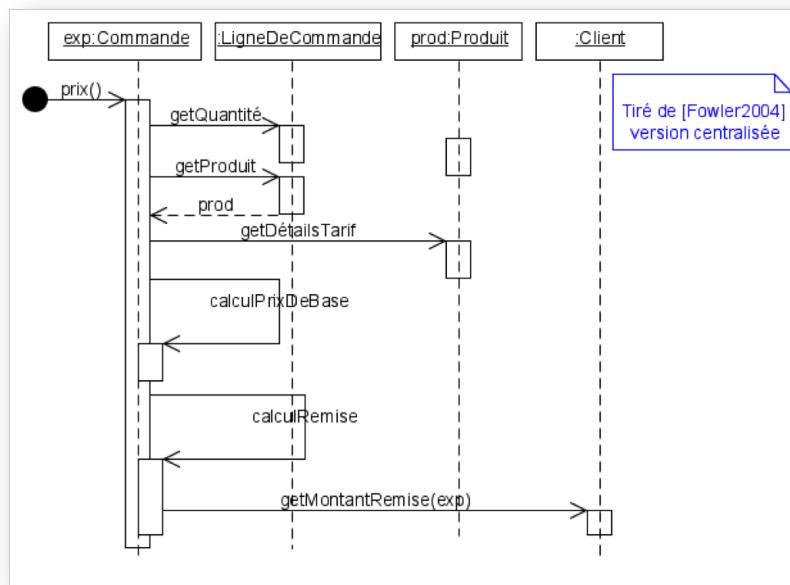


FIGURE 20.5 – Conception "centralisée"

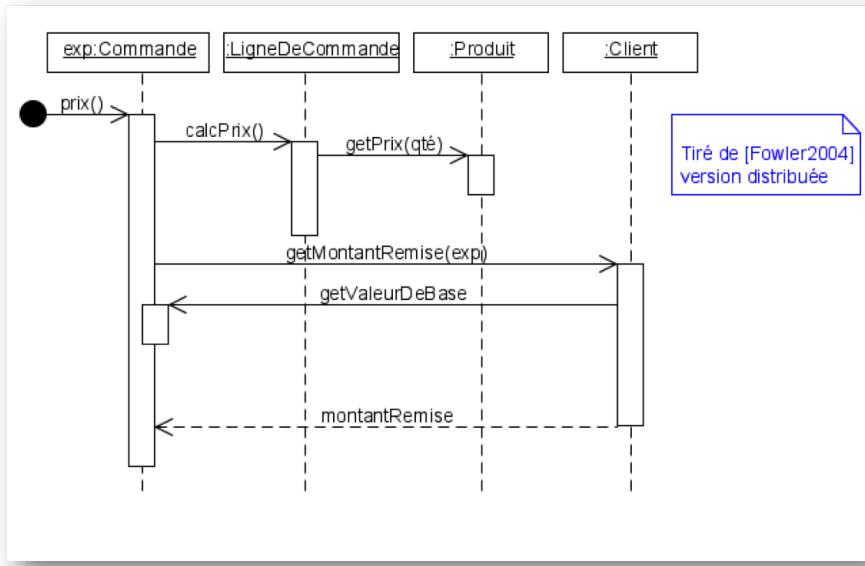


FIGURE 20.6 – Conception "objet"

**Note**

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 20.6).

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.

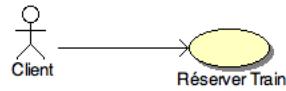


FIGURE 20.7 – Diagramme d’UC

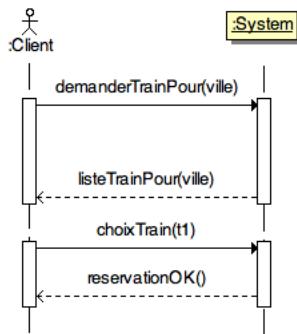


FIGURE 20.8 – Le DSS correspondant

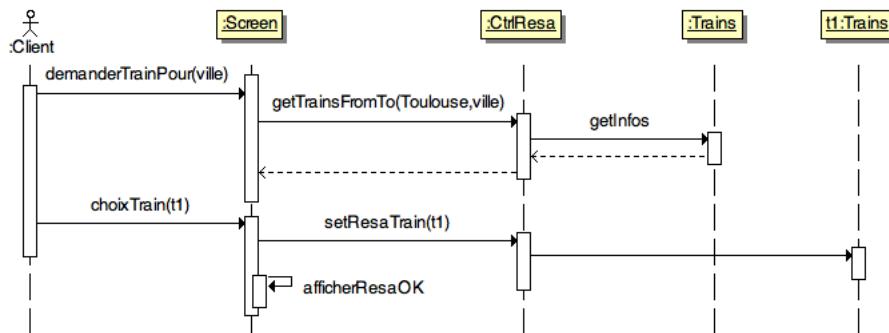


FIGURE 20.9 – Le DS correspondant

20.5 Diagramme d’états

SysML a repris le concept, déjà connu en UML, de machine à états (*State Machines*). Ce diagramme représente les différents états possibles d’un bloc particulier, et comment ce bloc réagit

à des événements en fonction de son état courant (en passant éventuellement dans un nouvel état). Cette réaction (nommée **transition**) possède un événement déclencheur, une condition (garde), un effet et un état cible.

Le diagramme d'états comprend également deux **pseudo-états** :

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

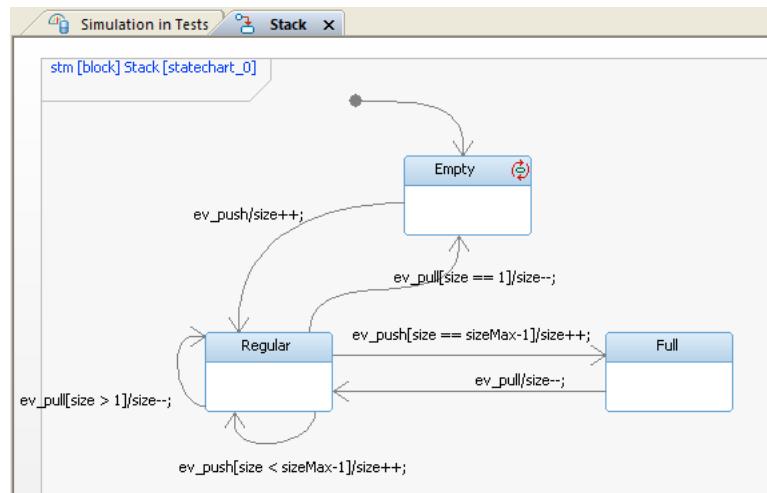


FIGURE 20.10 – Un exemple de diagramme d'état (R,UK)

Lorsqu'un état nécessite lui-même plus de détails, on crée 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'état (composite) à un état pour garder une représentation hiérarchique.

Un diagramme d'état 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.

Il existe encore d'autres concepts avancés que nous ne présenterons pas dans cette introduction car ils sont beaucoup moins utilisés (entry, exit, transition interne, etc.).

20.6 Diagrammes d'activité

Les diagrammes d'activité (*Activity Diagrams*) est utilisé pour représenter les flots de données et de contrôle entre les actions. Il est utilisé pour raffiner en général 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.

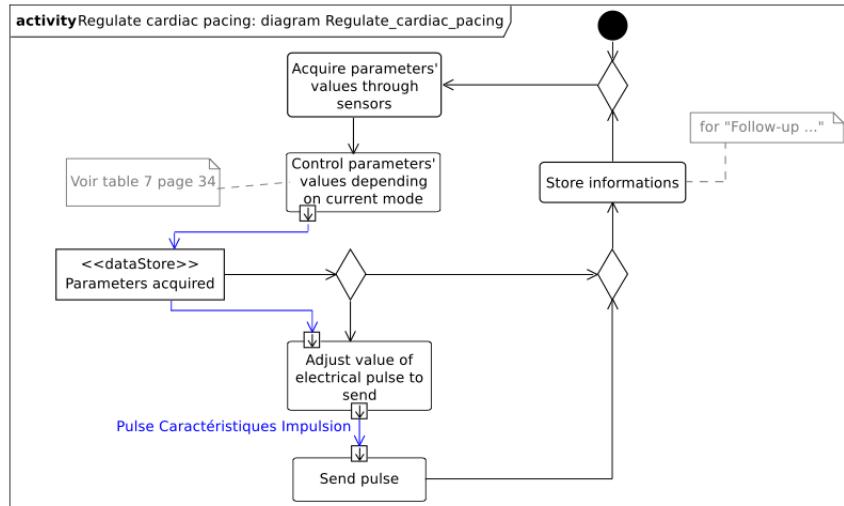


FIGURE 20.11 – Exemple de diagramme d'activité (tiré de [SeeBook2012])

Les éléments de base du diagramme d'activité sont :

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

20.7 Actions

Les actions sont les unités fondamentales pour spécifier les comportements en **SysML**. Une action représente un traitement ou une transformation. Les actions sont contenues dans les activités, qui leur servent alors de contexte.

20.8 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 **flocs 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.

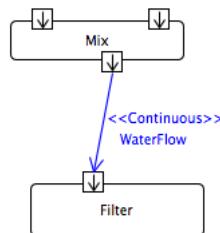


FIGURE 20.12 – Un exemple de flot continu (UK)

Pour permettre la modélisation des **flocs 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, événements, requêtes, etc.). On utilise pour cela des stéréotypes : <<continuous>> et <<discrete>>. Par défaut, un flot est supposé discret.



Définition : FlowProperty (OMG SysML v1.3, p. 63)

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.

20.9 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 20.13) :

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.

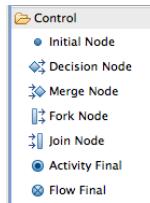


FIGURE 20.13 – Les différents contrôles de flow SysML

20.10 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 20.14 – Exemple de *callBehaviorAction* (UK)

20.11 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. Chapitre 25).

TABLE 20.1: Place du Comportement

	Requirements	Structure	Comportement	Transverse
Organisation			pkg	
Analyse			uc sd	
Conception			dss sd act	
Implémentation			stm	

20.12 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 ?

20.13 Exercices

Diagramme des cas d'utilisation

Placez dans un diagrammes des cas d'utilisation les différents acteurs et cas correspondant à l'étude de cas suivante (en indiquant les relations) :

Pour faciliter sa gestion, un entrepôt de stockage envisage de concevoir un système permettant d'allouer automatiquement un emplacement de stockage pour chaque produit du chargement des camions qui convoient le stock à entreposer. Lors de l'arrivée d'un camion, un employé doit saisir dans le système les caractéristiques de chaque article ; le système produit alors une liste où figure un emplacement pour chaque article. Lors du chargement d'un camion les caractéristiques des articles à charger dans un camion sont saisies par un employé afin d'indiquer au système de libérer les emplacements correspondant.

Chapitre 21

Les aspects transversaux

21.1 Fondements

On abordera ici les aspects transversaux comme :

- la traçabilité des exigences
- les mécanismes d’allocation
- le diagramme paramétrique

21.2 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 21.1: Traçabilité

	Requirements	Structure	Comportement
Requirements	<<deriveRqt>>, <<refine>>, <<copy>>		
Structure	<<allocate>>, <<satisfy>>		<<allocate>>
Comportement	<<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

21.3 Mécanismes d'allocation

Un mécanisme nouveau en **SysML** et important pour l'Ingénierie 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.

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

21.4 Diagramme paramétrique

C'est une forme particulière de *Internal Block Definition* (cf. Section 19.5). On y retrouve les contraintes, déjà vues (cf. Figure 19.2), mais cette fois-ci on a la représentation graphique des liens entre les données.

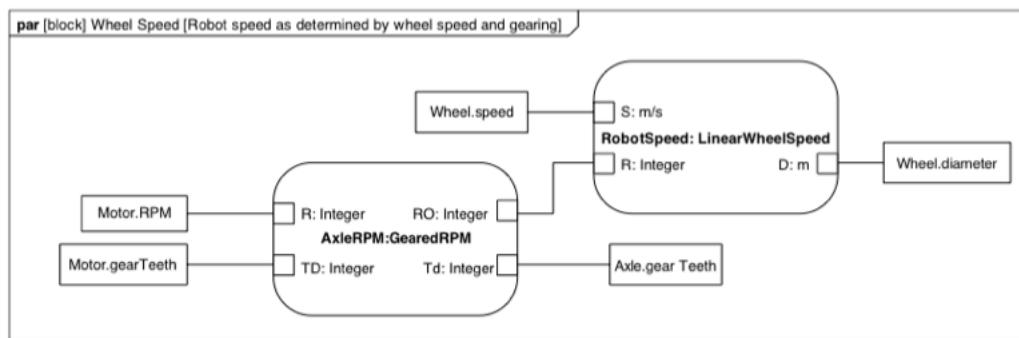


FIGURE 21.1 – Exemple de diagramme paramétrique

Il est regrettable que ce diagramme soit le moins utilisé (cf. Figure 2.5).

**Note**

Certaines approches utilisent des feuilles excel pour traduire les diagrammes paramétriques et contrôler l'impact des changements de valeurs de tel ou tel paramètre.

21.5 En résumé

En résumé l'expression du comportement du système en **SysML** est très similaire à ce qui est fait dans **UML**. On retrouve néanmoins le renforcement des liens entre éléments de modèles par les dépendances précises et les allocations. Un autre élément de renforcement entre éléments de modèles concerne le fait qu'un diagramme comportemental (comme une machine à état) est attachée à un élément bien précis (par exemple un bloc). Ces liens apparaissent entre blocs et machines à état, entre cas d'utilisation et diagrammes de séquence ou d'activité, etc.

21.6 Questions de révision

1. Quelles sont les différences entre <<satisfy>> et <<allocate>> ?
2. Pourquoi est-il important de relier un *use case* à au moins un *requirement* ?
3. L'inverse est-il aussi important ?

Quatrième partie

Partie 4 : Modéliser un système en SysML

Chapitre 22

Une démarche parmi d'autres

Nous allons aborder le développement complet de notre exemple fil rouge en suivant une démarche classique et simple (utilisée par exemple dans [SeeBook2012], où proche de la démarche globale enseignée dans nos cours de DUT Informatique, ou encore proche des documents de référence en la matière [HAS2012], [KAP2007],[FIO2012]) :

1. Spécification du système
2. Conception du système
3. Traçabilité et Allocations
4. Modèle de test

Nous partirons du modèle des exigences produit initialement. Mais avant tout, parlons outils.

22.1 Environnement de développement

Nous sommes des défenseurs des principes [DRY] et [TDD]. Nous allons donc réaliser nos diagrammes dans un outil et non "à la main" (de simples dessins). Nous choisissons ici l'outil **TOPCASED** pour des raisons que nous expliquerons ailleurs. La version utilisée pour réaliser les exemples de cette section est la version 5.2.

Un outil **SysML** seul ne suffit pas (cf. Section 22.1). Il faut penser à la documentation (cf. Section 22.1).

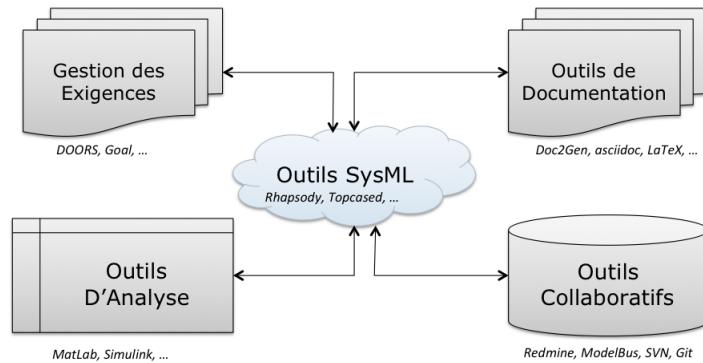


FIGURE 22.1 – Outilage autour de SysML

Outils

Il existe de nombreux outils **SysML**. Nous renvoyons le lecteur sur le site de **SysML-France** pour des informations sur les dernières versions des outils.

Génération de documentation

La plupart des outils permettent de générer de la documentation. Pour les outils basés **eclipse** comme **TOPCASED**, il est possible d'utiliser le plug-in **GenDoc2**.

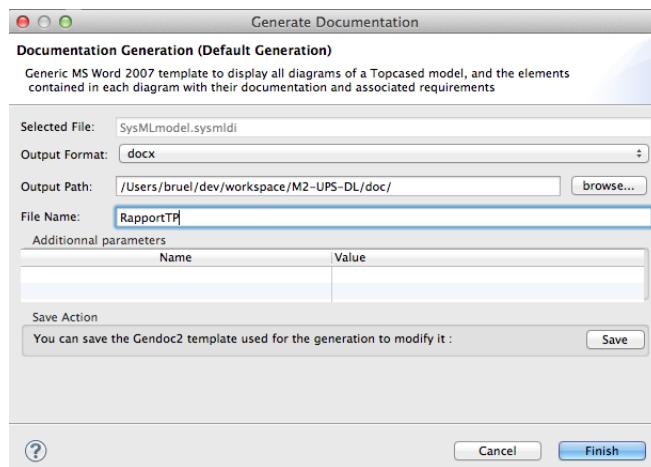


FIGURE 22.2 – Génération de documentation à partir de TOPCASED (1)

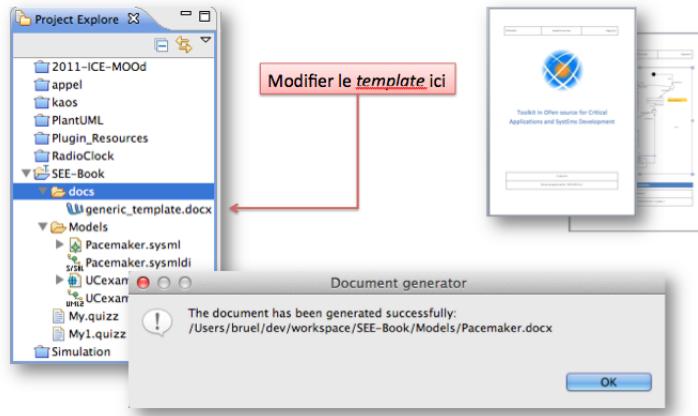


FIGURE 22.3 – Génération de documentation à partir de TOPCASED (1)

Les outils commerciaux comme **Rhapsody** permettent de générer de nombreux formats.

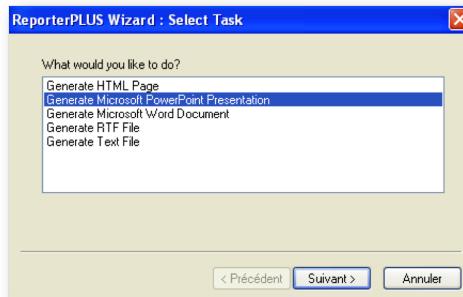


FIGURE 22.4 – Génération de documentation à partir de Rhapsody

Animation de modèles et simulation

Fortement liée aux outils, la possibilité d'animer les modèles ou encore d'effectuer des simulations est une exigence de plus en plus forte des ingénieurs systèmes.

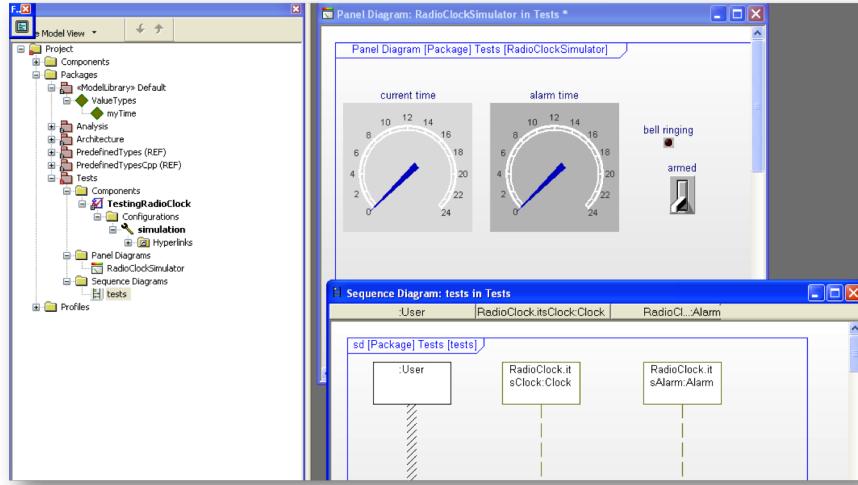
Il existe de nombreuses possibilités. Citons par exemple :

Génération de code VHDL

L'outil RTaW propose, via génération de code VHDL de simuler les modèles. Voir une démonstration [ici](#).

Simulation en Rhapsody

L'outil **Rhapsody** possède une interface très pratique pour faire du prototypage rapide.



Voir mon tutoriel (en anglais) disponible [ici](#).

Animation de modèles en Artisan

L'outil **Artisan** permet également de faire de l'animation de modèles.

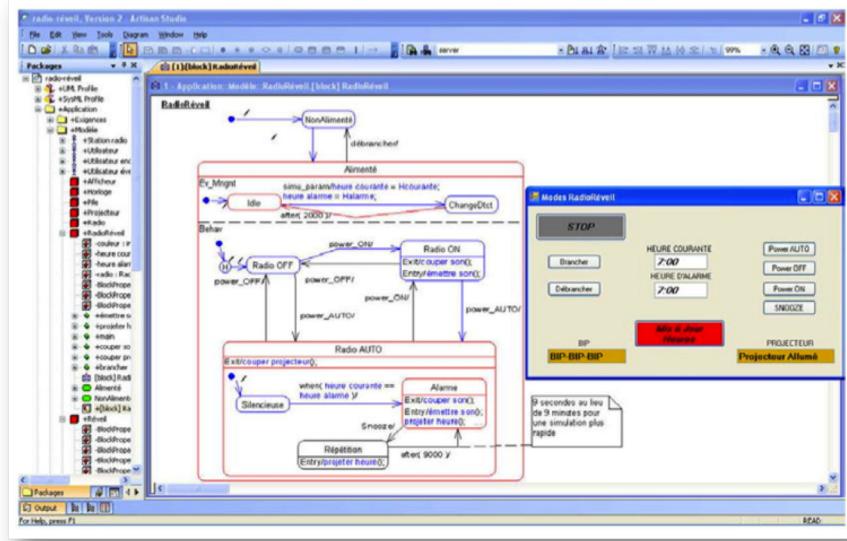


FIGURE 22.5 – Animation Artisan

22.2 Spécification du système

Il s'agit ici de décrire le contexte et d'identifier les principaux cas d'utilisation du système.

22.3 Conception du système

Chaque cas d'utilisation sera précisé (seq et act). Les données métier seront alors identifiées pour construire le modèle d'architecture logique (bdd et ibd) complété par la description des comportements complexes (stm). Enfin le modèle d'architecture physique permettra de déterminer les aspects déploiement et constructions physiques d'équipements/

22.4 Traçabilité et Allocations

Afin de consolider les différents modèles, les liens de traçabilité qui n'auront pas été déjà décrit¹ seront rajoutés en insistant sur les liens :

1. Il est recommandé de ne pas attendre pour matérialiser ces liens, mais de les exprimer dès que rencontrés dans telle ou telle modélisation.

- de satisfaction des exigences par les éléments de l'architecture,
- d'allocation des éléments du modèle fonctionnel vers les éléments logiques,
- d'allocation des éléments logiques vers les éléments de l'architecture physique.

22.5 Modèle de test

Nous insistons dans l'ensemble de nos formations sur les approches *test-driven*, alors nous montrons dans cette section comment participer à la qualité du développement d'un système en formalisant (par exemple avec des diagrammes de séquence de scénarios à éviter) les test et les jeux de test.

Chapitre 23

Recettes et bonnes pratiques

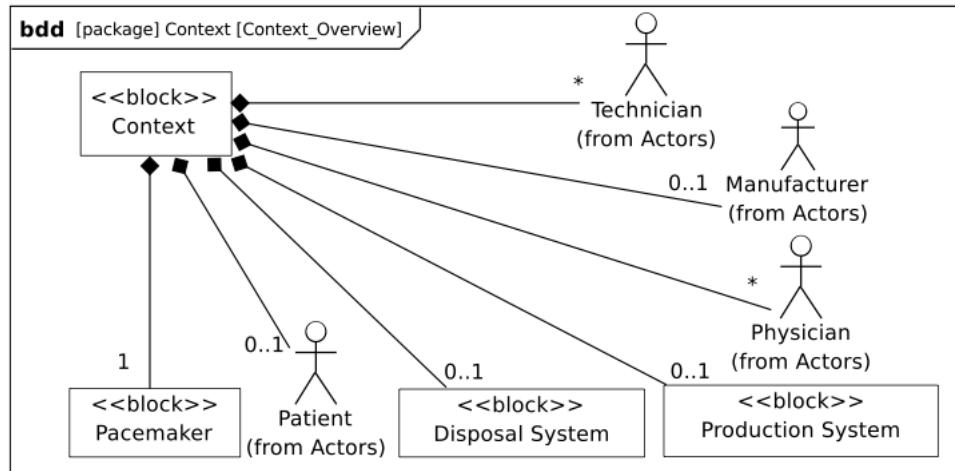
La plupart des ouvrages sur un langage enseignent les éléments de ce langage, comme nous l'avons fait à la partie précédente. Nous allons ici partir du principe inverse : comment modéliser tel ou tel partie ou vue de mon système avec SysML. Un peu à la manière des ouvrages du type *Cookbook*, nous allons donner une liste non exhaustives de recettes. Les choix des éléments de modélisation sont arbitraires ou tirés de discussions (comme ce sera mentionné si c'est le cas).

23.1 Architecture



Note

C'est conseillé. Un block `System` permet de raccrocher tous les éléments qui le composent à un même niveau. Dans l'exemple ci-dessous le système (le bloc `Pacemaker`) est lui-même un simple composant d'un élément de plus haut niveau : le contexte du système (le bloc `Context`) qui relie alors le système à son environnement. Voir aussi Section [22.3](#).

FIGURE 23.1 – Le contexte du Pacemaker ([\[SeeBook2012\]](#))

23.2 Comportement



Note

Un diagramme d'état peu modéliser les différents modes et les événements qui produisent les changements de mode.

Cinquième partie

Partie 5 : Pour aller plus loin

Chapitre 24

Considérations méthodologiques

Exemples de démarche autour de **SysML**, (cf. Chapitre 12).

Chapitre 25

Analyses et simulation

To be completed...

Chapitre 26

Exercices de révision

Reprendre ici les questions des chapitres (à organiser en fichiers!).

26.1 Quizz

Sujet

Un quizz en ligne est disponible [ici](#) (me contacter pour le mot de passe).

En voici une capture d'écran :

9 There are _____ different kind of Flow ports (write a number - no letters).
Points: 1
Réponse:

10 Use and Refine are some kind of _____.
Points: 1
Réponse:

11 The white diamond indicates in SysML:
Points: 1
Veuillez choisir une réponse.
 a. composition
 b. delegation
 c. generalisation
 d. aggregation

12 Match the following drawings:
Points: 1
asynchronous
reply messages
synchronous

FIGURE 26.1 – Exemple de QCM sur SysML

Corrigé

L'ensemble des questions du quizz a été généré à partir de ce fichier **quizz** (qui contient les réponses).

26.2 Mots croisés

Sujet

Voici un petit exercice (en anglais pour l'instant, désolé) pour changer :

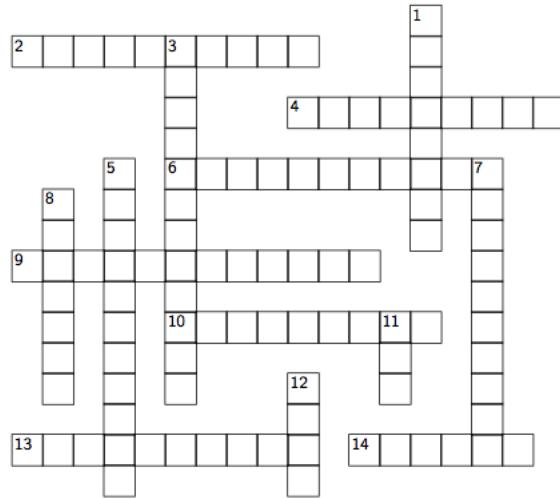


FIGURE 26.2 – Mots-croisés sur SysML

Vertical (across)

- 2. outside-inside connection
- 4. the full name of a model element is also a ... name
- 6. the black diamond in SysML
- 9. History is one of them
- 10. what a block can do
- 13. between states
- 14. a supporter of SysML

Horizontal (down)

- 1. used to describe a flow of actions
- 3. message represented by a regular (unfilled) arrow
- 5. each use case is advised to be linked to at least one of them
- 7. they are handled in SysML by Packages
- 8. communication entity in a sd
- 11. a supporter of SysML
- 12. number of diagrams in SysML

Sixième partie

Annexes

Chapitre 27

Liens utiles

- Sites officiels
 - Le site de l'association [SysML-France](#)
 - Le site de l'[OMG](#) (Object Management Group)
 - Le portail [SysML](#) de l'OMG (Object Management Group)
 - [La spécification elle-même](#)
 - Le site de l'[INCOSE](#) (International Council on Systems Engineering)
 - Le site de l'[AFIS](#) (Association Française d'Ingénierie Système)
- Blogs
 - Le site de [Tim Weilkiens](#)
 - La démarche [Caminao](#)
 - [Un WiKi avec de nombreux exemples](#)
- Outils SysML
 - [TopCased](#)
 - [Papyrus](#)
 - [Artisan](#)
 - [Rhapsody](#)
- Outils de production
 - Les conseils généraux de Scott Ambler sur [Ecrire un livre technique](#)
 - Les conseils techniques de Matthew Mc Cullough sur [Ecrire un livre technique](#)
 - [AsciiDoc](#) comme moteur de base.

- [Pandoc](#) pour la conversion de documents.
- [git-scribe](#) pour la génération des documents à partir d'[AsciiDoc](#).
- Divers
 - Pour en savoir plus sur l'[auteur](#)

Chapitre 28

Conventions

Il existe un certain nombre de conventions complémentaires aux règles de la spécification elle-même. Nous ne les donnons ici qu'à titre indicatif. Il est important pour une organisation qui souhaite utiliser **SysML** comme notation pour ses modèles de se mettre d'accord sur ce type de convention. En voici quelques-unes :

- Convention pour les noms :
 - de blocs commencent par une majuscule (origine : **UML**)
 - de cas d'utilisation (qui représentent une action) doivent être un verbe à l'infinitif (origine : **UML**)
 - d'activité (qui représentent une action) doivent être un verbe à l'infinitif (origine : **UML**)
 - d'attributs commencent par une minuscule et ne sont pas au pluriel (origine : **UML**)
- Convention pour les *requirements* :
 - ...
- Dépendances
 - En général un cas d'utilisation qui n'est inclus (<<include>>) que dans un seul autre cas est fusionné dans ce dernier
 - Lorsqu'un cas d'utilisation possède plusieurs cas <<refine>> qui pointent vers lui, on considère que ces différents cas sont des options possibles de raffinement (cf. Chapitre 28).



Note

Pour les origines UML de certaines conventions, cf. [\[Styles\]](#).

Chapitre 29

Le temps et sa prise en compte dans les modèles

Il existe plusieurs façons de représenter les informations temporelles.

SysML permet par exemple d'ajouter des contraintes temporelles sur le diagramme de séquence. Il existe deux types de contraintes :

- la contrainte de durée, qui permet d'indiquer une contrainte sur la durée exacte, la durée minimum ou la durée maximum entre deux événements ;
- la contrainte de temps, qui permet de positionner des étiquettes associées à des instants dans le diagramme au niveau de certains messages et d'ainsi contraindre leur relation.

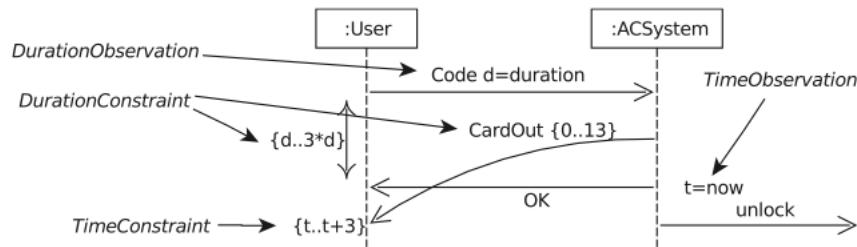


FIGURE 29.1 – Exemple de contrainte temporelle (tirée de [SysML])

Néanmoins, pour une prise en compte industrielle des contraintes temporelles, il conviendra d'utiliser le profil dédié à ces aspects : le profil MARTE.

Chapitre 30

FAQ

Cette *Frequently Asked Question* a été construite par expérience, en regroupant les questions des étudiants durant mes différentes interventions. J'ai aussi ajouté des questions souvent rencontrées dans les journées organisées par [SysML-France](#).



Note

Voir aussi cette [FAQ](#) très bien faite.

Cette FAQ peut servir de base à la révision d'examens (cf. aussi Chapitre [26](#)).

30.1 Peut-on avoir un *requirement* contenu plusieurs fois ?

Non. Le lien de *containment* est en fait une action qui place le "contenu" dans le "contenant". Dans TOPCASED, le diagramme laisse les liens précédents à l'écran, mais dans le modèle, c'est bien le dernier *containment* réalisé qui est pris en compte. Dans la figure ci-dessous le lien A-C a été "dessiné" après celui B-C.

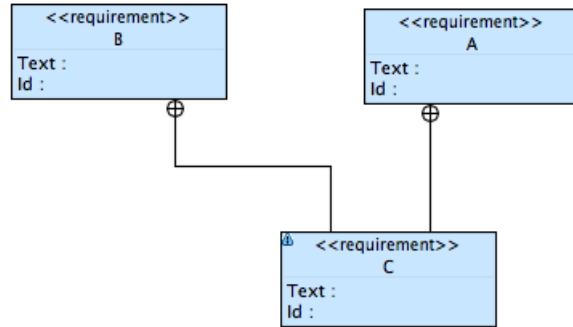


FIGURE 30.1 – Exemple de divergence modèle/diagramme (diagramme)

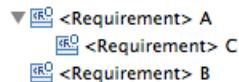


FIGURE 30.2 – Exemple de divergence modèle/diagramme (modèle)

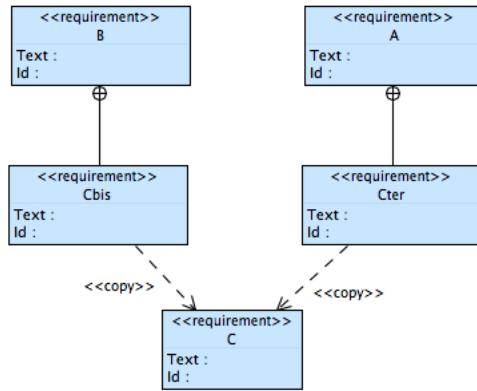
**Note**

Ce "bug" provient du fait que le lien de *containment* n'est pas un lien de dépendance, mais plutôt une représentation graphique de la contenance.

30.2 Comment alors peut-on "partager" un *requirement* ?

(En lien avec la question précédente)

L'organisation **SysML** des *requirements* est en fait un arbre. Pour réaliser ce "partage" certains utilisent un lien <<copy>> pour créer plusieurs copies d'un même *requirement*. Personnellement je n'aime pas cette solution.

FIGURE 30.3 – Exemple de partage de *requirement*

 **Définition : Réutilisation d'exigences (OMG SysML v1.3, Fig.16.6, p. 152)**

...the use of the Copy dependency [...] allow a single requirement to be reused in several requirements hierarchies.

30.3 Peut-on avoir un lien <<satisfy>> entre exigences?

Techniquement oui (<<satisfy>> étant dérivé de <<dependency>>), mais ça n'a pas beaucoup de sens que de dire qu'un besoin est satisfait par un autre. Il s'agit le plus souvent d'un lien <<deriveReqt>>.

 **Note**

Certaines méthodes utilisent ce lien pour par exemple exprimer qu'une exigence cliente est satisfaite par une exigence système (comme la méthode [Harmony]).

30.4 Quelle est la différence entre <<deriveReqt>> et <<refine>> ?

La norme n'impose pas de sémantique précise à <<deriveReqt>>. Il y a généralement deux interprétations.

1. Un usage classique est de l'utiliser pour ajouter des exigences plus détaillées déduites à partir d'autres exigences. Un exemple issue de la norme est une exigence de puissance moteur déduite (`deriveReqt`) depuis l'exigence sur l'accélération d'un véhicule.
2. Une vision plus stricte, aussi illustré par l'exemple précédent, est que l'exigence dérivée est une condition nécessaire (un pré-requis) à l'exigence cible.

Autre exemple respectant 1 mais pas 2 : "Le véhicule doit posséder 4 roues." est dérivé de "Le véhicule doit se déplacer sur route." En effet, un aéroglisseur répondrait aussi l'exigence initiale et n'a pourtant pas de roues.

Quant au `<<refine>>` il est utilisé pour indiquer qu'un élément de modèle (qui peut être lui-même un *requirement*) est un raffinement (au sens niveaux d'abstraction, du plus abstrait au plus concret) d'un *requirement*. Par exemple, un *use case* ou un diagramme d'activité peut être un raffinement d'une exigence fonctionnelle (textuelle par exemple).

30.5 A quoi sert le lien `<<trace>>` ?

Il est utilisé pour indiquer que l'on souhaite conserver un lien de traçabilité entre les éléments (par exemple entre un élément de modélisation et un document). Il est recommandé d'utilisé une de ces versions plus précises (`<<deriveReqt>>` ou `<<satisfy>>` par exemple).

30.6 Quelle est la version courante de la spécification et comment l'obtenir?

Verson 1 . 3 et disponible à l'URL: <http://www.sysml.org/docs/specs/OMGSysML-v1.3-12-06-02.pdf>

30.7 Quels en sont les changements notables depuis la dernière version ?

(en lien avec la question précédente)

Les changements notables par rapport à la 1 . 2 concernent :

- synchronisation avec les changements d'UML 2.3
- le métamodèle de *Conjugate ports* et sa notation
- le nommage des *activity regions* "interruptible"
- inclusion de *UML instance*
- inclusion des *structured activity nodes* d'UML

- inclusion des *multiple item flow* d’UML
- améliorations du support à *Unit* et *QuantityKind* pour les *value types*, et ajout d’un modèle (non normatif) pour définir les systèmes d’unités et de quantités.

**Note**

SysML v1.3 *Revision Task Force* dirigée par Roger Burkhart et Rick Steiner améliore de manière régulière la spécification en fonction des retours des utilisateurs.

30.8 Divers

Quelques autres questions que je laisse à votre sagacité :

- Pourquoi les ingénieurs systèmes auraient-ils besoin d’un n-ième langage de modélieration ?
- Quelles sont les relations entre “open source SysML” et “OMG SysML” ?
- Quelle est la feuille de route pour SysML 2.0?
- Quelles sont les relations entre UML et SysML? Peut-on les utiliser ensemble?
- Peut-on "customizer" SysML?
- Quel langage est le plus facile à apprendre, SysML ou UML?

Chapitre 31

FABQ

Cette *Frequently Asked Bad Question* est une compilation des questions trouvées parfois dans les forums et qui montrent l'incompréhension qui entoure encore **SysML**.

31.1 Comment installer SysML?

SysML n'est pas un programme qu'on installe. Il existe de nombreux outils qui chacun possède sa propre façon d'être installé sur votre machine (en fonction de votre système d'exploitation, si c'est un *plugin*, etc.).

Bibliographie

Bibliographie

- [1] [FIO2012] Fiorèse S., Meinadier J., Découvrir et comprendre l'ingénierie système, AFIS 2012.
- [2] [FMS] Friedenthal...
- [3] [HAS2012] Haskins C., SE Handbook Working Group, INCOSE Systems Engineering Handbook: Version 3.2.2, International Council on Systems Engineering, 2012.
- [4] [KAP2007] Kapurch S., NASA Systems Engineering Handbook, 2007 ([pdf](#)).
- [5] [] ENSI Bourges/PRISM.
- [6] [REQ2012] Guide Bonnes Pratiques en Ingénierie des Exigences, AFIS 2012.
- [7] [Roques2010] **Pascal Roques**. SysML par l'exemple - Un langage de modélisation pour systèmes complexes. Eyrolles. À acheter [ici](#).
- [8] [SeeBook2012] Kordon et al. To be published. XXX
- [9] [Sommerville1997] Ian Sommerville, Pete Sawyer. Requirements Engineering: A Good Practice Guide. Wiley, 1997.
- [10] [SysML] OMG. Systems modeling language version 1.3. Technical report, 2012.
- [11] [taoup] Eric Steven Raymond. The Art of Unix Programming. Addison-Wesley. ISBN 0-13-142901-9.
- [12] [Walsh1999] Norman Walsh & Leonard Muellner. DocBook - The Definitive Guide. O'Reilly & Associates. 1999. ISBN 1-56592-580-7.
- [13] [Harmony] Bruce Powel Douglass. Real-Time Agility: The Harmony/ESW Method for Real-Time and Embedded Systems Development. Addison-Wesley Professional, 2009. ISBN-10: 0-321-54549-4
- [14] [Styles] Scott W. Ambler. The Elements of UML 2.0 Style. Cambridge University Press, 2005. ISBN: 0-521-61678-6
- [15] [Fowler2004] Martin Fowler. UML 2.0 INFORMATIQUE PROFESSIONNELLE, 2004.

Glossaire

Acronymes SysML

act

Raccourcis pour Diagramme d'ACTivité dans une cartouche **SysML**

bdd

Raccourcis pour Block Definition Diagram dans une cartouche **SysML**

dss

Diagramme de Séquence Système (un sd où seul le système dans sa globalité est représenté¹)

ibd

Raccourcis pour Internal Block Diagram dans une cartouche **SysML**

par

Raccourcis pour Parametric Diagram dans une cartouche **SysML**

pkg

Raccourcis pour PaKaGe Diagram dans une cartouche **SysML**

req

Raccourcis pour REQuirements Diagram dans une cartouche **SysML**

sd

Raccourcis pour SEQuence Diagram dans une cartouche **SysML**

stm

Raccourcis pour STate Machine dans une cartouche **SysML**

uc

Raccourcis pour Use Case Diagram dans une cartouche **SysML**

Définitions générales

1. Il ne s'agit pas d'un acronyme **SysML** à proprement parler mais nous l'utilisons beaucoup.

Ressources

Les définitions ci-dessous sont regroupées à titre indicatif. Je vous invite à consulter les sources suivantes :

- Glossaire du [Software Engineering Institute](#)
 - [IEEE Computer Dictionary Online](#)
 - [Wikipedia](#) – Version française
-

DRY

Don't Repeat Yourself : Un bon principe qui veut qu'on évite de répéter des tâches manuelles (comme les tests) en utilisant plutôt des scripts et des programmes.

INCOSE

International Council on Systems Engineering : une organisation fondée en 1990 pour faire avancer les technologies d'Ingénierie Système.

IPT

Integrated Product Team : une équipe classique en développement système.

OMG

Object Management Group : L'organisme international chargé des principales normes liés à l'objet (CORBA, UML, etc.).

TDD

Test Driven Development : Développements dirigés par les tests. On écrit les tests avant d'écrire le code. On travaille son code tant que les tests ne passent pas.

TRL

Technology Readiness Level : système de mesure employé par des agences gouvernementales américaines et par de nombreuses compagnies (et agences) mondiales afin d'évaluer le niveau de maturité d'une technologie (cf. [Wikipedia](#)).

SysML

System Modeling Language ™ : le langage de modélisation de systèmes maintenu par l'[OMG](#).

Index

- AADL, 13
- act, 7, 40, 42, 86, 117
- AFIS, 49
- AsciiDoc, 3–5, 106, 107
- bdd, 6, 39, 40, 42, 59, 61, 65, 72, 80, 95, 117
- dss, 72, 86, 117
- eclipse, 92
- ibd, 6, 40, 42, 64, 69, 72, 95, 117
- INCOSE, 6, 36
- IS, 22–24, 27, 29, 35, 63, 88, 118
- Méthodes, 33, 100
- MARTE, 13, 109
- OMG, 2, 6, 10, 13, 36, 37, 118
- par, 6, 40, 72, 78, 117
- pkg, 86, 117
- Raton laveur, 7
- req, 7, 40, 42, 53, 117
- Rhapsody, 4, 93, 94
- sd, 7, 40, 75, 86, 104, 117
- STI2D, 11, 12
- stm, 7, 40, 42, 86, 95, 117
- SysML, 2, 4, 5, 7, 10, 11, 13, 20, 22, 36, 38, 39, 42, 47, 55, 61, 68, 81, 83–85, 87–89, 91, 92, 100, 108, 109, 111, 115, 117
- SysML-France, 3, 11, 92, 110