

Introduction ‡ SysML

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME
1.3	12/10/2013		JMB

Part I

Partie 1 : Introduction

Chapter 1

Avant-propos

1.1 Sur ce document

1.1.1 A qui est destinÉ ce document?

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

1.1.2 A qui n'est-il pas destinÉ?

Si vous appartenez ‡ 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](#))

1.1.3 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),
- ‡ 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. Chapter ??) les ouvrages et autres documents utilisés.

Je tiens ‡ remercier mes collègues qui m'ont aidé dans mon entreprise :

- Nicolas Belloir de l'**Université de Pau et des Pays de l'Adour**, Laurent Nonne de l'**IUT de Blagnac** et Karina Aguilar de l'**Universidad Autónoma de Guadalajara**;
- mes collègues de **SysML-France** : **Pascal Roques** (PRFC), Agusti Canals (C-S) et **Loïc Fèjoz** (RTaW);
- mon maître d'**AsciiDoc** : Jean-Michel Inglebert.

1.2 Sur l'auteur

- Professeur ‡ l'**Université de Toulouse**
- Co-fondateur de l'association **SysML-France** en 2009
- Membre du comité Éditorial de la revue **SoSyM** depuis sa création en 2002
- Membre du *Steering Committee* de la conférence ACM/IEEE **MODELS** depuis 2008
- Enseignant en modélisation depuis 1995
- Chef du département informatique de l'**IUT de Blagnac** de 2009 ‡ 2012
- Co-responsable de l'axe Systèmes Ambiants de l'**IRIT** depuis 2009
- Mariée, une (merveilleuse) fille

1.3 Comment lire ce document?

1.3.1 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 cliquables ne vous servent à rien, mais n'hésitez pas à en consulter la version [Électronique](#)!

1.3.2 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, gr,ce 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. Chapter ??),
- 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. [?simpara].

Note

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

**Caution**

Ces notes indiquent des points importants qui réclament votre attention.

Tip

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.

Convention : Titre du conseil spécifique

Conseil spécifique aux formateurs.

Note

Modélisation SysML incorrecte.

Note

Modélisation SysML partiellement correcte ou pouvant prêter à confusion.

Note

Modélisation SysML correcte.

1.3.3 Pourquoi parler de "document"?

Parce que j'ignore la version que vous êtes en train de lire. À partir de l'original, plusieurs versions ont été générées gr,ce 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!)

1.3.4 Utilisation et autres mentions légales

Dernière MAJ : 12/10/2013 - 09:52:25 CEST

Document généré par **Jean-Michel Bruel** via **AsciiDoc** (version 8.6.8) de *Stuart Rackham*. La version présentée a été générée en utilisant **W3C HTML Slidy** © de *Dave Raggett*, améliorée 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 transposée.**

N'hésitez pas à m'envoyer vos remarques en tout genre en m'écrivant [ici](#).

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'essaie 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 cités.

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.

¹ voir aussi le concept de *Mind Maps*.

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 ou de prépa ne connaissent pas UML™ et sont pourtant formés à SysML™.
- Ceux qui connaissent déjà UML™ auront sûrement plaisir à retrouver les bases.

Chapter 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
- Logo officiel : icons/sysml.jpeg

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 SysML™ n'est qu'une redéfinition de la classe UML™.

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.

Note

On ne dit pas "le SysML" mais tout simplement "SysML".

2.4 Références et liens utiles

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

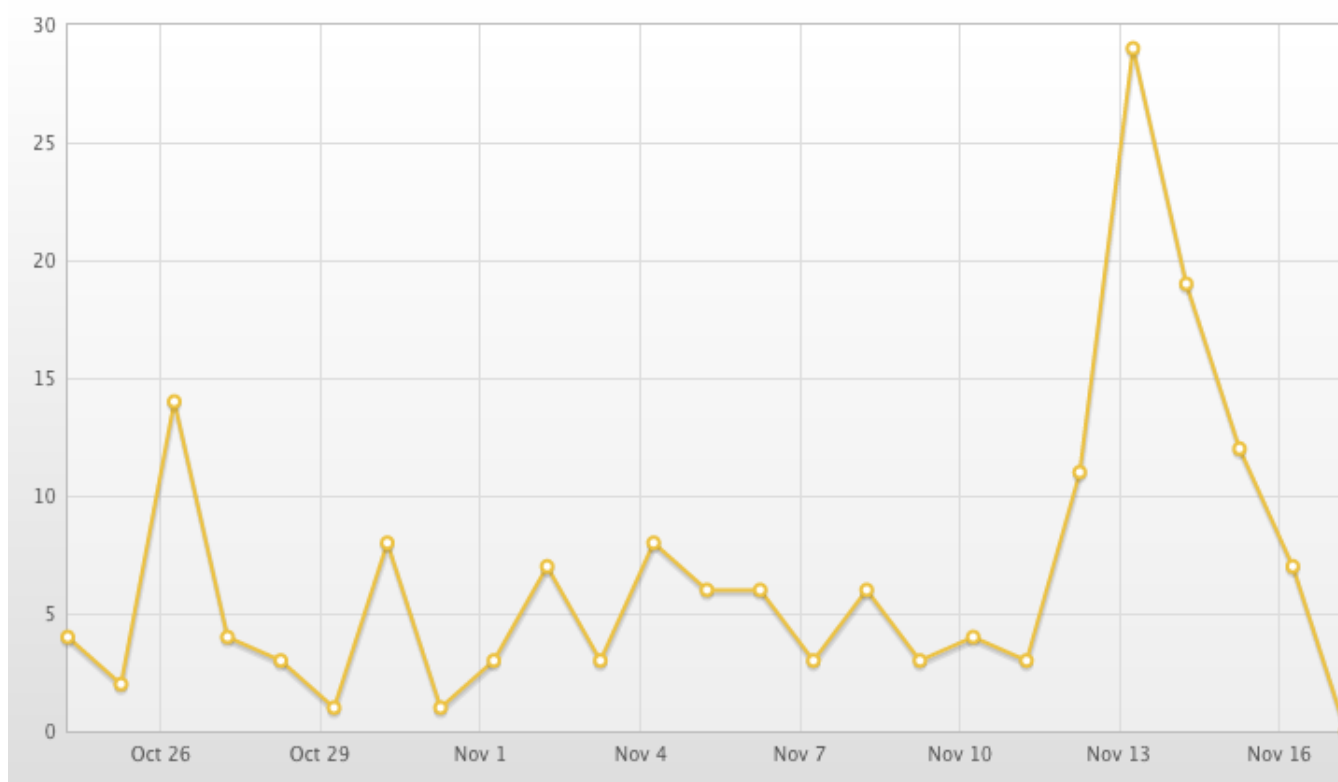


Figure 2.1: Trends : Twitter

Note

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

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

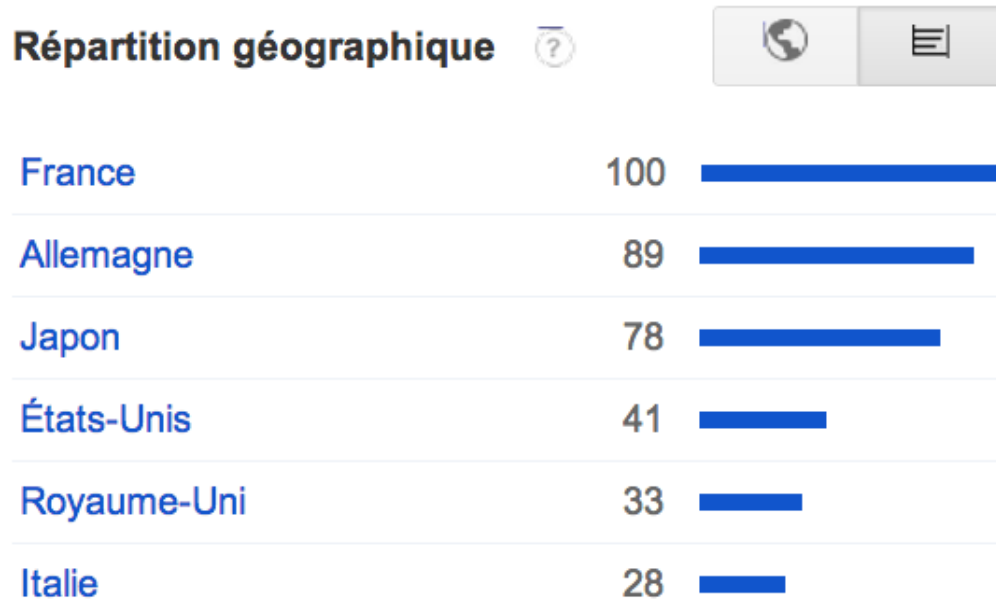
Répartition géographique ?



0 100

Région | Ville

Figure 2.2: *Trends* : Google (Carte)

Figure 2.3: *Trends* : Google (Liste)

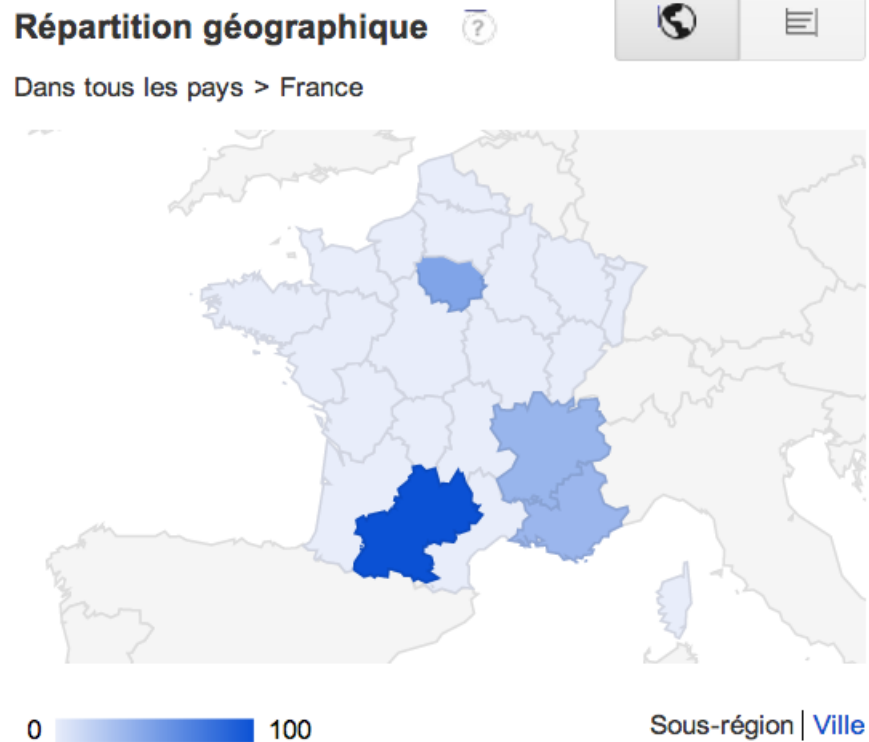
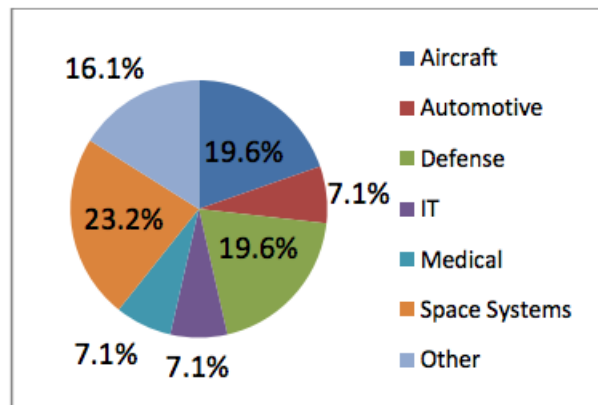


Figure 2.4: Effet SysML-France?

À noter également que l'OMGTM a réalisé en 2009 une enquête sur l'utilisation de SysMLTM² dont voici deux extraits :

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



Application of SysML by System Type

Figure 2.5: Principaux domaines d'utilisation

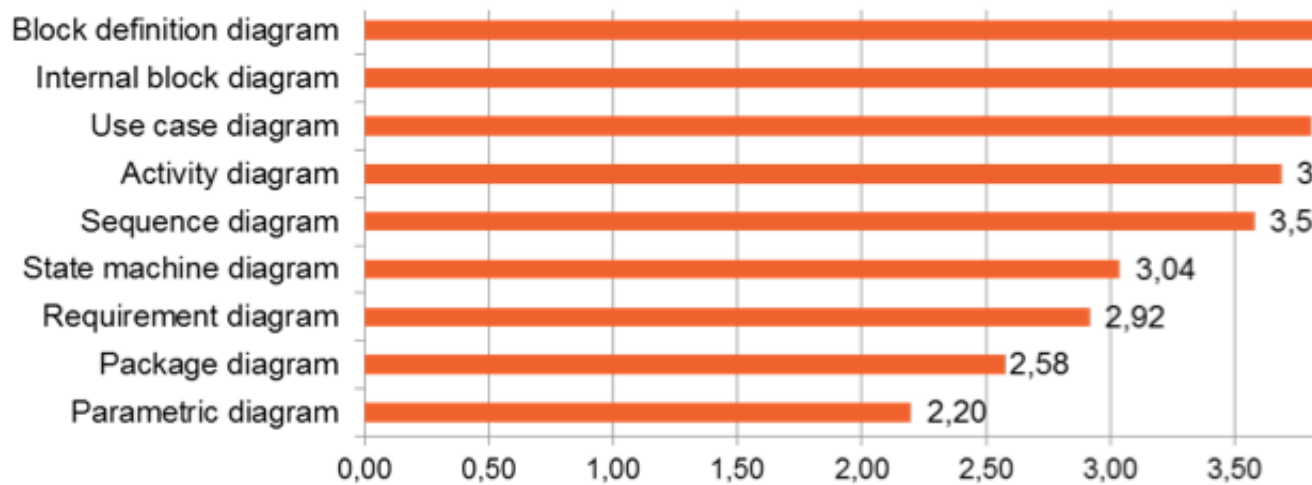


Figure 2.6: Diagrammes les plus utilisés

Chapter 3

¿ propos du Bac STI2D et des classes prÈpas

Si vous utilisez cet ouvrage dans le cadre du bac STI2D (Sciences et Technologies de l'Industrie et du DÈveloppement Durable) qui a introduit depuis 2011 la notation **SysML™** au programme ou encore dans le cadre des classes prÈparatoires aux Ècoles d'ingÈnieur qui devraient l'adopter bientÙt, nous donnons ici des conseils sur l'utilisation de ce cours ¹.

L'objectif dans ces niveaux n'est pas de former des spÈcialistes de **SysML™** mais de permettre ‡ tous d'apprendre une notation pour la modÈlisation de systÈme qui se veut universelle. Il ne faut donc pas viser la complÈtude ou mÙme demander trop de dÈtails. La logique de la dÈmarche de modÈlisation et l'importance de la communication devront primer.

Note

A l'heure oÙ nous Ècrivons ces lignes, il est en effet prÈvu de l'enseigner en classe prÈpa dÈs 2013.

3.1 Utilisation pratique en STI2D

Cet ouvrage est tout ‡ fait utilisable dans le cadre des ces cours. Pour STI2D, seul un sous-ensemble des diagrammes de **SysML™** a ÈtÈ retenu. Les ÈlÈves et les enseignants du bac STI2D pourront trouver dans ce document des ÈlÈments utiles sur ces diagrammes :

- diagramme des exigences (cf. Section ??)

¹ Je remercie au passage les collÈgues de LycÈe rencontrÈs dans le cadre de **SysML-France** pour nos fructueuses discussions ‡ ce sujet.

- diagramme des cas d'utilisation (cf. Section ??)
- diagramme de sÈquences (cf. Section ??)
- diagramme d'Ètats (cf. Section ??)
- diagramme de dÈfinition de blocs (cf. Section ??)
- diagramme de blocs internes (cf. Section ??)

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

Ils servent trois objectifs principaux inscrits au programme et dont les ÈlÈves pourront Ègalement trouver des ÈlÈments de rÈflexion :

- ModÈlisation des exigences (cf. Chapter ??)
- ModÈlisation structurelle (cf. Chapter ??)
- ModÈlisation comportementale (cf. Chapter ??)

Tip

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

3.2 Classe prÈparatoire et UPSTI

L'Union des Professeurs de Sciences et Techniques Industrielles (**UPSTI**) prÈpare ‡ leur o˘ nous Ècrivons ces lignes un document de cadrage en terme de mÈthodologie et de dÈmarche. Nous n'insisterons donc pas sur ces aspects, mais donnerons quelques pistes dans le chapitre commun sur les conseils d'enseignement **PÈdagogie**.

Notons que dans ce cadre l˘, les 9 diagrammes sont utilisables mˆme si pour l'instant les diagrammes paramÈtriques et de paquetages sont un peu mis en retrait.

3.3 Pour aller plus loin

Les questions et exercices de fin de chapitres de la partie notation seront peut-ˆtre d'un niveau plus avancÈ pour servir vÈritablement d'exercices, mais pourront amener ‡ une rÈflexion encadrÈe par l'enseignant.

Chapter 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 exemple complets :

- Le radio-réveil de [Pascal Roques](#) [[Roques2010](#)], un exemple simpliste mais didactique qui a le mérite d'être déjà connu des modelleurs [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

Tip

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

¹ Nous avons réalisé le chapitre d'introduction ‡ [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.

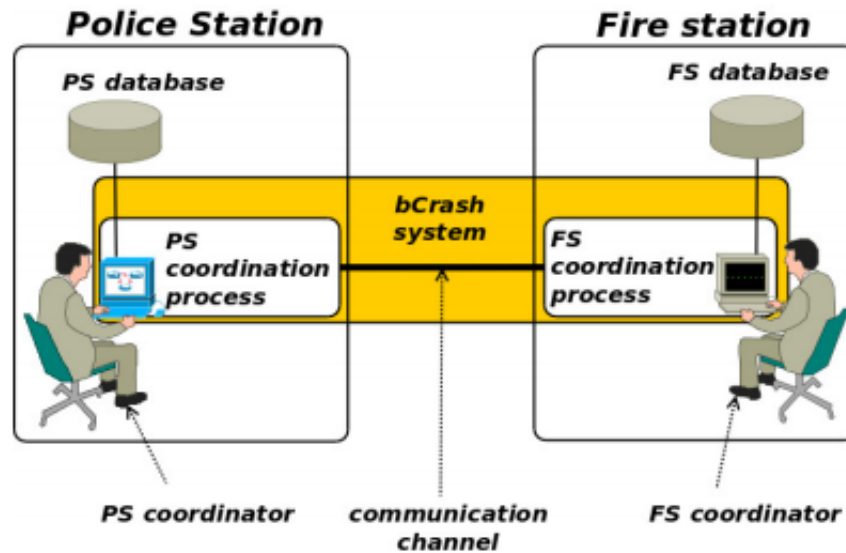


Figure 4.1: Le système CMS

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 c'est-à-dire 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.

4.1.1 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.

4.1.2 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.

4.1.3 Coordinateur du poste de police (PSC)

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

4.1.4 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.

4.1.5 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.

4.1.6 Témoin (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.

4.1.7 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 :

**Caution**

La traduction a été principalement obtenue automatiquement, alors n'hésitez pas en cas de doutes à poser des questions!

4.1.7.1 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 reporté ou interrompu quand une crise est imminente sans affecter les capacités des systèmes.

4.1.7.2 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.

4.1.7.3 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 déployées (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.

4.1.7.4 Sécurité

- Le système doit définir des politiques d'accès pour les différentes catégories d'utilisateurs. La politique d'accès doit écrire 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.

4.1.7.5 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.

4.1.7.6 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.

4.1.7.7 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.

4.1.7.8 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

Part II

Partie 2 : IngÈnierie systÈme

Chapter 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	Flux
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 élément 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 Section ??) 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 difficile d'exprimer, de comprendre (vos modèles) et de valider.

Transverse

Un certain 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, etc., 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.).

Chapter 6

Différence avec l'ingénierie logicielle

Enseignant en informatique, je me retrouve souvent ‡ enseigner SysML™ ‡ 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 ‡ :

- 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.

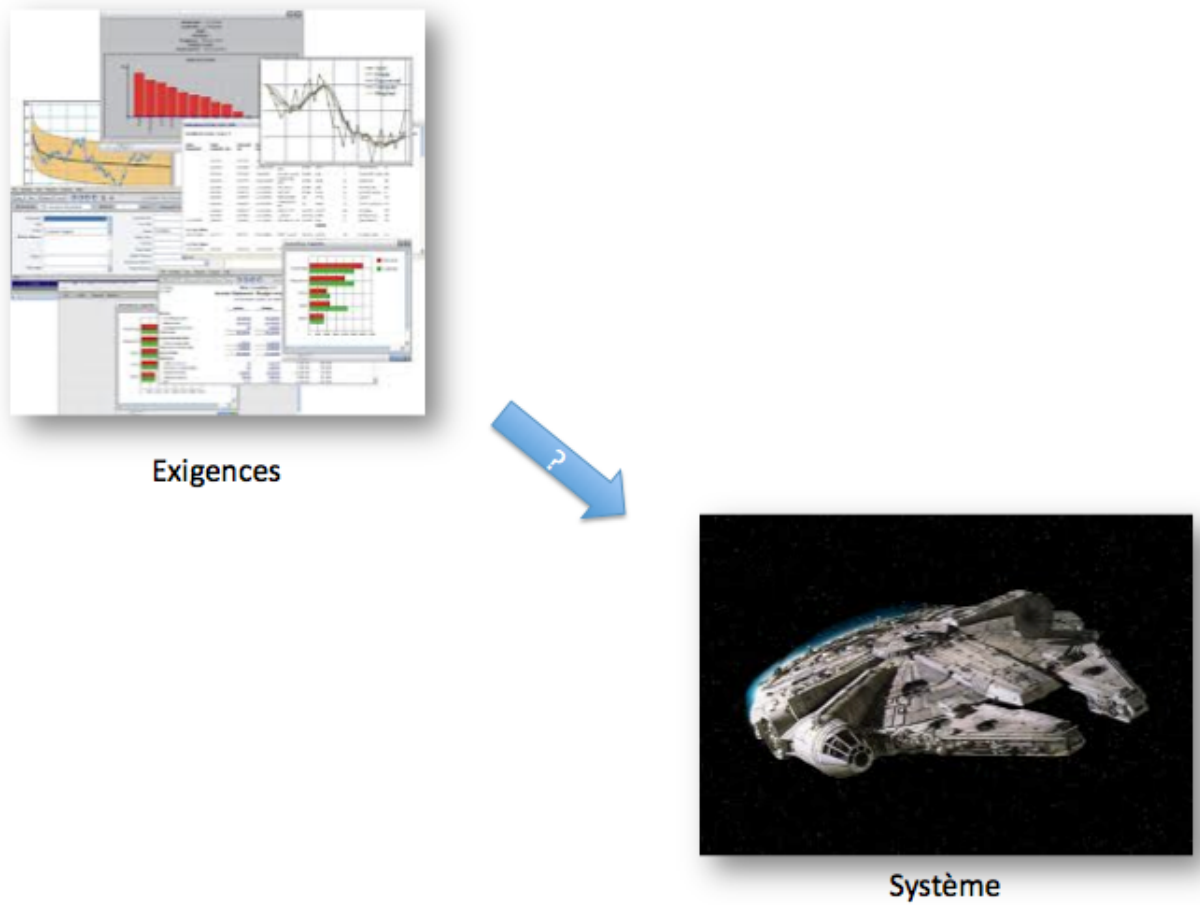


Figure 6.3: Des exigences au système

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

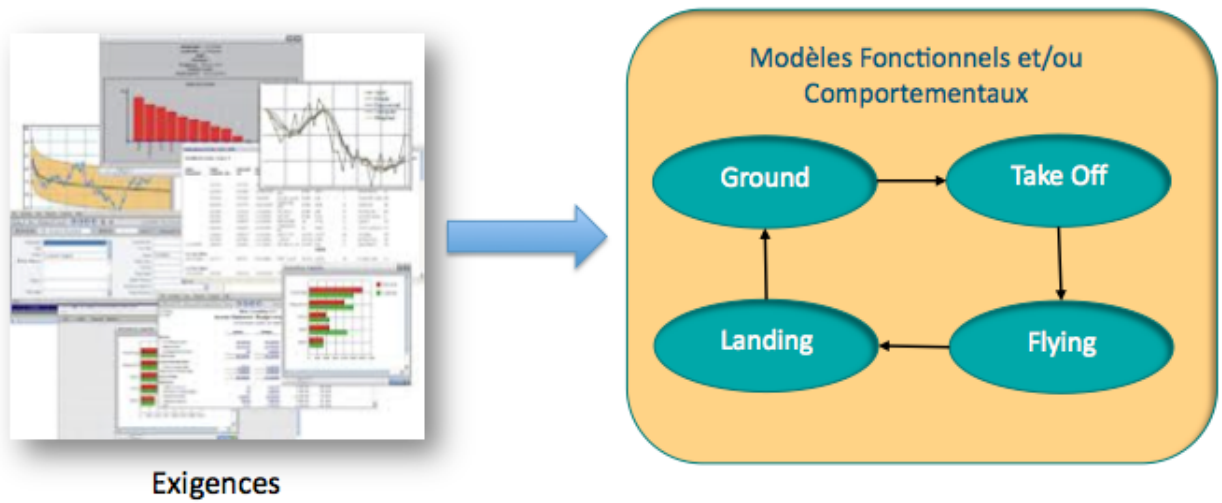


Figure 6.4: Analyse Fonctionnelle et/ou Comportementale

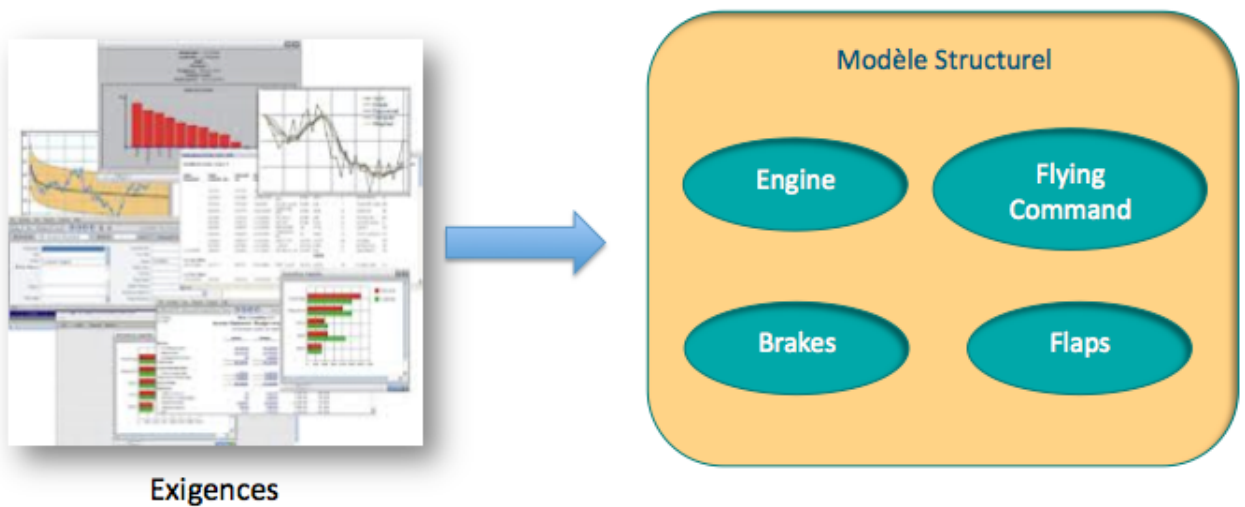


Figure 6.5: Analyse Structurelle

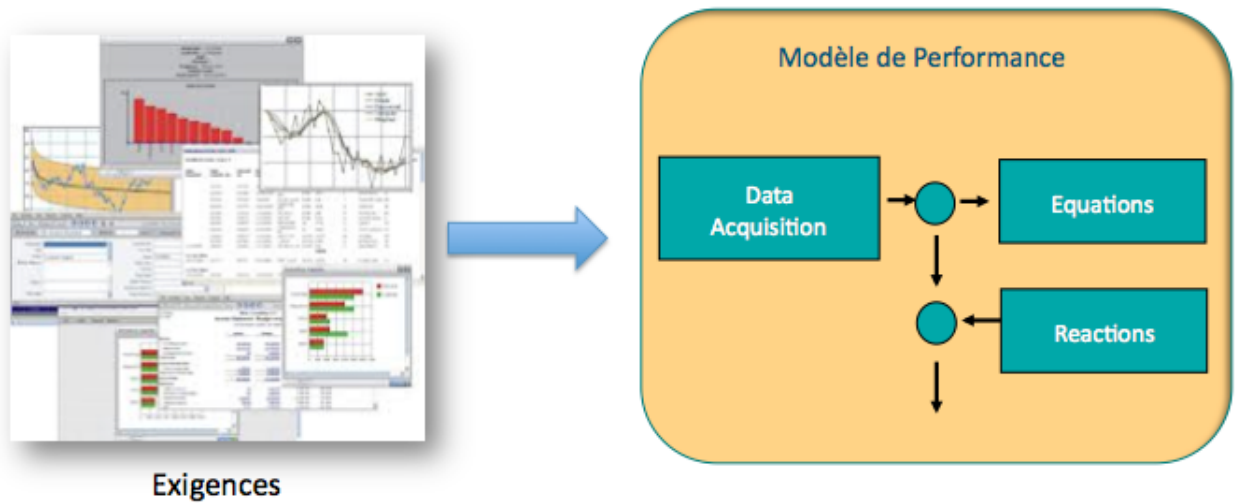


Figure 6.6: Analyse de performance

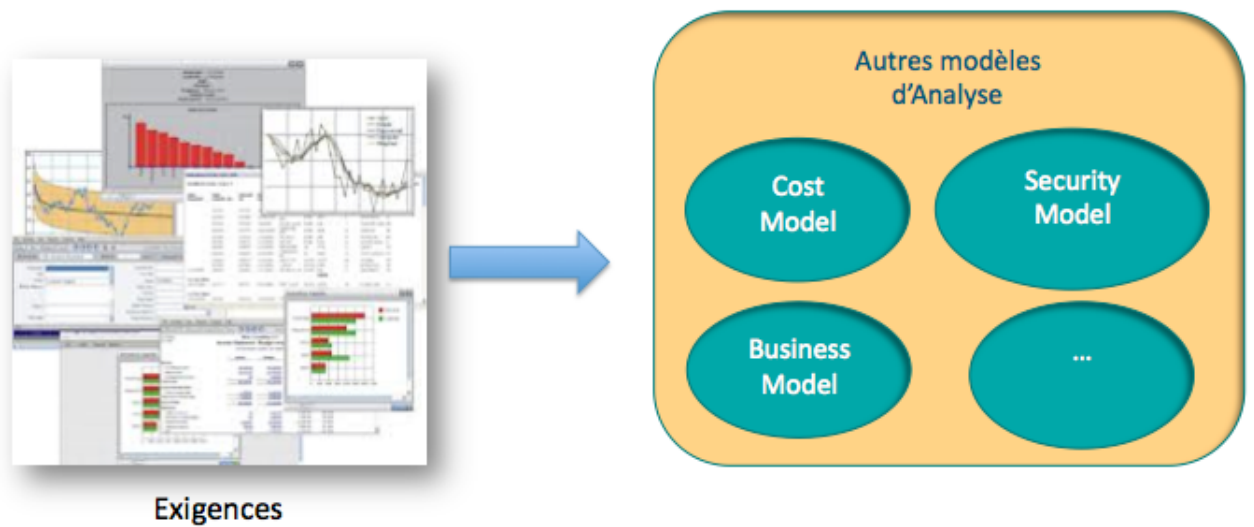


Figure 6.7: Analyses spécifiques

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

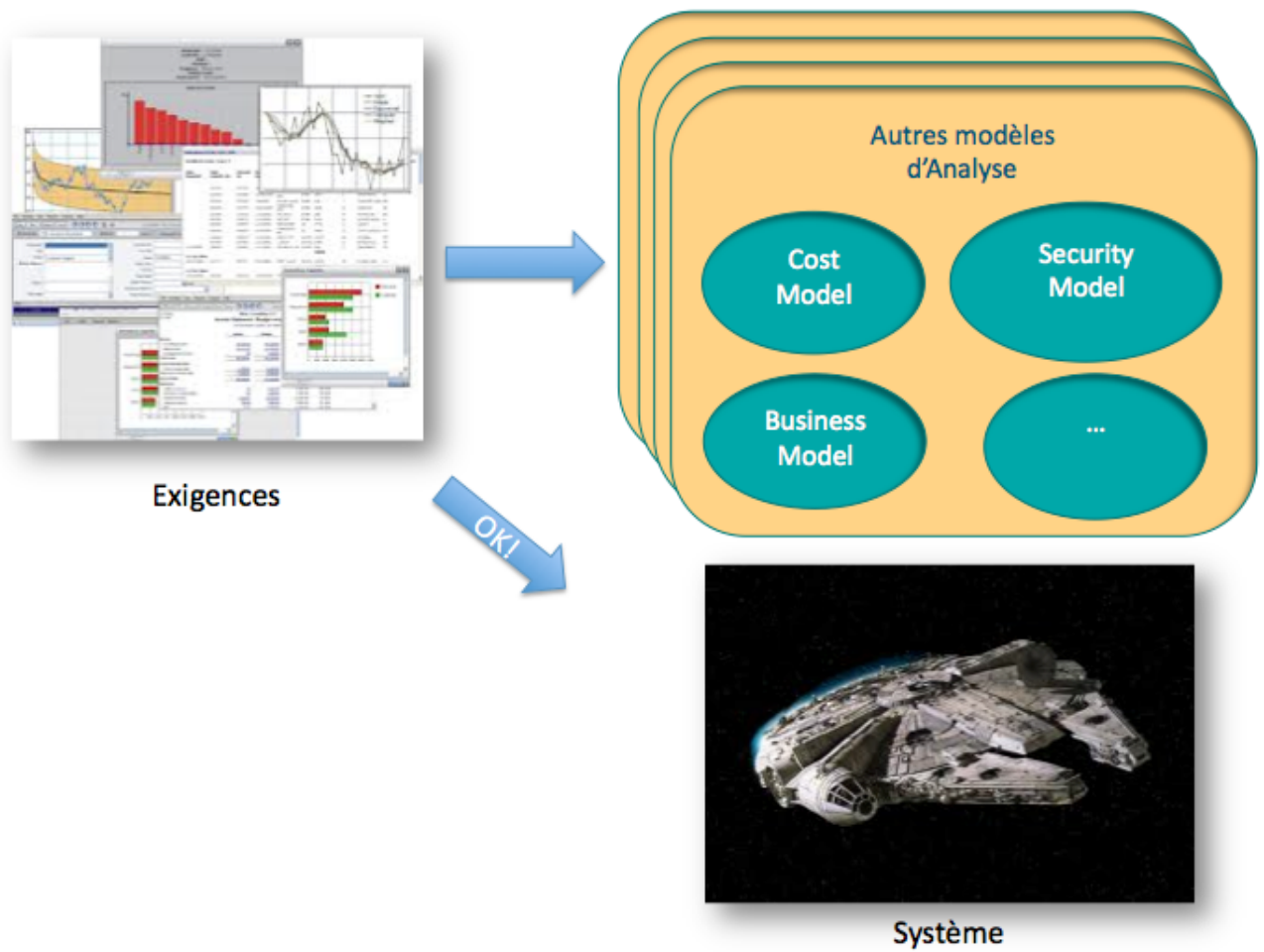


Figure 6.8: Des exigences au système

Chapter 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, AFIS, ...).

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

Chapter 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. Chapter ??).

Chapter 9

Les exigences

L'ingénierie des exigences est d'une importance capitale en Ingénierie Système. En général les exigences sont exprimées par des ingénieurs dédiés à cette activité. La complexité des systèmes modernes (embarqués, communicants, critiques, ...).



Figure 9.1: 300 corps de métiers sont parfois présents sur un chantier

Besoins, exigences : question de vocabulaire

La difficulté de l'emploi massif de l'anglais en Ingénierie Système fait qu'il existe souvent une confusion entre les termes anglais et leurs traduction anglaise. Nous précisons donc ici notre utilisation des termes :

Requirements

Exigences, c'est à dire une fonction ou une propriété que doit satisfaire le système considéré. Par nature une exigence doit pouvoir être vérifiable. En génie logiciel on parle plus classiquement des spécifications ("spec") pour parler des contraintes à respecter pour un système. Les ingénieurs systèmes ont depuis longtemps intégré le terme d'exigences comme traduction directe de *requirement*.

Besoins

Il s'agit des exigences du client. En SysML™ on va plus les retrouver dans les cas d'utilisation. Ils sont à l'origine des *requirements* tels que définis plus haut.

Il est important pour une exigence qu'elle ne soit pas ambiguë (contrairement au terme *en* dans la consigne exprimée par la maman dans l'illustration ci-dessous : "*Ramène moi 1 bouteille de lait. S'il y a des oeufs, ramène m'en 6.*").

Joke

Figure 9.2: Humour (taken from [here](#))

Chapter 10

L'architecture du système

Liens avec AADL, ...

Chapter 11

Le comportement du système

Liens avec la V&V

Chapter 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. Chapter ?? pour la présentation de quelques méthodes bien identifiées) :

Example 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.

Example 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. Table ??).

Example 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 ‡ votre cas particulier.

Part III

Partie 3 : La notation SysML

Chapter 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 "unifi  e" pour les mod  les : **UML™**. N  anmoins cette notation est peu adapt  e pour l'Ing  nierie Syst  me :

- UML 1.x   tait compl  tement inadapt  e :
 - Principalement pour les syst  mes d'information
 - Peu de liens entre les diagrammes
 - Peu de liens entre les mod  les et les exigences
- UML 2.x n'est pas beaucoup mieux si ce n'est :
 - Implication des ing  nieurs syst  mes pour sa d  finition
 - Introduction du diagramme de structure composite

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

- Standard *De facto* en g  nie logiciel
- Fournit beaucoup de concepts utiles pour d  crire des syst  mes (m  me complexes)
- Stable et extensible (gr,ce notamment au m  canisme 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)

Chapter 14

Introduction ‡ SysML

14.1 Fiche d'identité

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

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

14.2 Différence avec UML

La figure [suivante](#), tirée de la spécification, résume bien les liens entre **SysML™** et **UML™**, ‡ 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, ...

Tip

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

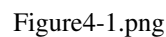


Figure 14.2: Les 9 diagrammes SysML et leur lien avec UML

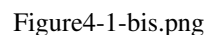


Figure 14.3: Version abrégée des diagrammes

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 **SysML™**¹, mais d'utiliser des outils dédiés (cf. Section ??). Ils respectent en général la norme **OMG SysML v1.3** (bien qu'il faille se méfier).

Note

Notez que la norme permet de faire des adaptations graphiques (cf. la discussion <http://www.realtimetework.com/2011/08/is-sysml-too-abstract/>).

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

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

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

Voici deux exemples de non respect de la notation qui illustre le type d'erreur que l'on trouve souvent dans les modèles qui circulent sur Internet ou même parfois dans certains livres.

14.5.1 Diagramme de bloc

Par exemple dans ce diagramme les blocs ne respectent pas la syntaxe graphique de **SysML™** :

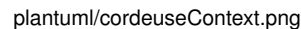
¹ Sauf bien sûr au brouillon ou sur un tableau, notamment quand on travaille en Équipe.

Erreur : mauvais symboles graphiques pour les blocs

cordeuseContext.png

Pour rappel, la notation `jmb :Personne` permet de représenter un **objet** (une instance d'une classe ou d'un bloc). C'est donc une notation utilisée par exemple dans les participants d'un diagramme de séquence ou encore les parties d'un diagramme interne de bloc.

Donc dans le diagramme ci-dessus, l'acteur est correct (on peut mettre des acteurs dans un bdd, cf. OMG SysML v1.3 p.32), par contre les objets `Block : . . .` est une erreur de notation.

Solution : utiliser un outil (ici BOUML)

plantuml/cordeuseContext.png

**Warning**

Attention, il est tout à fait possible de représenter des instances dans un bdd (cf. OMG SysML v1.3 p.34), même si c'est très peu courant.

14.5.2 Diagramme de séquence

Erreur : pb avec les participants et la boucle

cordeuseSeqBad.png

Plusieurs problèmes de non respect de la notation :

- il manque le rectangle aux participants
- les participants semblent être des blocs et non des instances
- la boucle devrait avoir une condition (même "toujours" pour une boucle infinie)

Note

Le dernier problème est plus une convention qu'une véritable erreur. Cf. Chapter ??.

14.6 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
- MagicDraw
- ...

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.



Warning

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.

14.7 Cadre pour les diagrammes

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

- Chaque diagramme SysML™ décrit un élément précis (nommé) de modélisation
- Chaque diagramme SysML™ doit être représenté à l'intérieur d'un cadre (*Diagram Frame*)
- L'entête du cadre, appelé aussi **cartouche**, indique les informations sur le diagramme :
 - le **type** de diagramme (*req*, *act*, *bdd*, *ibd*, *stm*, etc. en gras) qui donne immédiatement une indication sur le point de vue porté à l'élément de modélisation (comportement, structure, etc.)

- le type de l'Élément (par exemple *package*, *block*, *activity*, etc.), optionnel
- le nom de l'Élément (unique)
- le nom du diagramme ou de la vue, optionnel

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

pacemaker-context.png

Figure 14.4: Exemple de diagramme SysML

Convention : Utilisation systématique des cartouches

Tout diagramme proposé à un Étudiant pour écrire un système devrait posséder un entête précis.

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 Table ??) :

Table 14.1: Organisation

	Requirements, cf. Section ??	Structure, cf. Section ??	Comportement, cf. Section ??	Transverse, cf. Section ??
Organisation	pkg	pkg, bdd	pkg	
Analyse, Conception, Implémen- tation ²	req	bdd, ibd, sd, par	uc, sd, stm, act	par

14.8 Organisation

	Requirements	Structure	Comportement	Transverse
Organisation				

² En fonction du niveau de détail.

	Requirements	Structure	Comportement	Traverse
Analyse				
Conception				
Implémentation				

14.8.1 Fondements

On abordera :

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

14.8.2 Le *Package Diagram*

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

- Il est identique à UML™, et classique pour les développeurs (java notamment)
- Il permet d'organiser les modèles en créant un espace de nommage (cf Section ??)

Les modèles peuvent être organisés selon toutes sortes de considération (cf. Section ??) :

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

14.8.3 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

Tip

Un point de vue (*viewpoint*) est utilisÉ pour matÉrialiser une perspective particuliÈre de modÈlisation. 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>>) ‡ un point de vue.

14.8.4 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., sÈcuritÈ, performance, ...)
- etc.

pkg-organisation2.png

Figure 14.5: Exemple d'organisation simple

pkg-organisation-modelview.png

Figure 14.6: ReprÉsentation de cette organisation dans un outil

pkg-organisation.png

Figure 14.7: Un autre exemple d'organisation

pkg-topcased.png

Figure 14.8: Un autre exemple d'organisation

Tip

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

pkg-template.png pkg-topcased-default.png

14.8.5 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 d'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 précédé par son (ou ses) *package(s)* (e.g., `Structure::Products::Clock`).

Tip

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).

14.8.6 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>>

14.8.7 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 14.2: Organisation

	Requirements	Structure	Comportement	Interverse
Organisation	package	package	package	dependencies
...				

14.8.8 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) ?

14.9 Les exigences

	Requirements	Structure	Comportement	Interverse
Organisation				
Analyse				

	Requirements	Structure	Comportement	Flux
Conception				
Implémentation				

14.9.1 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*

Tip

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

14.9.2 L'organisation des *Requirements*

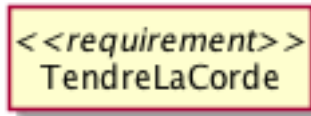
Il ne s'agit pas ici de revenir sur les exigences elles-mêmes, mais plutôt de voir comment SysML™ permet de les exprimer, de les manipuler et surtout de les lier avec le reste du système.

14.9.2.1 Représentation de base

Un *Requirement* en SysML™ n'est qu'un bloc particulier.

Définition : Requirements (OMG SysML v1.3, p. 139)

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

Figure 14.9: Un *Requirement* en SysML

14.9.2.2 Différents types d'organisation

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

Plusieurs types d'organisations sont possibles :

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

14.9.2.3 Tableaux de *Requirements*

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

Définition : Requirements Table (OMG SysML v1.3, p. 145)

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

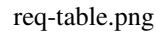


Figure 14.10: Exemples tableau d'exigences (OMG SysML v1.3, p. 145)

La plupart des outils modernes permettent le passage entre outils classiques de gestion des exigences (comme **DOORS™**) et outils de modélisation **SysML™** (comme **Modelio**, illustré ci-dessous).

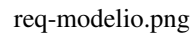


Figure 14.11: Import Modelio de tableau d'exigences

14.9.3 Les *Requirements properties*

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

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

14.9.4 Les *Requirements links*

Les principales relations entre *requirement* sont :

Containment

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

Refinement

Pour décrire un ajout de précision ($\ll refine \gg$), comme par exemple une précision.

Derivation

Pour indiquer une différence de niveau d'abstraction ($\ll deriveReq \gg$), par exemple entre un système et un de ses sous-systèmes.

Tip

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

req-exp1.png

Figure 14.12: 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](#).

topcased-req-connections.png

Figure 14.13: Relations liées au *requirements* dans TOPCASED

14.9.5 Les *Requirements Diagrams*

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

hsuv-reqs1.png

Figure 14.14: Exemples de composition d'exigences

14.9.6 Stéréotyper les *Requirements*

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

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

14.9.7 Annotations des *Requirements*

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

hsuv-reqs2.png

Figure 14.15: Exemples de *rationale* et *problem*

14.9.8 Les considérations sur la traçabilité

Une fois que les *requirements* ont été définis et organisés, il est utile de les lier au moins aux *use cases* (en utilisant <<refine>> par exemple) et aux éléments structurels (en utilisant <<satisfy>> par exemple), mais ceci sera abordé dans la partie Section ??.

Note

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

14.9.9 Les *Use Case Diagrams*

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

req-uc-relation.png

Figure 14.16: Exemple de lien entre *use case* et *requirements*

Ce diagramme est exactement identique à celui d'**UML™**.

UCGestionNotes.png

Figure 14.17: Exemple de diagramme des cas d'utilisation

uc.png

Figure 14.18: Autre exemple de diagrammes des cas d'utilisation

Tip

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.

14.9.10 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 14.3: Déclinaison des Exigences

	Requirements	Structure	Comportement	Transverse
Organisation	\oplus -, <<deriveReq>>			
Analyse	<<satisfy>>, <<refine>>	<<satisfy>> entre reqs et UC	<<refine>>	
Conception	<<allocate>>			
Implémentation	<<satisfy>>, <<verify>>			

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

zigzag.png

Figure 14.19: Exemple de démarche (SYSMOD Zigzag pattern)

14.9.11 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**?

14.10 L'architecture du système

	Requirements	Structure	Comportement	Flux
Organisation				
Analyse				
Conception				
Implémentation				

14.10.1 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)

14.10.2 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 ??).

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

14.10.3 Block Definition Diagrams

14.10.3.1 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™.

pacemaker-context.png

Figure 14.20: bdd du système dans son environnement

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

Properties

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

Operations

Les méthodes supportées par les instances du bloc.

Constraints

Les contraintes (cf. Figure ??)

Allocations

Les allocations (cf. Section ??)

Requirements

Les exigences liées à ce bloc.

User defined

On peut définir ses propres compartiments.

constraints.png

Figure 14.21: Exemple de définition de contraintes

14.10.3.2 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 ??)

references

Les ÈlÈments auquel le bloc a accÈs (via des associations ou des agrÈgations)

constraint properties

Les contraintes que doivent respecter les propriÈtÈs (nous les verrons plus en dÈtail, cf. Section ??).

Note

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

14.10.3.3 Value Types

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

valueType.png

Figure 14.22: DÈfinition de *Value Types*.

14.10.3.4 Associations entre blocs

Il existe deux types de relations entre blocs :

- l'association (y compris l'agrÈgation et la composition)
- la gÈnÈralisation/spÈcialisation

Ces deux types de relations, bien connues en UML™, permettent de matÈrialiser les liens qui existent entre les ÈlÈments du systÈme. Avant d'aborder les associations, il est important de diffÈrencier la description d'ÈlÈments structurels sous la forme d'un bloc (au travers d'un `bdd` par exemple) et ces ÈlÈments pris individuellement. Ces derniers sont des **instances** individuelles du mÊme bloc. Cette notion, trÈs prÈsente dans les approches orientÈes objets est souvent plus ardue ‡ apprÈhender pour les ingÈnieurs systÈmes. Il faut bien comprendre que la modÈlisation d'un bloc consiste ‡ reprÈsenter l'ensemble des ÈlÈments qui caractÈrisent tout une sÈrie d'objets (des moteurs, des pompes, des donnÈes, etc.). Il serait fastidieux de les reprÈsenter tous (individuellement), et c'est donc leur "signature" que l'on reprÈsente. C'est pour cela qu'un bloc n'est pas un ÈlÈment physique, mais simplement sa reprÈsentation, tandis qu'une instance de ce bloc reprÈsentera elle cet ÈlÈment physique. C'est le cas notamment des participants d'un diagramme de sÈquence ou encore des parties d'un composÈ, qui sont des instances et non des blocs.

14.10.3.5 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.

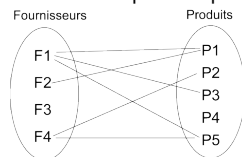
Une association possède plusieurs propriétés :

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 blocs **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 liée toute instance du bloc de départ. Elle est représentée par un couple (M..N).



Caution

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

cardinalite.png

Figure 14.23: Exemple d'association

14.10.3.6 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 données**.

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).

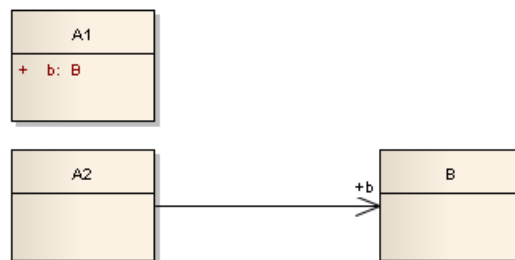


Figure 14.24: Deux façons de représenter une propriété de type B

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 d terminer si une association $A \rightarrow B$ est une association dirig e (navigable dans un sens), une agr gation ou une composition est de raisonner en terme d'impl mentation :

- c'est une agr gation si b est initialis  dans le constructeur de A ;
- c'est une composition si il est aussi d truit dans le destructeur de A ;
- c'est une association dirig e simple si aucun des deux cas pr c dents ne s'applique.

compo.png

Figure 14.25: Exemple de composition

14.10.3.7 G n ralisation/Sp cialisation

Lorsque plusieurs blocs ont des caract ristiques en communs (propri t s, associations, comportement), il peut  tre utile de "factoriser" ces  l ments en un bloc dont les autres vont "h riter". Quand on r alise ces liens hi rarchiques (on utilise souvent le terme "est un") en partant des blocs diff rents pour  tablir un nouveau bloc contenant les points communs on parle de **g n ralisation**.   l'inverse, quand on constate qu'un bloc poss de r ellement plusieurs d clinaisons diff rentes et que l'on cr e alors des blocs sp cifiques, on parle alors de **sp cialisation**.

genspec.png

Figure 14.26: Exemple de lien de g n ralisation/sp cialisation

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

14.10.4 Internal Block Diagrams

Un *ibd* d crit la structure interne d'un bloc sous forme de :

parts

Les parties qui constituent le syst me (ses sous-syst mes)

ports

El ment d'interaction avec un bloc

connecteurs

Liens entre ports

14.10.4.1 Parts

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

**Caution**

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.

parts.png

Figure 14.27: Exemple de *Parts*

parts2.png

Figure 14.28: Autre exemple de *Parts*

14.10.4.2 Ports (SysML 1.2)

**Caution**

La dernière version de la spécification **OMG SysML v1.3** préconise l'abandon des ports tels que définis dans la version 1.2. Nous présentons les nouvelles notions dans la [section qui suit](#).

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

Les ports :

- préservent l'encapsulation du bloc
- matérialise le fait que les interactions avec l'extérieur (via un port) sont 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*^a) et requis (*required*^a) entre les blocs.

Les connecteurs peuvent traverser les "frontières" sans exiger de ports à chaque hiérarchie.

ports-flots.png

Figure 14.29: Exemples de flots

Définition : Ports (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.

flots.png

Figure 14.30: Exemples de flots multi-physique entre ports

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

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

Les Flux peuvent être :

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

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.

14.10.4.3 Ports (SysML 1.3)

La nouvelle spécification **OMG SysML v1.3** introduit les concepts de:

proxy port

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

full port

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

Note

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

14.10.5 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*

14.10.5.1 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

constraints.png

Figure 14.31: Exemple de contraintes

Définition : ConstraintBlock (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.

14.10.5.2 Diagramme paramétrique

C'est une forme particulière de *Internal Block Definition*

param.png

Figure 14.32: Exemple de diagramme paramétrique

14.10.5.3 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*;

blockconf.png

Figure 14.33: Exemple de bloc de configuration

14.10.6 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 ??).

dss.png

Figure 14.34: Exemples de DSS

14.10.7 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 14.4: Place des aspects structurels

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

14.10.8 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 ?

14.11 Le comportement du système

	Requirements	Structure	Comportement	Environnement
Organisation				
Analyse				
Conception				
Implémentation				

14.11.1 Fondements

On abordera :

- les *Use Case Diagrams*

- les *Sequence Diagrams*
- les *State Machines*
- les *Activity Diagrams*

14.11.2 Use Case Diagrams

Les Éléments de base :

Acteurs

Les principaux Éléments extérieurs au système considéré, et participant qui participent (on parle parfois d'acteurs principaux). Ils ont souvent un rôle. 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)

Tip

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.

14.11.3 Le Diagramme des Cas d'Utilisation

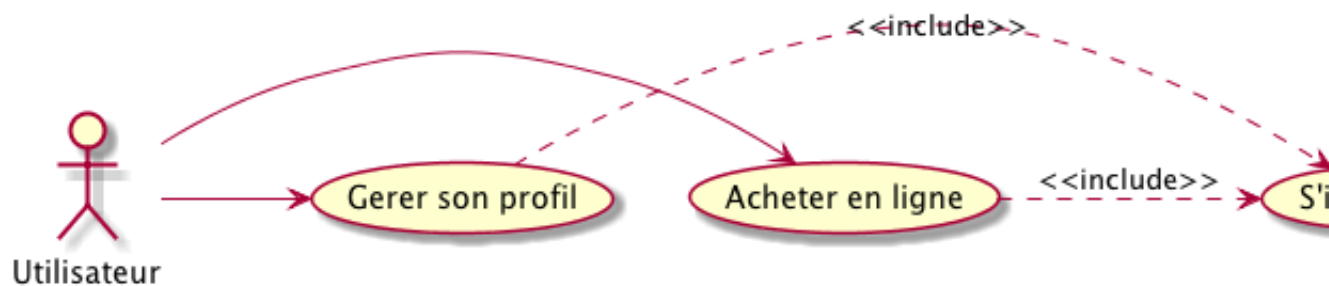
Le **Diagramme des Cas d'Utilisation** est un diagramme **UML™** permettant de représenter :

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

14.11.3.1 Cas d'Utilisation (Use Case)

Exemple de cas d'utilisation

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



14.11.3.2 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)

14.11.3.3 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 sd).

14.11.3.4 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 ??)
 - les acteurs principaux
 - les acteurs secondaires
- exemples de types d'acteurs définis 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.

14.11.3.5 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.

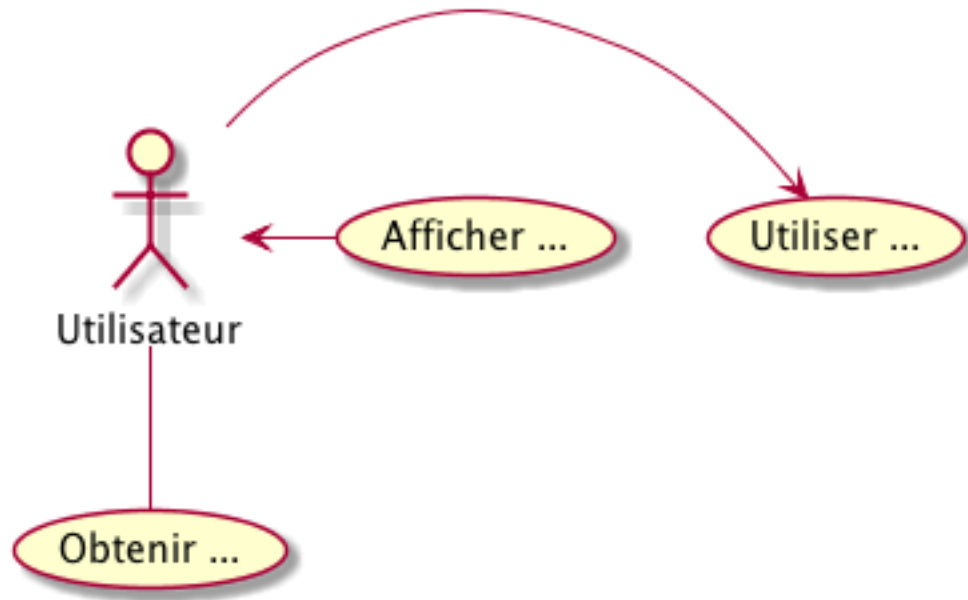


Figure 14.35: Relations orientées

14.11.3.6 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)

Diagramme d'UC

Figure 14.36: Notation dans le diagramme d'UC

Tip

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

14.11.3.7 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 Tim Weilkiens) préconisent de ne pas utiliser les acteurs et les cas d'utilisation (cf. son [blog](#))

14.11.3.8 Exemples complets (UML) : Gestion des notes

Exemple de Diagramme d'UC

Figure 14.37: Autre exemple de diagramme d'UC

14.11.4 Sequence Diagrams

14.11.4.1 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, ...)

**Caution**

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

14.11.4.2 Exemple

dsxpl.png

Exemple de diagramme de séquence

Figure 14.38: Exemple de diagramme de séquence (3)

14.11.4.3 Notions avancées

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

- `loop` (boucle)
- `alt` (alternative)
- `opt` (optionel)
- `par` (parallèle)

- *region* (région critique - un seul *thread* à la fois)

Un algorithme

Figure 14.39: Exemple d' algorithme...

Sa modélisation

Figure 14.40: Et le diagramme correspondant

14.11.4.4 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 détecter 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 ??) et la version "objet" (Figure ??).

Conception 'centralisée'

Figure 14.41: Conception "centralisée"

Conception 'objet'

Figure 14.42: 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 ??).

14.11.4.5 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.

Diagramme d'UC

Figure 14.43: Diagramme d'UC

Le DSS correspondant

Figure 14.44: Le DSS correspondant

Le DS correspondant

Figure 14.45: Le DS correspondant

14.11.5 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.

Un diagramme d'État

Figure 14.46: 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 États. 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) ‡ 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.).

14.11.6 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 ‡ préciser la séquence d'actions ‡ réaliser, mais aussi ce qui est produit, consommé ou transformé au cours de l'exécution de cette activité.

act-pcmk1.png

Figure 14.47: 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.

14.11.7 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.

14.11.8 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 **flots d'objets** (reliant une action et un objet consommé ou produit). Les *object flow*, associées aux broches d'entrée/sortie (*input/output pin*) permettent alors de décrire les transformations sur les objets manipulés.

Un flot continu

Figure 14.48: Un exemple de flot continu (UK)

Pour permettre la modélisation des **flots continus**, SysML™ ajoute à UML™ la possibilité de caractériser la nature du débit qui circule sur le flot : continu (par exemple, courant électrique, fluide, etc.) ou discret (par exemple, 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.

14.11.9 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 ??) :

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.

flow-ctrl.png

Figure 14.49: Les différents contrôles de flow SysML

14.11.10 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.

act-call.png

Figure 14.50: Exemple de *callBehaviorAction* (UK)

14.11.11 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 ??).

Table 14.5: Place du Comportement

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

14.11.12 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 ?

14.11.13 Exercices

14.11.13.1 Diagramme des cas d'utilisation

Placez dans un diagramme 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.

14.12 Les aspects transversaux

Table 14.6: Aspects transversaux

	Requies	Structur	Comportement	Transverse
Organisation				
Analyse				
Conception				
Implémentation				

14.12.1 Fondements

On abordera ici les aspects transversaux comme :

- la traçabilité des exigences
- les mécanismes d'allocation

- le diagramme paramétrique

14.12.2 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 14.7: Traçabilité

	Requirements	Structure	Comportement
Requirements	deriveReq t>>, <<ref ine>>, <<cop y>>		
Structure	<<all ocat e>>, <<saf isf y>>		<<all ocat e>>
Comportement	ref ine>>		

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

14.12.3 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 ou les diagrammes d'activités par exemple. Parler des `<<allocate>>`.

14.12.4 Diagramme paramétrique

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

param.png

Figure 14.51: Exemple de diagramme paramétrique

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

Note

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

14.12.5 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 États) est attaché à un élément bien précis (par exemple un bloc). Ces liens apparaissent entre blocs et machines à États, entre cas d'utilisation et diagrammes de séquence ou d'activité, etc.

14.12.6 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 ?

Part IV

Partie 4 : Modéliser un système en SysML

Chapter 15

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 [\[SeeBook2012\]](#), ou 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.

15.1 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 ??). Il faut penser à la documentation (cf. Section ??).

Outillage autour de SysML

Figure 15.1: Outillage autour de SysML

15.1.1 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.

15.1.2 G  n  ration de documentation

La plupart des outils permettent de g  n  rer de la documentation. Pour les outils bas  s [eclipse](#) comme [TOPCASED](#), il est possible d'utiliser le plug-in `GenDoc2`.

GENDOC

Figure 15.2: G  n  ration de documentation ‡ partir de TOPCASED (1)

GENDOC

Figure 15.3: G  n  ration de documentation ‡ partir de TOPCASED (1)

Les outils commerciaux comme [Rhapsody](#) permettent de g  n  rer de nombreux formats.

Rhapsody

Figure 15.4: G  n  ration de documentation ‡ partir de Rhapsody

15.1.3 Animation de mod  les et simulation

Fortement li  e aux outils, la possibilit   d'animer les mod  les ou encore d'effectuer des simulations est une exigence de plus en plus forte des ing  nieurs syst  mes.

Il existe de nombreuses possibilit  s. Citons par exemple :

G  n  ration de code VHDL

L'outil `RTaW` propose, via g  n  ration de code VHDL de simuler les mod  les. Voir une d  monstration [ici](#).

Simulation en Rhapsody

L'outil [Rhapsody](#) poss  de une interface tr  s pratique pour faire du prototypage rapide.

Animation

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

Animation de modÈles en Artisan

L'outil *Artisan* permet Également de faire de l'animation de modÈles.

Animation

Figure 15.5: Animation Artisan

15.2 SpÈcification du systÈme

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

15.3 Conception du systÈme

Chaque cas d'utilisation sera prÈcisÈ (*seq* et *act*). Les donnÈes mÈtier seront alors identifiÈes pour construire le modÈle d'architecture logique (*bdd* et *ibd*) complÈtÈ par la description des comportements complexes (*stm*). Enfin le modÈle d'architecture physique permettra de dÈterminer les aspects d'Éploiement et constructions physiques d'Équipements/

15.4 TraÁabilitÈ et Allocations

Afin de consolider les diffÈrents modÈles, les liens de traÁabilitÈ qui n'auront pas ÈtÈ dÈj‡ dÈcrit ¹ seront rajoutÈs en insistant sur les liens :

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

15.5 ModÈle de test

Nous insistons dans l'ensemble de nos formations sur les approches *test-driven*, alors nous montrons dans cette section comment participer ‡ la qualitÈ du dÈveloppement d'un systÈme en formalisant (par exemple avec des diagrammes de sÈquence de scÈnarios ‡ Éviter) les test et les jeux de test.

¹ Il est recommandÈ de ne pas attendre pour matÈrialiser ces liens, mais de les exprimÈs dÈs que rencontrÈs dans telle ou telle modÈlisation.

Chapter 16

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 exhaustive de recettes. Les choix des éléments de modélisation sont arbitraires ou tirés de discussions (comme ce sera mentionné si c'est le cas).

16.1 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 ??.

Le contexte du Pacemaker

Figure 16.1: Le contexte du Pacemaker ([\[SeeBook2012\]](#))

16.2 Comportement

Note

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

Part V

Partie 5 : PÈdagogie

Chapter 17

Enseigner SysML

Cette partie est dédiée à l'enseignement de SysML™. J'y distille des conseils et des remarques issues des nombreuses questions soulevées dans le cadre des journées SysML-France ou UPSTI.

17.1 Respect des notations SysML

Je recommande vraiment l'utilisation d'outils (même de dessins, mais dédiés à SysML™). Ils respectent en général la norme **OMG SysML v1.3** (bien qu'il faille se méfier). Eviter de "dessiner" des diagrammes.

Par contre la norme permet de faire des adaptations graphiques (cf. la discussion <http://www.realtimeatwork.com/-2011/08/is-sysml-too-abstract/>).

17.2 Diagramme de bloc

Par exemple dans ce diagramme les blocs ne respectent pas la syntaxe graphique de SysML™ :

Erreur : mauvais symboles graphiques pour les blocs

cordeuseContext.png

Pour rappel, la notation `jmb :Personne` permet de représenter un **objet** (une instance d'une classe ou d'un bloc). C'est donc une notation utilisée par exemple dans les participants d'un diagramme de séquence ou encore les parties d'un diagramme interne de bloc.

Donc dans le diagramme ci-dessus, l'acteur est correct (on peut mettre des acteurs dans un bdd, cf. OMG SysML v1.3 p.32), par contre les objets `Block : . . .` est une erreur de notation.

Solution : utiliser un outil (ici BOUML)

plantuml/cordeuseContext.png



Warning

Attention, il est tout à fait possible de représenter des instances dans un bdd (cf. OMG SysML v1.3 p.34), même si c'est très peu courant.

17.3 Diagramme de séquence

Erreur : pb avec les participants et la boucle

cordeuseSeqBad.png

Plusieurs problèmes de non respect de la notation :

- il manque le rectangle aux participants
- les participants semblent être des blocs et non des instances
- la boucle devrait avoir une condition (même "toujours" pour une boucle infinie)

Note

Le dernier problème est plus une convention qu'une véritable erreur. Cf. Chapter ??.

17.4 Diagramme des cas d'utilisation

17.4.1 Utilisation du système

Problème : Fournir n'est pas Obtenir...

ucBad.png

Il vaut mieux définir les cas d'utilisation du point de vue de (ou des) utilisateurs plutôt que du système. Cf. Chapitre ???. Pour rappel, un cas d'utilisation est un regroupement de scénarios qui correspondent à un but d'un des acteurs (dans le domaine du problème considéré et selon la granularité envisagée).

Dans le diagramme ci-dessus il aurait fallu écrire :

Solution : Prendre le point de vue de l'utilisateur

plantuml/uc7.png

17.4.2 Interaction avec les cas d'utilisation principaux

Diagramme des Cas d'Utilisation erroné

image001.jpg

Erreur : oubli d'interaction

Dans le diagramme [?note], l'acteur Utilisateur n'interagit ni avec le cas Tendre la corde ni avec Positionner raquette !

Solution (extrait) : Modifier le diagramme en conséquence

plantuml/uc1.png

La solution n'est que partiellement satisfaisante (cf. point suivant).

Note

J'ai fait disparaître le lien entre Tendre la corde et Corde, mais uniquement car j'ai déduit de l'exemple que Programmer n'interagissait pas avec Corde. Et donc que Tendre la corde n'interagit avec Corde que dans le sous-cas d'utilisation Maintenir la tension. C'est typiquement une ambiguïté que le "client" doit lever.

Tip

L'utilisation de stéréotypes comme <<TopLevel>> ou <<Principal>> peut permettre d'éviter d'interagir avec ces cas d'utilisation.

17.4.3 Utilisation des <<include>>

Il faut faire attention à ne pas abuser des <<include>>. Par exemple ma recommandation en SysML/UML est de ne jamais avoir à :

Problème : mauvaise utilisation de l'<<include>>

plantuml/uc2.png

Dans la figure [?note], il n'y a aucune raison de ne pas inclure le cas d'utilisation s'identifier directement dans le cas d'utilisation Acheter en ligne. Et avoir ainsi :

Solution 1 : On englobe les <<include>> "isolés"

plantuml/uc3.png

J'enseigne qu'un <<include>> devrait toujours concerner un cas inclut dans plusieurs autres, comme la figure [?note] :

Solution 2 : On "mutualise" les <<include>>

plantuml/uc4.png

Néanmoins je vois deux raisons valables pour décomposer les cas d'utilisation avec des <<include>> qui se retrouvent isolés :

1. Pour indiquer que seulement une partie du cas d'utilisation principal interagit avec un acteur (secondaire la plupart du temps). C'est ce qui est fait dans [?note].
2. Pour faire de la décomposition fonctionnelle (cf. point suivant).

17.4.4 Niveau de détails des UC

Faut-il minimiser le nombre de cas d'utilisation ou au contraire détailler? Normalement un bon diagramme des UC est indépendant de la solution, il exprime plutôt le problème (les attentes).

Néanmoins dans le cadre de l'enseignement en prépa comme support graphique à une analyse fonctionnelle, pourquoi pas détailler. À ce moment-là, utiliser des stéréotypes pour différencier les cas d'utilisation (<<TopLevel>> et <<Operational>> par exemple comme dans la documentation SysML 1.3 pp 185-186 sur le HybridSUV).

Note

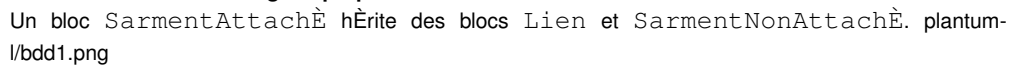
La question de l'utilisation des cas d'utilisation pour exprimer les activités du système reste ‡ trancher. Le diagramme des activités étant tout de même plus adapté a priori (cf. Chapter ??). Le principal défaut du diagramme [?note] est surtout de mélanger des cas d'utilisations de niveaux différents. N'oublions pas que derrière chaque UC, il devrait y avoir un but d'une partie prenante.

17.5 Diagrammes de bloc

17.5.1 Héritage

Attention ‡ la notion d'héritage, complexe ‡ appréhender. On ne peut surtout pas dire :

Erreur : Confondre héritage et propriété

Un bloc `SarmentAttaché` hérite des blocs `Lien` et `SarmentNonAttaché`.  `plantuml/bdd1.png`

La relation doit pouvoir se lire "est un". Or, un `SarmentAttaché` n'est pas un `SarmentNonAttaché` (c'est même le contraire)!

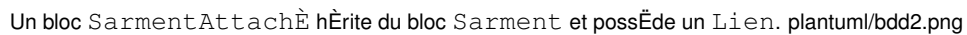
Lorsque plusieurs blocs ont des caractéristiques en communs (propriétés, associations, comportement), il peut être utile de "factoriser" ces éléments en un bloc dont les autres vont "hériter". Quand on réalise ces liens hiérarchiques (on utilise souvent le terme "est un") en partant des blocs différents 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**.

 `genspec.png`

Figure 17.1: Exemple de lien de généralisation/spécialisation

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

Solution

Un bloc `SarmentAttaché` hérite du bloc `Sarment` et possède un `Lien`.  `plantuml/bdd2.png`

17.5.2 Cardinalités

Attention aux cardinalités indiquées dans les associations.

Erreur : Le système est composé de 32 propulseurs!

cardBad.png

Il ne s'agit pas d'une erreur de syntaxe SysML, mais d'une erreur de conception. Un Alistar est composé de 8 Propulseurs et un Propulseurs est composé de 4 Propulseurs arrières. $4 \times 8 = 32$.

Solution possible

plantuml/bdd4.png plantuml/bdd5.png

17.6 Diagramme d'États

17.6.1 Différence entre UML et SysML sur les machines à État.

SysML a repris (quasiment, cf. plus bas) tel quel le diagramme d'États UML :

Définition : State Machines (OMG SysML v1.3, p. 189)

SysML reuses many of the major diagram types of UML. In some cases, the UML diagrams are strictly reused, such as use case, sequence, state machine, and package diagrams, whereas in other cases they are modified so that they are consistent with SysML extensions.

Il y a une exception près, les *protocol state machines* qui ont été retirées pour des raisons de simplification :

Définition : State Machines (OMG SysML v1.3, p. 119)

The UML concept of protocol state machines is excluded from SysML to reduce the complexity of the language.

17.6.2 Lien avec le Grfcet

Le **Grfcet** étant plus proche des Réseaux de Petri, la correspondance la plus proche serait peut-être le diagramme d'activité.

Voir aussi la FAQ [Comment traduire un grfcet en machine à État ?](#).

17.7 Divers

17.7.1 Du danger d'une lecture trop rapide de la norme

C'est important de faire référence à la norme quand on avance un fait. J'essaye de m'y atteler personnellement dans mes écrits. Néanmoins il faut faire attention car on fait souvent des citations qui finalement ne sont que des extraits.

Erreur : citation sortie de son contexte

"...[SysML] limite à 1 le nombre de régions dans un état composite (note de bas de page p. [17 de la norme]...)."

Cette partie de la norme **OMG SysML v1.3** mentionne effectivement cette phrase, mais comme un exemple de notes qui peuvent se retrouver dans des manuels d'outils qui ne respecteraient pas la norme justement!!

Solution : Faire attention au contexte (OMG SysML v1.3, p. 17)

"In the case of "PARTIAL" support for a compliance point, in addition to a formal statement of compliance, implementors, and profile designers must also provide feature support statements."

Un autre exemple issu d'**UML™** où en fait la norme parle d'une convention adoptée pour ses propres meta-modèles :

Erreur : citation sortie de son contexte

If no multiplicity is shown on an association end, it implies a multiplicity of exactly 1 (UML Superstructure Specification v2.4.1 p.18).

Il faut donc bien faire attention avec les extraits de norme.

Tip

Quand on cite un extrait de la norme, citer le numéro de page du document papier et non celui du PDF.

Part VI

Partie 6 : Pour aller plus loin

Chapter 18

Considérations méthodologiques

Exemples de démarche autour de SysML™, (cf. Chapter ??).

Chapter 19

Analyses et simulation

To be completed

Chapter 20

Exercices de rÈvision


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


20.1 Quizz


20.1.1 Sujet

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


En voici une capture d'Ècran :

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 20.1: Exemple de QCM sur SysML

20.1.2 Corrigé

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

20.2 Mots croisÈs

20.2.1 Sujet

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

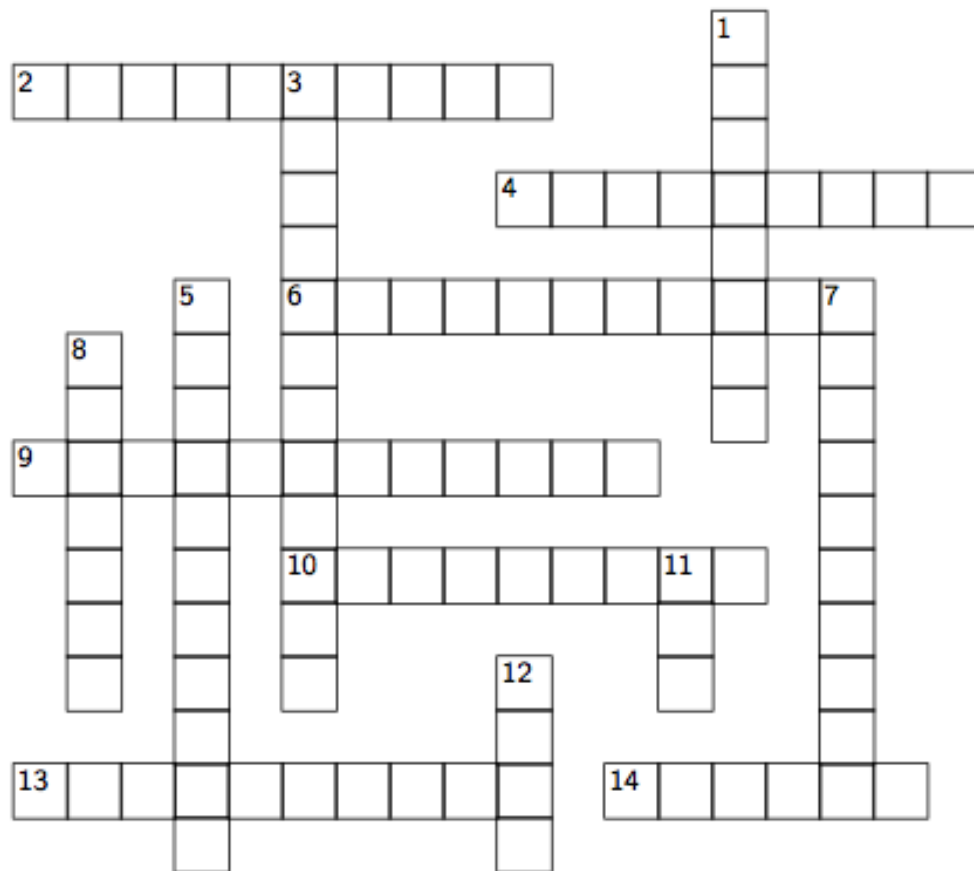


Figure 20.2: Mots-croisÈs sur SysML

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

Part VII

Annexes

Chapter 21

Liens et références 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)
- Spécial STI2D et UPSTI
 - [Programme du BO](#)
 - [Le forum STI2D](#)
 - [Slides STI2D](#)
- 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 g n raux 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 g n ration des documents ‡ partir d’[AsciiDoc](#).
- Divers
 - Pour en savoir plus sur l’[auteur](#)

Chapter 22

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.

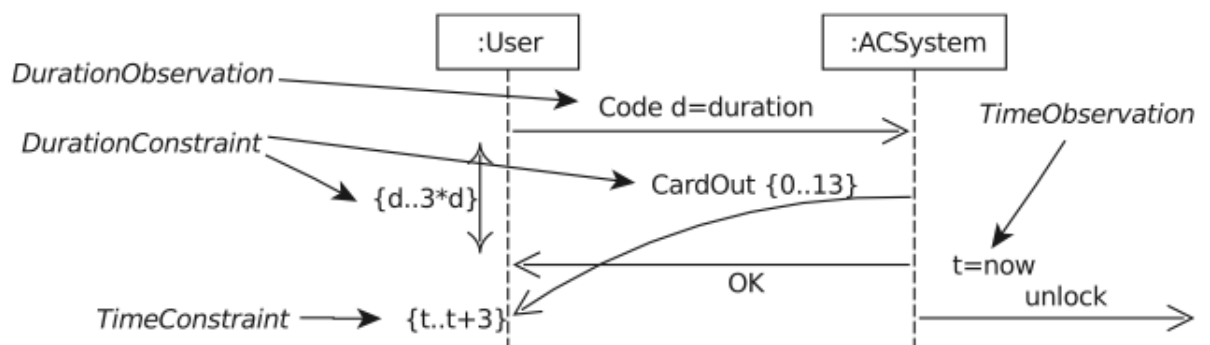


Figure 22.1: Exemple de contrainte temporelle (tirée de [SysML])

Néanmoins, pour une prise en compte industriel des contraintes temporelles, il conviendra d'utiliser le profil d'ÉdÉ ces aspects : le profil **MARTE**.

Chapter 23

Conventions et recommandations

Il existe un certain nombre de conventions complémentaires aux règles formelles de syntaxe que l'on trouve dans la spécification elle-même (rappelées à la fin de cette section). 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 dans lequel vous pourrez piocher pour définir **vos propres conventions**.

Note

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

23.1 Points de vue

1. Les cas d'utilisation concernent les utilisateurs du système et non le système lui-même. Ainsi les cas d'utilisation Obtenir les coordonnées actuelles ou Enregistrer une trace sont de bons cas d'utilisation d'un GPS. Alors que Economiser la batterie ou Crypter les données ne sont pas de bons cas.
2. Eviter de placer les matières premières comme des acteurs dans un diagramme d'utilisation, mais plutôt (Éventuellement) comme des Éléments de diagramme de définition de bloc (principalement celui de contexte).

23.2 Cas d'utilisation et acteurs

1. Placer les acteurs principaux à gauche (e.g., [?note]).

2. Placer les acteurs secondaires à droite (e.g., [?note]).
3. Différencier les acteurs humains (*stickman*) des autres (stereotype <<actor>>) e.g., [?note].
4. Différencier les cas d'utilisation selon :
 - leur importance (e.g., <<Principal>>, <<Secondaire>>);
 - leur type (e.g., <<TopLevel>>, <<Operational>>).

23.3 Nommage des divers Éléments

1. Les noms de blocs commencent par une majuscule (source : [UML™](#)).
2. Les noms de cas d'utilisation (qui représentent une façon d'utiliser le système du point de vue de l'acteur) doivent être un verbe à l'infinitif (source : [UML™](#)).
3. Les noms d'activité (qui représentent une action) doivent être un verbe à l'infinitif (source : [UML™](#)).
4. Les noms d'attributs commencent par une minuscule et ne sont pas au pluriel (source : [UML™](#)).
5. Nommer les "associationEnd" du point de vue du propriétaire et en minuscule (source : [\[ENS2013\]](#)).

23.4 Les requirements

1. Différencier les exigences descriptives et prescriptives. On pourra par exemple utiliser les stéréotypes <<UserRequirement>> et <<SystemRequirement>>. On pourra aussi définir deux *packages* différents : *DescriptiveRequirements* et *PrescriptiveRequirements* (source : [\[SysML4STI2D\]](#))

23.5 Les dépendances

1. En général un cas d'utilisation qui n'est inclus (<<include>>) que dans un seul autre cas est fusionné dans ce dernier.
2. 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 (source : [\[SysML4STI2D\]](#)).

23.6 Blocs et associations

1. Utiliser la référence (faible agrégation) avec parcimonie car c'est un concept complexe à appréhender, surtout pour des non informaticiens (source : [\[ENS2013\]](#)).

23.7 Divers

1. Toujours mentionner les multiplicités pour éviter toute ambiguïté, et de manière plus générale, éviter au maximum de se contenter des conventions par défaut (comme les [officielles](#)), sources d'incompréhensions potentielles.
2. Toujours indiquer les conditions des `loop`, `alt`, etc. dans un diagramme de séquence.
3. Vérifier la complétude et la non intersection des conditions des transitions sortant d'un même état.
4. Ne pas hésiter à ajouter des légendes (sous forme de notes par exemple) à vos diagrammes un peu complexe.

23.8 Conventions "officielles" (dans la spécification elle-même)

1. Les classes représentées dans un bdd sont par défaut des `<<block>>` (source : [OMG SysML v1.3](#), p. 38).
2. En l'absence de multiplicité sur les connecteurs, la multiplicité par défaut est "1" de chaque côté (source : [OMG SysML v1.3](#), p. 42).
3. En l'absence de multiplicité sur les associations unidirectionnelles (flèches), la multiplicité par défaut est "1" (source : [OMG SysML v1.3](#), p. 62).

Chapter 24

FAQ

Cette FAQ (*Frequently Asked Questions*) a ÉtÉ construite par expÈrience, en regroupant les questions des Étudiants durant mes diffÈrentes interventions. J'ai aussi ajoutÈ des questions souvent rencontrÈes dans les journÈes organisÈes par [SysML-France](#).

Note

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

Cette FAQ peut servir de base ‡ la rÈvision d'examens (cf. aussi Chapter ??).

24.1 Quelle est la version courante de la spÈcification et comment l'obtenir?

Version 1.3 et disponible ‡ l'URL: <http://www.sysml.org/docs/specs/OMGSysML-v1.3-12-06-02.pdf>

24.2 Quels en sont les changements notables depuis la derniÈre version ?

(en lien avec la question prÈcÈdente)

Les changements notables par rapport ‡ 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 de *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.

24.3 Quelle est la différence entre les différents ports : proxy, nested, flow, ... ?

La version 1.3 a précisé la sémantique des ports et abandonne le concept de *flow specification* au profit d'une description via un bloc possédant des *flow properties* (cf. [Flow properties](#)).

Préciser les propriétés des Flow Ports

flowport2.png flowport1.png

Ainsi :

- Les ports par défauts (hérités d'**UML™**) permettent d'indiquer des services fournis et requis.
- Les ports de type flot (*flow*) spécifient que quelque-chose "circule" dans ou depuis un bloc.
- Si le bloc représentant le port possède lui-même des ports, on parle alors de *nested ports* (cf. [Nested Ports](#)).
- Enfin, un *proxy port* permet de donner accès à l'extérieur d'un bloc à l'un des ports de l'un de ses composants.

nestedPort.png

Figure 24.1: *Nested Ports*

Note

Voir aussi les blogs :

- <http://model-based-systems-engineering.com/2012/04/02/whats-new-in-sysml-1-3/>
 - <http://model-based-systems-engineering.com/2013/09/23/sysml-full-ports-versus-proxy-ports/>
-

24.4 Qu'est-ce que la certification SysML et comment la passer?

Les informations sur la certification sont sur le [site d'ÉdÉ](#) de l'OMG™.

La certification (qui coÉte environ 300€ par niveau) permet de garantir que celui qui la possÉde maÓtrise la notation. Il faut actuellement s'inscrire via le site de [Pearson Vue](#).

Tip

Il faut vraiment maÓtriser les concepts eux-mÉme (le mÉtamodÉle), et pas seulement avoir une bonne pratique. De plus, il faut Également bien maÓtriser l'anglais car les questions/rÉponses sont en anglais et leur formulation peut comporter des piÉges.

Pour vous tester sur le mÉta-modÉle, vous pouvez vous tester sur le quizz rÉalisÉ par [LoÓc FÉjoz](#) : <http://www.realtimeatwork.com/2010/06/test-quizz/>

24.5 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 modÉle, c'est bien le dernier *containment* rÉalisÉ qui est pris en compte. Dans la figure ci-dessous le lien A-C a ÉtÉ "dessinÉ" aprÉs celui B-C.

topcased-containment-1.png

Figure 24.2: Exemple de divergence modÉle/diagramme (diagramme)

topcased-containment-2.png

Figure 24.3: Exemple de divergence modÉle/diagramme (modÉle)

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.

24.6 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.

topcased-containment-3.png

Figure 24.4: 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.

24.7 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 `<<deriveReq>>`.

Note

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

24.8 Quelle est la différence entre `<<deriveReq>>` et `<<refine>>` ?

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

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

Autre exemple respectant 1 mais pas 2 : "Le véhicule doit posséder 4 roues." est dérivée 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).

24.9 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'utiliser une de ces versions plus précises (`<<deriveReq>>` ou `<<satisfy>>` par exemple).

24.10 Comment SysML permet-il la validation et la vérification des exigences ?

Comment SysML™ permet de vérifier et de valider les exigences ?

La **validation** d'une exigence est de la responsabilité des parties prenantes. À partir de la spécification des exigences, ils valident qu'il s'agit bien du bon produit ‡ construire. Typiquement, le diagramme des exigences sert de base ‡ cette validation.

La **vérification** d'une exigence est de la responsabilité de l'ingénieur système et/ou de l'analyste système. C'est ‡ eux de montrer la correspondance entre les éléments constituant du système et les exigences spécifiées. C'est principalement pour cette activité que sont utilisés les relations `<<satisfy>>` et `<<verify>>`.

Requirements-VV.png

Figure 24.5: Validation et Vérification des exigences (Reproduced by Permission © 2003-2013 PivotPoint Technology Corp)

24.11 ¿ quoi sert un diagramme des UC avec un seul cas d'utilisation ?

plantuml/uc6.png

Tout simplement ‡ relier les autres diagrammes ‡ ce cas d'utilisation. Par exemple le comportement du systÈme, l'architecture, etc. (les solutions) pourront Être reliÈes (<<satisfy>>) ‡ ce cas. Il faut aussi ne pas perdre de vue qu'un diagramme peut Èvoluer. On pourra trÈs bien rajouter au diagramme des cas non encore pris en compte comme Transporter le systÈme, Recycler le systÈme, etc.

24.12 Comment faut-il comprendre "interaction" dans les diagrammes d'UC ?

La dÈfinition de la norme **OMG SysML v1.3** :

DÈfinition : Actors (OMG SysML v1.3, p. 123)

... Actors represent classifier roles that are external to the system that may correspond to users, systems, and or other environmental entities. They may interact either directly or indirectly with the system ...

Pour la comprendre il ne faut pas utiliser le Larousse franÁais (qui rend presque le caractÈre rÈciproque obligatoire), mais plutÙt la comprendre dans son acception informatique (comme dans "Interaction Homme-Machine"). Par exemple un message (appel de mÈthode) d'un ÈlÈment vers un autre dans un diagramme de sÈquence est appelÈ en **SysML™** une *interaction*.

SysML™ permet de **prÈciser le caractÈre rÈciproque** ou non de l'interaction par exemple entre un Acteur et un Cas d'utilisation :

plantuml/uc5.png

Figure 24.6: Trois types d'interaction diffÈrentes possibles en SysML

24.13 Les "matiÈres premiÈres" font-elles parties des acteurs ?

On pourrait les considÈrer comme des acteurs en se fiant ‡ la dÈfinition (cf. **Acteurs**) et en les associant ‡ des *environmental entities*. Mais elles sont ÈchangÈes avec les entitÈs de l'environnement,

ce qui n'est pas la même chose. Autrement dit, il faut indiquer qui fournit les matières premières et qui recueille les matières produites éventuellement en tant qu'acteurs, pas les matières elles-mêmes, qui seront représentées comme des flux échangés ou des blocs. Eventuellement on peut les retrouver dans le diagramme de contexte ou encore dans un diagramme structurel comme le diagramme de bloc interne.

Tip

Je recommande la lecture (anglaise, *sorry*) de l'excellent plaidoyer pour la mort des acteurs : <http://model-based-systems-engineering.com/tag/sysml/>.

**Warning**

Donc non, le soleil et le vent ne sont pas des acteurs!! (ni la corde, ni la raquette)

24.14 Pour l'analyse fonctionnelle : diagramme des UC ou des activités ?

Pour alimenter le débat, je renvoie aux exemples donnés par Loïc FÉjoz lors de la journée UPSTI. Voici un diagramme classique FAST :

fast.png

Figure 24.7: Diagramme FAST

sysml-fast.png

Figure 24.8: Représentation possible en SysML

On pourrait croire que ces schémas sont identiques mais pas du tout. SysML™ présente juste l'organisation du modèle et non l'arbre d'appel. De plus, FAST a tendance ‡ focaliser sur une solution unique alors que SysML™ permet de repousser ce choix voir d'explorer des alternatives.

24.15 Quelle est la différence entre un *join* de type "OU" et un *fusion* dans un diagramme d'activités ?

Pour rappel le *join* est utilisé pour représenter la synchronisation (le "rendez-vous") et représente donc une conjonction "ET". Mais le comportement par défaut pouvant être modifié par un '`<<joinSpecification>>`', on peut indiquer un "OU" pour signifier l'attente d'un seul des deux jetons. Mais justement le *fusion* est là pour ça normalement. Ces 2 concepts étant issus d'**UML™**, il nous faut nous référer à la norme de ce dernier:

Définition : Types de diagrammes (UML Superstructure v2.4.1 p. 399)

All tokens offered on incoming edges are offered to the outgoing edge. There is no synchronization of flows or joining of tokens.

Il n'y a donc sémantiquement aucune différence.



Warning

Attention, il n'y a pas beaucoup de sens à utiliser un *join* de cette façon, car non seulement il ne "joint" plus rien, mais en plus contrairement à un *join* qui "consomme" les 2 jetons, une telle utilisation aurait pour effet de laisser les 2 jetons passer, mais l'un après l'autre!

24.16 Un système peut-il avoir plusieurs États?

On lit souvent qu'il n'y a pour un système ou un composant "qu'un seul État actif à la fois".

Du point de vue structurel, si l'on considère que l'État à un instant t d'un composant est représenté par la valeur de ses attributs à cet instant, alors cette composition est unique. Néanmoins cela prête souvent à confusion avec le fait qu'un composant dont le comportement est décrit par une machine à États avec des régions concurrentes peut se retrouver dans plusieurs États (**SysML™** du coup) en même temps.

Il faut donc bien faire la différence entre :

- l'**État du système** (en tant qu'association de valeurs d'attributs à un instant T) qui lui est unique à un instant donné, et
- l'**État d'une machine à États** (en tant qu'abstraction d'un ensemble d'États au sens précédent).

Exemple d'un système ayant plusieurs États, au sens des machine à États (fourni par <mailto:loic.fejoz@realtimeatwork.com>[Loïc FÉjoz])

Soit un système avec une unique variable *i*. Les États du système seraient par exemple l'ensemble des entiers naturels (positifs). Dans une machine à État UML, on pourrait par exemple avoir l'État *pair* et l'État *impair*. Mais on pourrait aussi avoir, deux-sous États (*nul* et *non-nul*) et rajouter en concurrence (*multiple de 4* ou *non*, etc.).

24.17 Comment traduire un Grafcet en machine à État ?

En automatique, la notation la plus connue pour représenter la dynamique d'un système est le **Grafcet**. Il semble qu'il existe une synthèse du passage **Grafcet** \Rightarrow diagramme d'État. J'attends de le trouver pour l'intégrer ici. Car sans être un spécialiste, ce que j'en ai lu me fait plutôt penser à un diagramme d'activité (notamment par leur proximité à tous les deux avec les Réseaux de Petri).

24.18 Divers

Quelques autres questions que je laisse à votre sagacité :

- Pourquoi les ingénieurs systèmes auraient-ils besoin d'un n-ième langage de modélisation ?
- 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?

Chapter 25

FABQ

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

25.1 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.).

25.2 Comment exécuter un diagramme SysML?

Les diagrammes SysML™ ne sont que des dessins, des représentations graphiques, ils ne s'exécutent donc pas. Bien sûr de nombreux travaux et outils portent sur les modèles exécutables. Il s'agit alors pour un outil de proposer de reproduire la dynamique d'un diagramme. Il s'agit en général de diagramme de comportement (par exemple une machine ‡ État) pour lequel l'outil propose de simuler l'arrivée d'événement et de reproduire (plus ou moins graphiquement) le comportement modélisé (par exemple le franchissement d'une transition).

Chapter 26

Références

Voici quelques références utiles.

- [ENS2013] L. Gendre et J.-M. Virely, SysML. Tutoriel du 13/06/2013. ENS Cachan.
- [FIO2012] Fiorèse S., Meinadier J., Découvrir et comprendre l'ingénierie système, AFIS 2012.
- [FMS] A. Moore, R. Steiner, S. Friedenthal, A Practical Guide to SysML, The MK/OMG Press, MK/OMG Press, 2011 (2nd edition).
- [HAS2012] Haskins C., SE Handbook Working Group, INCOSE Systems Engineering Handbook: Version 3.2.2, International Council on Systems Engineering, 2012.
- [KAP2007] Kapurch S., NASA Systems Engineering Handbook, 2007 ([pdf](#)).
- [MeDICIS] ENSI Bourges/PRiSM.
- [REQ2012] Guide Bonnes Pratiques en Ingénierie des Exigences, AFIS 2012.
- [Roques2010] [Pascal Roques](#). SysML par l'exemple - Un langage de modélisation pour systèmes complexes. Eyrolles. [acheter ici](#).
- [SeeBook2012] Embedded Systems Analysis and Modeling with SysML, UML and AADL, F. Kordon, J. Hugues, A. Canals, A. Dohet, Wiley, 2013.
- [Sommerville1997] Ian Sommerville, Pete Sawyer. Requirements Engineering: A Good Practice Guide. Wiley, 1997.
- [SysML] OMG. Systems modeling language version 1.3. Technical report, 2012.
- [taoup] Eric Steven Raymond. *The Art of Unix Programming*. Addison-Wesley. ISBN 0-13-142901-9.

- [Walsh1999] Norman Walsh & Leonard Muellner. DocBook - The Definitive Guide. O'Reilly & Associates. 1999. ISBN 1-56592-580-7.
- [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
- [Styles] Scott W. Ambler. The Elements of UML 2.0 Style. Cambridge University Press, 2005. ISBN: 0-521-61678-6
- [Fowler2004] Martin Fowler. UML 2.0 INFORMATIQUE PROFESSIONNELLE, 2004.
- [Fejoz2013] Loïc Fejoz. SysML4STI2D, prÈsentation de SysML en STI2D, 2004. Disponible [ici](#).

Chapter 27

Glossaire

Acronymes SysML

act

Raccourcis pour Diagramme d'**ACT**ivité dans une cartouche SysML™

bdd

Raccourcis pour **B**lock **D**efinition **D**igram dans une cartouche SysML™

dss

Diagramme de **S**équence **S**ystÈme (un **sd** où seul le systÈme dans sa globalité est représenté¹)

ibd

Raccourcis pour **I**nternal **B**lock **D**igram dans une cartouche SysML™

par

Raccourcis pour **P**arametric **D**igram dans une cartouche SysML™

pkg

Raccourcis pour **P**a**K**a**G**e **D**igram dans une cartouche SysML™

req

Raccourcis pour **RE**quirements **D**igram dans une cartouche SysML™

sd

Raccourcis pour **SE**quence **D**igram dans une cartouche SysML™

stm

Raccourcis pour **ST**ate **M**achine dans une cartouche SysML™

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

uc

Raccourcis pour Use Case Diagram dans une cartouche SysML™

Définitions générales

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.

FAQ

Frequently Asked Questions : une liste de questions souvent posées (et les réponses correspondantes) sur un thème donné.

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 liées à l'objet (CORBA, UML, etc.).

TDD

Test Driven Development : Développement 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](#)).

STI2D

Sciences et Technologies de l'Industrie et du Développement Durable : série du baccalauréat qui met l'accent sur les technologies transversales et qui a introduit en 2011 l'enseignement de SysML™.

SysML

System Modeling Language TM : Le langage de modélisation de systèmes maintenu par l'OMGTM.

UML

Unified Modeling Language TM : Le langage de modélisation généraliste maintenu par l'OMGTM.

Dernière MAJ : 12/10/2013 - 09:52:25 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, améliorée 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 ‡ l'Identique 3.0 non transposée](#).