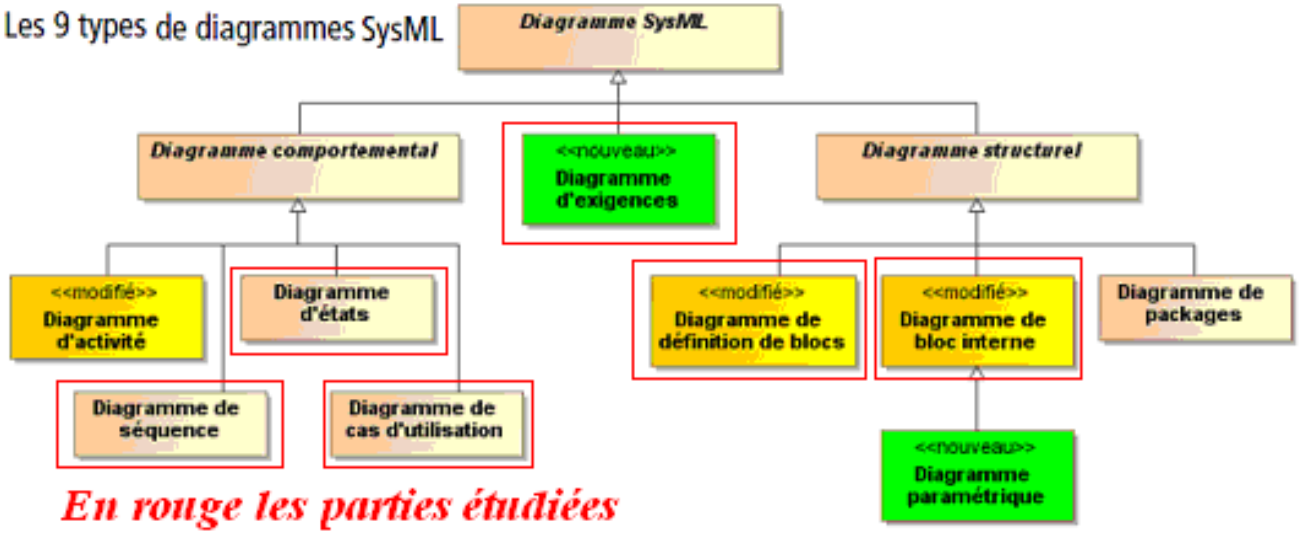
Ce cours est largement inspiré du livre « sysml par l'exemple » de Pascal Roques et est étayé par la présentation sysml d'un radio réveil.

1.) GENERALITES

1.1.) HISTORIQUE

OMG = Object Management Group.
L'OMG est un groupement d'industriels dont l'objectif est de standardiser autour des technologies objet, afin de garantir l'interopérabilité des développements. L'OMG comprend actuellement plus de 800 membres, dont les principaux acteurs de l'industrie informatique (Sun, IBM, etc.), mais aussi les plus grandes entreprises utilisatrices dans tous les secteurs d'activité.(www.omg.org).
L'OMG a annoncé l'adoption de SysML en juillet 2006 et la disponibilité de la première version officielle(SysML v. 1.0) en septembre 2007. Très récemment, une nouvelle spécification SysML v. 1.1 a été rendue publique (décembre 2008). Le groupe de travail pour la révision 1.2 est déjà en action…

1.2.) PRESENTATION



1.2.1.) Modélisation d'exigences

le diagramme d'exigences (montre les exigences du système et leurs relations).
Le diagramme d'exigences capture les hiérarchies d'exigences, ainsi que leurs relation de dérivation, de satisfaction, de vérification et de raffinement. Ces relations fournissent la capacité de relier les exigences les unes aux autres, ainsi qu'aux éléments de conception et aux cas de tests.

1.2.2.) Modélisation comportementale

1.2.2.1.) Diagramme d'activité :

montre l'enchainement des actions et décisions au sein d'une activité complexe.
représente les flots de données et de contrôle entre les actions.

1.2.2.2.) Diagramme de séquence :

montre la séquence verticale des messages passés entre blocs au sein d'une interaction.
représente les interactions entre les parties du système qui collaborent.

1.2.2.3.) Diagramme d'états :

montre les différents états et transitions possibles des blocs dynamiques) ;
décrit les transitions entre états et les actions que le système ou ses parties réalisent en réponse aux événements.

1.2.2.4.) Diagramme de cas d'utilisation :

montre les interactions fonctionnelles entre les acteurs et le système à l'étude.
fournit une description de haut niveau des fonctionnalités du système.

1.2.3.) Modélisation structurelle

1.2.3.1.) Diagramme de définition de blocs :

montre les briques de base statiques : blocs, compositions, associations, attributs, opérations, généralisations, etc.).
Le diagramme de définition de blocs (block definition diagram) décrit la hiérarchie du système et les classifications système/composant.

1.2.3.2.) Diagramme de bloc interne :

montre l'organisation interne d'un élément statique complexe.
Le diagramme de bloc interne (internal block diagram) décrit la structure interne du système en termes de parties, ports et connecteurs.

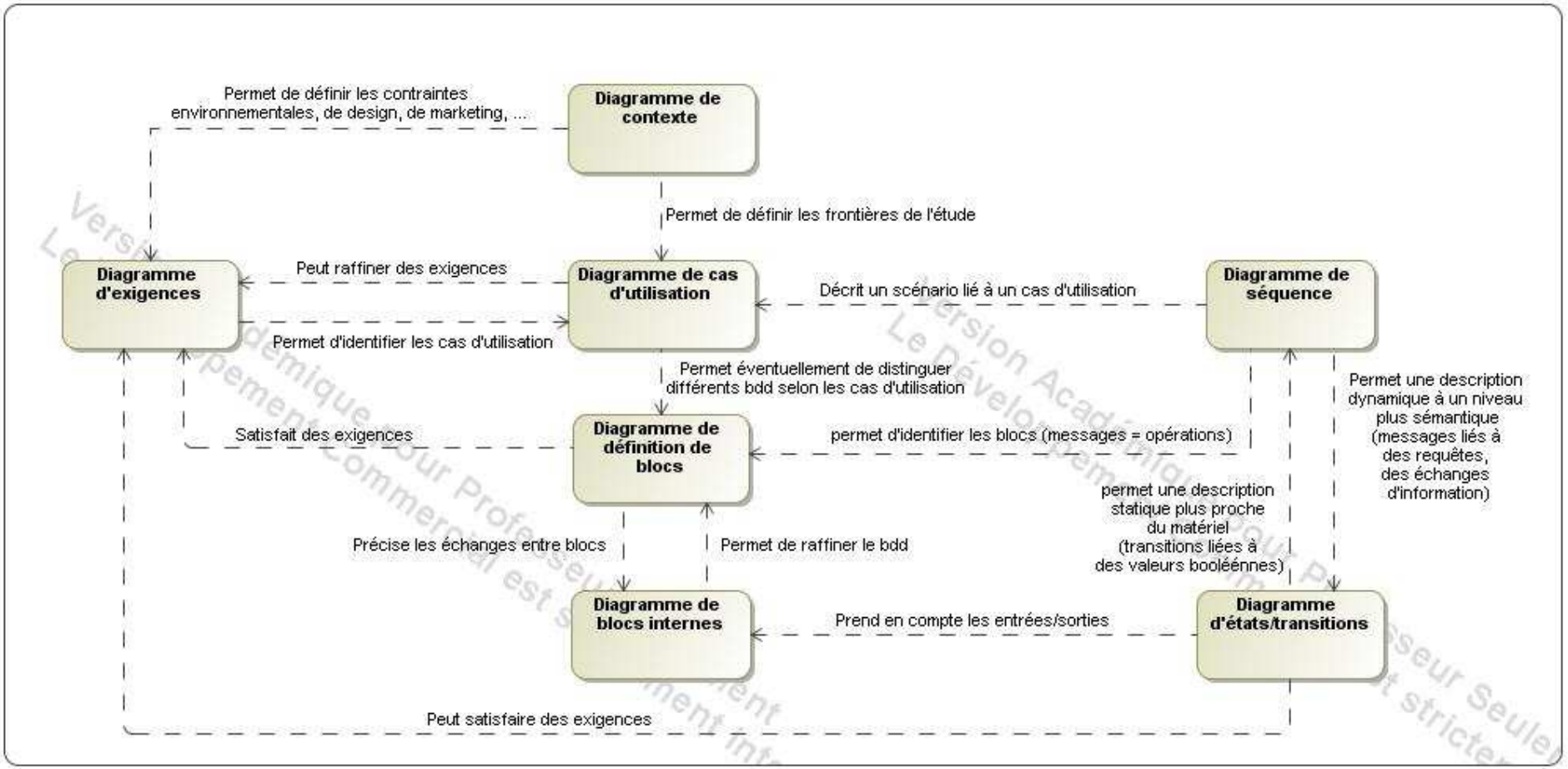
1.2.3.3.) Diagramme paramétrique :

représente les contraintes du système, les équations qui le régissent.

1.2.3.4.) Diagramme de packages :

montre l'organisation logique du modèle et les relations entre packages.
Le diagramme de packages est utilisé pour organiser le modèle.

1.3.) RECAPITULATIF



2.) LA MODELISATION DES EXIGENCES (DIAGRAMMES D'EXIGENCE)

2.1.) DEFINITION D'UNE EXIGENCE

Une exigence permet de spécifier une capacité ou une contrainte qui doit être satisfaite par un système.
Elle peut spécifier une fonction que le système devra réaliser ou une condition de performance, de fiabilité, de sécurité, etc.
Les exigences servent à établir un contrat entre le client et les réalisateurs du futur système.
Les deux propriétés de base d'une exigence sont :

* Un identifiant unique (permettant ensuite de gérer la traçabilité avec l'architecture, etc…...).
* Un texte descriptif.

2.2.) CARTOUCHE DE DIAGRAMME

Un cartouche positionné en haut à gauche du diagramme dans un pentagone permet de spécifier le type de diagramme SysML, le type de l'élément concerné, l'élément concerné et le nom du diagramme.
Autres propriétés
Il est courant de définir d'autres propriétés pour les exigences, par exemple :

priorité (par exemple : haute, moyenne, basse) ;
source (par exemple : client, marketing, technique, législation, etc.) ;
risque (par exemple : haut, moyen, bas) ;
statut (par exemple : proposée, validée, implémentée, testée, livrée, etc.) ;
méthode de vérification (par exemple : analyse, démonstration, test, etc.).

2.3.) RELATIONS DES DIAGRAMMES D'EXIGENCE

Les exigences peuvent être reliées entre elles par des relations de contenance, de raffinement ou de dérivation :

2.3.1.) La contenance ou confinement :

Elle permet de décomposer une exigence composite en plusieurs exigences unitaires, plus faciles ensuite à tracer vis-à-vis de l'architecture ou des tests
En d'autres termes :
Y contient X

2.3.2.) Le raffinement:

Consiste en l'ajout de précisions, par exemple de données quantitatives.
En d'autres termes :
un ou plusieurs éléments du modèle redéfinissent une exigence.

2.3.3.) La dérivation:

Consiste à relier des exigences de niveaux différents, par exemple, des exigences système à des exigences de niveau sous-système, etc…. Elle implique généralement des choix d'architecture.
En d'autres termes :
Permet de relier une exigence d'un niveau général à une exigence d'un niveau plus spécialisée mais exprimant la même contrainte.

2.4.) COMPLEMENTS

2.4.1.) Notes graphiques

SysML permet d'utiliser des notes graphiques sur tous les types de diagrammes (forme de post-it).
Deux mots-clés particuliers ont été définis afin de représenter :

* des problèmes à résoudre (« **problem** ») ;
* des justificatifs (« **rationale** »).

2.4.2.) Les types d'exigences

Il est également possible de faire la distinction entre différents types d'exigences, au lieu d'utiliser un unique type banalisé :

* fonctionnelle
* performance ;
* fiabilité ; •
* sécurité ; •
* volumétrie ; •
* physique ; •
* interface ; • etc.

Les exigences non fonctionnelles (performance, fiabilité, etc.) apportent souvent de la précision aux exigences fonctionnelles, comme illustré sur la figure suivante, où nous avons représenté une exigence d'interface et une exigence physique raffinant une exigence fonctionnelle de base.

2.4.3.) Traçabilité

La gestion des exigences est l'ensemble des activités permettant de définir et de suivre les exigences d'un système au cours d'un projet.
Elle permet de:

s'assurer de la cohérence entre ce que fait réellement le projet et ce qu'il doit faire ;
de faciliter l'analyse d'impact en cas de changement.

Le diagramme d'exigences permet ainsi tout au long d'un projet de relier les exigences avec d'autres types d'élément SysML par plusieurs relations .

2.4.3.1.) Refine

exigence < == > élément comportemental (cas d'utilisation, diagramme d'états, etc.) :

2.4.3.2.) Satisfy

exigence < == > bloc d'architecture : un ou plusieurs éléments du modèle permettent de

satisfaire une exigence

2.4.3.3.) Verify :

exigence < == > cas de test un ou plusieurs éléments du modèle permettent de vérifier et valider une exigence.

2.4.4.) Cas de test

Un cas de test représente une méthode de vérification de la satisfaction d'une exigence.
Il est représenté en SysML par un rectangle avec le mot-clé « Test Case ».

3.) MODELISATION COMPORTEMENTALE (LES DIAGRAMMES DE CAS D'UTILISATION)

3.1.) GENERALITES

3.1.1.) Acteur

Rôle joué par un utilisateur humain ou un autre système qui interagit directement avec le système étudié. Un acteur participe à au moins un cas d'utilisation.
La recommandation commune consiste à faire prévaloir l'utilisation de la forme graphique du stick man pour les acteurs humains et une représentation rectangulaire pour les systèmes connectés.

3.1.2.) Cas d'utilisation

Un cas d'utilisation (use case, ou UC) représente un ensemble de séquences d'actions qui sont réalisées par le système et qui produisent un résultat observable intéressant pour un acteur particulier. Chaque cas d'utilisation spécifie un comportement attendu du système considéré comme un tout, sans imposer le mode de réalisation de ce comportement. Il permet de décrire ce que le futur système devra faire, sans spécifier comment il le fera.
Un cas d'utilisation doit être relié à au moins un acteur
Le diagramme de cas d'utilisation est un schéma qui montre les cas d'utilisation (ovales) reliés par des associations (lignes) à leurs acteurs (icône du stick man, ou représentation graphique équivalente). Chaque association signifie simplement « participe à ».

3.1.3.) Scénario

Un scénario représente une succession particulière d'enchaînements, s'exécutant du début à la fin du cas d'utilisation, un enchaînement étant l'unité de description de séquences d'actions.
Un cas d'utilisation contient en général un scénario nominal et plusieurs scénarios alternatifs (qui se terminent de façon normale) ou d'erreur (qui se terminent en échec).
On peut d'ailleurs proposer une définition différente pour un cas d'utilisation : « ensemble de scénarios d'utilisation d'un système reliés par un but commun du point de vue de l'acteur principal ».

3.2.) RELATIONS DE CAS D'UTILISATION

Pour affiner le diagramme de cas d'utilisation, SysML définit trois types de relations standardisées entre cas d'utilisation :

3.2.1.) Relation d'inclusion

Le cas d'utilisation de base en incorpore explicitement un autre, de façon obligatoire.
En d'autres termes :
le cas d'utilisation source comprend obligatoirement le cas inclus

3.2.2.) Relation d'extension

Le cas d'utilisation de base en incorpore implicitement un autre, de façon optionnelle, à un endroit spécifié indirectement dans celui qui procède à l'extension (appelé extension point).
En d'autres termes :
le cas d'utilisation source est une extension possible du cas d'utilisation destination

3.2.3.) Relation de généralisation/spécialisation :

les cas d'utilisation descendants héritent la description de leur parent commun. Chacun d'entre eux peut néanmoins comprendre des interactions spécifiques supplémentaires.
En d'autres termes :
X est une sorte de Y

3.2.4.) Relation d'association:

Associer = « participer à.. »
En d'autres termes :
L'utilisateur X participe au cas Y

3.3.) COMPLEMENTS

3.3.1.) Stéréotype

Les mots-clés SysML comme « include » et « extend » sont notés entre guillemets typographiques.

3.3.2.) Acteur généralisé

Deux acteurs ou plus peuvent présenter des similitudes dans leurs relations aux cas d'utilisation. On peut exprimer ce concept en créant un acteur généralisé qui modélise les aspects communs aux différents acteurs concrets.

4.) MODELISATION COMPORTEMENTALE (DIAGRAMME DE SEQUENCE)

Le diagramme de séquence montre la séquence verticale des messages passés entre éléments (lignes de vie) au sein d'une interaction.

4.1.) LIGNE DE VIE

Représentation de l'existence d'un élément participant dans un diagramme de séquence.

Une ligne de vie possède un nom et un type. Elle est représentée graphiquement par une ligne verticale en pointillés.

4.2.) MESSAGE

Élément de communication unidirectionnel entre lignes de vie qui déclenche une activité dans le destinataire.
La réception d'un message provoque un événement chez le récepteur.
Un message asynchrône est représenté par une flèche évidée.

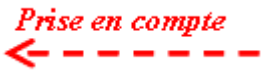
Un message asynchrone n'attend pas de réponse.

Un message synchrone (émetteur bloqué en attente de réponse) est représenté par une flèche pleine. La flèche pointillée représente un retour. Cela signifie que le message en question est le résultat direct du message précédent.

La flèche qui boucle (message réflexif) permet de représenter un comportement interne.

Message asynchrône: Pas d'attente de réponse

Message synchrône: ***A***ttente de réponse



Message de réponse: Réponse au message

Message réflexif: Attente de fin d'éxécution

4.3.) DIAGRAMME DE SEQUENCE « SYSTEME » (DSS)

Nous utilisons le terme de diagramme de séquence « système » pour souligner le fait que dans ce type de diagramme de séquence, nous considérons le système entier comme une boîte noire et le représentons par une seule ligne de vie. Le comportement du système est décrit vu de l'extérieur, sans préjuger de comment il le réalisera.

4.4.) COMPLEMENTS :

4.4.1.) Fragments combinés

SysML propose une notation très utile : le fragment combiné.
Chaque fragment possède un opérateur et peut être divisé en opérandes.
Les principaux opérateurs **sont** :

4.4.1.1.) Loop (boucle)

Le fragment peut s'exécuter plusieurs fois, et la condition de garde explicite l'itération ;

4.4.1.2.) Opt (optionnel)

Le fragment ne s'exécute que si la condition fournie est vraie ;

4.4.1.3.) Alt (alternatif).

Seul le fragment possédant la condition vraie s'exécutera.

4.4.2.) Contraintes temporelles

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

4.4.2.1.) Contrainte de durée

Elle permet d'indiquer une contrainte sur la durée exacte, la durée minimale ou la durée maximale entre deux événements .

4.4.2.2.) Contrainte de temps :

Elle permet de positionner des labels associés à des instants dans le scénario au niveau de certains messages et de les relier ainsi entre eux.

5.) MODELISATION SRUCTURELLE(DIAGRAMME DE BLOC)

5.1.) PROPRIETES DES BLOCS

Les propriétés sont les caractéristiques structurelles de base des blocs. Elles peuvent être de deux types principaux :

5.1.1.) Les valeurs (value properties) :

Elles décrivent des caractéristiques quantifiables en terme de value types (domaine de valeur, dimension et unité optionnelles).

5.1.2.) Les parties (part properties) :

Elles décrivent la hiérarchie de décomposition du bloc en termes d'autres blocs.

5.2.) LES RELATIONS

5.2.1.) Composition

La relation entre blocs, dans laquelle un bloc représente le tout et les autres ses parties.

X entre dans la composition de Y et est indispensable.

5.2.2.) Agrégation

l'agrégation est utile : pour représenter le fait que la contenance n'est pas vraiment structurelle et obligatoire, mais plus conjoncturelle

X entre dans la composition de Y sans être indispensable :

5.2.3.) Association

L'association est une relation n'impliquant pas de contenance, comme la composition ou l'agrégation, mais une relation d'égal à égal.

X utilise Y

6.) MODELISATION STRUCTURELLE (DIAGRAMME DE BLOC INTERNE)

On peut représenter la connexion entre les éléments (*parts* en anglais, que je traduis par ***Parties***) d'un bloc au moyen d'un diagramme de bloc interne.
Ce diagramme montre principalement les relations entre éléments de même niveau, ainsi que les éventuelles multiplicités des parties.
Le cadre de l'ibd représente le bloc englobant. Il fournit le contexte pour tous les éléments du diagramme.
Chaque extrémité d'une relation de composition, ou élément du compartiment parts, apparaît comme un bloc à l'intérieur du cadre ibd.
Le nom du bloc est identique à ce qui apparaitrait dans le compartiment parts, soit :
nom_partie : nom_bloc

6.1.) CONNECTEUR

Le connecteur est un concept structurel utilisé pour relier deux parties et leur fournir l'opportunité d'interagir, bien que le connecteur ne dise rien sur la nature de cette interaction.
Les connecteurs permettent également de relier plus finement les parties à travers des ports.
L'extrémité d'un connecteur peut posséder une multiplicité qui décrit le nombre d'instances qui peuvent être connectées par des liens décrits par le connecteur.
Soyons précis :
* une association (agrégation, ou composition) relie des blocs (Bdd)
    * un connecteur relie des parties (Ibd).
    * un lien relie des instances.
    * en programmation orientée objet, une instance d'une classe est un objet avec un comportement correspondant à cette classe et un état initial.

6.2.) PORTS ET INTERFACES

Le diagramme de bloc interne permet également de décrire la logique de connexion, de services et de flots entre blocs grâce au concept de « port » :

On retrouve 2 types de ports

6.2.1.) Flux (flow port) :

ce type de port autorise la circulation de flux physiques entre les blocs.
La nature de ce qui peut circuler va des fluides aux données, en passant par l'énergie ;

6.2.2.) Standard :

ce type de port autorise la description de services logiques entre les blocs, au moyen d'interfaces regroupant des opérations.

Les ports de type « flux » sont soit atomiques (un seul flux), soit composites (agrégation de flux de natures différentes).

7.) MODELISATION COMPORTEMENTALE (DIAGRAMMES D'ETAT)

7.1.) GENERALITES

Le diagramme d'états décrit graphiquement comment un bloc réagit à des événements en fonction de son état courant et comment il passe dans un nouvel état.

7.2.) ETAT

Il représente une situation durant la vie d'un bloc pendant laquelle * il satisfait une certaine condition et il exécute une certaine activité ou bien il attend un certain événement.
Un bloc passe par une succession d'états durant son existence.
Un état a une durée finie, variable selon la vie du bloc, en particulier en fonction des événements qui lui arrivent.

7.3.) ÉVENEMENT

Spécification d'une occurrence qui peut déclencher une réaction sur un élément et qui possède une localisation dans le temps et l'espace.
Un événement peut porter des paramètres qui matérialisent le flot d'information ou de données reçu par l'élément.

7.4.) TRANSITION

Une transition décrit la réaction d'un bloc lorsqu'un événement se produit (généralement le bloc change d'état, mais pas forcément). En règle générale, une transition possède un événement déclencheur, une condition de garde, un effet et un état cible.

7.5.) CONDITION

Une condition (ou condition de garde) est une expression booléenne qui doit être vraie lorsque l'évènement arrive pour que la transition soit déclenchée. Elle se note entre crochets.
Elle peut concerner les valeurs du bloc concerné ainsi que les paramètres de l'événement déclencheur.
Plusieurs transitions avec le même événement doivent avoir des conditions de garde différentes.

7.6.) ETAT COMPOSITE (AUSSI APPELE SUPER-ETAT)

Il permet d'englober plusieurs sous-états exclusifs. On peut ainsi factoriser des transitions déclenchées par le même événement et amenant vers le même état cible (comme power_ON ou power_OFF dans l'exemple), tout en spécifiant des transitions particulières entre les sous-états.
Il permet de placer plusieurs états initiaux dans un même diagramme d'états.

8.) LES RELATIONS DE SYSML

8.1.) DIAGRAMME D'EXIGENCES

Conteneur : Y contient X

Refine un ou plusieurs éléments du modèle *redéfinissent une exigence*.

DeriveReqt : permet de *relier une exigence d'un niveau général à une exigence d'un niveau plus spécialisée* mais exprimant la même contrainte.

8.2.) DIAGRAMME DE CAS D'UTILISATION

Extend : le cas d'utilisation source est *une extension possible* du cas d'utilisation destination.

Include : le cas d'utilisation source *comprend obligatoirement* le cas inclus.

Généralisation : le cas d'utilisation source *comprend obligatoirement* le cas inclus.

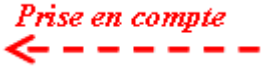
Association : Participe au cas d'utilisation.

8.3.) DIAGRAMME DE SEQUENCE (DSS)

Message asynchrône: Pas d'attente de réponse

Message synchrône: **A**ttente de réponse

Message de réponse: Réponse au message



Message réflexif: Représentation de comportement interne

8.4.) DIAGRAMME DE BLOC (BDD)

Composition :

X entre dans la composition de Y et est indispensable

Agrégation :

X entre dans la composition de Y sans être ndispensable

Généralisation : X est une sorte de Y

8.5.) DIAGRAMME DE PACKAGE

Contenance : Y contient X

Dépendance : X dépend de Y

8.6.) RELATIONS ENTRE EXIGENCES ET AUTRES DIAGRAMMES

exigence <==> élément comportemental (cas d'utilisation, diagramme d'états, etc.) :
Refine

exigence <==> bloc d'architecture (traçabilité):
Satisfy un ou plusieurs éléments du modèle permettent de *satisfaire une exigence*.

exigence <==> cas de test :
Verify un ou plusieurs éléments du modèle permettent de *vérifier et valider une exigence*.

8.7.) TOUS TYPES

Notes de types **Problem (**des problèmes à résoudre) **et rationale (**des justificatifs)**.**