

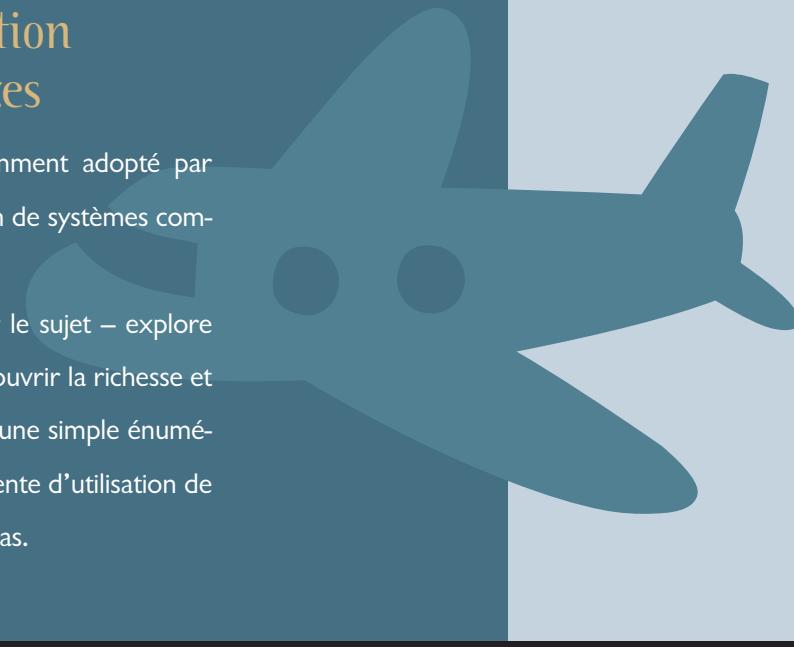
Pascal Roques

# SysML par l'exemple

## Un langage de modélisation pour systèmes complexes

Le nouveau langage de modélisation SysML, récemment adopté par l'OMG, est dédié à la problématique de la conception de systèmes complexes (satellites, avions, etc.).

Cet ouvrage introductif – le premier en français sur le sujet – explore l'ensemble des diagrammes SysML pour en faire découvrir la richesse et les atouts. Très illustré, le livre propose bien plus qu'une simple énumération de concepts : il transmet une démarche cohérente d'utilisation de SysML en prenant pour fil conducteur une étude de cas.



**EYROLLES**

Prix : 25 €

Code éditeur : G85006

ISBN : 978-2-212-85006-2

© Eyrolles 2009

# Avant-propos

## Objectifs du livre

L'ingénierie système est une démarche méthodologique générale qui englobe l'ensemble des activités adéquates pour concevoir, faire évoluer et vérifier un système apportant une solution économique et performante aux besoins d'un client tout en satisfaisant l'ensemble des parties prenantes. Depuis longtemps, les ingénieurs système ont utilisé des techniques de modélisation. Parmi les plus connues, on trouve SADT et SA/RT, qui datent des années 80, ainsi que de nombreuses approches basées sur les réseaux de Pétri ou les machines à états finis. Mais ces techniques sont limitées par leur portée et leur expressivité ainsi que par la difficulté de leur intégration avec d'autres formalismes, ainsi qu'avec les exigences système.

L'essor d'UML dans le domaine du logiciel et l'effort industriel de développement d'outils qui l'accompagne ont naturellement conduit à envisager son utilisation en ingénierie système. Cependant, du fait de sa conception fortement guidée par les besoins du passage à la programmation par objets, le langage était, tout au moins dans ses premières versions, peu adapté à la modélisation des systèmes complexes et donc au support de l'ingénierie système (IS).

La version 2 d'UML, officialisée en 2005, a introduit plusieurs nouveaux concepts et diagrammes utiles pour l'IS. En particulier, le diagramme de structure composite avec les concepts de classe structurée, partie, port et connecteur, permet maintenant de décrire l'interconnexion statique des parties d'un système complexe. Les avancées du diagramme de séquence permettent également de décrire des scénarios d'interaction de façon descendante en ajoutant progressivement des niveaux d'architecture. Mais il restait toujours la barrière psychologique du vocabulaire orienté informatique : classe, objet, héritage, etc.

La communauté de l'IS a voulu définir un langage commun de modélisation pour les ingénieurs système, adapté à leur problématique, comme UML l'est devenu pour les informaticiens. Ce nouveau langage, nommé SysML, est fortement inspiré de la version 2 d'UML, mais ajoute la possibilité de représenter les exigences du système, les éléments non-logiciels (mécanique, hydraulique, capteur...), les équations physiques, les flux continus (matière, énergie, etc.) et les allocations.

La version 1.0 du nouveau langage de modélisation SysML a été adoptée officiellement par l'OMG le 19 septembre 2007 ! Il n'est donc pas étonnant que la littérature sur le sujet soit balbutiante et qu'il n'existe pas encore de livre en français sur le sujet...

Mon ambition dans cet ouvrage introductif est donc de vous faire découvrir ce nouveau langage de modélisation pour l'ingénierie système. Fort de mon passé de consultant en modélisation dans les domaines aéronautique et spatial, de ma pratique pédagogique en tant que formateur UML et SysML, ainsi que de mon expérience d'auteur sur UML (plus de 40 000 exemplaires vendus), j'espère parvenir à vous faire apprécier la richesse et les atouts de SysML.

Ce livre s'adresse avant tout aux professionnels de l'ingénierie système, c'est à dire à tous ceux qui ont en charge des systèmes complexes, incluant du logiciel et du matériel, que ce soit dans l'aéronautique, l'astronautique, l'automobile, l'énergie, le transport, l'armement,

## Structure de l'ouvrage

Après une brève introduction sur la problématique de l'ingénierie système, je détaillerai l'historique du langage SysML et sa filiation avec UML. Je présenterai ensuite les principes du processus de modélisation appliqués dans le livre.

La partie I de l'ouvrage concerne la modélisation des exigences. Nous verrons que SysML innove en proposant un diagramme permettant de dessiner graphiquement les exigences système et surtout de les relier aux éléments structurels ou dynamiques de la modélisation, ainsi qu'à d'autres exigences de niveau sous-système ou équipement. Nous apprendrons également à mettre en œuvre la technique des cas d'utilisation, déjà présente en UML, au niveau système. Nous verrons enfin une première application du diagramme de séquence pour décrire les interactions entre les acteurs et le système « boîte noire ».

La partie II concerne la modélisation structurelle. Nous apprendrons à utiliser le concept universel de « bloc » proposé par SysML pour modéliser tout élément structurel, ainsi que les deux types de diagrammes associés. Nous verrons tout d'abord comment définir les éléments structurels de base de notre modèle dans le diagramme de définition de blocs. Nous apprendrons ensuite à décrire la décomposition des éléments complexes avec le diagramme interne de bloc. Nous verrons enfin comment structurer notre modèle en *packages*, à des fins de travail en équipe ou de réutilisation.

La partie III concerne la modélisation dynamique. Nous verrons toute la puissance du diagramme d'états, pour modéliser le cycle de vie des éléments à dynamique prédominante, ainsi que celle du diagramme d'activité, qui permet de modéliser avec précision des algorithmes complexes.

La partie IV concerne la modélisation transverse. SysML permet de décrire des équations grâce au nouveau diagramme paramétrique. Nous verrons également comment décrire plusieurs types de liens de traçabilité entre éléments de modélisation, et en particulier comment mettre en œuvre le concept fondamental d'allocation.

Une première annexe récapitule les acronymes et les définitions. Une seconde annexe présente d'autres diagrammes de l'étude de cas réalisés avec des outils de modélisation différents : *Enterprise Architect*, *Topcased* et *Artisan Studio*.

## Remerciements

Cet ouvrage n'aurait jamais vu le jour sans les encouragements et les retours motivants des lecteurs de mes livres sur UML, mais aussi des nombreux stagiaires des cours SysML que je donne depuis plus de deux ans déjà ! Ils m'ont donné l'énergie et l'envie de partager une fois encore mes connaissances et mon expérience à travers ce support que j'apprécie tant.

Merci à mes relecteurs techniques pour leurs judicieuses remarques :

- Tony Cornuaud (ingénieur en informatique, évangéliste SysML chez Magneti Marelli à Châtellerault) ;
- Christophe Surdieux (ancien de Valtech, maintenant chez Altran, certifié sur le cours MODSY que j'ai créé à Valtech Training) ;

- Olivier Casse (expert en langages et outils de modélisation de systèmes embarqués, ancien d'I-Logix/Telelogic).

Merci à la société NoMagic de m'avoir fourni une licence de l'outil MagicDraw UML ([www.magicdraw.com](http://www.magicdraw.com)) avec son plugin SysML que je trouve excellent, et grâce auquel j'ai réalisé la majeure partie des diagrammes du livre.

Merci à mon employeur, la société Artal et sa nouvelle filiale de conseil A2 (Artal Innovation), de m'avoir accordé le temps nécessaire à la rédaction de cet ouvrage dans le cadre de mon activité de R&D.

Merci aussi à Éric Sulpice et Muriel Shan Sei Fan des Éditions Eyrolles pour leur témoignage renouvelé de confiance et leur proposition de tester un nouveau format pour ce livre introductif à SysML. Un grand bravo également à l'équipe des éditrices qui a contribué notamment à la réussite de ce projet.

Enfin, comment ne pas terminer par un clin d'œil affectueux à celle qui partage ma vie depuis maintenant plus de six ans, et dont l'énergie m'aide à avancer. Merci à Sylvie, la femme qui m'accompagne...

Pascal Roques, février 2009  
Consultant senior A2 (Artal Innovation)  
[pascal.roques@a2-ortal.fr](mailto:pascal.roques@a2-ortal.fr)  
blog : [consultants.a2-ortal.fr/proques](http://consultants.a2-ortal.fr/proques)

# Table des matières

<b>AVANT-PROPOS .....</b>	<b>II</b>
<b>INTRODUCTION.....</b>	<b>1</b>
<b>Problématique de l'ingénierie système 2</b>	
Les normes d'IS 3	
IEEE 1 220 : Standard for Application and Management of the Systems Engineering Process, 1999 4	
EIA 632 : Processes for Engineering a System, 1998 5	
ISO 15 288 : Systems engineering – System life cycle processes, 2003 6	
Pourquoi SysML ? 8	
La modélisation 8	
SADT, SA/RT, etc. 9	
UML 10	
Lacunes d'UML pour l'IS 12	
SysML 14	
Organisation du livre 19	
L'étude de cas du livre 21	
 Première partie	
<b>La modélisation des exigences .....</b>	<b>22</b>
<b>1. LE DIAGRAMME D'EXIGENCES .....</b>	<b>23</b>
<b>Notation de base 24</b>	
<b>Compléments 29</b>	
<b>Traçabilité 31</b>	

2. LE DIAGRAMME DE CAS D'UTILISATION .....	34
Notation de base 35	
Décrire les cas d'utilisation 40	
Compléments 41	
Classification des acteurs 41	
Relations entre cas d'utilisation 43	
3. LE DIAGRAMME DE SÉQUENCE « SYSTÈME » .....	48
Notation de base 49	
Diagramme de séquence « système » 51	
Compléments 52	
Fragments combinés 52	
Cadre référence 54	
Contraintes temporelles 56	
Deuxième partie	
La modélisation d'architecture.....	58
4. DIAGRAMME DE DÉFINITION DE BLOCS .....	59
Bloc et propriété 60	
ValueType 63	
Partie 65	
Composition 66	
Agrégation 67	
Association 69	
Généralisation 72	
Opération 75	
Étude de cas 76	
5. LE DIAGRAMME DE BLOC INTERNE .....	80
Parties et connecteurs 81	
Parties et références 82	

Connecteur 84	
<b>Ports et interfaces 89</b>	
Types de ports 89	
Flow ports 90	
Item flow 92	
Flow specification 94	
Interface 95	
Port standard 98	
<b>Étude de cas 99</b>	
<b>6. LE DIAGRAMME DE PACKAGES .....</b>	<b>106</b>
<b>    Package 107</b>	
<b>    Contenance et dépendance 109</b>	
Contenance 109	
Dépendance 110	
<b>    Model, view et viewpoint 113</b>	
<b>    Étude de cas 114</b>	
 Troisième partie	
<b>La modélisation dynamique .....</b>	<b>116</b>
<b>7. LE DIAGRAMME D'ÉTATS.....</b>	<b>117</b>
<b>    Notation de base 118</b>	
<b>    Compléments 123</b>	
Événement interne ou temporel 123	
État composite 125	
Autres notations avancées 132	
Animation du diagramme d'états 137	
<b>8. LE DIAGRAMME D'ACTIVITÉ .....</b>	<b>139</b>
<b>    Notation de base 140</b>	
<b>    Object node, object flow 148</b>	

**Compléments 151**

- Appel d'activité 151
- Signaux et événements 153
- Région interruptible 154
- Région d'expansion 156
- Compléments sur les flots d'objet 157
- Opérateur de contrôle 159

**Étude de cas 160****Quatrième partie**

<b>La modélisation transverse .....</b>	<b>164</b>
<b>9. LE DIAGRAMME PARAMÉTRIQUE.....</b>	<b>165</b>
<b>Notation de base 166</b>	
<b>Compléments 170</b>	
<b>10. ALLOCATION ET TRAÇABILITÉ.....</b>	<b>172</b>
<b>Le concept d'allocation 173</b>	
<b>Les différentes représentations 174</b>	
Diagramme de définition de blocs (bdd) 174	
Diagramme d'activité 176	
Représentation tabulaire 178	
Diagramme de bloc interne (ibd) 180	
Diagramme de séquence 182	
<b>A. ACRONYMES ET DÉFINITIONS .....</b>	<b>184</b>
<b>Acronymes 185</b>	
<b>Définitions 186</b>	
<b>B. DIAGRAMMES EA, TOPCASED ET ARTISAN STUDIO .....</b>	<b>193</b>
<b>TopCased 194</b>	
<b>Enterprise Architect (EA) 202</b>	
<b>Artisan Studio (encore merci à Olivier Casse !) 209</b>	

# Introduction

Dans ce chapitre introductif, nous dévoilons les objectifs et l'historique du nouveau langage de modélisation SysML. Après un tour rapide de la problématique de l'ingénierie système et des différentes normes existantes sur le sujet, nous expliquons pourquoi l'OMG a décidé de définir un nouveau langage de modélisation. Nous présentons également rapidement les différents diagrammes SysML qui seront détaillés dans la suite du livre.

- ▶ Ingénierie système (IS)
- ▶ modélisation
- ▶ UML
- ▶ SysML

# Problématique de l'ingénierie système

L'ingénierie système (IS) est une démarche méthodologique pour maîtriser la conception des systèmes et produits complexes. On peut aussi la définir comme « une approche interdisciplinaire rassemblant tous les efforts techniques pour faire évoluer et vérifier un ensemble intégré de systèmes, de gens, de produits et de solutions de processus de manière équilibrée au fil du cycle de vie pour satisfaire aux besoins client ». Les pratiques de cette démarche sont aujourd'hui répertoriées dans des normes, réalisées à l'aide de méthodes et supportées par des outils.

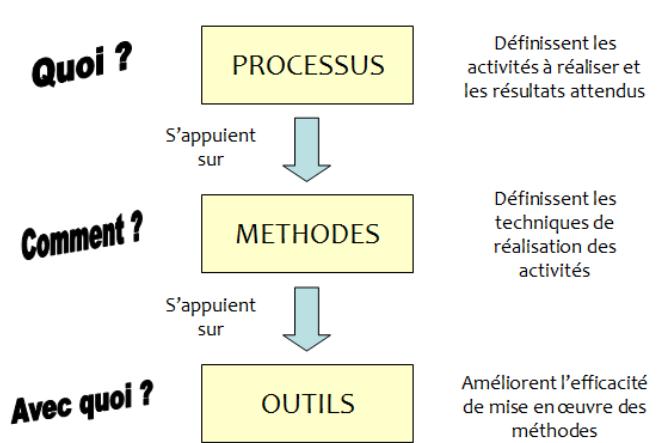
## B.A.-BA Système

Un système est un ensemble de composants interreliés qui interagissent les uns avec les autres d'une manière organisée pour accomplir une finalité commune (NASA 1995).

Un système est un ensemble intégré d'éléments qui accomplissent un objectif défini (INCOSE 2004).

**Figure 1**

Les processus, méthodes et outils



Les normes d'IS décrivent les pratiques du métier en termes de processus et d'activités de manière invariante par rapport aux domaines d'application de l'ingénierie système. Les méthodes d'ingénierie système fournissent des démarches techniques pour réaliser ces activités générales. Contrairement aux normes, elles dépendent des secteurs d'application et résultent de choix industriels. La mise en œuvre des processus et des méthodes est assistée par des outils, aujourd'hui très généralement informatisés.

## Les normes d'IS

### RÉFÉRENCE AFIS

L'Association française d'ingénierie système (AFIS) vise à promouvoir l'ingénierie système par la présentation et l'explication de ses principes et de son approche multidisciplinaire en vue de la réalisation de systèmes complexes réussis. Nous nous sommes beaucoup inspirés des textes et images présents sur son site pour ce sous-chapitre.

► [www.afis.fr](http://www.afis.fr)

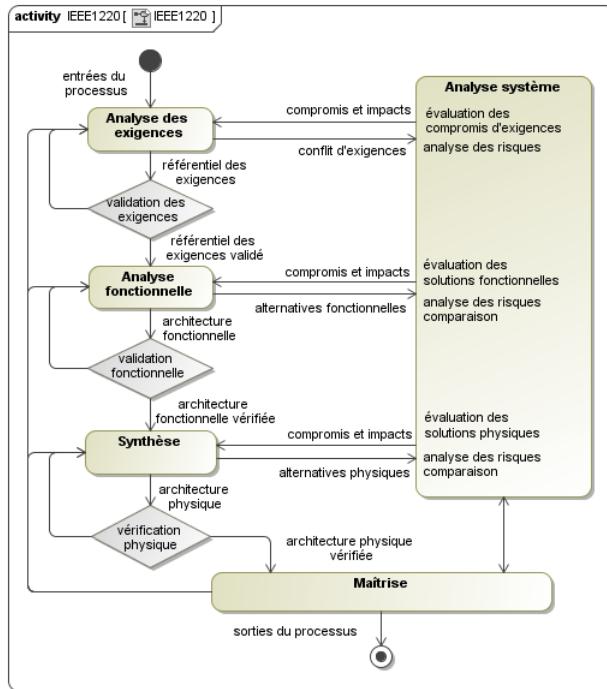
Trois normes générales d'ingénierie système décrivant les processus du métier d'IS sont actuellement disponibles. Elles en définissent les types d'activité à réaliser et les types de résultat produit. Elles recouvrent des champs différents, de manière d'autant plus approfondie que leur champ est limité, et ainsi elles se complètent.

- IEEE 1 220 : *Standard for Application and Management of the Systems Engineering Process*
- EIA 632 : *Processes for Engineering a System*
- ISO 15 288 : *Systems engineering – System life cycle processes*

## IEEE 1 220 : Standard for Application and Management of the Systems Engineering Process, 1999

Issue du standard militaire MIL STD 499B, cette norme de l'IEEE, dont la version initiale date de 1994, se focalise sur les processus techniques d'ingénierie système allant de l'analyse des exigences jusqu'à la définition physique du système.

**Figure 2**  
Les processus  
selon IEEE 1220



Les trois processus d'analyse des exigences, d'analyse fonctionnelle et allocation et de synthèse, finement détaillés, comprennent chacun leur sous-processus de vérification ou de validation.

Le processus d'analyse système a pour but d'analyser dans un cadre pluridisciplinaire les problèmes (conflits d'exigences ou solutions alternatives) issus des processus principaux afin de préparer les décisions.

Le processus de maîtrise de l'IS (control) concerne tout particulièrement la gestion technique de l'ingénierie système et la maîtrise de l'information tant du système que du projet.

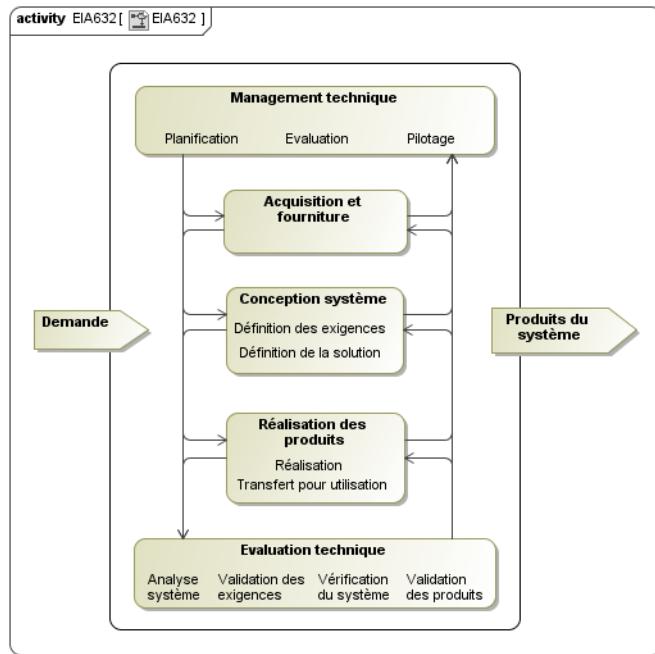
### **EIA 632 : Processes for Engineering a System, 1998**

Cette norme de l'EIA complète les processus techniques de définition du système en couvrant la réalisation des produits jusqu'à leur mise en service (transfert vers l'utilisation). De plus, elle incorpore les processus contractuels d'acquisition et de fourniture.

Processus techniques et processus contractuels sont encadrés :

- par les processus de management (selon leur forme traditionnelle avec les trois sous-processus de planification, évaluation, pilotage) ;
- et par les processus d'évaluation des résultats des activités (processus de vérification s'assurant que l'activité a été bien faite, processus de validation s'assurant que le résultat répond au besoin, les deux justifiant de la conformité, et processus d'analyse système justifiant des choix réalisés tout au long de la définition et donc de l'optimisation du système).

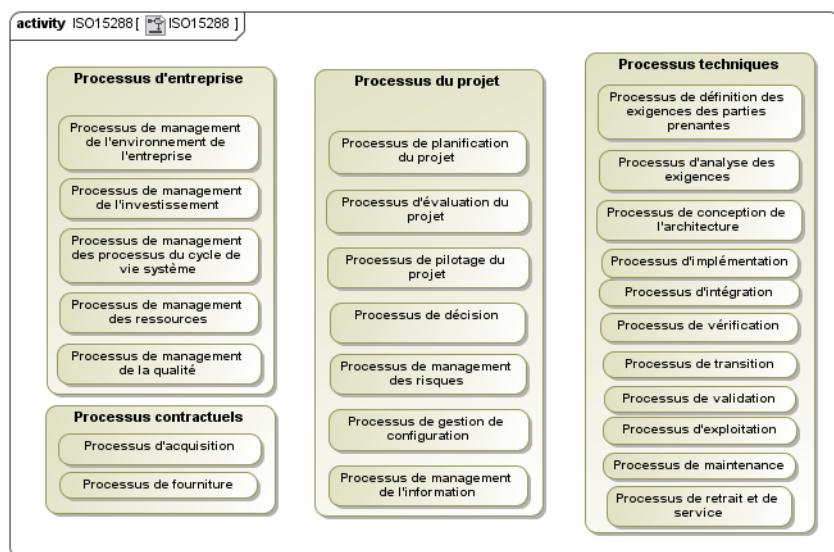
**Figure 3**  
Les processus selon EIA 632



### ISO 15 288 : Systems engineering – System life cycle processes, 2003

Inspirée sur le plan de la forme par la norme ISO/CEI 12 207 – AFNOR Z67 - 150 (Typologie des processus du cycle de vie du logiciel), cette norme de l'ISO étend les processus techniques à tout le cycle de vie du système (elle couvre ainsi les processus d'exploitation, de maintien en condition opérationnelle et de retrait de service).

**Figure 4**  
Les processus  
selon ISO 15 288



# Pourquoi SysML ?

## La modélisation

L'ingénierie système est une démarche méthodologique générale qui englobe l'ensemble des activités adéquates pour concevoir, faire évoluer et vérifier un système apportant une solution économique et performante aux besoins d'un client tout en satisfaisant l'ensemble des parties prenantes.

La transformation d'un besoin émergeant en la définition d'un système lui apportant une solution met en œuvre de multiples activités intellectuelles faisant passer progressivement de concepts abstraits à la définition rigoureuse de produits. Il est nécessaire de s'appuyer sur des représentations tant du problème que de ses solutions possibles à différents niveaux d'abstraction pour appréhender, conceptualiser, concevoir, estimer, simuler, valider, justifier des choix, communiquer. C'est le rôle de la modélisation !

Les métiers mis en œuvre en IS ont, de tous temps, utilisé des modèles allant de représentations des plus concrètes, telles que les plans ou modèles réduits, aux plus abstraites, telles que les systèmes d'équations.

### ATTENTION Qu'est-ce qu'un bon modèle ?

A est un bon modèle de B si A permet de répondre de façon satisfaisante à des questions prédéfinies sur B (d'après D.T. Ross). Un bon modèle doit donc être construit :

- au bon niveau de détail ;
- selon le bon point de vue.

Pensez à l'analogie de la carte routière. Pour circuler dans Toulouse, la carte de France serait de peu d'utilité. Par contre, pour aller de Toulouse à Paris, la carte de la Haute-Garonne ne suffit pas... À chaque voyage correspond la « bonne » carte !

## SADT, SA/RT, etc.

Depuis longtemps, les ingénieurs système ont utilisé des techniques de modélisation. Parmi les plus connues, on trouve SADT et SA/RT, qui datent des années 80, ainsi que de nombreuses approches basées sur les réseaux de Pétri ou les machines à états finis.

### B.A.-BA SADT

SADT = Structured Analysis and Design Technic.

SADT utilise un enchaînement graphique de boîtes élémentaires pouvant être raffinées de façon descendante en d'autres modèles SADT (boîtes + flots). Les modèles utilisés peuvent être des actigrammes (les boîtes sont les fonctions et les flots sont les données) ou des datagrammes (les boîtes sont les données et les flots sont les transformations).

### B.A.-BA SA/RT

SA/RT = Structured Analysis with Real Time extensions.

L'analyse structurée (SA) utilise le diagramme de flots de données (DFD) qui est un modèle hiérarchique permettant une approche descendante par raffinement.

L'extension temps-réel de SA (SA/RT), proposée par exemple par Ward et Mellor, est basée sur l'utilisation d'un DFD complété par un diagramme de flot de contrôle (CFD) en pointillés pour ajouter une sémantique d'exécution au diagramme. En effet, un DFD ne permet pas d'exprimer l'évolution temporelle de l'exécution.

Ces techniques sont limitées par leur portée et leur expressivité ainsi que par la difficulté de leur intégration avec d'autres formalismes, comme avec les exigences système.

SADT et SA, initialement utilisées pour l'informatique de gestion, ne couvrent pas en standard les vues dynamiques. C'est pour cela que la société Verilog (où j'ai travaillé comme consultant de 1986 à 1995), avait ajouté la possibilité de décrire des diagrammes

d'états dans les feuilles de l'arbre fonctionnel SADT pour pouvoir faire de la simulation (outil ASA). Mais l'implémentation de ces extensions à SADT ou SA était toujours propre à l'éditeur de l'outil associé, donc non standardisé.

## UML

UML se définit comme un langage de modélisation graphique et textuel destiné à comprendre et décrire des besoins, spécifier et documenter des systèmes, esquisser des architectures logicielles, concevoir des solutions et communiquer des points de vue.

UML unifie à la fois les notations et les concepts orientés objet. Il ne s'agit pas d'une simple notation graphique, car les concepts transmis par un diagramme ont une sémantique précise et sont porteurs de sens au même titre que les mots d'un langage.

### B.A.-BA OMG

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](http://www.omg.org)

UML 2 s'articule autour de treize types de diagrammes, chacun d'eux étant dédié à la représentation des concepts particuliers d'un système logiciel. Ces types de diagrammes sont répartis par l'OMG en deux grands groupes :

- six diagrammes structurels :

1. diagramme de classes (montre les briques de base statiques : classes, associations, interfaces, attributs, opérations, généralisations, etc.) ;
  2. diagramme d'objets (montre les instances des éléments structurels et leurs liens à l'exécution) ;
  3. diagramme de packages (montre l'organisation logique du modèle et les relations entre packages) ;
  4. diagramme de structure composite (montre l'organisation interne d'un élément statique complexe) ;
  5. diagramme de composants (montre des structures complexes, avec leurs interfaces fournies et requises) ;
  6. diagramme de déploiement (montre le déploiement physique des « artifacts » sur les ressources matérielles) ;
- sept diagrammes comportementaux :
    1. diagramme de cas d'utilisation (montre les interactions fonctionnelles entre les acteurs et le système à l'étude) ;
    2. diagramme de vue d'ensemble des interactions (fusionne les diagrammes d'activité et de séquence pour combiner des fragments d'interaction avec des décisions et des flots) ;
    3. diagramme de séquence (montre la séquence verticale chronologique des messages passés entre objets au sein d'une interaction) ;
    4. diagramme de communication (montre la communication entre objets dans le plan au sein d'une interaction) ;
    5. diagramme de temps (fusionne les diagrammes d'états et de séquence pour montrer l'évolution de l'état d'un objet au cours du temps) ;

6. diagramme d'activité (montre l'enchaînement des actions et décisions au sein d'une activité) ;
7. diagramme d'états (montre les différents états et transitions possibles des objets d'une classe).

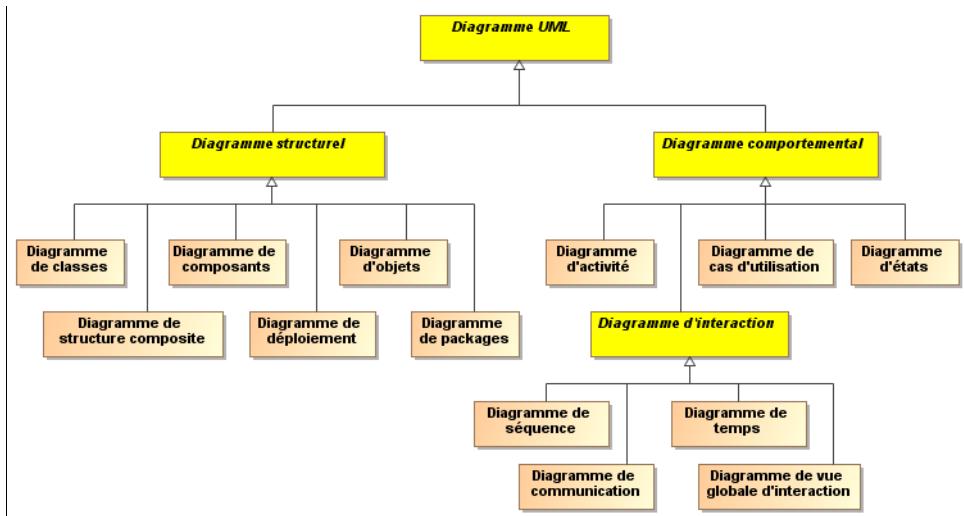


Figure 5 Les 13 types de diagrammes UML 2

## Lacunes d'UML pour l'IS

L'essor d'UML dans le domaine du logiciel et l'effort industriel de développement d'outils qui l'accompagne ont naturellement conduit à envisager son utilisation en ingénierie système. Cependant, du fait de sa conception fortement guidée par les besoins du

passage à la programmation par objets, le langage était, tout au moins dans ses premières versions, peu adapté à la modélisation des systèmes complexes et donc au support de l'ingénierie système.

Les nombreuses tentatives d'utilisation d'UML en IS ont en effet montré que certaines faiblesses d'UML devaient être comblées pour en faire un langage efficace pour les ingénieurs système. On peut citer :

- le besoin de décrire les exigences directement dans le modèle, et d'en assurer la traçabilité vers l'architecture ;
- le besoin de représenter des éléments non-logiciels et d'en spécifier le type (mécanique, circuit, hydraulique, câblage, capteur, etc.) ;
- le besoin de représenter des équations physiques, des contraintes ;
- le besoin de représenter des flux continus (matière, énergie, etc.) ;
- le besoin de représenter des allocations logique/physique, structure/dynamique, etc.

Quelques tentatives de profils UML ont vu le jour ces dernières années, comme SOC et MARTE.

#### RÉFÉRENCE MARTE

MARTE = Modeling and Analysis of Real Time and Embedded systems.

MARTE est le profil UML 2 pour la modélisation et l'analyse des systèmes temps réel embarqués. Il succède à UML Profile for Schedulability, Performance and Time. MARTE est conçu de manière à permettre l'utilisation de techniques d'analyse quantitative variées. Un des apports principaux de MARTE consiste en la définition de mécanismes décrivant le temps tels que les événements temporels et les horloges. Il supporte ainsi trois modèles temporels différents : le temps réel (chronométrique), le temps logique, et le temps logique synchrone. MARTE propose aussi un modèle de description des plateformes d'exécution, élément essentiel des systèmes temps réels embarqués.

► [www.omgmarте.org](http://www.omgmarте.org)

**RÉFÉRENCE System on a Chip (SoC)**

UML Profile for SoC Specification, OMG, version 1.0.1, formal/06-08-01, august 2006.

## SysML

Le monde du logiciel a fini par se mettre d'accord à la fin des années 90 sur un formalisme de modélisation unifié : UML. Par contre, de leur côté, les ingénieurs système n'ont pas réussi à faire émerger un langage de modélisation uniformisé, malgré les tentatives de standardisation des processus d'ingénierie système évoquées précédemment.

En 2003, l'INCOSE a décidé de faire d'UML ce langage commun pour l'IS. UML contenait en effet déjà à l'époque nombre de diagrammes indispensables, comme les diagrammes de séquence, d'états, de cas d'utilisation, etc. Le travail sur la nouvelle version UML 2, entamé à l'OMG à peu près à la même période, a abouti à la définition d'un langage de modélisation très proche du besoin des ingénieurs système, avec les améliorations notables aux diagrammes d'activité et de séquence, ainsi que la mise au point du diagramme de structure composite.

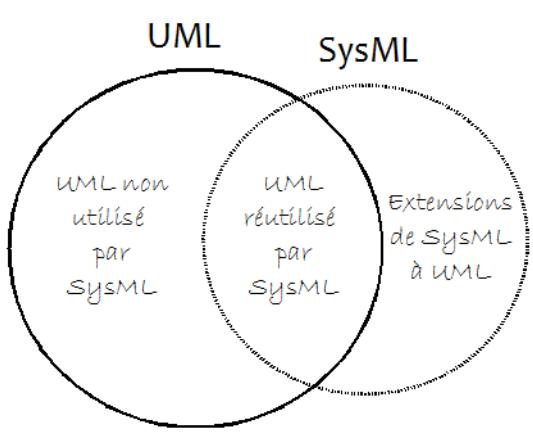
**B.A.-BA INCOSE**

L'International Council on Systems Engineering (INCOSE) est un organisme sans but lucratif, fondé en 1990. Sa mission est de faire progresser l'état de l'art et la pratique de l'Ingénierie Système dans l'industrie, les universités et les organismes gouvernementaux, par la promotion d'approches évolutives et interdisciplinaires visant à produire des solutions technologiques appropriées qui répondent aux besoins de la société.

► [www.incose.org](http://www.incose.org)

Cependant, il restait une barrière psychologique importante à l'adoption d'UML par la communauté de l'IS : sa teinture « logicielle » indélébile ! La possibilité d'extension d'UML, grâce au concept de stéréotype, a permis d'adapter le vocabulaire pour les ingénieurs système. En éliminant les mots « objet » et « classe » au profit du terme plus neutre « bloc », bref en gommant les aspects les plus informatiques d'UML, et en renommant ce langage de modélisation, l'OMG veut promouvoir SysML comme un nouveau langage différent d'UML, tout en profitant de sa filiation directe.

**Figure 6**  
SysML en tant que profil UML 2



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...

### B.A.-BA Profil UML

Un profil UML permet d'adapter UML à un domaine particulier ou à une problématique spécifique. Les profils mettent en jeu le concept central de stéréotype. Un stéréotype est une sorte d'étiquette nommée que l'on peut coller sur tout élément d'un modèle UML. Par exemple, coller un stéréotype « continuous » sur un flot dans un diagramme d'activité signifie que le flot en question n'est plus un flot UML standard, c'est-à-dire discret, mais qu'il est continu. SysML ajoute ainsi un certain nombre de concepts à UML 2 en définissant un ensemble de stéréotypes. Mais il simplifie également le langage de modélisation en laissant de côté certains concepts UML, jugés non indispensables.

### RÉFÉRENCE Site SysML

Toutes les informations officielles sur le langage SysML se trouvent sur un site web unique :  
► [www.omg.sysml.org](http://www.omg.sysml.org)

SysML s'articule autour de neuf types de diagrammes, chacun d'eux étant dédié à la représentation des concepts particuliers d'un système. Ces types de diagrammes sont répartis par l'OMG en trois grands groupes :

- quatre diagrammes comportementaux :
  1. diagramme d'activité (montre l'enchaînement des actions et décisions au sein d'une activité complexe) ;
  2. diagramme de séquence (montre la séquence verticale des messages passés entre blocs au sein d'une interaction) ;
  3. diagramme d'états (montre les différents états et transitions possibles des blocs dynamiques) ;
  4. diagramme de cas d'utilisation (montre les interactions fonctionnelles entre les acteurs et le système à l'étude) ;

- un diagramme transverse : le diagramme d'exigences (montre les exigences du système et leurs relations) ;
- quatre diagrammes structurels :
  1. diagramme de définition de blocs (montre les briques de base statiques : blocs, compositions, associations, attributs, opérations, généralisations, etc.) ;
  2. diagramme de bloc interne (montre l'organisation interne d'un élément statique complexe) ;
  3. diagramme paramétrique (représente les contraintes du système, les équations qui le régissent) ;
  4. diagramme de packages (montre l'organisation logique du modèle et les relations entre packages).

#### ATTENTION Différences UML 2/SysML

Les diagrammes UML 2 qui n'ont pas été retenus par SysML, principalement par souci de simplification, sont :

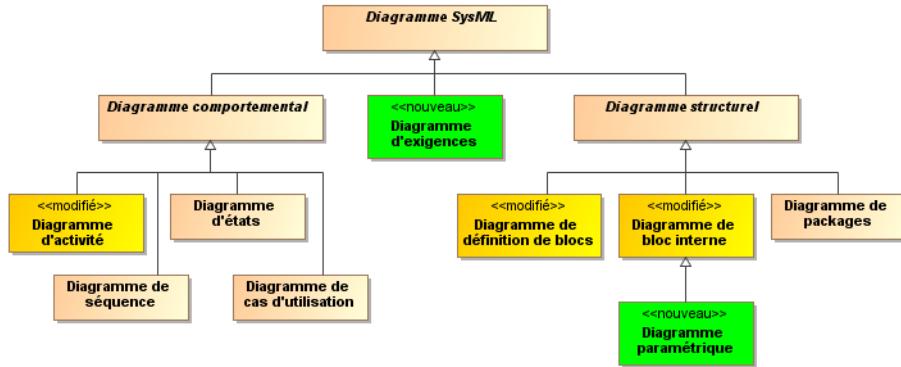
- le diagramme d'objets ;
- le diagramme de composants ;
- le diagramme de déploiement ;
- le diagramme de vue d'ensemble des interactions ;
- le diagramme de communication ;
- le diagramme de temps.

Récapitulons : 13 diagrammes (UML 2) – 6 + 2 (exigences, paramétrique) = 9 diagrammes SysML !

La structure du système est représentée par les diagrammes de blocs. Le diagramme de définition de blocs (*block definition diagram*) décrit la hiérarchie du système et les classifications système/composant. Le diagramme de bloc interne (*internal block diagram*) décrit

la structure interne du système en termes de parties, ports et connecteurs. Le diagramme de packages est utilisé pour organiser le modèle.

**Figure 7**  
Les 9 types  
de diagrammes  
SysML



Les diagrammes comportementaux incluent le diagramme de cas d'utilisation, le diagramme d'activité, le diagramme de séquence et le diagramme de machines à états. Le diagramme de cas d'utilisation fournit une description de haut niveau des fonctionnalités du système. Le diagramme d'activité représente les flots de données et de contrôle entre les actions. Le diagramme de séquence représente les interactions entre les parties du système qui collaborent. Le diagramme de machines à états décrit les transitions entre états et les actions que le système ou ses parties réalisent en réponse aux événements.

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.

Le diagramme paramétrique permet de représenter des contraintes sur les valeurs de paramètres système tels que performance, fiabilité, masse, etc. Il s'agit d'une spécialisation du diagramme de bloc interne où les seuls blocs utilisables sont des contraintes entre paramètres permettant de représenter graphiquement des équations et relations mathématiques. Ce nouveau diagramme fournit ainsi un support pour les études d'analyse système.

SysML inclut également une relation d'allocation pour représenter différents types d'allocation, comme celles de fonctions à des composants, de composants logiques à composants physiques, ainsi que de software à hardware.

## Organisation du livre

Après ce premier chapitre introductif, qui nous a permis de poser les bases du langage SysML, nous allons détailler un par un, dans la suite de ce livre, les différents types de diagrammes permettant de représenter un système suivant tous les points de vue souhaités.

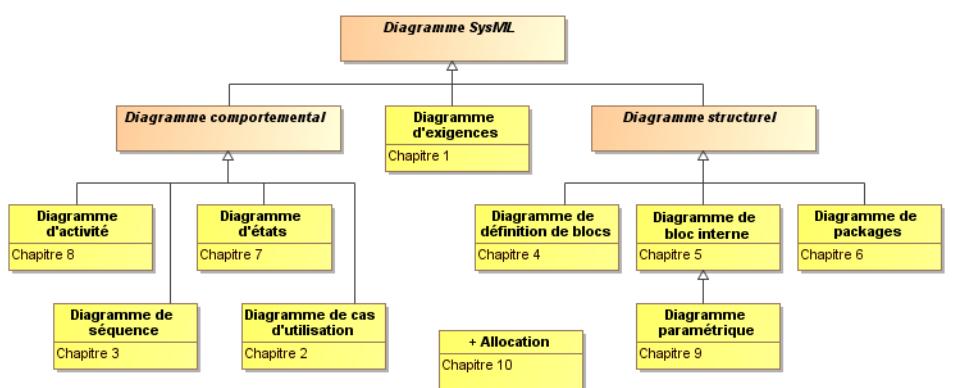
Cependant, à des fins pédagogiques, nous allons étudier les différents types de diagramme dans un ordre particulier, correspondant à une utilisation « naturelle » du langage SysML pour modéliser un système, en partant des exigences, en explicitant ensuite les aspects structurels, puis comportementaux et enfin transverses. Il est clair que dans la réalité, la démarche de modélisation ne saurait être si linéaire, comme l'impose cette présentation écrite, mais comprendrait plutôt des allers-retours entre les vues structurelles et dynamiques, ainsi qu'une utilisation itérative des mêmes types de diagrammes en allant du général au particulier.

Nous avons donc choisi de présenter dans l'ordre :

- le diagramme d'exigences ;

- le diagramme de cas d'utilisation ;
- le diagramme de séquence ;
- le diagramme de définition de blocs ;
- le diagramme de bloc interne ;
- le diagramme de packages ;
- le diagramme d'états ;
- le diagramme d'activité ;
- et le diagramme paramétrique.

La figure suivante replace chacun des chapitres suivants dans le cadre de la présentation des différents types de diagrammes. Il est à noter que le chapitre 10 traitera de l'important concept transverse d'allocation, bien que ne donnant pas lieu à un diagramme à part entière.



**Figure 8** Les chapitres du livre d'après les diagrammes SysML

Ce schéma général nous servira ainsi d'introduction pour chaque chapitre du livre, afin de bien le replacer dans la vision d'ensemble du langage de modélisation SysML.

## L'étude de cas du livre

La difficulté en ingénierie système est de trouver une étude de cas suffisamment représentative, mais pas trop complexe, ni trop liée à un domaine technique particulier.

Dans le cadre de ce livre, j'ai réutilisé un exemple qui m'a servi de nombreuses fois lors des sessions de formation SysML que j'ai réalisées depuis plus de deux ans. Il s'agit d'un radio-réveil à projecteur, système facile à comprendre par tout un chacun, mais dont la simplicité apparente est trompeuse. Je vous engage par exemple à consulter le diagramme d'états du chapitre 7 pour vous en convaincre !

**Figure 9**

Le radio-réveil de l'étude de cas



# PREMIÈRE PARTIE

## La modélisation des exigences

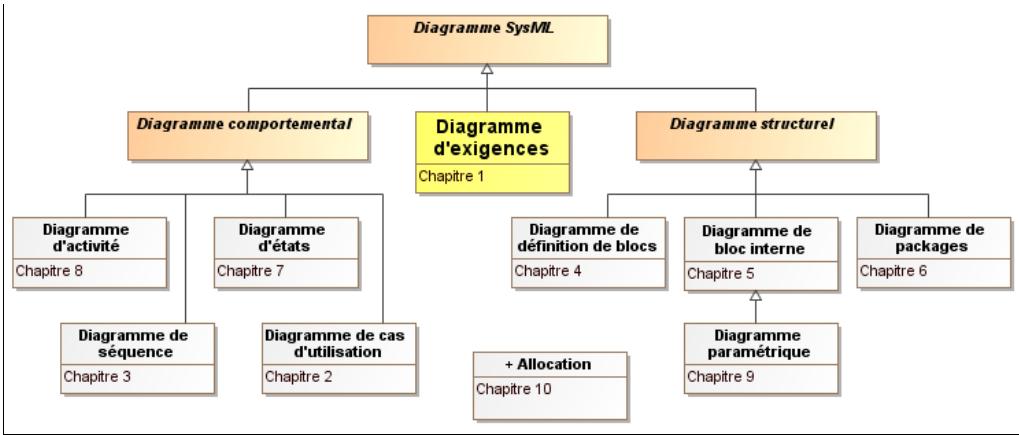
La partie I de l'ouvrage concerne la modélisation des exigences. Nous verrons que SysML innove en proposant un diagramme d'exigences permettant de dessiner graphiquement les exigences système et surtout de les relier ensuite aux éléments structurels ou dynamiques de la modélisation, ainsi qu'à des exigences de niveau sous-système ou équipement. Nous apprendrons également à mettre en œuvre la technique des cas d'utilisation, déjà présente en UML, au niveau système. Nous verrons enfin une première application du diagramme de séquence au niveau système « boîte noire ».

# 1

## Le diagramme d'exigences

Ce chapitre présente le diagramme d'exigences. Ce diagramme capture les hiérarchies d'exigences, ainsi que leurs relations de dérivation, de raffinement, de satisfaction et de vérification. Ces relations fournissent la capacité de relier les exigences les unes aux autres, ainsi qu'aux éléments de conception et aux cas de tests.

- ▶ exigence
- ▶ traçabilité
- ▶ cas de test



## Notation de base

Le diagramme d'exigences permet de représenter graphiquement les exigences dans le modèle.

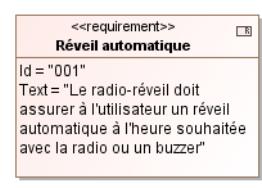
### B.A.-BA Exigence

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.

**Figure 1–1**

Première exigence du radio-réveil



Dans l'exemple du radio-réveil, la première exigence fondamentale concerne la capacité à assurer à l'utilisateur un réveil automatique à l'heure souhaitée avec la radio ou un buzzer. On peut également lister des exigences sur le réglage de la radio, de l'horloge et de l'alarme, ainsi que sur la nécessité d'un mécanisme de sauvegarde. Une première ébauche de diagramme d'exigences est donnée par la figure suivante.

#### B.A.-BA Cartouche de diagramme

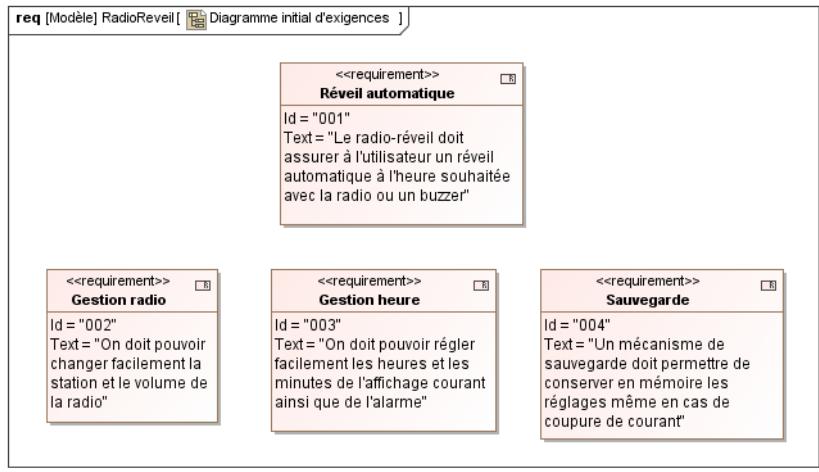
Un cartouche positionné en haut à gauche du diagramme dans un pentagone permet de spécifier le type de diagramme SysML, le type de l'élément concerné, l'élément concerné, et le nom du diagramme.

Dans l'exemple suivant, il s'agit d'un diagramme d'exigences (req) nommé « Diagramme initial d'exigences », concernant l'élément RadioRéveil de type Modèle.

Le cartouche général du diagramme d'exigences est de la forme :

**req** [package ou exigence] nom de l'élément [nom du diagramme]

**Figure 1–2**  
Diagramme initial  
d'exigences du radio-réveil



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.).

Nous pouvons enrichir le diagramme d'exigences précédent en ajoutant une exigence sur la capacité de projeter l'heure courante au plafond. Ce nouveau « requirement » aura comme statut : proposée, au lieu de validée pour les autres.

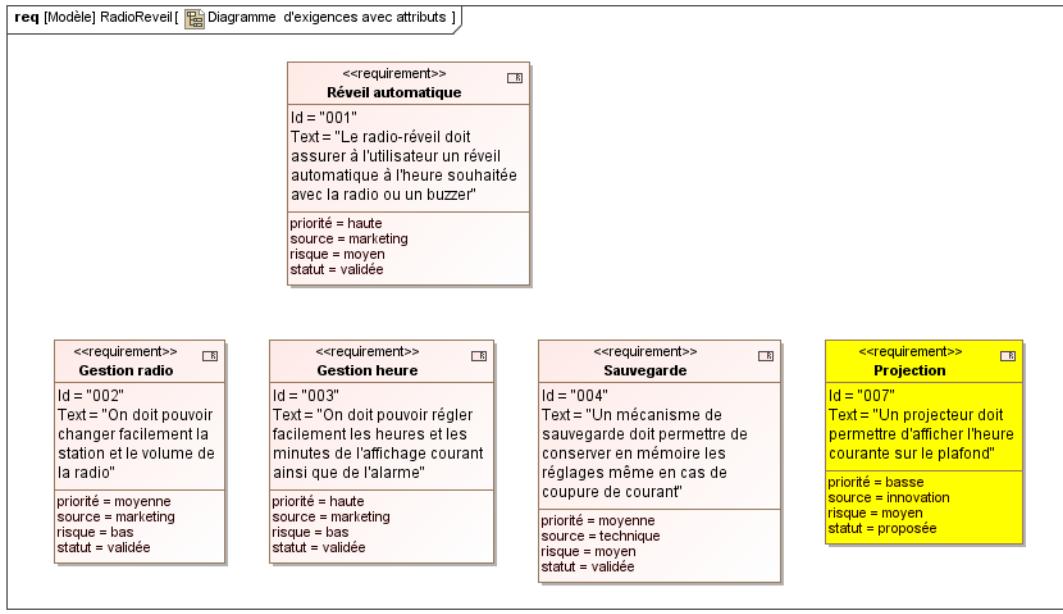


Figure 1–3 Exigences du radio-réveil avec attributs

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

- la contenance (ligne terminée par un cercle contenant une croix du côté du conteneur) permet de décomposer une exigence composite en plusieurs exigences unitaires, plus faciles ensuite à tracer vis-à-vis de l'architecture ou des tests ;

- le raffinement (« refine ») consiste en l'ajout de précisions, par exemple de données quantitatives ;
- la dérivation (« deriveReqt ») 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.

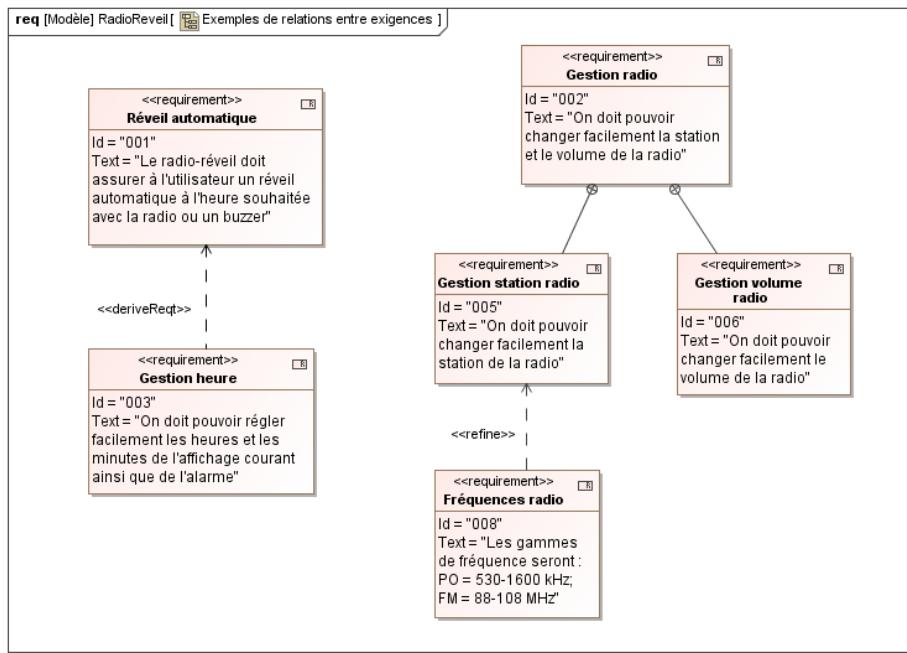


Figure 1-4 Exemple de relations entre les exigences du radio-réveil

Dans notre exemple, l'exigence sur la gestion de la radio est exprimée par une phrase contenant une conjonction de coordination « et ». Ceci est souvent le symptôme d'une exigence composite devant être décomposée en exigences unitaires, ce que nous avons fait avec les deux exigences sur le réglage de la station et le réglage du volume.

L'exigence sur le réglage de la station peut être raffinée en précisant les valeurs des gammes de fréquence devant être supportées.

Enfin, on peut considérer que la gestion de l'heure est une exigence qui dérive de l'exigence générale en ce sens qu'elle implique un niveau d'architecture supplémentaire, à savoir la notion d'horloge.

## Compléments

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` »).

Nous en verrons l'illustration sur la prochaine figure.

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

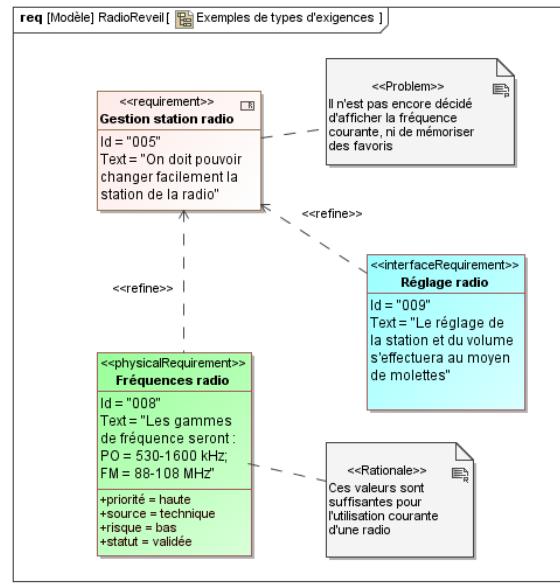
- fonctionnelle ;
- performance ;
- fiabilité ;
- sécurité ;

- 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.

**Figure 1–5**

Exemples de types d'exigence du radio-réveil



# 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 :

- exigence – élément comportemental (cas d'utilisation, diagramme d'états, etc.) : « `refine` » ;
- exigence – bloc d'architecture : « `satisfy` » ;
- exigence – cas de test : « `verify` ».

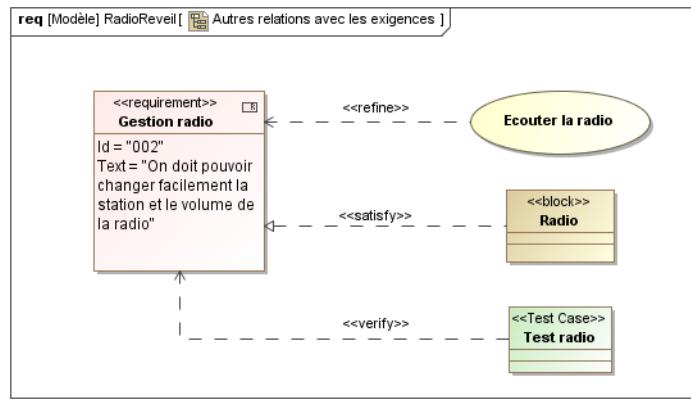
## B.A.-BA Cas de test

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` ».

Dans notre exemple, j'ai créé un cas d'utilisation *Écouter la radio* (voir le chapitre 2 sur les cas d'utilisation), un bloc d'architecture Radio (voir le chapitre 4) ainsi qu'un cas de test *Test radio*. J'ai ensuite relié ces nouveaux éléments à l'exigence de *Gestion radio* par les relations de traçabilité adéquates.

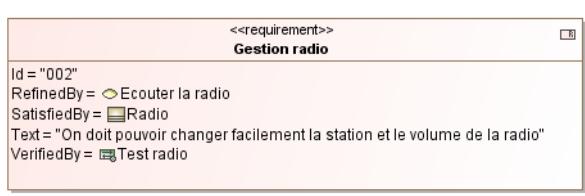
Une autre représentation graphique proposée par SysML consiste à faire apparaître les relations de traçabilité par une note attachée à l'exigence, ou encore par des attributs ou

**Figure 1–6**  
Autres relations avec les exigences du radio-réveil



des compartiments supplémentaires. Les deux figures qui suivent montrent ces différentes solutions.

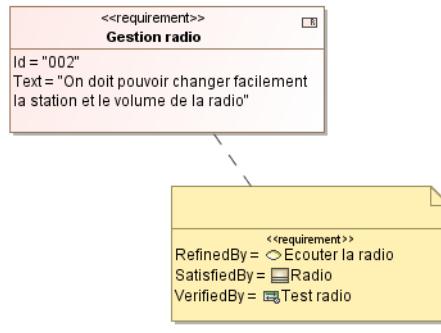
**Figure 1–7**  
Ajout de la traçabilité  
à l'intérieur d'une exigence



Une possibilité supplémentaire de représentation consiste à utiliser une forme tabulaire. C'est à la charge de l'outil de modélisation de proposer cette forme graphique, en général

**Figure 1–8**

Ajout de la traçabilité d'une exigence au moyen d'une note



calculée de façon automatique à partir des relations déclarées dans des diagrammes ou dans les fiches de propriété des éléments. Un exemple, réalisé par MagicDraw, est donné sur la figure suivante.

**Figure 1–9**

Exemple de représentation tabulaire de la relation « satisfy »

	Projecteur [Radio...]	Radio [Radio-réveil]	Réveil [Radio-réveil]
1	1	1	
1			
1			
1			

Below the table, there is a hierarchical tree structure under the heading "Exigences":

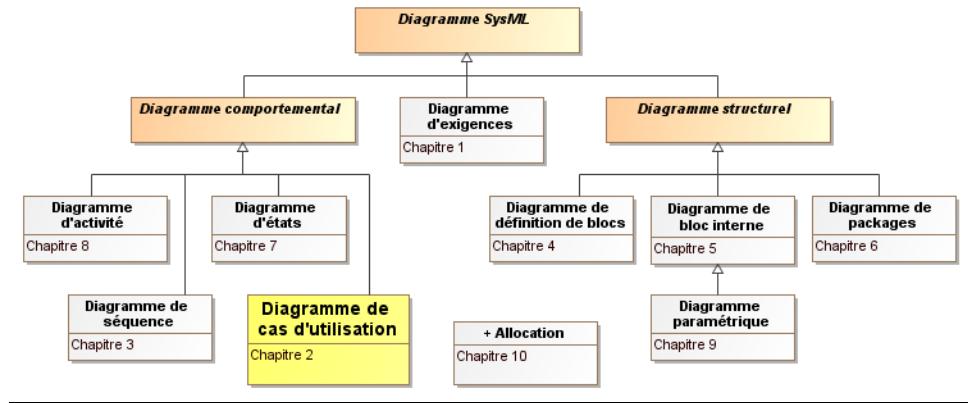
- Exigences
  - Gestion heure [Exigences]
  - Gestion radio [Exigences]
  - Projection [Exigences]

# 2

## Le diagramme de cas d'utilisation

Ce chapitre présente le diagramme de cas d'utilisation, qui fournit une description de haut niveau des fonctionnalités du système.

- ▶ cas d'utilisation
- ▶ acteur
- ▶ inclusion, extension
- ▶ généralisation



## Notation de base

Il est souhaitable de représenter les services attendus du système à l'étude par un modèle de cas d'utilisation. Ce modèle contient un ou plusieurs diagrammes de cas d'utilisation, montrant les interactions fonctionnelles entre les acteurs et le système à l'étude.

### B.A.-BA 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.

### B.A.-BA Cas d'utilisation

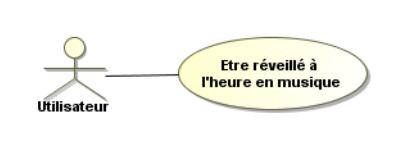
Un cas d'utilisation (use case, ou UC) représente un ensemble de séquences d'actions qui sont réalisées par le système et qui produisent un résultat observable intéressant pour un acteur particulier. Chaque cas d'utilisation spécifie un comportement attendu du système considéré comme un tout, sans imposer 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 ».

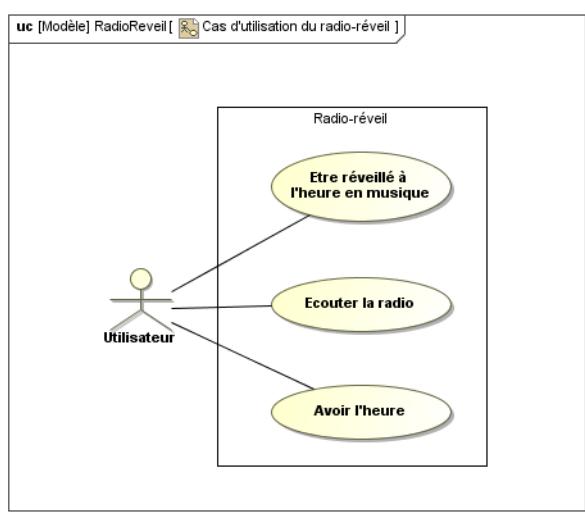
Pour notre étude de cas, une première version du diagramme de cas d'utilisation consiste à considérer un seul acteur (l'utilisateur) connecté à un unique cas d'utilisation (être réveillé à l'heure en musique).

**Figure 2–1**  
Acteur et cas d'utilisation



Ensuite, on peut se dire que l'utilisateur, alors qu'il est réveillé, est susceptible d'utiliser le radio-réveil en tant que simple radio ou horloge. Chaque cas d'utilisation doit bien représenter un service autonome rendu par le système et fournissant un résultat observable et intéressant pour l'acteur concerné. Si l'on dessine un rectangle englobant autour des cas d'utilisation pour matérialiser le radio-réveil, on obtient la figure suivante.

**Figure 2–2**  
Acteur et cas d'utilisation  
(suite)



#### ATTENTION Cas d'utilisation

Une erreur fréquente concernant les cas d'utilisation consiste à vouloir descendre trop bas en termes de granularité. Un cas d'utilisation représente un ensemble de séquences d'actions réalisées par le système, et le lien entre ces séquences d'actions est précisément l'objectif métier de l'acteur. Le cas d'utilisation ne doit donc pas se réduire systématiquement à une seule séquence, et encore moins à une simple action.

Limitez à 20 le nombre de vos cas d'utilisation de base (en dehors des cas inclus, spécialisés, ou des extensions). Avec cette limite arbitraire, on reste synthétique et on ne tombe pas dans le piège de la granularité trop fine des cas d'utilisation.

Les acteurs candidats sont systématiquement :

- les utilisateurs humains directs : faites donc en sorte d'identifier tous les profils possibles, sans oublier l'administrateur, l'opérateur de maintenance, etc. ;
- les autres systèmes connexes qui interagissent aussi directement avec le système étudié, souvent par le biais de protocoles bidirectionnels.

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.

#### ATTENTION Acteurs

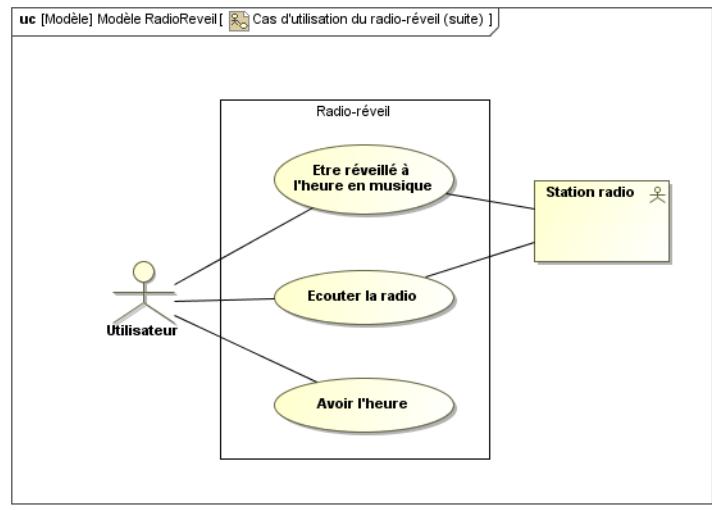
Ne confondez pas rôle et entité concrète. Une même entité concrète peut jouer successivement différents rôles par rapport au système étudié, et être modélisée par plusieurs acteurs. Réciproquement, le même rôle peut être tenu simultanément par plusieurs entités concrètes, qui seront alors modélisées par le même acteur.

Nous appelons acteur principal celui pour qui le cas d'utilisation produit un résultat observable. Par opposition, nous qualifions d'acteurs secondaires les autres participants du cas d'utilisation. Les acteurs secondaires sont souvent sollicités pour des informations complémentaires ; ils peuvent uniquement consulter ou informer le système lors de l'exécution du cas d'utilisation.

Une bonne pratique consiste à faire figurer les acteurs principaux à gauche des cas d'utilisation, et les acteurs secondaires à droite. Si nous ajoutons les stations radio en tant qu'acteurs secondaires pour les cas d'utilisation du radio-réveil liés à la radio, nous obtenons la figure suivante.

**Figure 2-3**

Acteurs principal humain et secondaire non-humain



#### B.A.-BA Cartouche

Le cartouche général du diagramme de cas d'utilisation est de la forme :

**uc** [package ou bloc] nom de l'élément [nom du diagramme]

# Décrire les cas d'utilisation

Chaque cas d'utilisation doit être décrit textuellement (à l'aide d'un plan-type). On peut également associer à chaque cas d'utilisation un ou plusieurs diagrammes de séquence comme expliqué dans le chapitre qui suit.

La fiche de description textuelle d'un cas d'utilisation n'est pas normalisée par SysML. Nous préconisons pour notre part la structuration suivante :

- sommaire d'identification (obligatoire) : inclut titre, résumé, dates de création et de modification, version, responsable, acteurs, etc. ;
- description des scénarios (obligatoire) : décrit le scénario nominal, les scénarios (ou enchaînements) alternatifs, les scénarios (ou enchaînements) d'échec, mais aussi les préconditions (l'état du système pour que le cas d'utilisation puisse démarrer) et les postconditions (ce qui a changé dans l'état du système à la fin du cas d'utilisation) ;

## B.A.-BA 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 ».

- exigences non fonctionnelles (optionnel) : ajoute, si c'est pertinent, les informations suivantes : fréquence, volumétrie, disponibilité, fiabilité, intégrité, confidentialité, performances, concurrence, etc. Précise également les contraintes d'interface homme-machine comme des règles d'ergonomie, une charte graphique, etc.

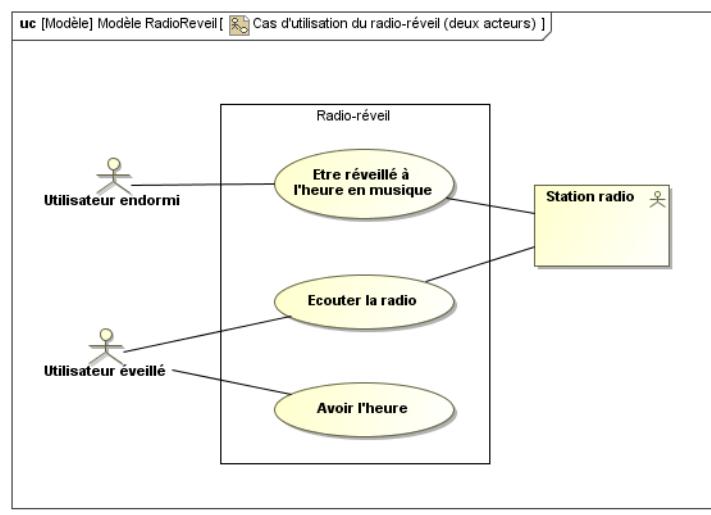
# Compléments

## Classification des acteurs

On pourrait imaginer distinguer les cas d'utilisation du radio-réveil selon que l'utilisateur est endormi ou déjà réveillé. En effet, c'est l'utilisateur endormi qui souhaite être réveillé à l'heure, alors que c'est l'utilisateur éveillé qui va écouter la radio ou regarder l'heure. Une version alternative du diagramme de cas d'utilisation est donnée sur la figure suivante.

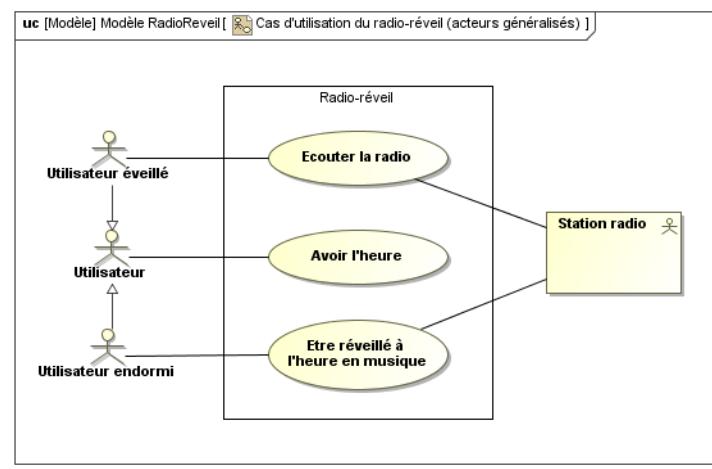
**Figure 2–4**

Première variante du diagramme de cas d'utilisation



**Figure 2–5**

Variante du diagramme avec généralisation d'acteurs



Mais on peut même aller un peu plus loin : grâce au dispositif de projection au plafond, l'utilisateur (à moitié) endormi peut lui aussi avoir l'heure ! On pourrait ainsi considérer que tout utilisateur peut avoir l'heure, qu'il soit endormi ou réveillé, mais que seul l'utilisateur endormi veut se faire réveiller et que seul l'utilisateur éveillé peut choisir d'écouter la radio.

Pour éviter d'avoir deux acteurs principaux (utilisateur endormi et utilisateur éveillé) connectés au même cas d'utilisation, SysML permet de créer un acteur généralisé *Utilisateur* qui factorise les comportements communs aux deux acteurs. On dit alors que les acteurs *utilisateurendormi* et *utilisateurréveillé* sont des spécialisations de l'acteur *Utilisateur*. Ils peuvent chacun posséder leurs propres cas d'utilisation spécifiques.

### B.A.-BA 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.

## Relations entre cas d'utilisation

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

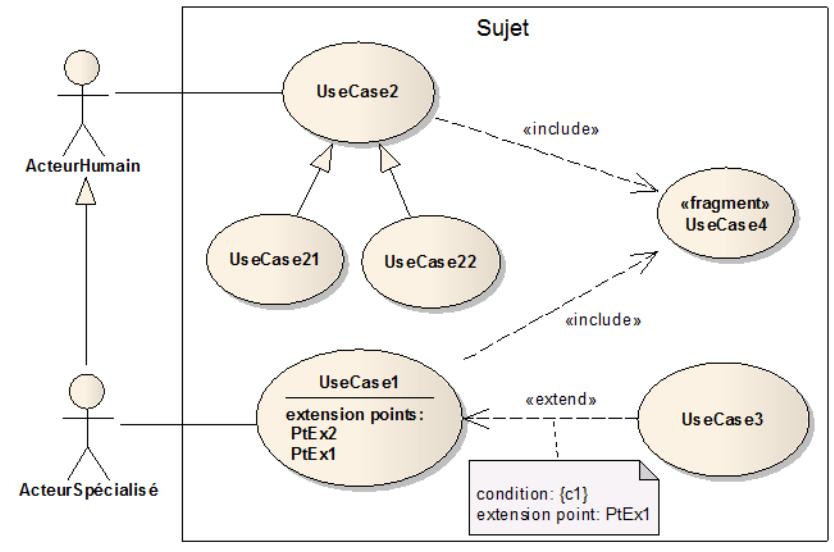
- une relation d'**inclusion**, formalisée par le mot-clé « `include` » : le cas d'utilisation de base en incorpore explicitement un autre, de façon obligatoire.
- une relation d'**extension**, formalisée par le mot-clé « `extend` » : le cas d'utilisation de base en incorpore implicitement un autre, de façon optionnelle, à un endroit spécifié indirectement dans celui qui procède à l'extension (appelé *extension point*).
- une relation de **généralisation**/spécialisation (flèche blanche) : les cas d'utilisation descendants héritent la description de leur parent commun. Chacun d'entre eux peut néanmoins comprendre des interactions spécifiques supplémentaires.

La représentation graphique de toutes ces relations est donnée à la figure 2-6.

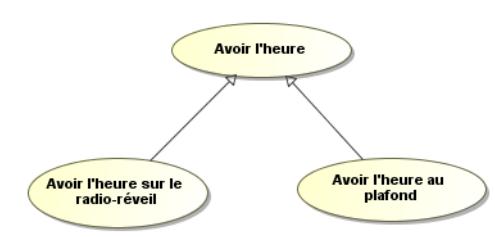
Essayons de donner quelques exemples sur notre étude de cas.

Le cas d'utilisation *Avoir l'heure* pourrait se spécialiser suivant que la lecture de l'heure se fait directement sur le radio-réveil ou alors au plafond.

**Figure 2–6**  
Notations avancées  
du diagramme de cas  
d'utilisation

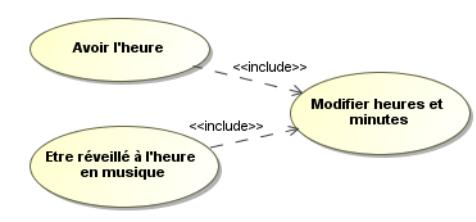


**Figure 2–7**  
Généralisation/spécialisation  
de cas d'utilisation



**Figure 2–8**

Inclusion de cas d'utilisation

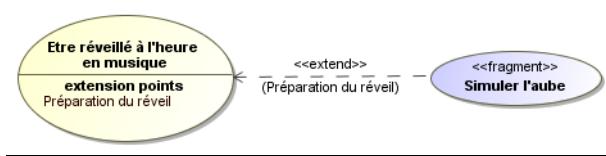


Pour la relation d'extension, souvent mal utilisée dans la pratique, nous pourrions prendre en compte une fonctionnalité optionnelle, telle que le simulateur d'aube (la lumière augmente progressivement pendant 30 à 90 minutes avant l'heure de réveil : le contact de la lumière avec les paupières a une incidence positive sur les rythmes hormonaux et prépare l'organisme au réveil...).

Ce nouveau cas d'utilisation peut être stéréotypé « fragment » afin d'exprimer le fait qu'il ne s'exécutera jamais de façon autonome, mais toujours dans le cadre d'un autre cas d'utilisation. Il s'agit d'une bonne pratique préconisée par de nombreux spécialistes, mais absolument pas obligatoire.

**Figure 2–9**

Extension de cas d'utilisation

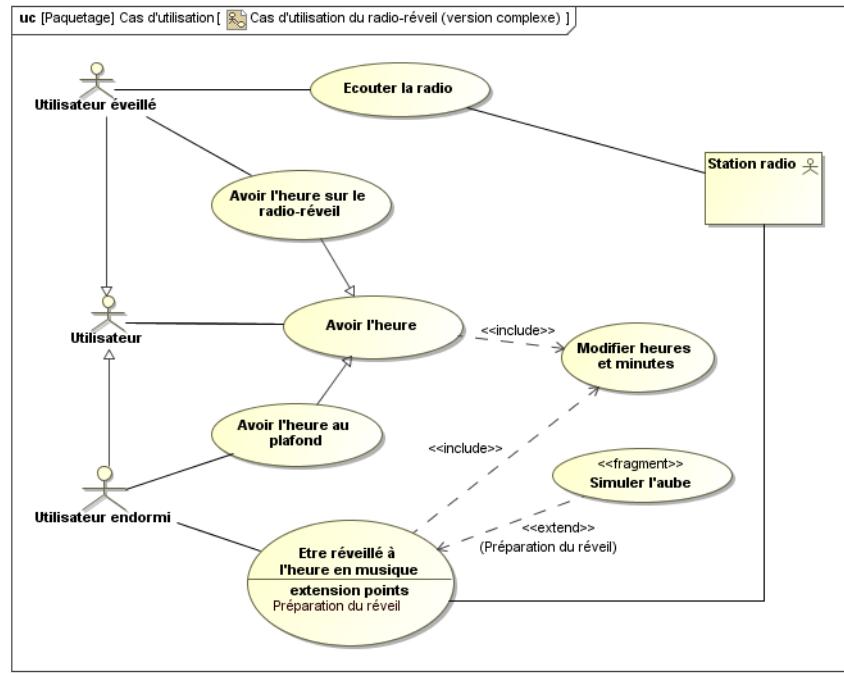


Une version complexe du diagramme de cas d'utilisation, incorporant toutes les relations possibles entre acteurs et cas d'utilisation est donnée par le schéma suivant. Il s'agit d'un exemple de notation, pas d'une recommandation méthodologique ! En effet, nous considérons que le diagramme de la figure 2–3, page 39, est probablement tout à fait suffisant dans notre cas.

**ATTENTION Relations entre UC**

N'abusez pas des relations entre cas d'utilisation (inclusion, extension, généralisation) : elles peuvent rendre les diagrammes de cas d'utilisation trop difficiles à déchiffrer pour les experts métier qui sont censés les valider.

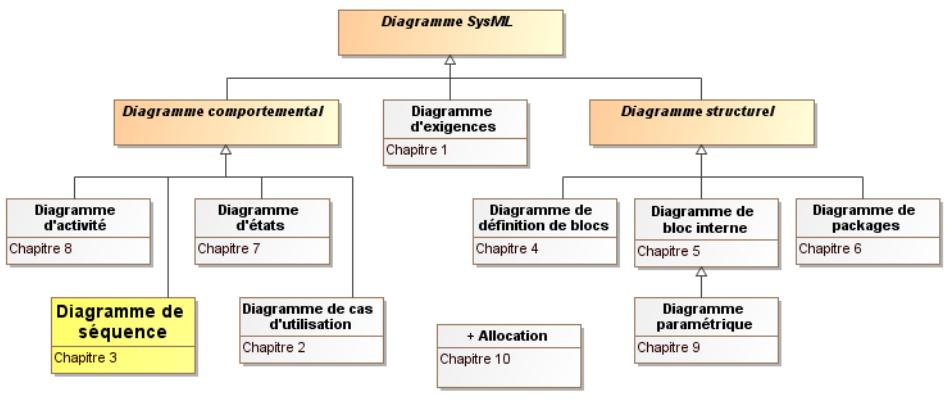
**Figure 2-10**  
Version complexe  
du diagramme de cas  
d'utilisation



# Le diagramme de séquence « système »

Ce chapitre présente le diagramme de séquence « système ». Pour documenter les cas d'utilisation, la description textuelle est indispensable, car elle seule permet de communiquer facilement avec les utilisateurs et de s'entendre sur le vocabulaire « métier » employé. En revanche, le texte présente des désavantages puisqu'il est difficile de montrer comment les enchaînements se succèdent, ou à quel moment les acteurs secondaires sont sollicités. En outre, la maintenance des évolutions s'avère souvent fastidieuse. Il est donc recommandé de compléter la description textuelle par un ou plusieurs diagrammes de séquence SysML.

- ▶ diagramme de séquence
- ▶ interaction
- ▶ scénario



## Notation de base

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

### B.A.-BA 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 vertillée.

### B.A.-BA 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.

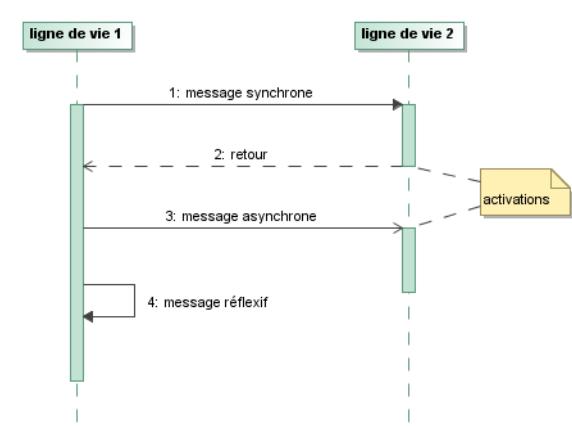
La flèche pointillée représente un retour. Cela signifie que le message en question est le résultat direct du message précédent. Un message synchrone (émetteur bloqué en attente de réponse) est représenté par une flèche pleine, alors qu'un message asynchrone est représenté par une flèche évidée. La flèche qui boucle (message réflexif) permet de représenter un comportement interne.

### B.A.-BA Activation

Les bandes verticales le long d'une ligne de vie représentent des périodes d'activation. Elles sont optionnelles, mais permettent de mieux comprendre la flèche pointillée du message de retour. Toutefois, dans un souci de simplicité, nous ne l'utiliserons généralement pas.

**Figure 3-1**

Notation de base du diagramme de séquence



## Diagramme de séquence « système »

Pour les messages propres à un cas d'utilisation, les diagrammes de séquence « système » montrent non seulement les acteurs externes qui interagissent directement avec le système, mais également ce système (en tant que boîte noire) et les événements système déclenchés par les acteurs. L'ordre chronologique se déroule vers le bas et l'ordre des messages doit suivre la séquence décrite dans le cas d'utilisation.

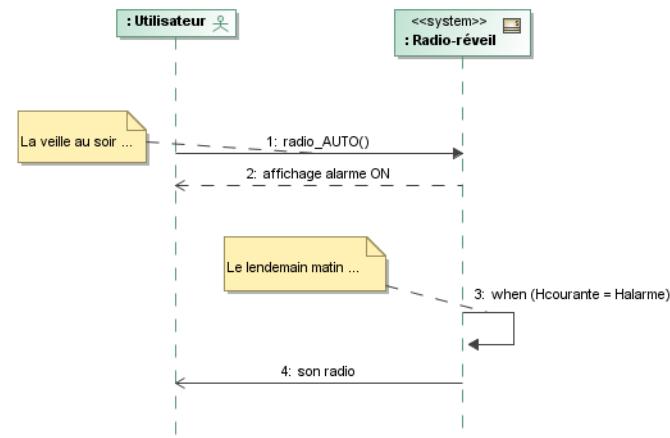
### B.A.BA Diagramme de séquence « système » (dss)

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 réalise.

Nous recommandons de présenter les dss en montrant l'acteur principal à gauche, puis une ligne de vie unique représentant le système en boîte noire, et, enfin, les éventuels acteurs secondaires sollicités durant le scénario à droite du système.

Un premier exemple de dss du cas d'utilisation *Être réveillé à l'heure en musique* est donné à la figure suivante. Remarquez l'utilisation de notes (post-it) pour mieux documenter le diagramme. Le premier message est un message synchrone, donnant lieu à un retour : l'affichage d'un point à côté de l'heure indiquant que l'alarme est positionnée. Le fait que radio-réveil détecte que l'heure courante devient égale à l'heure d'alarme est représenté par un message réflexif avec le mot-clé *when*. Le dernier message est un signal asynchrone.

**Figure 3-2**  
Dss simple du cas  
Être réveillé à l'heure



## Compléments

### Fragments combinés

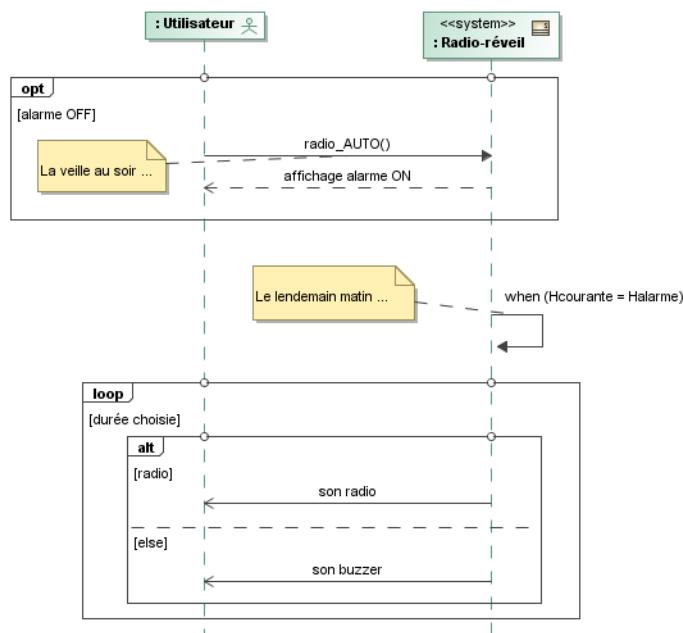
SysML propose une notation très utile : le fragment combiné. Chaque fragment possède un opérateur et peut être divisé en opérandes. Les principaux opérateurs sont :

- **loop** : boucle. Le fragment peut s'exécuter plusieurs fois, et la condition de garde explicite l'itération ;
- **opt** : optionnel. Le fragment ne s'exécute que si la condition fournie est vraie ;
- **alt** : fragments alternatifs. Seul le fragment possédant la condition vraie s'exécutera.

Essayons d'illustrer ces trois opérateurs sur l'exemple précédent.

Le son qui sort de la radio est continu pendant plusieurs minutes, ce n'est pas un simple signal unitaire. Pour le représenter, positionnons un fragment de boucle. En fait, l'utilisateur sera réveillé par la radio ou le buzzer suivant son choix. Nous pouvons donc ajouter un fragment `alt` avec deux opérandes. Enfin, le premier message n'est pas nécessaire si l'alarme était déjà positionnée la veille : il est donc optionnel. Le dss correspondant devient comme illustré sur la figure suivante.

**Figure 3-3**  
Dss avec fragments du cas  
Être réveillé à l'heure



## Cadre référence

Un diagramme de séquence peut en référencer un autre grâce à un second type de rectangle avec le mot-clé `ref`. Cette notation est très pratique pour modulariser les diagrammes de séquence, et créer des hyperliens graphiques exploitables par les outils de modélisation.

En fait, dans le scénario nominal du cas d'utilisation étudié, l'utilisateur a la possibilité avant de se coucher de modifier les réglages de l'horloge et de la radio. Si nous ne voulons pas décrire le détail de ces interactions dans le même diagramme de séquence, il suffit d'utiliser des cadres `ref` optionnels.

Pour compléter, nous allons illustrer un quatrième opérateur de fragment combiné : `par` (parallèle). Grâce à ce mot-clé, plusieurs opérandes pourront être exécutés en parallèle, ce qui ne serait pas facile à modéliser avec un diagramme de séquence classique.

À l'heure d'alarme, en parallèle de l'activation de la radio, le projecteur est également allumé (sauf bien sûr s'il l'était déjà).

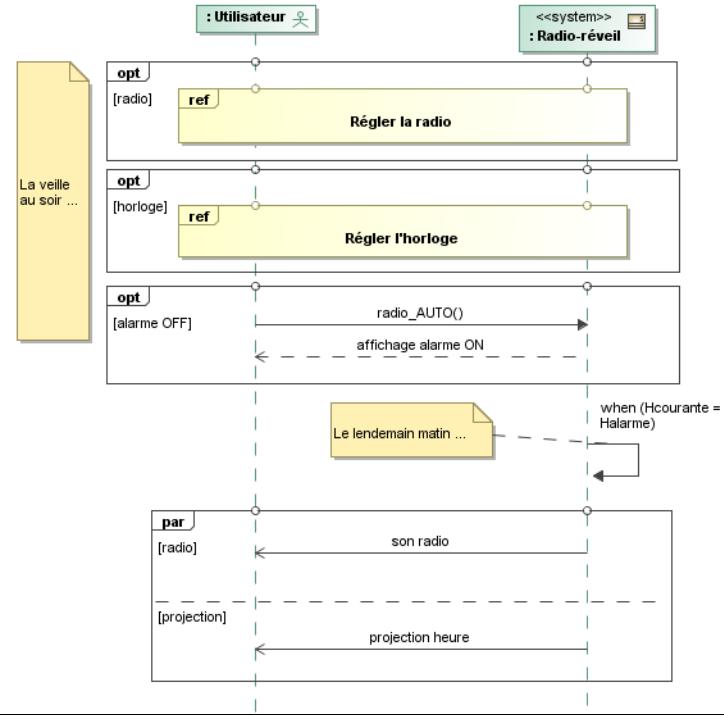
### B.A.-BA Cartouche

Le cartouche général du diagramme de séquence est de la forme–:

**sd** [interaction] nom de l'interaction [nom du diagramme]

**Figure 3–4**

Dss avec références du cas  
Être réveillé à l'heure



## Contraintes temporelles

SysML permet d'ajouter des contraintes temporelles sur le diagramme de séquence.

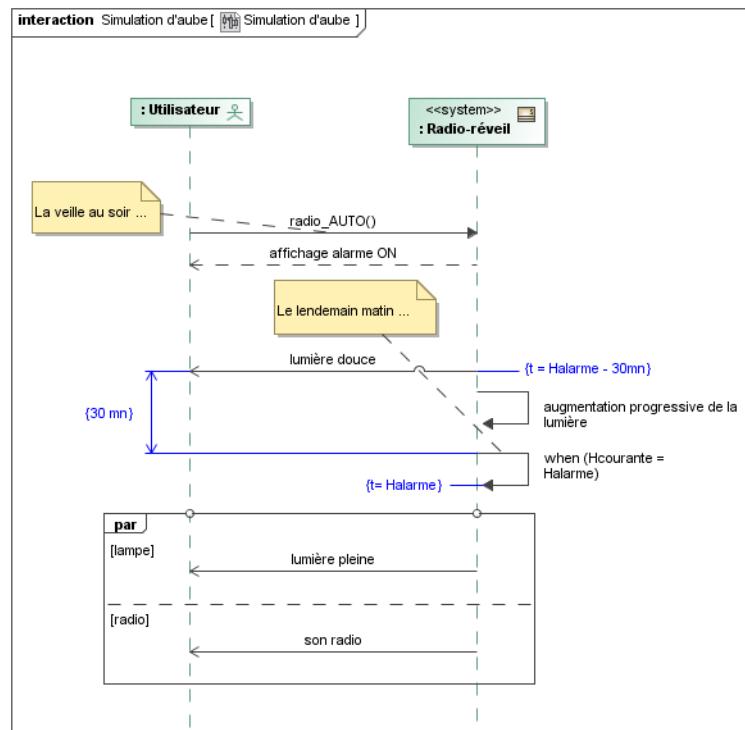
Il existe deux types de contraintes :

- la contrainte de durée, qui permet d'indiquer une contrainte sur la durée exacte, la durée minimale ou la durée maximale entre deux événements ;
- la contrainte de temps, qui permet de positionner des labels associés à des instants dans le scénario au niveau de certains messages et de les relier ainsi entre eux.

Nous avons modélisé le simulateur d'aube en utilisant les deux types de contraintes. Dans la réalité, il aurait fallu choisir l'un ou l'autre, mais cela nous permet d'illustrer les deux cas.

Pour simuler l'aube, nous avons supposé que la lampe commence à émettre doucement trente minutes avant l'heure du réveil. Le message 3 : `lumière douce` modélise ce début d'éclairage. La contrainte de durée est représentée par une double flèche en prolongement de ce message et du déclenchement de l'alarme (message 5), avec la durée entre accolades : {30 mn}. La contrainte de temps, pour sa part, est représentée en associant une contrainte { $t = \text{Halarme}$ } en prolongement du message 5, et une autre contrainte { $t = \text{Halarme} - 30 \text{ mn}$ } en prolongement du message 3.

**Figure 3-5**  
Dss avec contraintes temporelles



# DEUXIÈME PARTIE

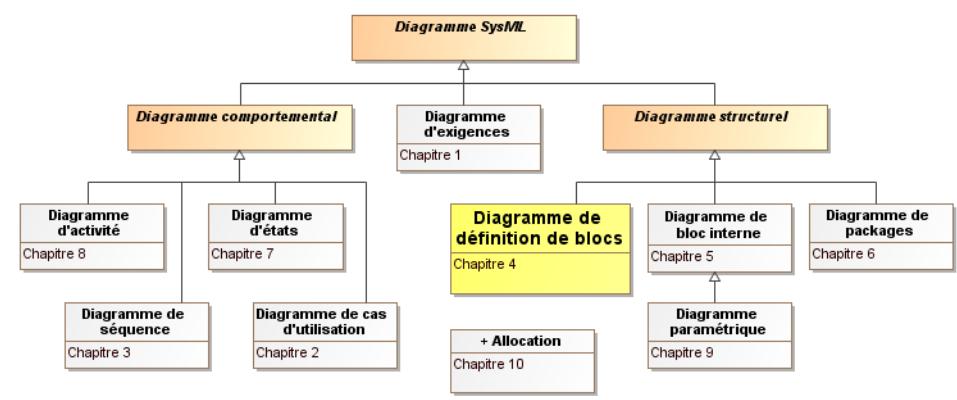
## La modélisation d'architecture

La partie II concerne la modélisation structurelle. Nous apprendrons à utiliser le concept universel de « bloc » proposé par SysML pour modéliser tout élément structurel, ainsi que les deux types de diagrammes associés. Nous verrons tout d'abord comment définir les éléments structurels de base de notre modèle dans le diagramme de définition de blocs. Nous apprendrons ensuite à décrire la décomposition des éléments complexes avec le diagramme interne de bloc. Nous verrons enfin comment structurer notre modèle en packages, à des fins de travail en équipe ou de réutilisation.

# Diagramme de définition de blocs

Ce chapitre présente le diagramme de définition de blocs. Le bloc SysML (« block ») constitue la brique de base pour la modélisation de la structure d'un système. Il peut représenter un système complet, un sous-système ou un composant élémentaire. Les blocs sont décomposables et peuvent posséder un comportement. Le diagramme de définition de blocs (block definition diagram ou bdd) décrit la hiérarchie du système et les classifications système/composant.

- ▶ bloc et bdd
- ▶ valeur, propriété, value type
- ▶ composition, agrégation, association et généralisation



## Bloc et propriété

Le bloc SysML (*block*) constitue la brique de base pour la modélisation de la structure d'un système. Il peut représenter un système complet, un sous-système ou un composant élémentaire. Le bloc permet de décrire également les flots qui circulent à travers un système.

Les blocs sont décomposables et peuvent posséder un comportement. On peut s'en servir pour représenter des entités physiques, mais aussi des entités logiques ou conceptuelles.

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

- les valeurs (*value properties*) décrivent des caractéristiques quantifiables en terme de *value types* (domaine de valeur, dimension et unité optionnelles) ;

- les parties (*part properties*) décrivent la hiérarchie de décomposition du bloc en termes d'autres blocs.

Par exemple, si l'on considère le bloc Voiture, on peut définir des valeurs :

- numéro d'immatriculation ;
- kilométrage ;
- vitesse courante ;
- etc.

La valeur vitesse courante sera typée par un *value type* Vitesse, lui-même défini par un type de base réel, une dimension Distance/temps et une unité km/h. Dans les pays anglo-saxons, l'unité pourra être miles/h.

Pour ce même bloc, on pourra définir des parties telles que :

- un moteur ;
- quatre roues ;
- quatre portes ;
- un coffre ;
- etc.

Chaque bloc (ou type) définit un ensemble d'instances partageant les propriétés du bloc, mais possédant chacune une identité unique. Par exemple : ma voiture, de numéro d'immatriculation : 2009 UML 31, et de kilométrage : 92 000, est une instance du bloc Voiture.

Le diagramme de définition de bloc est utilisé pour représenter les blocs, leurs propriétés, leurs relations.

### B.A.-BA Cartouche

Le cartouche général du diagramme de définition de bloc est de la forme :  
**bdd** [package ou bloc] nom de l'élément [nom du diagramme]

Dans un bdd, un bloc est représenté graphiquement par un rectangle découpé en compartiments. Le nom du bloc apparaît tout en haut, et constitue l'unique compartiment obligatoire. Tous les autres compartiments ont des labels indiquant ce qu'ils contiennent : valeurs, parties, etc.

Le mot-clé « **block** » apparaît par défaut, sauf si nous définissons de nouveaux mots-clés tels que « **system** », « **subsystem** », etc. Dans l'exemple suivant, nous avons modélisé le radio-réveil en tant que système à l'étude, avec une valeur : couleur, et deux parties : radio et réveil.

**Figure 4-1**  
Le bloc radio-réveil  
en tant que système

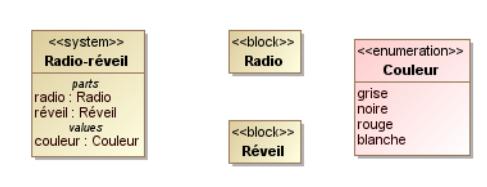


Nous pouvons préciser le type des parties et des valeurs :

- la partie radio est de type Radio (un nouveau bloc à créer) ;
- la partie réveil est de type Réveil (un nouveau bloc à créer) ;
- la valeur couleur est de type Couleur (une énumération à créer).

**Figure 4-2**

Le bloc radio-réveil avec quelques propriétés typées



#### NOTA MagicDraw et les paramètres de blocs

La couleur de fond des blocs, l'énumération, etc. est automatiquement gérée par MagicDraw qui différencie certains concepts. J'ai gardé majoritairement les couleurs standards proposées par l'outil, sauf quand j'ai trouvé qu'une modification pouvait améliorer le confort du lecteur.

## ValueType

Les valeurs (*value properties*) sont utilisées pour modéliser les caractéristiques quantitatives des blocs. Il est très important de bien définir les types de ces valeurs en termes de *value types* réutilisables.

SysML permet d'associer à chaque type de valeur une dimension et une unité optionnelles. On peut également spécifier une valeur initiale (nom valeur = valeur initiale), ainsi qu'une multiplicité (1 par défaut).

#### B.A.-BA Multiplicité

Une multiplicité est un intervalle entre une borne inférieure et une borne supérieure :

- la borne inférieure peut-être 0 (optionnelle) ou n'importe quel entier positif ;
- la borne supérieure peut être 1, plusieurs (noté \*), ou un entier positif.

La multiplicité est notée entre crochets. Si les bornes sont égales, on n'écrit qu'une valeur et la valeur par défaut en SysML est [1].

Les *value types* sont basés sur les types de base proposés par SysML, à savoir :

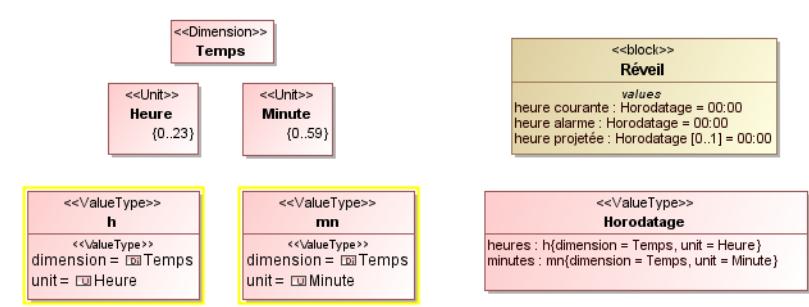
- les types primitifs : *integer*, *string*, *boolean*, *real*, etc. ;
- les énumérations qui définissent un ensemble de valeurs nommées (comme la couleur dans le paragraphe précédent) ;
- les types structurés, permettant de définir plusieurs champs, chacun étant à son tour une valeur.

Reprenons l'exemple du radio-réveil : l'heure courante affichée par le réveil, l'heure d'alarme, et l'heure projetée au plafond sont toutes les trois du même type. Il est important de définir une fois pour toutes un *value type* Horodatage, contenant deux champs (heures et minutes), par exemple : 23:59. On parle ici de *value type* structuré, lui-même basé sur deux types simples h et mn, définis avec une dimension Temps et chacun une unité.

L'ensemble de ces définitions est représenté sur le diagramme suivant. Notez les contraintes sur les unités permettant de borner les valeurs possibles, ainsi que les valeurs par défaut dans le bloc Réveil. Nous avons également illustré la notion de multiplicité en indiquant que l'heure projetée est optionnelle (projecteur allumé ou pas) : [0..1].

**Figure 4–3**

Le bloc Réveil avec l'ensemble des value types



### B.A.-BA Contrainte

Une contrainte est simplement une condition portant sur un ou plusieurs éléments du modèle qui doit être vérifiée par les éléments concernés. Elle est notée entre accolades { }, et peut être insérée au besoin dans une note graphique (le post-it).

## Partie

Revenons sur la notion de partie (*part property*) introduite au premier paragraphe.

Il s'agit d'une relation de composition entre blocs, aussi appelé relation « tout-partie », dans laquelle un bloc représente le tout, et les autres ses parties. Une instance du tout peut contenir plusieurs instances d'une partie, grâce à la notion de multiplicité, évoquée précédemment.

Les parties (*parts*) peuvent être listées dans un compartiment du bloc, avec le format suivant :

`nom partie : nom bloc [multiplicité]`

Si nous reprenons l'exemple de la voiture qui contient quatre roues, une première représentation en est donnée sur la figure suivante. Dans cet exemple, la voiture est le tout, et les roues sont représentées par des parties. Chacune des quatre roues a une définition commune donnée par le bloc Roue.

Figure 4–4

Premier bdd du bloc Voiture

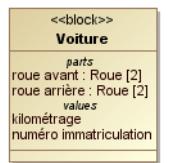


## Composition

La relation de composition entre blocs, dans laquelle un bloc représente le tout et les autres ses parties, peut également être représentée graphiquement. Le côté du tout est indiqué par un losange plein. La multiplicité de ce même côté ne peut être que 1 ou 0..1, car une instance de partie ne peut exister que dans une instance de tout au maximum à un moment donné. Dans notre exemple, nous allons préciser qu'une roue peut ne pas appartenir à une

**Figure 4–5**

Deuxième bdd du bloc Voiture



Une solution encore plus détaillée consiste à faire apparaître quatre parties de même type :

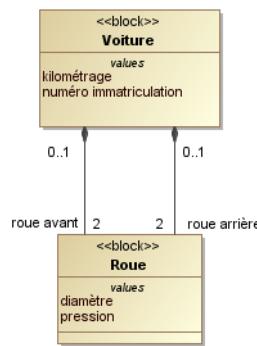
- roue avant gauche : Roue ;
- roue avant droite : Roue ;
- roue arrière gauche : Roue ;
- roue arrière droite : Roue.

## Agrégation

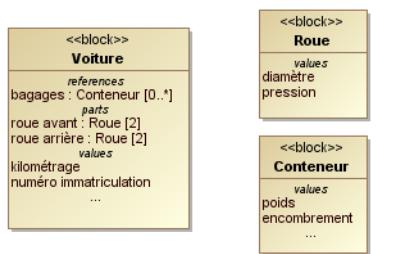
La relation d'agrégation (losange vide) est beaucoup moins forte que la relation de composition (losange plein). En particulier, il n'y a pas de contrainte de multiplicité du côté du tout, et donc pas nécessité d'une structure stricte d'arbre. Mais il y a d'autres cas où l'agrégation est utile : pour représenter le fait que la contenance n'est pas vraiment structurelle et obligatoire, mais plus conjoncturelle. Si nous voulons exprimer le fait que la voiture contient éventuellement des bagages lors d'un départ en vacances, nous utiliserons l'agrégation et pas la composition. En effet, contrairement aux roues qui font partie intégrante de la voiture de même que le moteur et les sièges, on ne peut pas dire la même

**Figure 4–6**

Troisième bdd du bloc Voiture

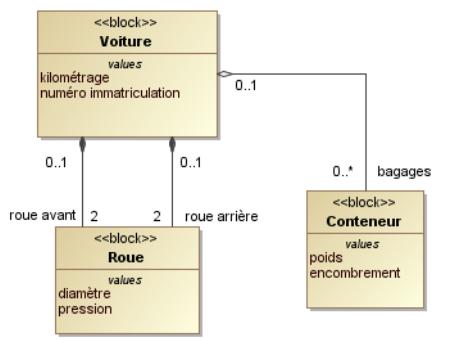


**Figure 4–7**  
bdd du bloc Voiture  
avec références



La représentation graphique alternative de la relation d'agrégation est donnée ci-après.

**Figure 4–8**  
bdd du bloc Voiture  
avec agrégation



# Association

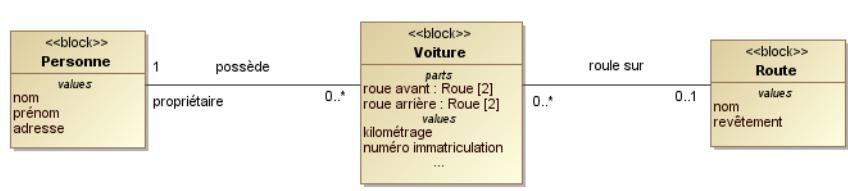
Un dernier type de relation entre blocs s'appelle l'association. L'association est une relation n'impliquant pas de contenance, comme la composition ou l'agrégation, mais une relation d'égal à égal. Par exemple, notre voiture roule habituellement sur une route (au sens large). Elle est la propriété d'une personne, qui peut en posséder plusieurs et joue le rôle de propriétaire. Ces deux relations sont représentées par des associations en SysML (lignes simples).

## B.A.-BA Association

Une association représente une relation sémantique durable entre deux blocs.

Exemple : Une personne peut posséder des voitures. La relation possède est une association entre les blocs Personne et Voiture. Attention : même si le verbe qui nomme une association semble privilégier un sens de lecture, une association entre blocs est par défaut bidirectionnelle. Donc implicitement, l'exemple précédent inclut également le fait qu'une voiture est possédée par une personne. Les compositions et les agrégations sont des cas particuliers d'association.

**Figure 4–9**  
bdd du bloc Voiture  
avec associations



### B.A.-BA Multiplicités d'une Association

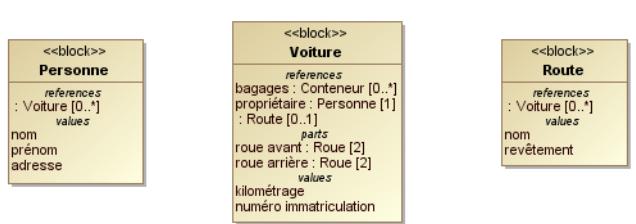
Aux deux extrémités d'une association, on doit faire figurer une indication de multiplicité. Elle spécifie sous la forme d'un intervalle le nombre d'instances qui peuvent participer à une relation avec une instance de l'autre bloc dans le cadre d'une association. Une instance est un exemplaire d'un certain bloc possédant une identité propre.

Exemple : une personne peut posséder plusieurs voitures (entre zéro et un nombre quelconque) ; une voiture est possédée par une seule personne.

Les associations donnent lieu à des références dans les deux blocs reliés (comme les agrégations). Ainsi la voiture a une référence supplémentaire appelée propriétaire, de type Personne, et une référence optionnelle anonyme de type Route. Les blocs Personne et Route ont pour leur part une référence multiple de type Voiture.

**Figure 4-10**

bdd avec compartiments  
références complétés



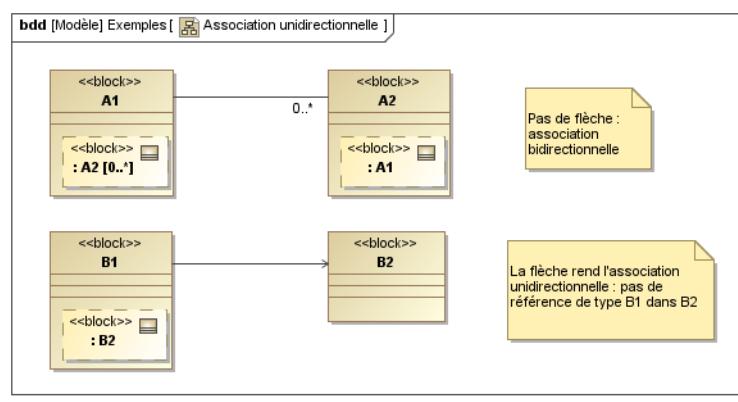
Si l'on souhaite exprimer le fait que dans notre contexte, la voiture doit avoir une référence vers la route empruntée, mais que la route par contre n'a pas besoin de connaître toutes les voitures qui l'empruntent à un moment donné, il faut rendre l'association entre ces deux blocs unidirectionnelle.

### ATTENTION Association unidirectionnelle

Pour l'instant, nous avons utilisé des associations et compositions bidirectionnelles, qui donnent des références ou des parties des deux côtés. Si l'on souhaite qu'il n'y ait qu'une référence (ou partie) que d'un seul côté, on peut rendre l'association unidirectionnelle en ajoutant une flèche pointant vers le type de la référence. Les références ou parties peuvent être ajoutées textuellement dans des compartiments spécifiques (comme précédemment), ou même dessinées graphiquement dans un compartiment dédié appelé structure (comme sur le diagramme suivant).

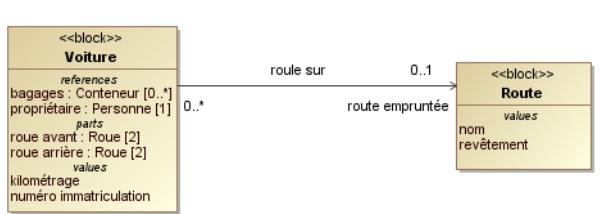
**Figure 4-11**

Structure des blocs suivant le type d'association



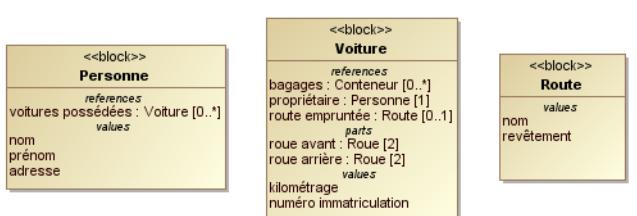
Une association unidirectionnelle possède une flèche pointant vers le bloc référencé (dans notre cas : Route). Il est également souhaitable de nommer l'extrémité du côté Route, afin de nommer la référence correspondante.

**Figure 4-12**  
bdd avec association  
unidirectionnelle



Si l'on ne montre pas l'association, mais les compartiments des blocs, on remarque qu'il y a bien une référence de type Route dans Voiture, mais pas de référence de type Voiture dans Route.

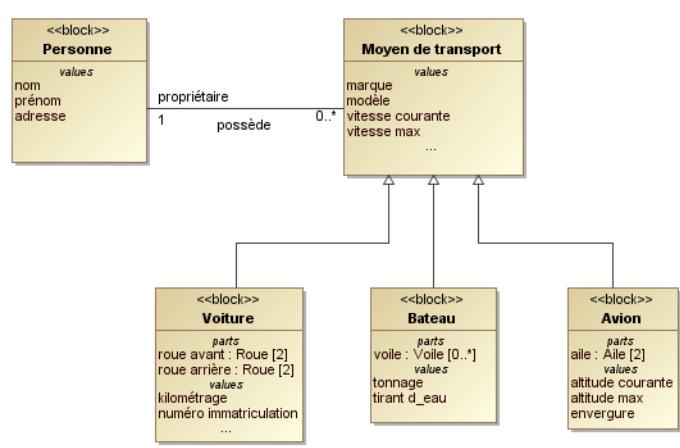
**Figure 4-13**  
bdd avec compartiments  
mis à jour suite à l'association  
unidirectionnelle



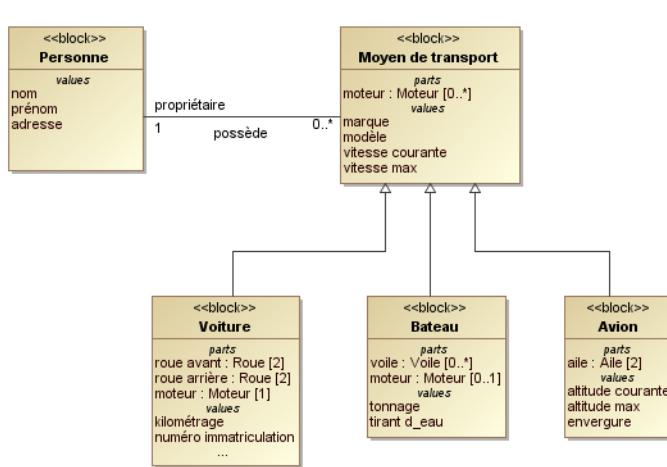
## Généralisation

Toutes les définitions qui apparaissent dans un bdd peuvent être organisées dans une hiérarchie de classification. Le but est souvent de factoriser des propriétés communes (valeurs, parties, etc.) à plusieurs blocs dans un bloc généralisé. Les blocs spécialisés

**Figure 4-14**  
bdd avec relation  
de généralisation



**Figure 4–15**  
bdd avec relation de généralisation et restrictions



Sur le diagramme précédent, nous avons ainsi ajouté une partie moteur de multiplicité variable au bloc généralisé Moyen de transport. Les blocs Voiture et Bateau redéfinissent la multiplicité de la partie moteur, alors que le bloc Avion en hérite simplement.

# Opération

Tout bloc possède également des propriétés comportementales, les principales étant appelées opérations. Une opération représente soit :

- une requête synchrone (l'émetteur est bloqué en attente d'une réponse), avec ses éventuels paramètres (ou arguments) en entrée, sortie, ou les deux ;
- une requête asynchrone, aussi appelée réception (l'émetteur n'est pas bloqué en attente d'une réponse). Chaque réception est associée à un signal qui définit un message avec ses éventuels paramètres. Des réceptions définies dans des blocs différents peuvent répondre au même type de signal, ce qui permet de réutiliser la définition de messages communs.

Les opérations sont montrées dans un compartiment supplémentaire avec leur signature (nom, paramètres et type de retour) :

```
nom_opération (liste de paramètres) : type_retour
```

La liste de paramètres est une liste d'éléments séparés par des virgules, chaque élément étant noté :

```
direction nom_paramètre : type_paramètre
```

la direction pouvant prendre une des valeurs suivantes : *in*, *out*, *inout*.

Les réceptions sont souvent montrées dans un compartiment à part. Les signaux peuvent à leur tour être définis comme des blocs avec un mot-clé « *signal* ».

Sur l'exemple de la voiture, nous allons ajouter quelques opérations comme la démarrer ou l'arrêter, changer une roue, ainsi que la vendre à quelqu'un. Nous allons également prendre en compte la réception de signaux GPS.

**Figure 4–16**  
bdd avec opérations  
et réceptions



## Étude de cas

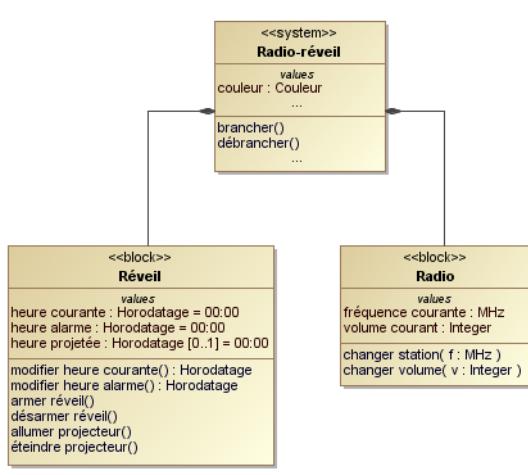
Reprendons notre étude de cas maintenant que nous avons vu les principaux concepts du bdd. Nous allons considérer que notre système (le radio-réveil) contient deux blocs principaux : un réveil et une radio. Ces blocs de premier niveau représentent des concepts logiques, des fonctionnalités, et pas encore des composants physiques. Nous verrons plus précisément comment projeter les composants physiques sur les composants logiques quand nous parlerons de l'important concept d'allocation au chapitre 10.

Si nous commençons à répartir les propriétés et les opérations, en fonction des responsabilités que nous décidons d'attribuer aux différents blocs, nous obtenons un premier bdd comme indiqué sur la figure suivante.

Ajoutons quelques éléments à ce premier diagramme :

- la radio reçoit les signaux HF des stations de radio ;

**Figure 4-17**  
Premier bdd du radio-réveil

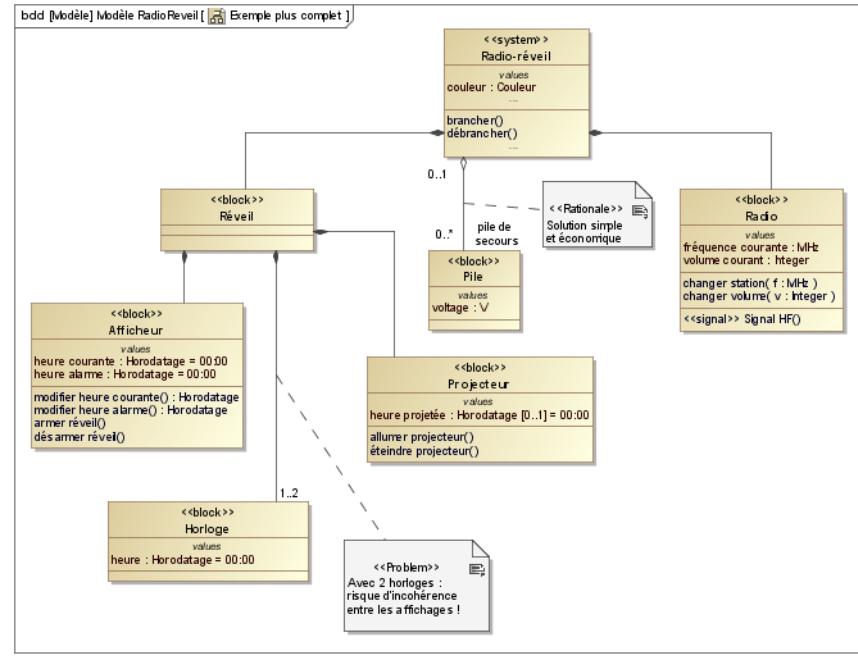


- les exigences précisent la nécessité d'un dispositif de sauvegarde pour conserver en mémoire les réglages en cas de coupure de courant. Pour cela, nous choisissons de pouvoir intégrer une ou plusieurs piles amovibles. La note avec le mot-clé « **rationale** » explique ce choix. Les piles sont récupérables en fin de vie du radio-réveil : nous utiliserons plutôt une relation d'agrégation ;
- le réveil contient un bloc afficheur, un bloc projecteur, et un bloc horloge. En fait, selon les radio-réveils, il y a un soit un unique bloc horloge garantissant l'affichage d'une heure cohérente sur le radio-réveil et au plafond, ou bien deux horloges différentes : une pour l'afficheur standard, une autre pour le projecteur. Le problème dans ce cas étant le risque d'incohérence entre l'heure projetée au plafond et l'heure du radio-réveil qui est la référence vis à vis de l'alarme, comme j'en ai fait l'expérience

rapidement avec mon premier radio-réveil à projecteur acheté pas cher... Nous allons expliquer ces deux possibilités grâce à une note avec le mot-clé « `problem` ».

**Figure 4–18**

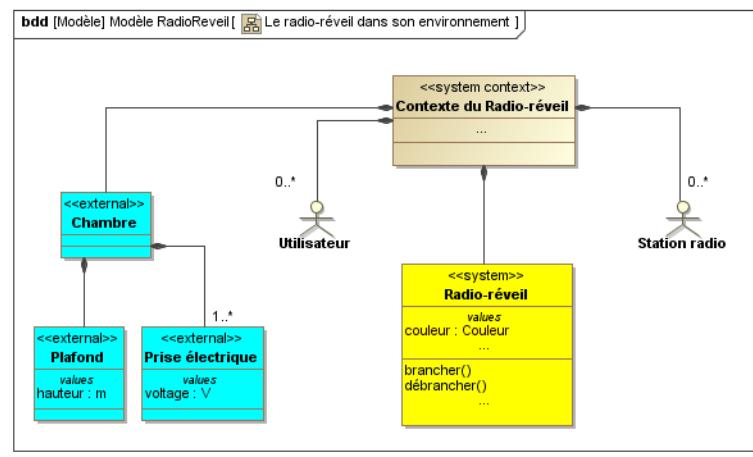
Exemple plus complet  
de bdd du radio-réveil



D'un point de vue méthodologique, il est souvent intéressant de remonter d'un cran et de modéliser le contexte du bloc principal (celui qui porte le mot-clé « `system` »). On peut ainsi représenter l'environnement du système, avec dans notre exemple, non seulement les

utilisateurs humains et les stations de radio, mais également l'environnement physique : la chambre avec son plafond sur lequel sera projetée l'heure la nuit, ainsi que ses prises électriques. Notez l'utilisation à titre d'exemple du stéréotype non standard « `external` » et d'une couleur de remplissage différente permettant de différencier certains éléments externes qui ne sont pas des acteurs à proprement parler (ils n'interviennent pas dans les cas d'utilisation).

**Figure 4–19**  
bdd du radio-réveil  
dans son environnement

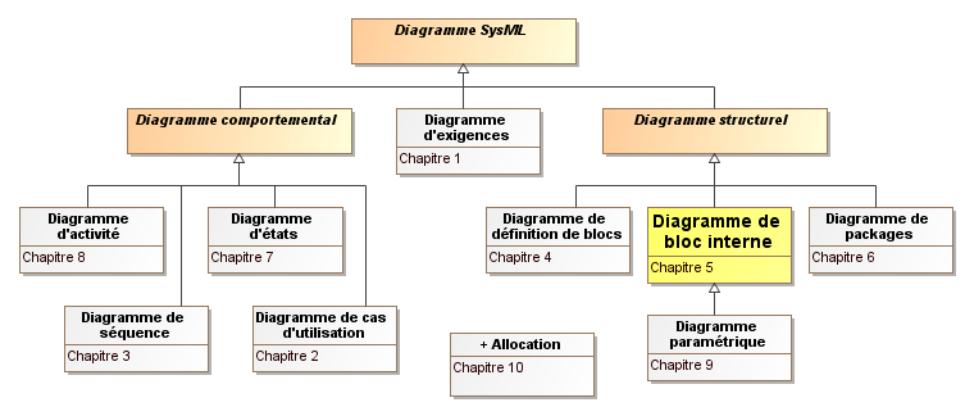


# 5

## Le diagramme de bloc interne

Ce chapitre présente le diagramme de bloc interne. Le diagramme de bloc interne (*internal block diagram* ou ibd) décrit la structure interne du système en termes de parties, ports et connecteurs.

- ▶ bloc et ibd
- ▶ partie
- ▶ port et connecteur



## Parties et connecteurs

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.

### B.A.-BA Cartouche

Le cartouche général du diagramme de bloc interne est de la forme :  
**ibd [bloc] nom du bloc [nom du diagramme]**

## Parties et références

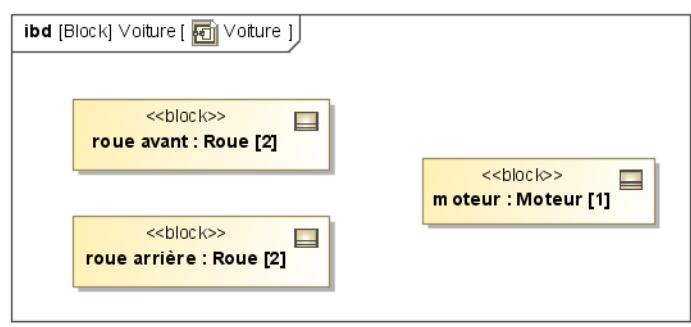
Dans l'exemple de la voiture, nous pouvons commencer par représenter les roues avant et arrière ainsi que le moteur. La relation de composition du bdd se traduit de la façon suivante dans l'ibd :

- 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. Le nom du bloc est identique à ce qui apparaîtrait dans le compartiment parts, soit : `nom_partie : nom_bloc [multiplicité]`.

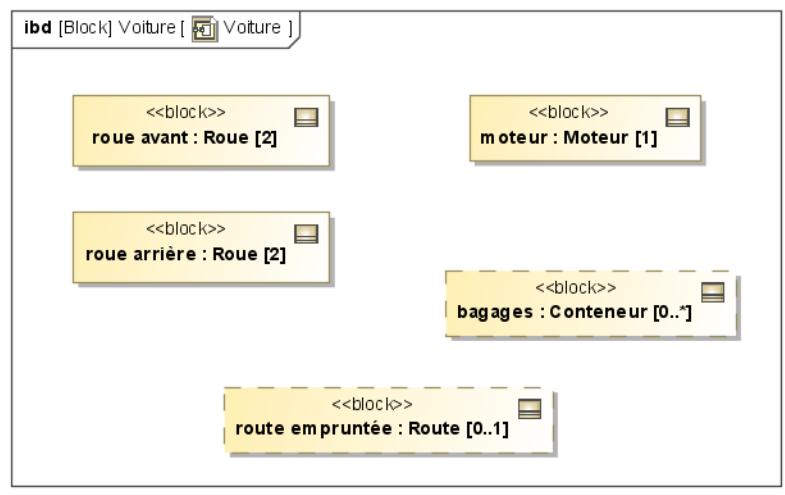
La multiplicité peut également être représentée dans le coin supérieur droit du rectangle. S'il n'y a pas d'indication de multiplicité, elle est supposée valoir exactement 1.

**Figure 5–1**

Premier ibd du bloc Voiture

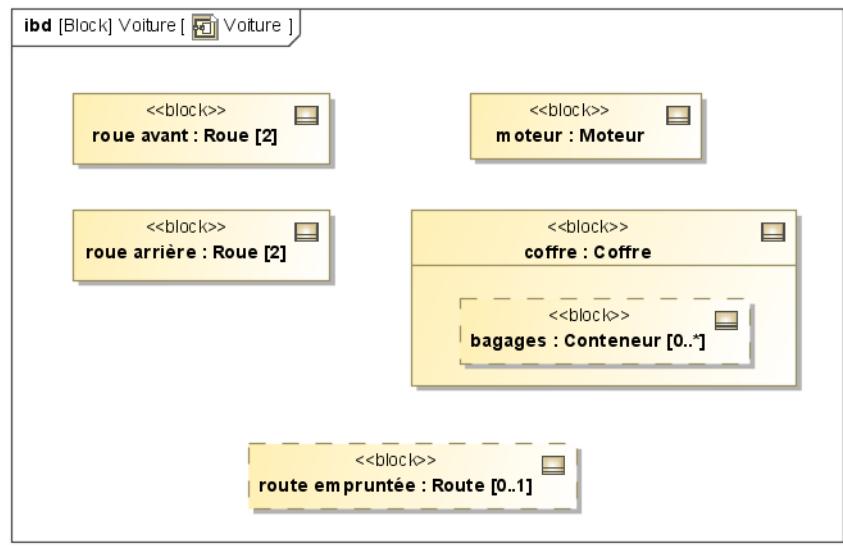


**Figure 5–2**  
ibd du bloc Voiture  
avec références



Il est important de noter qu'on peut représenter plusieurs niveaux de décomposition ou de référence sur un même ibd. Dans notre exemple, si nous voulons exprimer le fait que les bagages sont dans le coffre de la voiture, il suffit d'ajouter une partie coffre à la voiture, et de déplacer la référence bagages à l'intérieur de cette nouvelle partie.

**Figure 5–3**  
ibd du bloc Voiture  
à plusieurs niveaux



Le diagramme de bloc interne est surtout utile pour montrer les connexions contextuelles entre les parties d'un bloc, ce qui ne peut pas être représenté sur un bdd.

## 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 (comme décrit au paragraphe suivant). L'extrémité d'un connecteur peut pos-

séder une multiplicité qui décrit le nombre d'instances qui peuvent être connectées par des liens décrits par le connecteur.

#### ATTENTION Vocabulaire SysML

Soyons précis :

- une association (agrégation, ou composition) relie des blocs ;
- un connecteur relie des parties ;
- un lien relie des instances.

Un connecteur peut être typé par une association. Il peut ainsi posséder une flèche unidirectionnelle, si l'association qui le type la possède. Son nom complet est de la forme :

`nom_connecteur : nom_association`

Une partie peut être connectée à plusieurs autres parties, mais il faut représenter un connecteur séparé pour chaque liaison.

Dans notre exemple, à l'intérieur du bloc Voiture, le moteur est relié aux deux roues avant, mais pas aux roues arrière. Les quatre roues sont (normalement...) reliées à la route.

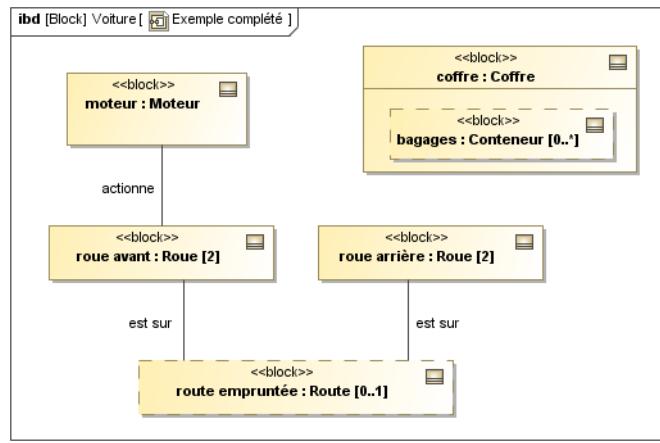
Sur le diagramme suivant est représenté l'ibd d'un bateau à moteur. Notez que le moteur, du même type que dans le cas de la voiture, est cette fois-ci connecté à des hélices, parce que nous sommes dans le contexte d'un bateau.

La multiplicité est ici représentée sur le connecteur, et pas dans la partie, pour montrer l'autre possibilité.

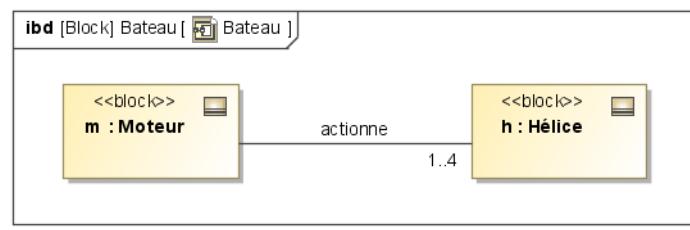
Pour bien comprendre l'intérêt de l'ibd par rapport au bdd, essayons de représenter les associations entre le bloc Moteur et les blocs Roue et Hélice sur un premier bdd.

Quels sont les problèmes de ce premier bdd ?

**Figure 5–4**  
ibd du bloc Voiture avec connecteurs

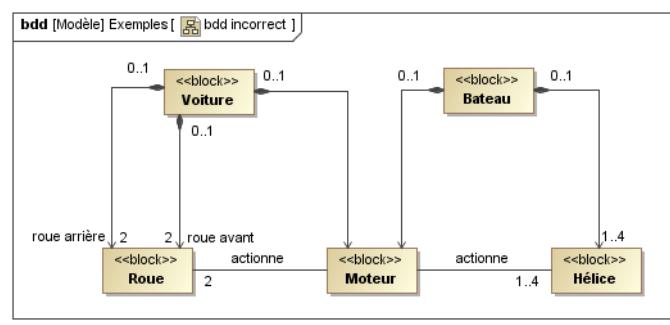


**Figure 5–5**  
ibd du bloc Bateau



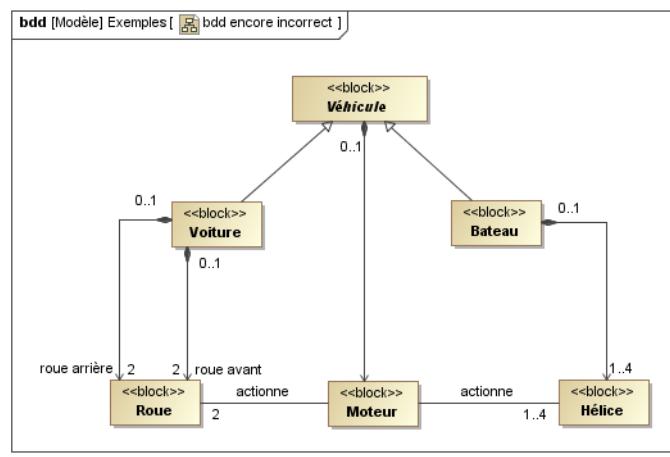
On pourrait déjà argumenter sur le fait qu'une instance de Moteur peut appartenir à 0 ou 1 voiture et 0 ou 1 bateau. Ce qui voudrait dire qu'une même instance de Moteur pourrait être à la fois dans une voiture et un bateau. La sémantique de la composition en SysML

**Figure 5–6**  
bdd incorrect (premier essai)



vise à interdire cette possibilité, mais il serait encore plus clair de l'exprimer sur le diagramme en introduisant par exemple un bloc abstrait généralisé Véhicule.

**Figure 5–7**  
bdd incorrect (deuxième essai)



**B.A.-BA Bloc abstrait**

Un bloc est dit abstrait si sa définition ne permet pas de l'instancier.

On se sert souvent de blocs abstraits dans les arbres de généralisation pour factoriser des propriétés structurelles ou comportementales communes à d'autres blocs concrets (instanciables).

Un bloc abstrait est représenté en italique.

Mais ce n'était malheureusement pas le principal problème de ces deux bases de données ! En effet, les associations dans un bdd sont définies une fois pour toutes, et doivent être vérifiées dans tous les contextes, pour toutes les instances. Dans le cas des deux associations Actionne, cela signifie donc que :

- le moteur d'une certaine voiture peut actionner les hélices d'un bateau ;
- le moteur d'un bateau peut actionner les roues d'une voiture ;
- le moteur d'une voiture peut même actionner les roues d'une autre instance de voiture...

Nous voyons donc clairement que le modèle proposé n'est pas du tout satisfaisant. C'est justement pour cela que SysML propose le diagramme de bloc interne : pour représenter des liaisons contextuelles entre des parties à l'intérieur d'un certain bloc englobant.

# Ports et interfaces

## Types de ports

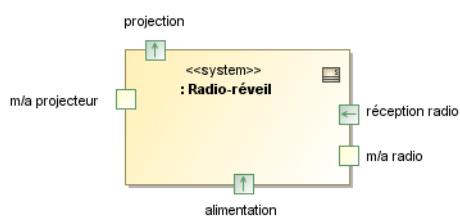
Le diagramme de bloc interne permet également de décrire la logique de connexion, de services et de flots entre blocs grâce au concept de « port ». Les ports définissent les points d'interaction offerts (*provided*) et requis (*required*) entre les blocs. Un bloc peut avoir plusieurs ports qui spécifient des points d'interaction différents. Les ports peuvent être de deux natures :

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

La distinction entre ces deux types de ports est souvent d'ordre méthodologique. Les *flow ports* sont bien adaptés pour représenter des flux continus d'entités physiques, alors que les ports standards sont bien adaptés à l'invocation de services, typiquement entre composants logiciels. Une combinaison des deux est souvent utile, mais les ports standards ne peuvent être connectés directement aux *flow ports* et réciproquement.

Dans notre exemple de radio-réveil, les boutons de marche-arrêt du projecteur et de la radio sont typiquement des ports standards. L'entrée d'énergie électrique comme les ondes radio, la projection de lumière ou la diffusion de son sont typiquement des *flow ports*.

**Figure 5–8**  
Exemples de ports  
du radio-réveil



## Flow ports

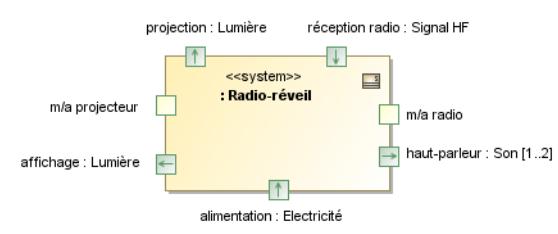
Les ports de type « flux » sont soit atomiques (un seul flux), soit composites (agrégation de flux de natures différentes). Dans l'exemple précédent, les *flow ports* Projection, Réception radio et Alimentation sont tous atomiques. Cela signifie qu'ils ne spécifient qu'un seul type de flux en entrée ou en sortie (ou les deux), la direction étant simplement indiquée par une flèche à l'intérieur du carré représentant le port. Pour leur part, les ports standards sont simplement représentés par des carrés.

Les *flow ports* peuvent être typés par un bloc, un *value type* ou un signal représentant le type d'élément pouvant circuler en entrée ou en sortie du port. Nous allons utiliser un bloc Lumière pour typer le *flow port* de projection, un bloc Électricité pour l'alimentation, et le signal Signal HF (créé au chapitre précédent) pour la réception radio. Plusieurs ports peuvent avoir le même type, comme la projection et l'affichage de l'heure par exemple.

On peut également associer une multiplicité à un port. Il est ainsi possible d'exprimer de façon concise qu'un ordinateur possède plusieurs ports USB identiques, ou qu'un radio-réveil peut avoir un ou deux haut-parleurs. La notation complète d'un *flow port* est donc de la forme : `nom_port : nom_item [multiplicité]`.

**Figure 5–9**

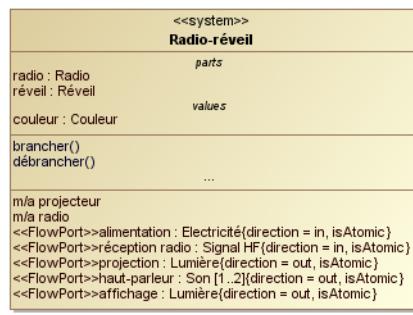
Exemples de flow ports types du radio-réveil



Les ports faisant partie de la définition des blocs, il est tout à fait possible de les faire apparaître dès le bdd. Dans ce cas, ils peuvent être représentés soit graphiquement comme précédemment, soit sous la forme d'un compartiment supplémentaire.

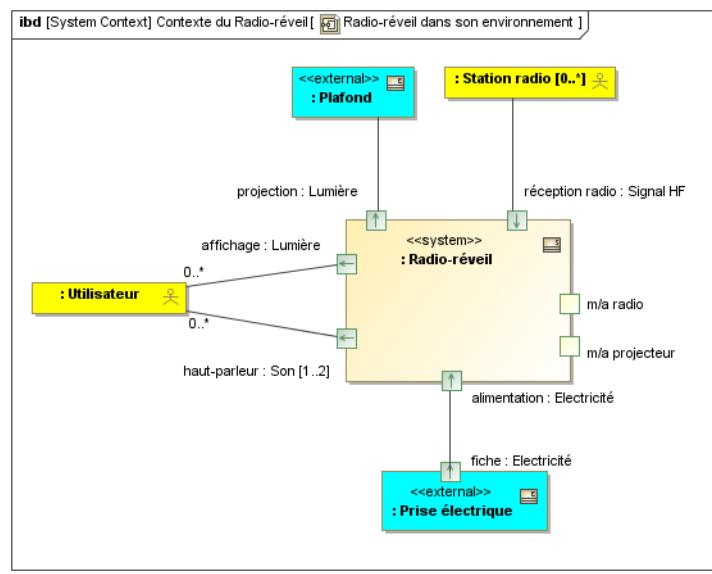
**Figure 5–10**

Flow ports représentés dans un bdd (compartiment)



Mais l'intérêt principal de l'ibd consiste là encore à pouvoir décrire les connexions entre les ports de différents blocs au moyen de connecteurs, alors que le bdd ne permet que de définir les ports sans les connecter. Nous allons ainsi pouvoir connecter la projection au plafond de la chambre, la réception radio aux stations, etc.

**Figure 5-11**  
Exemples de flow ports connectés du radio-réveil



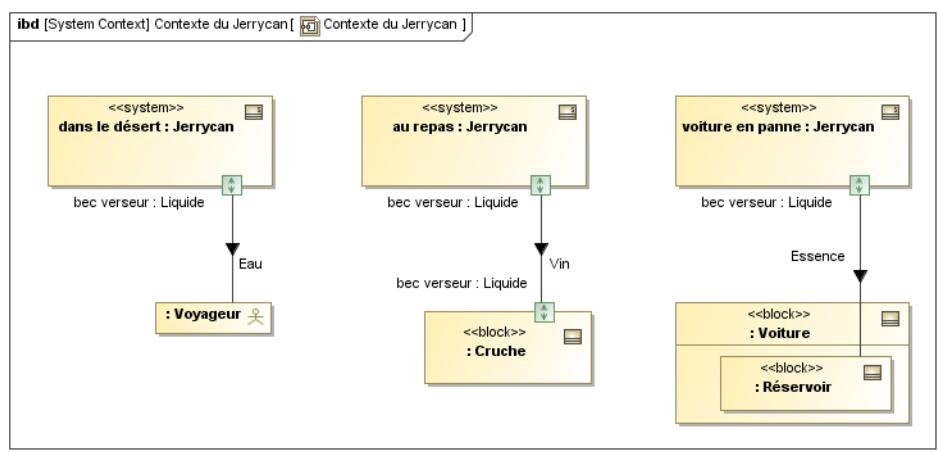
## Item flow

Les éléments de flot (*item flows*) permettent de décrire ce qui circule réellement sur les connecteurs, alors que les *flow ports* définissent ce qui peut circuler. La distinction n'est pas toujours utile, mais peut se révéler néanmoins très pratique dans certains cas. Prenons l'exemple d'un réservoir de liquide. Il a typiquement deux ports atomiques typés Liquide, l'un en entrée, l'autre en sortie, voire même un seul port d'entrée-sortie (un jerrycan par exemple).

**Figure 5–12**  
Définition d'un jerrycan  
tous usages (bdd)



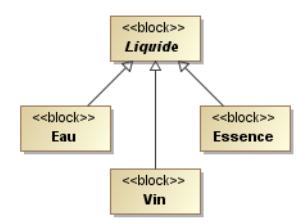
Des instances de Jerrycan utilisées dans des contextes différents acceptent différents types de flux, du moment que ceux-ci sont des spécialisations de Liquide. Dans le désert, dans une fête, ou en cas de panne sèche, ce qui circulera réellement pourra être représenté précisément grâce à la notation SysML de l'item *flow* (flèche noire nommée sur le connecteur), comme indiqué sur le diagramme suivant.



**Figure 5–13** Trois utilisations d'un jerrycan avec des flux différents

Tout ceci n'est bien sûr possible que parce que Eau, Vin et Essence sont des spécialisations de Liquide, comme déclaré dans un bdd.

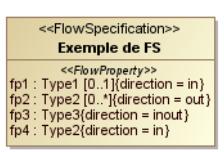
**Figure 5–14**  
Trois spécialisations  
du bloc Liquide (bdd)



## Flow specification

Quand un point d'interaction a une interface complexe avec plusieurs flux, le *flow port* correspondant doit être modélisé comme un *flow port* composite (ou non atomique). Dans ce cas, le port doit être typé par une spécification de flux (flow specification). Cette spécification de flux doit être définie dans un bdd. Elle inclut plusieurs propriétés de flux, chacune ayant un nom, un type et une direction.

**Figure 5–15**  
Notation de la Flow  
Specification (bdd)

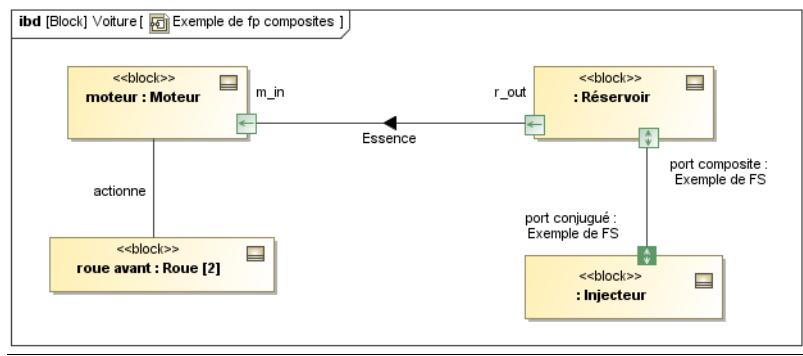


Un *flow port* composite est indiqué graphiquement par deux crochets se faisant face (< >) dessinés à l'intérieur du symbole du port. Quand deux parties interagissent, elles échan-

## Interface

Pour décrire un comportement basé sur l'invocation de services, le port standard est tout à fait adapté. Mais au lieu d'assigner directement des opérations aux ports, il est plus intéressant de les regrouper en ensembles cohérents appelés interfaces.

**Figure 5–16**  
Notation des flow ports  
composites et conjugués



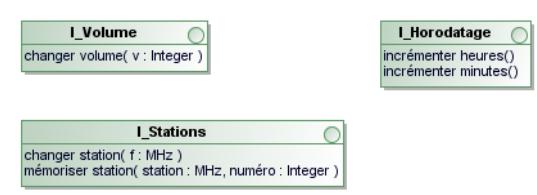
### B.A.BA Interface

Une interface est un ensemble d'opérations abstraites (sans algorithmes) constituant une sorte de contrat qui devra être réalisé par un ou plusieurs blocs. Graphiquement, une interface est soit représentée comme un bloc avec un mot-clé « `interface` » ou le symbole d'un cercle, soit directement comme un cercle dans la notation condensée.

Pour le radio-réveil, nous pouvons par exemple commencer à définir (dans un bdd) des interfaces pour la gestion du volume de la radio, le réglage et la mémorisation des stations, la mise à jour des heures et des minutes, etc.

**Figure 5-17**

Exemples d'interfaces pour le radio-réveil (bdd)



Une interface fournie (*provided interface*) sur un port spécifie les opérations que le bloc fournit. Une interface requise (*required interface*) spécifie les opérations dont le bloc a besoin pour réaliser son comportement. En général, un autre bloc les lui fournira au moyen d'une interface fournie.

Un même bloc peut fournir et/ou requérir plusieurs interfaces. Dans l'exemple ci-après, les deux blocs B1 et B2 fournissent l'interface I2, par contre B2 utilise l'interface I1 qui est fournie par B1, et l'interface I3. Le bloc B1 fournit les deux interfaces I1 et I2 : il possède donc au moins la réunion des opérations des deux interfaces.

### B.A.BA Réalisation et utilisation

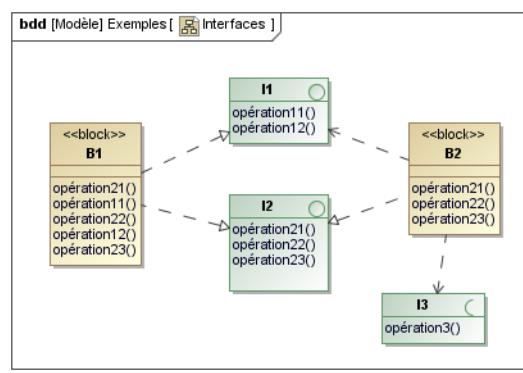
La relation de réalisation entre un bloc et une interface se dessine comme une généralisation en pointillés. Si l'interface est représentée par un simple cercle, la réalisation devient un simple trait.

La relation d'utilisation entre un bloc et une interface se dessine comme une flèche évidée en pointillés. Dans la notation graphique condensée, l'utilisation devient un simple trait et l'interface est représentée par un demi-cercle.

Certains outils (comme MagicDraw) mélagent les deux notations.

**Figure 5-18**

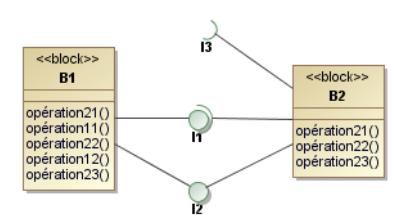
Exemple de réalisation et d'utilisation d'interfaces



La forme graphique condensée est donnée à la figure suivante. L'inconvénient de cette notation est qu'on ne peut pas savoir quelles sont les opérations définies dans chaque interface.

**Figure 5–19**

Notation graphique condensée des interfaces (bdd)



## Port standard

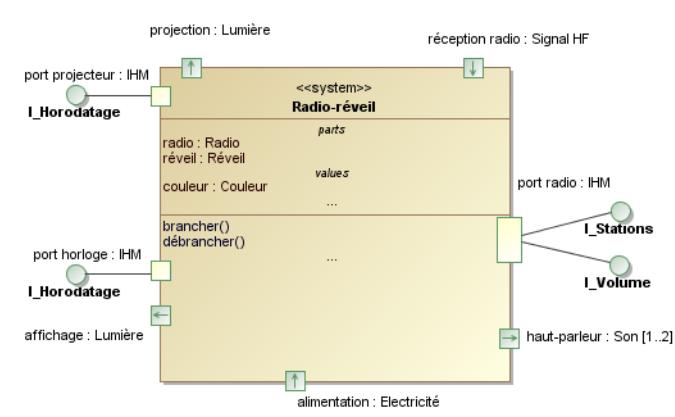
Plutôt que de relier directement les interfaces aux blocs par des relations de réalisation ou d'utilisation, on peut préciser à quel port standard est rattachée chaque interface. Un même port peut posséder plusieurs interfaces requises ou fournies. Plusieurs ports peuvent être reliés aux mêmes interfaces.

Sur notre exemple, nous avons créé un port pour la radio permettant à la fois de régler le volume et les stations. Il s'agit d'une abstraction de l'interface homme-machine (IHM) de la radio. Ce port fournit donc deux interfaces. Au contraire, si nous imaginons le cas où le projecteur possède une horloge différente de celle du réveil (variante des radio-réveils à bas prix...), les deux ports de réglage de l'horloge du radio-réveil et du projecteur réalisent la même interface de modification des heures et minutes ([I\\_Horodatage](#)). Mais malheureusement, rien ne garantit que les implémentations fournissent la même valeur !

Tous ces cas sont illustrés par la figure suivante, qui utilise classiquement la notation condensée des interfaces. Attention, pour simplifier, certains ports qui apparaissaient sur des diagrammes précédents n'ont pas été montrés.

**Figure 5–20**

Exemple de ports standards et d'interfaces (bdd)



## Étude de cas

Reprenons notre étude de cas maintenant que nous avons vu les principaux concepts de l'ibd, et affiné certains concepts du bdd. Si nous traduisions les relations de composition du bdd de la fin du chapitre 4 dans un ibd, nous obtenons le diagramme qui suit. Notez que l'outil MagicDraw a fait suivre automatiquement les ports externes du radio-réveil sur le contour du diagramme. La pile de secours apparaît en pointillés (référence).

Prenons tout d'abord la décision de modéliser prioritairement par la suite un radio-réveil ayant une horloge unique. Nous allons ainsi enlever l'indication de multiplicité [1..2] sur la partie `h : Horloge`. En fait, nous pourrions tout à fait envisager de mener en parallèle deux modèles alternatifs pour comparer les deux solutions de conception. Les diagrammes dynamiques montreraient certainement des différences également.

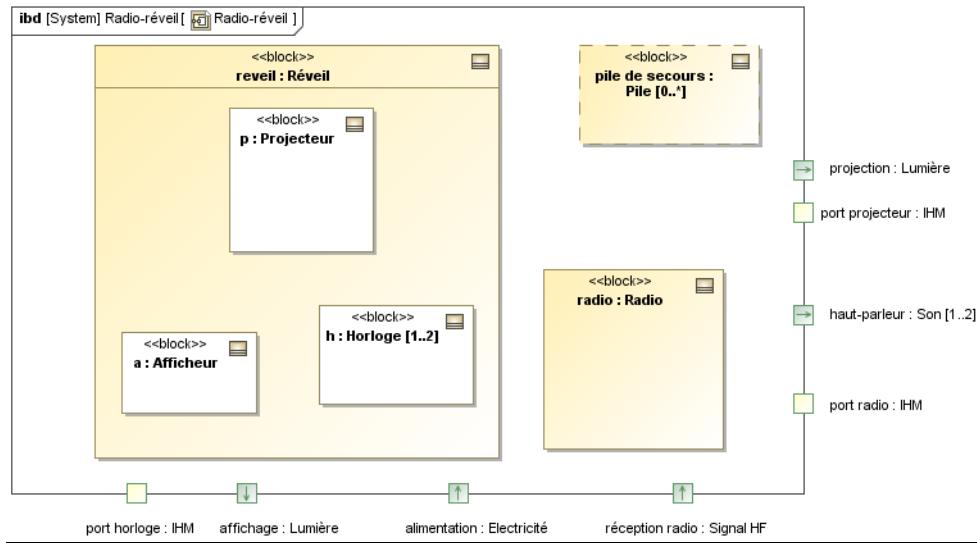


Figure 5–21 Première version de l'ibd du radio-réveil

Ensuite, nous allons commencer à décrire le fait que l'horloge du réveil doit avoir une connexion avec l'afficheur, le projecteur, mais aussi avec la radio (pour l'activer à l'heure d'alarme). La pile de secours, pour sa part, n'est connectée qu'à l'horloge. Pour l'instant, nous n'ajouterons pas de ports aux parties.

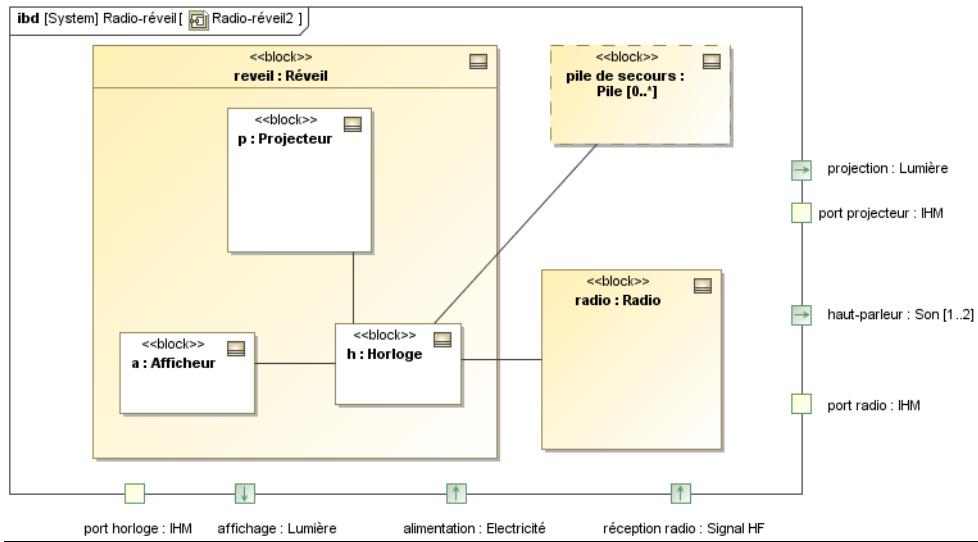


Figure 5–22 Deuxième version de l’ibd du radio-réveil

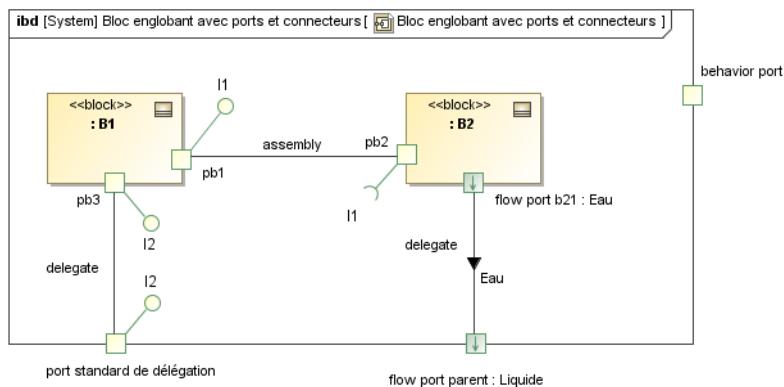
Il nous reste à connecter l’extérieur (radio-réveil boîte noire) à l’intérieur (parties). Là encore, nous pourrions commencer sans ajouter de port sur les parties. Mais nous allons le faire pour montrer le cas fréquent de conception descendante (*top-down*) où les ports et interfaces externes doivent être délégués aux parties.

## ATTENTION Behavior port vs delegation port

Il y a deux cas à considérer lorsqu'un bloc gère les interactions survenant sur ses ports. Soit il les traite directement lui-même, soit il délègue le traitement à ses parties.

- Si le bloc traite directement les interactions, le port est appelé port de comportement (behavior port). Les éléments envoyés ou reçus doivent alors être gérés par une propriété dynamique, par exemple une opération, une réception, ou même une machine à états (voir le chapitre 7). Ce port n'est donc pas relié aux parties.
- Dans l'autre cas, si le bloc délègue le traitement à ses parties, le port est appelé port de délégation (delegation port). Le connecteur qui relie des ports de blocs de niveaux différents est alors appelé connecteur de délégation, alors qu'un connecteur qui relie des ports au même niveau est appelé connecteur d'assemblage (assembly connector).

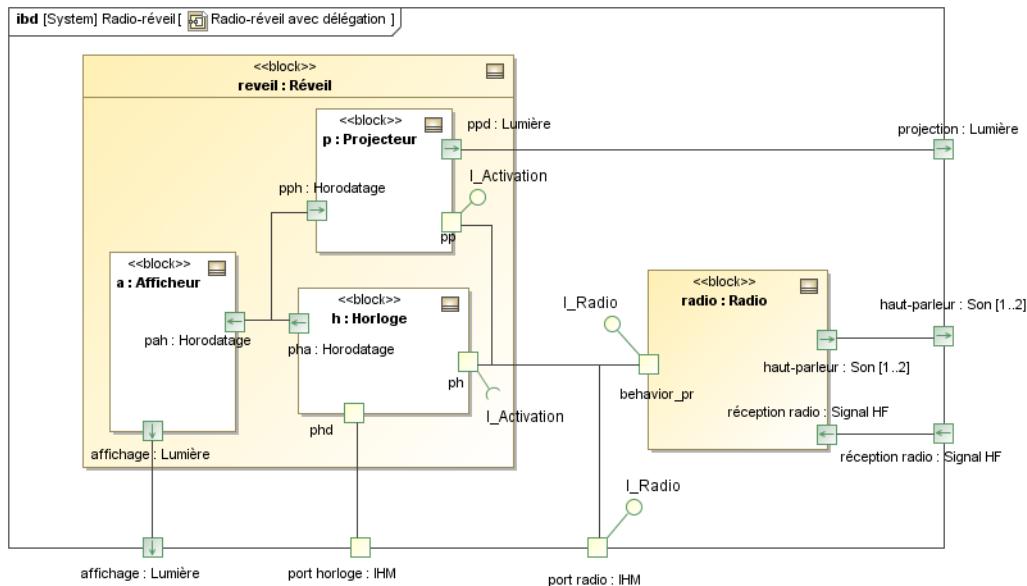
Dans tous les cas, les connecteurs de délégation ou d'assemblage doivent vérifier des contraintes de compatibilité. Ces contraintes sont légèrement différentes dans le cas des ports standard. En effet, dans le cas de la délégation, les ports doivent fournir ou utiliser les mêmes interfaces, alors que dans le cas de l'assemblage, l'un fournit ce que l'autre requiert. Les règles de compatibilité en cas de types spécialisés sont les mêmes que pour les flow ports (comme sur l'exemple avec les types Eau et Liquide).



**Figure 5-23** Exemple de connecteurs sur un ibd

Sur l'ibd précédent, les connecteurs allant des ports du bloc englobant aux ports des parties sont des connecteurs de délégation. On voit que sur les ports standards reliés, la même interface I2 est fournie. Alors que pour les ports des parties reliés par un connecteur d'assemblage, l'un (pb1) fournit I1 alors que l'autre (pb2) l'utilise. Notez le port de comportement appelé behavior port du bloc englobant (en haut à droite) : il n'est pas relié à un port d'une partie.

Dans notre étude de cas, nous avons choisi de représenter seulement un sous-ensemble des ports et des connecteurs pour ne pas surcharger le diagramme, mais ce dernier illustre néanmoins les exemples précédents.



**Figure 5-24** ibd partiel du radio-réveil

On notera le rôle central de l'horloge qui active la radio et le projecteur (à l'heure d'alarme), et qui fournit aussi l'horodatage à l'afficheur et au projecteur. L'activation est représentée par une interface requise par l'horloge (message envoyé) et fournie (message

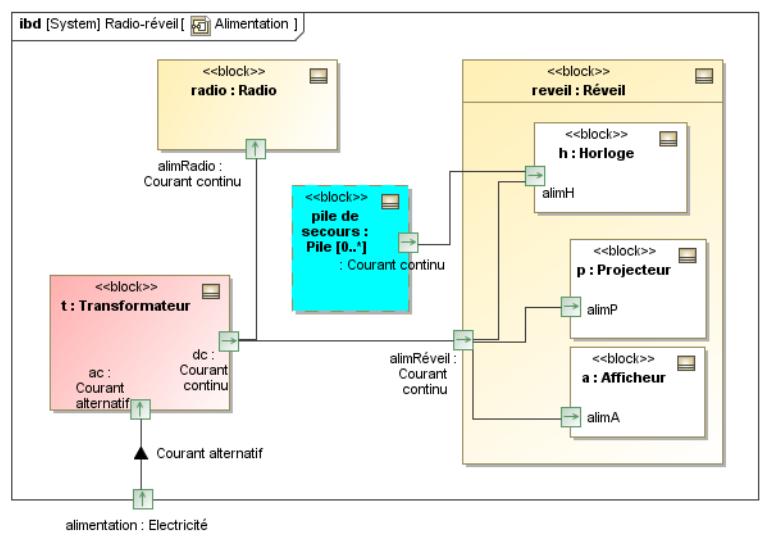
reçu) par le projecteur. La Radio fournit une interface `I_Radio` qui doit être également compatible avec l'activation, mais qui doit aussi inclure les aspects IHM, par délégation du port externe `port Radio : IHM`. L'interface `I_Radio` est donc un sur-ensemble de l'interface `I_Activation`, en termes d'opérations et donc de messages.

On voit aussi un exemple de port de comportement (*behavior port*) sur la partie radio qui va traiter les messages envoyés par l'horloge et l'utilisateur au travers d'une machine à états (celle du chapitre 7). L'horodatage est un flux continu représenté par des connecteurs d'assemblage entre *flow ports*.

Remarquez encore que les *flow ports* de délégation sont orientés dans le même sens, alors que les *flow ports* d'assemblage sont orientés dans le sens contraire (conjugués). Nous avons préféré ne pas créer de ports sur la partie de niveau intermédiaire appelée `réveil : Réveil` pour ne pas surcharger ce diagramme.

Pour donner une autre illustration de la puissance de SysML, et la possibilité de réaliser plusieurs diagrammes complémentaires, à la manière de calques superposables, nous avons dessiné un autre ibd en nous focalisant uniquement sur l'alimentation électrique du radio-réveil. Du coup, nous avons introduit un nouveau bloc `t : Transformateur`, dont le rôle est de transformer le courant alternatif fourni par la prise de courant en courant continu. Le transformateur alimente ainsi toutes les parties du radio-réveil, et nous avons pris la peine cette fois-ci de traverser « proprement » la partie réveil en ajoutant un port intermédiaire. Du coup, il apparaît clairement que la pile de secours n'alimente que la partie horloge, conformément aux exigences initiales.

**Figure 5–25**  
ibd du radio-réveil concernant  
l'alimentation électrique

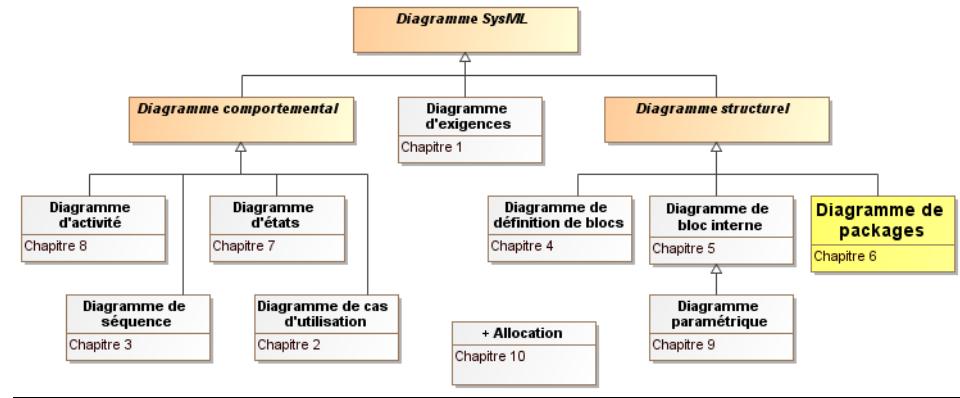


# 6

## Le diagramme de packages

Ce chapitre présente le diagramme de packages. Le diagramme de packages montre l'organisation logique du modèle et les éventuelles relations entre les packages.

- ▶ package, paquetage
- ▶ contenance
- ▶ dépendance
- ▶ modèle
- ▶ vue (*view*), point de vue (*viewpoint*)



## Package

En SysML, tout élément de modèle est contenu dans un seul conteneur. Quand un conteneur est détruit ou copié, ses éléments contenus sont également détruits ou copiés. Certains éléments contenus sont à leur tour des conteneurs, ce qui conduit classiquement à une hiérarchie de contenance d'éléments de modèle.

Les packages sont un exemple important de conteneur, les blocs en sont un autre (comme nous l'avons vu au chapitre précédent pour les valeurs, les parties, les opérations, etc.).

Les différents packages d'un modèle ainsi que leurs relations peuvent être montrés dans un diagramme de packages (*package diagram*).

## B.A.-BA Package

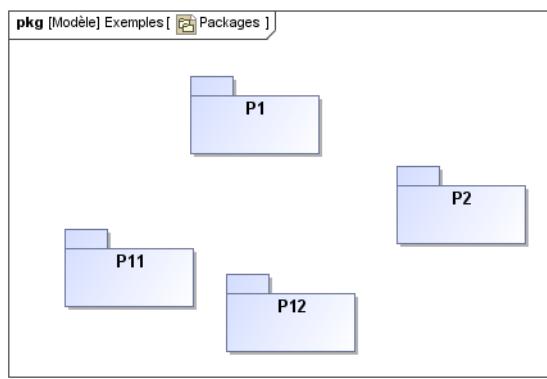
Mécanisme général de regroupement d'éléments tels que blocs, interfaces, mais aussi acteurs, cas d'utilisation, etc. Les packages peuvent être imbriqués dans d'autres packages.  
Un package constitue un espace de noms (namespace) pour les éléments qu'il contient.  
Certains auteurs utilisent la version française du terme : paquetage, mais nous préférerons garder package...

## B.A.-BA Cartouche

Le cartouche général du diagramme de package est de la forme :

**pkg [type de package] nom du package [nom du diagramme]**

Le type de package peut être : modèle, package ou vue (voir paragraphes suivants).



**Figure 6-1** Diagramme de packages simple

# Contenance et dépendance

Il y a deux types de relations possibles entre packages : la contenance et la dépendance.

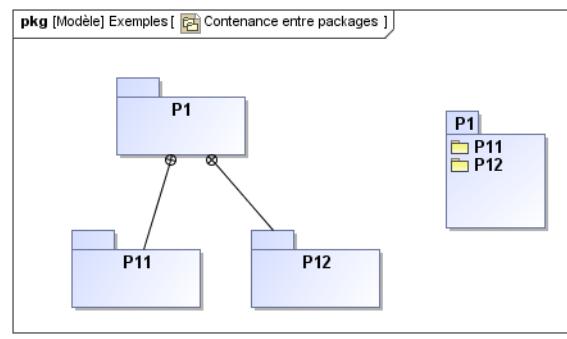
## Contenance

La relation de contenance peut être montrée de deux façons :

- un trait avec une croix entourée du côté du conteneur ;
- les packages contenus graphiquement à l'intérieur du package englobant.

La première solution permet de représenter des hiérarchies complètes graphiquement en gardant la même taille de police de caractères.

**Figure 6–2**  
Contenance entre packages

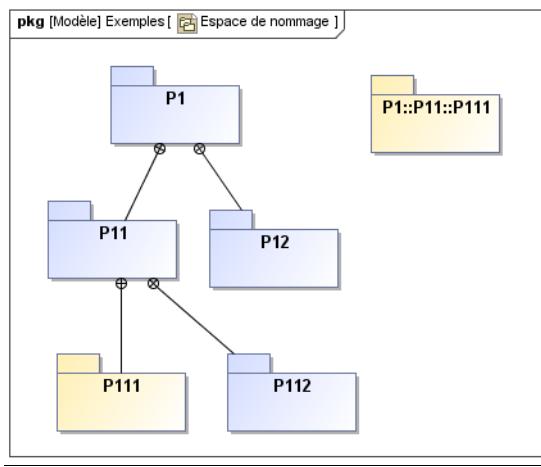


Nous avons dit qu'un package est également un espace de noms pour les éléments contenus. Pour nommer un élément, il est ainsi possible d'utiliser un nom qualifié (*qualified*

## Dépendance

Selon l'organisation du modèle et les choix de structuration, les éléments de différents packages sont souvent reliés entre eux. Nous avons vu par exemple que les blocs peuvent être reliés par des associations, des compositions, des généralisations, etc. Ces relations entre éléments induisent des relations de dépendance entre les packages englobants :

**Figure 6–3**  
Contenance entre packages  
et nom qualifié



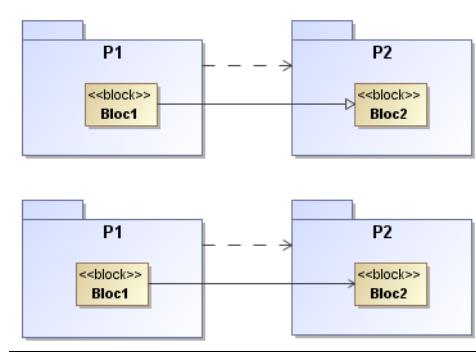
Dans l'exemple suivant, le package P111 est contenu dans le package P11 qui est lui-même contenu dans le package P1. Son nom qualifié est donc P1::P11::P111.

- une relation de généralisation entre les blocs Bloc1 et Bloc2 appartenant respectivement aux packages P1 et P2 se traduit ainsi par une relation de dépendance de P1 vers P2 ;
- une association directionnelle entre les blocs Bloc1 et Bloc2 appartenant respectivement aux packages P1 et P2 se traduit également par une relation de dépendance de P1 vers P2.

Il est à noter que la flèche pointillée de dépendance entre les packages englobants est toujours dans le même sens que les flèches de généralisation ou d'association orientée.

**Figure 6–4**

Relations entre blocs  
et relations de dépendance  
entre packages

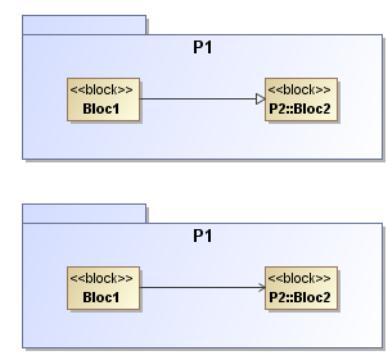


**ATTENTION Association bidirectionnelle**

Lorsqu'une association ne possède pas de flèche, nous avons expliqué que cela signifie qu'elle est navigable dans les deux sens. Les blocs aux extrémités possèdent alors au minimum une référence chacun sur l'autre. Cela induit une dépendance mutuelle au niveau des packages englobant, ce qui est rarement souhaitable.

Si l'on veut représenter les relations entre les blocs dans le contexte du package P1, on a alors intérêt à utiliser le nom qualifié du bloc Bloc2, comme illustré sur la figure suivante.

**Figure 6–5**  
Relations entre blocs  
et espaces de noms



**ATTENTION Forte cohésion et faible couplage**

La structuration d'un modèle est une activité délicate. Elle doit s'appuyer sur deux principes fondamentaux : forte cohésion interne et faible couplage externe. Le second principe consiste à minimiser les dépendances entre packages, en particulier en évitant les dépendances mutuelles et les dépendances cycliques.

# Model, view et viewpoint

Un modèle est un type particulier de package. Il constitue une sorte de racine pour tous les packages d'une hiérarchie. Le choix de ce niveau est un choix méthodologique ou organisationnel.

Un modèle (*model*) est représenté par le symbole du package agrémenté d'un triangle en haut à droite.

Figure 6–6

Représentation graphique d'un modèle

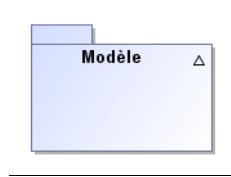
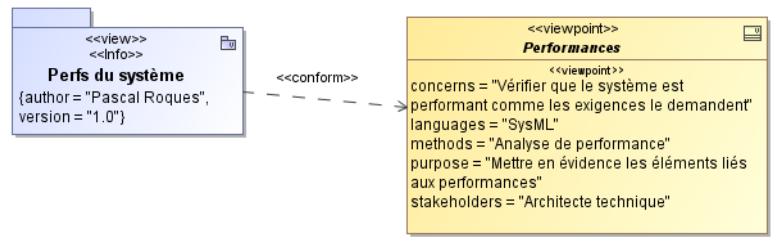


Figure 6–7

Représentation graphique d'une vue et d'un point de vue



Une vue (*view*) est une sorte de package utilisée pour montrer une perspective particulière sur un modèle, comme la sécurité, ou les performances.

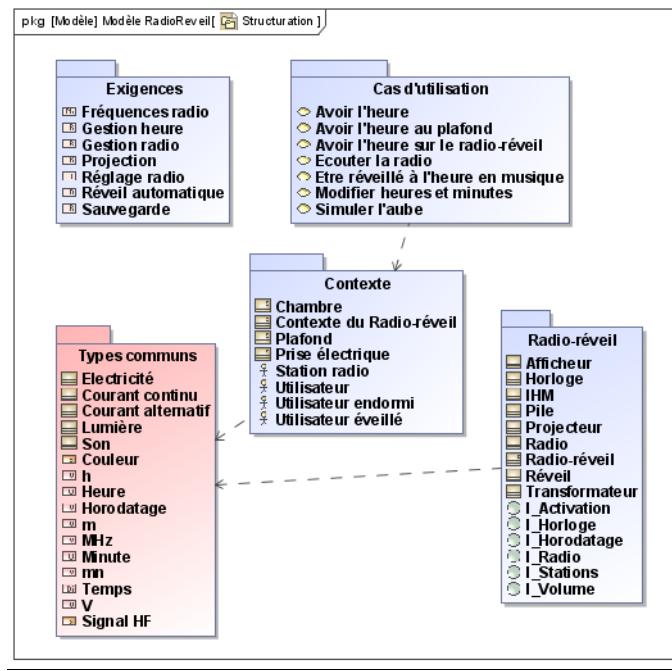
## Étude de cas

Reprendons notre étude de cas et structurons les blocs déjà créés en packages.

Regroupons les exigences entre elles, les cas d'utilisation, les blocs extérieurs au radio-réveil, les types de base, et les blocs internes au radio-réveil dans des packages différents. Ensuite, nous pouvons représenter tous ces packages ainsi que leur contenu sur un diagramme de packages synthétique. Si nous ajoutons des dépendances vers le package regroupant les types communs, ainsi que des cas d'utilisation vers le contexte (qui contient les acteurs), nous obtenons le diagramme suivant :

**Figure 6–8**

Structuration du modèle en packages



Insistons sur le fait que le découpage en packages proposé ici est un choix purement arbitraire. On peut également regrouper des éléments de natures différentes dans un même package (cas d'utilisation, blocs, activités, etc.), pour insister davantage sur la décomposition architecturale du système.

# TROISIÈME PARTIE

## La modélisation dynamique

La partie III concerne la modélisation dynamique. Les diagrammes comportementaux incluent le diagramme de cas d'utilisation, le diagramme d'activité, le diagramme de séquence et le diagramme d'états. Nous avons déjà vu le diagramme de cas d'utilisation au chapitre 2 et le diagramme de séquence au chapitre 3. Nous verrons dans cette partie toute la puissance du diagramme d'états, pour modéliser le cycle de vie des éléments à dynamique prédominante, ainsi que celle du diagramme d'activité, qui permet de modéliser avec précision des algorithmes complexes.

# Le diagramme d'états

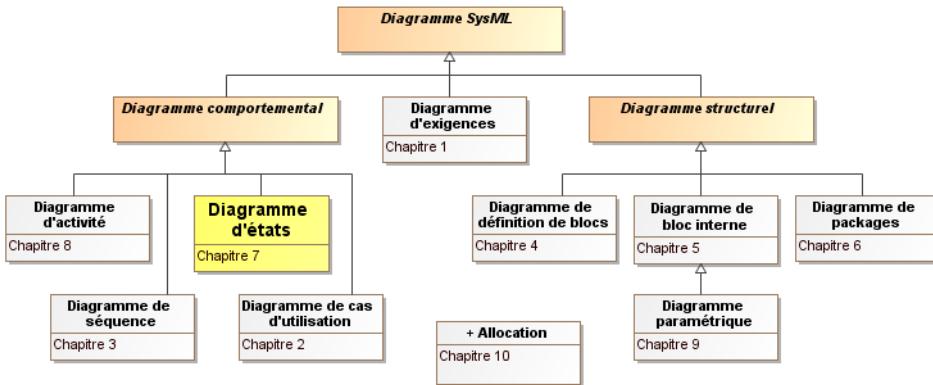
Ce chapitre présente le diagramme d'états. Le diagramme de machines à états décrit les transitions entre états et les actions que le système ou ses parties réalisent en réponse aux événements.

- ▶ état
- ▶ transition
- ▶ événement
- ▶ condition
- ▶ effet
- ▶ activité

## Notation de base

SysML a repris le concept bien connu de machine à états finis, qui consiste à s'intéresser au cycle de vie d'une instance générique d'un bloc particulier au fil de ses interactions, dans tous les cas possibles. Cette vue locale d'un bloc, qui décrit comment il réagit à des événements en fonction de son état courant et comment il passe dans un nouvel état, est représentée graphiquement sous la forme d'un diagramme d'états.

Sur le premier diagramme d'états, nous avons fait figurer deux états appelés respectivement État 1 et État 2. L'événement 1 fait passer de État 1 à État 2, l'événement 2 porte un paramètre appelé param et fait revenir de État 2 à État 1.



### B.A.-BA État

Un état représente une situation durant la vie d'un bloc pendant laquelle :

- il satisfait une certaine condition ;
- 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.

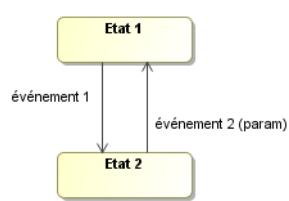
### B.A.-BA Événement

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.

**Figure 7–1**

Premier diagramme d'états



Heureusement, tous les blocs du modèle ne requièrent pas nécessairement une machine à états. Il s'agit donc de trouver ceux qui ont un comportement dynamique complexe nécessitant une description plus poussée. Cela correspond à l'un des deux cas suivants :

- le bloc réagit différemment lors d'occurrences successives du même événement : chaque type de réaction caractérise un état particulier ;
- le bloc doit organiser certaines opérations dans un ordre précis : dans ce cas, des états séquentiels permettent de préciser la chronologie forcée des événements d'activation.

#### B.A.-BA 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.

#### B.A.-BA Condition

Une condition (ou condition de garde) est une expression booléenne qui doit être vraie lorsque l'événement 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.

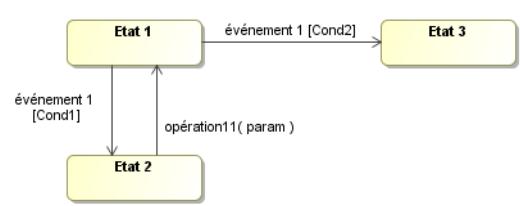
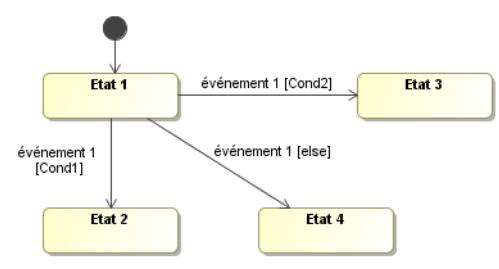


Figure 7-2 Second diagramme d'états

Sur le deuxième diagramme d'états, nous avons ajouté un troisième état appelé État 3. L'événement 1 fait passer de État 1 à État 2 si la condition de garde Cond1 est vraie, et de État 1 à État 3 si la condition de garde Cond2 est vraie. D'un point de vue méthodologique, il convient de vérifier que Cond2 est bien la condition inverse de Cond1, ou carrément d'utiliser explicitement l'inverse de Cond1 : [NOT Cond1]. Précisons que si ni Cond1 ni Cond2 ne sont vraies, l'événement 1 est perdu, et que si les deux conditions sont vraies simultanément, le diagramme est dit « non-déterministe » et on ne peut savoir dans quel état le bloc passera.

Lorsqu'il y a plus de deux conditions, il est recommandé d'utiliser le mot-clé `else` pour garantir l'exhaustivité comme illustré sur le diagramme suivant, à la place de l'expression complexe [NOT Cond1 AND NOT Cond2].

**Figure 7–3**  
Utilisation du mot-clé `else`



Notez également que l'événement déclencheur peut correspondre à l'invocation d'une opération du bloc : c'est le cas pour opération11 qui possède un paramètre et qui fait passer le bloc de État 2 à État 1.

### B.A.-BA Effet, action, activité

Une transition peut spécifier un comportement optionnel réalisé par le bloc lorsque la transition est déclenchée. Ce comportement est appelé « effet » : cela peut être une simple action ou une séquence d'actions. Une action peut représenter la mise à jour d'une valeur, un appel d'opération, ainsi que l'envoi d'un signal à un autre bloc. L'exécution de l'effet est unitaire et ne permet de traiter aucun événement supplémentaire pendant son déroulement. Les activités durables (do-activity) ont une certaine durée, peuvent être interrompues et sont associées aux états.

### B.A.-BA État initial, état final

En plus de la succession d'états « normaux » correspondant au cycle de vie d'un bloc, 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.
- Il est possible d'utiliser plusieurs états finals (ou finaux, les deux se disent...) pour distinguer par exemple la destruction normale de l'élément en fin de vie d'une destruction accidentelle.

La notation de base du diagramme d'états est donnée par la figure suivante. L'état initial est État 1, l'état final est atteint à partir de État 3 suite à l'événement destruction. La transition déclenchée par opération11 définit un effet appelé effet1. État 3 possède une activité durable appelée activité1. Dans l'état 3, l'événement 2 déclenche un effet mais ne fait pas changer d'état (transition réflexive).

### B.A.-BA Cartouche

Le cartouche général du diagramme d'états est de la forme :

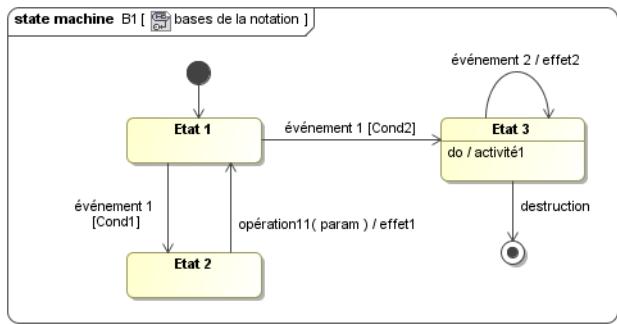
**stm [machine à états] nom de la machine à états [nom du diagramme]**

Le type d'élément de modèle est optionnel puisqu'il s'agit toujours d'une machine à états.

L'outil MagicDraw utilise la légère variante suivante :

**state machine nom de l'élément englobant [nom du diagramme] ;**  
comme on peut le voir sur le diagramme précédent.

**Figure 7–4**  
Notation de base du diagramme d'états



## Compléments

### Événement interne ou temporel

La démarche de construction d'un diagramme d'états peut s'énoncer comme suit :

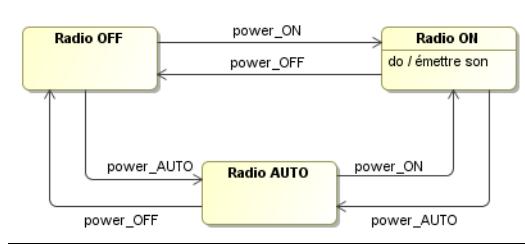
- représenter tout d'abord la séquence d'états qui décrit le comportement nominal d'un bloc, avec les transitions qui y sont associées ;
- ajouter progressivement les transitions qui correspondent aux comportements alternatifs ou d'erreur ;
- compléter les effets sur les transitions et les activités dans les états.

Nous allons appliquer cette démarche à la construction du diagramme d'états du bloc Radio.

Commençons par déclarer trois états principaux correspondant aux trois positions du bouton physique permettant d'allumer la radio, de l'éteindre, ou d'armer l'alarme du réveil. Les événements de changement de position sont nommés power\_ON, power\_OFF, power\_AUTO. Dans l'état Radio ON, nous avons déclaré une activité durable do / émettre son. Notez que pour l'instant, nous ne distinguons pas d'état initial, la création du radio-réveil étant peu intéressante à modéliser dans notre contexte d'utilisation.

**Figure 7–5**

Début du diagramme d'états du radio-réveil



En fait, dans l'état Radio AUTO, la radio est silencieuse jusqu'à ce que l'heure courante devienne l'heure d'alarme : `when (Hcourante = Halarme)`.

**ATTENTION Événement interne**

Les changements d'état internes(`change event`) se modélisent en utilisant le mot-clé `when` suivi d'une expression booléenne dont le passage de faux à vrai déclenche la transition.

Ensuite, la radio s'éteint toute seule au bout de 59 mn : `after (59 mn)`.

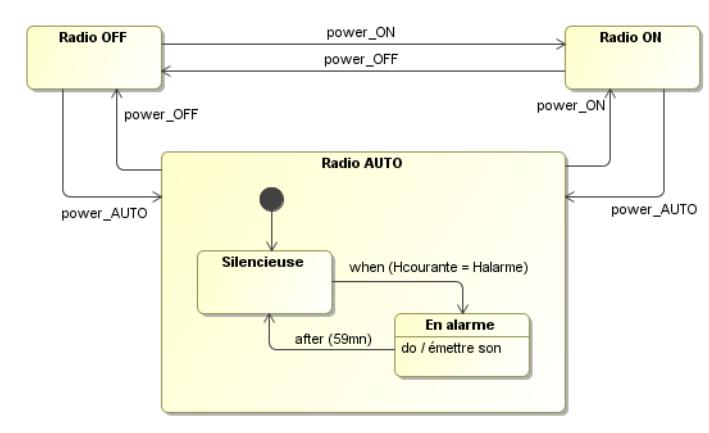
### ATTENTION Événement temporel

Le passage du temps (time event) se modélise en utilisant le mot-clé `after` suivi d'une expression représentant une durée, décomptée à partir de l'entrée dans l'état courant.

## État composite

Pour modéliser ce comportement plus détaillé, une première solution consiste à transformer l'état Radio AUTO en état composite (encore appelé super-état), et à dessiner les nouveaux états à l'intérieur. Notez qu'il faut ajouter un sous-état initial pour indiquer que lorsque le bloc passe dans l'état Radio AUTO, il rentre en fait directement dans le sous-état Silencieuse.

**Figure 7–6**  
Suite du diagramme  
d'états du radio-réveil



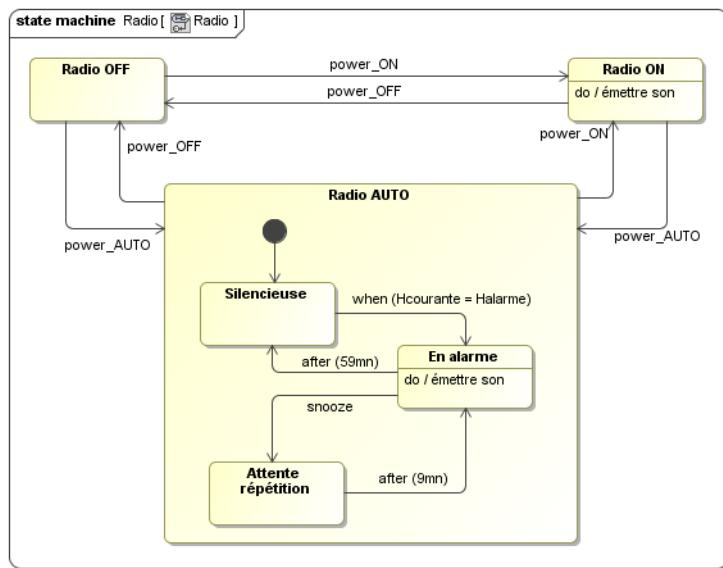
### B.A.-BA État composite

Un état composite (aussi appelé super-état) 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.

Ajoutons maintenant le comportement du bouton « Snooze ». Quand l'utilisateur appuie sur ce bouton, il interrompt provisoirement le son de la radio qui se réactive automatiquement après 9 mn.

**Figure 7–7**

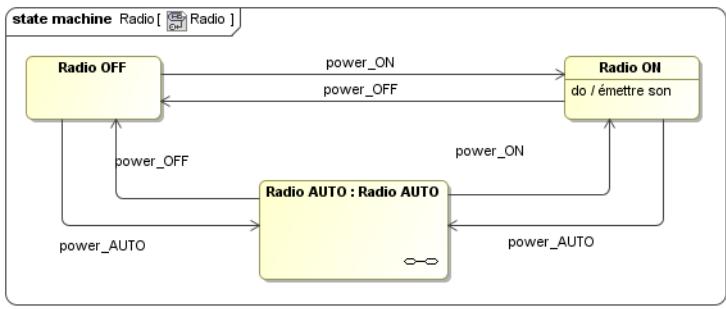
Suite du diagramme d'états du radio-réveil avec snooze



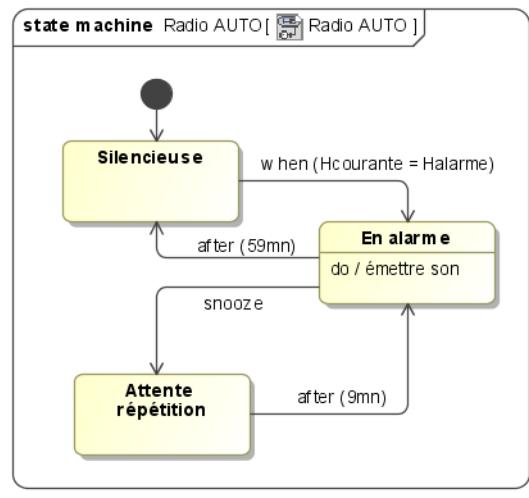
Pour modéliser ceci, nous ne pouvons pas nous contenter de revenir dans le sous-état Silencieuse. En effet, il nous faut réactiver le son au bout de 9 mn, donc une transition déclenchée par l'événement temporel *after (9 mn)*, qui revient dans l'état En alarme. Mais comme nous allons dans Silencieuse dès l'entrée dans le super-état Radio AUTO, cela voudrait dire que la radio s'active au bout de 9 mn, sauf si l'heure d'alarme arrive avant... Il faut donc absolument créer un troisième sous-état dans Radio AUTO.

Une autre façon de représenter un état composite consiste à ajouter un symbole en forme d'halte en bas à droite du rectangle à coins arrondis, puis à décrire les transitions entre ses sous-états dans un autre diagramme. On peut ainsi faire de la décomposition hiérarchique d'états, en gardant chaque niveau lisible et relativement simple. On peut même réutiliser des machines à états décrites par ailleurs. C'est pour cela qu'un état composite possède un nom plus complexe, du type Nom de l'état : Nom de la machine à états. Nous avons donc repris le diagramme précédent en transformant Radio AUTO en sous-machine à états représentée dans un second diagramme.

**Figure 7–8**  
Diagramme d'états  
du radio-réveil avec état  
composite non expansé



**Figure 7–9**  
Diagramme d'états  
de la sous-machine  
Radio AUTO



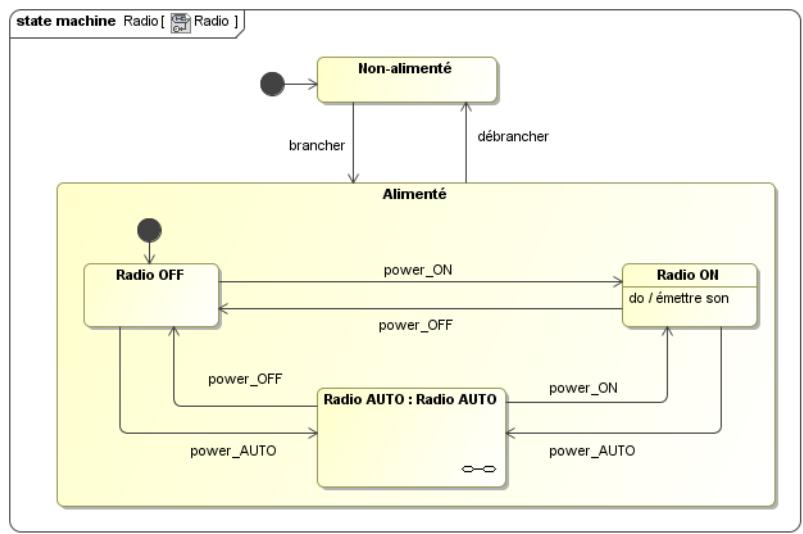
Imaginons maintenant que nous souhaitons modéliser la mémorisation de l'état de la radio après une coupure de courant intempestive. Il faut déjà ajouter deux états : Alimenté et Non-alimenté, avec deux transitions provoquées par les événements brancher et débrancher. L'état Alimenté correspond en fait au super-état de tous ceux que nous avons modélisé jusqu'à présent. Au passage, nous avons opté pour faire de l'état Non-alimenté notre état initial.

Ce modèle est bon si l'on considère que lorsque l'on rebranche le radio-réveil après l'avoir débranché, il retourne toujours dans l'état Radio OFF (sous-état initial). Mais si l'on souhaite un comportement un peu plus intelligent qui conserve l'état courant de la radio (ON, OFF ou AUTO) en cas de débranchement intempestif (avec pile de sauvegarde),

ce modèle n'est plus satisfaisant. SysML fournit une construction très pratique dans ce cas : le pseudo-état History.

**Figure 7–10**

Diagramme d'états avec gestion de l'alimentation

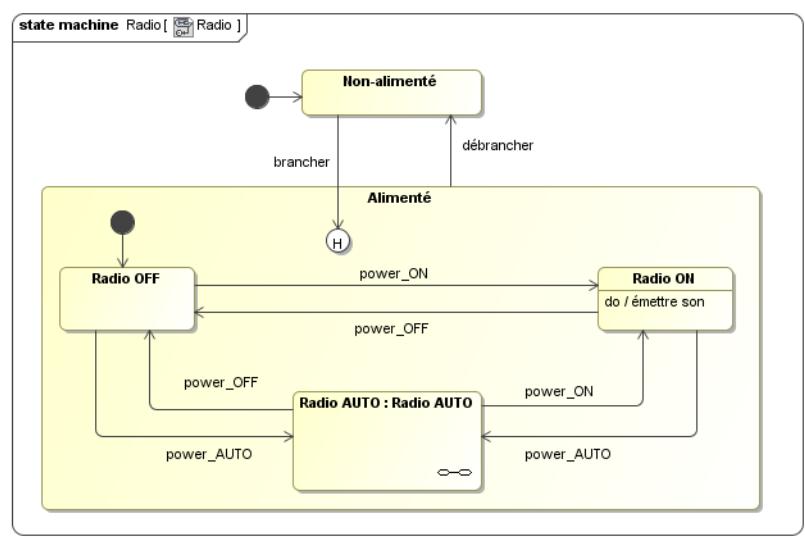


#### ATTENTION Pseudo-état History

L'activation du pseudo-état History permet à un super-état de se souvenir du dernier sous-état séquentiel qui était actif avant une transition sortante. Une transition vers l'état History rend à nouveau actif le dernier sous-état actif, au lieu de le ramener vers le sous-état initial.

Le diagramme d'états devient alors comme sur la figure suivante.

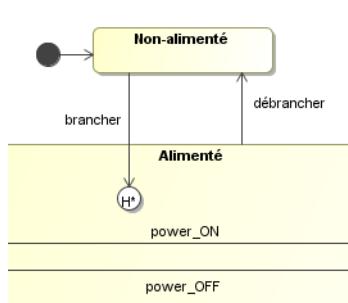
**Figure 7–11**  
Diagramme d'états avec gestion de l'alimentation et historique



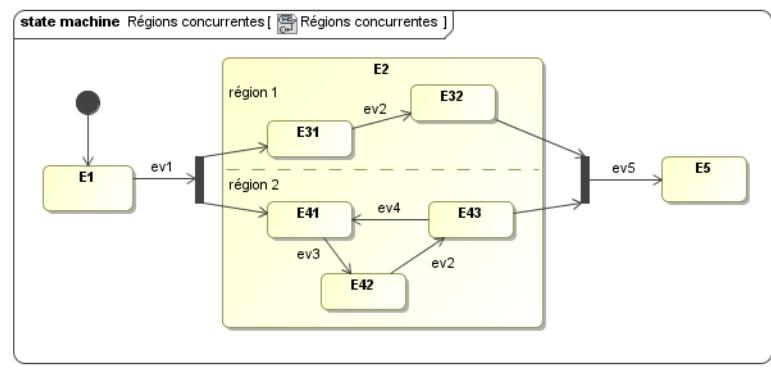
Si l'on voulait que la notion d'historique se propage dans la hiérarchie d'états et de sous-états, quel que soit le niveau de profondeur de leur décomposition, SysML propose un *deep history*, noté avec une étoile en exposant du  $H : H^*$ .

Un état composite peut également contenir des régions concurrentes, il suffit graphiquement de le séparer par des traits pointillés. Chaque région peut alors être nommée (optionnel). Elle contient ses propres états et ses propres transitions. Les régions sont dites concurrentes car elles peuvent évoluer en parallèle et indépendamment. L'état courant de l'élément concerné devient alors un vecteur à plusieurs lignes (autant que de régions).

**Figure 7–12**  
Notation de l'historique profond (deep history)



**Figure 7–13**  
Régions concurrentes



Dans l'exemple précédent, à partir de l'état E1, quand l'événement ev1 arrive, l'élément passe dans l'état composite E2. Cela signifie qu'il est à la fois dans les états disjoints E31 et E41. Ensuite, suivant l'ordre d'arrivée des événements ev2, ev3 ou ev4, chaque région va évoluer indépendamment. Pour passer à l'état E5, il faudra que l'élément soit à la fois dans E32 et E43 quand ev5 arrivera.

Cette notion d'états parallèles est certes intéressante, mais complexifie la lecture du diagramme d'états. Avec les concepts de parties, ports et connecteurs fournis par le diagramme de bloc interne (ibd) et la possibilité de décomposition associée, il est relativement facile de se passer des régions concurrentes en décrivant indépendamment la dynamique de chacune des parties.

## Autres notations avancées

Pour modéliser des effets qui sont exécutés par toutes les transitions en entrée ou en sortie d'un état, SysML propose des raccourcis de notation utilisant des mots clés à l'intérieur du symbole de l'état.

### B.A.-BA Effets d'entrée et de sortie

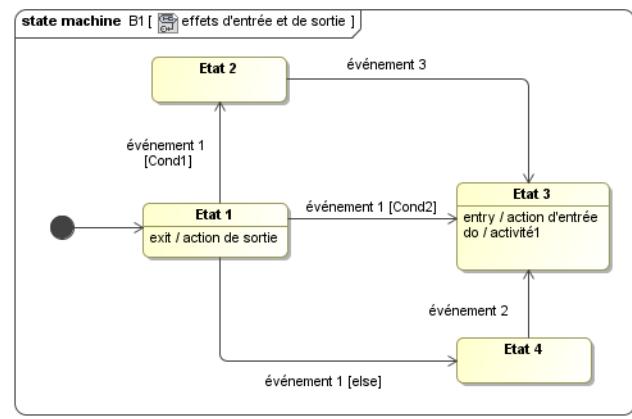
Un effet d'entrée (introduit par le mot-clé `entry` à l'intérieur du symbole d'un état) représente un effet qui est exécuté chaque fois que l'on entre dans cet état. Cela permet de factoriser un même effet qui sera déclenché par toutes les transitions qui entrent dans l'état. L'effet de sortie (introduit par le mot-clé `exit`) est l'effet symétrique en sortie de l'état.

Le diagramme qui suit utilise ces deux notations, pour éviter de dupliquer les effets sur les transitions (comme sur celui d'après, plus fouillis).

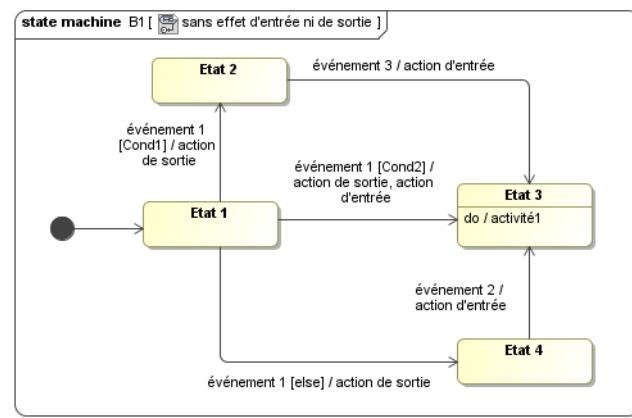
Notez au passage qu'un effet peut contenir une séquence d'actions, comme celui déclaré sur la transition entre État 1 et État 3 : l'action de sortie sera exécutée, suivie immédiatement de l'action d'entrée.

Imaginons maintenant que l'activité 1 de l'état 3 se termine automatiquement une fois que le traitement correspondant est achevé avec succès. Comment représenter la transi-

**Figure 7–14**  
Effets d'entrée et de sortie



**Figure 7–15**  
Même diagramme sans effets d'entrée et de sortie



tion qui fait sortir de l'état à la compléition de l'activité, puisqu'elle n'est pas déclenchée par un événement de réception de message, ni par un timeout ? Nous pourrions utiliser le mot-clé `when` indiquant un changement interne, mais comment nommer ce changement simplement ? SysML propose un autre raccourci de notation : la transition de compléition (ou transition automatique).

### B.A.-BA Transition de compléition

Une activité durable à l'intérieur d'un état peut être soit :

- continue : elle ne cesse que lorsque se produit un événement qui fait sortir l'élément de l'état ;
- finie : elle peut également être interrompue par un événement, mais elle cesse de toute façon d'elle-même au bout d'un certain temps, ou quand une certaine condition est remplie.

La transition de compléition d'une activité finie, aussi appelée transition automatique, est représentée en SysML sans nom d'événement ni mot-clé.

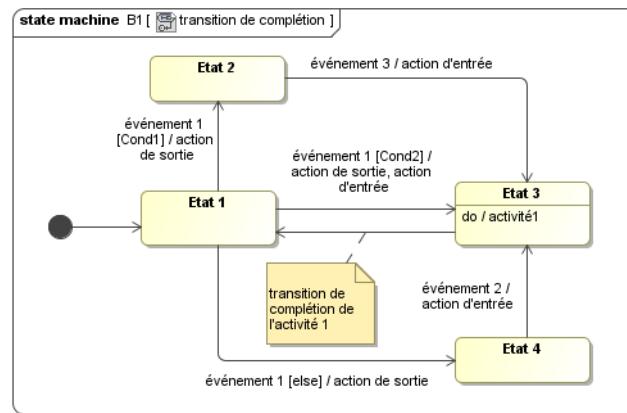


Figure 7-16 Diagramme d'état avec transition de compléition

### ATTENTION Transition propre ou transition interne ?

Dans le cas d'une transition propre, l'élément quitte son état de départ pour y retourner aussitôt. Mais cela peut avoir des effets secondaires non négligeables comme l'interruption puis le redémarrage d'une activité durable, ainsi que la réalisation d'effets en entrée ou en sortie de l'état. De plus, lorsqu'un état est décomposé en sous-états, une transition propre ramène forcément l'élément dans le sous-état initial.

Pour résoudre ce problème, SysML propose le concept de transition interne (*internal transition*). Une transition interne représente un couple (événement/effet) qui n'a aucune influence secondaire sur l'état courant. Elle est notée simplement à l'intérieur du symbole de l'état, comme illustré sur la figure suivante. Dans l'état 2, événement 2 peut arriver et déclencher alors uniquement effet2. Dans l'état 3, par contre, événement 2 déclenche successivement : action de sortie, effet2, puis action d'entrée. De plus, l'activité 1 est interrompue, puis redémarrée.

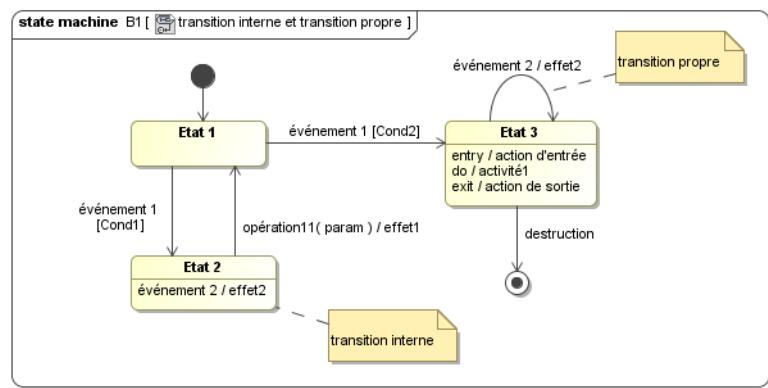


Figure 7-17 Transition propre ou transition interne

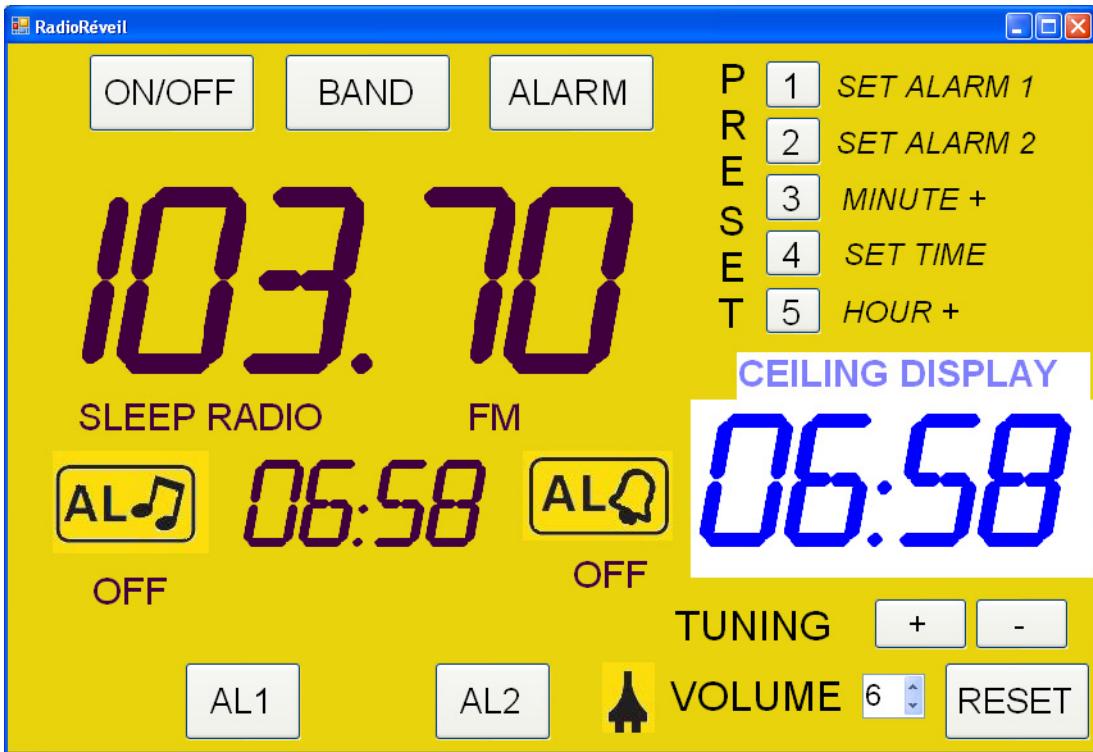


Figure 7-18 IHM du Radio-réveil simulée en VB

Le formalisme du diagramme d'états SysML contient encore quelques subtilités avancées et rarement indispensables (entry point, exit point, jonction, etc.). Ce diagramme est très

puissant, mais aussi très complexe. N'oubliez pas d'adapter le niveau de votre modélisation à vos objectifs et à vos lecteurs !

## Animation du diagramme d'états

Certains outils UML/SysML permettent d'aller plus loin que la simple édition de modèles : citons en particulier l'animation (ou l'exécution) des diagrammes d'états. C'est le cas par exemple de l'outil Rhapsody d'IBM/Telelogic, ou d'Artisan Studio de Artisan.

Olivier Casse, spécialiste de l'outil Artisan, m'a proposé de réaliser un prototype exécutable du Radio-réveil grâce à sa connaissance experte de ce genre d'outil. Il a également réalisé une IHM en VisualBasic, afin de permettre de stimuler le diagramme d'états par des événements externes, et de visualiser les valeurs des sorties, comme dans la réalité.

Le téléchargement de l'exemple complet est possible sur un site qui sera précisé sur celui des éditions Eyrolles :

- diagrammes SysML Artisan (téléchargeables dans la version gratuite Artisan Studio Uno disponible sur le site de l'éditeur : [www.artiansoftwaretools.com/StudioUno](http://www.artisansoftwaretools.com/StudioUno)) ;
- sources VB avec l'exécutable résultant de l'IHM et appels à l'API Artisan pour la connexion en mode simulation de la version industrielle d'Artisan Studio, nécessitant une licence ;
- exécutable du modèle du radio-réveil généré en C++ à partir de la version complète d'Artisan (car Uno ne permet pas la génération de code ni la simulation) et relié à une autre IHM fonctionnant de façon autonome, sans l'outil Artisan Studio.

Une copie d'écran de l'exécution du diagramme d'états sous Artisan est donnée ci-après.

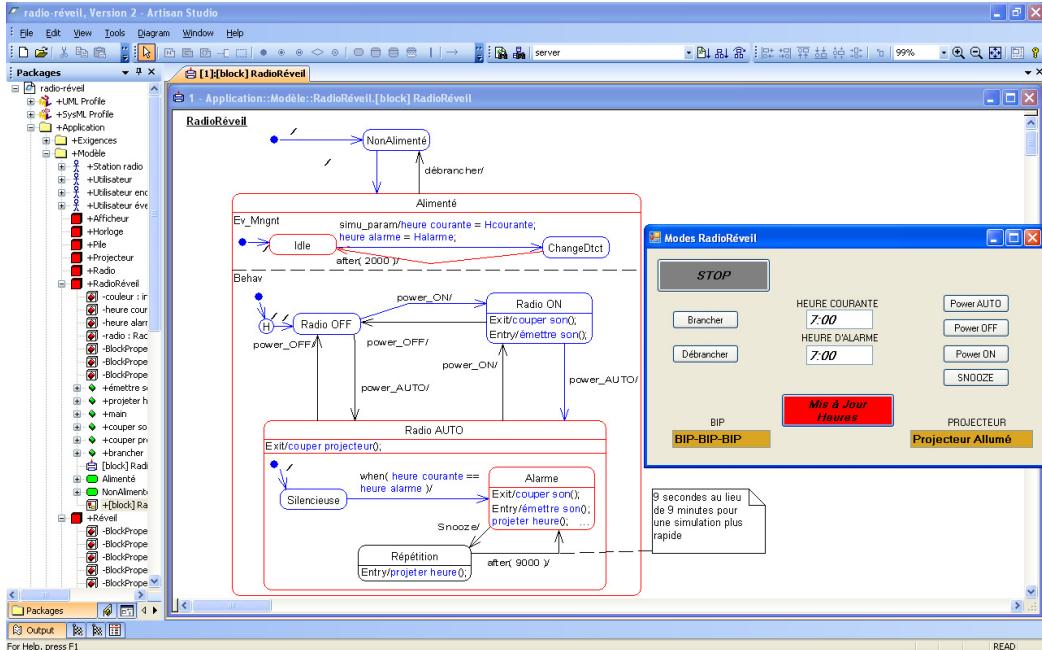


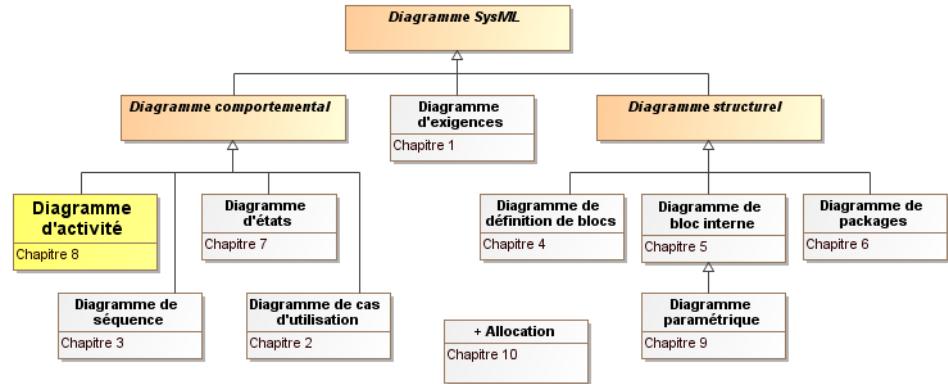
Figure 7–19 Copie d'écran de la simulation du Radio-réveil avec Artisan

# 8

## Le diagramme d'activité

Ce chapitre présente le diagramme d'activité. Le diagramme d'activité représente les flots de données et de contrôle entre les actions. Il est utilisé majoritairement pour l'expression de la logique de contrôle et d'entrées/sorties.

- ▶ activité
- ▶ action
- ▶ décision
- ▶ join/fork



## Notation de base

Le diagramme d'activité est l'un des diagrammes dynamiques proposés par SysML. Il ressemble fondamentalement à un traditionnel diagramme fonctionnel (à la SADT ou SA/RT), montrant le flot de contrôle d'action en action. Mais il propose des capacités supplémentaires importantes comme celle de pouvoir faire le lien avec les blocs de la modélisation structurelle et celle de pouvoir modéliser des flux continus.

### B.A.-BA Action

L'action est l'unité fondamentale de spécification comportementale en SysML. Elle représente un traitement ou une transformation. Les actions sont contenues dans les activités, qui fournissent leur contexte.

### B.A.-BA Flot

Un flot est un contrôle de séquençage pendant l'exécution de nœuds d'activité. Les flots de contrôle sont de simples flèches reliant deux nœuds (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 éléments de base du diagramme d'activité sont les suivants :

- des actions ;
- des flots de contrôle entre actions ;
- des décisions (aussi appelées branchements conditionnels) ;
- un début et une ou plusieurs fins possibles.

### B.A.-BA Décision

Une décision est un nœud de contrôle structuré représentant un choix dynamique entre plusieurs conditions qui doivent être mutuellement exclusives. Elle est représentée par un losange qui possède un arc entrant et plusieurs arcs sortants.

La notation graphique de base est présentée sur la figure suivante.

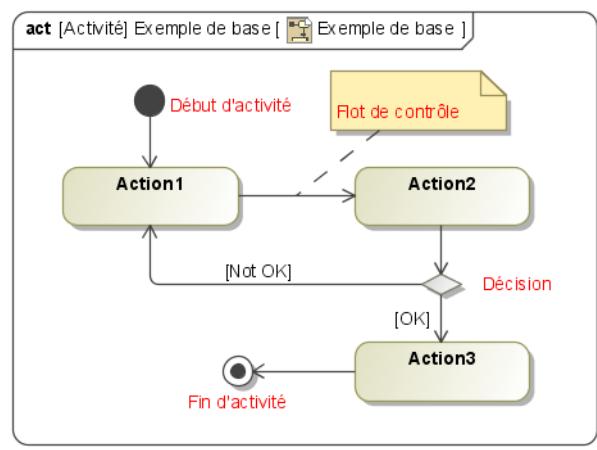
Dans le diagramme précédent, l'activité démarre avec l'action 1. Lorsque l'action 1 est terminée, elle produit un jeton (*token*) qui va alimenter l'action 2, via le flot de contrôle. L'action 2 démarre alors et possède sa propre durée. Lorsque l'action 2 se termine, elle produit à son tour un jeton qui arrive jusqu'à la décision. Suivant la condition OK ou Not OK, le jeton va activer l'action 3 ou l'action 1. Si c'est l'action 3 qui est activée, elle s'exécute et sa terminaison fait également terminer l'activité englobante. Si c'est l'action 1 qui a été activée de nouveau, elle s'exécute et sa terminaison fait de nouveau exécuter l'action 2 puis tester la condition, etc.

## B.A.-BA Cartouche

Le cartouche général du diagramme d'activité est de la forme :

**act [activité] nom de l'activité [nom du diagramme]**

Le type d'élément de modèle est optionnel puisqu'il s'agit toujours d'une activité.



**Figure 8-1** Les bases du diagramme d'activité

Par le biais des jetons, le concept de flot de contrôle permet ainsi d'exprimer des contraintes sur la séquence d'exécution des actions. Un jeton qui arrive sur un flot de contrôle entrant permet le démarrage de l'action concernée. La terminaison d'une action met un jeton à disposition sur le flot de contrôle sortant. Lorsqu'un flot de contrôle connecte deux actions, l'action cible à l'extrémité du flot de contrôle ne peut pas démarrer tant que l'action source ne s'est pas terminée.

## RÉFÉRENCE Réseau de Petri

Ce concept formel de jeton vient du formalisme des réseaux de Petri, un modèle mathématique servant à représenter divers systèmes (informatiques, industriels...) travaillant sur des variables discrètes. Un réseau de Petri se représente par un graphe orienté reliant des places et des transitions. Deux places ne peuvent pas être reliées entre elles, ni deux transitions. Les places peuvent contenir des jetons, représentant généralement des ressources disponibles. La distribution des jetons dans les places est appelée marquage du réseau de Petri. Les entrées d'une transition sont les places desquelles part une flèche pointant vers cette transition, et les sorties d'une transition sont les places pointées par une flèche ayant pour origine cette transition.

Un réseau de Petri évolue lorsqu'on exécute une transition : des jetons sont retirés dans les places en entrée de cette transition et des jetons sont déposés dans les places en sortie de cette transition. L'exécution d'une transition est une opération indivisible qui est déterminée par la présence du jeton sur la place d'entrée.

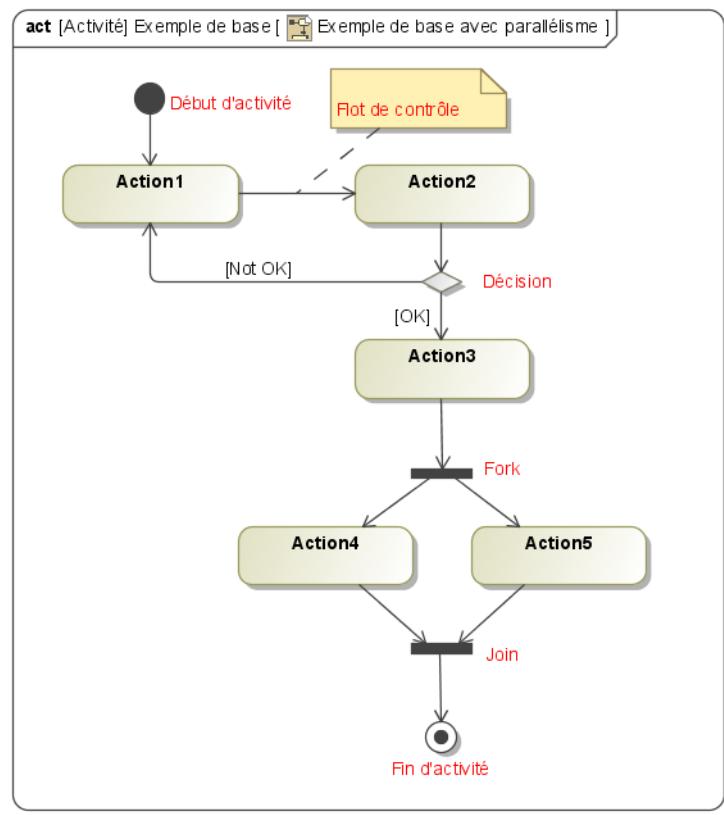
La décision est l'un des nœuds de contrôle proposés par SysML. Le début et la fin de l'activité englobante en sont également. Nous allons expliquer maintenant le *fork* (débranchement parallèle) et le *join* (synchronisation ou rendez-vous).

Dans ce nouveau diagramme, que se passe-t-il à la terminaison de l'action 3 ? Le jeton qui est produit se divise en passant le *fork*. Puisque le *fork* possède deux flots en sortie, cela signifie que deux jetons indépendants existent maintenant en parallèle. Chacun des deux jetons va activer une action. L'action 4 et l'action 5 démarrent donc exactement en même temps et s'exécutent de façon indépendante.

### B.A.-BA Fork

Un fork est un nœud de contrôle structuré 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.

**Figure 8–2**  
Diagramme d'activité  
avec parallélisme



La fin d'activité n'aura lieu que lorsque le *join* aura produit son jeton. Or, un *join* ne produit son jeton de sortie que lorsque tous ses flots d'entrée possèdent leur jeton. Il faut donc que les deux actions 4 et 5 soient terminées, dans un ordre quelconque, pour que l'activité englobante se termine.

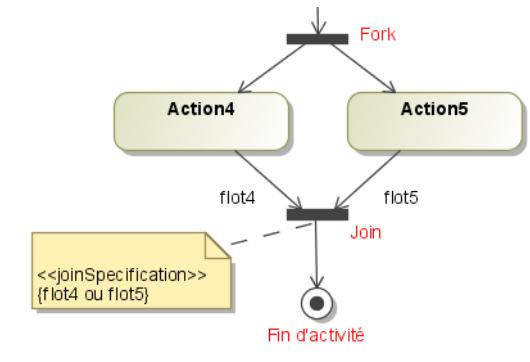
#### B.A.-BA Join

Un join est un nœud 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.

#### ATTENTION Join specification

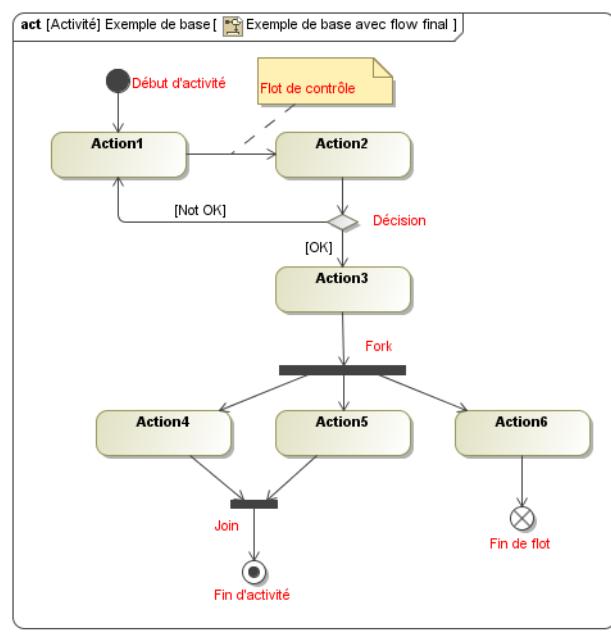
Le comportement par défaut du join peut être redéfini par une spécification : join specification. Il doit s'agir d'une expression logique utilisant les noms des flots entrants, qui doivent donc être nommés pour l'occasion.

**Figure 8-3**  
Join specification



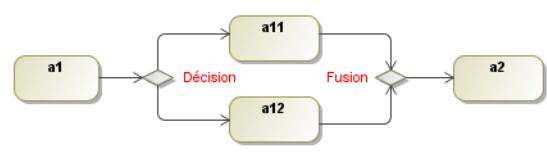
Un autre nœud de contrôle intéressant est la fin de flot (*flow final*). Contrairement à la fin d'activité qui est globale à l'activité, la fin de flot est locale au flot concerné et n'a pas d'effet sur l'activité englobante. Enrichissons notre exemple : ajoutons une troisième action parallèle. Nous ne voulons pas synchroniser cette nouvelle action 6 avec les deux précédentes. Nous souhaitons que la terminaison d'action 6 n'ait aucun impact sur le reste de l'activité, alors que la terminaison conjointe des actions 4 et 5 interrompt l'activité, même si l'action 6 est en cours.

**Figure 8–4**  
Fin de flot



Il existe un dernier nœud de contrôle assez subtil : la fusion (*merge*). C'est l'inverse de la décision : le même symbole du losange, mais cette fois-ci avec plusieurs flots entrants et un seul sortant.

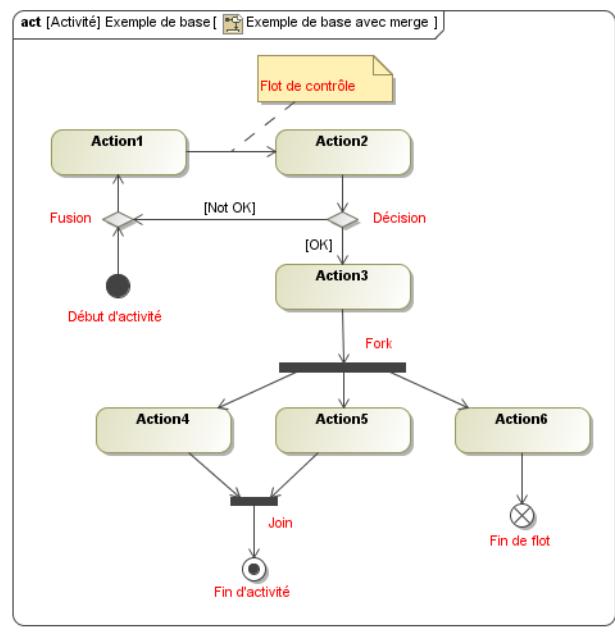
**Figure 8–5**  
Fusion de flots (merge)



Dans le petit exemple précédent, si l'on avait relié directement les actions a11 et a12 à l'action a2, celle-ci n'aurait jamais été activée ! En effet, elle aurait attendu indéfiniment un jeton sur chaque flot entrant, donc deux jetons, alors qu'il n'existe qu'un seul jeton... La solution consiste donc à intercaler un nœud de fusion, permettant de garantir un seul flot entrant à l'action a2, qui sera bien activée dès que l'un des deux flots sortant de a11 ou de a12 sera porteur d'un jeton.

Si nous revenons au diagramme 4, on pourrait argumenter que l'action 1 possède deux flots entrants, et qu'il n'y aura jamais un jeton sur chaque ! La solution correcte consiste donc à ajouter une fusion comme illustré sur le prochain diagramme.

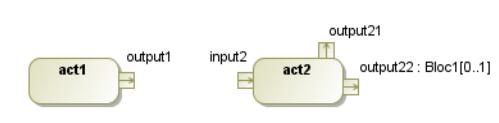
**Figure 8–6**  
Exemple de base complété



## Object node, object flow

Le diagramme d'activité sert non seulement à préciser la séquence d'actions à réaliser, mais aussi ce qui est produit, consommé ou transformé au cours de l'exécution de cette activité.

**Figure 8–7**  
Actions avec broches



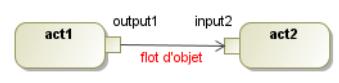
#### ATTENTION Multiplicité des broches

Chaque broche possède une multiplicité qui décrit le nombre minimum et maximum de jetons que l'action va consommer ou produire au cours d'une exécution. Les broches jouent ainsi un rôle de buffer où les jetons d'entrée et de sortie peuvent être stockés avant ou pendant l'exécution de l'action. Si la multiplicité minimale est zéro, la broche est optionnelle, sinon elle est obligatoire.

Sur le diagramme précédent, les broches, non reliées, sont dessinées avec une flèche indiquant la direction d'entrée ou de sortie. Par contre, lorsque les broches de plusieurs

actions sont reliées par un flot d'objet, les flèches sont généralement omises puisque le sens est évident.

**Figure 8–8**  
Actions avec broches  
et flot d'objet



#### ATTENTION Sémantique d'exécution

Résumons les règles d'activation et de terminaison d'une action :

- l'activité englobante doit être en exécution ;
- pour qu'une action démarre, il faut un jeton sur chaque flot de contrôle et au moins le nombre minimum de jetons sur chaque broche d'entrée ;
- l'action démarre alors son exécution et tous les jetons présents sur les broches d'entrée sont disponibles pour être consommés ;
- pour qu'une action se termine, il faut que le nombre de jetons produits sur chaque broche de sortie soit supérieur ou égal à la multiplicité minimale ;
- une fois que l'action est terminée, tous les jetons présents sur les broches de sortie sont disponibles pour les actions connectées en entrée. Un jeton est également disponible sur chaque flot de contrôle en sortie ;
- la terminaison de l'activité englobante entraîne la terminaison de l'action, quelque soit son propre état.

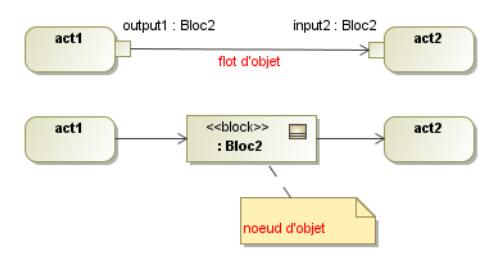
Revenons sur les flots d'objet. Nous avons vu une première façon de les représenter : une ligne connectant une broche de sortie à une broche d'entrée avec une flèche vers cette dernière. Il est clair que la direction du flot d'objet doit être compatible avec le sens des broches aux extrémités, mais aussi que les types des broches doivent être compatibles. Cela signifie que les blocs typant les broches connectées sont identiques ou dans une rela-

## Compléments

### Appel d'activité

Les activités fournissent le contexte dans lequel les actions s'exécutent. Mais les activités elles-mêmes peuvent être utilisées et surtout réutilisées à travers des actions d'appel (*call-BehaviorAction*). Il est ainsi possible de se constituer des bibliothèques d'activités réutilisables, décrivant des algorithmes classiques dans le contexte métier. Il est également possible de décrire des traitements très complexes de façon descendante avec un nombre de niveaux de décomposition arbitraire.

**Figure 8–9**  
Actions avec broches,  
flot d'objet et nœud d'objet



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.`

**Figure 8–10**

Notation de l'action d'appel

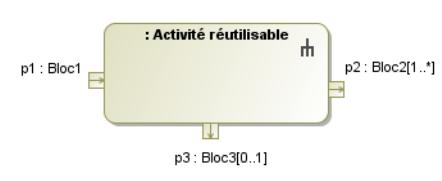


L'action d'appel pouvant posséder des broches d'entrée/sortie, l'activité appelée peut posséder à son tour des paramètres qui peuvent être typés, inclure une direction (*in*, *out*, *inout*) et une multiplicité :

`nom du paramètre : type du paramètre[multiplicité]`

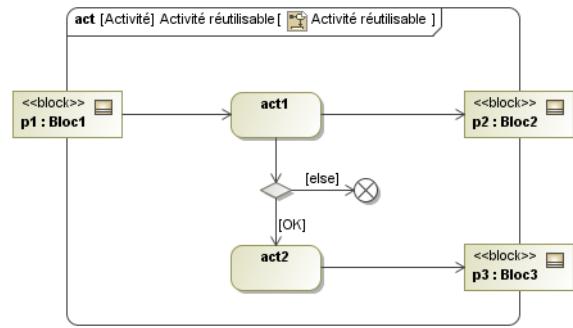
**Figure 8–11**

Notation de l'action d'appel avec paramètres



Ces paramètres sont représentables graphiquement par des rectangles à l'extérieur du cadre du diagramme d'activité précisant l'algorithme de l'activité réutilisée. Il n'y a pas de notation particulière pour indiquer la direction des paramètres, du coup il est recommandé de positionner les paramètres d'entrée à gauche et ceux de sortie à droite. Une fois que les paramètres de l'activité ont été connectés aux actions à l'intérieur, la direction devient explicite.

**Figure 8–12**  
Notation de l'activité  
avec paramètres



Le diagramme précédent montre deux actions à l'intérieur de l'activité réutilisable. L'action act1 consomme le paramètre d'entrée p1 et produit le paramètre de sortie p2. En parallèle, act1 active act2 si la condition OK est vraie. Dans ce cas, act2 produit le paramètre de sortie p3.

## Signaux et événements

En plus de consommer et de produire des paramètres, une activité peut recevoir et émettre des signaux. L'idée forte est de permettre à des activités de communiquer en incluant dans une activité l'émission d'un signal et dans une autre la réception d'événements. Il faut utiliser pour cela des types d'action particuliers, possédant chacun une représentation graphique spécifique :

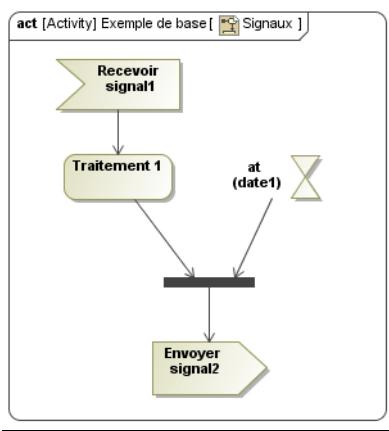
- *accept event action* ;

- *send signal action* ;
- *accept time event*.

Ce dernier type (*accept time event*) correspond à l'expiration d'un timer (*after*) ou à l'occurrence d'une date (*at*). Il n'a qu'une broche de sortie mais pas de broche d'entrée.

Nous verrons l'utilisation la plus courante des *accept event actions* dans le paragraphe suivant (région interruptible).

**Figure 8–13**  
Notation des signaux

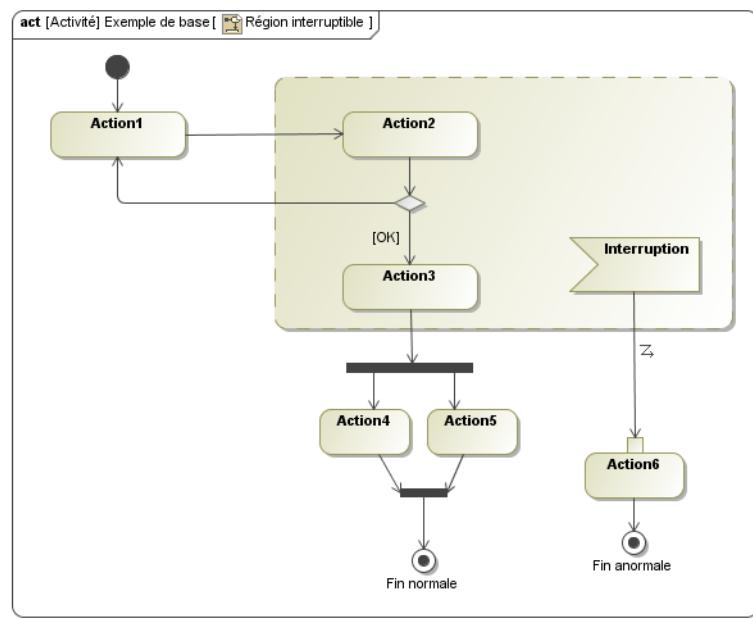


## Région interruptible

Nous avons expliqué que toutes les actions se terminent lorsque l'activité englobante se termine. Il peut arriver qu'on veuille terminer seulement un sous-ensemble des actions pour une raison particulière.

SysML propose le concept de région interruptible pour modéliser cette situation. Elle est représentée graphiquement par un rectangle aux bords arrondis en pointillés qui englobe un sous-ensemble des actions de l'activité. Il doit exister également un mécanisme d'interruption, typiquement une réception d'événement (*accept event*) suivie d'un flot d'interruption, représenté par un éclair.

**Figure 8–14**  
Région interruptible



Dans l'exemple précédent, l'événement Interruption est significatif si un jeton se trouve dans l'action 2 ou dans l'action 3. Dans ce cas, il dérouté le jeton présent vers l'action 6, au lieu de poursuivre sa route naturelle vers le *fork* avant les actions 4 et 5. Les actions autres que l'action 2 ou l'action 3 ne sont pas interruptibles.

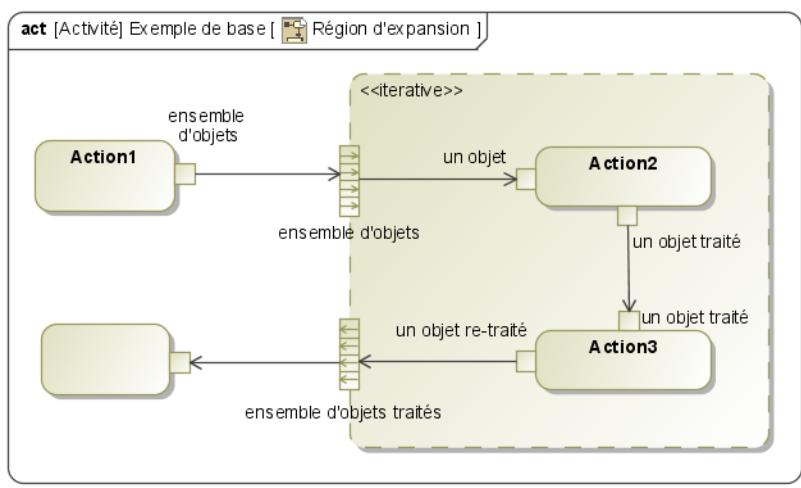
## Région d'expansion

Une région d'expansion est un nœud dans une activité qui accepte un ensemble d'objets en entrée, les traite individuellement et finalement les retourne en sortie tous traités. Cela permet de représenter finement le traitement de chaque objet d'un ensemble, dans le cas d'un algorithme commun. La région d'expansion se représente comme la région interruptible, mais avec un port multiple (quatre carrés au lieu d'un). Un mot-clé permet de préciser comment est traité l'ensemble :

- « *iterative* » : les objets de l'ensemble sont traités séquentiellement ;
- « *parallel* » : les objets de l'ensemble sont traités en parallèle ;
- « *streaming* » : les objets de l'ensemble sont traités comme un seul flot, un peu comme sur une chaîne de montage. Chaque objet n'a pas à attendre que le traitement de l'objet précédent soit complètement terminé.

Sur l'exemple précédent, l'ensemble d'objets en entrée de la région d'expansion est traité individuellement et séquentiellement par les actions 2 et 3, puis fourni globalement en sortie.

**Figure 8-15**  
Région d'expansion



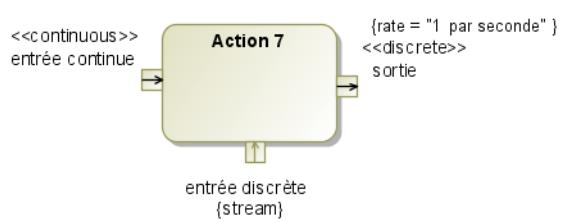
## Compléments sur les flots d'objet

Le comportement par défaut d'une action (ou activité) consiste à accepter des jetons en entrée au démarrage de celle-ci, et à fournir les jetons en sortie à sa terminaison. Au contraire, dans le cas d'une broche ou d'un paramètre de type `{stream}`, l'action ou l'activité va pouvoir continuer à accepter des entrées ou à produire des sorties pendant son exécution. Pensez typiquement à la lecture d'une vidéo sur internet : la lecture commence alors que la vidéo continue de se charger en mémoire.

Pour permettre la modélisation de systèmes continus, SysML permet de caractériser la nature du débit qui circule sur le flux : continu ou discret. On utilise pour cela des stéréotypes : « `continuous` » et « `discrete` ». Par défaut, un flux est supposé discret.

SysML autorise également la définition de taux (*rate*) sur les flux de type `{stream}` pour permettre de caractériser les distributions d'entrées et de sorties. Un flot continu est en fait une sorte de flot pour lequel le taux serait infini, ou inversement la période entre deux jetons nulle.

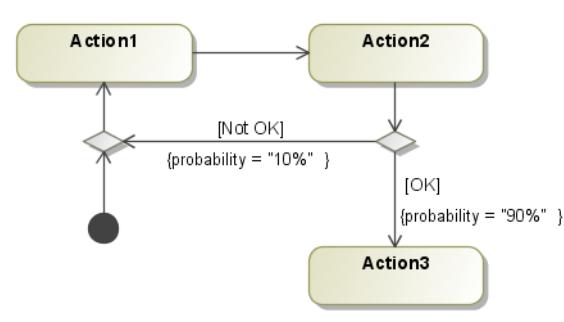
**Figure 8-16**  
Types de flots



L'action 7 sur le diagramme précédent possède trois broches :

- une broche d'entrée de type « `continuous` » ;
- une entrée discrète de type `{stream}` ;
- une sortie discrète avec un taux de 1 par seconde.

**Figure 8-17**  
Flots avec probabilité



SysML permet d'ajouter une indication de probabilité aux flots sortant d'une décision. Dans ce cas, tous les flots alternatifs doivent posséder une probabilité et la somme des probabilités doit être égale à 1.

## Opérateur de contrôle

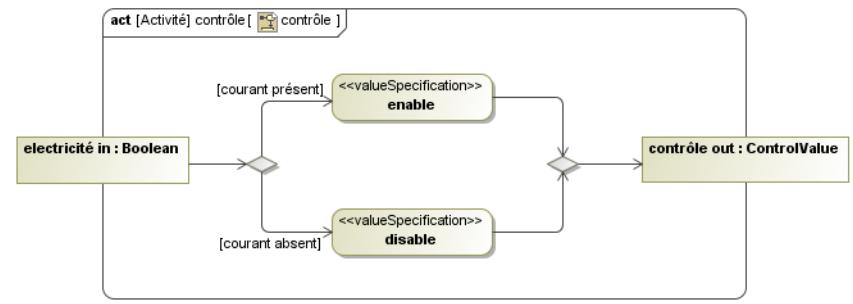
Une action avec des entrées/sorties discrètes et pas de type `{stream}` a une sémantique de démarrage et d'arrêt simple. Par contre, pour une action avec des entrées/sorties continues ou de type `{stream}`, il faut pouvoir intervenir de l'extérieur sur l'activation ou la désactivation. SysML propose à cet effet une valeur de contrôle (*ControlValue*) de type énumération ne possédant que deux valeurs possibles : *enable* et *disable*. La valeur *enable* correspond à un jeton de contrôle entrant, alors que la valeur *disable* correspond à la terminaison forcée de l'action. Un comportement particulier, appelé opérateur de contrôle (*ControlOperator*) est chargé de produire cette valeur de contrôle à travers un paramètre de sortie, de type *ControlValue*.

**Figure 8–18**  
Appel de l'opérateur  
de contrôle



Cet opérateur de contrôle doit être décrit à son tour par un diagramme d'activité expliquant comment est produite la *ControlValue*. Un exemple est donné sur le diagramme suivant, qui illustre la désactivation d'activités si le courant est absent, comme dans le cas du radio-réveil.

**Figure 8-19**  
Diagramme d'activité de l'opérateur de contrôle



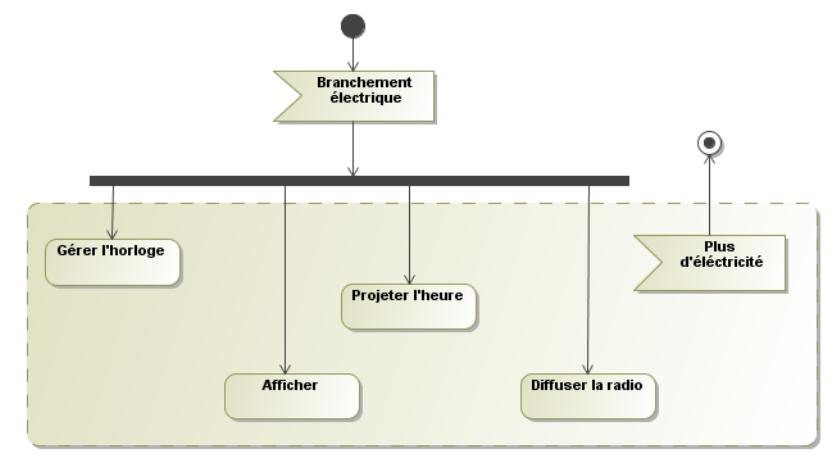
## Étude de cas

Pour notre radio-réveil, une première façon de démarrer le diagramme d'activité consiste à représenter les grandes fonctionnalités, après le branchement électrique. Les fonctions principales (gérer l'horloge, afficher, projeter l'heure et diffuser la radio) sont toutes démarées en parallèle dès le branchement.

L'activité englobante ne s'arrête que si l'alimentation électrique n'est plus disponible. Pour cela, une région interruptible est tout à fait adaptée.

Si nous nous plaçons dans l'hypothèse plus favorable d'une alimentation de secours, conformément aux exigences, l'absence d'alimentation électrique désactive toutes les fonctionnalités, sauf la gestion de l'horloge qui est préservée. Une façon élégante de le modéliser consiste à utiliser un opérateur de contrôle, comme expliqué dans le paragraphe précédent.

**Figure 8–20**  
Début du diagramme  
d'activité de l'étude  
de cas

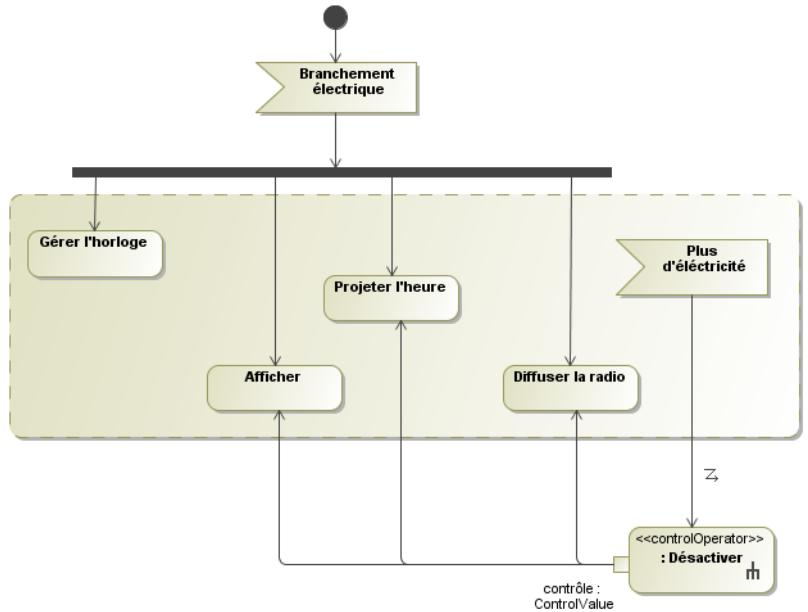


Si nous nous intéressons maintenant aux flots d'objet, on peut dire que l'action Gérer l'horloge fournit toutes les minutes l'horodatage courant aux actions d'affichage et de projection. Ces deux dernières produisent une sortie continue de type Horodatage.

De même, l'action diffuser la radio reçoit en permanence des ondes radio qu'elle transforme en son.

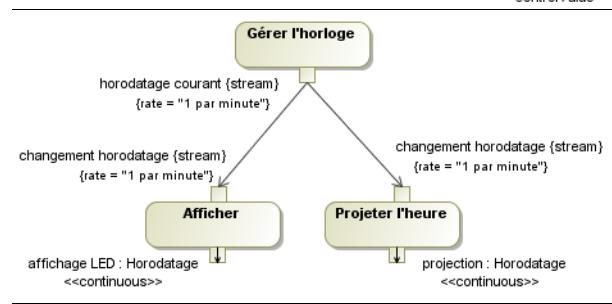
## **Figure 8–21**

### Exemple d'opérateur de contrôle



## **Figure 8–22**

### Exemple de flots d'objet



**Figure 8–23**

Autre exemple de flots continus



# QUATRIÈME PARTIE

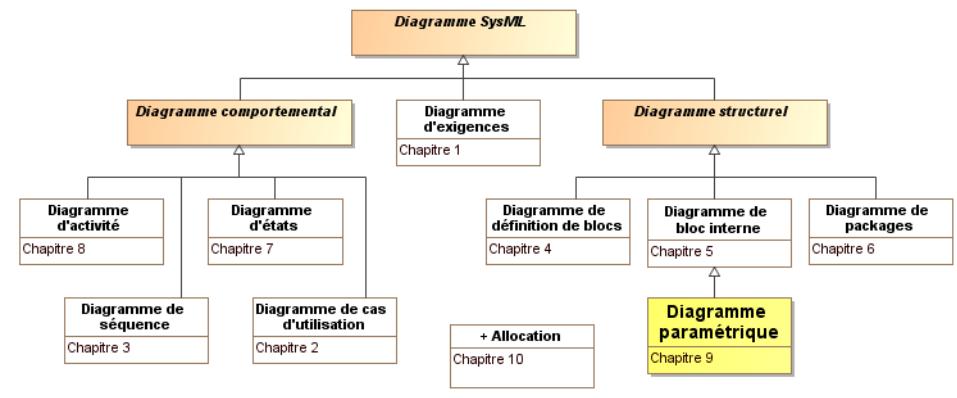
## La modélisation transverse

La partie IV concerne la modélisation transverse. SysML permet de décrire plusieurs types de liens de traçabilité entre éléments de modélisation, et en particulier de mettre en œuvre le concept fondamental d'allocation. Nous verrons également comment décrire des équations grâce au nouveau diagramme paramétrique.

# Le diagramme paramétrique

Ce chapitre présente le diagramme paramétrique. Le diagramme paramétrique permet de représenter des contraintes sur les valeurs de paramètres système tels que performance, fiabilité, masse, etc. Il s'agit d'une spécialisation du diagramme de bloc interne où les seuls blocs utilisables sont des contraintes entre paramètres permettant de représenter graphiquement des équations et des relations mathématiques. Ce nouveau diagramme fournit ainsi un support précieux pour les études d'analyse système.

- ▶ **contrainte**
- ▶ **équation**
- ▶ **value binding**
- ▶ **exigence**
- ▶ **verifiedBy**



## Notation de base

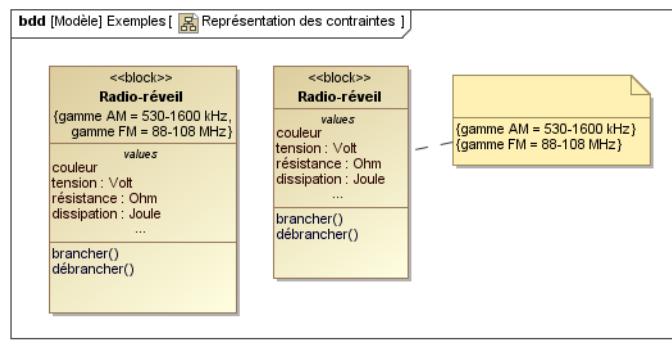
Le diagramme paramétrique est une technique de modélisation spécifique à SysML (par rapport à UML 2) permettant d'intégrer dans le modèle des représentations de contraintes ou d'équations à des fins d'analyse.

Chaque contrainte est d'abord définie par des paramètres ainsi qu'une règle décrivant l'évolution des paramètres les uns par rapport aux autres. Une contrainte est représentée par un bloc avec un stéréotype « `constraint` ». Il faut donc déclarer les contraintes dans un bdd, comme pour les blocs plus classiques. Un exemple donnant la déclaration de deux lois électriques, la loi d'Ohm et l'effet Joule, est montré sur la figure suivante. Notez que les contraintes apparaissent également dans le bloc englobant Radio-réveil si nous le souhaitons.

## B.A.-BA Contrainte (rappel)

SysML inclut un mécanisme général d'expression de contraintes dans tout type de diagramme sous forme d'une simple chaîne de caractères entre accolades. Il n'impose pas de langage particulier, car le modélisateur peut avoir envie d'utiliser celui qui est le plus pertinent dans son contexte.

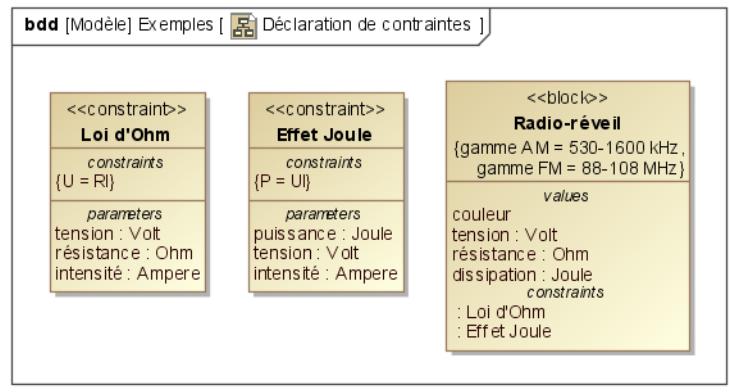
Une contrainte peut ainsi être montrée soit dans un compartiment spécifique appelé constraints, soit dans le comportement du nom de l'élément, soit sous forme d'une note attachée incluant le texte de la contrainte.



**Figure 9-1** Représentations graphiques des contraintes dans les diagrammes SysML

Les contraintes s'appuient ensuite sur le diagramme paramétrique, qui est une spécialisation du diagramme de bloc interne, pour permettre leur composition et leur mise en relation. Les *constraint properties* sont représentées différemment des *part properties* de l'ibd : ce sont des rectangles aux coins arrondis. Du coup, le mot-clé « constraint » est optionnel.

**Figure 9–2**  
Définition de contraintes  
dans un bdd



Les paramètres formels des équations sont représentés par des ports et peuvent ainsi être connectés les uns aux autres. Ils sont déclarés sous la forme habituelle : **nom du paramètre : Type [multiplicité]**.

#### B.A.-BA Cartouche

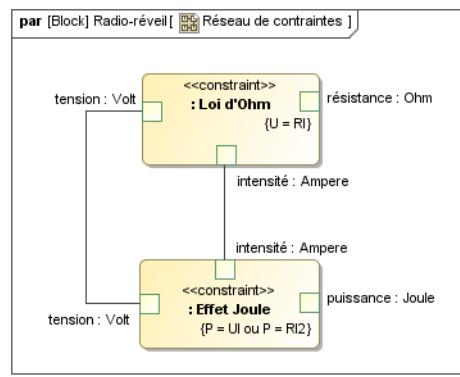
Le cartouche général du diagramme paramétrique est de la forme :  
**par [type d'élément] nom de l'élément [nom du diagramme]**  
 le type d'élément peut être un bloc ou une contrainte.

Montrons la connexion des deux équations précédentes via leurs paramètres communs sur le diagramme suivant. Les deux éléments de type *constraint property* représentent chacun un usage des deux contraintes déclarées précédemment. Les paramètres de type Volt et

Ampere sont connectés entre eux. Notez qu'ils sont graphiquement complètement à l'intérieur de la contrainte, à la différence des ports classiques dans l'ibd. Ils pourraient avoir des noms différents, ce qui pourrait être le cas si les équations avaient été définies dans des contextes différents. Il suffit que les types soient compatibles pour pouvoir les relier entre eux.

**Figure 9–3**

Système d'équations dans un diagramme paramétrique



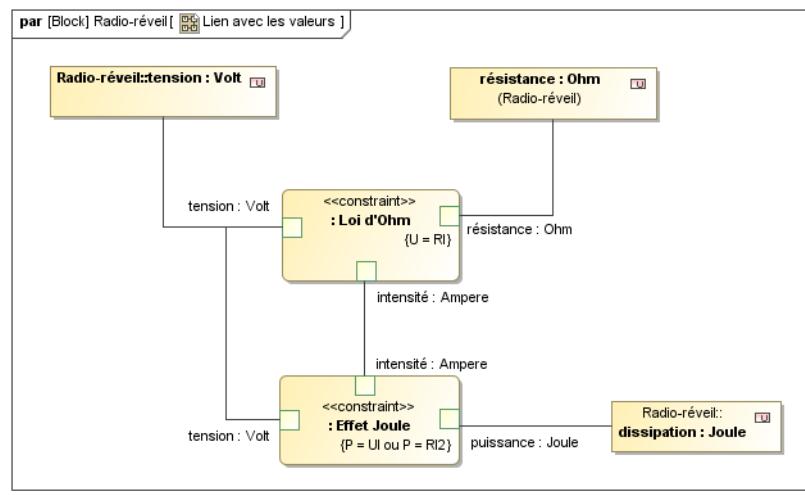
**ATTENTION Diagramme structurel !**

Le diagramme paramétrique est un diagramme structurel ! Quand on exprime la loi d'Ohm sous la forme  $U = RI$ , cela ne signifie pas que l'on calcule  $U$  à partir des valeurs  $R$  et  $I$ . Il n'y a pas de notion d'entrée/sortie. On aurait d'ailleurs pu écrire  $R = U/I$  ou  $I = U/R$ . Voilà pourquoi le diagramme paramétrique n'est pas une spécialisation du diagramme d'activité, mais d'un diagramme structurel : le diagramme interne de bloc.

# Compléments

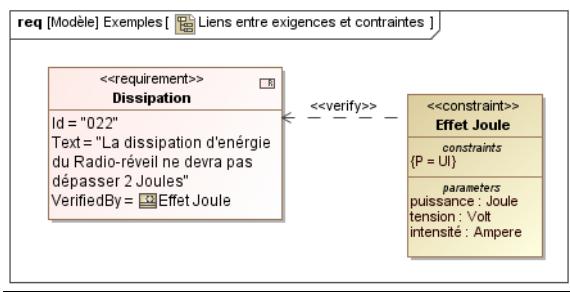
La contrainte s'instancie ensuite en reliant des valeurs (appartenant aux blocs du modèle) aux paramètres formels : c'est la notion de *value binding*.

**Figure 9–4**  
Diagramme paramétrique  
avec value binding



Sur le diagramme précédent, on voit apparaître trois *value properties* : tension, résistance et dissipation, qui ont été déclarées dans le bloc Radio-réveil au niveau du bdd. Pour préciser à quel bloc appartiennent ces valeurs, remarquez la notation donnant le chemin complet, par exemple : [Radio-réveil::tension](#). Une notation pointée est également possible ([Radio-réveil.tension](#)), ainsi que l'affichage du nom du possesseur entre parenthèses au-dessous.

**Figure 9–5**  
Lien entre exigences  
et contraintes

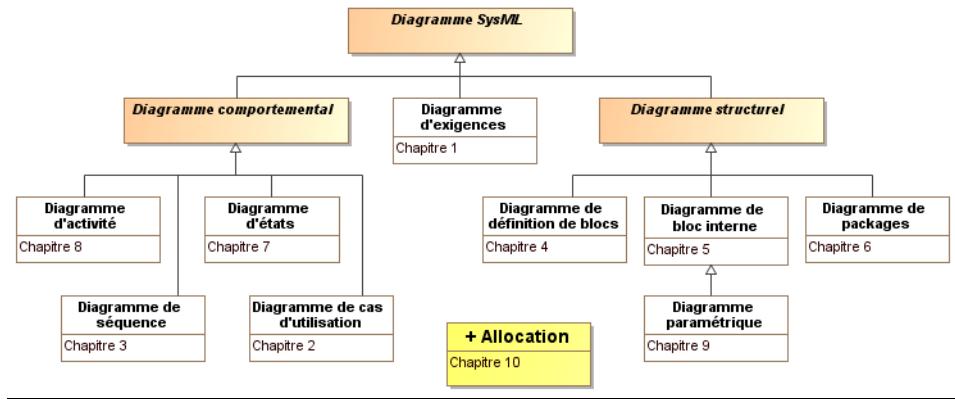


L'intérêt du diagramme paramétrique tient des capacités que commencent à offrir les éditeurs SysML d'injecter des données dans d'autres outils d'analyse. Citons d'ores et déjà le couplage de MagicDraw avec ParaMagic qui permet d'effectuer des simulations à partir des contraintes du diagramme paramétrique, en s'appuyant sur un solveur mathématique appelé Mathematica™. On peut également citer le couplage de l'outil IBM-Telelogic/Rhapsody vers Matlab/Simulink.

# Allocation et traçabilité

Ce chapitre présente le concept d'allocation et ses possibilités de représentation. SysML inclut en effet un mécanisme général permettant de représenter différents types d'allocation, incluant l'allocation de fonctions à des composants, de composants logiques à composants physiques, ainsi que du software au hardware.

- ▶ allocation
- ▶ traçabilité



## Le concept d'allocation

L’allocation est un mécanisme général en ingénierie système pour interconnecter des éléments de différents types. Par exemple, pour relier un flot d’objet dans un diagramme d’activité à un connecteur dans un diagramme de bloc interne (ibd), ou une action à une partie, etc. Ces relations sont souvent présentes dans un modèle classique, mais uniquement sous forme de commentaires ou de notes, donc peu visibles. Le gros apport de SysML avec le mécanisme d’allocation est de rendre visible et exploitable toutes ces interconnexions.

Comme le mécanisme d'allocation est très général, SysML n'a pas ajouté un diagramme spécifique, mais plutôt un mécanisme avec plusieurs représentations graphiques disponible dans de nombreux diagrammes.

### B.A.-BA Allocation

Une allocation est une relation entre éléments de types différents, ou de niveaux d'abstraction différents. Elle permet d'allouer un élément cible à un ou plusieurs éléments sources. Elle peut se traduire graphiquement par une flèche pointillée avec le mot-clé « `allocate` », un compartiment spécifique avec les mots-clés `allocatedFrom`, `allocatedTo`, une note, etc.

Nous avons d'ailleurs déjà vu une sorte d'allocation dans le chapitre 2 avec le diagramme d'exigences : la relation « `satisfy` » peut tout à fait être vue comme l'allocation des exigences aux blocs.

## Les différentes représentations

### Diagramme de définition de blocs (bdd)

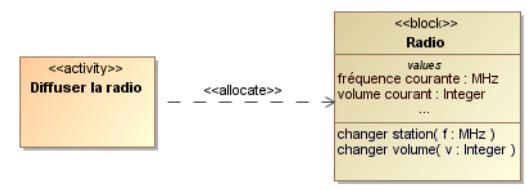
Le diagramme de définition de blocs permet de décrire des allocations entre des éléments représentés graphiquement comme des rectangles : principalement des blocs, mais aussi des activités, etc.

La première représentation graphique est donnée par une flèche pointillée avec le mot-clé « `allocate` ».

Une seconde façon consiste à montrer les relations dans un compartiment supplémentaire. L'élément pointé par la relation « `allocate` » possède une propriété `allocatedFrom`, l'élément cible possède une propriété `allocatedTo`.

**Figure 10–1**

Allocation d'une activité à un bloc dans un bdd



Notez que le bloc radio possède également une propriété *Satisfies*, depuis le chapitre sur la gestion des exigences.

**Figure 10–2**

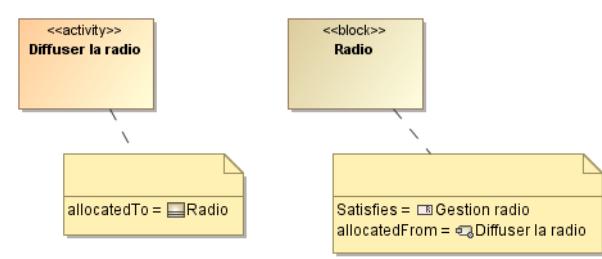
Allocations dans des compartiments



Une troisième façon de représenter ces propriétés consiste à les faire figurer dans une note attachée.

**Figure 10–3**

Allocations dans des notes



## Diagramme d'activité

Le diagramme d'activité peut être augmenté en représentant l'allocation des actions sur les blocs, grâce au concept de partition (ou *swimlane*). Si nous reprenons le diagramme d'activité du chapitre 8 avec la région interruptible et que nous ajoutons les couloirs d'allocation, nous obtenons le diagramme d'activité suivant.

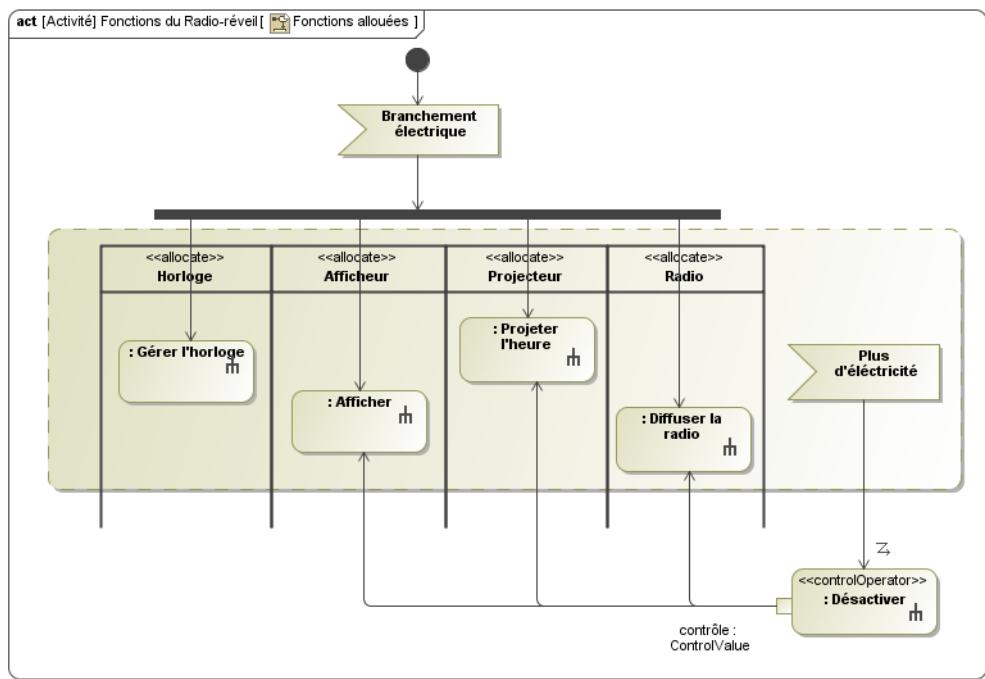


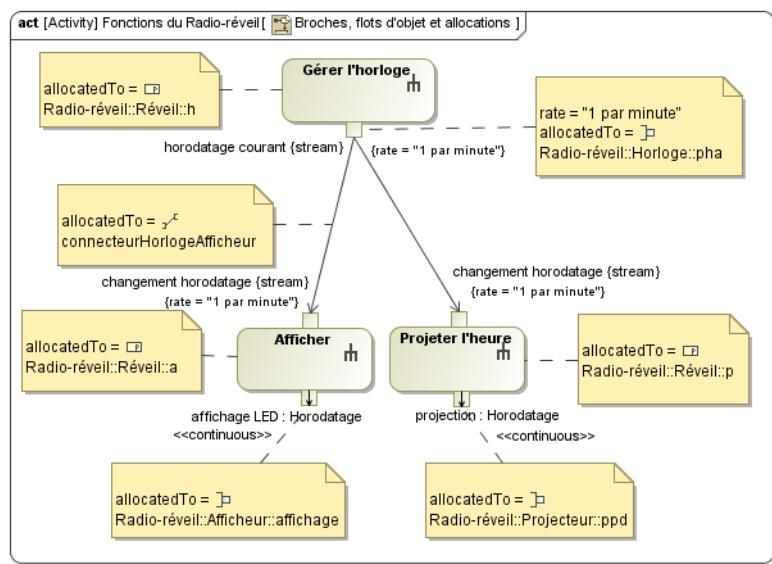
Figure 10–4 Allocations dans des couloirs du diagramme d'activité

Nous pouvons également allouer plus finement les broches d'entrée/sortie, et les flots d'objet du diagramme d'activité réalisé au chapitre 8, représentant les broches et flots d'objet entre les fonctions.

Pour notre étude de cas, nous avons repris le diagramme d'activité réalisé au chapitre 8, représentant les broches et flots d'objet entre les fonctions. Nous lui avons ajouté l'allocation des broches de sortie aux ports des blocs concernés, ainsi que l'allocation du flot d'objet entre Gérer l'horloge et Afficher, et le connecteur structurel entre les parties Horloge et Afficheur du Radio-réveil.

**Figure 10–5**

Allocations de broches et flots d'objets à des parties, ports et connecteurs



## Représentation tabulaire

Comme c'est également le cas pour les exigences, SysML préconise de pouvoir représenter les allocations sous forme tabulaire. Il est à la charge de l'outil de modélisation de fournir ce genre de représentation, en complément des représentations graphiques. La figure suivante est générée automatiquement par MagicDraw à partir des diagrammes précédents.

**Figure 10–6**  
Représentation tabulaire  
des allocations

	Afficheur [Radio-réveil]	Afficher:Afficher ...	Diffuser la radio:Dif...	Gérer l'horloge:G...	Projeter l'heure:P...	Horloge [Radio-réveil]	Projeteur [Radio-réveil]	Radio [Radio-réveil]
Radio-réveil	1	1	1	1	1	1	1	1
Afficheur [Radio-réveil]								
Horloge [Radio-réveil]								
Projecteur [Radio-réveil]								
Radio [Radio-réveil]								
Fonctions du Radio-réveil [Radio-réveil]	1				1	1	1	
Afficher:Afficher [Radio-réveil:Fonc...]								
Diffuser la radio:Diffuser la radi...								
Gérer l'horloge:Gérer l'horloge [Radi...								
Projeter l'heure:Projeter l'heure [Ra...								

En fait, l'outil permet même d'aller plus loin en montrant les allocations de façon hiérarchique entre éléments plus fins (comme ports et broches, par exemple).

	+affichage : Typ...	0	Afficher;Afficher ...	affichage LED : T...	Diffuser la radio:...	Object Flow:flot h...	Gérer l'horloge:G...	horodatage coura...	Projeter l'heure:P...	projection : Type...	+pha : Types co...	+ppd : Types co...	Radio [Radio-réveil]	+radio : Radio-ré...	Connector [Radio-...	Connector [Radio-...	-a-led : Radioréve...	-ha : Radio-réveil...	-hp : Radio-réveil...	-p : Radioréveill...	Connector [Radio-...	Connector [Radio-...	-a : Radioréveil...	Connector [onne...	-h : Radioréveill...	-p : Radioréveill...	
Radio-réveil	1	1	1	2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	2	1	
Radio [Radio-réveil]																											
Projecteur [Radio-réveil]																											
+ppd : Types communs:...																											
Afficheur [Radio-réveil]																											
+affichage : Types com...																											
Réveil à double horloge [Ra...																											
Horloge [Radio-réveil]																											
+pha : Types communs:...																											
Réveil [Radio-réveil]																											
Radio-réveil [Radio-réveil]																											
+radio : Radio-réveil:R...																											
Fonctions du Radio-réveil [...	1																										
Afficher;Afficher [Radio...																											
Diffuser la radio:Diffuse...																											
Object Flow:flot horloge...																											
Gérer l'horloge:Gérer l'h...																											
Projeter l'heure:Projeter...																											
Gérer l'horloge:Gérer l'h...																											
horodatage courant ...																											
Afficher;Afficher [Radio...	1																										
affichage LED : Type...																											
Projeter l'heure:Projeter...																											
projection : Types co...																											

Figure 10–7 Représentation tabulaire détaillée des allocations

## Diagramme de bloc interne (ibd)

Le diagramme de bloc interne permet d'ajouter des allocations d'actions, flots, etc. présents dans le diagramme d'activité. Nous avons vu le mécanisme inverse dans le paragraphe précédent. C'est là que l'on apprécie l'aide d'un outil SysML (par rapport à un pur outil de dessin) qui doit savoir répercuter les allocations déclarées une fois pour toutes sur l'ensemble des diagrammes où figurent des éléments concernés.

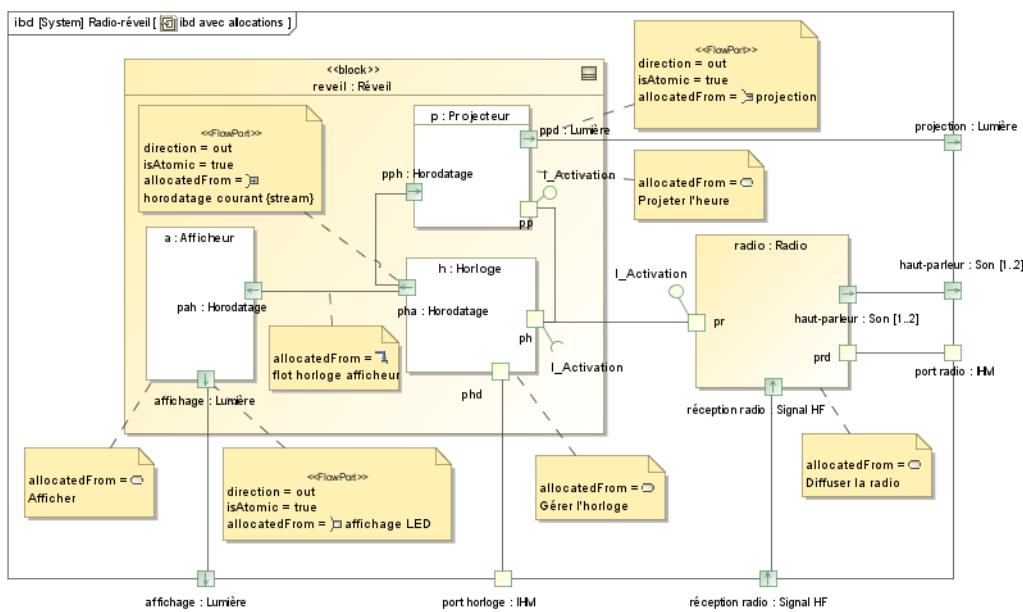
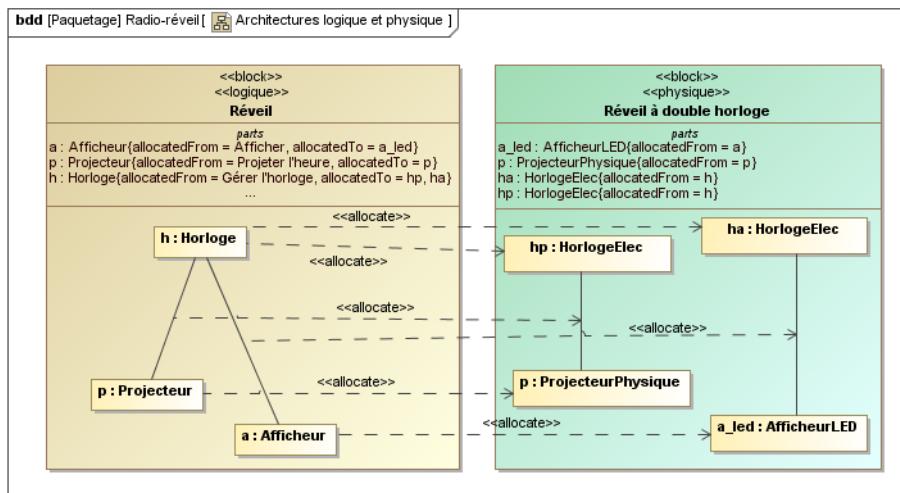


Figure 10–8 Allocations dans un ibd

**Figure 10–9**  
Allocations  
structurelles  
entre niveaux  
logique et  
physique



Dans notre étude de cas, nous avons évoqué la possibilité au niveau de l'implémentation d'avoir deux horloges séparées pour l'afficheur et le projecteur, ou une horloge unique garantissant la cohérence des affichages. Au niveau logique, il n'y a qu'une fonctionnalité

de gestion d'horloge. On pourrait donc faire une première description structurelle dans ce sens, puis explorer plusieurs architectures physiques candidates afin de réaliser une analyse comparative (*trade-off analysis*).

Sur le diagramme précédent, nous avons fait figurer côté à côté une vue logique simplifiée de la structure du réveil, et une vue physique simplifiée de ce même réveil. Notez les stéréotypes « *logique* » et « *physique* » que nous avons créés, puis ajoutés aux deux blocs, afin de mettre encore plus en évidence le point de vue adopté pour chacun. Notez aussi que SysML permet de dessiner une vue simplifiée de la structure interne d'un bloc dans un compartiment particulier dès le bdd. Cela nous facilite ainsi la vie pour déclarer les relations d'allocation qui sont de véritables projections du monde logique vers le monde physique.

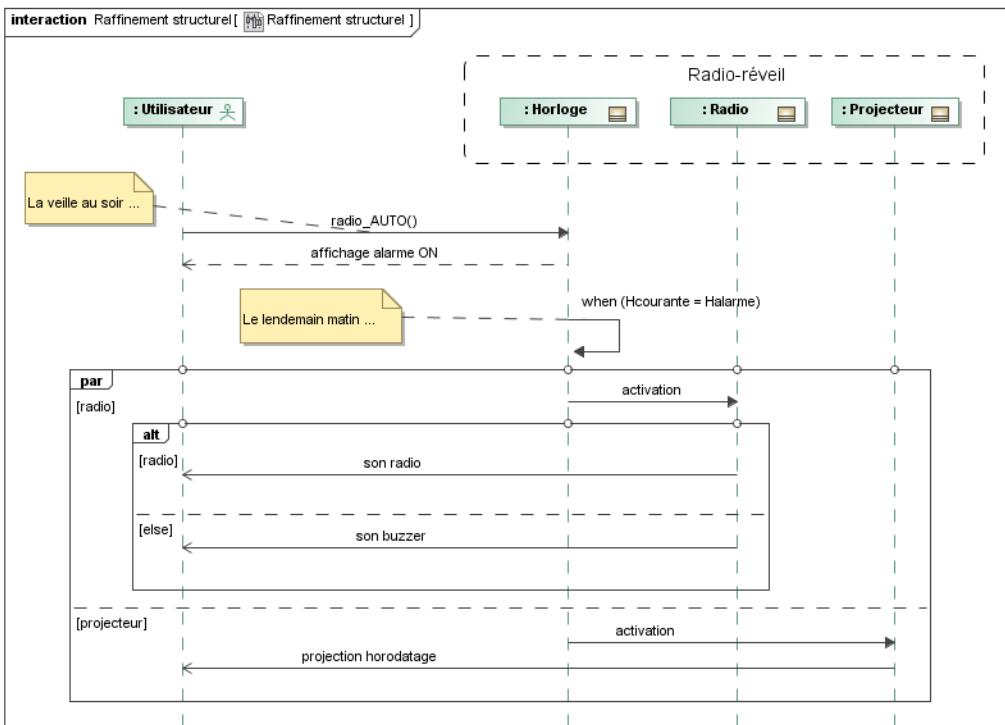
## Diagramme de séquence

Dans le chapitre 3, nous avons montré comment le diagramme de séquence système permet de décrire des interactions « boîte noire » entre le système à l'étude et ses acteurs. Cela peut d'ailleurs servir assez directement à dériver des scénarios de test fonctionnel.

Maintenant que nous avons identifié des parties à l'intérieur du système et précisé leurs connexions structurelles, nous allons pouvoir décomposer la ligne de vie représentant le système pour montrer les interactions internes (scénario « boîte-blanche »). On peut se servir de ce niveau pour en dériver des scénarios de test d'intégration entre les différents blocs qui vont constituer le système, et ceci de façon récursive.

Sur le diagramme de séquence suivant, j'ai repris le scénario nominal simplifié du cas d'utilisation principal : être réveillé à l'heure. À la place de la ligne de vie représentant le système boîte noire (Radio-réveil), j'ai fait figurer les trois parties importantes qui participent au scénario : l'horloge, la radio et le projecteur.

Le scénario boîte-blanche précise maintenant que c'est l'horloge qui joue le rôle de contrôleur et qu'elle active en parallèle la radio et le projecteur. Au passage, en creusant à ce niveau, on s'aperçoit qu'on avait oublié de faire figurer la projection sur le scénario boîte noire. C'est cela aussi la modélisation itérative et incrémentale !



**Figure 10-10** Décomposition structurelle dans le diagramme de séquence

# Acronymes et définitions

A

Cette annexe rappelle les différents acronymes utilisés dans le livre ainsi que les définitions les plus importantes.

# Acronymes

<b>AFIS</b>	Association française d'ingénierie système
<b>bdd</b>	block definition diagram
<b>CFD</b>	Control Flow Diagram
<b>DFD</b>	Data Flow Diagram
<b>dss</b>	diagramme de séquence système
<b>ibd</b>	internal block diagram
<b>INCOSE</b>	International Council on Systems Engineering
<b>IS</b>	Ingénierie système
<b>MARTE</b>	Modeling and Analysis of Real Time and Embedded systems
<b>OMG</b>	Object Management Group
<b>SADT</b>	Structured Analysis and Design Technic
<b>SA/RT</b>	Structured Analysis with Real Time extensions
<b>SOC</b>	System On Chip
<b>SysML</b>	Systems Modeling Language
<b>UC</b>	Use Case
<b>UML</b>	Unified Modeling Language

# Définitions

## 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.

## Acteur généralisé

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

## Action

L'action est l'unité fondamentale de spécification comportementale en SysML. Elle représente un traitement ou une transformation. Les actions sont contenues dans les activités, qui fournissent leur contexte.

## Activation

Les bandes verticales le long d'une ligne de vie d'un diagramme de séquence représentent des périodes d'activation. Elles sont optionnelles, mais permettent de mieux comprendre la flèche pointillée du message de retour. Toutefois, dans un souci de simplicité, nous ne l'utiliserons généralement pas.

## Association

Une association représente une relation sémantique durable entre deux blocs.

Exemple : Une personne peut posséder des voitures. La relation « possède » est une association entre les blocs Personne et Voiture. Attention : même si le verbe qui nomme une association semble privilégier un sens de lecture, une association entre blocs est par défaut bidirectionnelle. Donc implicitement, l'exemple précédent inclut également le fait qu'une voiture est possédée par une personne.

Les compositions et les agrégations sont des cas particuliers d'association.

## Bloc abstrait

Un bloc est dit abstrait si sa définition ne permet pas de l'instancier.

On se sert souvent de blocs abstraits dans les arbres de généralisation pour factoriser des propriétés structurelles ou comportementales communes à d'autres blocs concrets (instanciables).

Un bloc abstrait est représenté en italique.

## Cas d'utilisation

Un cas d'utilisation (use case) 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.

### Condition

Une condition (ou condition de garde) est une expression booléenne qui doit être vraie lorsque l'événement 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.

### Contrainte

Une contrainte est simplement une condition portant sur un ou plusieurs éléments du modèle qui doit être vérifiée par les éléments concernés. Elle est notée entre accolades { }, et peut être insérée au besoin dans une note graphique (le post-it).

### Décision

Une décision est un noeud de contrôle structuré représentant un choix dynamique entre plusieurs conditions qui doivent être mutuellement exclusives. Elle est représentée par un losange qui possède un arc entrant et plusieurs arcs sortants.

### Effet, action, activité

Une transition peut spécifier un comportement optionnel réalisé par le bloc lorsque la transition est déclenchée. Ce comportement est appelé « effet » : cela peut être une simple action ou une séquence d'actions. Une action peut représenter la mise à jour d'une valeur, un appel d'opération, ainsi que l'envoi d'un signal à un autre bloc. L'exécution de l'effet est unitaire et ne permet de traiter aucun événement supplémentaire pendant son déroulement. Les activités durables (*do-activity*) ont une certaine durée, sont interruptibles et sont associées aux états.

## État

Un état représente une situation durant la vie d'un bloc pendant laquelle :

- il satisfait une certaine condition ;
- 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.

## État composite

Un état composite (aussi appelé super-état) 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, tout en spécifiant des transitions particulières entre les sous-états.

## Événement

Spécification d'une occurrence remarquable qui a une localisation dans le temps et l'espace. Un événement peut porter des paramètres qui matérialisent le flot d'information ou de données entre éléments.

## 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.

### Flot

Un flot est un contrôle de séquençage pendant l'exécution de nœuds d'activité. Les flots de contrôle sont de simples flèches reliant deux nœuds (actions, décisions, etc.). Le diagramme d'activité permet également d'utiliser des flots d'objets (reliant une action et un objet consommé ou produit).

### Fork

Un fork est un nœud de contrôle structuré représentant un débranchement parallèle. Il est représentée 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.

### Interface

Une interface est un ensemble d'opérations abstraites (sans algorithmes) constituant une sorte de contrat qui devra être réalisé par un ou plusieurs blocs. Graphiquement, une interface est soit représentée comme un bloc avec un mot-clé « `interface` » ou le symbole d'un cercle, soit directement comme un cercle dans la notation condensée.

### Join

Un join est un nœud 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.

## 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.

## 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.  
La flèche pointillée représente un retour. Cela signifie que le message en question est le résultat direct du message précédent. Un message synchrone (émetteur bloqué en attente de réponse) est représenté par une flèche pleine, alors qu'un message asynchrone est représenté par une flèche évidée. La flèche qui boucle (message réflexif) permet de représenter un comportement interne.

## Multiplicité

Une multiplicité est un intervalle avec une borne inférieure et une borne supérieure :

- la borne inférieure peut-être 0 (optionnelle) ou n'importe quel entier positif ;
- la borne supérieure peut être 1, plusieurs (noté \*), ou un entier positif.

La multiplicité est notée entre crochets. Si les bornes sont égales, on n'écrit qu'une valeur et la valeur par défaut en SysML est [1].

Aux deux extrémités d'une association, on doit faire figurer une indication de multiplicité. Elle spécifie sous la forme d'un intervalle le nombre d'instances qui peuvent participer à une relation avec une instance de l'autre bloc dans le cadre d'une association.

## Package

Mécanisme général de regroupement d'éléments tels que blocs, interfaces, mais aussi acteurs, cas d'utilisation, etc. Les packages peuvent être imbriqués dans d'autres packages.

Un package constitue un espace de noms (*namespace*) pour les éléments qu'il contient.

## 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 ».

## Stéréotype

Les mots-clés SysML comme « `include` » et « `extend` » sont notés entre guillemets typographiques. Mais nous pouvons également créer nos propres mots-clés, tels que « `fragment` » (pour indiquer qu'un cas d'utilisation n'est qu'un fragment factorisé par d'autres cas d'utilisation) ou « `secondaire` » (pour indiquer qu'un cas d'utilisation est moins important que les autres). Ces mots-clés inventés par les modélisateurs s'appellent alors des stéréotypes.

## Système

Un système est un ensemble de composants interreliés qui interagissent les uns avec les autres d'une manière organisée pour accomplir une finalité commune (NASA 1995).

Un système est un ensemble intégré d'éléments qui accomplissent un objectif défini (INCOSE 2004).

## 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.

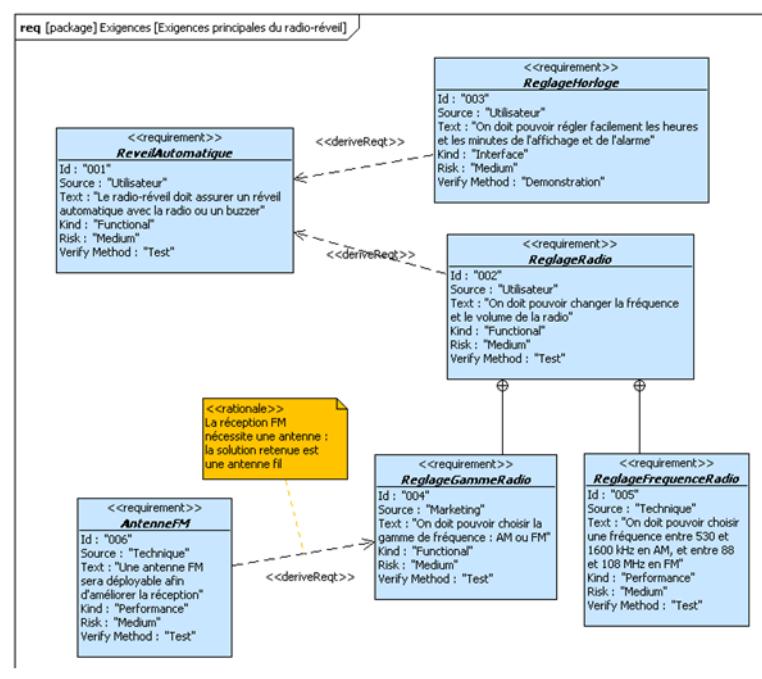
# B

## Diagrammes EA, TopCased et Artisan Studio

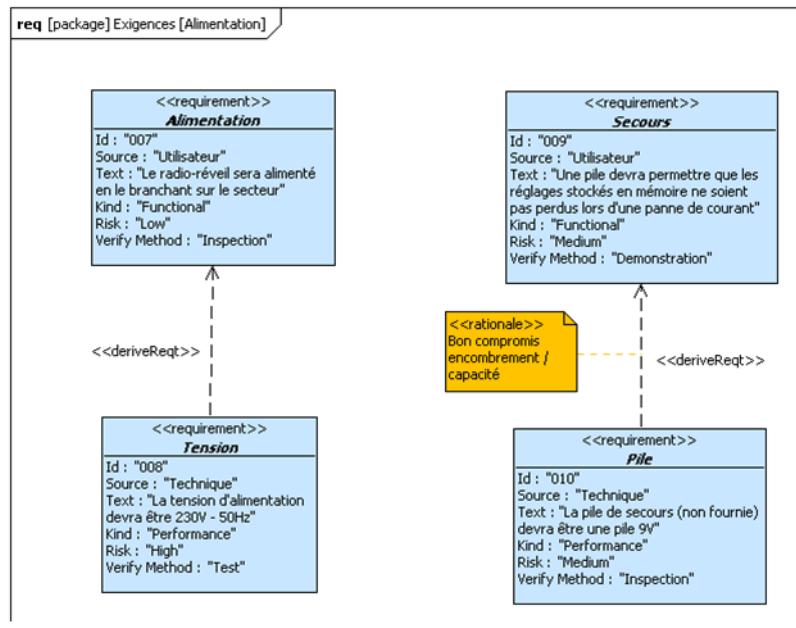
Cette annexe présente d'autres diagrammes réalisés pour l'étude de cas Radio-réveil avec les outils Enterprise Architect, TopCased et Artisan Studio. Ces diagrammes ont été faits indépendamment de ceux réalisés pour le livre avec MagicDraw. Ils ne sont pas toujours cohérents entre eux, mais illustrent les possibilités de modélisation offertes par d'autres outils.

# TopCased

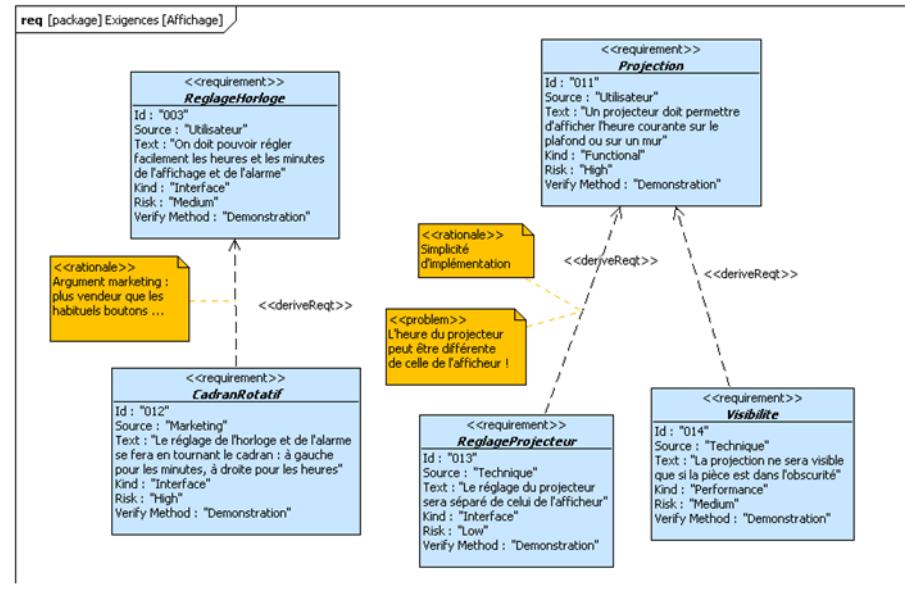
**Figure B-1**  
TopCased : exigences principales

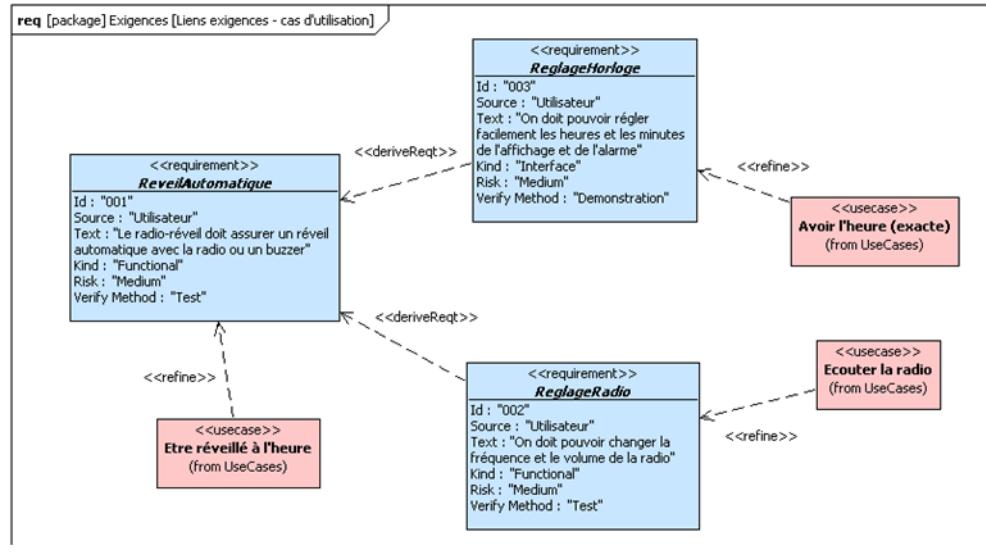


**Figure B–2**  
TopCased : exigences d'alimentation



**Figure B–3**  
TopCased :  
exigences  
d'affichage





**Figure B-4** TopCased : liens entre exigences et cas d'utilisation

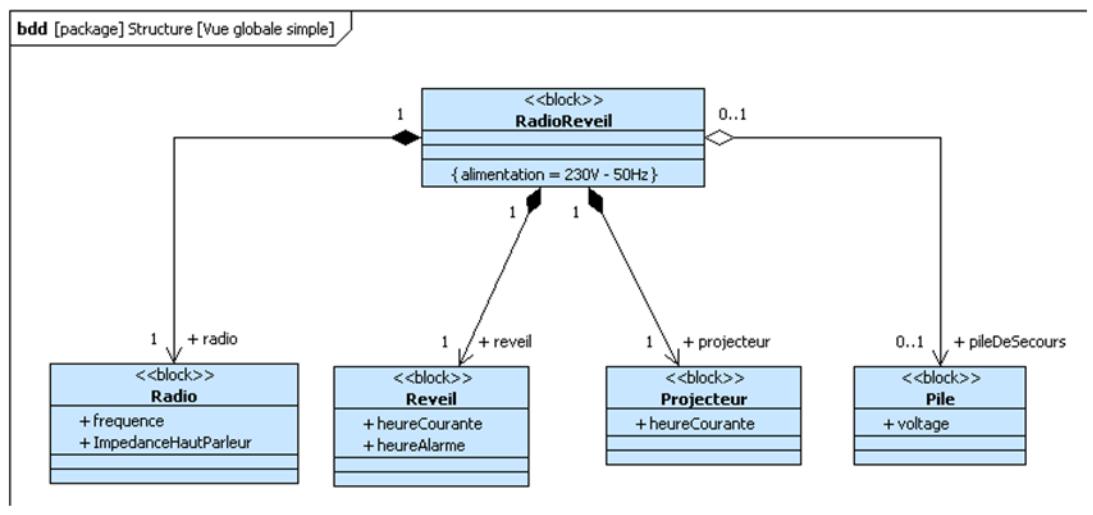
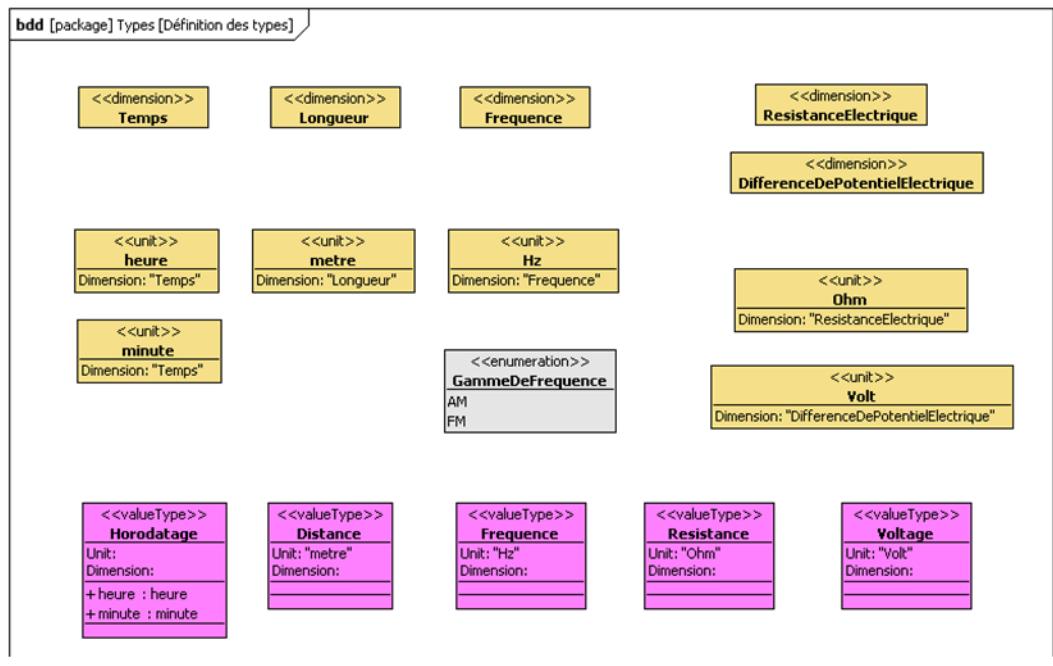


Figure B-5 TopCased : bdd simple



**Figure B–6** TopCased : bdd définissant des Value Types

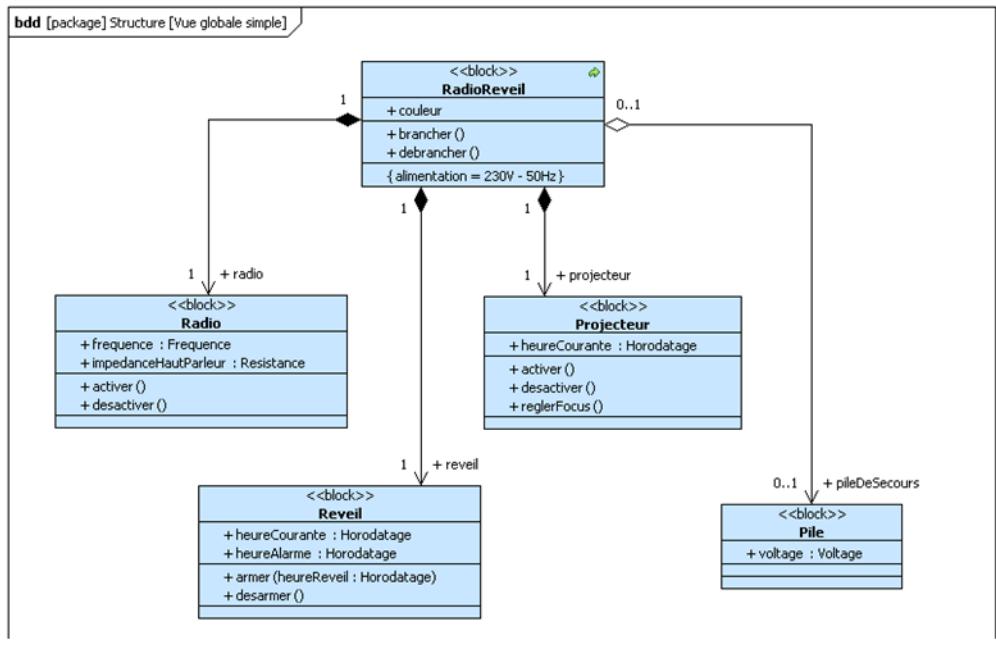


Figure B-7 TopCased : bdd complété

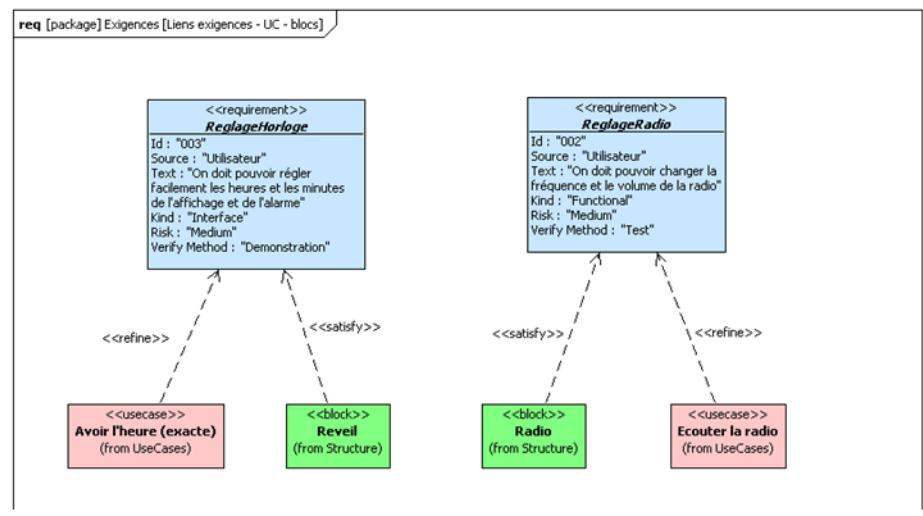
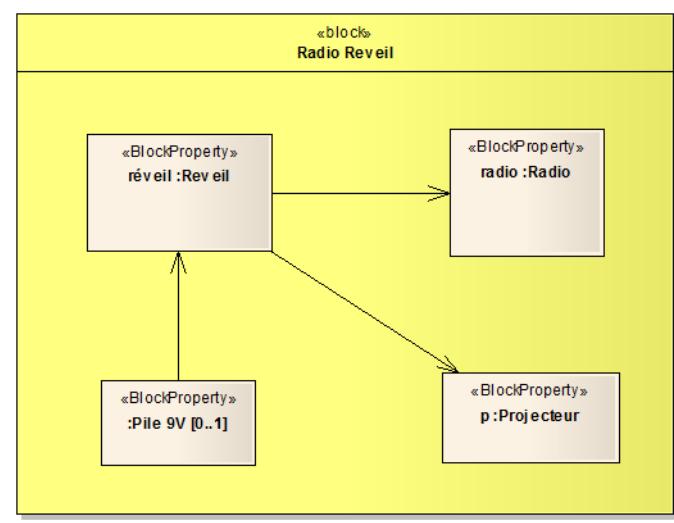


Figure B-8 TopCased : liens entre exigences, cas d'utilisation et blocs

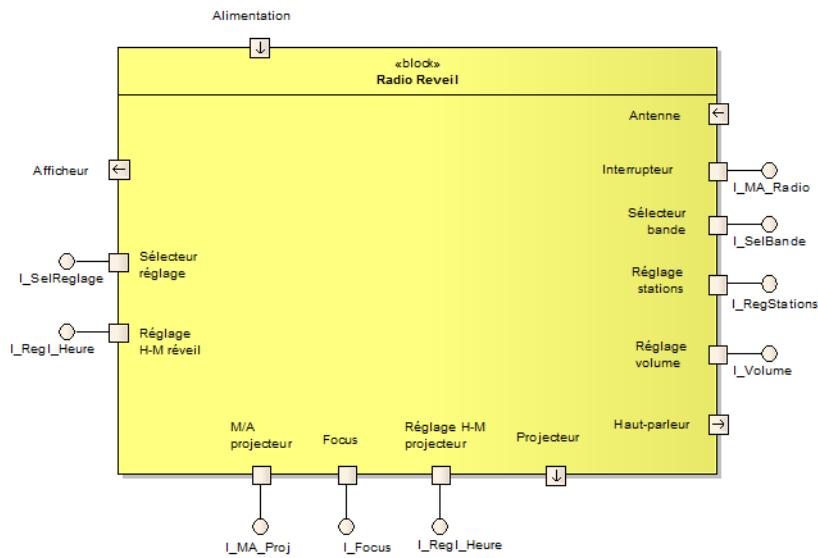
# Enterprise Architect (EA)

**Figure B-9**

EA : début de l'ibd  
du radio-réveil

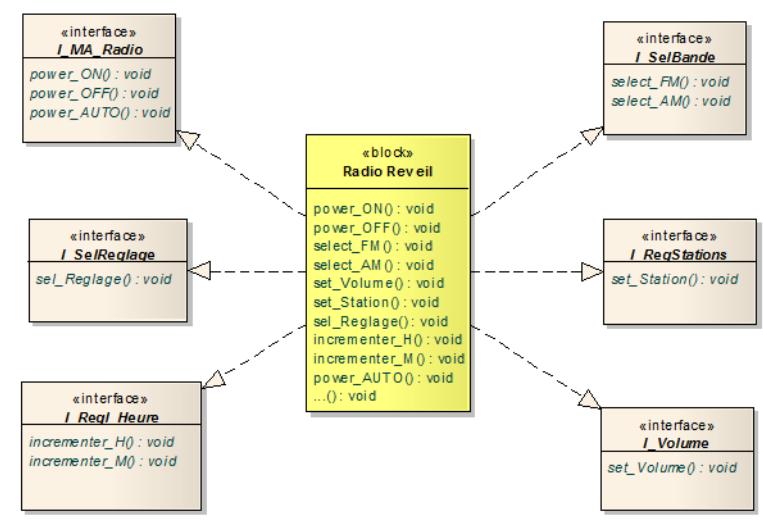


**Figure B-10**  
EA : ports et interfaces  
du radio-réveil



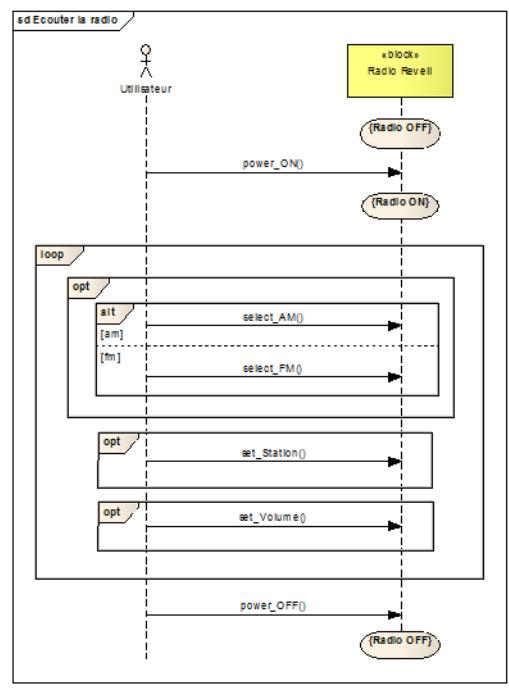
**Figure B-11**

EA : définition des interfaces du radio-réveil



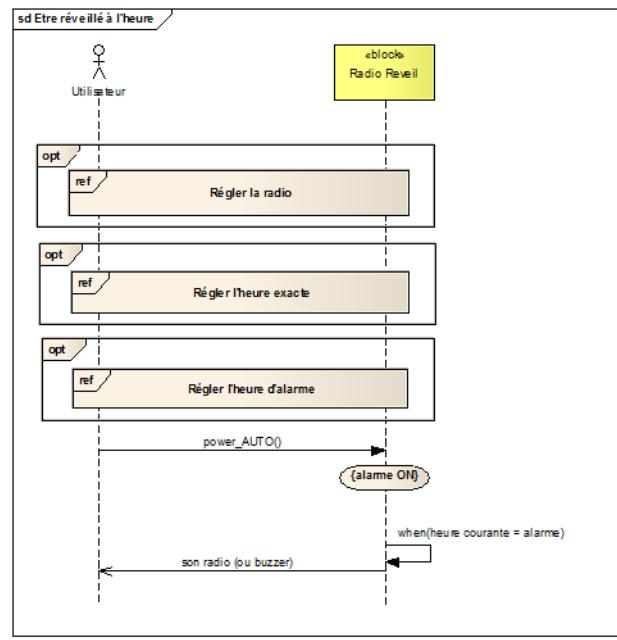
**Figure B-12**

EA : diagramme de séquence de « écouter la radio »



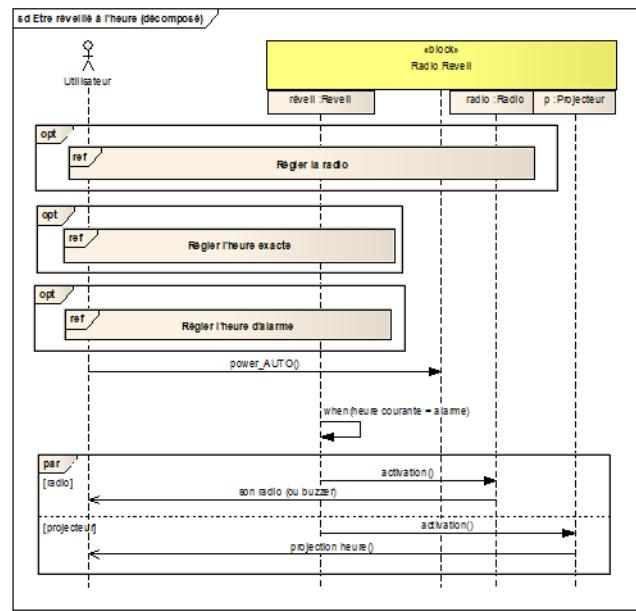
**Figure B-13**

EA : diagramme de séquence de « être réveillé »

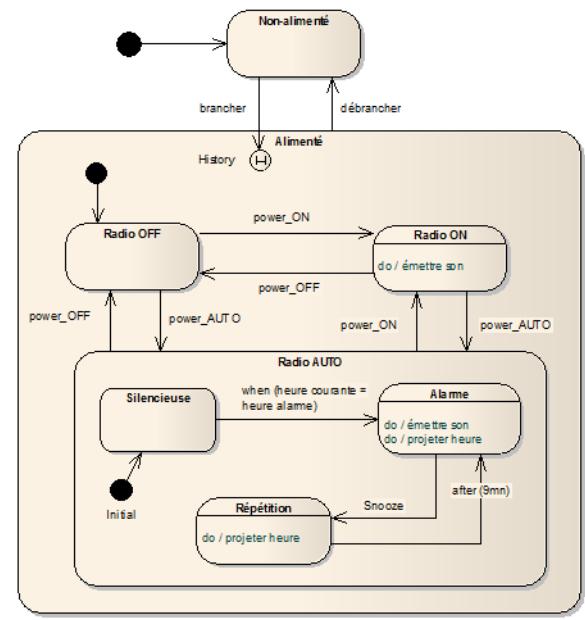


**Figure B-14**

EA : diagramme de séquence décomposé de « être réveillé »



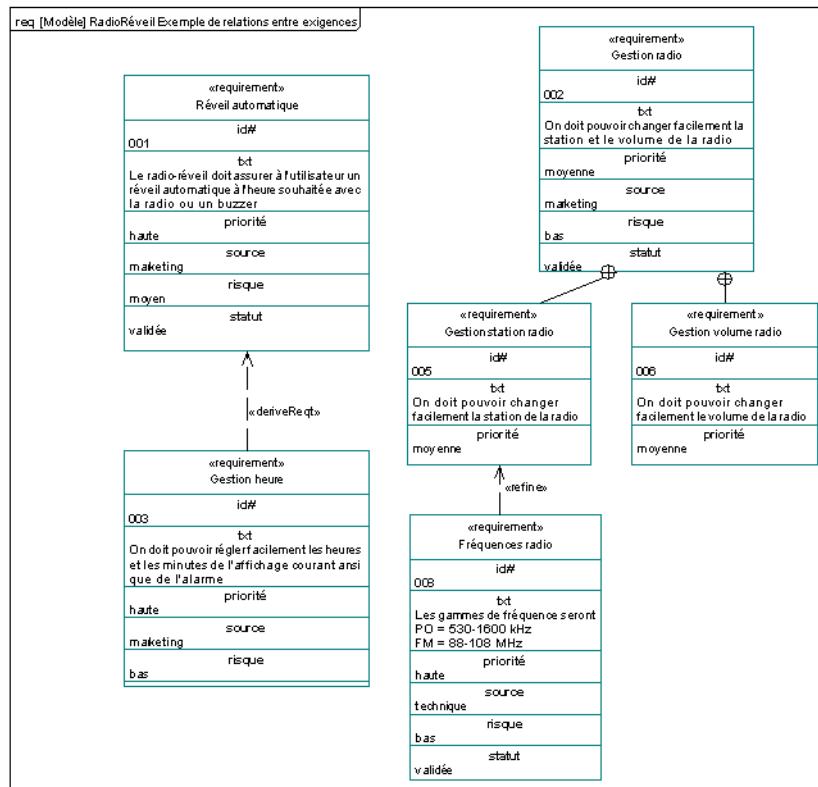
**Figure B-15**  
EA : diagramme d'états



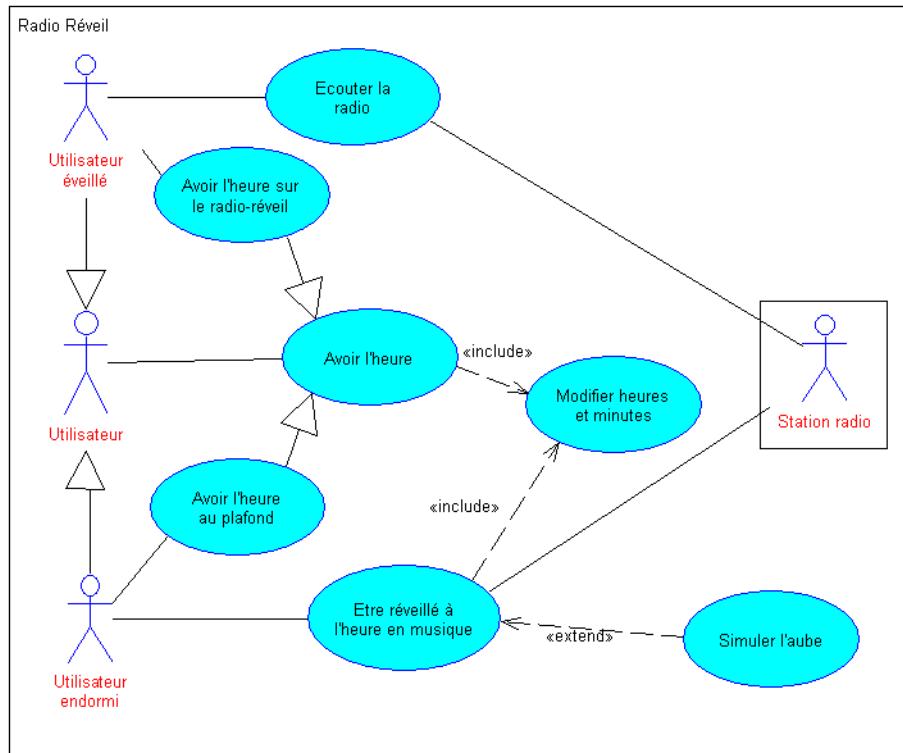
**Artisan Studio (encore merci à Olivier Casse !)**

## **Figure B-16**

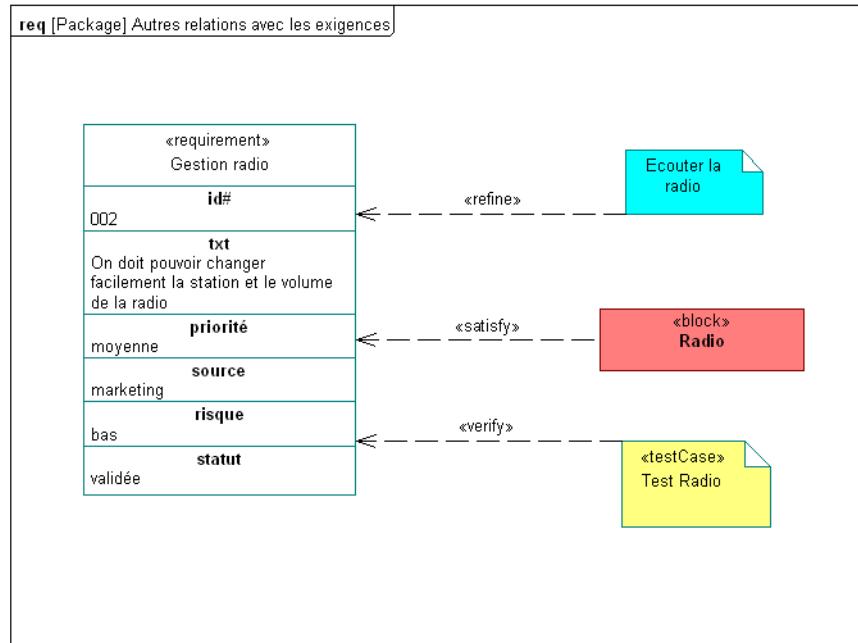
Artisan : relations entre exigences



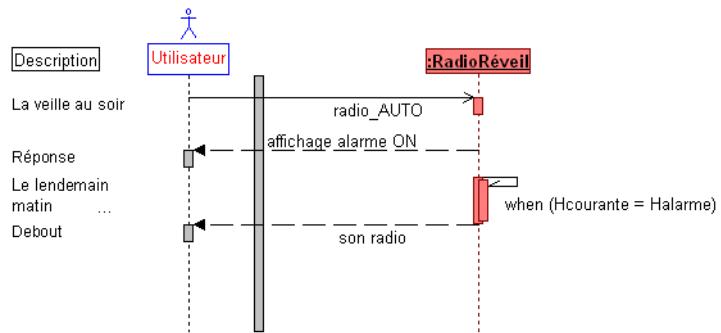
**Figure B-17**  
Artisan :  
cas d'utilisation



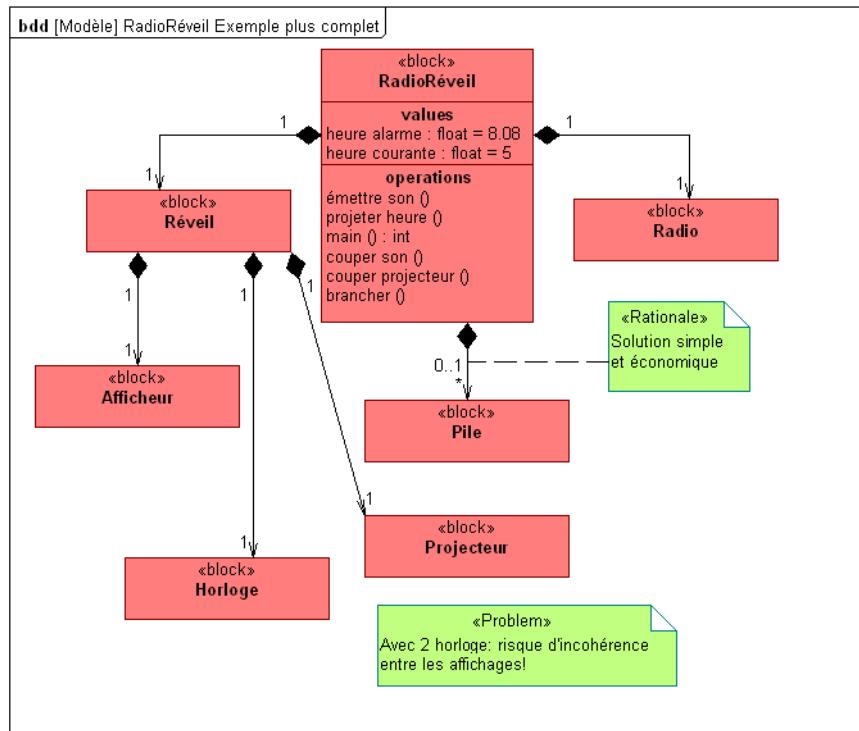
**Figure B-18**  
Artisan : relations entre exigences, cas d'utilisation et cas de test



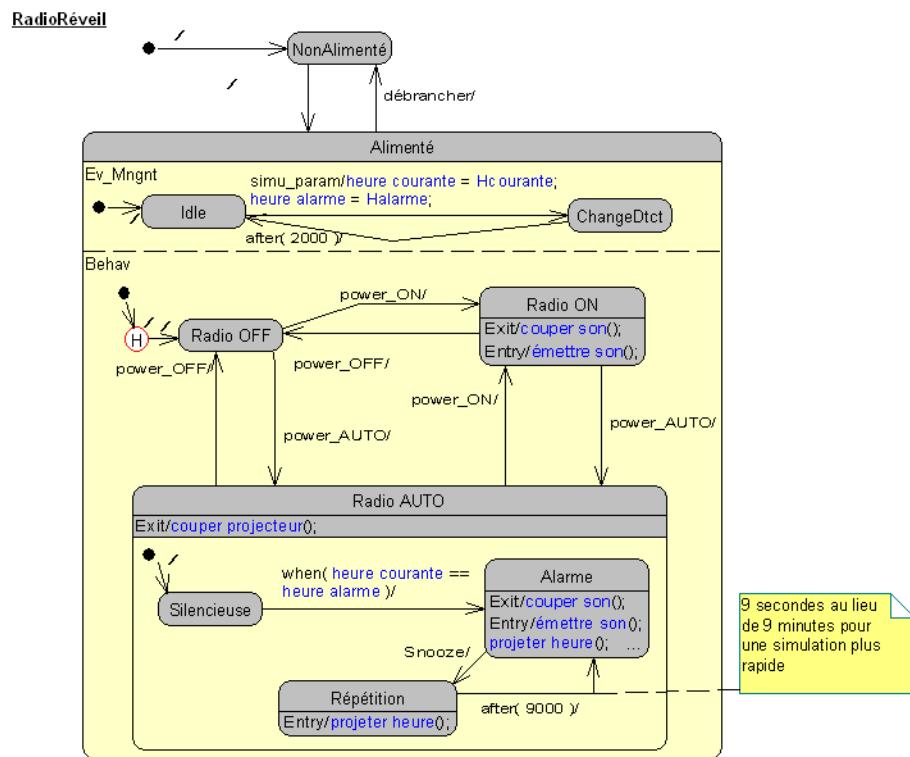
**Figure B-19**  
Artisan : diagramme  
de séquence système



**Figure B-20**  
Artisan :  
bdd complet



**Figure B-21**  
Artisan :  
diagramme  
d'états



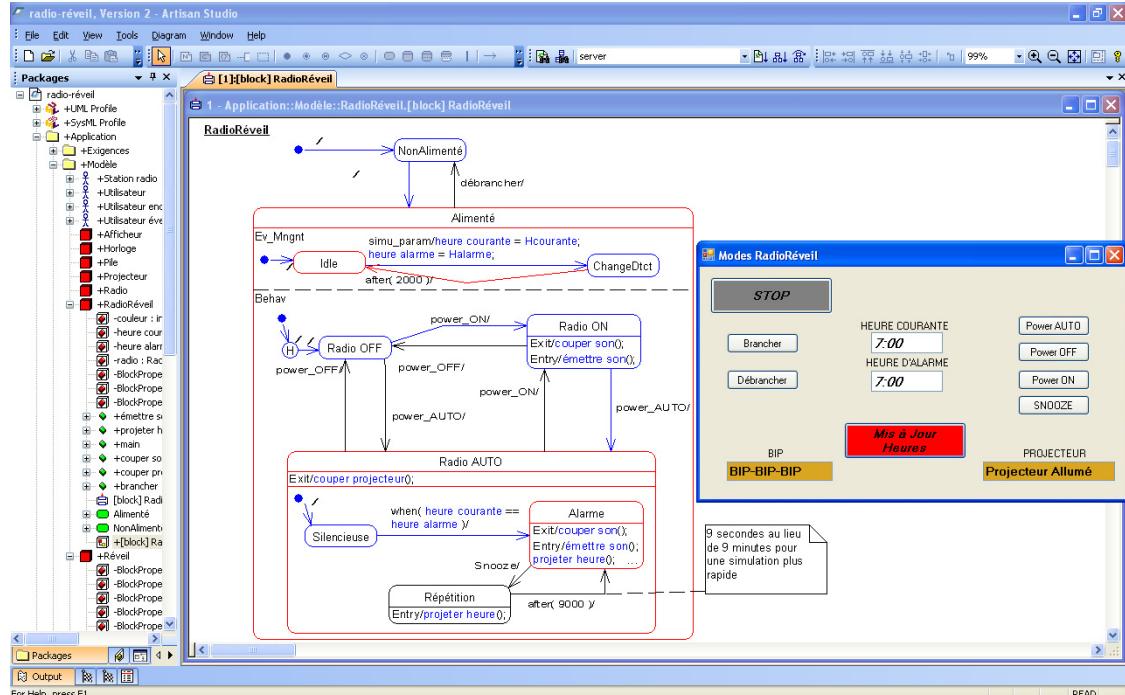


Figure B-22 Artisan : diagramme d'états animé

**Figure B-23**  
Artisan : diagramme d'activité

