Copyright © 2016 Eyrolles

UML 2.5

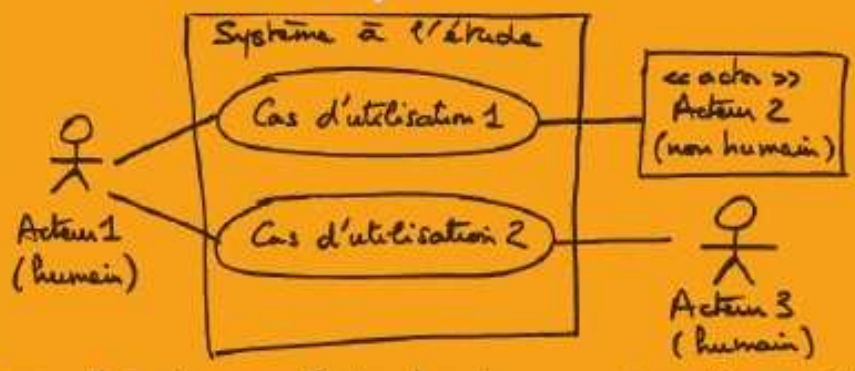
aide-mémento

3e édition

Pascal Roques

Diagramme de cas d'utilisation :

Montre les interactions fonctionnelles entre les acteurs et le système à l'étude



■ 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.

■ Cas d'utilisation (Use case) : ensemble de séquences d'actions réalisées par le système produisant un résultat observable intéressant pour un acteur particulier. Collection de scénarios reliés par un objectif utilisateur commun.

■ Association : utilisée dans ce type de diagramme pour relier les acteurs et les cas d'utilisation par une relation qui signifie simplement : « participe à ».

Trucs et astuces

■ Un cas d'utilisation peut faire participer plusieurs acteurs ; un acteur peut participer à plusieurs cas d'utilisation.

■ Appliquez la recommandation suivante : un seul acteur principal par cas d'utilisation. 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.

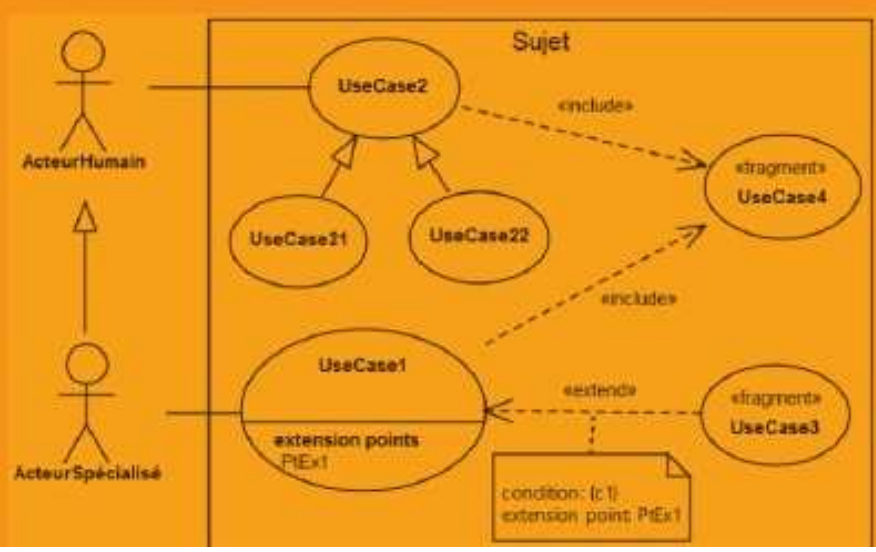
■ Dans la mesure du possible, disposez les acteurs principaux à gauche des cas d'utilisation, et les acteurs secondaires à droite.

■ Utilisez la forme graphique du stick man pour les acteurs humains, et la représentation rectangulaire avec le mot-clé <<actor>> ou un pictogramme pour les systèmes connectés.

■ Éliminez les acteurs « physiques » au profit des acteurs « logiques » : l'acteur principal est celui qui bénéficie de l'utilisation du système.

■ Répertoriez en tant qu'acteurs uniquement les entités externes *et non pas des composants internes* qui interagissent directement avec le système. Un moyen technique n'est pas un acteur pertinent.

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



Copyright © 2016 Eyrolles.

Diagramme de cas d'utilisation

■ Inclusion : le cas d'utilisation de base en incorpore explicitement un autre, de façon obligatoire, à un endroit spécifié dans son comportement.

■ Extension : le cas d'utilisation de base en incorpore implicitement un autre, de façon optionnelle, à un endroit spécifié dans celui qui procède à l'extension.

■ Généralisation : les cas d'utilisation descendants héritent de la description de leur parent commun. Chacun d'entre eux peut néanmoins comprendre des relations supplémentaires avec d'autres acteurs ou cas d'utilisation.

Trucs et astuces

■ Utilisez la relation d'inclusion entre cas d'utilisation pour éviter de devoir décrire plusieurs fois un comportement commun, en le factorisant dans un cas d'utilisation supplémentaire inclus de stéréotype « fragment ».

■ N'abusez pas des relations entre cas d'utilisation *surtout extension et généralisation* : elles peuvent rendre les diagrammes de cas d'utilisation difficiles à décrypter pour les experts métier qui sont censés les valider.

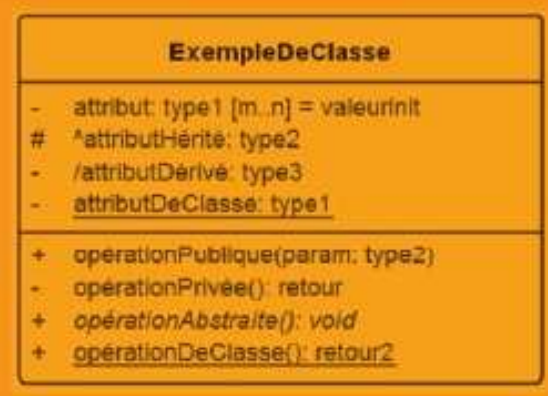
■ Limitez à 20 le nombre de vos cas d'utilisation de base *en dehors des cas inclus, spécialisés, ou des extensions*. Avec cette limite arbitraire, on reste synthétique et on ne tombe pas dans le piège de la granularité trop fine des cas d'utilisation. Un cas d'utilisation ne doit donc pas se réduire à une seule séquence, et encore moins à une simple action.

■ Le diagramme de cas d'utilisation n'est qu'une sorte de table des matières graphique des exigences fonctionnelles. N'y passez pas trop de temps et d'énergie, mais concentrez-vous plutôt sur la description détaillée des cas d'utilisation (texte, diagrammes de séquence, etc.) !

Copyright © 2016 Eyrolles.

Diagramme de classes

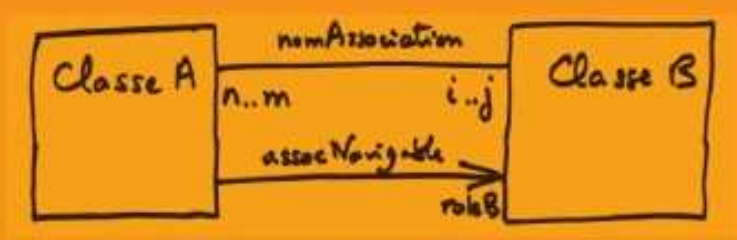
Montre les briques de base statiques : classes, associations, interfaces, attributs, opérations, généralisations, etc.



■ Classe : description abstraite d'un ensemble d'objets qui partagent les mêmes propriétés attributs et associations et comportements opérations et états.

■ Attribut : donnée déclarée au niveau d'une classe, éventuellement typée, et valorisée par chacun des objets de cette classe. Un attribut peut posséder une multiplicité et une valeur initiale.

■ Opération : élément de comportement des objets, défini de manière globale dans leur classe. Une opération peut déclarer des paramètres eux-mêmes typés ainsi qu'un type de retour. Une méthode est une implémentation d'une opération.

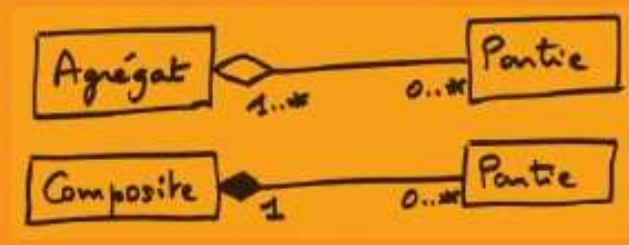


■ Association : relation sémantique durable entre deux classes qui décrit un ensemble de liens entre objets. Une association est décrite par un nom et des rôles, également optionnels. On spécifie sa navigabilité en ajoutant une flèche.

Diagramme de classes

■ **Multiplicité** : le nombre d'objets min..max de la classe cible qui peuvent participer à une relation avec un objet de la classe source dans le cadre d'une association. Multiplicités fréquentes :

- 0..1 = optionnel mais pas multiple
- 1 = exactement 1
- 0..* = * = quelconque
- 1..* = au moins 1



■ **Agrégation** : cas particulier d'association non symétrique exprimant une relation de contenance, mais sans contrainte supplémentaire.

■ **Composition** : forme forte d'agrégation, dans laquelle les parties ne peuvent appartenir à plusieurs agrégats à la fois et où le cycle de vie des parties est subordonné à celui de l'agrégat.

■ **Super-classe** : classe plus générale reliée à une ou plusieurs autres classes plus spécialisées sous-classes par une relation de généralisation.

■ **Généralisation** : relation entre classifieurs où les descendants héritent des propriétés de leur parent commun. Ils peuvent néanmoins comprendre chacun des propriétés spécifiques supplémentaires, mais aussi modifier les comportements hérités.

■ **Note** : commentaire visible dans le diagramme. Peut être attaché à un élément du modèle ou plusieurs par un trait pointillé.

Trucs et astuces

■ En analyse, évitez de confondre attribut et classe : si l'on ne peut demander à un élément que sa valeur, il s'agit d'un simple attribut; si l'on peut lui poser plusieurs questions, il s'agit plutôt d'une classe *qui possédera à son tour des attributs, ainsi que des associations avec d'autres classes.*

■ On doit faire figurer une indication de multiplicité à chaque extrémité navigable d'une association.

■ Un attribut optionnel peut être représenté en ajoutant une multiplicité [0..1] derrière son type.

■ Un attribut dérivé est un attribut dont la valeur peut être déduite d'autres informations disponibles dans le modèle. Conservez cet attribut, qui pourrait être considéré comme redondant, uniquement s'il correspond à un concept important aux yeux de l'expert métier.

■ Une association qui « reboucle » sur une seule classe permet de relier des objets de même type, en distinguant leurs rôles respectifs.

■ Les agrégations et les compositions n'ont pas besoin d'être nommées : elles signifient implicitement « contient », « est composé de ».

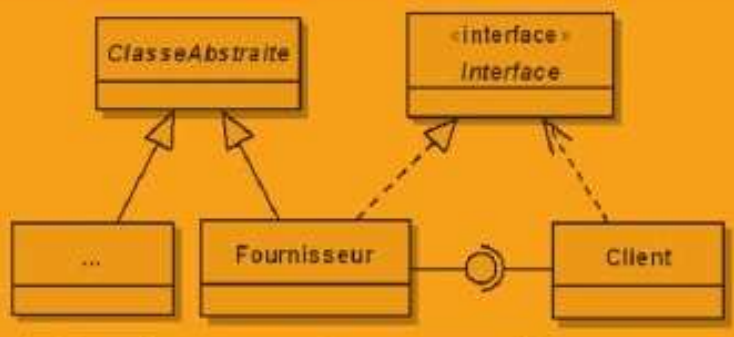
■ Utilisez la composition uniquement si la multiplicité est inférieure ou égale à 1 du côté du composite et si la destruction du composite entraîne celle de ses parties.

■ La relation de généralisation doit pouvoir se lire ainsi : est-une-sorte-de. Il est également fortement recommandé que toutes les propriétés de la superclasse soient valables pour toutes les sous-classes *règle des 100 %*. Les sous-classes peuvent uniquement ajouter des propriétés et redéfinir des opérations.

■ **Interface** : Ensemble nommé d'opérations qui caractérise le comportement

Copyright © 2016 Eyrolles.

Diagramme de classes



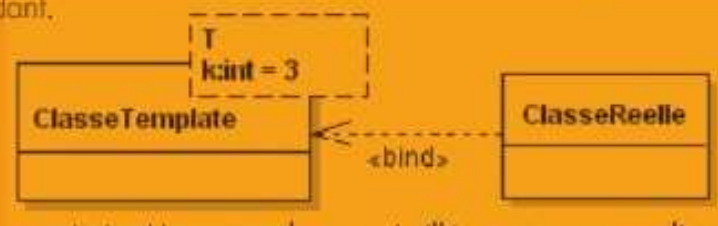
■ **Classe abstraite** : classe qui ne s'instancie pas directement par opposition à une classe concrète. Elle sert en général à factoriser dans une super-classe des propriétés communes à un certain nombre de sous-classes et se note en italique.

■ **Classe d'association** : association promue au rang de classe. Elle possède tout à la fois les caractéristiques d'une association et d'une classe et peut donc porter des attributs qui se valorisent pour chaque lien entre objets.

■ **Qualifieur (ou qualificatif)** : attribut qui permet de « partitionner » l'ensemble des objets en relation avec un objet donné dans le cadre d'une association multiple.

■ **Contrainte** : condition portant sur un ou plusieurs éléments du modèle et qui doit être vérifiée par les éléments concernés. Elle est notée entre accolades { }, et souvent insérée dans une note graphique le post-it.

■ **Dépendance** : relation sémantique entre deux éléments, dans laquelle la modification d'un des éléments l'élément indépendant peut affecter la sémantique de l'autre élément l'élément dépendant.



■ **Template** : Classe paramétrée. Un rectangle en pointillés contenant une liste de paramètres formels est dessiné dans l'angle supérieur droit. Chaque paramètre possède un nom et un type ainsi qu'une valeur par défaut optionnelle.

Trucs et astuces

■ En utilisant des interfaces plutôt que des classes, vous facilitez l'évolution de vos applications. Vous pourrez en effet changer l'implémentation des interfaces plus tard, sans aucun impact pour les classes qui les utilisent.

■ Une classe possédant au moins une opération abstraite *sans implémentation* est forcément abstraite. Attention : une super-classe n'est pas toujours abstraite.

■ Les classes d'association se rencontrent surtout dans le cas de multiplicités * - *. Par contre, il n'y aura qu'une seule instance de la classe d'association par couple d'objets reliés !

■ Attention : l'utilisation d'un qualifieur d'association modifie la multiplicité à l'autre extrémité ! Celle-ci passe fréquemment de 0..* à 0..1.

■ UML propose OCL *Object Constraint Language*, mais autorise tout formalisme pour décrire les contraintes : texte libre, pseudo-code, Java, etc.

■ Un lien durable entre objets va donner lieu à une association navigable entre les classes correspondantes ; un lien temporaire va donner lieu à une relation

Copyright © 2016 Eyrolles.

Diagramme de classes

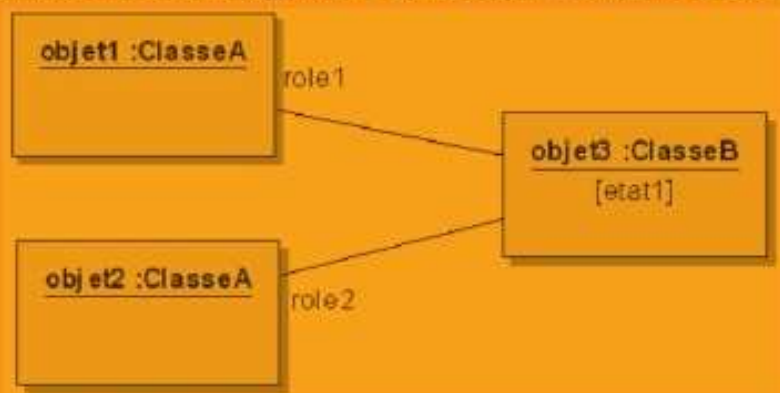
- De nombreuses relations structurelles impliquent une dépendance ! C'est le cas pour l'association navigable : le client dépend du fournisseur. C'est également vrai pour la généralisation : la sous-classe dépend de sa super-classe.
- La règle générale est simple : minimisez les dépendances !
- N'utilisez pas forcément toutes les subtilités du diagramme de classes : pensez à votre lecteur !

Diagramme d'objets

Montre les instances des éléments structurels et leurs liens au run-time

nomObjet : NomClasse

- Objet : entité aux frontières bien définies possédant une identité et encapsulant un état et un comportement ; un objet est une instance ou occurrence d'une classe. Un objet peut avoir connaissance d'un ou plusieurs autres objets : on parle alors de liens entre objets.



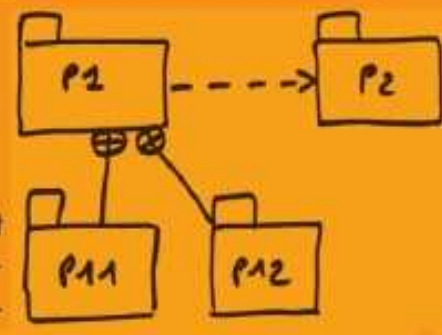
Trucs et astuces

- Le diagramme d'objets est un instantané, une photo, d'un sous-ensemble des objets d'un système à certain moment du temps, avec leurs propriétés et leurs relations. Il peut servir à donner un exemple pour un diagramme de classes complexe, à décrire une configuration initiale, etc.

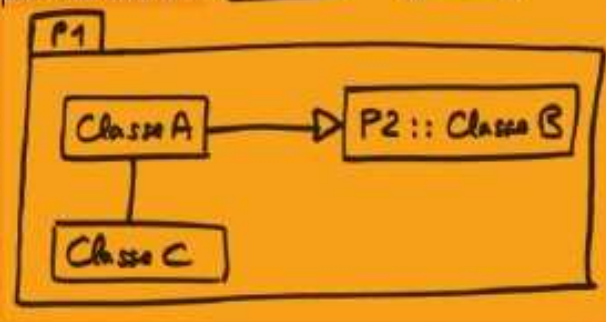
Copyright © 2016 Eyrolles.

Diagramme de packages

Montre l'organisation logique du modèle et les relations entre packages



- Package : mécanisme général de regroupement d'éléments tels que classes, interfaces, mais aussi acteurs, cas d'utilisation, etc. Les packages peuvent être imbriqués dans d'autres packages.
- Importation : relation de dépendance entre packages qui rend visibles les éléments publics d'un package au sein d'un autre package.



Trucs et astuces

- Les packages sont des espaces de noms (ou *namespaces*) : deux éléments ne peuvent pas porter le même nom au sein du même package. Par contre, deux éléments appartenant à des packages différents peuvent porter le même nom. Du coup, UML propose une notation complète pour un élément `nomPackage::nomElément`.

Diagramme de packages

- La structuration d'un modèle en packages doit s'appuyer sur deux principes fondamentaux : forte cohérence à l'intérieur de chaque package et faible couplage entre eux.
- Évitez au maximum les dépendances mutuelles entre packages ainsi que les dépendances circulaires !

Diagramme de profil

Définit un profil spécifique à un domaine ou une problématique par extension du métamodèle UML

- Profil : sorte de package permettant d'étendre un métamodèle de référence, comme celui d'UML. Un profil contient principalement des stéréotypes. L'application d'un ou plusieurs profils sur un package se montre par une relation <<apply>>.
- Stéréotype : définit de nouveaux concepts ou termes spécifiques par extension de métaclasses UML. Peut avoir des propriétés appelées *tags*.
- Extension : la relation d'extension entre stéréotypes et métaclasses UML est représentée graphiquement comme la relation de généralisation, mais avec une flèche noire.

Trucs et astuces

- Ce quatorzième type de diagramme, introduit officiellement par UML 2.3, n'est pas utilisé par tout un chacun, mais uniquement par ceux qui souhaitent définir une variante d'UML pour un domaine spécifique.
- SysML *(Systems Modeling Language)* est un exemple récent de profil UML 2.

Diagramme de séquence

Montre la séquence verticale des messages passés entre éléments au sein d'une interaction

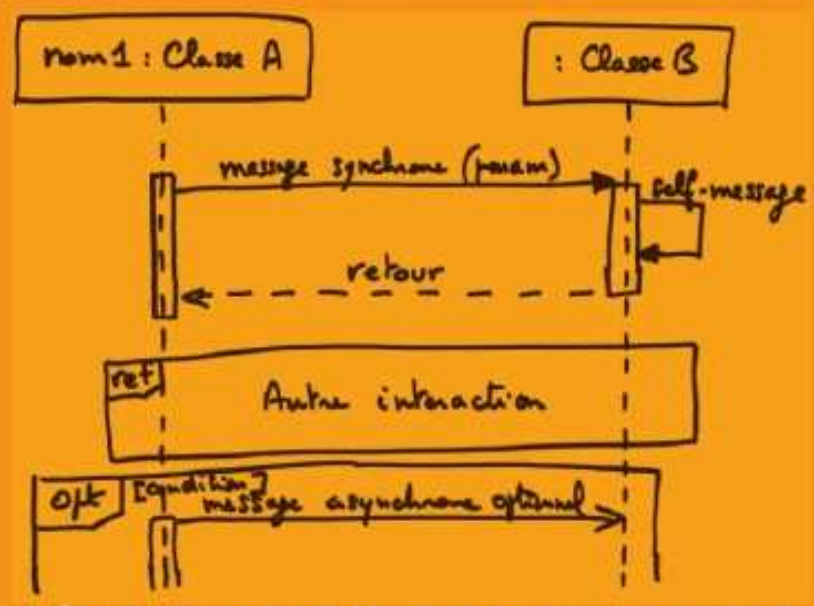
- Ligne de vie : représentation de l'existence d'un élément participant dans un diagramme de séquence. Une ligne de vie a un nom et un type.
- Message : élément de communication unidirectionnel entre lignes de vie qui déclenche une activité dans l'objet destinataire. La réception d'un message provoque un événement dans l'objet récepteur. La flèche pointillée représente un retour au sens UML. Cela signifie que le message en question est le résultat direct du message précédent.

Trucs et astuces

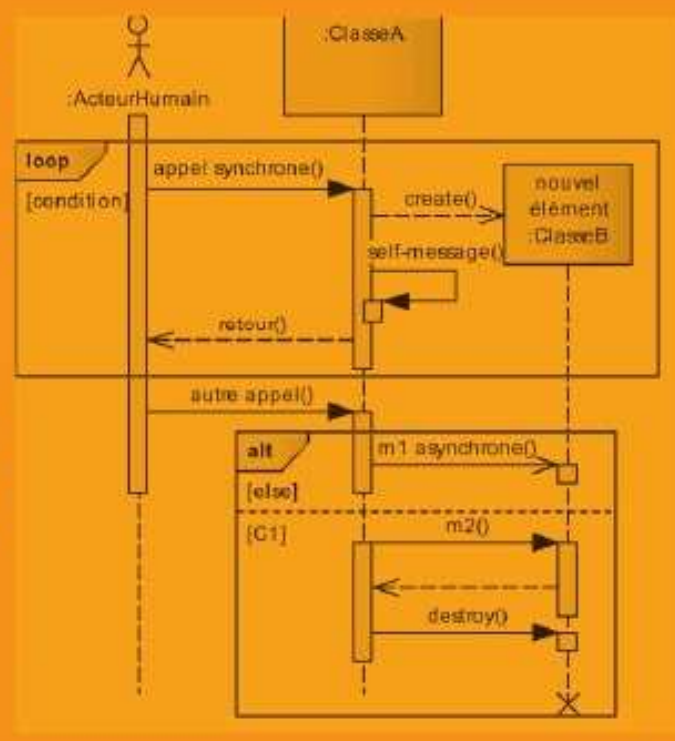
Copyright © 2016 Eyrolles.

Diagramme de séquence

- Ce type de diagramme illustre un scénario particulier au sein du système.
- Les lignes de vie ne sont pas des classes, mais pas vraiment des objets non plus. Il s'agit en fait de sorte d'instances génériques, plus précisément de rôles typés. La notation UML2 est `nom:Type`, sans soulignement *comme c'était le cas en UML1*. Le nom étant optionnel, on se retrouve souvent avec la notation `:NomType`.
- Un objet peut s'envoyer via un lien un message à lui-même. Il s'agit d'un traitement interne qui se traduit généralement par une méthode privée *private* en programmation objet alors que la réception d'un message venant d'un autre objet correspond forcément à l'invocation d'une méthode publique *public*.



- **Spécification d'activation** : bande blanche qui représente une période d'activité sur une ligne de vie.
- **Message synchrone** : envoi de message pour lequel l'émetteur se bloque en attente du retour et qui est représenté par une flèche pleine. Un message asynchrone, au contraire, est représenté par une flèche ouverte.
- **Occurrence d'interaction** : une interaction peut faire référence explicitement à une autre interaction grâce à un cadre avec le mot-clé ref et indiquant le nom de l'autre interaction.



Trucs et astuces

- La bande d'activation est optionnelle en analyse, mais peut se révéler utile en conception objet pour montrer l'imbrication des appels d'opérations.
- Un message peut porter des paramètres, eux-mêmes typés.
- UML2 propose une notation très utile : les cadres d'interaction. Chaque cadre possède un opérateur et peut être divisé en fragments. Les principaux opérateurs sont :

`loop` : boucle. Le fragment peut s'exécuter plusieurs fois, et la condition de garde explicite l'itération.

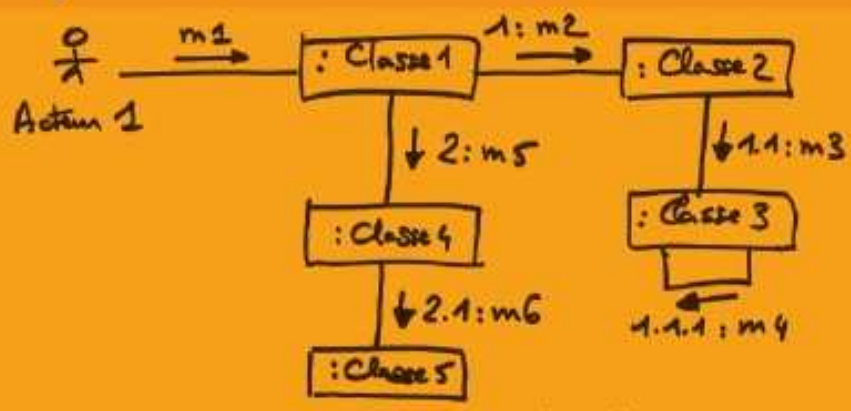
`opt` : optionnel. Le fragment ne s'exécute que si la condition fournie est vraie.

`alt` : fragments alternatifs. Seul le fragment possédant la condition vraie

Copyright © 2016 Eyrolles.

Diagramme de communication

Montre la communication entre éléments dans le plan au sein d'une interaction

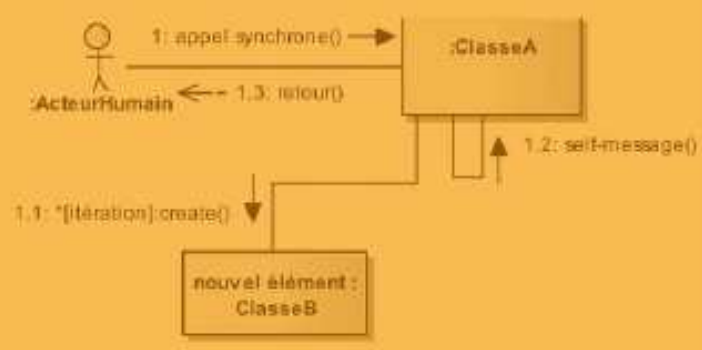


■ **Lien** : connexion sémantique entre éléments par laquelle un émetteur peut communiquer avec un récepteur par envoi de message.

Trucs et astuces

■ La notation de base *messages, participants* est la même que pour le diagramme de séquence. La spécificité du diagramme de communication consiste à la représentation explicite des liens entre participants et à la nécessité des numéros de messages.

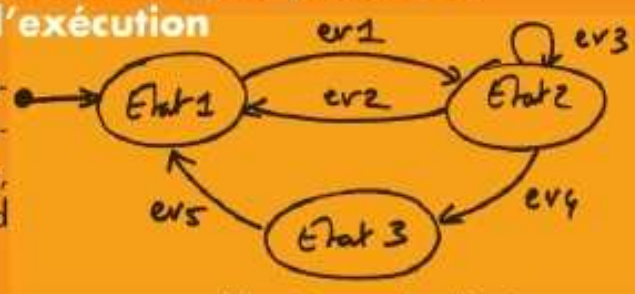
■ La numérotation décimale permet de montrer l'imbrication des envois de messages.



■ Les cadres d'interaction des diagrammes de séquence n'ont pas d'équivalent dans le diagramme de communication. Une boucle se représente par une notation de multiplicité * et une condition d'itération optionnelle avant le nom du message. La condition entre crochets permet aussi de pallier l'absence des opérateurs `opt` et `alt`.

Diagramme d'états

Montre les différents états et transitions possibles des objets d'une classe à l'exécution



■ **État** : condition ou situation qui se produit durant la vie d'un objet pendant laquelle il satisfait une certaine condition, exécute une activité particulière ou attend certains événements.

■ **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 objets.

Copyright © 2016 Eyrolles.

Diagramme d'états

■ Effet : action ou activité qui s'exécute lorsqu'une transition se déclenche. L'exécution de l'effet est unitaire et ne permet de traiter aucun événement supplémentaire pendant son déroulement.

Trucs et astuces

■ Le pseudo-état initial ne peut avoir que des flèches qui sortent, le pseudo-état final ne peut avoir que des flèches qui entrent.

■ Tous les éléments d'une transition sont optionnels. Même l'événement déclencheur est optionnel : cela signifie que l'activité finie de l'état source s'est terminée normalement.

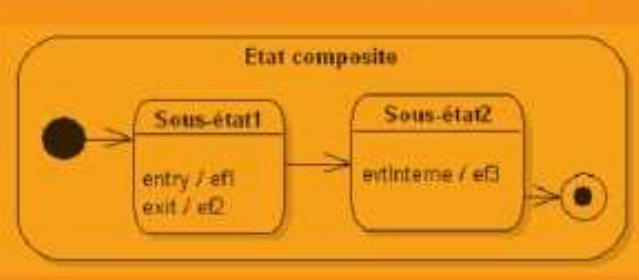
■ Un état peut contenir une activité durable *do-activity*. Cette activité durable est interruptible par un événement, mais peut également se terminer d'elle-même. On parle dans ce cas d'activité finie, par opposition à une activité continue. La transition de complétion d'une activité finie, aussi appelée transition automatique, est représentée sans nom d'événement ni mot-clé.

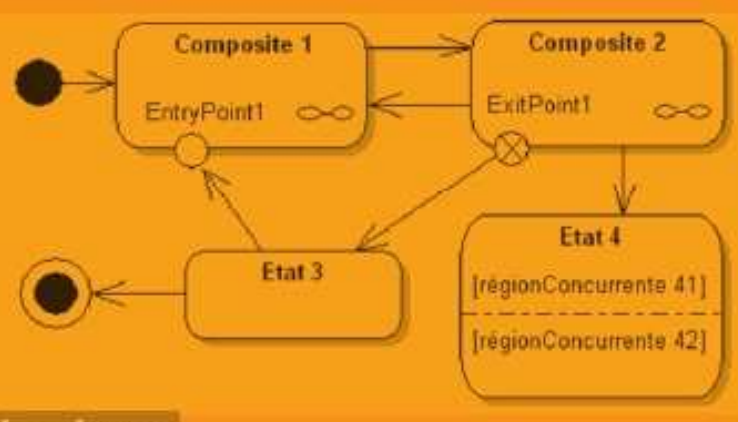
■ UML propose de déclarer les événements internes à un objet par les mots-clés `when` *condition*, `after` *durée*, ou `at` *date*.

■ Un événement *comme une transition* est par convention instantané, ou en tout cas ininterruptible *atomique*.

■ Attention, sur une transition, l'effet est toujours déclenché après l'évaluation à vrai de la condition de garde.

■ Sous-état : état englobé dans un super-état ou état composite. Les sous-états peuvent être séquentiels ou concurrents si l'état est découpé en régions concurrentes.





Trucs et astuces

■ Les effets d'entrée *entry* et de sortie *exit* sont exécutés respectivement lors de toute entrée ou de toute sortie de l'état concerné.

■ Les états peuvent réagir à des événements sans utiliser de flèche de transition. On parle alors de transition interne, notée à l'intérieur de l'état. Une transition interne est semblable à une transition qui reboucle sur l'état, mais sans effet secondaire *entry, exit, etc.*

■ Utilisez le concept d'état composite *ou super-état* pour factoriser des transitions déclenchées par le même événement et amenant au même état.

■ L'intérieur d'un état composite complexe peut être décrit dans un autre diagramme. Dans ce cas, une notation particulière *o-o* symbolisant deux états reliés est présente en bas à droite de l'état composite non expansé.

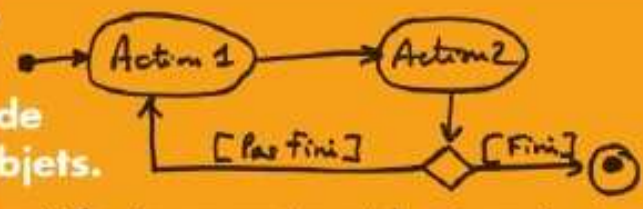
■ Les régions concurrentes permettent d'exprimer du parallélisme au sein d'un objet.

■ Ne perdez pas de temps à dessiner des diagrammes d'états qui ne

Copyright © 2016 Eyrolles.

Diagramme d'activité

Montre l'enchaînement des actions au sein d'une activité à l'aide de flots de contrôle et d'objets.



■ Action : unité fondamentale de spécification comportementale qui représente un traitement ou une transformation. Les actions sont contenues dans les activités, qui fournissent leur contexte.

■ Flot : 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.

■ Décision : nœud de contrôle structuré représentant un choix dynamique entre plusieurs conditions. Il possède un arc entrant et plusieurs arcs sortants. À l'inverse, une fusion merge représente un emplacement où plusieurs chemins de contrôle alternatifs s'assemblent.

Trucs et astuces

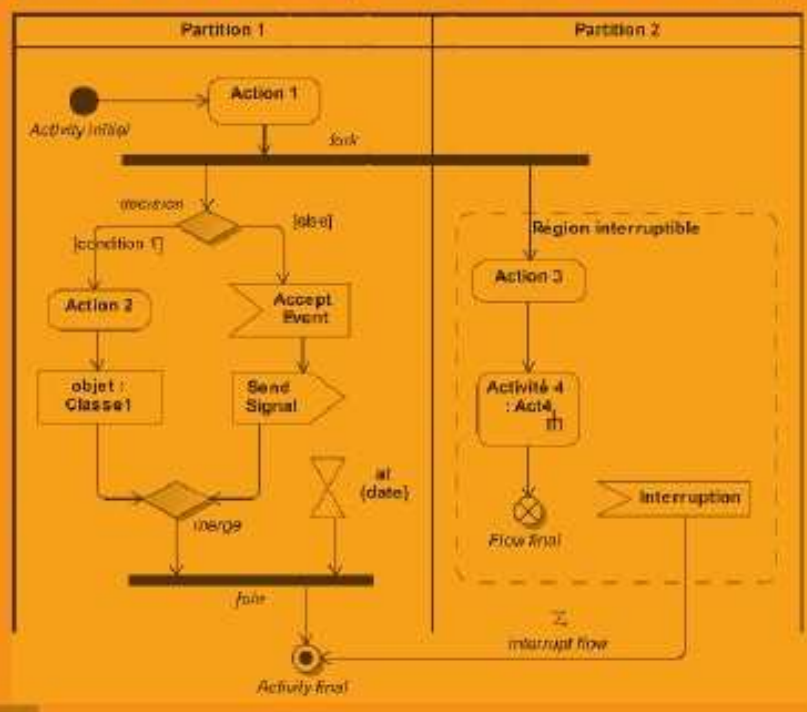
■ Une activité représente l'exécution d'un calcul complexe, alors qu'une action est un comportement élémentaire.

■ Les activités servent à modéliser à la fois le comportement séquentiel et simultané.

■ Une action est activée dès que tous ses flots d'entrée possèdent un jeton, s'exécute de façon unitaire et met à disposition un jeton sur chaque flot de sortie quand elle est terminée *sémantique inspirée des réseaux de Petri.*

■ Débranchement fork : nœud de contrôle structuré représentant le démarrage d'actions concurrentes. Il possède un arc entrant et plusieurs arcs sortants. À l'inverse, une jointure join représente un emplacement où plusieurs chemins de contrôle parallèles s'assemblent pour se synchroniser.

■ Région interruptible : permet de spécifier un ensemble de nœuds d'activité au sein duquel l'activité se termine si un certain événement d'interruption se produit.



Trucs et astuces

■ Un nœud initial possède un arc sortant et pas d'arc entrant. La fin d'activité possède un ou plusieurs arcs entrants et aucun arc sortant. Dès que l'un des arcs entrants est activé, l'exécution de l'activité s'interrompt complètement. Une fin de flot *flow final* permet de terminer une branche autonome sans interrompre complètement l'activité englobante.

■ Un flot d'objet représente l'existence d'un objet, générée par une action et utilisée par d'autres. Il est noté par un rectangle contenant un nom et indiquant optionnellement un type et un état.

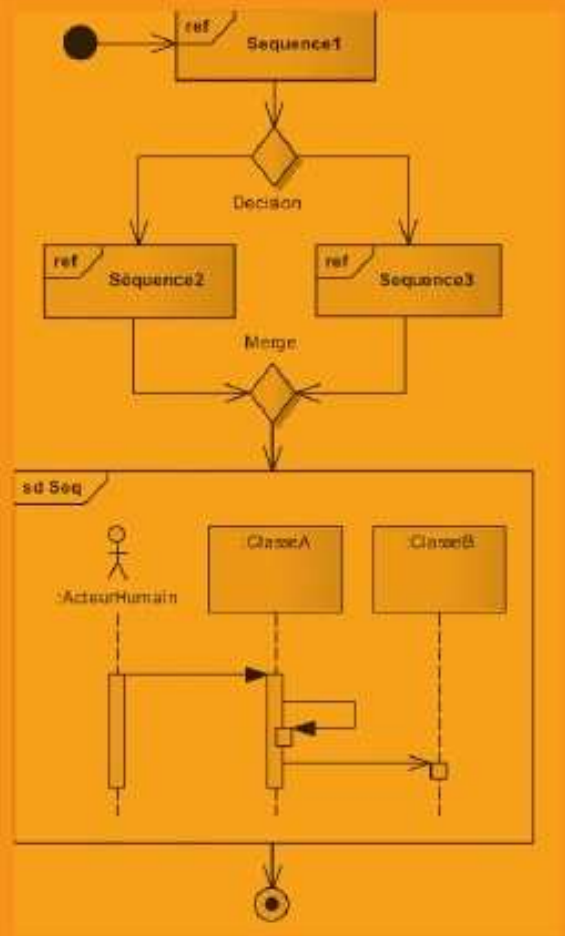
Copyright © 2016 Eyrolles

Diagramme d'activité

- Une action peut représenter l'appel d'une autre activité. On le représente par un symbole de râteau en bas à droite de l'action.
- Les nœuds d'une activité peuvent être organisés en partitions *verticales ou horizontales* permettant de séparer les responsabilités. C'est particulièrement utile pour modéliser des processus d'entreprise.
- N'utilisez pas forcément toutes les subtilités du diagramme d'activité : pensez à votre lecteur !

Diagramme de vue globale d'interaction

Combine les diagrammes d'activité et de séquence pour organiser des fragments d'interaction avec des décisions et des flots

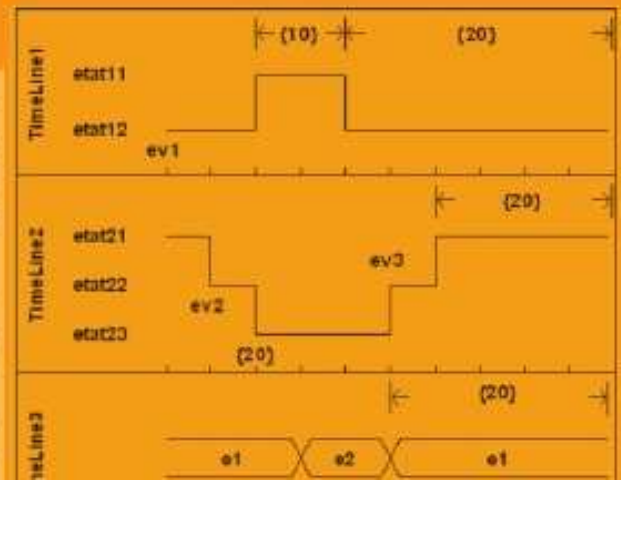


Trucs et astuces

- L'utilisation de ce type de diagramme *Interaction Overview* est intéressante au niveau des cas d'utilisation, voire de la dynamique du système global. Il permet de relier des diagrammes de séquence entre eux afin de montrer le cadre global dans lequel ils s'exécutent.
- Il peut être aussi une manière plus lisible d'exprimer des alternatives *en comparaison avec l'opérateur alt du diagramme de séquence.*

Diagramme de temps

Combine les diagrammes d'états et de séquence pour montrer l'évolution de l'état d'une ligne de vie au cours du temps et les messages qui modifient cet état



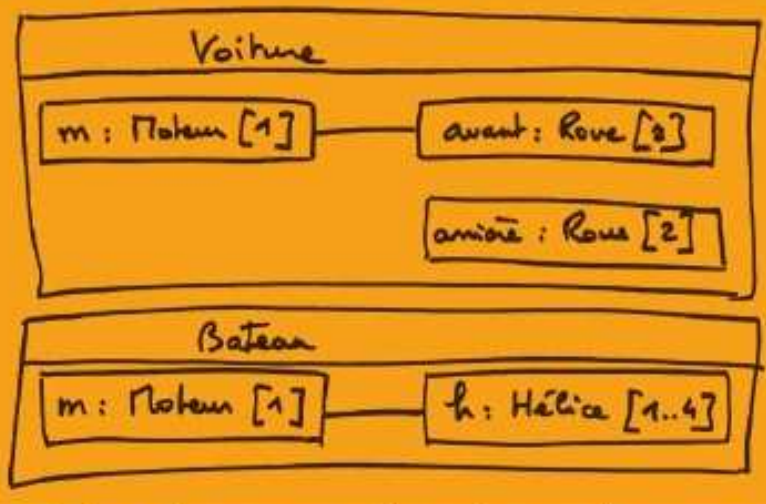
Trucs et astuces

- Le diagramme de temps *timing diagram* est une autre variante de diagramme d'interaction dans lequel l'accent est mis sur l'expression des contraintes temporelles.
- Il ressemble à un diagramme de séquence dans lequel les lignes de vie seraient horizontales et le temps proportionnel. Il montre les oscillations entre différents états ou conditions,

Copyright © 2016 Eyrolles.

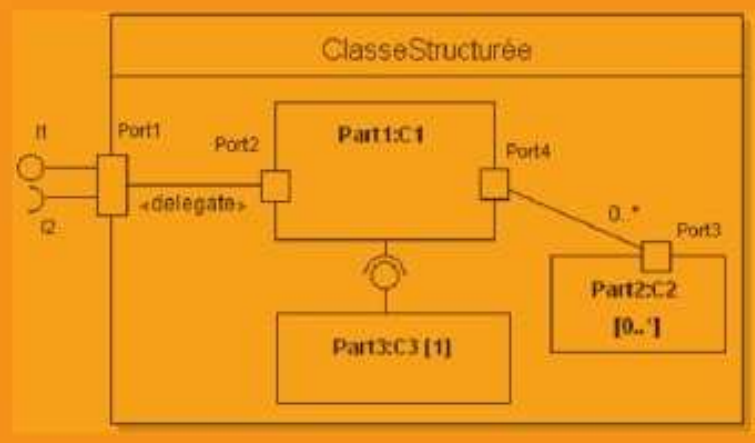
Diagramme de structure composite

Montre l'organisation interne d'un élément structurel complexe



■ Classe structurée : classe contenant des parties parts, des connecteurs et éventuellement des ports qui définissent à eux tous sa structure interne.

■ Partie : fragment structuré d'une classe qui décrit le rôle joué par une instance dans le contexte de la classe structurée. Elle possède un nom, un type et une multiplicité.



■ Connecteur : relation contextuelle entre des parties dans le contexte d'une classe structurée.

■ Port : point d'interaction individuel entre une classe et son environnement. Dans une classe structurée, on peut connecter les ports à des parties internes delegate ou à la spécification de comportement de l'objet dans son ensemble.

Trucs et astuces

■ La relation de composition classique fonctionne bien pour modéliser de la décomposition hiérarchique. Cependant, elle présente des limitations significatives lorsqu'il s'agit de relier des éléments au même niveau de décomposition. Préférer alors la classe structurée.

■ La différence entre un connecteur et une association classique est la suivante : chaque association est indépendante des autres, alors que tous les connecteurs d'une classe structurée partagent un contexte unique.

Diagramme de composants

Montre des structures complexes, avec leurs interfaces fournies et requises

Copyright © 2016 Eyrolles.

Diagramme de composants

Trucs et astuces

- Un composant est une sorte de classe structurée.
- Le diagramme de composants montre les unités logicielles à partir desquelles on construit le système informatique ainsi que leurs dépendances.

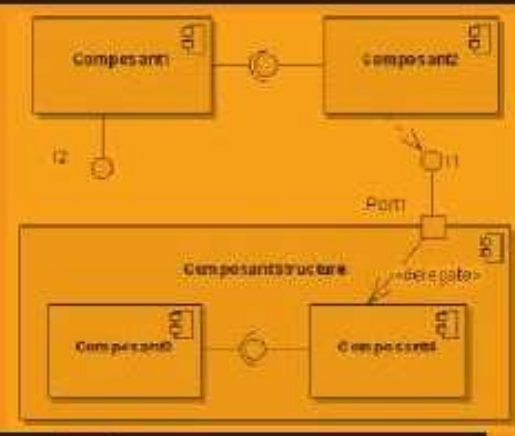
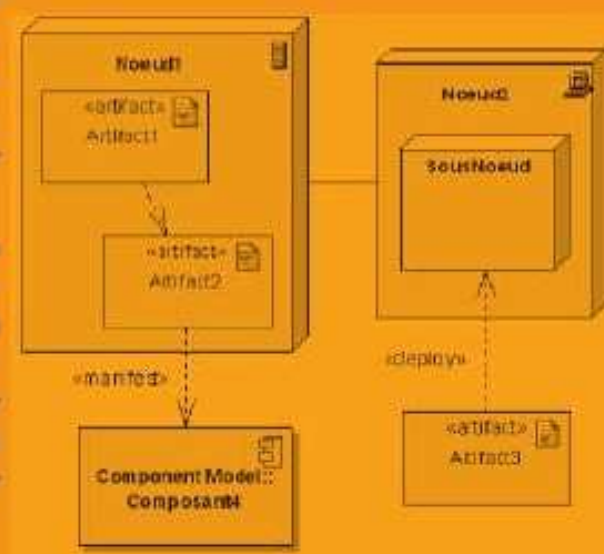


Diagramme de déploiement

Montre le déploiement physique des artéfacts sur les ressources matérielles

- Artéfact artifact : spécification d'un élément physique d'information modèle, fichier ou logiciel pouvant être lié à un composant relation `<<manifest>>` et déployé sur un nœud.
- Nœud node : élément physique existant à l'exécution et représentant une ressource de calcul, doté de mémoire et souvent de capacités de traitement. Un nœud peut être soit un simple élément matériel device, soit un environnement d'exécution logiciel.



Trucs et astuces

- Le diagramme de déploiement doit rester simple. Il permet de faire le lien entre le monde physique, matériel, et les abstractions logiques que sont les composants et les classes.
- Cette vue permet de traiter les conséquences de la distribution de l'architecture informatique et de l'allocation de ressources.
- Un artéfact peut être déployé sur un nœud en montrant l'imbrication graphique ou en le reliant par une relation « deploy ». Il est utile de montrer la relation « manifest » entre un artéfact et un composant.

Du même auteur...

P. Roques. – ***UML 2. Modéliser une application web, 4e édition***. N°12389, 2008, 246 pages (collection Les Cahiers du Programmeur).

P. Roques, F. Vallée. – ***UML 2 en action. De l'analyse des besoins à la conception, 4e édition***. N°12104, 2007, 382 pages (collection Architecte logiciel).

P. Roques. – ***UML 2 par la pratique. Études de cas et exercices corrigés, 7e édition***. N°12565, 2009, 396 pages (collection Noire).

P. Roques. – ***Modélisation de systèmes complexes avec SysML***. N°13641, 2014, 186 pages (collection Blanche).

Code éditeur : G14356
ISBN : 978-2-212-14356-0
7 €
Conception : Nord Compo