



Génie Logiciel 2

Code du module : F321

Présenté par:

Amal HALFAOUI (Epse GHERNAOUT)

a_halfaoui@mail.univ-tlemcen.dz

amal.halfaoui@gmail.com

Université Tlemcen -Licence L3- 2017/2018



Objectif du cours

- 1- S'initier à la **modélisation objet**
 - 2- Développer vos **capacités d'analyse** de problèmes de conception logiciel à travers une modélisation Objet
- * En utilisant le langage de modélisation **UML** et une démarche avec le processus unifié **UP**



Objectif du cours (en gros)

- présenter les diagrammes d'UML de manière générale en se focalisant sur les plus importants d'entre eux.
- illustrer la modélisation à l'aide des diagrammes d'UML en s'appuyant sur des exemples et des exercices en TD et TP.
- Présenter une démarche de mise en œuvre d'UML qui est fondée sur le processus standard du développement itératif et incrémental UP à l'aide de fiches de guides et un cas d'étude.



Organisation du cours

- 14 séances de cours
- 9 séances de TD (à partir du 24/10/2016)
- 5 séances de TP (à partir du 24/10/2016)

Outil de modélisation: Modélio

<https://www.modelio.org/downloads/downloadmodelio.html>

Equipe pédagogique:

Cours : Halfaoui Amal

TD : Halfaoui Amal () & Belhabib Amel ()

TP : Halfaoui Amal & Belhabib Amel & Elyebdri Zeyneb & Abdedjellil Hanane



Modalités d'évaluation du cours

- Note d'Examen final
- Note Projet/contrôle continu
- Note TP : consultation et Compte rendu après chaque TP (pénalisation en cas d'absence)



Plan du cours

1- Introduction:

rappels et définitions + Modélisation UML (passer en revue les principes, vues, diagrammes et démarches)

2- Modélisation d'un système avec UML

2-1 Modélisation de l'aspect fonctionnel d'un système

2-2 Modélisation de l'aspect statique d'un système

2-3 Modélisation de l'aspect dynamique d'un système

2-4 Modélisation de l'aspect déploiement d'un système

3- Mise en œuvre d'UML

Programme du cours (prévisionnel)



Date	Cours	
11/09/2017	Introduction, rappels et définition + Modélisation UML (passer en revue les principes, vues, diagrammes et démarches)	
18/09/2017	Diagramme de cas d'utilisation	2- Modélisation UML → 2-1 Modélisation de l'aspect fonctionnel d'un système
25/09/2017	Diagramme de cas d'utilisation	
2/10/2017	Diagramme de classe	2-2 Modélisation de l'aspect statique d'un système
9/10/2017	Diagramme de classe	
16/10/2017	Diagramme de classe / Diagramme de paquetage	
23/10/2017	Diagramme d'objet	
30/10/2017	Diagramme de séquence	2-3 Modélisation de l'aspect dynamique d'un système
6/11/2017	Diagramme de communication	
13/11/2017	Diagramme d'état transition / d'activité	
20/11/2017	Diagramme de composant / diagramme de déploiement	2-4 Modélisation de l'aspect déploiement
27/11/2017	Méthodes et approche de Mise en œuvre de UML	3- Mise en œuvre d'UML
18/12/2017	Mise en œuvre de UML (processus unifié)	



- Cours “**Conception Orienté Object avec UML**” (2016) de Selma Mouni.
- Cours “ **UML Contexte, méthodes et processus**” (2016) Aurélien Tabard
- Cours “**UML Introduction au génie logiciel et à la modélisation**” (2015) par Delphine Longuet
- Livre **UML par la pratique de Pascale Roque** 5^e édition (2006)
- Livre **UML 2 De l'apprentissage à la pratique** Par Laurent AUDIBERT (2009)



Rappels et définitions

- Logiciel et Génie logiciel
- Développement logiciel
- Modélisation



- Au cœur de toutes les entreprises
- Systèmes d'informations:
 - 20% pour le matériel
 - 80% pour le logiciel
- La problématique est principalement logiciel



- Qu'est ce qu'un logiciel ou une application?
- Combien de personnes pour développer un logiciel?
- Coût du développement comparativement à la production ?



- Ensemble d'entités nécessaires au fonctionnement d'un processus de traitement automatique de l'information
 - Programmes, données, documentation...
- Ensemble de programmes qui permet à un système informatique d'assurer une tâche ou une fonction en particulier

Logiciel = programme + utilisation

Rappel: logiciel (environnement)



- utilisateurs : - grand public (exp: logiciel de traitement de texte),
 - spécialistes (calcul météorologique),
 - développeurs (compilateur)
- autres logiciels: librairie, composant
- matériel : capteurs (système d'alarme)

Rappel: Développement logiciels (problèmes)



- Le développement est une activité relativement facile!!
- On peut créer rapidement un premier prototype
- Mais l'étendre devient rapidement difficile...



<https://www.flickr.com/photos/78044378@N00/3640037/>



<https://www.flickr.com/photos/10402746@N04/7165270428/>



- **La célèbre crise du logiciel**

- Constat du développement logiciel fin années 60:
- délais de livraison non respectés
- budgets non respectés
- ne répond pas aux besoins de l'utilisateur ou du client
- difficile à utiliser, maintenir, et faire évoluer

- **Bugs célèbres**

Sonde Mariner 1 (1962), PlayStation Network (avril 2011), Outil de chiffrement OpenSSL (mars 2014),..., Bug d'accélération de toyota (2010)

Rappel: Développement logiciel (problèmes: pourquoi?)



Les problèmes sont liés aux :

- Erreurs humaines
- Taille et complexité des logiciels
- Taille des équipes de conception/développement
- Manque de méthodes de conception
- Négligence de la phase d'analyse des besoins du client
- Négligence et manque de méthodes et d'outils des phases de validation/vérification

Solution au problèmes : Génie logiciel



- **Idée:** appliquer les méthodes classiques d'ingénierie au domaine du logiciel
- **Ingénierie (ou génie):** Ensemble des fonctions allant de la conception et des études à la responsabilité de la construction et au contrôle des équipements d'une installation technique ou industrielle



- **Définition:** Ensemble des **méthodes**, des **techniques** et des **outils** dédiés à la conception, au développement et à la maintenance des systèmes informatiques
- **Objectif:** Avoir des procédures systématiques pour des logiciels de grande taille afin que
 - la spécification corresponde aux besoins réels du client
 - le logiciel respecte sa spécification;
 - les délais et les coûts alloués à la réalisation soient respectés

Processus de développement logiciel



- **Phases du cycle de vie logiciel**
 - Expression des besoins
 - Analyse
 - Conception
 - Programmation
 - Validation et vérification
 - Maintenance
- **Différents modèles du cycle** : cascades, V, spirale
- **Pour chaque phase**: Utilisation et production de documents

Documentation d'un projet logiciel



- **Documents produits**

- Analyse des besoins → Cahier des charges fonctionnel
- Conception → Dossier de conception
- Programmation → Code documenté + manuel d'utilisation

- **Problèmes:**

- La rédaction du cahier des charges est le plus souvent en langage naturel (informel)
- Ambiguïtés: plusieurs sens d'un même mot selon les personnes ou les contextes
- Contradictions, oublis, redondances difficiles à détecter
- Difficultés à trouver une information
- Mélange entre les niveaux d'abstraction (spécification vs. conception)



- **L'objectif du GL est de:**
 - Améliorer la qualité ;
 - Réduire les délais;
 - Optimiser les coûts.
- **Face à cela 3 principaux défis du logiciel d'aujourd'hui**
 - Composants de plus en plus complexe
 - Applications de plus en plus larges/distribuées
 - Des besoins qui évoluent en cours de route
- **Solution**

Modélisation



Définition : Modèle

Un modèle est une représentation **abstraite** et **simplifiée** (i.e. qui exclut certains détails), d'une entité complexe (phénomène, processus, système, logiciel etc.) du monde réel en vue de le **décrire**, de **l'expliquer** ou de le **prévoir**

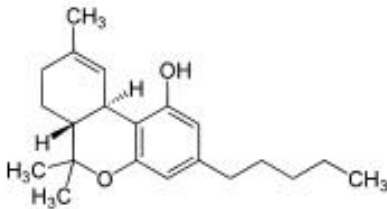
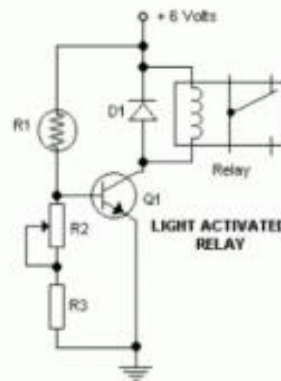
- **Modéliser**: abstraire la réalité pour mieux comprendre le système à réaliser / réalisé
- L'objectif de la modélisation est de **maîtriser la complexité** : en éliminant les détails qui n'influencent pas son comportement de manière significative

Modélisation (Principe 2/2)



“Un beau dessin vaut mieux qu'un long discours ”

- Les modèles sont donc souvent graphiques, même si l'objet à créer n'est pas matériel.



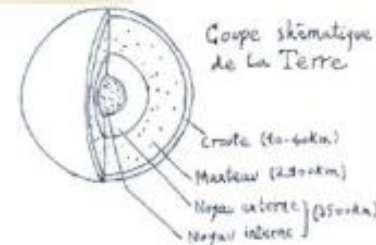
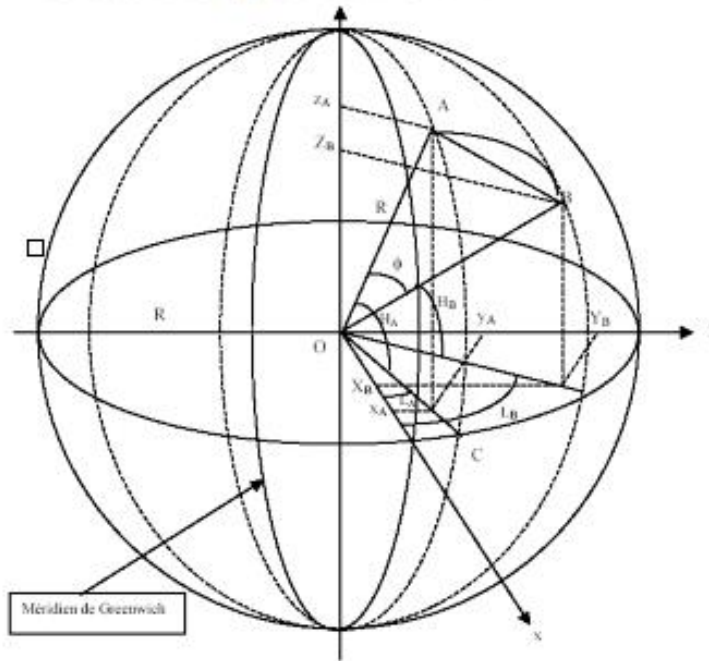
Modèle (caractéristiques)



- Le modèle **doit être relié au monde réel** (système réel qu'il représente)
- **Ne pas confondre modèle et système**

Exp : La carte du monde est un modèle de la terre

- Permettant de poser certaines questions ...
- mais pas d'autres.



Modèle (caractéristiques)



- peut être exprimé avec différents **niveaux d'abstraction / raffinement** :

Exp: répartition électrique de l'immeuble, de la cage d'escalier, de l'appartement, de la pièce

- peut être présenté selon **différentes vues** :

Une seule « vue » du système n'est pas suffisante pour Les intervenants multiples du projet informatique possèdent des préoccupations multiples

Modèle (caractéristiques)



- Différents niveaux d'abstraction
- Différents points de vue

Exp: Google Map





- **Gérer la complexité des grandes applications**
 - permet de comprendre le système et réfléchir sur la conception
 - séparation des préoccupations: plusieurs vues sur le problème (vue statique, comportementale)
 - mieux répartir les tâches entre les nombreux intervenants.
- **Faciliter la Communication**
 - offre une vision plus abstraite pour communiquer entre les nombreux acteurs d'un gros projet (car on peut difficilement communiquer avec du code entre analystes et développeurs)

Avantages de la modélisation (suite)



- **Augmenter la productivité**
 - Générer du code à partir des modèles ;
 - Maîtriser la variabilité : Un modèle générique d'un produit avec plusieurs variantes;
 - Facteur de réduction des coûts et des délais .
- **Pérenniser un savoir-faire** (pour des projets qui durent sur des années)
 - capitaliser un savoir faire indépendant du code et des technologie;
 - capturer le métier sans se soucier du détail technique;
 - Fournir un guide pour la construction du système ;
 - documenter le système et les décisions prises.



- **Langage de modélisation** : une syntaxe commune, graphique, pour modéliser visualiser, spécifier, construire et documenter un logiciel (exp : OMT, UML, ...).
- **Les méthodes de modélisation** sont souvent des **méthodes de développement (analyse + conception)** qui comportent une partie modélisation.
- Un langage n'est pas une méthode !!
- Une méthode de modélisation utilise ou définit un langage de modélisation accompagnée d'une démarche

**Méthode = langage + démarche (+ outils)
façon de modéliser et façon de travailler**



- **Définitions**

- guide plus ou moins formalisé;
- démarche reproductible permettant d'obtenir des solutions fiables à un problème donné

- **Capitalise**

- l'expérience de projets antérieurs ;
- les règles dans le domaine du problème

- **Une méthode définit**

- des concepts de modélisation (obtenir des modèles à partir d'éléments de modélisation, sous un angle particulier, représenter les modèles de façon graphique)
- une chronologie des activités (construction de modèles)
- un ensemble de règles et de conseils pour tous les participants

Evolution des méthodes de modélisation



1^{ère} génération

Méthodes de modélisation par les fonctions

2^{ème} génération

Méthodes de modélisation par les données

génération actuelle

Méthodes de modélisation **orientée-objet**

Génération émergente

Méthodes agiles



- Décomposition d'un problème en sous-problèmes

Système = ensemble de fonctions

- Analyse fonctionnelle hiérarchique : fonctions et sous-fonctions avec fonctions entrées, sorties, contrôles (proche du fonctionnement de la machine)
 - les fonctions contrôlent la structure : si la fonction bouge, tout bouge
 - données non centralisées
- Méthodes de programmation structurée
 - IDEF0 puis **SADT**
- **Points faibles**
 - focus sur fonctions en oubliant les données, règles de
 - décomposition non explicitées, réutilisation hasardeuse



- Approches dites « systémiques »

Système = structure + comportement

- **Modélisation des données et des traitements (séparation des données des traitements)**
 - privilégie les flots de données et les relations entre structures de données (apparition des SGBD) , construire le système c'est construire sa base de donnée
 - traitements = transformations de données dans un flux (notion de processus)
- **Exemple de méthode: MERISE**
- **Points forts**
 - cohérence des données, niveaux d'abstraction bien définis.
- **Points faibles**
 - manque de cohérence entre données et traitements, faiblesse de la modélisation de traitement (mélange de contraintes et de contrôles), cycles de développement trop figés (cascade)



- **Système = ensemble d'objets qui collaborent**
 - Il est considéré de façon statique (ce que le système est : données) et dynamique (ce que le système fait : fonctions)
 - évolution fonctionnelle possible sans remise en cause de la structure statique du logiciel
- **Démarche : Conception pour et par réutilisation**
 - passer du monde des objets (du discours) à celui de l'application en complétant des modèles (pas de transfert d'un modèle à l'autre)
 - à la fois ascendante et descendante, récursive, encapsulation
 - **abstraction** forte **orientée vers la réutilisation** : notion de composants, modularité, extensibilité, adaptabilité (objets du monde), souples
- **Exemple de Méthodes : UP et ses variantes → reposent sur le langage de modélisation UML**



- **Méthodes adaptatives** (vs. prédictives)
 - Itérations courtes
 - Lien fort avec le client
 - Fixer les délais et les coûts, mais pas la portée
- **Insistance sur les hommes**
 - les programmeurs sont des spécialistes, et pas des unités interchangeables
 - Mise sur la communication humaine
 - équipes auto-organisées
- **Processus auto-adaptatif**
 - révision du processus à chaque itération
- **Exemple de méthodes : eXtreme Programming, scrum, ...**
 - Modélisation que pour les parties inhabituelles, difficiles ou délicates de la conception → utilisation **UML**



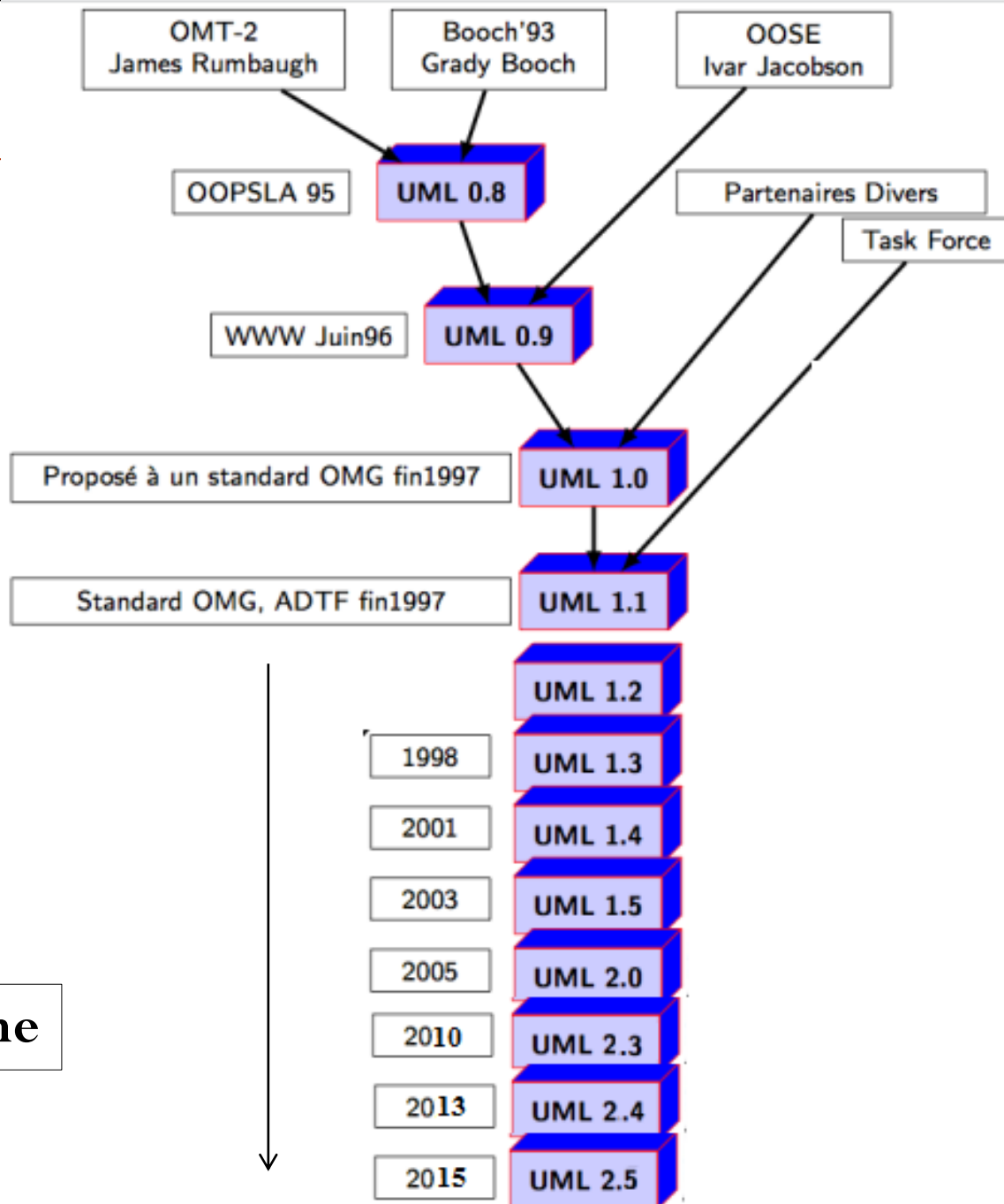
Modélisation avec UML



UML historique

- Au départ, plus de 150 méthodes et langages !
- Unification progressive de plusieurs méthodes, de remarques des utilisateurs, des partenaires
- **OMG (Object Management Group)**
Groupe créé à l'initiative de grandes sociétés informatiques américaines afin de normaliser les systèmes à objets.

UML est une Norme



UML (Unified Modeling Language)



- **Langage:**

Syntaxe et règles d'écriture

Notations graphiques normalisées (diagrammes)

- **de modélisation:**

- Abstraction du fonctionnement et de la structure du système
- Spécification et conception

- **Unifié:**

- Standard défini par l'OMG (Object Management Group)
- Fusion de plusieurs notations antérieures: Booch, OMT, OOSE

UML est un Langage graphique pour visualiser, spécifier, construire et documenter un logiciel

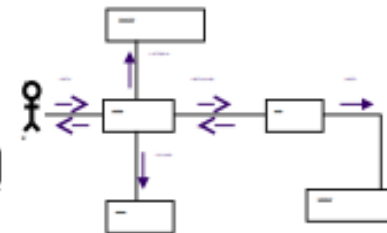
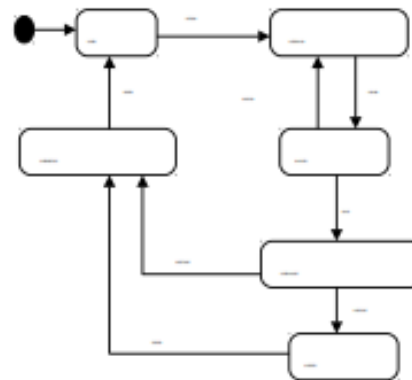
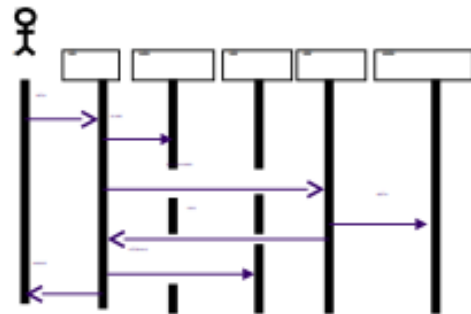
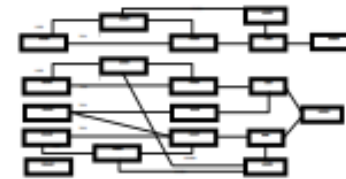
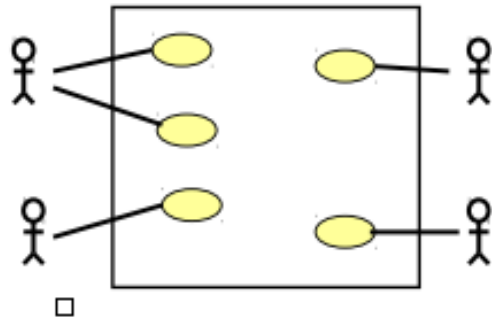
- **Langage graphique** : Ensemble de diagrammes permettant de modéliser le logiciel selon **différentes vues et différents niveaux d'abstraction**
- **Modélisation orientée objet** : UML permet de modéliser une application selon une **vision objet**, indépendamment du langage de programmation
- UML reste au **niveau d'un langage** et ne propose pas de processus de développement
 - ni ordonnancement des tâches,
 - ni répartition des responsabilités,
 - ni règles de mise en œuvre.

UML n'est pas une méthode de conception



- **Un modèle UML** : représente l'ensemble de tous les éléments modélisés. La présentation d'un modèle UML se compose de plusieurs documents écrits en langage courant et d'un ou plusieurs documents formalisés (Les diagrammes)
- **Vue** : projection d'un modèle suivant une perspective qui omet les éléments non pertinents pour cette perspective. Elle se manifeste dans des diagrammes
ex. : vue statique, vue fonctionnelle...
- **Un diagramme** est un dessin (représentation graphique) montrant une partie du modèle (graphique)

UML :Un ensemble de diagramme



UML : Un aperçu

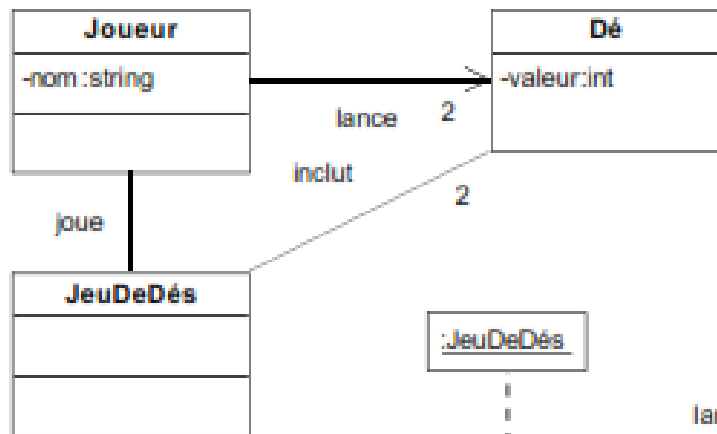


diagramme de classe,
vue statique

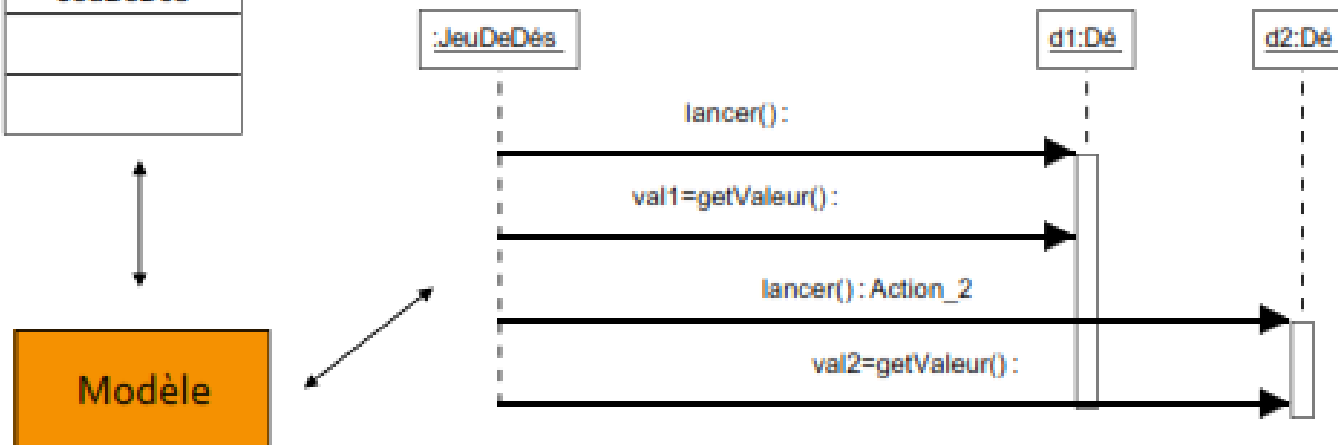
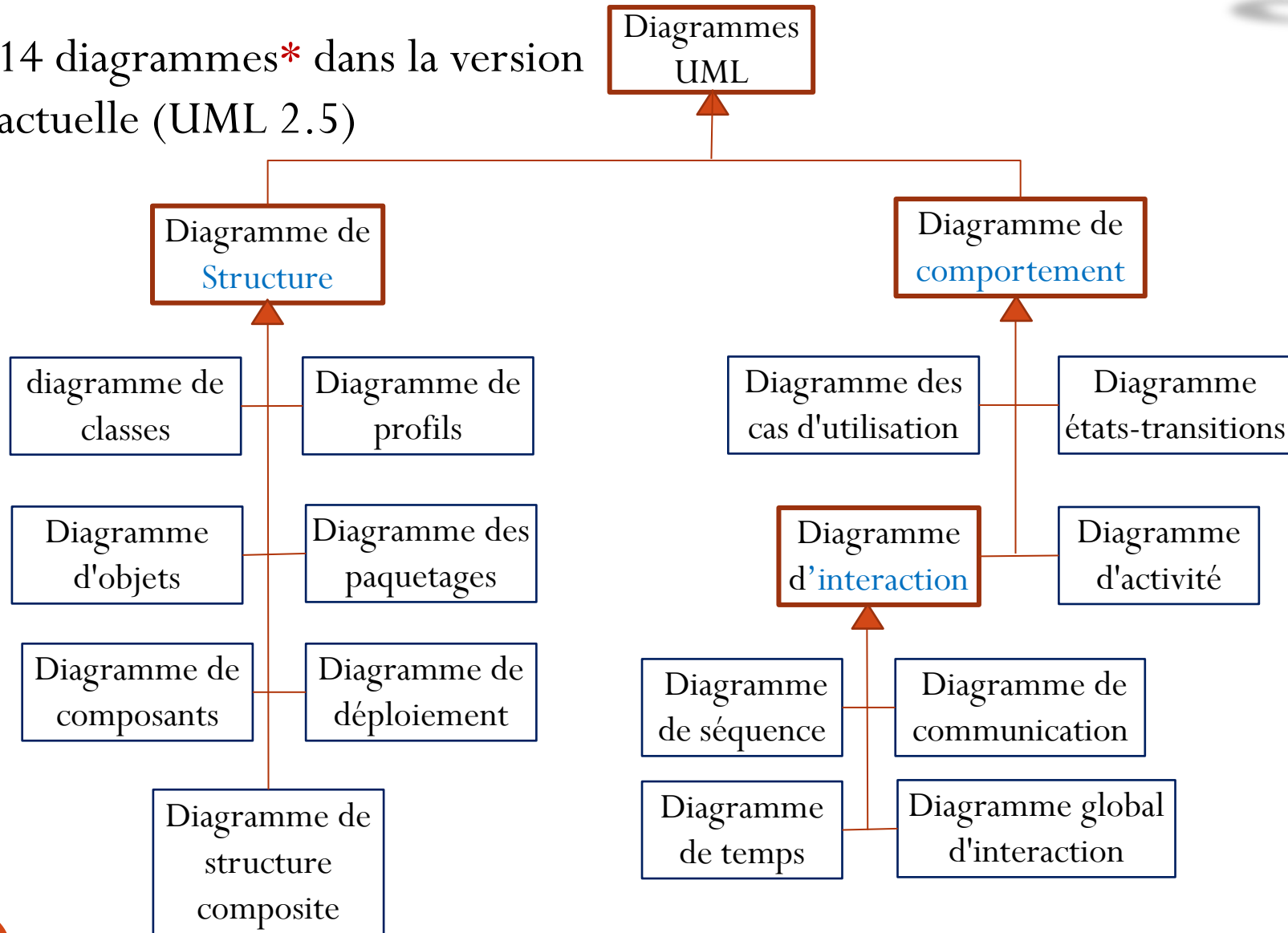


diagramme de séquences,
vue dynamique

Diagrammes UML



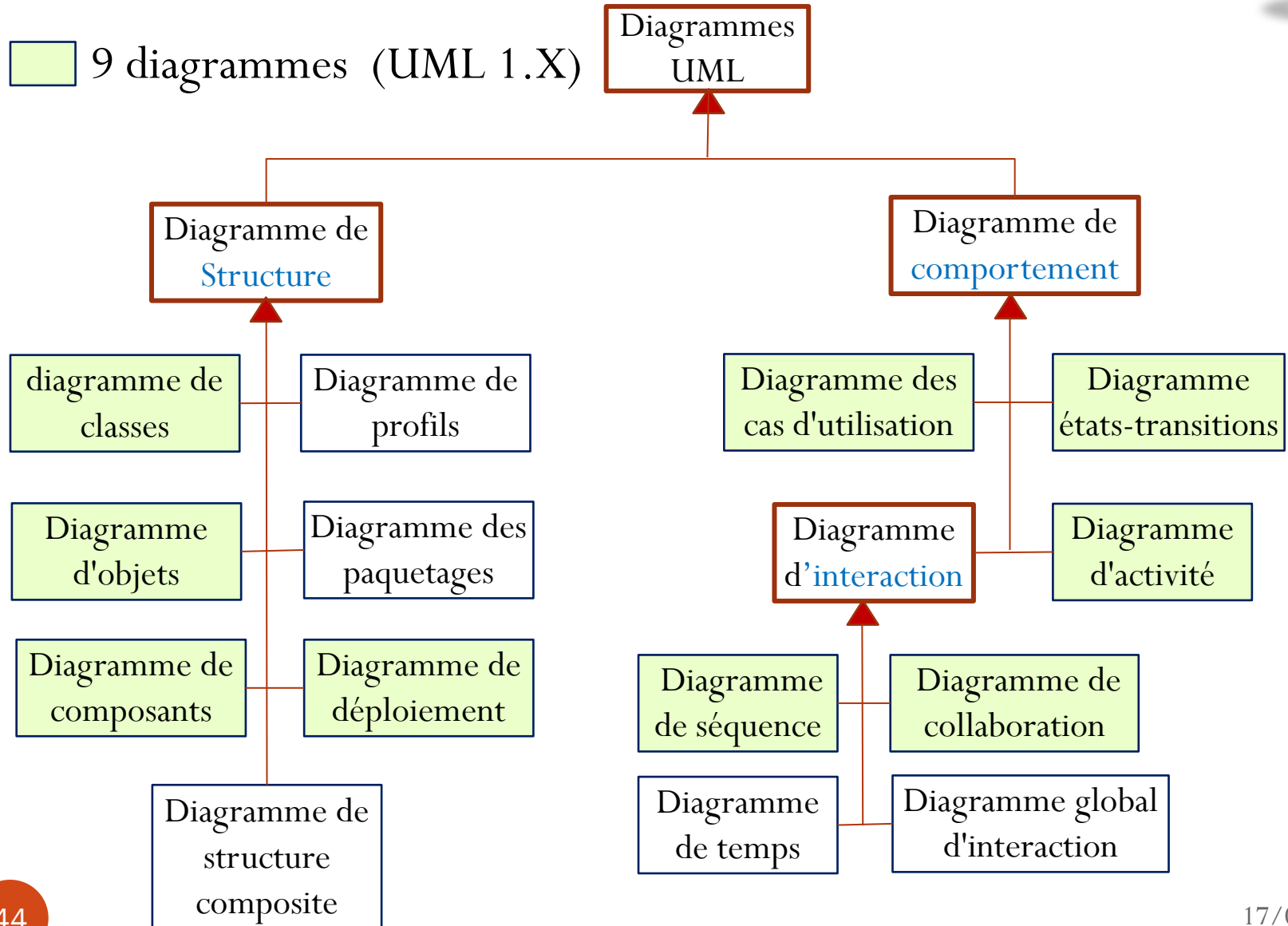
- 14 diagrammes* dans la version actuelle (UML 2.5)



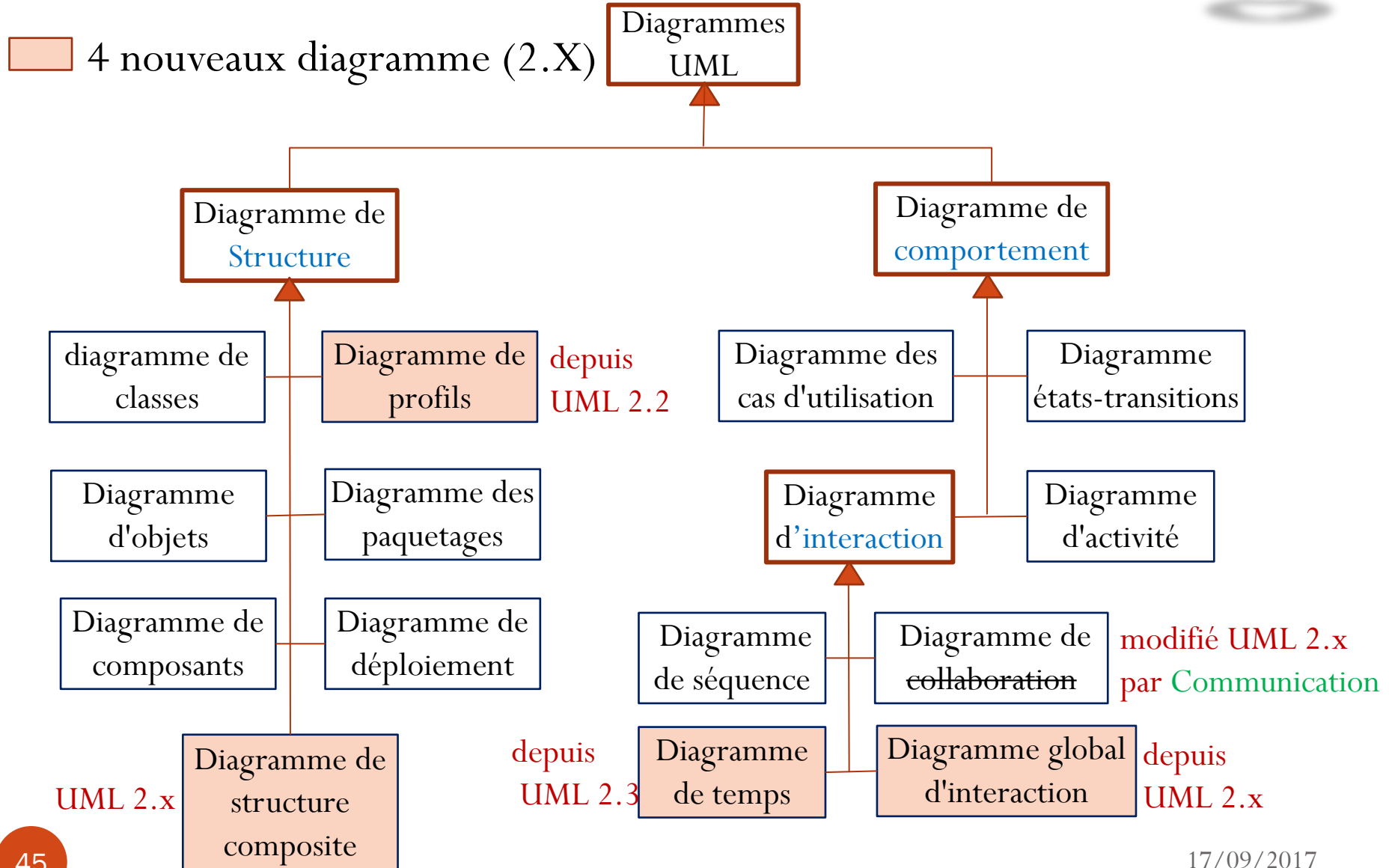
Diagrammes UML



9 diagrammes (UML 1.X)



Diagrammes UML



Phases du développement couverte par UML



- Expression des besoins
- Analyse
- Conception
- Réalisation
- Validation
- Déploiement
- Maintenance

Bien couvertes par UML

Discutable:

Réalisation: si utilisation d'UML comme langage de programmation.

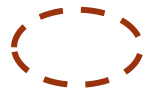
Validation: génération automatique de cas de tests mais loin d'être exhaustifs

Déploiement: un diagramme UML pour ça

Maintenance: possibilité de faire du reverse eng. Application de Design patterns, round-trip eng.

Dans ce cours beaucoup plus **les 3 premières phases**
les autres en master

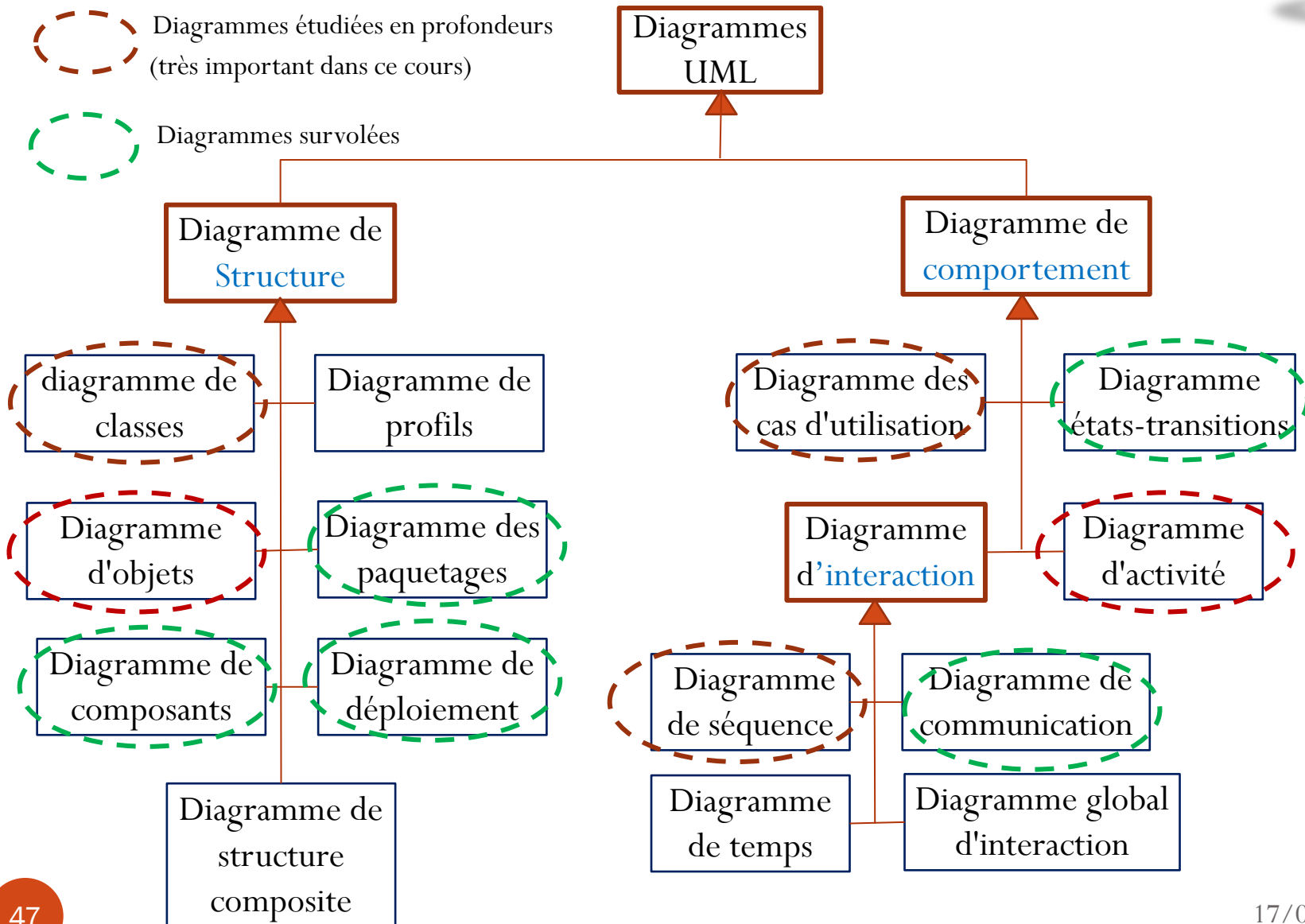
Diagrammes UML étudiés dans ce cours



Diagrammes étudiés en profondeurs
(très important dans ce cours)



Diagrammes survolées



Trois manières d'utiliser UML



1- Comme un langage pour faire des croquis et des esquisses, ébauches (**explorer**)

- Support de communication pour concevoir les parties critiques
- des modèles pas forcément 100% complets

Objectif: analyser + réfléchir + décider

- La manière la plus utilisée par les méthodes agiles



2- Comme langage de spécification de modèles, de patron,... (**Spécifier**)

- Diagrammes formels relativement détaillés.
- Génération d'un squelette de code à partir des diagrammes (exp: classes et signature **pas le corps**) →
Nécessité de compléter le code pour obtenir un exécutable

Objectif : (1) + pérenniser + génération d'une partie du code

- La manière la plus utilisée dans la gestion des gros projets



3- Comme programmation (**produire**)

- Génération automatique d'un exécutable à partir des diagrammes **Model Driven Architecture / MDA**
- Limité à des applications bien particulières

Objectif : (2) + génération automatique complète de tout le code.



- la manière dont on aimerait utiliser UML dans le future (vision utopique)
- nécessite des outils sophistiquées (pas assez mature pour le moment malgré quelques tentatives)

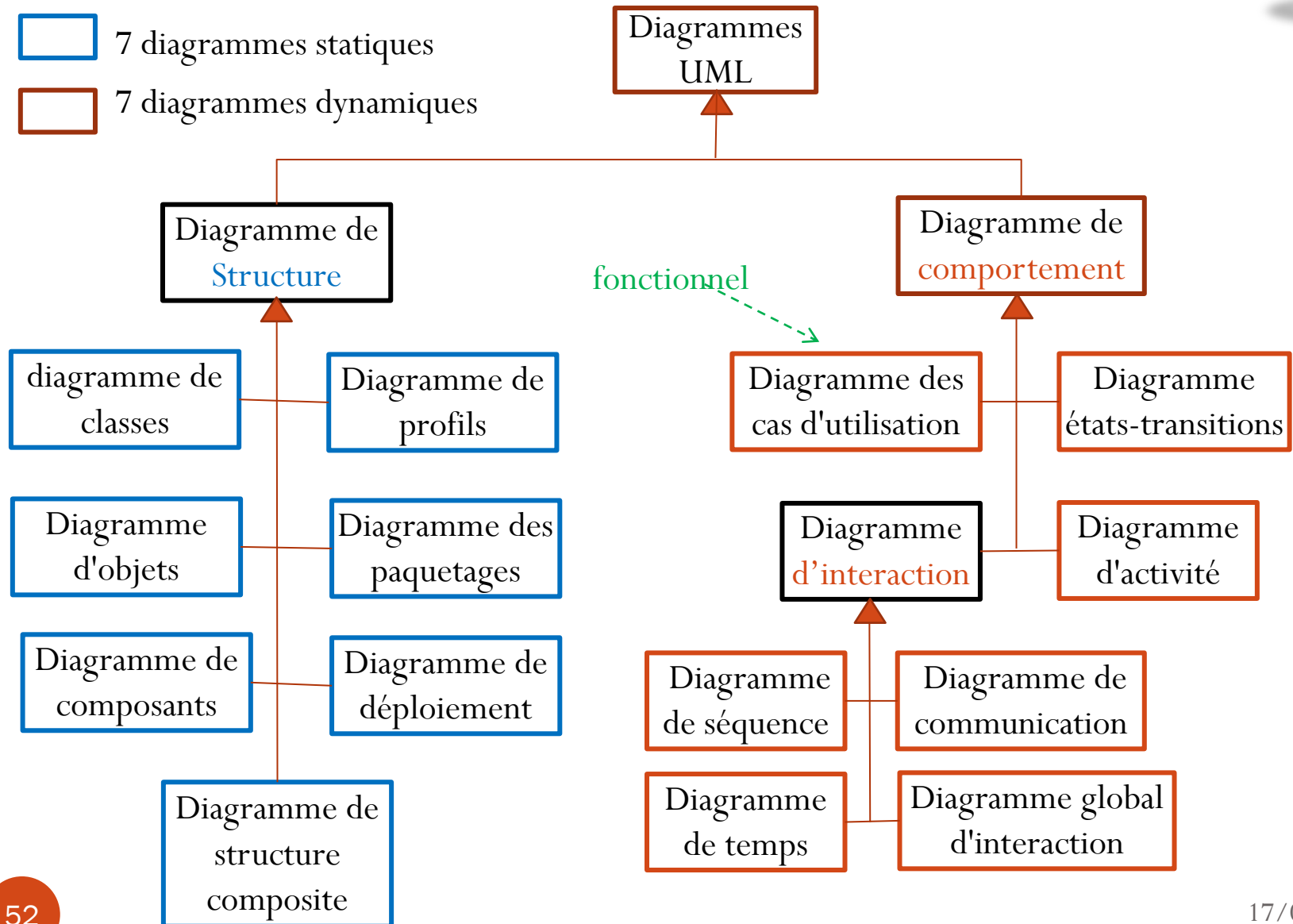
Plusieurs modèles (Vues/niveaux d'abstractions) 1 / 3

- Modèles statique vs Dynamique
 - Statiques → Décrire les aspects structurels
 - Dynamique → Décrire les comportements et les interactions

Modèle statique/ dynamique (diagrammes)



-  7 diagrammes statiques
-  7 diagrammes dynamiques



Plusieurs modèles (Vues/niveaux d'abstractions) 2/3

- **Modèles destinés à différents utilisateurs (phases du cycle de vie)**
 - Pour l'utilisateur et l'analyste → décrire **le quoi**: les **grandes fonctions** du système (contexte, entités du domaine :métiers) (**modèle d'analyse** → décrit le problème sans spécifier la solution)
 - Pour les concepteurs/développeurs → Décrire **le comment**: choix techniques (**modèle de conception** → décrit la solution)

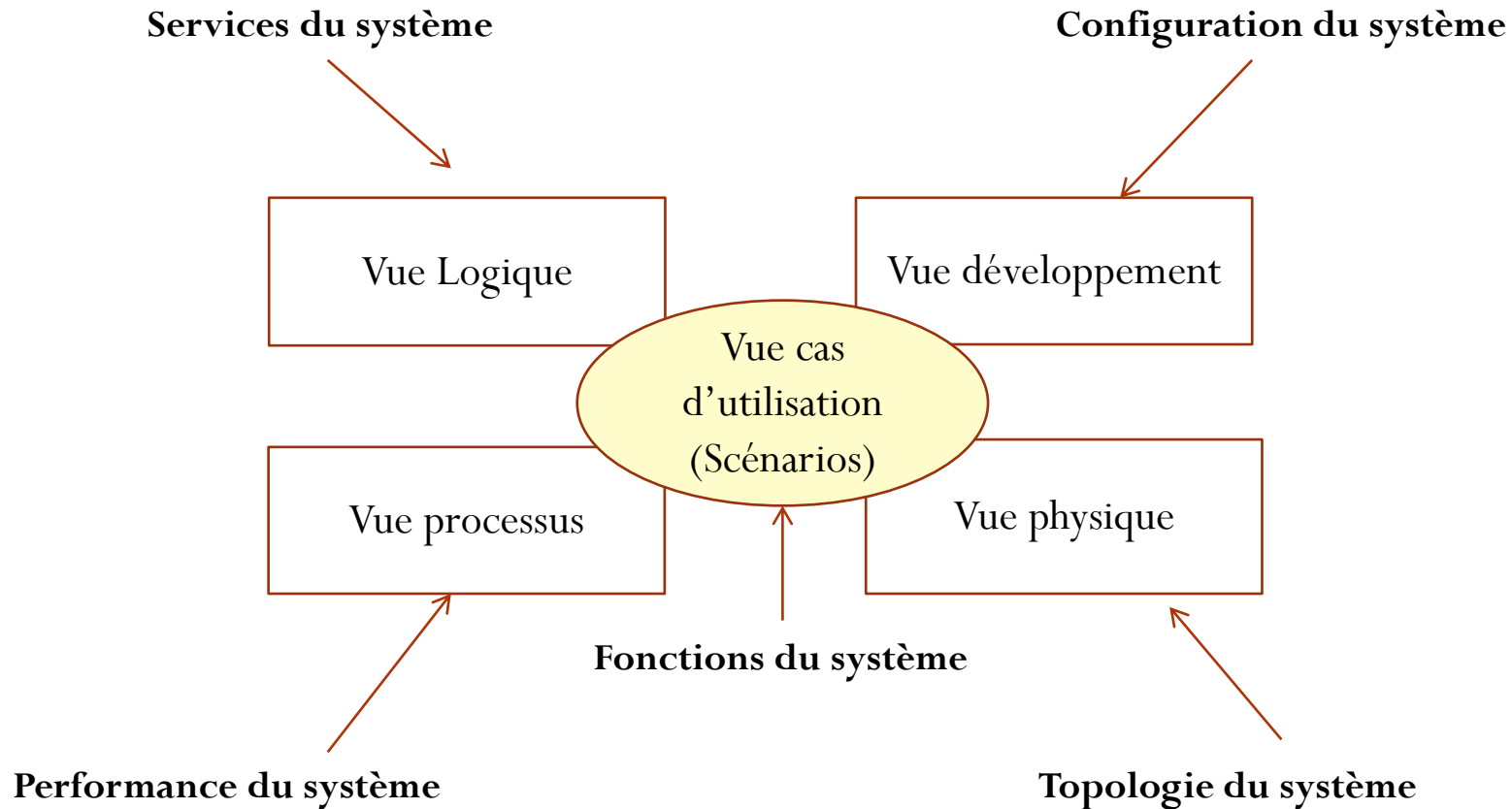
Plusieurs diagrammes sont en commun entre ces deux modèles mais expriment des niveaux de granularité (abstraction) différents.

- Exp:**
- modèle d'analyse → Diagramme de classes participantes (sans méthodes ni détails de codage) niveau système
 - modèle de conception → Diagramme de classe de conception (complet prêt à être généré en un langage de programmation)

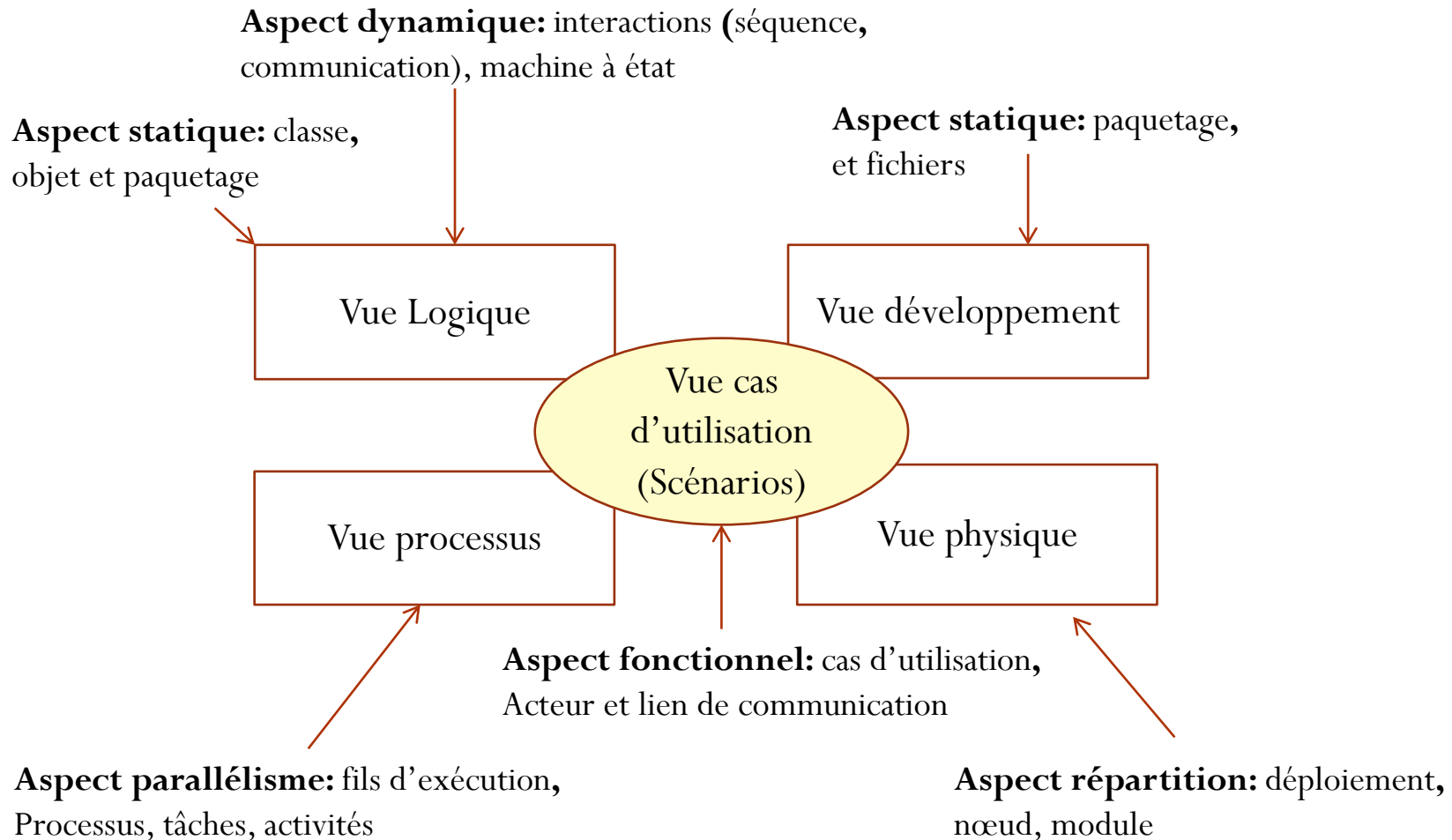
Plusieurs modèles (Vues/niveaux d'abstractions) 3/3

- Modèle: les 4+1 vues : de kruchten
 - une vue **logique** / **processus** → fonctionnement du système
 - une vue **développement** / **physique** → Architecture du système
 - le tout guidé par la vue des **cas d'utilisation** qui représente les besoins des utilisateurs (fonctions métier du système)

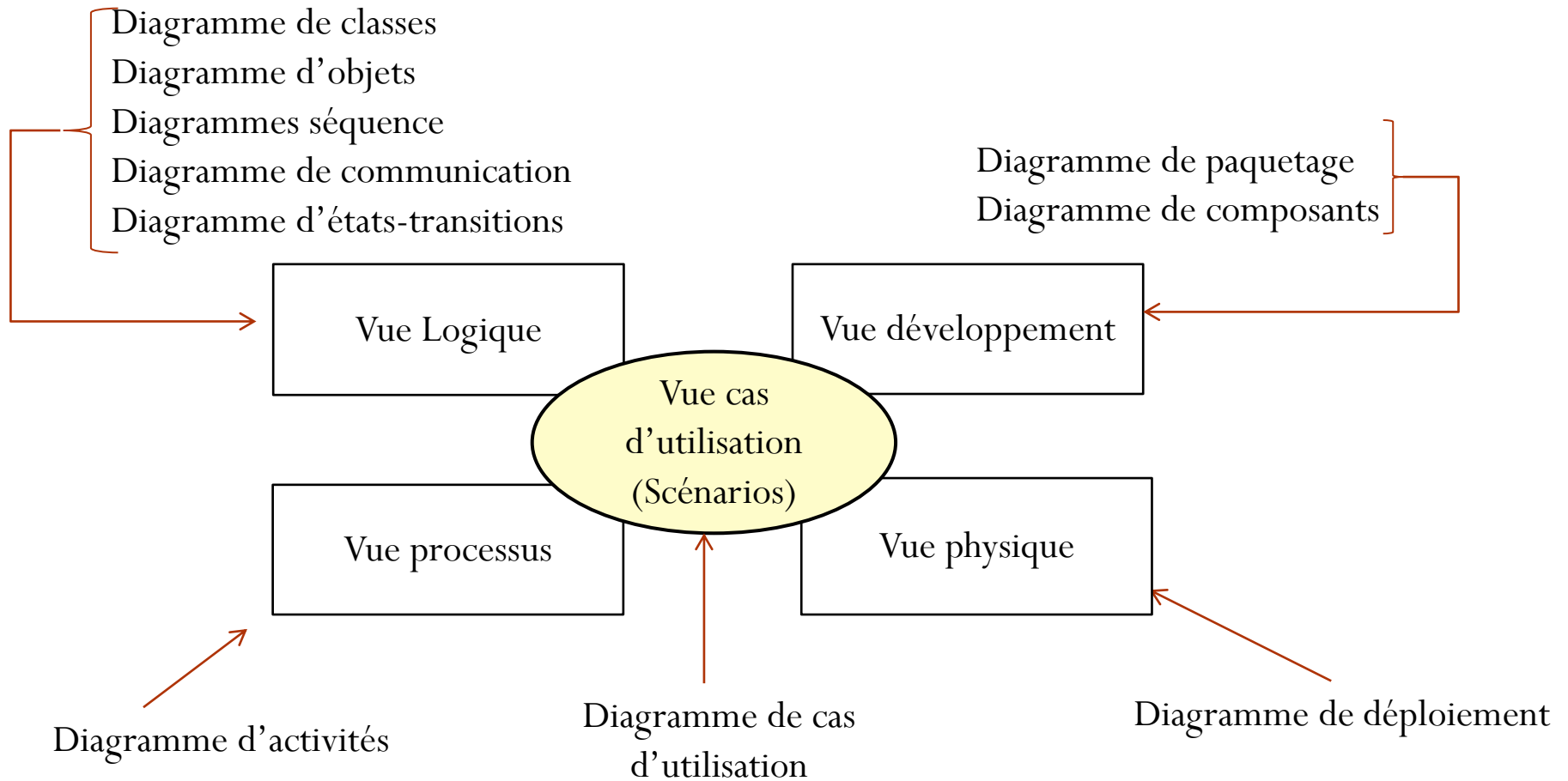
Les 4+1 vues d'un système, de Kruchten



Les 4+1 vues d'un système, de Kruchten



Les 4+1 vues (Diagrammes)



Les 4+1 vues (Diagrammes et aspects)



bleu Statique

Rouge Dynamique

● Analyse

● Conception

Diagramme de classes

Diagramme d'objets

Diagrammes de séquence

Diagramme de communication

Diagramme d'états-transitions

● Diagramme de paquetage

● Diagramme de composants

Vue Logique

Vue développement

Vue cas
d'utilisation
(Scénarios)

Vue processus

Vue physique

Diagramme d'activités

Diagramme de cas
d'utilisation (Aspect
fonctionnel)

● Diagramme de déploiement



- La modélisation permet des représentations, à différents **niveaux d'abstractions** et selon **plusieurs vues**, de l'information nécessaire à la production et à l'évolution d'un logiciel (Indépendamment du processus du développement).
- Le modèle est l'unité de base du développement il permet de :
 - visualiser le système comme il est/était ou devrait être ;
 - valider le modèle vis à vis des clients ;
 - spécifier les structures de données et le comportement du système ;
 - fournir un guide pour la construction du système ;
 - documenter le système et les décisions prises.
- Actuellement le développement des logiciels est fait en utilisant des méthodes orientées objet (méthode = langage de modélisation + démarche + outils)



- UML n'est pas une méthode... mais des principes de conception orientée objet sont sous-jacents
 - aux diagrammes
 - aux façons de les présenter

donc

- difficile de présenter uniquement les diagrammes
 - on parlera aussi de méthodes, de bonnes pratiques
- Différentes façons de voir UML : différentes façons de penser
 - la conception
 - l'objectif et l'efficacité d'un processus de génie logiciel

donc

- essayer de comprendre le point de vue de l'auteur pour chaque publication sur site UML ou un livre (attention aux livres pour un niveau approfondi)

Annexe :

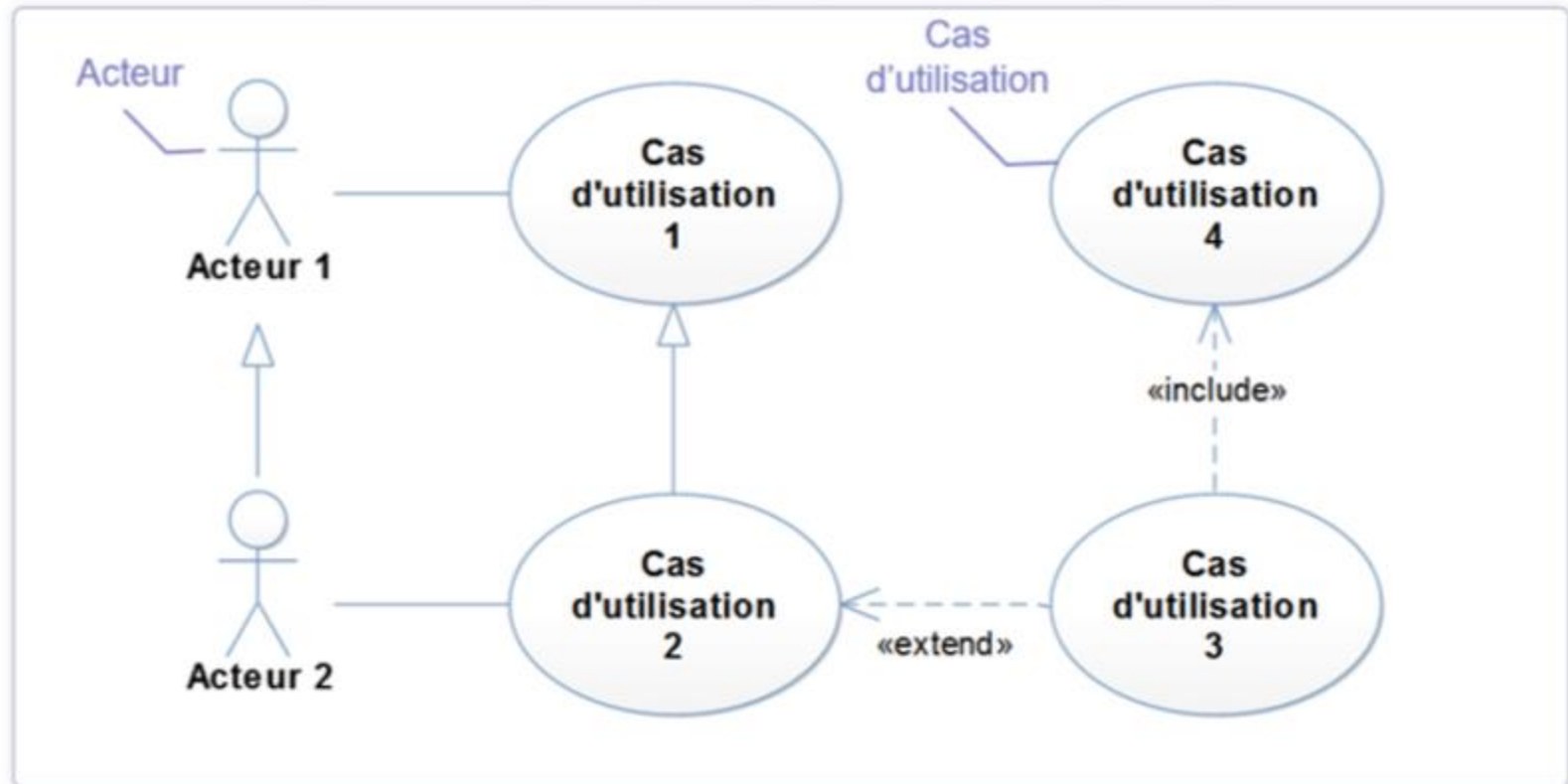
Survol des 14 Diagrammes

Les images sont en partie prises du site <https://blog.khalilmamouny.com/uml-survol-des-14-diagrammes-de-la-version-2-5/>

Annexe : survol des 14 Diagrammes



Diagramme de cas d'utilisation
(Use Case Diagram)

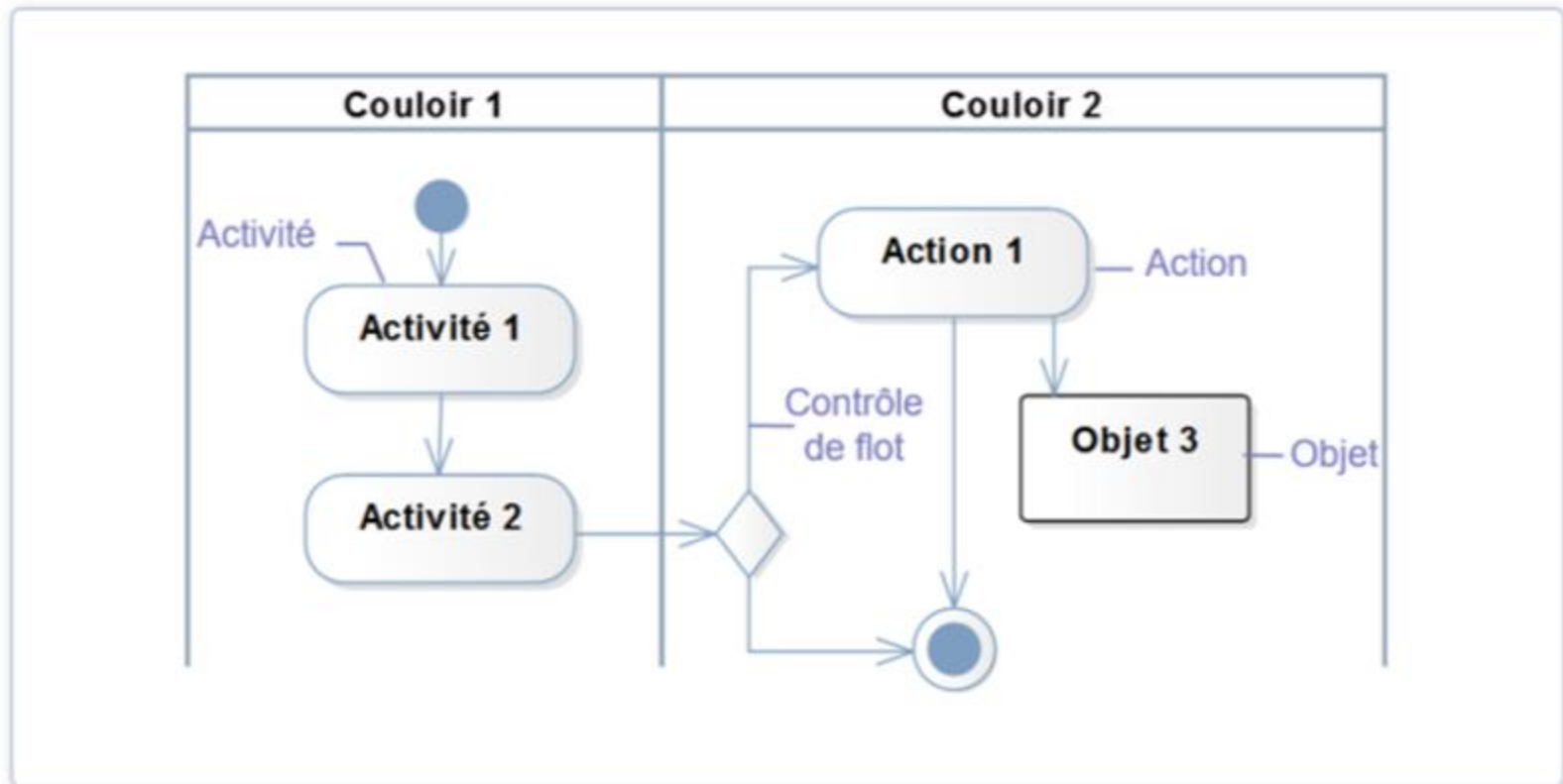


Le diagramme de cas d'utilisation sert à présenter les fonctionnalités d'un système ainsi que les interactions entre ces fonctionnalités et les utilisateurs. Chaque fonctionnalité est présentée sous forme de cas d'utilisation.

Annexe : survol des 14 Diagrammes



Diagramme d'activités (Activity Diagram)

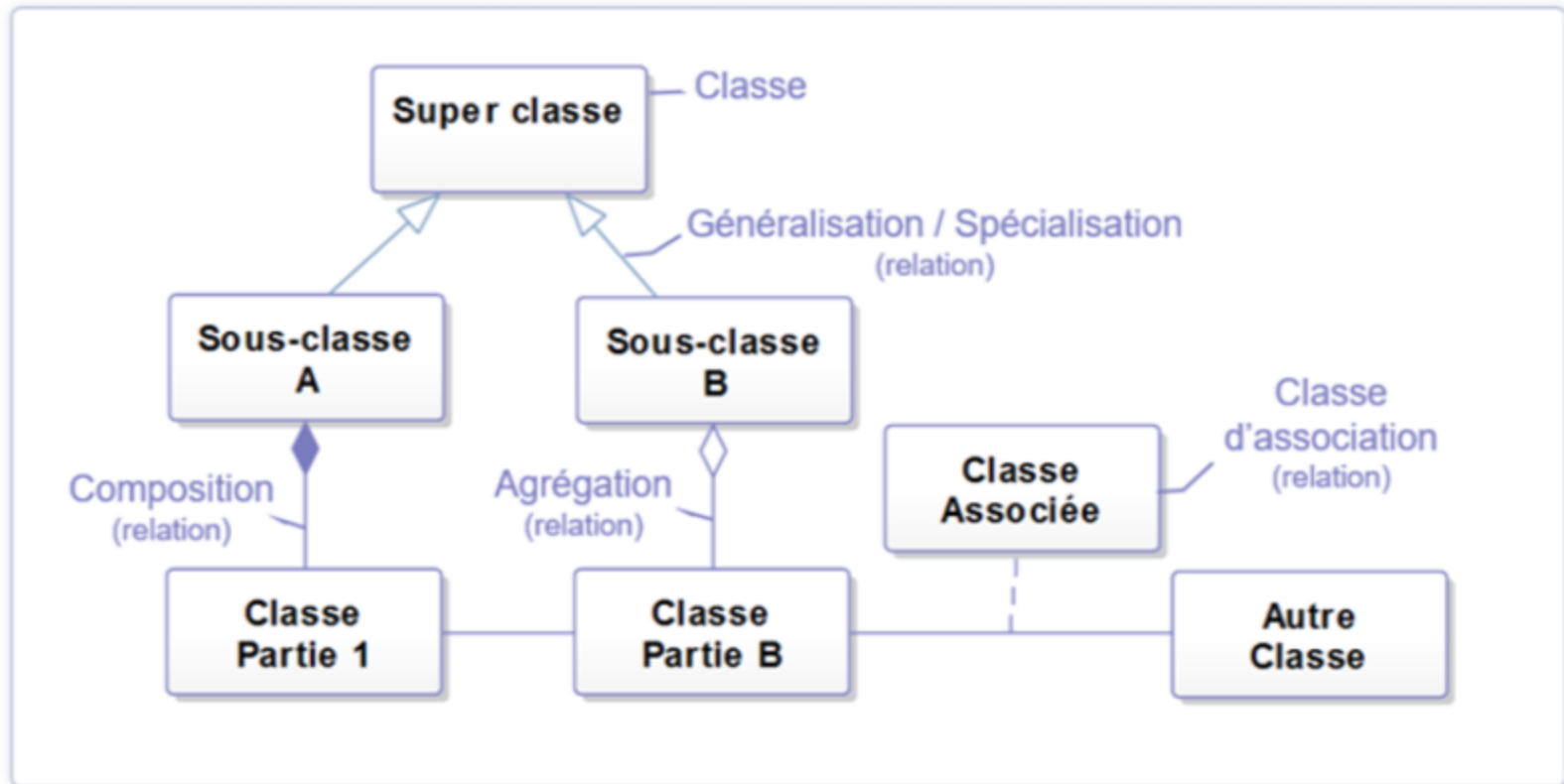


Le diagramme d'activités sert à décrire, à bas niveau, le comportement du système.
Le comportement est présenté sous forme d'enchaînements d'activités,
d'actions et de flots d'objets.

Annexe : survol des 14 Diagrammes



Diagramme de classes (Class Diagram)



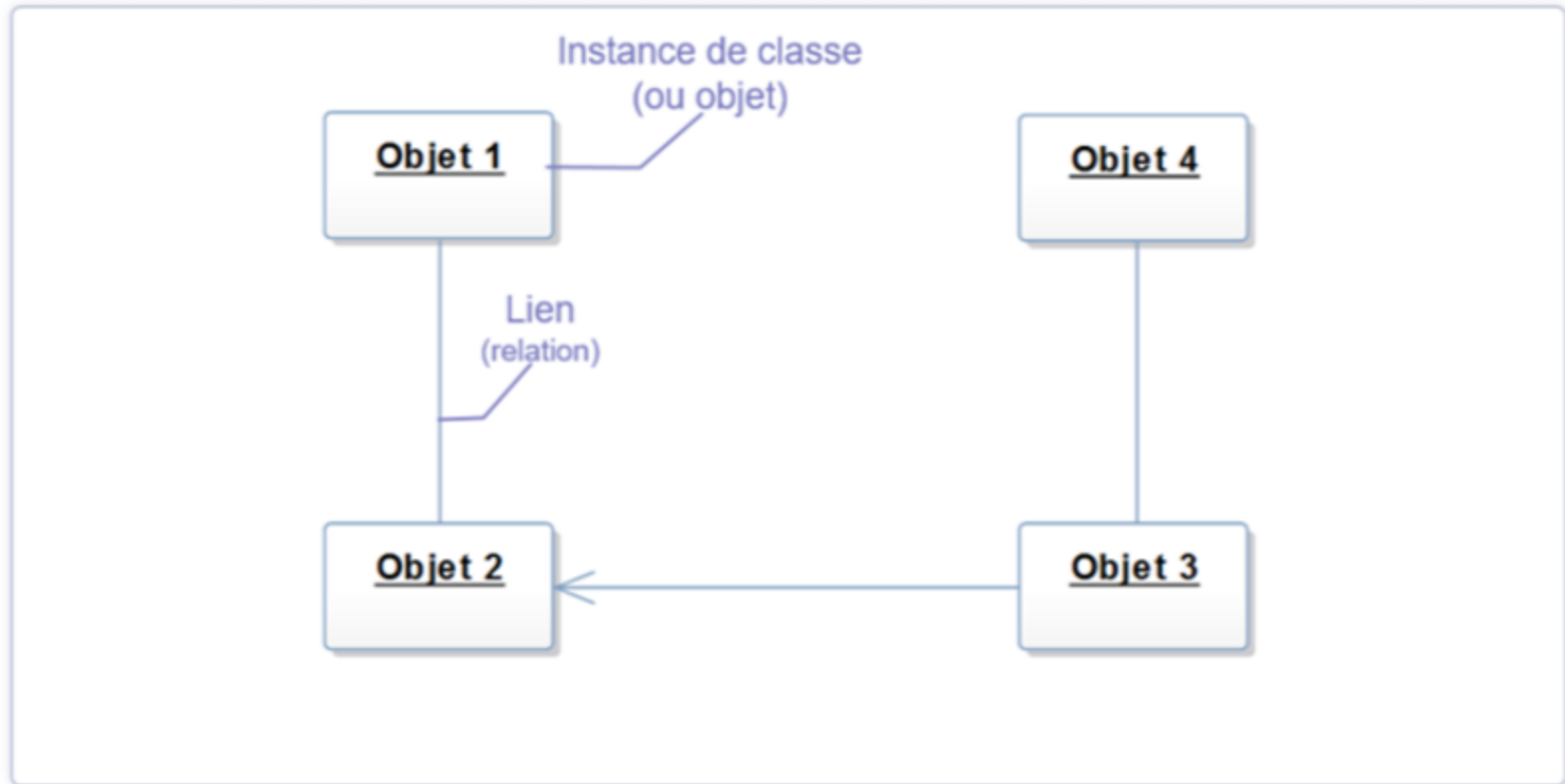
Le diagramme de classes sert à décrire la structure logique d'un système, en la représentant sous forme de classes et de relations entre les classes.

N.B. : Toutes les relations du diagramme de classes ne sont pas représentées dans ce diagramme

Annexe : survol des 14 Diagrammes



Diagramme d'objets (Object Diagram)



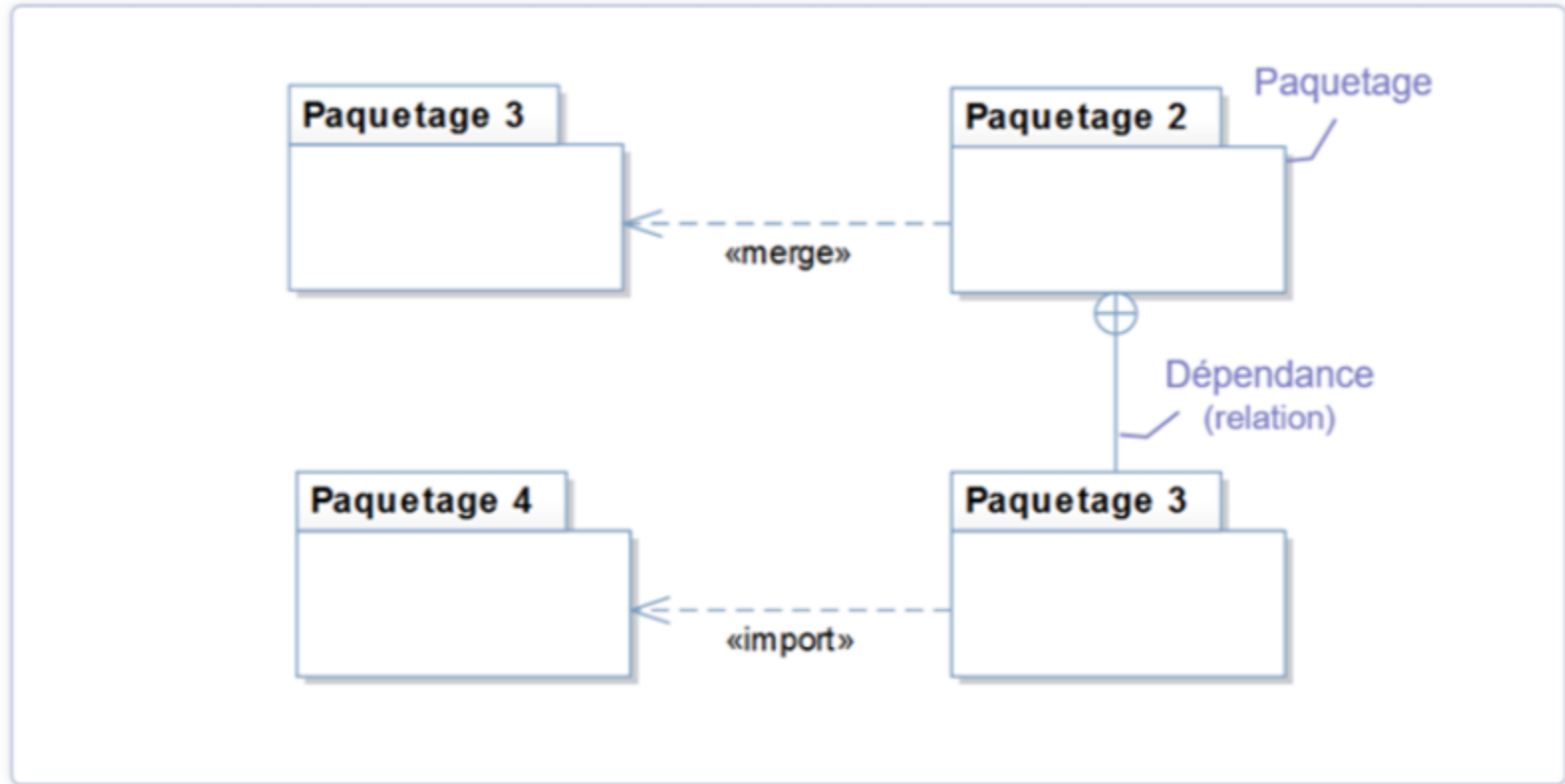
Le diagramme d'objets sert à décrire la structure logique d'un système, dans un contexte précis sous forme d'instances de classes et de relations entre ces instances.

Une instance de classe est dite aussi «objet» et la relation s'appelle également «lien».

Annexe : survol des 14 Diagrammes



Diagramme de paquetages (Package Diagram)

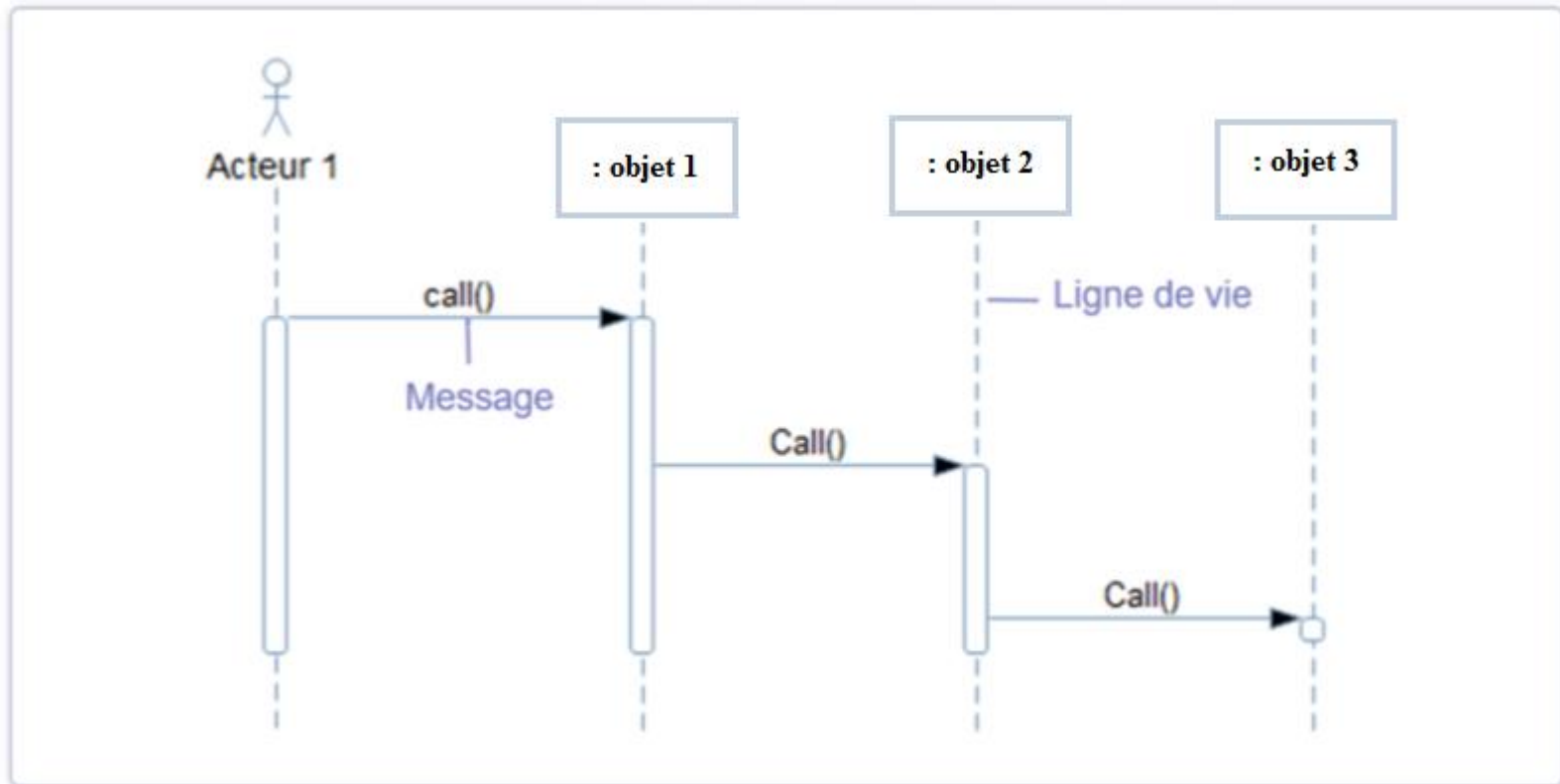


Le diagramme de paquetages sert à décrire les paquetages, leur contenu; et, leurs interrelations.

Annexe : survol des 14 Diagrammes



Diagramme de séquence
(Sequence Diagram)

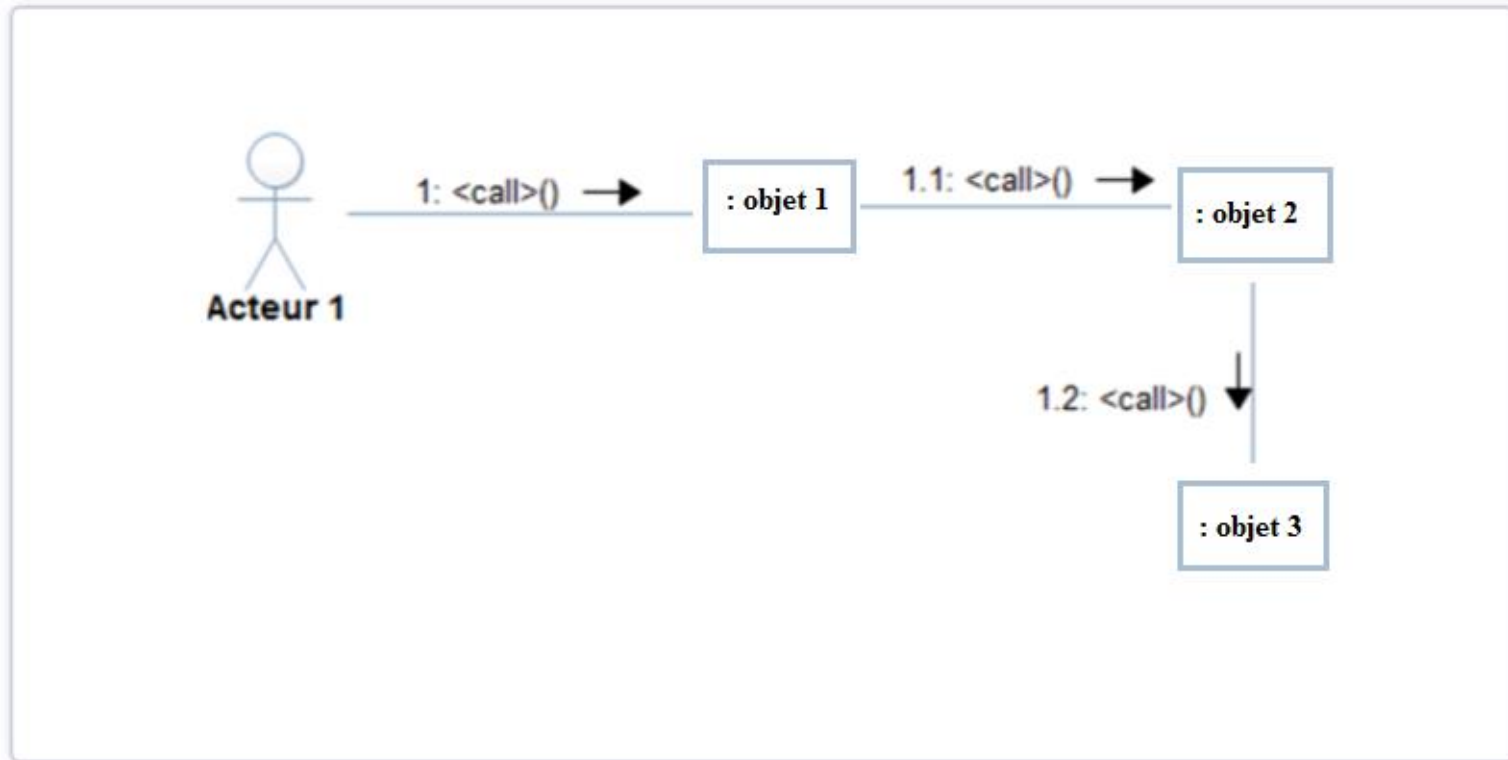


Le diagramme de séquence sert à décrire la collaboration entre les éléments (lignes de vie) d'un système. La collaboration est présentée sous forme d'échanges de messages entre les lignes de vie en mettant en évidence la chronologie de ces échanges.

Annexe : survol des 14 Diagrammes



Diagramme de communication (Communication Diagram)

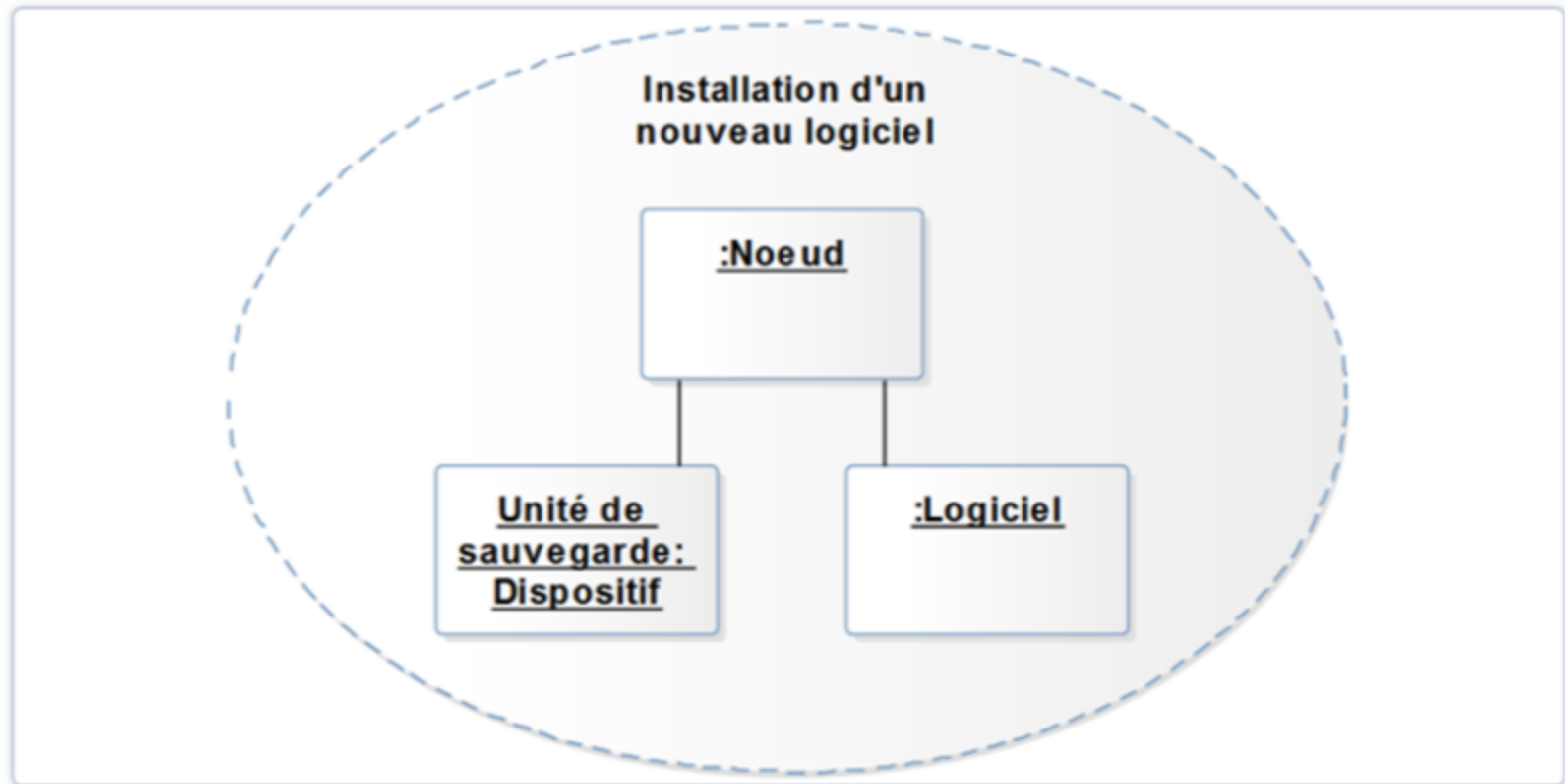


Le diagramme de communication sert à décrire la collaboration entre, d'une part, les éléments constituant un système (par exemple, les objets) et, d'autre part, ces éléments et les acteurs. La collaboration est présentée sous forme d'échanges de messages entre les éléments.

Annexe : survol des 14 Diagrammes



Diagramme de structure composite (Composite Structure Diagram)

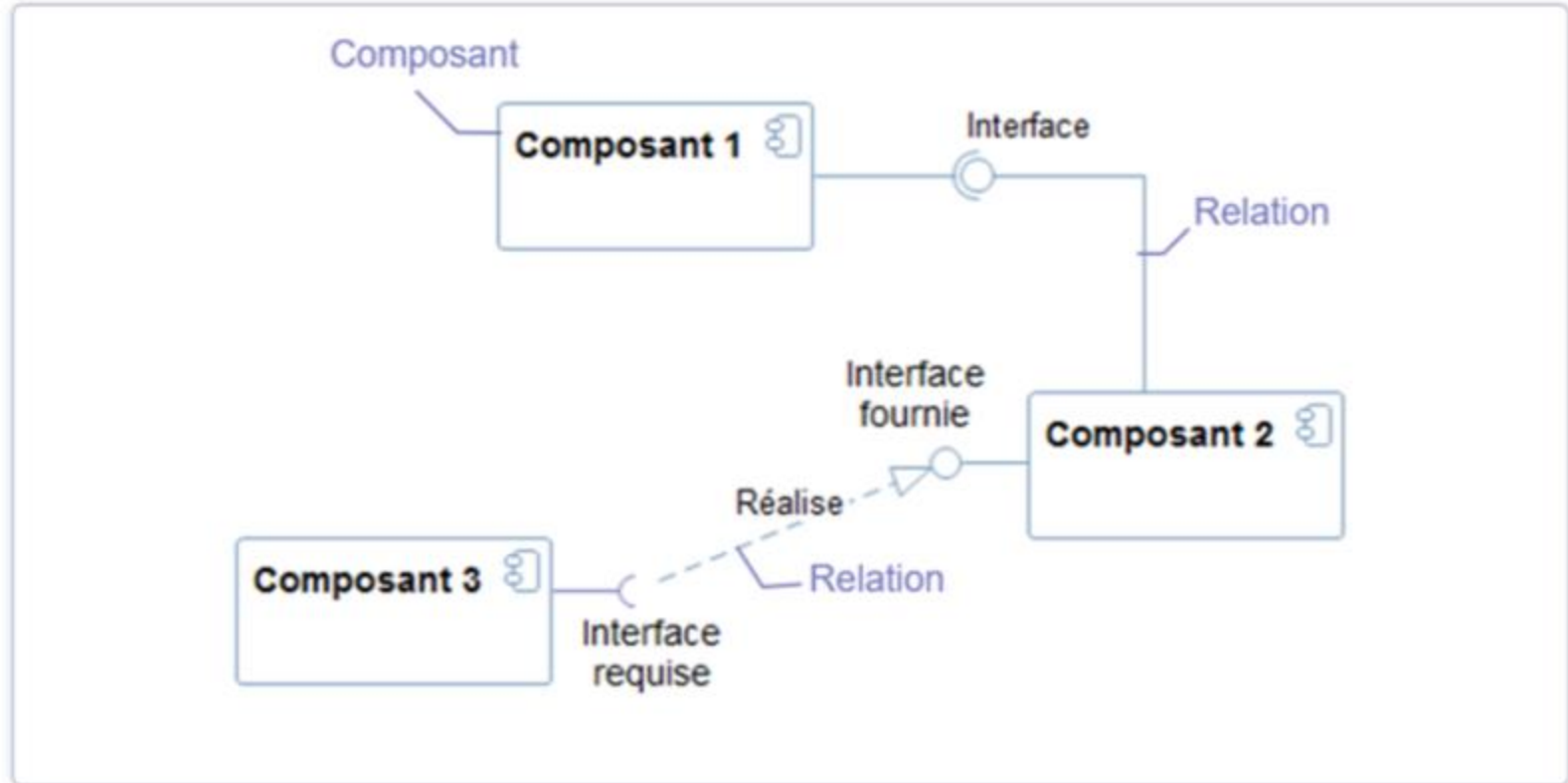


Ce diagramme de structure composite sert à décrire la fonction d'un élément structurel composite en décrivant visuellement sa structure interne.

Annexe : survol des 14 Diagrammes



Diagramme de composants (Component Diagram)

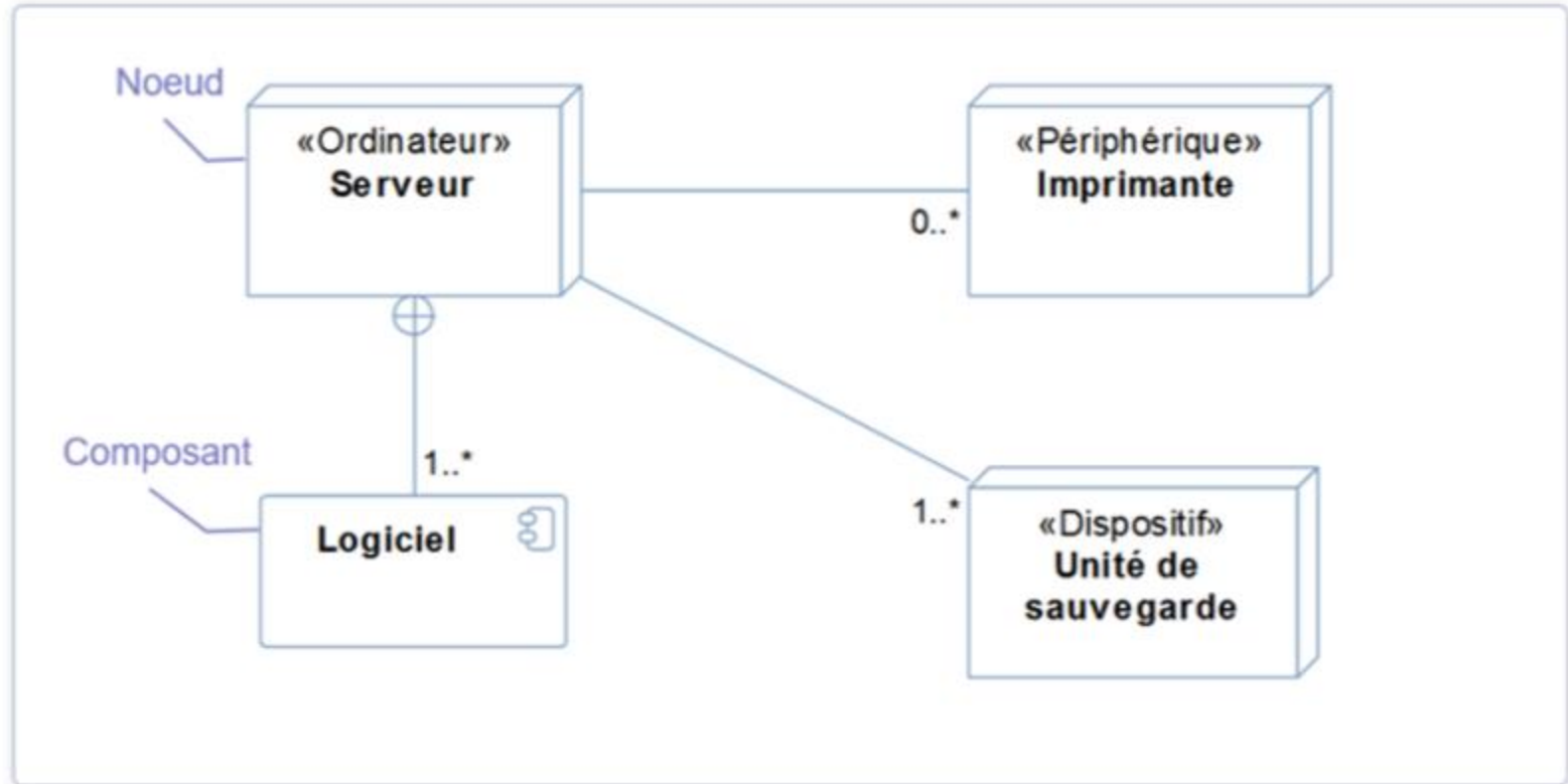


Le diagramme de composants sert à décrire comment un logiciel est structuré.
La structure est présentée sous forme de composants et
de relations entre les composants.

Annexe : survol des 14 Diagrammes



Diagramme de déploiement
(Deployment Diagram)



Le diagramme de déploiement sert à décrire comment, et dans quels nœuds, les composants et les autres artéfacts sont ou seront déployés.

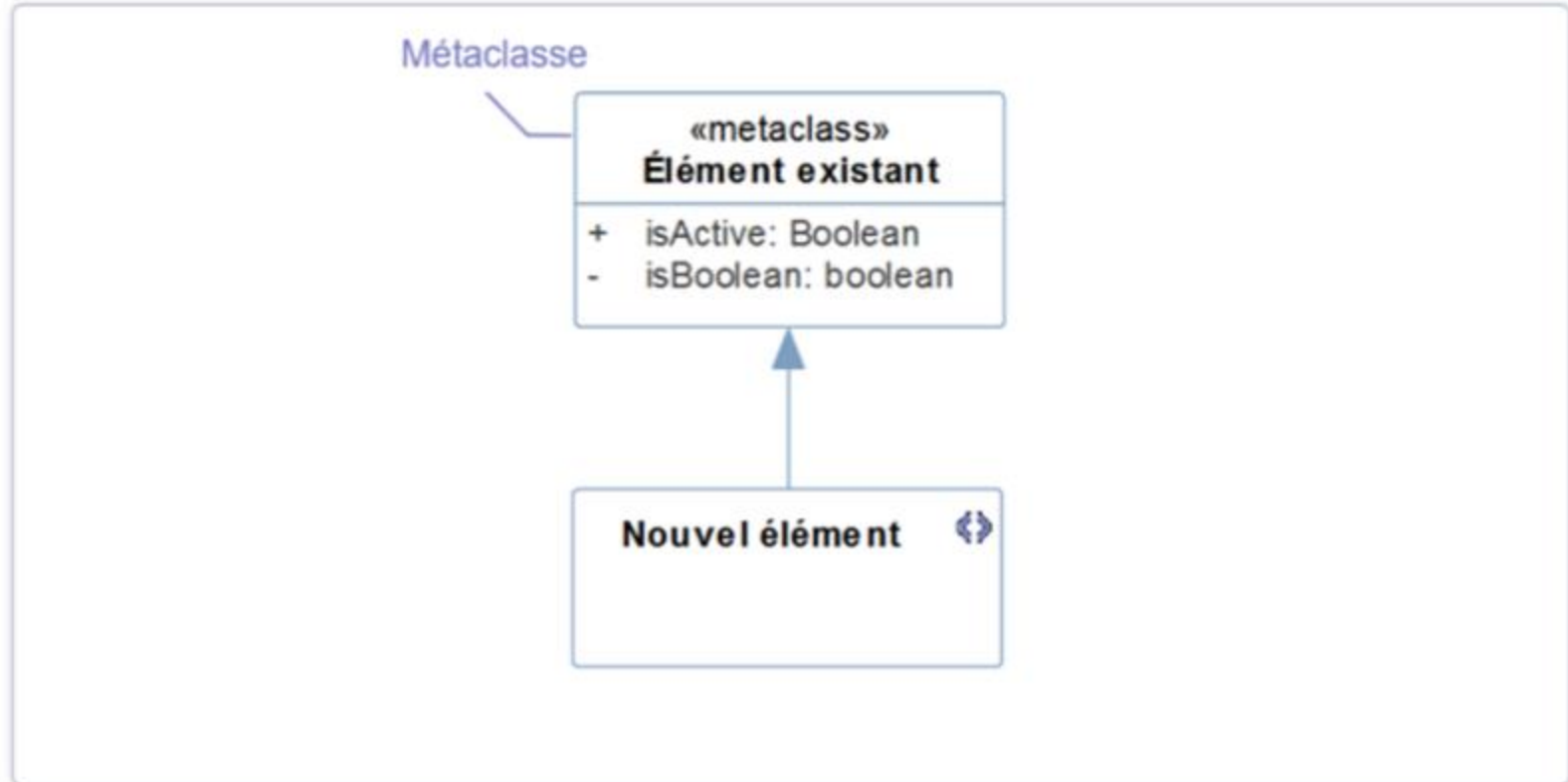
Un nœud étant un dispositif matériel.

Ac

Annexe : survol des 14 Diagrammes



Diagramme de profils (Profile Diagram)

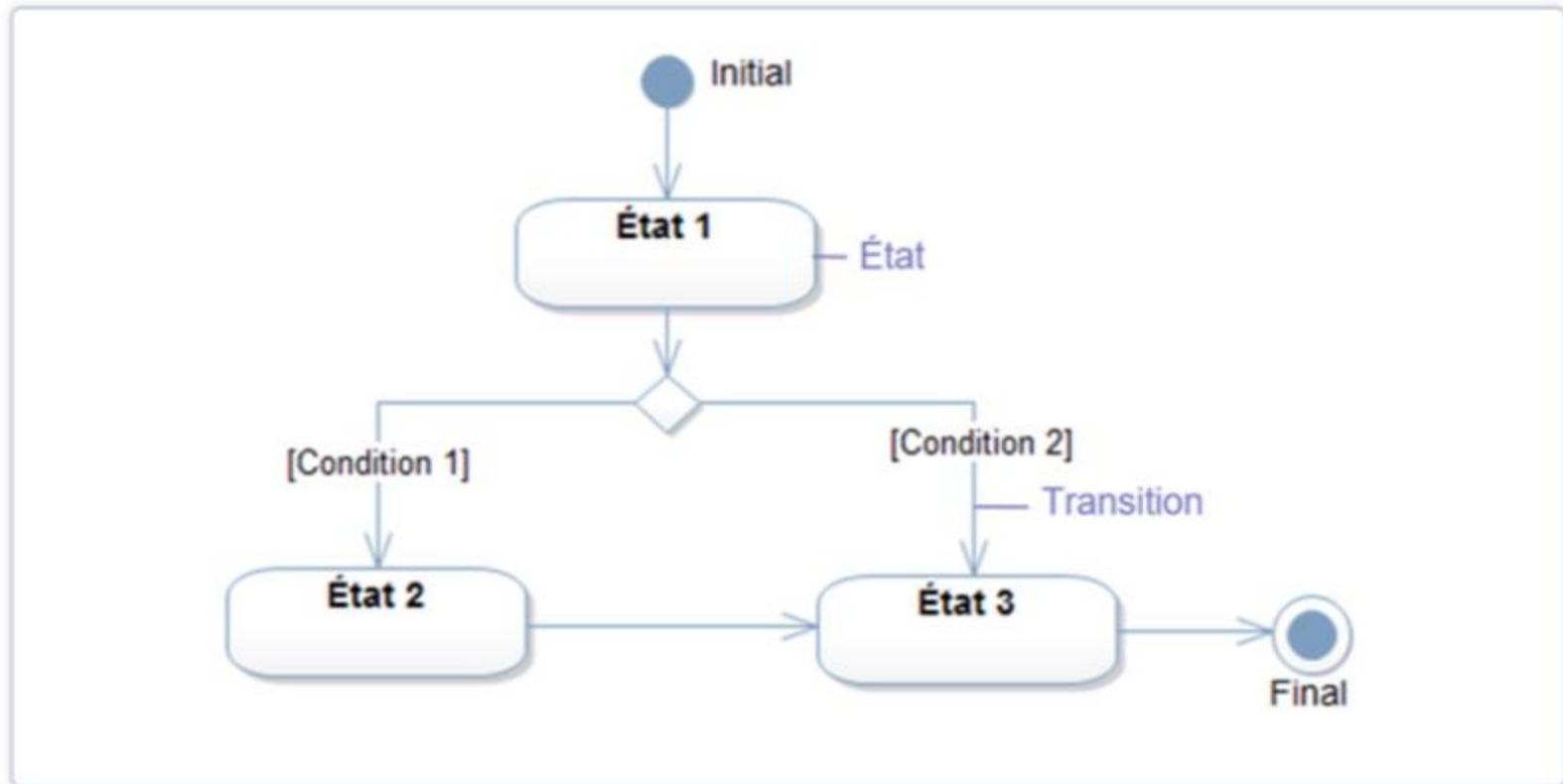


Le diagramme de profils sert à étendre et à spécialiser la notation UML en permettant au modélisateur de créer de nouveaux stéréotypes et d'adapter des règles à un domaine donné. Que le domaine soit d'affaires ou technique.

Annexe : survol des 14 Diagrammes



Diagramme d'états-transitions
(State Machine Diagram)

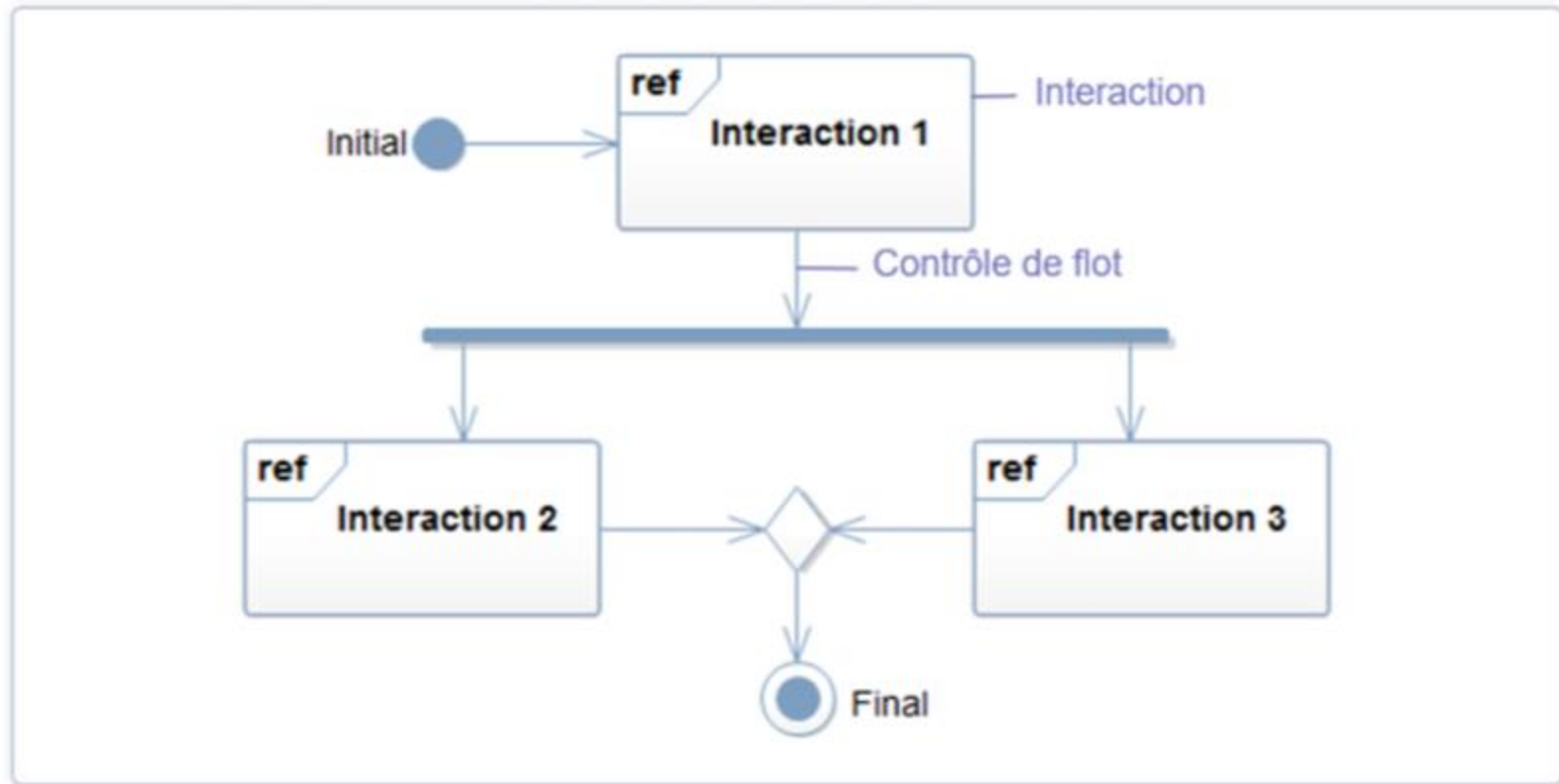


Le diagramme d'états-transition sert à décrire le comportement interne d'un élément (par exemple, un objet). Le comportement est présenté sous forme d'un graphe constitué d'états et de transitions d'états.

Annexe : survol des 14 Diagrammes



Diagramme global d'interactions
(Interaction Overview Diagram)

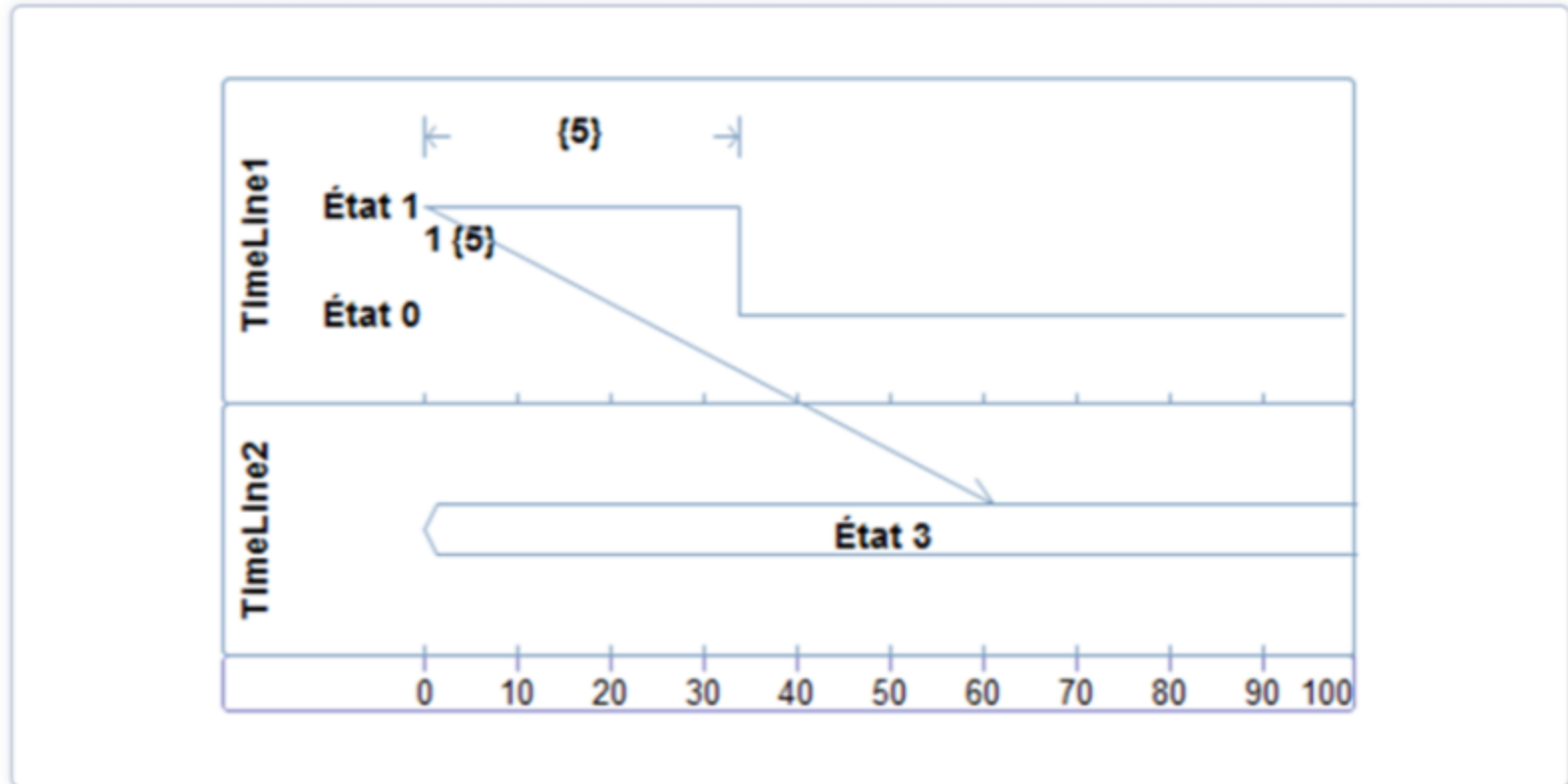


Le diagramme global d'interactions sert à décrire le comportement, à haut niveau, d'un système. Le comportement est présenté sous forme de séquences d'interactions entre les objets. Chaque interaction est décrite séparément et en détail dans un diagramme de comportement approprié.

Annexe : survol des 14 Diagrammes



Diagramme de temps
(Timing Diagram)



Le diagramme de temps sert à décrire les états des éléments et à spécifier les messages échangés entre ces éléments dans le temps.