



Introduction to UML

Support de cours
Nicolas FIGAY
26 Octobre 2022

Objectifs

- Comprendre pourquoi UML a été créé , à quoi il sert et à qui il sert
- Comprendre le langage, les concepts sous jacents, ces constructions
- Faire le lien avec le développement d'un système informatisé, en fonction du rôle des acteurs impliqués et de la phase de vie du projet de développement
- Identifier les modèles nécessaires pour certaines des tâches associées à ces rôle
- Savoir choisir les outils, diagrammes et types d'élément du modèle à utiliser pour créer ces modèles

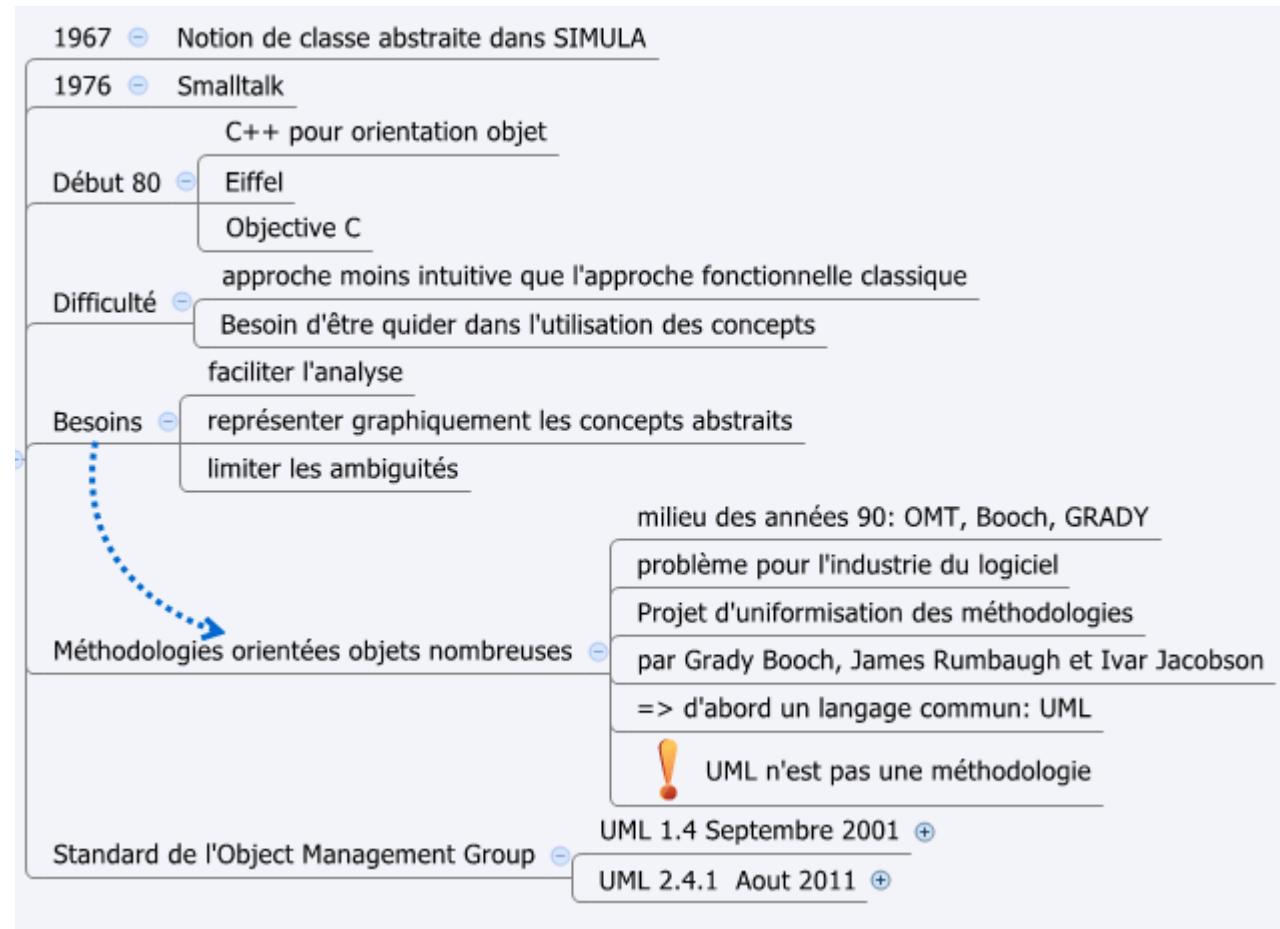
Ce que doit savoir faire l'étudiant après l'intervention

- Dans la phase d'analyse des besoins
 - o Créer un modèle entité relation avec les diagrammes appropriés, en indiquant les restrictions à appliquer
 - o Créer un modèle permettant de capturer ce que doivent les utilisateurs pouvoir faire avec le système et selon quelles interactions, avec les diagrammes appropriés
 - o Créer un modèle permettant de capturer les besoins d'interaction avec des systèmes existants, avec les diagrammes appropriés
 - o Créer un modèle permettant de mettre en contexte le cas d'utilisation dans le cadre d'un processus d'entreprise, avec les diagrammes appropriés
- Dans la phase de conception
 - o Modéliser l'ensemble des composants de la plateforme d'exécution et l'architecture technique associé avec les diagrammes appropriés
 - o Modéliser l'ensemble des composants fonctionnels de l'application nécessaire à la réalisation des use cases avec les diagrammes appropriés
 - o Modéliser les liens entre ces fonctions et les composants de l'architecture technique qui permet de les réaliser
- Dans la phase de développement
 - o Spécifier via un modèle un composant à réaliser par des développeurs
 - o Modéliser comment ce composant devra être déployé pour le développeur et l'intégrateur du composant
 - o Identifier les modèles à réutiliser pour la documentation de l'utilisateur, de l'administrateur métier, de l'administrateur technique, de l'architecte ou du développeur qui doit faire évoluer l'application

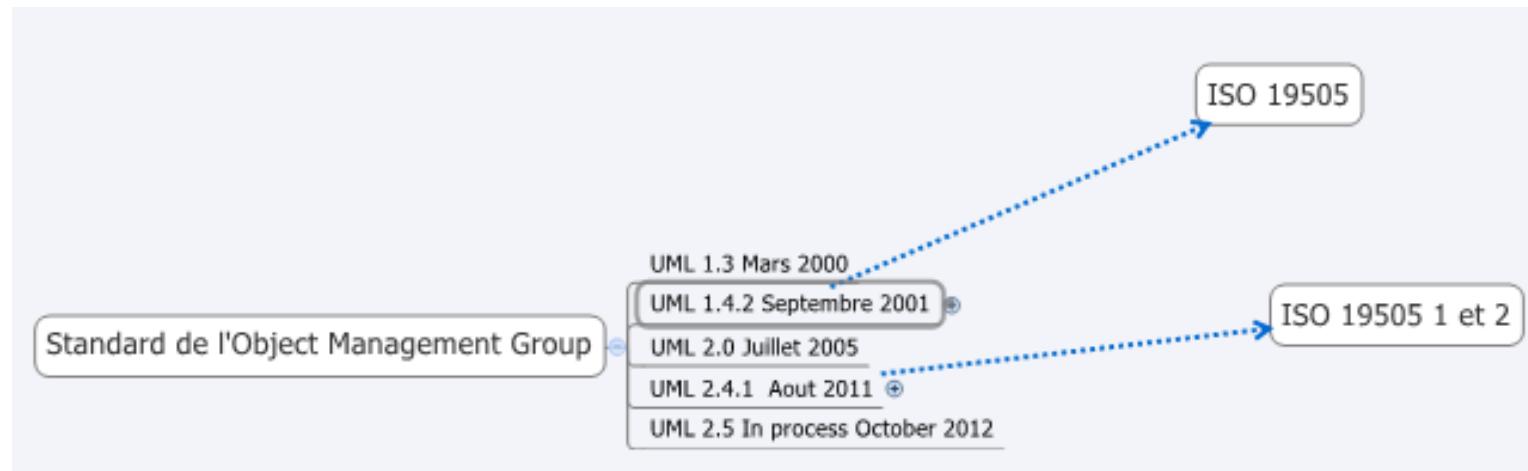
Ce que doit savoir l'étudiant après l'intervention

- Connaitre les origines de UML, en lien avec les paradigmes « objet » et « composant ». Avoir compris ce que sont ces paradigmes
- Savoir ce qui motive l'utilisation de UML dans l'industrie logicielle
- Connaitre les divers types de diagrammes d'UML2, et savoir dire s'ils sont statiques ou dynamique
- Savoir faire la distinction entre le modèle, les éléments du modèle et les diagrammes
- Connaitre les éléments de modélisation de base
- Connaitre l'ordre de grandeur du nombre de type d'élément de UML2, par rapport à d'autres langages
- Savoir les liens d'UML2 avec les spécifications de l'OMG et les normes: UML2, MDA, SPEM, MOF, OCL, XMI, QVT
- Savoir où aller chercher les spécifications de l'OMG et ce que l'on y trouve
- Savoir en quoi UML2 supporte l'ingénierie par les modèles pour les applications à base de composant

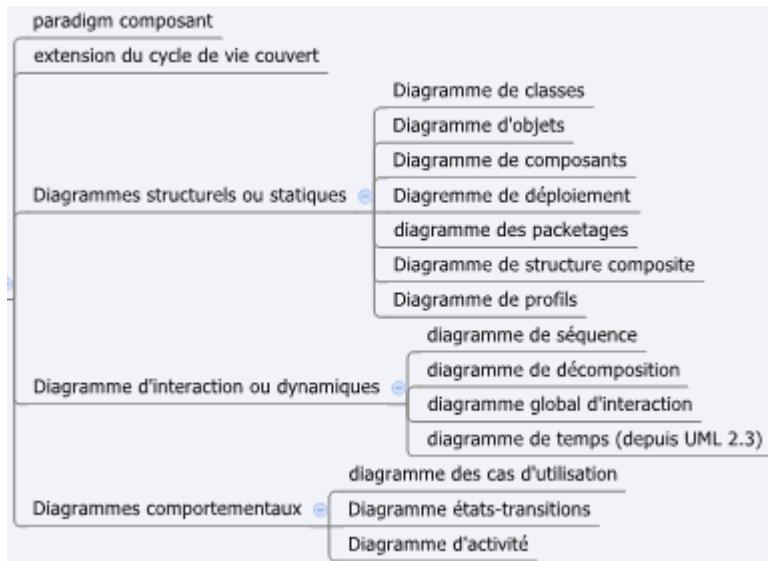
Historique



Standards

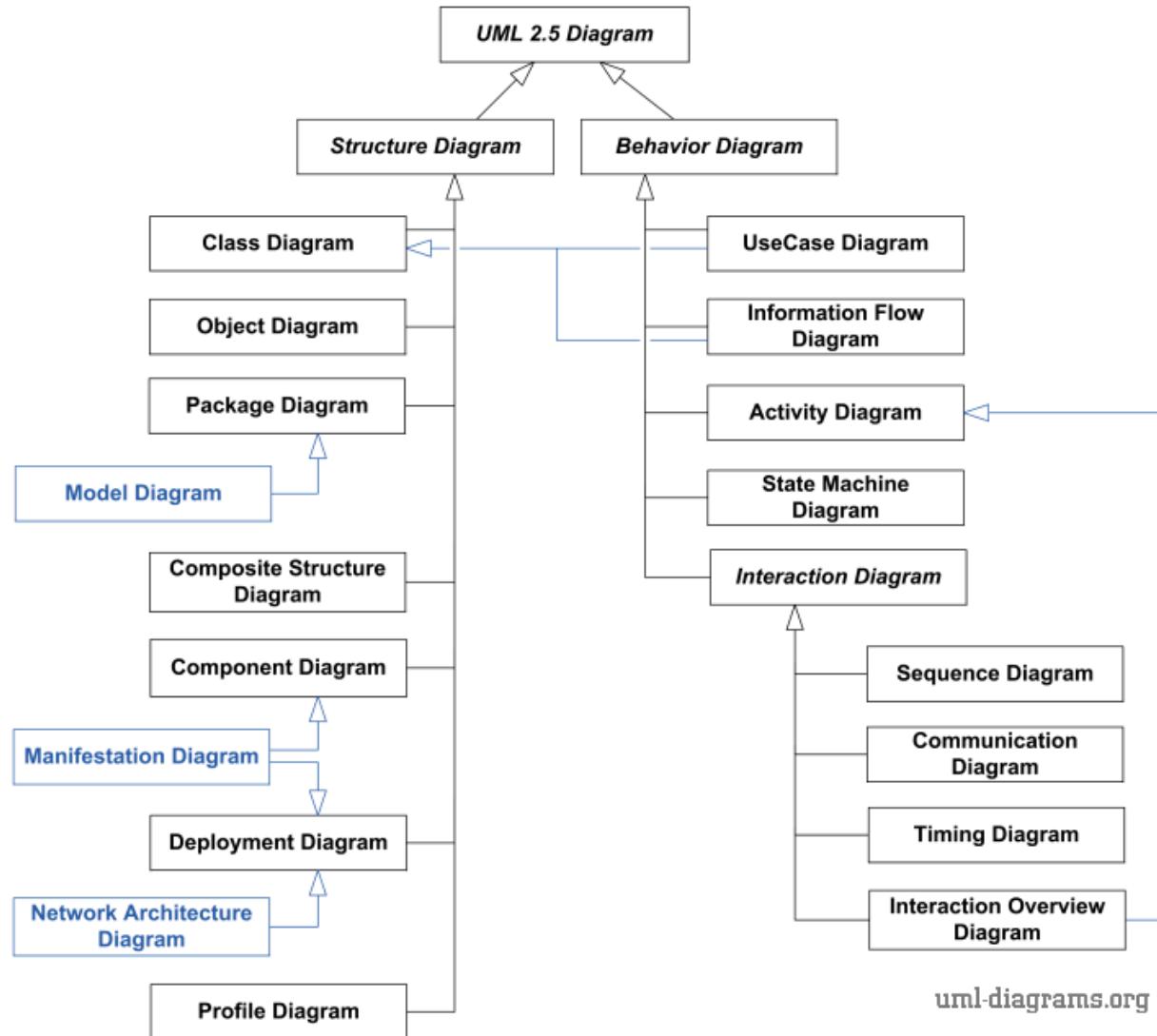


Les divers diagrammes en UML 2



Présentation des diagrammes UML 2.5

- La spécification UML définit deux types principaux de diagramme UML : les diagrammes de structure et les diagrammes de comportement.
- Les diagrammes de structure montrent la structure statique du système et de ses parties à différents niveaux d'abstraction et de mise en œuvre et comment ils sont liés les uns aux autres. Les éléments d'un diagramme de structure représentent les concepts significatifs d'un système et peuvent inclure des concepts abstraits, réels et de mise en œuvre.
- Les diagrammes de comportement montrent le comportement dynamique des objets dans un système, qui peut être décrit comme une série de changements apportés au système au fil du temps.
- Les diagrammes UML 2.5 peuvent être classés hiérarchiquement comme indiqué dans l'image à gauche.



Diagrammes de structure

Un diagramme de structure

- montre la **structure statique du système** et de **ses parties sur différents niveaux d'abstraction et de mise en œuvre** et comment ces parties sont liées les unes aux autres. Les éléments d'un diagramme de structure représentent les **concepts significatifs d'un système** et peuvent inclure des concepts **abstraits, réels et de mise en œuvre**.
- n'utilisent **pas de concepts liés au temps**, ne montre **pas les détails du comportement dynamique**. Cependant, il peut montrer des relations avec les comportements des classificateurs présentés dans les diagrammes de structure.
- Le **diagramme de classes** est un diagramme de structure statique qui décrit la structure d'un système au niveau des classificateurs (classes, interfaces, etc.). Il montre quelques classificateurs du système, sous-système ou composant, différentes relations entre classificateurs, leurs attributs et opérations, contraintes.

Le diagramme d'objets

- était défini dans la spécification UML 1.4.2, désormais obsolète, comme "un graphique d'instances, y compris les objets et les valeurs de données."
- est une instance d'un diagramme de classes ; il montre un instantané de l'état détaillé d'un système à un point dans le temps." Il a également déclaré que le diagramme d'objets est "un diagramme de classes avec des objets et aucune classe".
- La spécification UML 2.4 ne fournit simplement aucune définition de diagramme d'objets.
- Certains éléments majeurs du diagramme d'objets sont des spécifications d'instances nommées et anonymes pour les objets, des emplacements avec des spécifications de valeur et des liens (instances d'association).

Diagrammes de structure

- Le diagramme de package montre les packages et les relations entre les packages.
- Le diagramme de modèle est un diagramme de structure auxiliaire UML qui montre une abstraction ou une vue spécifique d'un système, pour décrire les aspects architecturaux, logiques ou comportementaux du système. Il pourrait montrer, par exemple, l'architecture d'une application multicouche (ou multicouche) - modèle d'application multicouche.
- Le diagramme de structure composite peut être utilisé pour montrer la structure interne d'un classificateur ou le comportement A d'une collaboration
- Les diagrammes de structure interne montrent la structure interne d'un classificateur - une décomposition du classificateur en ses propriétés, ses parties et ses relations.
- Le diagramme d'utilisation de la collaboration montre les objets d'un système coopérant les uns avec les autres pour produire un certain comportement du système.
- Le diagramme de composants montre les composants et les dépendances entre eux. Ce type de diagrammes est utilisé pour le développement basé sur des composants (CBD), pour décrire des systèmes avec une architecture orientée services (SOA).
- Le diagramme de déploiement montre l'architecture du système en tant que déploiement (distribution) d'artefacts logiciels vers des cibles de déploiement.

Notez que les composants ont été directement déployés sur les nœuds dans les diagrammes de déploiement UML 1.x. Dans UML 2.x, les artefacts sont déployés sur les nœuds et les artefacts peuvent manifester (implémenter) des composants. Les composants sont déployés sur les nœuds indirectement via des artefacts.

Diagrammes de structure

- Le diagramme de déploiement de niveau de spécification (également appelé niveau de type) présente une vue d'ensemble du déploiement d'artefacts sur des cibles de déploiement, sans faire référence à des instances spécifiques d'artefacts ou de nœuds.
- Le diagramme de déploiement au niveau de l'instance montre le déploiement d'instances d'artefacts sur des instances spécifiques de cibles de déploiement. Il peut être utilisé, par exemple, pour montrer les différences de déploiement dans les environnements de développement, de mise en scène ou de production avec les noms/ID de serveurs ou de périphériques de build ou de déploiement spécifiques.
- Alors que les diagrammes de composants montrent les composants et les relations entre les composants et les classificateurs, et les diagrammes de déploiement - déploiements d'artefacts vers des cibles de déploiement, certains diagrammes intermédiaires manquants sont des diagrammes de manifestation à utiliser pour montrer la manifestation (mise en œuvre) de composants par des artefacts et la structure interne des artefacts.
Étant donné que les diagrammes de manifestation ne sont pas définis par la spécification UML 2.4, la manifestation de composants par des artefacts pourrait être montrée à l'aide de diagrammes de composants ou de diagrammes de déploiement.

Diagrammes de structure

- ❑ Les diagrammes de déploiement peuvent également être utilisés pour montrer l'architecture de réseau logique ou physique du système. Ce type de diagrammes de déploiement - non formellement défini dans UML 2.5 - pourrait être appelé diagrammes d'architecture de réseau.
- ❑ Le diagramme de profil est un diagramme UML auxiliaire qui permet de définir des stéréotypes personnalisés, des valeurs étiquetées et des contraintes. Le mécanisme de profil a été défini dans UML pour fournir un mécanisme d'extension léger à la norme UML. Les profils permettent d'adapter le métamodèle UML à différentes plates-formes (telles que J2EE ou .NET), ou domaines (tels que la modélisation en temps réel ou de processus métier). Les diagrammes de profil ont été introduits pour la première fois dans UML 2.0.

Diagrammes de comportement

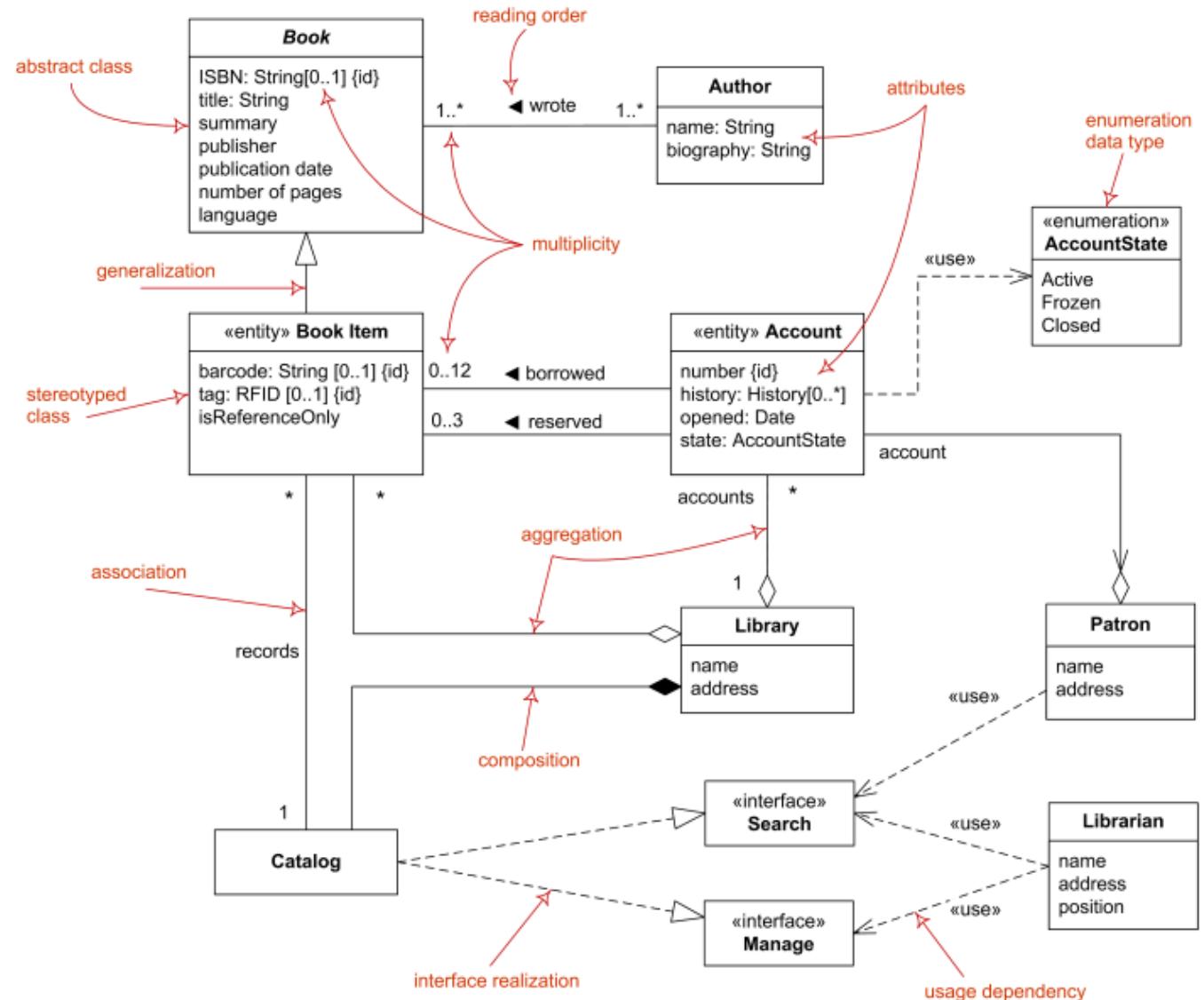
- Les **diagrammes de comportement** montrent le **comportement dynamique des objets dans un système**, qui peut être décrit comme une série de changements apportés au système au fil du temps.
- Les **diagrammes de cas d'utilisation** sont des diagrammes de comportement utilisés pour décrire un ensemble d'actions (cas d'utilisation) que certains systèmes (sujets) doivent ou peuvent effectuer en collaboration avec un ou plusieurs utilisateurs externes du système (acteurs) pour fournir des résultats observables et précieux aux acteurs ou autres parties prenantes du ou des systèmes.
Notez que la spécification UML 2.4 définit également les diagrammes de cas d'utilisation comme une spécialisation des diagrammes de classes (qui sont des diagrammes de structure). Les diagrammes de cas d'utilisation peuvent être considérés comme un cas particulier de diagrammes de classes où les classificateurs sont limités à être des acteurs ou des cas d'utilisation et la relation la plus utilisée est l'association.
- Les **diagrammes d'activité** montre la séquence et les conditions de coordination des comportements de niveau inférieur, plutôt que les classificateurs qui possèdent ces comportements. Ceux-ci sont communément appelés modèles de flux de contrôle et de flux d'objets.
- Les **diagrammes de machine à états** est utilisé pour modéliser un comportement discret à travers des transitions d'états finis. En plus d'exprimer le comportement d'une partie du système, les machines à états peuvent également être utilisées pour exprimer le protocole d'utilisation d'une partie d'un système. Ces deux types de machines à états sont appelés machines à états comportementales et machines à états de protocole.

Diagrammes de comportement

- **diagrammes de séquence**: le type de diagramme d'interaction le plus courant, qui se concentre sur l'échange de messages entre les lignes de vie (objets).
- **diagrammes de synthèse des interactions**: définit les interactions via une variante de diagrammes d'activités d'une manière qui favorise la vue d'ensemble du flux de contrôle. Les diagrammes de vue d'ensemble des interactions se concentrent sur la vue d'ensemble du flux de contrôle où les nœuds sont des interactions ou des utilisations d'interaction. Les lignes de vie et les messages n'apparaissent pas à ce niveau de vue d'ensemble.
- **diagrammes de communication** (appelés diagrammes de collaboration dans UML 1.x): une sorte de diagramme d'interaction, qui se concentre sur l'interaction entre les lignes de vie où l'architecture de la structure interne et comment cela correspond au passage du message est centrale. Le séquençage des messages est donné par un schéma de numérotation de séquence.
- **chronogrammes**: utilisés pour montrer les interactions lorsqu'un objectif principal du diagramme est de raisonner sur le temps. Les chronogrammes se concentrent sur les conditions changeant au sein et entre les lignes de vie le long d'un axe de temps linéaire.

Présentation des diagrammes de classes et d'objets

- Le diagramme de classes est un diagramme de structure UML qui montre la structure du système conçu au niveau des classes et des interfaces, montre leurs caractéristiques, contraintes et relations - associations, généralisations, dépendances, etc.
- Certains types courants de diagrammes de classes sont :
 - diagramme de modèle de domaine ou diagramme de classes d'implémentation
 - Le diagramme d'objets peut être considéré comme un diagramme de classes au niveau de l'instance qui montre les spécifications d'instance des classes et des interfaces (objets), des slots avec des spécifications de valeur et des liens (instances d'association).



Présentation des diagrammes de classes et d'objets

- Le diagramme de classes est un diagramme de structure UML qui montre la structure du système conçu au niveau des classes et des interfaces, montre leurs caractéristiques, contraintes et relations
 - associations, généralisations, dépendances, etc.
- Certains types courants de diagrammes de classes sont :
 - diagramme de modèle de domaine ou diagramme de classes d'implémentation
 - Le diagramme d'objets peut être considéré comme un diagramme de classes au niveau de l'instance qui montre les spécifications d'instance des classes et des interfaces (objets), des slots avec des spécifications de valeur et des liens (instances d'association).

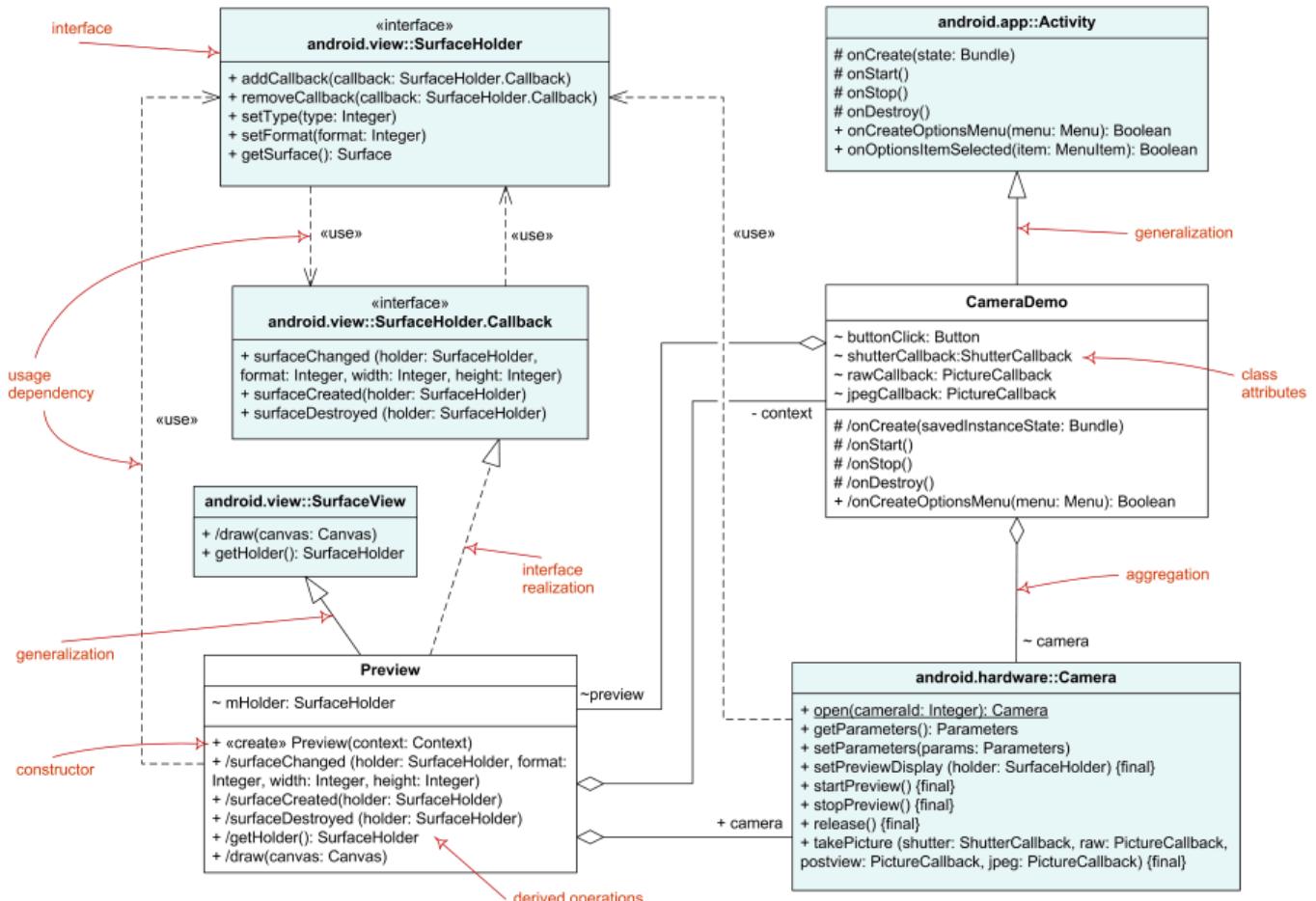


Diagramme d'objet

- Le diagramme d'objets était défini dans la spécification UML 1.4.2, désormais obsolète, comme "un graphique d'instances, y compris les objets et les valeurs de données. Un diagramme d'objets statique est une instance d'un diagramme de classes ; il montre un instantané de l'état détaillé d'un système à un point dans le temps." Il a également déclaré que le diagramme d'objets est "un diagramme de classes avec des objets et aucune classe".
- La spécification UML 2.4 ne fournit simplement aucune définition du diagramme d'objets, sauf que "les nœuds et les arêtes suivants sont généralement dessinés dans un diagramme d'objets : spécification d'instance et lien (c'est-à-dire association)".
- Notez que la hiérarchie standard des diagrammes UML 2.4 (voir Présentation des diagrammes UML 2.4) montre que les diagrammes de classes et les diagrammes d'objets n'ont aucun rapport. Certaines autres sources UML faisant autorité indiquent que les diagrammes de composants et les diagrammes de déploiement contenant uniquement des spécifications d'instance sont également des types particuliers de diagrammes d'objets. J'ai vraiment mal à la tête avec tout ce bordel et j'ai besoin de boire quelque chose... OMG s'il te plaît répare ça !
- La vue d'ensemble du diagramme d'objets ci-dessous montre certains éléments majeurs du diagramme d'objets : spécifications d'instance nommées et anonymes pour les objets, emplacements avec des spécifications de valeur et liens (instances d'association).

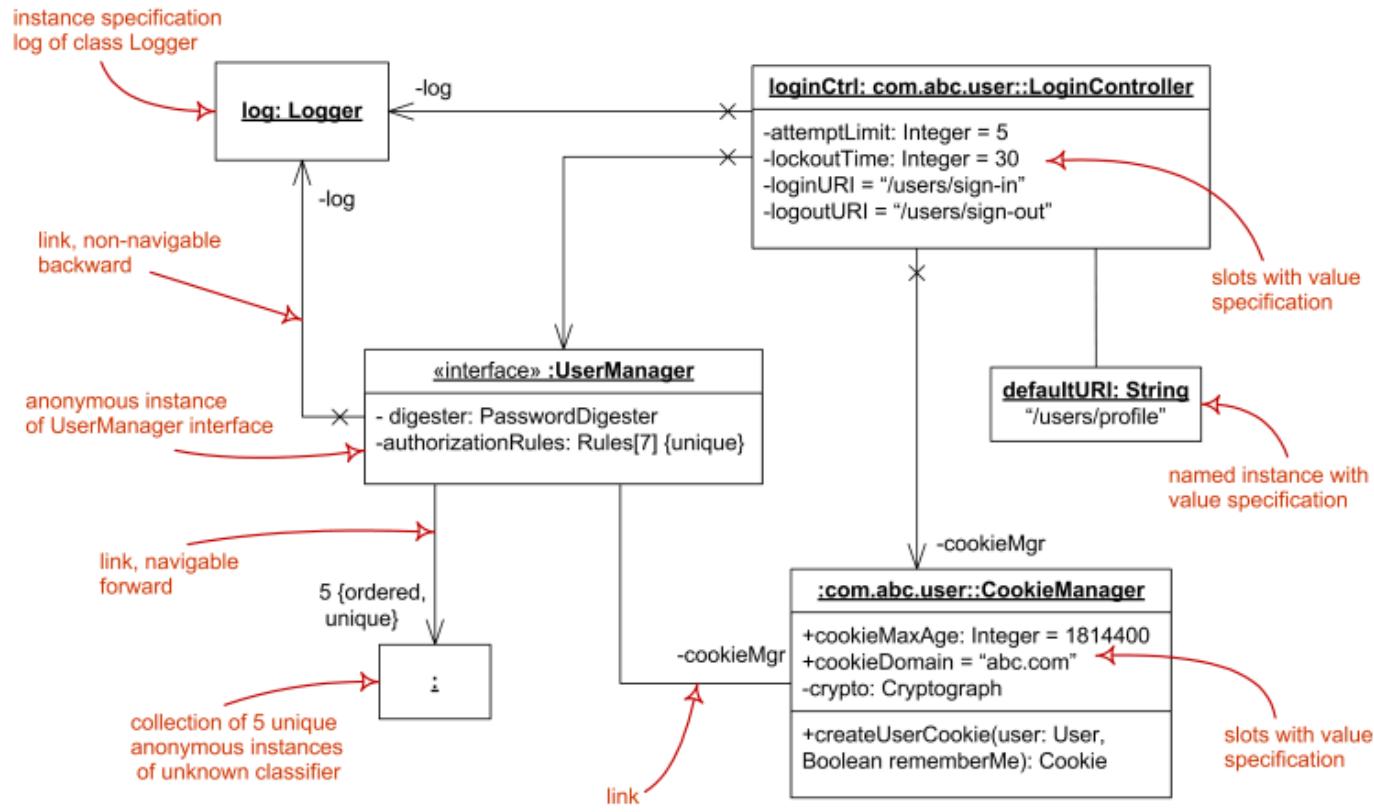
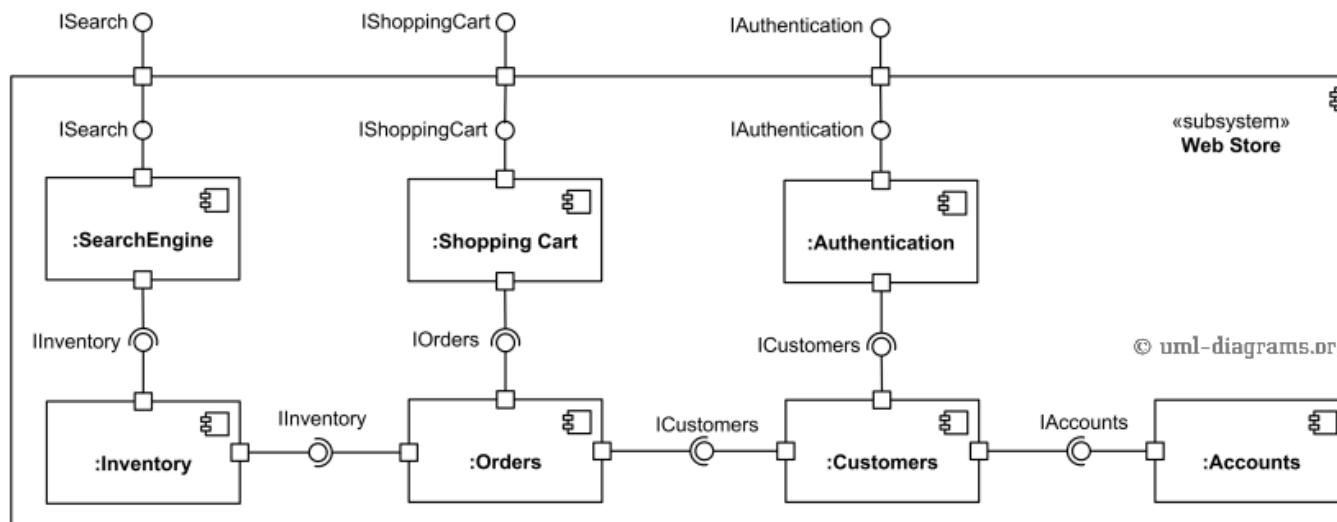
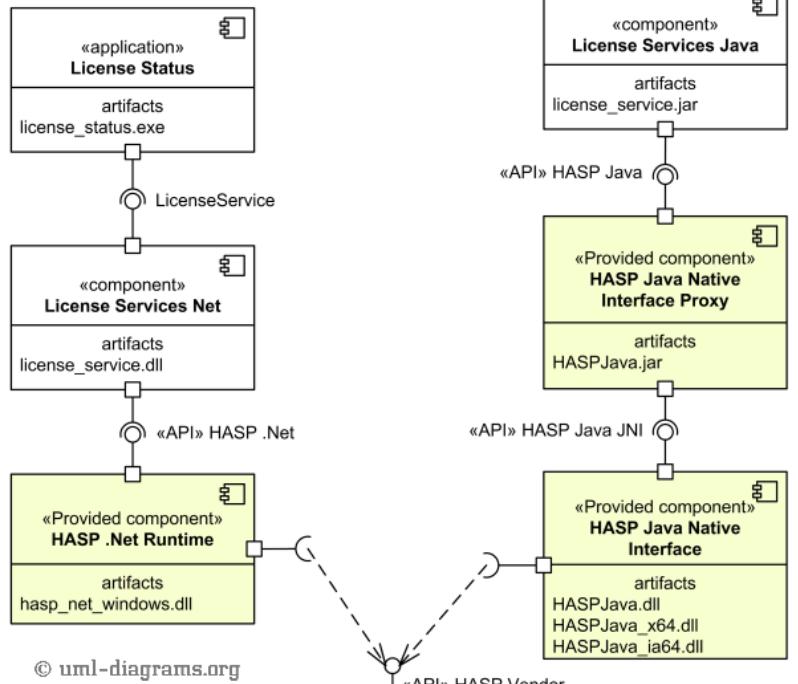
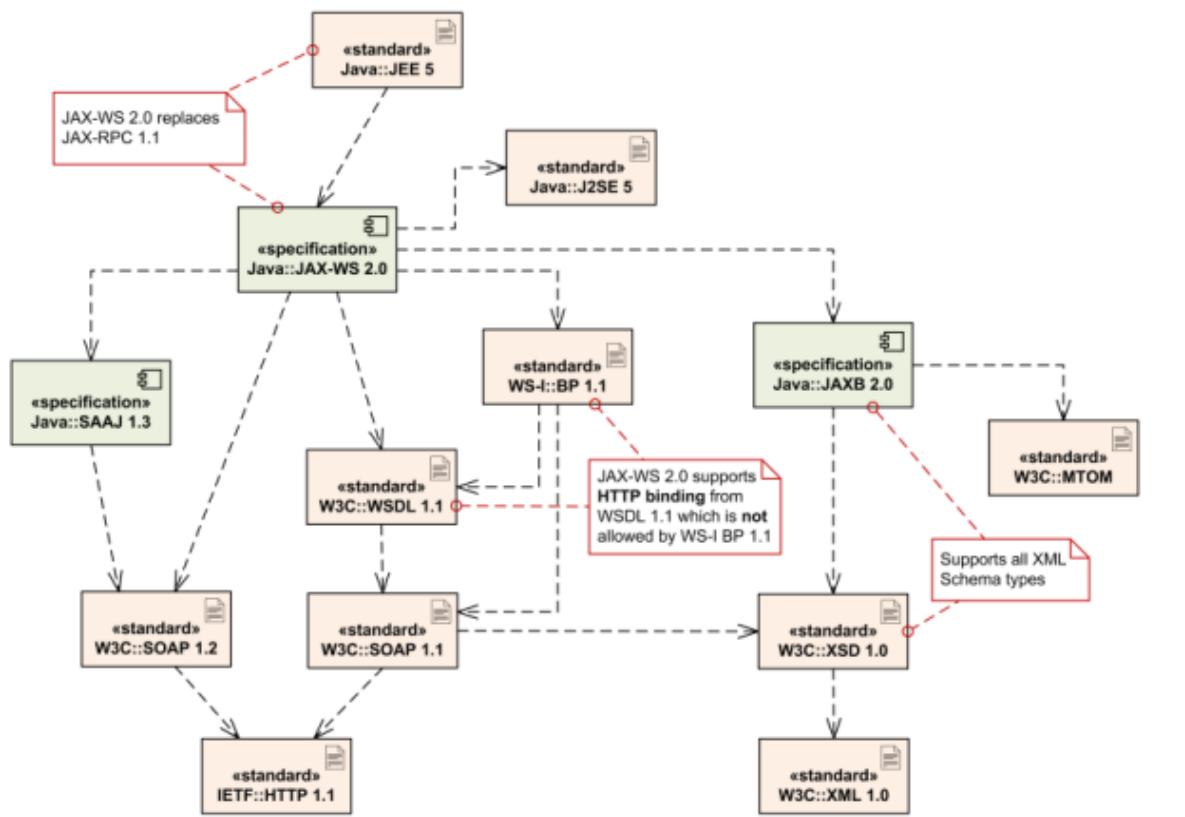


Diagramme des composants

- Le **diagramme de composants** montre les composants et les dépendances entre eux. Ce type de diagrammes est utilisé dans le développement basé sur des composants (CBD) pour décrire des systèmes avec une architecture orientée services (SOA).
- Le développement basé sur les composants est basé sur des hypothèses selon lesquelles les composants précédemment construits pourraient être réutilisés et que les composants pourraient être remplacés par d'autres composants « équivalents » ou « conformes », si nécessaire.
- Les artefacts qui implémentent le composant sont destinés à pouvoir être déployés et redéployés indépendamment, par exemple pour mettre à jour un système existant.
- Les composants en UML peuvent être:
 - logiques (exemples: composants métier, composants de processus), et
 - physiques (exemples: composants CORBA, composants EJB, composants COM+ et .NET, composants WSDL, etc.),
 - des artefacts qui les implémentent et des nœuds sur lesquels ils sont déployés et exécutés. Il est prévu que des profils basés sur des composants seront développés pour des technologies de composants spécifiques et les environnements matériels et logiciels associés.
- Les nœuds et leurs frontières sont généralement dessinés dans un diagramme de composants : composant, interface, interface fournie, interface requise, classe, port, connecteur, artefact, réalisation du composant, utilisation.



Les éléments de bases de UML2

Les principaux éléments de base à connaître sont

- les classes d'objet (EN Object Class),
- les objets ou spécification d'instance (EN Object, InstanceSpecification),
- les cas d'utilisation (EN Use Case),
- les acteurs (EN Actor),
- les activités (EN Activity),
- les relations (généralisation, association, agrégation, composition),
- les paquetages (EN package),
- les composants (EN component),
- les composites (EN composite),
- les classifiants (EN classifier) et
- les stéréotypes (EN stereotype).

Chacun est destiné à être utilisé dans tel ou tel type de diagramme UML.

Exploration de l'outillage UML

Exploration des divers types d'outils

Les outils nombreux et variés supportant UML

UML est une norme.

C'est aussi un standard de facto, car supporté par des outils très nombreux sur le marché.

Ces outils sont de divers types

- **Outils de dessins à partir de bibliothèques de composants visuels**

- Dessins statiques avec positionnement à la main (ex: Powerpoint, Keynote, Open Office, Visio, Dia, etc.)
- Dessins statiques avec positionnement automatisé ou « automated layouts » en anglais (ex: Yed)
- Dessins sous forme de graphes que l'on peut décorer avec des icônes, tels des cartes heuristiques (« mindmaps ») ou des cartes de sujets (« topic maps »)
- Outils de production de graphes interactifs combinant positionnement interactif et extension avec des traitements, à plat (ex: vis.js) ou composites (ex: cytoscape.js)

- **Outils de modélisation visuelle: il y a simultanément production de diagrammes et de modèles**

- Modèles textuel permettant de produire les diagrammes automatiquement (ex: PlantUML) - permet de gérer les définitions textuelles de modèle comme du code, avec les outils de gestion de configuration de code. Les capacités associées à ces outils sont néanmoins très limitées par rapport aux outils de modélisation visuelle et aux ateliers d'ingénierie par les modèles
- Création de diagrammes entraînant la production d'un modèle, éventuellement direction directe dans le modèle

Les diagrammes sont des vues sur le modèle, dédiés à répondre à certaines questions et pour diverses catégories d'utilisateur: c'est le principe des points de vue d'architecture, dont l'usage permet de définir des pratiques communes de production de modèles pour des équipes ou des communautés.

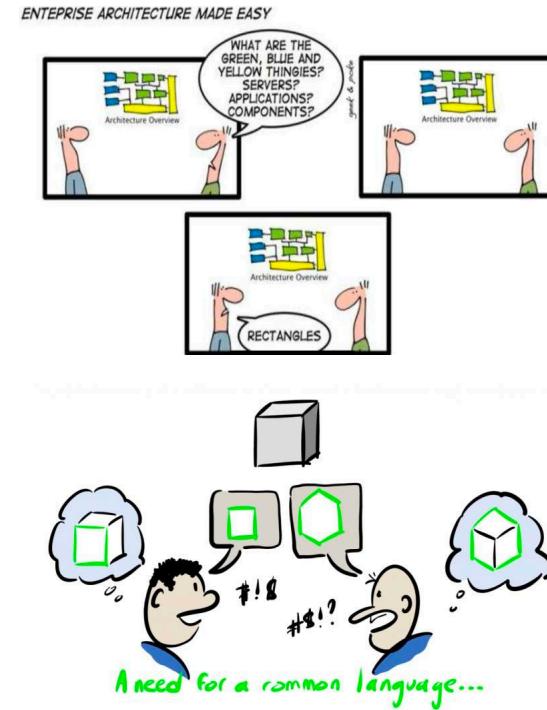
Attention: un diagramme est une vue partielle du modèle. L'ensemble du modèle se construit mentalement à partir des vues.

Quand les vues sont trop nombreuses et le système décrit trop complexe, l'outil de modélisation visuelle permet d'être aidé par l'ordinateur en supportant

- La maintenance de cohérence des vues à partir du modèle
- Les requêtes (l'outil constitue une base de données interrogable, localement ou dans des référentiels centralisés)
- La validation automatique sur base de règles et de contraintes
- La génération automatique de documents
- La génération de modèle à partir de code (rétro ingénierie)
- La simulation dans un processus d'ingénierie de système logiciel
- L'automatisation de la production de plans de test
- L'analyse de fonctionnement de système en opération à partir de son jumeau numérique (le modèle du système agrège les observations des outils qui le surveille et permet de les restituer sous forme de modèles visuels annotés)

Les outils de modélisation visuelle les plus sophistiqués sont des ateliers de modélisation qui implémentent les spécifications de l'OMG pour l'ingénierie par les modèles, structurés avec le Model Driven Architecture (MDA) de l'OMG (ex: Magic Draw, Enterprise Architect, Papyrus, Modélia, Visual Paradigm, etc.)

Il existe également des ateliers de modélisation, visant à créer des plateformes de modélisation pour des langages de modélisation spécifiques à un domaine autre que UML (ex: Obeo Designer)



MODELISER N'EST PAS DESSINER!

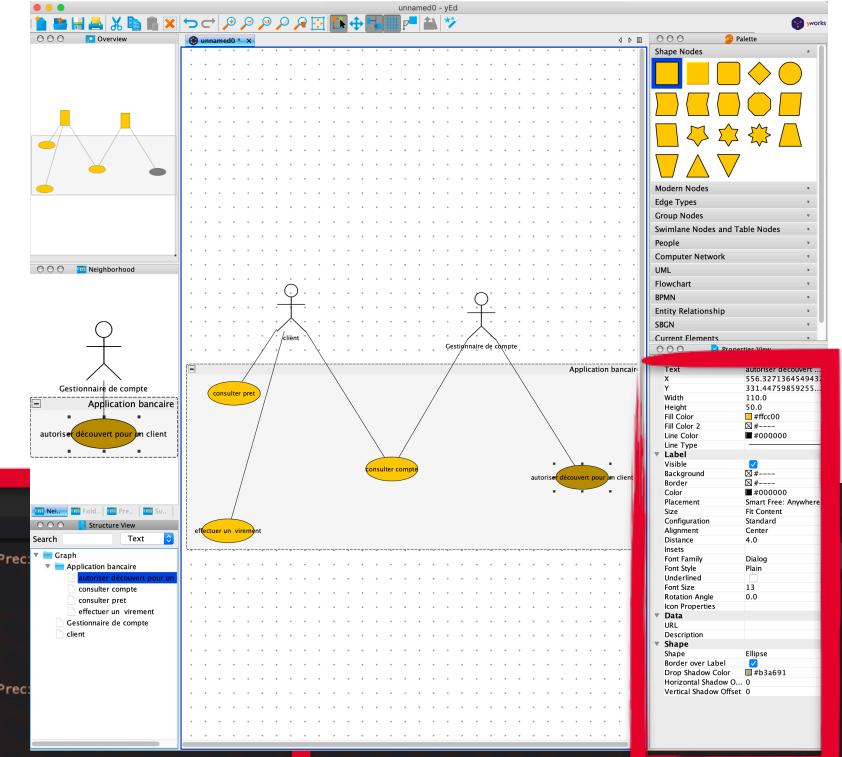
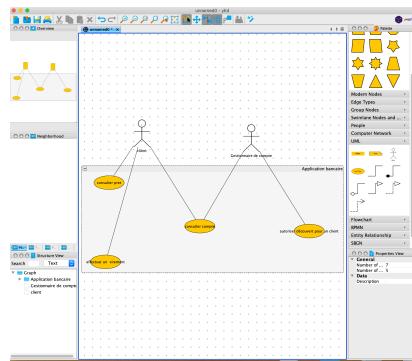
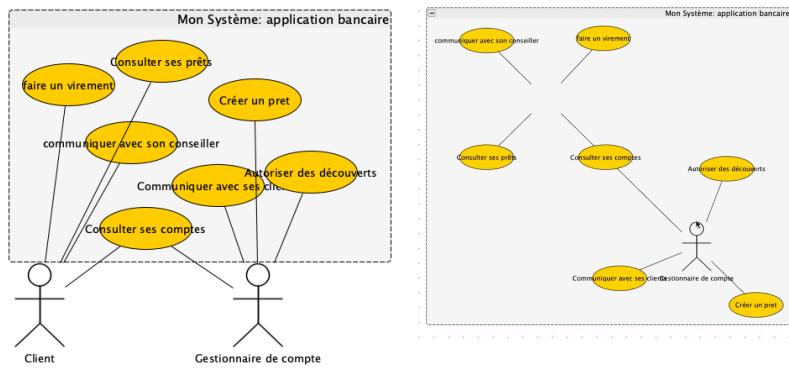
L'ensemble de ces outils a en commun le fait de permettre de produire des diagrammes à partir de la spécification du langage visuel de UML. Mais UML ne se limite pas à un langage visuel, il vient avec un métamodèle (description formelle du langage manipulable par l'ordinateur), un format d'échange basé sur XML (XML Interchange Model ou XML), des capacités d'extension de langage à partir de profils. Il s'inscrit dans le cadre plus large du Model Driven Architecture de l'OMG qui permet d'utiliser le noyau d'UML pour créer des plateformes de modélisation basée sur d'autres langages, ainsi que les transformations de modèle: modèle à modèle, texte vers modèle, modèle vers texte. Des spécifications de l'Object Management Group existe pour supporter l'ensemble de ces fonctionnalités que peuvent implémenter les ateliers de modélisations qui répondent au besoin de l'ingénierie par les modèles. Certains langages métiers sont spécifiés en tant qu'extension d'UML par des profils. En particulier:

- Business Process Modeling Language (BPMN)
- System Modeling Language (SysML)

D'autres sont spécifiés sous forme de métamodèle formalisé UML comme langage de modélisation du métamodèle (défini dans les spécifications liées au Meta Object Facilities ou MOF). UML est complété par une spécification relative à la modélisation de contraintes: Object Constraint Language ou OCL.

UML avec Yed: dessiner UML avec arrangements automatiques (automated layouts)

MODELISER N'EST PAS DESSINER!



Un cas d'utilisation dessiné avec Yed avec plusieurs arrangements automatiques
(gain de temps non négligeable pour produire des diagrammes)

```

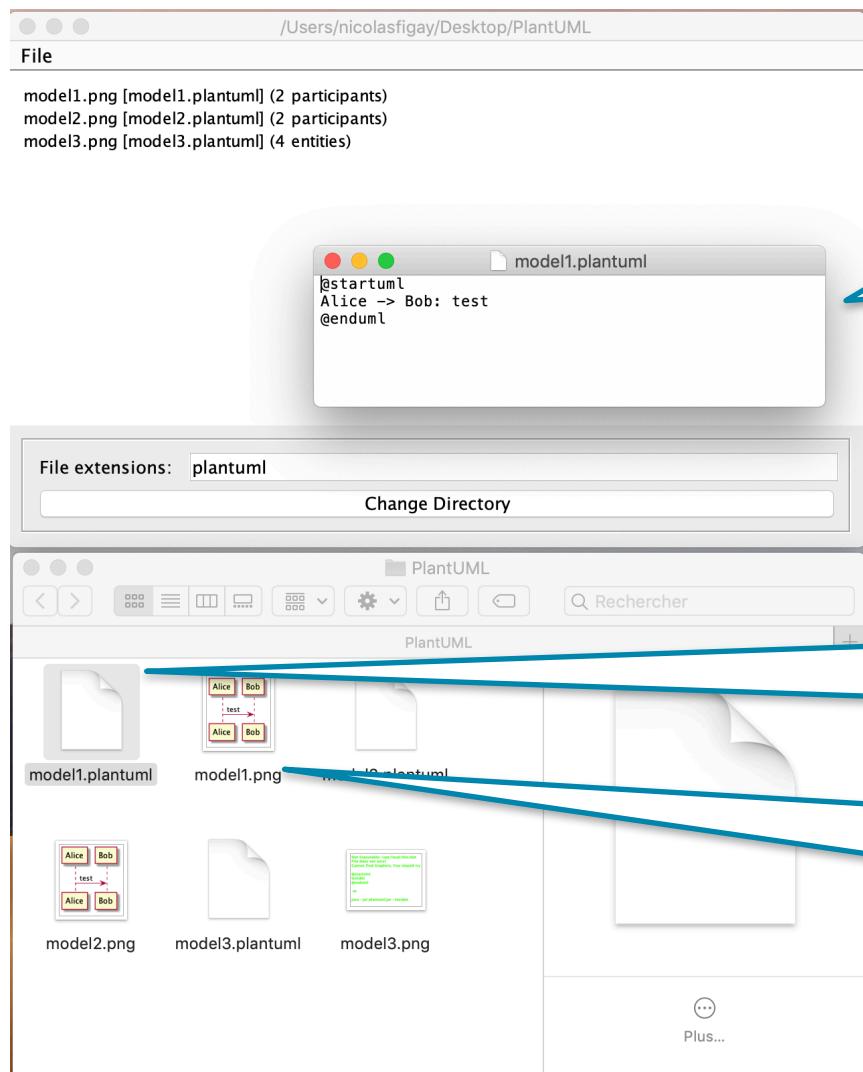
use_case.svg x
Users > nicolasfigay > Desktop > use_case.svg
143   <y>
144     <g font-size="15px" stroke-linecap="butt" transform="matrix(1,0,0,1,16,15)" text-rendering="geometricPrecision" font-family="sans-serif" shape-rendering="geometricPrecision" fill="none" stroke="rgb(255,204,0)" stroke-width="1px">
145       <text x="200.251" xml:space="preserve" y="16.502" clip-path="url(#clipPath2)" stroke="none">Mon Système: application bancaire</text>
146       <rect x="-0.9995" y="0" clip-path="url(#clipPath2)" fill="none" width="455.6724" stroke-dasharray="6,2" rx="4" ry="4" height="300.666"/>
147     </g>
148     <g fill="rgb(255,204,0)" text-rendering="geometricPrecision" shape-rendering="geometricPrecision" transform="matrix(1,0,0,1,16,15)" stroke="rgb(255,204,0)" stroke-width="1px">
149       <ellipse rx="55" ry="25" clip-path="url(#clipPath2)" cx="162" cy="260.666" stroke="none"/>
150     </g>
151     <g font-size="13px" stroke-linecap="butt" transform="matrix(1,0,0,1,16,15)" text-rendering="geometricPrecision" font-family="sans-serif" shape-rendering="geometricPrecision" fill="none" stroke="rgb(255,204,0)" stroke-width="1px">
152       <text x="90.3286" xml:space="preserve" y="265.5791" clip-path="url(#clipPath2)" stroke="none">Consulter ses comptes</text>
153       <ellipse rx="55" fill="none" ry="25" clip-path="url(#clipPath2)" cx="162" cy="260.666"/>
154     </g>
155     <g fill="rgb(255,204,0)" text-rendering="geometricPrecision" shape-rendering="geometricPrecision" transform="matrix(1,0,0,1,16,15)" stroke="rgb(255,204,0)" stroke-width="1px">
156       <ellipse rx="55" ry="25" clip-path="url(#clipPath2)" cx="145" cy="158.666" stroke="none"/>
157     </g>
158     <g font-size="13px" stroke-linecap="butt" transform="matrix(1,0,0,1,16,15)" text-rendering="geometricPrecision" font-family="sans-serif" shape-rendering="geometricPrecision" stroke-miterlimit="1.45" fill="none" stroke="rgb(255,204,0)" stroke-width="1px">
159       <text x="39.5654" xml:space="preserve" y="163.5791" clip-path="url(#clipPath2)" stroke="none">communiquer avec son conseiller</text>
160       <ellipse rx="55" fill="none" ry="25" clip-path="url(#clipPath2)" cx="145" cy="158.666"/>
161     </g>
162     <g fill="rgb(255,204,0)" text-rendering="geometricPrecision" shape-rendering="geometricPrecision" transform="matrix(1,0,0,1,16,15)" stroke="rgb(255,204,0)" stroke-width="1px">
163       <ellipse rx="55" ry="25" clip-path="url(#clipPath2)" cx="181" cy="61.666" stroke="none"/>
164     </g>
165     <g font-size="13px" stroke-linecap="butt" transform="matrix(1,0,0,1,16,15)" text-rendering="geometricPrecision" font-family="sans-serif" shape-rendering="geometricPrecision" stroke-miterlimit="1.45" fill="none" stroke="rgb(255,204,0)" stroke-width="1px">
166       <text x="120.0593" xml:space="preserve" y="66.5791" clip-path="url(#clipPath2)" stroke="none">Consulter ses prêts</text>
167       <ellipse rx="55" fill="none" ry="25" clip-path="url(#clipPath2)" cx="181" cy="61.666"/>
168     </g>
169     <g fill="rgb(255,204,0)" text-rendering="geometricPrecision" shape-rendering="geometricPrecision" transform="matrix(1,0,0,1,16,15)" stroke="rgb(255,204,0)" stroke-width="1px">
170       <ellipse rx="55" ry="25" clip-path="url(#clipPath2)" cx="70" cy="89.666" stroke="none"/>
171     </g>
172     <g font-size="13px" stroke-linecap="butt" transform="matrix(1,0,0,1,16,15)" text-rendering="geometricPrecision" font-family="sans-serif" shape-rendering="geometricPrecision" stroke-miterlimit="1.45" fill="none" stroke="rgb(255,204,0)" stroke-width="1px">
173       <text x="16.0005" xml:space="preserve" y="94.5791" clip-path="url(#clipPath2)" stroke="none">faire un virement</text>
174       <ellipse rx="55" fill="none" ry="25" clip-path="url(#clipPath2)" cx="70" cy="89.666"/>
175     </g>
176     <g fill="rgb(255,204,0)" text-rendering="geometricPrecision" shape-rendering="geometricPrecision" transform="matrix(1,0,0,1,16,15)" stroke="rgb(255,204,0)" stroke-width="1px">
177       <ellipse rx="55" ry="25" clip-path="url(#clipPath2)" cx="248" cy="209.666" stroke="none"/>
178     </g>
179     <g font-size="13px" stroke-linecap="butt" transform="matrix(1,0,0,1,16,15)" text-rendering="geometricPrecision" font-family="sans-serif" shape-rendering="geometricPrecision" stroke-miterlimit="1.45" fill="none" stroke="rgb(255,204,0)" stroke-width="1px">
180       <text x="152.2107" xml:space="preserve" y="214.5791" clip-path="url(#clipPath2)" stroke="none">Communiquer avec ses clients</text>
181       <ellipse rx="55" fill="none" ry="25" clip-path="url(#clipPath2)" cx="248" cy="209.666"/>
182     </g>
183   </y>

```

Essayez de retrouver le cas d'utilisation
« Communiquer avec ses clients » dans ce fichier texte

Dessin assisté par ordinateur
des primitives relatives à des
objets visuels plus qu'à ce
qui est représenté: acteurs,
cas d'utilisations, etc.

UML avec PlantUML: décrire le modèle avec du texte puis générer automatiquement un diagramme statique



1- Un langage textuel spécifique pour définir le diagramme UML à dessiner

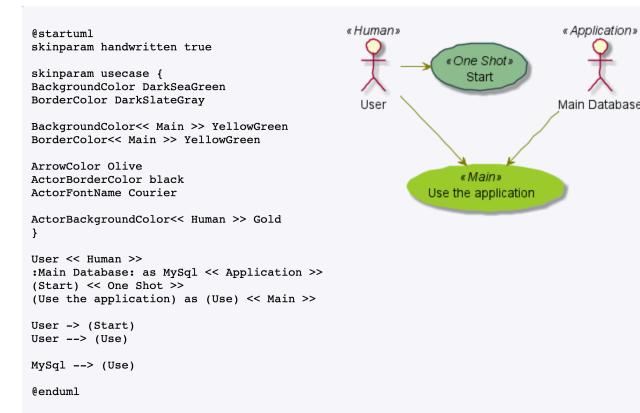
2- Le fichier texte est mis dans un répertoire sur lequel tourne PlantUML

3- Un diagramme est produit automatiquement sous forme d'image raster ou vectorielle (SVG)

Le langage textuel utilisé par PlantUML est spécifique à l'outil!

A noter qu'UML n'est pas défini avec un langage textuel standardisé.

Le langage textuel de PlantUML permettant de définir des diagrammes et des éléments du modèle, mais permet également de préciser des informations de présentations visuelles. Modèle et représentations visuelles sont mélangées, mais c'est la description du modèle qui prévaut, le modèle se dessinant même si on ajoute pas d'information relatives à la présentation. Il n'y a pas de mots clefs pour les construction du langage UML (dans l'exemple, pas de mot clé Actor ou UseCase »): la syntaxe est compacte mais requiert un apprentissage.



Les fichiers PlantUML utilisés pour la génération des diagrammes étant textuels, ils peuvent être gérés en configuration par des gestionnaires de configuration de code.

Si le langage visuel UML est bien respecté et couvert, mieux pour certains diagrammes que des ateliers de modélisation UML, ces derniers sont plus sophistiqués et seuls permettent une approche industrielle d'ingénierie par les modèles.

Pas encore de la modélisation visuelle DE LA MODÉLISATION VISUELLE!

Les diagrammes de séquence illustrés avec PlantUML

<https://plantuml.com/fr/>

PlantUML en un mot ↗

PlantUML est un composant qui permet de dessiner rapidement des:

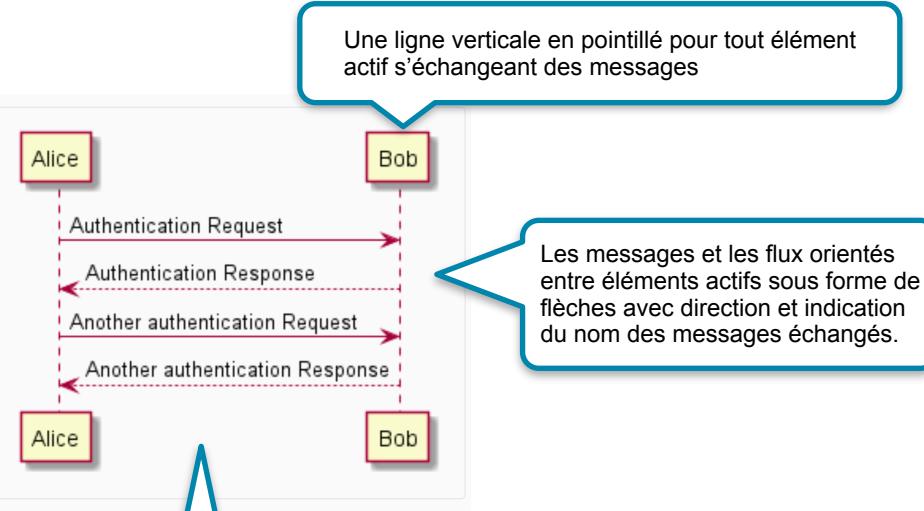
- diagrammes de séquence
- diagrammes de cas d'utilisation
- diagrammes de classes
- diagrammes d'objet
- diagrammes d'activité (ici l'ancienne syntaxe)
- diagrammes de composant
- diagrammes de déploiement
- diagrammes d'état
- diagrammes de temps

Certains autres diagrammes (hors UML) sont aussi possibles:

- données au format JSON
- données au format YAML
- diagrammes de réseaux (nwdiag)
- maquettes d'interface graphique (salt)
- diagrammes Archimate
- diagrammes de langage de description et de spécification (LDS) ou *Specification and Description Language (SDL)*
- diagrammes ditaa
- diagrammes de Gant
- diagrammes d'idées (mindmap)
- organigramme ou *Work Breakdown Structure (WBS)*
- notation mathématique avec AsciiMath ou JLaTeXMath
- diagrammes entité relation (ER/IE)

Exemple basique

```
@startuml  
Alice --> Bob: Authentication Request  
Bob --> Alice: Authentication Response  
  
Alice --> Bob: Another authentication Request  
Alice <-- Bob: Another authentication Response  
@enduml
```



Une ligne verticale en pointillé pour tout élément actif s'échangeant des messages

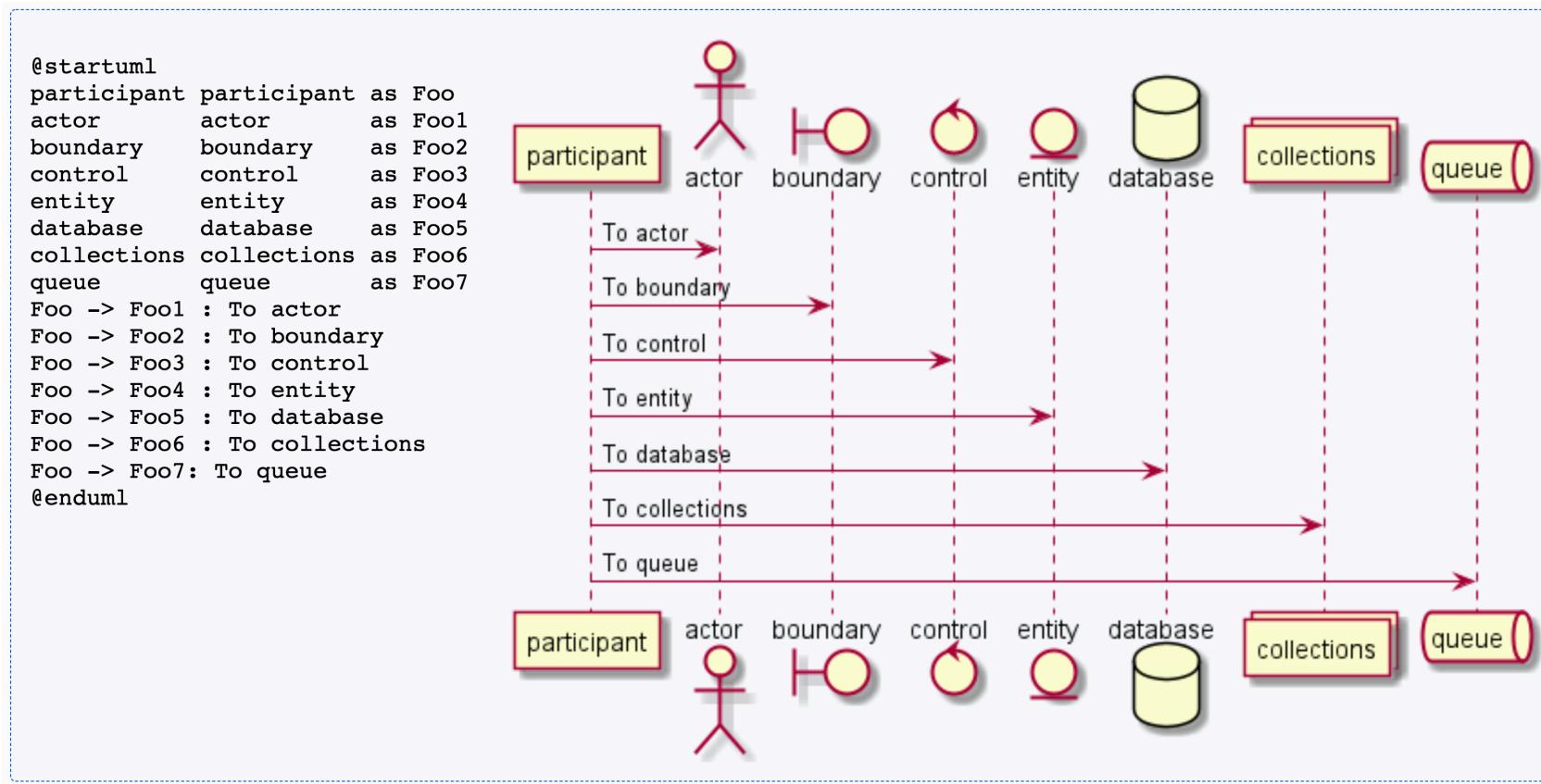
Les messages et les flux orientés entre éléments actifs sous forme de flèches avec direction et indication du nom des messages échangés.

Ces séquences étant relative à un comportement fonction du temps, le diagramme de séquence est par conséquent un diagramme dynamique.

L'ordre des messages de haut en bas indique l'ordre dans lequel les échanges de messages sont séquencés, de la même manière que l'ordre des lignes d'un programme .

Les diagrammes de séquence illustrés avec PlantUML

Exemple plus sophistiqué: ordre des participants (de gauche à droite) et type des participants (actor, boundary, control ...)

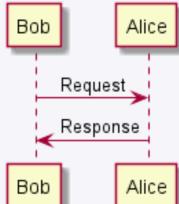


Les diagrammes de séquence illustrés avec PlantUML

Exemples plus sophistiqués

Direction des messages

```
@startuml
skinparam sequenceMessageAlign right
Bob -> Alice : Request
Alice --> Bob : Response
@enduml
```



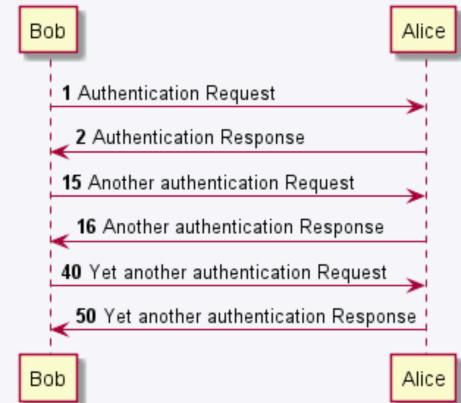
Numérotation automatique des messages

```
@startuml
autonumber
Bob -> Alice : Authentication Request
Bob <- Alice : Authentication Response

autonumber 15
Bob -> Alice : Another authentication Request
Bob <- Alice : Another authentication Response

autonumber 40 10
Bob -> Alice : Yet another authentication Request
Bob <- Alice : Yet another authentication Response

@enduml
```

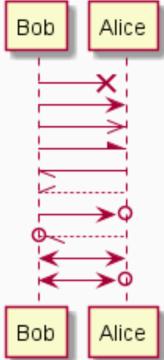


Styles de flèches (pas toujours une sémantique UML standard pour les symboles utilisés)

```
@startuml
Bob ->x Alice
Bob -> Alice
Bob --> Alice
Bob -\ Alice
Bob \\<- Alice
Bob //--- Alice

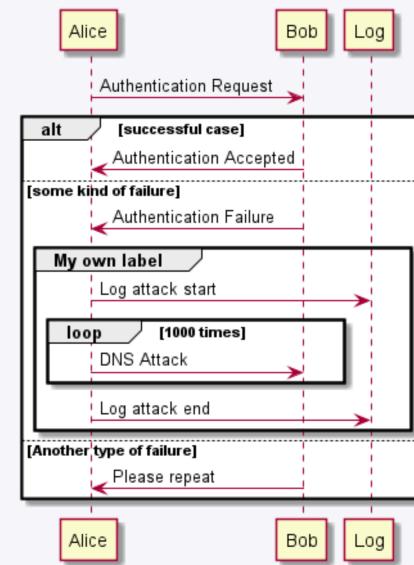
Bob ->o Alice
Bob o\\-- Alice

Bob <-> Alice
Bob <->o Alice
@enduml
```



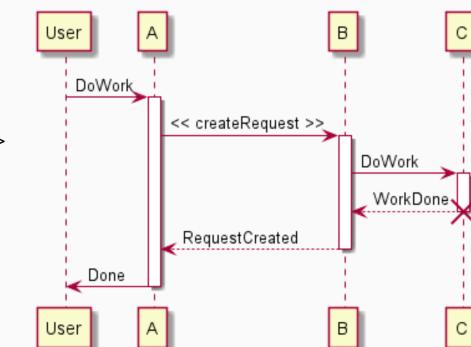
Groupes imbriqués, branchement alternatifs, boucles
(comme pour la programmation procédurale)

```
@startuml
Alice -> Bob: Authentication Request
alt successful case
    Bob -> Alice: Authentication Accepted
else some kind of failure
    Bob -> Alice: Authentication Failure
    group My own label
        Alice -> Log : Log attack start
        loop 1000 times
            Alice -> Bob: DNS Attack
        end
        Alice -> Log : Log attack end
    end
else Another type of failure
    Bob -> Alice: Please repeat
end
@enduml
```



Lignes de vie
(participant actif ou inactif)

```
@startuml
participant User
User -> A: DoWork
activate A
A --> B: << createRequest >>
activate B
B --> C: DoWork
activate C
C --> B: WorkDone
destroy C
B --> A: RequestCreated
deactivate B
A -> User: Done
deactivate A
@enduml
```



Les ateliers de méta-modélisation UML

- ❑ Il s'agit des outils s'appuyant sur les spécifications de l'OMG pour l'ingénierie par les modèles, selon le cadre d'architecture fourni par le Model Driven Architecture (MDA)
- ❑ Un atelier de méta-modélisation permet:
 1. De produire des modèles et des ensembles de diagrammes cohérents
 2. D'exprimer des contraintes et de les valider en s'appuyant sur le langage OCL
 3. De générer de la documentation (Modèle vers Texte)
 4. De générer du code (Modèle vers Texte)
 5. De générer des modèles à partir du code (Text to Model)
 6. De se connecter à des outils de simulation
 7. De créer des langages spécifiques à des domaines, par le biais de profils UML ou de modélisation de langages avec les facilités pour les Meta-Objets (Meta Object Facilities) et des générateurs d'atelier de modélisation visuelle à partir de nouveaux langages de modélisation
 8. De transformer de modèles en changeant éventuellement de langages de modélisation (Modèle vers Modèle)
- ❑ Il existe une catégorie d'outils pour 7 et 8, qui sont des outils de génération de plateformes de modélisation (exemple: OBEO Designer)
- ❑ Il existe aussi une catégories d'outils automatisant des chaines de transformation de modèle vers la génération, de tests et de déploiement de composants sur des plate-formes d'exécution (exemple: AndroMDA)

On peut utiliser ces ateliers sans avoir besoin d'utiliser toutes ces fonctionnalités et en tirer parti pour des usages simples (Ouf!)

Choisir et installer un atelier de modélisation

- Papyrus, modélisation UML sur Eclipse MDT
- Open Modélio

Note: l'utilisation se fait normalement facilement sur tout type de système d'exploitation. Néanmoins, et durant le cours, il s'est avéré qu'il peut exister quelques complications liées à la configuration de la machine où est installée le logiciel, notamment en ce qui concerne Java.

Installation Papyrus:

- 1- Télécharger Eclipse IDE (2021-09 au moment où le support de cours a été produit), pour l'OS de la machine que vous ciblez (<https://www.eclipse.org/downloads/>)
- 2- L'application téléchargée vous donne le choix entre plusieurs environnements: choisissez MDT
- 3- Lancer l'éditeur et choisissez votre espace de travail
- 4- Dans le menu Help>>MarketPlace, tapez Papyrus et choisissez la dernière version de Papyrus SysML (1.6.2.2a ici)
- 5- Quand le plugin est installé, il faut choisir Papyrus dans le menu Windows>>Perspective>>Select Perspective>>Others
- 6- Il faut alors créer un Projet Papyrus à partir du menu New
- 7- Lors de la création du projet, on indique le type de diagramme à créer.
- 8- A partir du menu Papyrus, on a alors accès à la création des diagrammes qui apparaissent sous forme d'onglet dans le panneau des diagrammes
- 9- Le panneau « Model Explorer » donne le contenu du modèle, le panneau « Project Explorer » les fichiers constituant le projet
- 10- Le panneau des propriétés donne les propriétés des objets sélectionnés

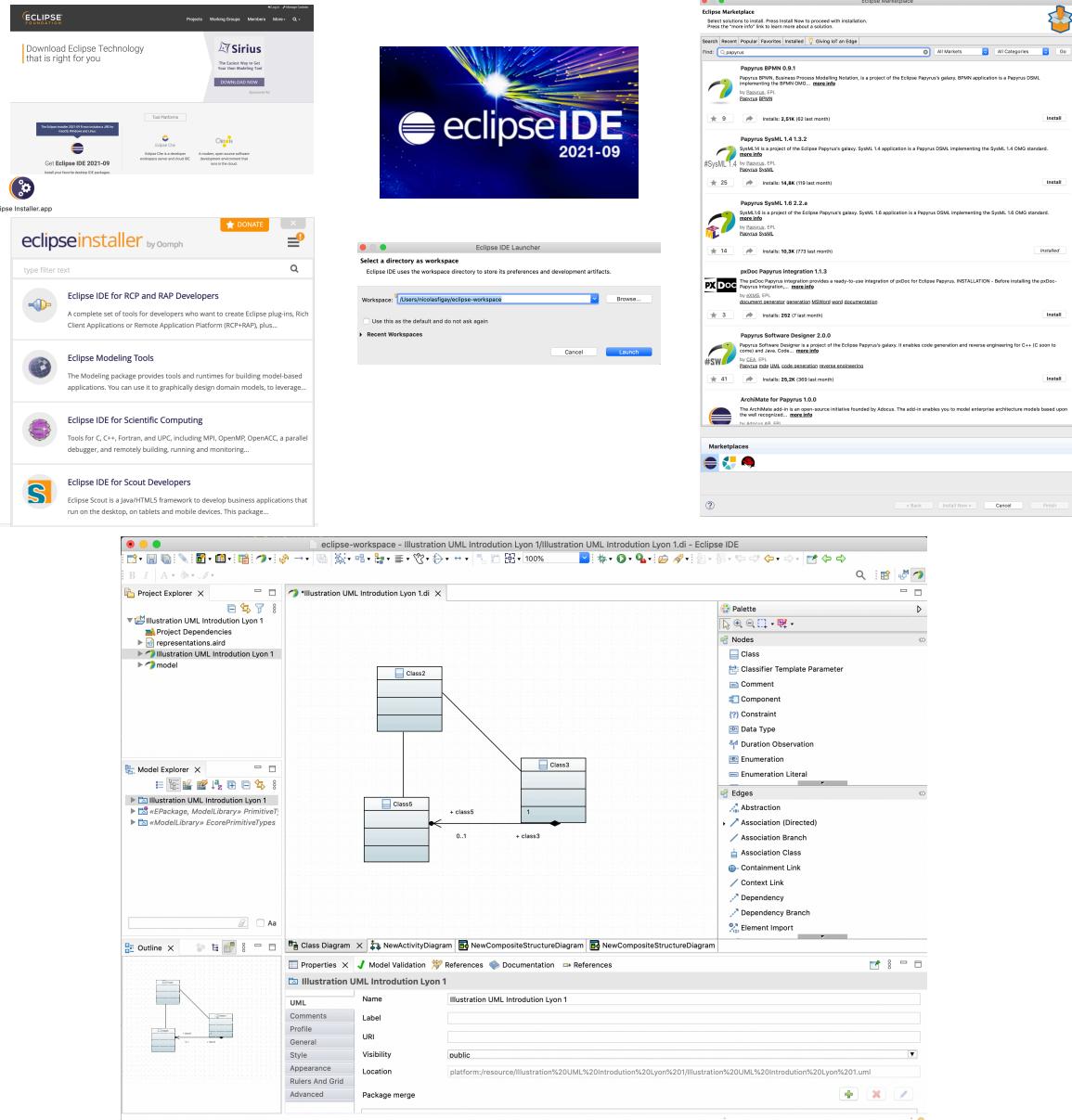
De manière générale, l'éditeur de diagramme est assez peu ergonomique, les constructeurs présentés dans la palette peu clair en dépit qu'il s'agisse de constructeurs UML. Le support de la norme est assez inégale selon le type de diagrammes à construire. Parce qu'il intègre des éditeurs OCL avec des outils de validation, et toute la palette du développeur pour travailler sur les fichiers, Papyrus reste un outil pour spécialistes et développeurs, dont l'ergonomie est plus que discutable. Il est souvent utilisé pour des projets de recherche. A éviter pour les débutants, mais également pour les projets où la qualité des diagrammes et l'ergonomie est souhaitée.

Comme alternative gratuite pour découvrir UML, on se tournera plutôt vers Open Modelio.

Les meilleurs outils en termes de conformité à la norme UML et d'ergonomie restent Magic Draw, Enterprise Architect et Hopex (ex Méga).

J'ai pour ma part une préférence pour les deux premiers en termes d'ergonomie. Magic Draw est le plus proche en termes de conformité à la spécification, mais est beaucoup moins ouvert et performant en termes d'import/export standardisé que EA. Enfin EA est moins cher et intègre par défaut de nombreux langages par défaut qui complémentent UML: BPMN, ArchiMate, etc. Modélio est également une très bonne alternative, d'une très grande qualité. Qui plus est, il est français. Choix à faire en fonction de l'étude des besoins et des caractéristiques de chaque outil, sans oublier prix et la simplicité.

Mon choix pour la prise en main d'UML si on ne peut investir dans l'achat d'un logiciel: Open Modélio



Modélio

- ❑ Installation de Modélio
 - ❑ Modélio devrait être à priori simple à installer.
 - ❑ Il existe néanmoins un soucis pour les paramètres de Java lors d'une installation sur un Mac avec OSx
 - ❑ Le problème n'ayant pas été résolu, l'outil n'a pas pu être utilisé durant le cours.
- ❑ Leçon à retenir
 - ❑ L'installation d'un atelier n'est pas toujours simple, et il faut le prendre en compte pour la sélection des outils dans un contexte industriel (usage par une équipe de concepteurs et d'architectes qui collaborent)

Modélio pour illustrer les diagrammes d'activité

- Installation de Modélio
- Rapide prise en main

Magic Draw

- Magic Draw est un des outils leader du Marché des ateliers de modélisation UML
 - Initialement produit par la société No Magic
 - Racheté par Dassault Systèmes pour l'intégrer dans les solutions d'ingénierie assistées par ordinateur (mécanique, électrique, etc.)
 - Lien vers le produit et ses caractéristiques: <https://www.3ds.com/products-services/catia/products/no-magic/magicdraw/>
- En lisant la fiche du produit, vous retrouverez les caractéristiques d'une atelier de métamodélisation tel qu'évoquées dans ce qui précède:

Industry standards-compliance and support

MagicDraw supports the UML 2 metamodel, the latest XMI standard for data storage and the most popular programming languages for implementation.

Unlike other UML modeling and architecture environments, MagicDraw makes it easy for you to deploy a Software Development Life Cycle (SDLC) environment that best suits the needs of your business. Our approach to standards and our Open API makes it easy for you to integrate with applications that work together, best supporting the needs of your business. We integrate with many leading products: IDEs, requirements, testing, estimation, MDD, database, and others.

MagicDraw is central to MDD solutions

MagicDraw is by far the tool of choice in the world of Model Driven Architecture. Major MDD vendors select and recommend MagicDraw. MagicDraw integrates with IO Software ArcStyler, AndroMDA, openArchitectureWare, Codagen Architect, and others.

Independence from specific development methodology

MagicDraw provides independence from any specific software development process, conforming nicely to your company process; allowing centralization of business and process modeling, requirements capture and design.

MagicDraw is not tied to any one phase of your project. Start MagicDraw from any point in your architecture and modeling process; it doesn't matter, for example, if your project is presently in a requirements or maintenance phase.

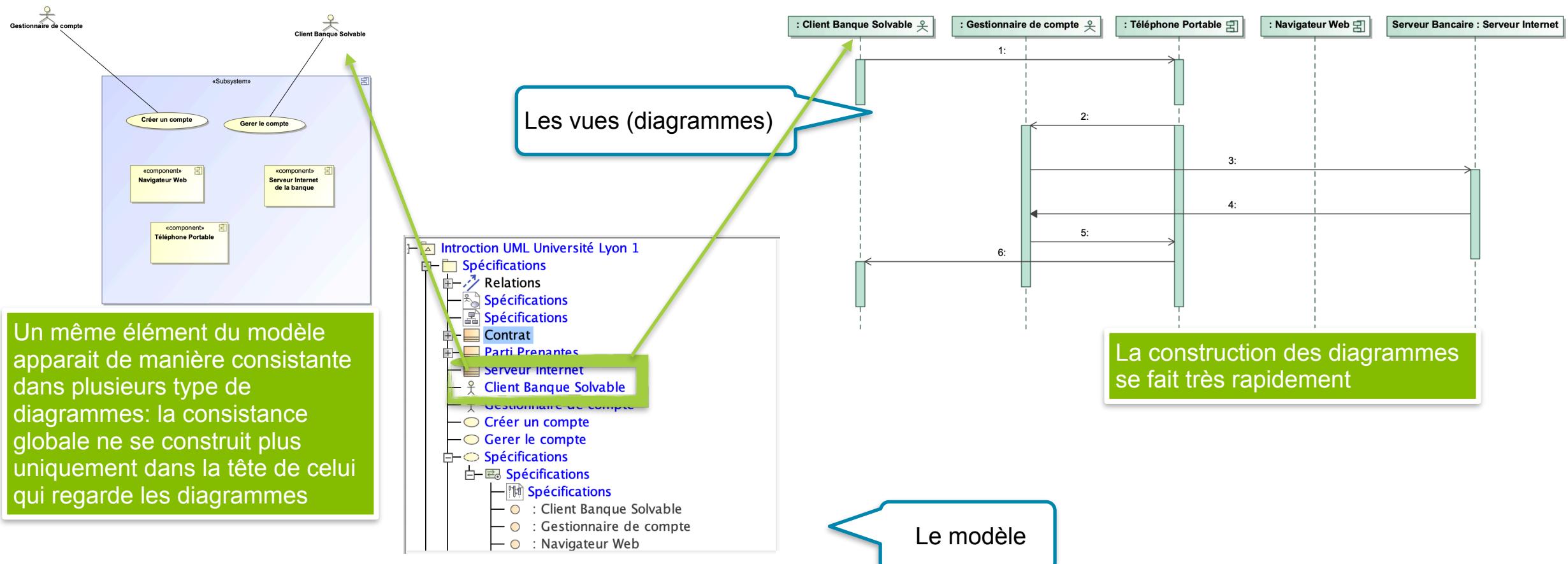
Usability, intuitiveness and quick start

MagicDraw provides intuitive controls within a very well designed GUI which allows users to model without having to spend time learning about the controls. Experience how quickly you get used to MagicDraw. With MagicDraw you will save time and improve productivity. Download the [UML Quick reference guide](#).

Magic Draw - Une plateforme de modélisation

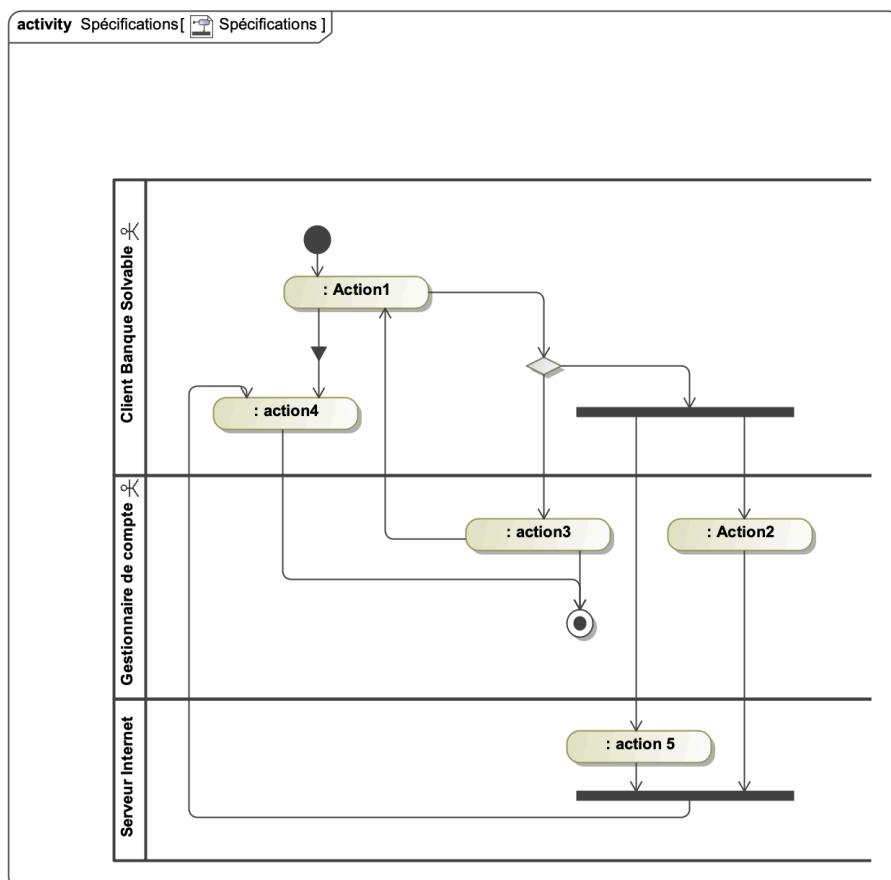
- On peut refaire avec Magic Draw les diagrammes réalisés précédemment avec les autres outils
 - plus rapidement
 - avec une cohérence entre le diagramme assurer par le modèle et l'outil
 - de manière plus ergonomique que Papyrus
 - avec des progrès potentiels pour l'arrangement dynamique des diagrammes

Illustration avec construction jointe d'un diagramme de cas d'utilisation, de composants et d'interaction en quelque minutes:



Magic Draw - Illustration de la construction d'un diagramme d'activité

- C'est un diagramme comportemental, qui décrit une activité en tant qu'un ensemble d'action
 - avec des liens de séquence
 - avec des flux d'entrée-sortie avec des types d'objets transportés (lien vers des classes ou autres éléments du modèle)
 - Eléments de début et de fin de l'activité
 - Ajout de « swimlanes », auquel on attache acteur ou composant, à qui sont assignés les actions



- Utilisable pour la modélisation de tout type d'activité: algorithme, programme, process métier ...
- Intégration possible avec les autres types de diagramme et les représentations composites et d'échanges de signaux
- Possibilité de capturer des modèles de processus de type IDEF0 (activités structurée imbriquées avec flux d'information)

Parfois considéré comme trop générique et pas assez adapté à des domaines spécifiques (ex: modélisation de processus métier)
Emporte l'ensemble d'UML, et souvent considéré comme trop complexe
Pas de notion de collaboration comme pour BPMN ou ArchiMate. On peut néanmoins fournir des modèles plus sophistiqués par le biais des constructions des diagrammes de composant

Outils de modélisation UML (y compris Magic Draw) souvent moins ergonomiques que les outils de modélisation des processus métiers en BPMN

De nombreuses similitudes entre les divers langages de modélisation de processus

Diagramme de processus métier en BPMN

Diagrammes d'activité en UML

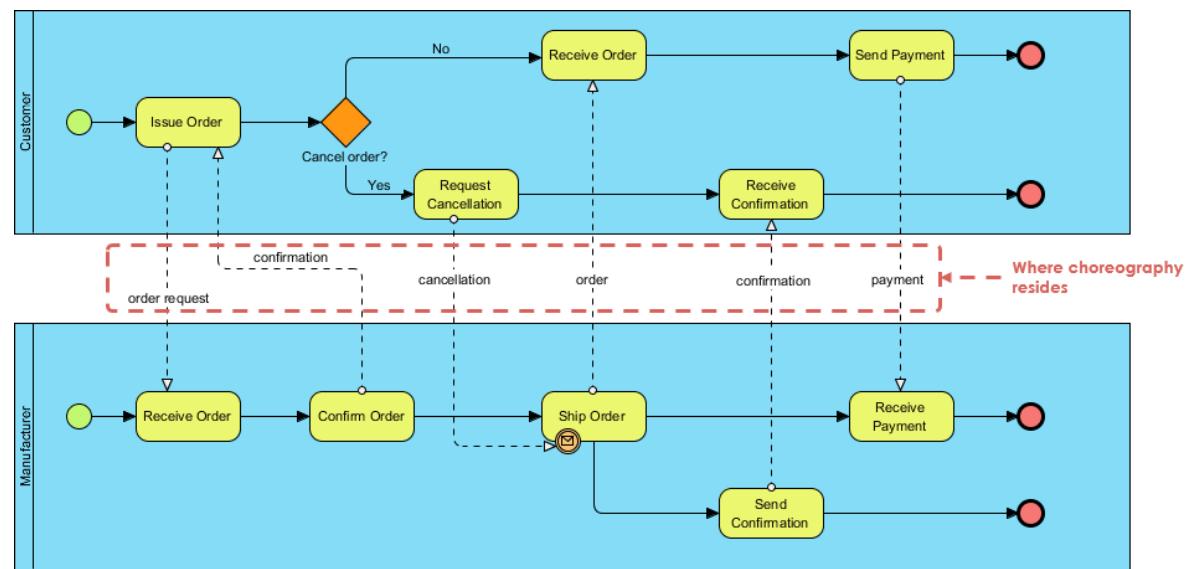
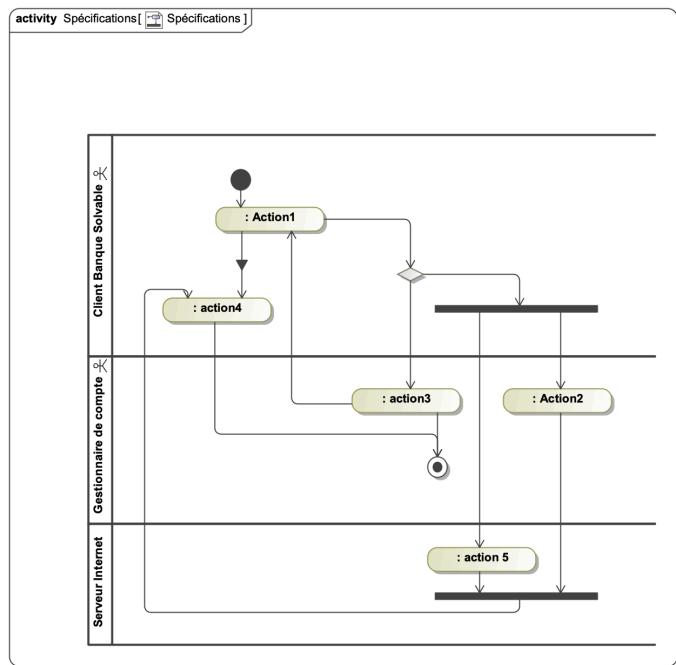
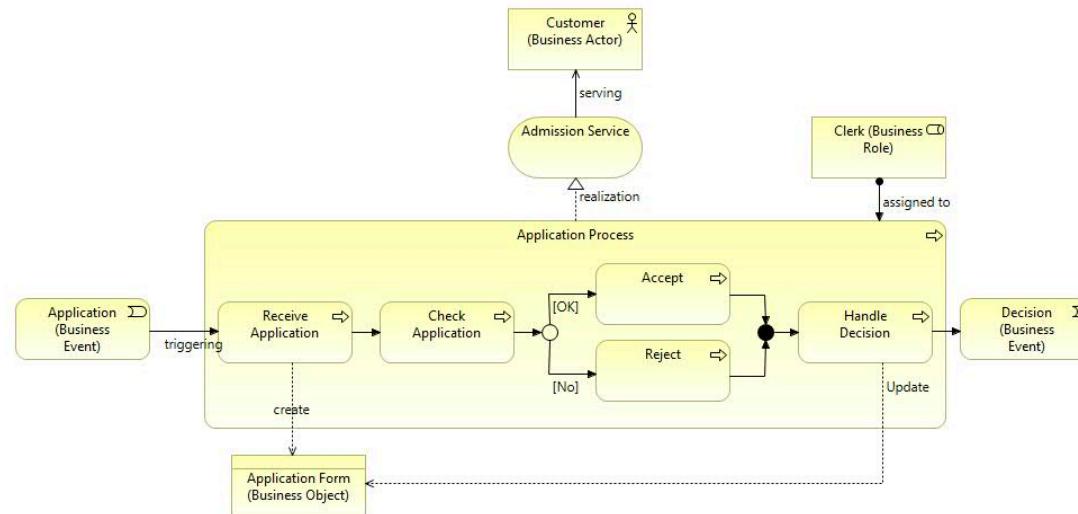


Diagramme d'une vue Processus Métier en ArchiMate



Diagrammes d'activité en UML (ou SysML: ce sont les mêmes)

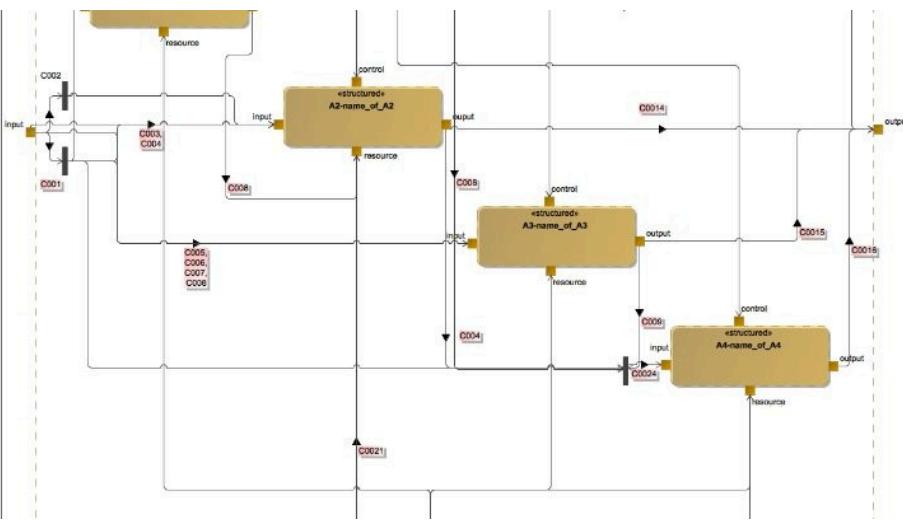


Diagramme d'activité en IDEF0

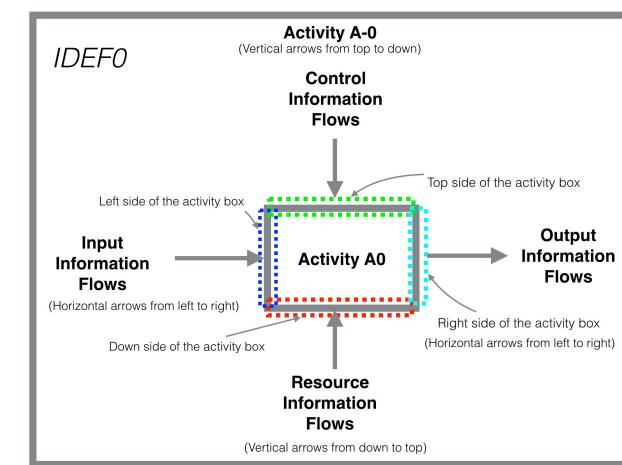
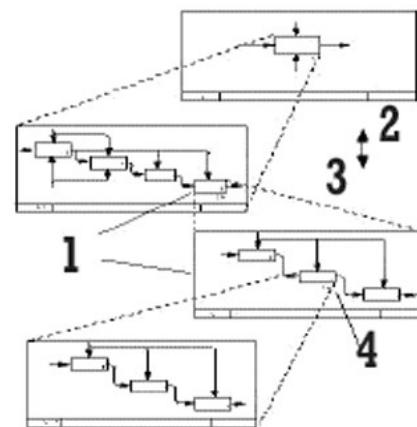
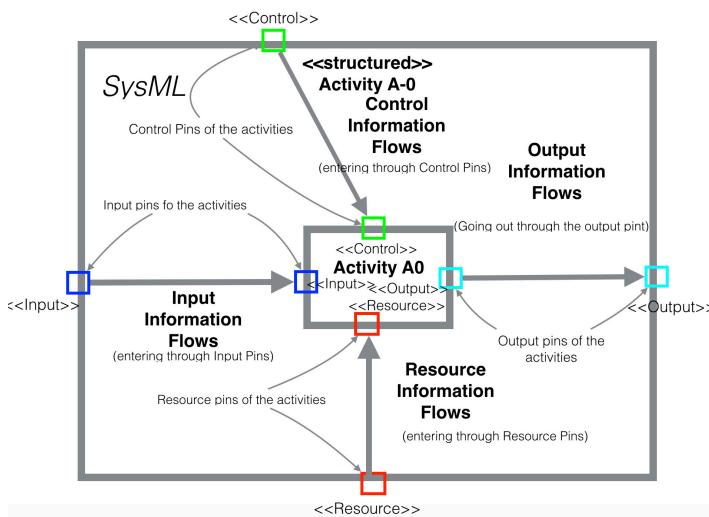
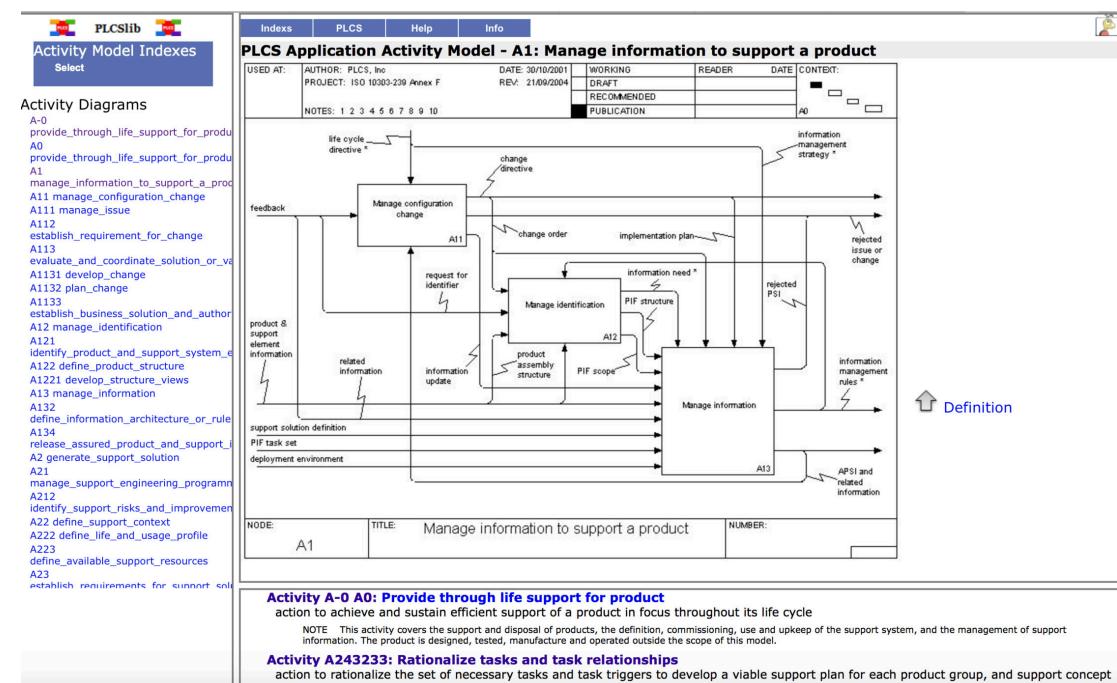


Diagramme d'activités en UML ou autres langages

- ❑ Cela dépend des besoins et du contexte
- ❑ Ces langages se rapproche, et il est possible d'établir des passerelles avec des représentations équivalents.
- ❑ UML reste à la fois spécialisé (métamodèle complet prenant en compte tout le langage) mais parfois très générique (pas de spécificité à un domaine)
- ❑ L'équivalence des langages peut se gérer avec les ateliers de modélisation, permettant la production de modèles polyglottes consistants.

Vers des plateformes plus industrielles

- Base de données centralisée pour contenir tous les modèles et supporter la collaboration
- Serveur Web en complément pour publier et éventuellement acquérir des données par les non spécialistes UML
- Sélection d'une méthodologie et mise en place de pratiques communes
- Gérer les points de vue, pour définir les diagrammes à destination de parti prenants identifiés et afin de répondre à leurs besoins et questions (organisation des vues)
- Gérer le partitionnement physique des éléments du modèle, en fonction de qui doit produire quoi: organisation du contenu du modèle en s'appuyant sur la décomposition en package du modèle.
- Gestion en configuration des divers modules, afin de gérer l'évolution du référentiel des modèles, l'applicabilité et l'obsolescence des modules (considérés comme documents) et des composants représentés (considérés comme produits et articles de configuration).

Plus que la maîtrise du langage UML, passer à cette étape requiert des compétences en ingénierie des modèles et en ingénierie des systèmes logiciels, industriels ou organisationnels basée sur les modèles. Il faut également pouvoir gérer l'échange des données représentant un produit et les processus associés tout au long du cycle de vie d'un produit et entre les partenaires impliqués, qui peuvent avoir des outils et des pratiques hétérogènes. C'est ce qui rend important de s'appuyer sur des normes comme UML, condition nécessaire mais pas toujours suffisante, pour préparer et construire l'interopérabilité requise entre les organisations et leurs plateformes de modélisation.

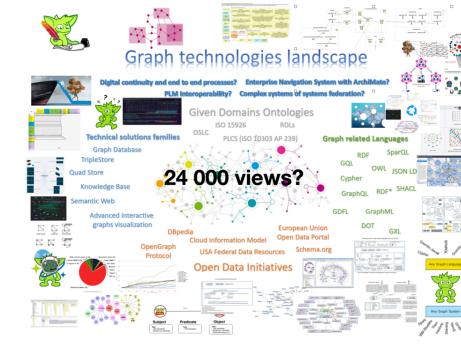
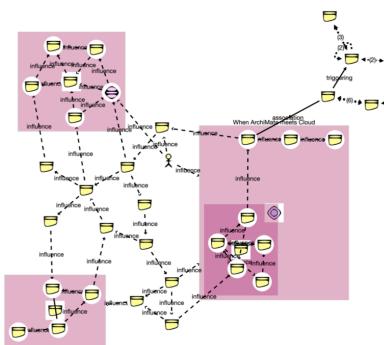
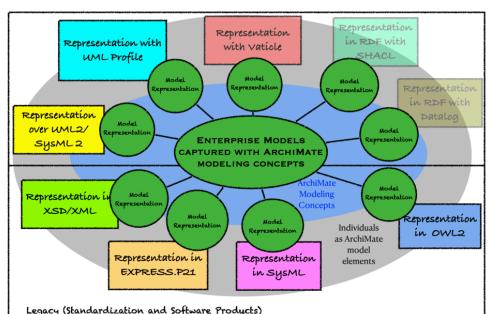
Les types de questions auxquelles il faut savoir répondre

- Quels sont les divers types d'outils UML, de l'outil de dessin à l'atelier de modélisation, en sachant positionner des outils (au moins Yed et PlantUML)
- Dessiner n'est pas modéliser (en sachant expliquer et illustrer)
- Quels sont les objets qu'on retrouve dans les fichiers pour les outils permettant de dessiner des modèles: des primitives du langage UML ou des primitives de dessin?
- Diagramme de séquence: pourquoi est il dynamique?
- Principe des séquences: comment sait on l'ordre d'exécution des échanges des messages? (en prenant en compte blocs et boucles)
- Le langage textuel de description de Plant UML est-il normalisé?
- Avantages d'ateliers de modélisation, illustrés à partir de Magic Draw
- Quelques points de comparaison entre Diagrammes d'activités UML et autres langages
 - UML plus générique: pourquoi?
 - UML plus complet: pourquoi?
- Pourquoi Magic Draw est préférable à Papyrus en termes de création de diagrammes et de consistance globale?
- Quelles sont les caractéristiques des plateformes industrielles?

Perspectives d'avenir

- ❑ Diagrammes dynamiques interactifs, se basant sur des techniques de visualisation plus avancées.
- ❑ Analogie avec évolution carte routière papier/GPS: on se dirige avec des environnements fournissant des diagrammes interactifs dynamiques 2D, 3D ou multi-dimensionnels
- ❑ Liens avec l'analyse des données et la résolution de problèmes en s'appuyant sur la théorie des graphes/réseaux, les réseaux de neurones (deep learning) ou la logique descriptive (ontologies du web).
- ❑ Avec l'évolution d'internet ou les cyber systèmes (exemple voiture autonome connecté à un ville intelligente) qui peut publier et analyser en dynamique des flux de données (data stream) issue de l'observation par des capteurs, modèles reflétant la réalité (jumeaux numériques) et non pas modèles de prescription uniquement.
- ❑ Utilisation de graphes composites pour prendre en compte les descriptions modulaires de systèmes composites
- ❑ Modèles multilingues agrégeant de multiples représentations (hypermodel)
- ❑ Quelques liens:
 - ❑ <https://www.linkedin.com/pulse/some-advanced-visualization-technics-embracing-complexity-figay/>
 - ❑ <https://www.linkedin.com/pulse/semantic-cartography-archimate-first-step-enterprise-navigation/>
 - ❑ <https://www.linkedin.com/pulse/from-archimate-language-web-ontology-dr-nicolas-figay/>
 - ❑ <https://www.linkedin.com/pulse/emerging-landscape-distributed-knowledge-ontology-semantic-figay/>

Hypermodel with ArchiMate



Exploration de l'usage d'UML

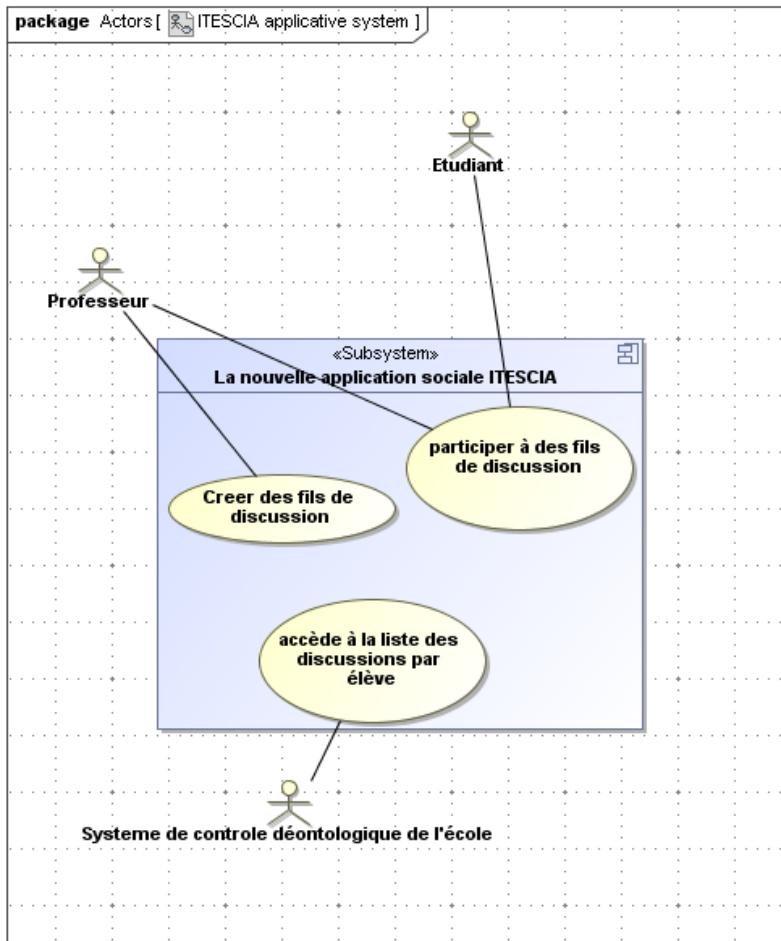
Illustrations avec cas pratiques

Phase d'analyse de besoins (pour l'analyste métier)

- 1 – capturer ce que fait le système
 - => Modèle de spécification à partir d'un diagramme de use case
- 2- Interactions homme/machine pour chaque use case
 - => Modèle d'interaction entre l'utilisateur à partir du front-end (frontal) de l'application et l'application
 - Possibilité 1 – diagramme de séquences(pas la préférée)
 - Possibilité 2- diagramme d'activité capturant les enchainements d'écran, les déclenchements d'action, les flux d'entrée sortie associés aux transitions (recommandée)
- Utiliser un diagramme de composant plutôt qu'un diagramme de classes – plus proche du paradigme composant
- Faire attention à la manière d'utiliser composants et instances de composant
- Faire attention entre la partie logique et la partie physique, entre le fonctionnel et la réalisation – difficile quand on décrit un système existant qui va évoluer, tous les composants ne sont pas au même niveau de réalisation.

Capturer ce que fait le système

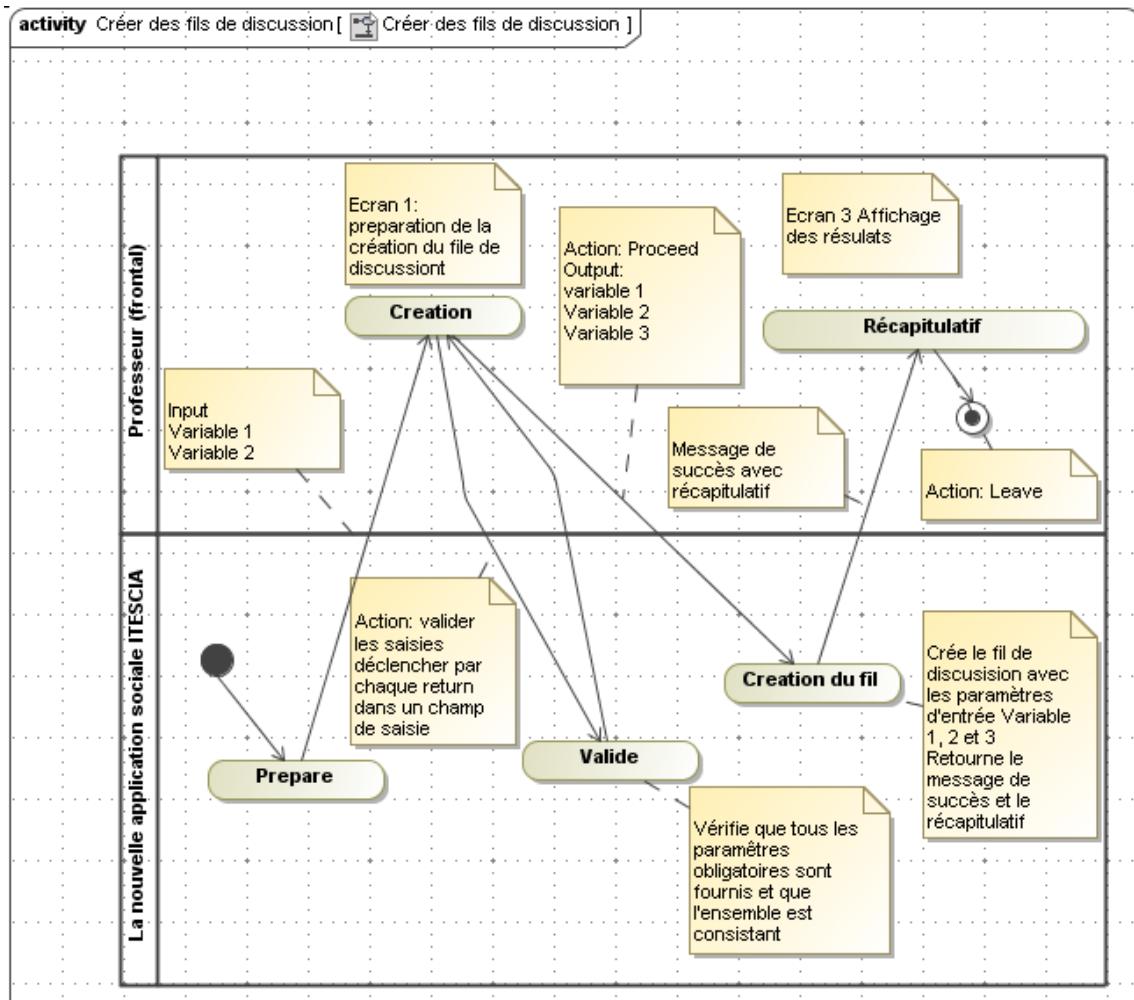
Diagramme de use case



- ❑ Le système à considérer, la nouvelle application
- ❑ Les divers acteurs concernées: professeurs et étudiants
- ❑ Le cas d'utilisation capture ce que l'acteur fait avec le système:
 - o Ex: Le professeur crée des fils de discussion avec la nouvelle application sociale de l'université Lyon 1
- ❑ On peut éventuellement ajouter des acteurs qui sont des systèmes externes à interconnecter et qui consommeront des services proposés par l'application
 - o Le système de contrôle déontologique accède à la liste des discussion des élèves par le biais de l'application sociale

Les interactions hommes machine

Diagramme d'activité



- Le diagramme est associé au cas d'utilisation « crée des fils de discussion »
- Des actions sont réalisées par le système, qui donne ensuite la main à l'utilisateur par le biais d'écran affichant des informations et en réclamant, et fournissant des déclencheurs pour des actions
- Les information à afficher sont capturée par des variables associées aux flèches entrantes pour les activités liées à un écran, et qui sont une sortie de la fonction applicative
- Les informations à modifier et qui seront des entrées d'une fonction du système sont sur les flèches sortantes, chaque action une flèche sortante et un traitement.
- Toute la logique de l'interaction est capturée, les maquettes d'écran sont des mises en forme visuelles de l'écran, réalisée par des Designer d'écran. Dans le cas du Web, ce sont des Web designer qui crée des feuilles de style pour la présentation des informations affichées

Eléments découverts

- Le cas d'utilisation
- L'acteur
- L'activité

Un cas pratique

- ❑ Informatisation gestion des achats
- ❑ Nouveau système automatisant les achats
 - o Demande d'achat par une personne dans l'entreprise
 - o Le service des achats cherche parmi les fournisseurs dans sa base - si non trouvé, recherche de nouveau fournisseurs et enregistrement dans la base des fournisseurs
 - o Passage de la commande
 - o Réception de la commande, qui est livrée et réceptionnée. Si conforme à la commande, le paiement est déclenché et une ligne de dépense est ajoutée automatiquement par le nouveau système aux lignes comptable
- ❑ Capturer et valider les besoins de l'utilisateur, et ce que doit faire le système pour les achats, à partir de diagrammes UML appropriés: use cases, activité.,collaboration/séquences.
- ❑ Chaque élève livrera un petit document de capture des besoins avec les diagrammes produits et une description, en assurant sa cohérence
- ❑ Objectif: prise en main d'un outil permettant de produire des diagrammes UML pour fournir ce mini dossier.
- ❑ Contexte particulier à l'introduction à UML pour Université Lyon 1 année 2021-2022
 - ❑ usage de Observable avec PlantUML, selon l'approche travailler lors des TPs
 - ❑ Tout mettre en français

Préparer l'environnement pour pouvoir travailler sur Internet

- Pour ceux qui n'ont ni compte Google, ni compte Github, en créer un
- Puis pour tous, s'inscrire sur Observable avec un de ces comptes
 - <https://observablehq.com/>
 - Sign in
 - vous signer avec un compte existant, google ou GitHub ...
- Créer un Notebook avec la liaison plantUML
 - coller la ligne : import {plantuml} from « @karlerasmus/plantuml-in-observable » dans une cellule
- Nous sommes prêt à travailler avec plantUML!

The screenshot shows the Observable platform interface. At the top, there's a navigation bar with links for Explore, Learn, Community, and Search. On the right, there's a 'New' button and a user icon. Below the navigation, the user's profile picture and name 'Nicolas Figay' are displayed, along with their GitHub handle '@509be501d75d5017'. A sidebar on the left lists navigation options: Home, Profile, Notebooks, Collections, Suggestions, Likes, Trash, and Settings. The main area is titled 'Notebooks' and displays a message: 'No notebooks found.'

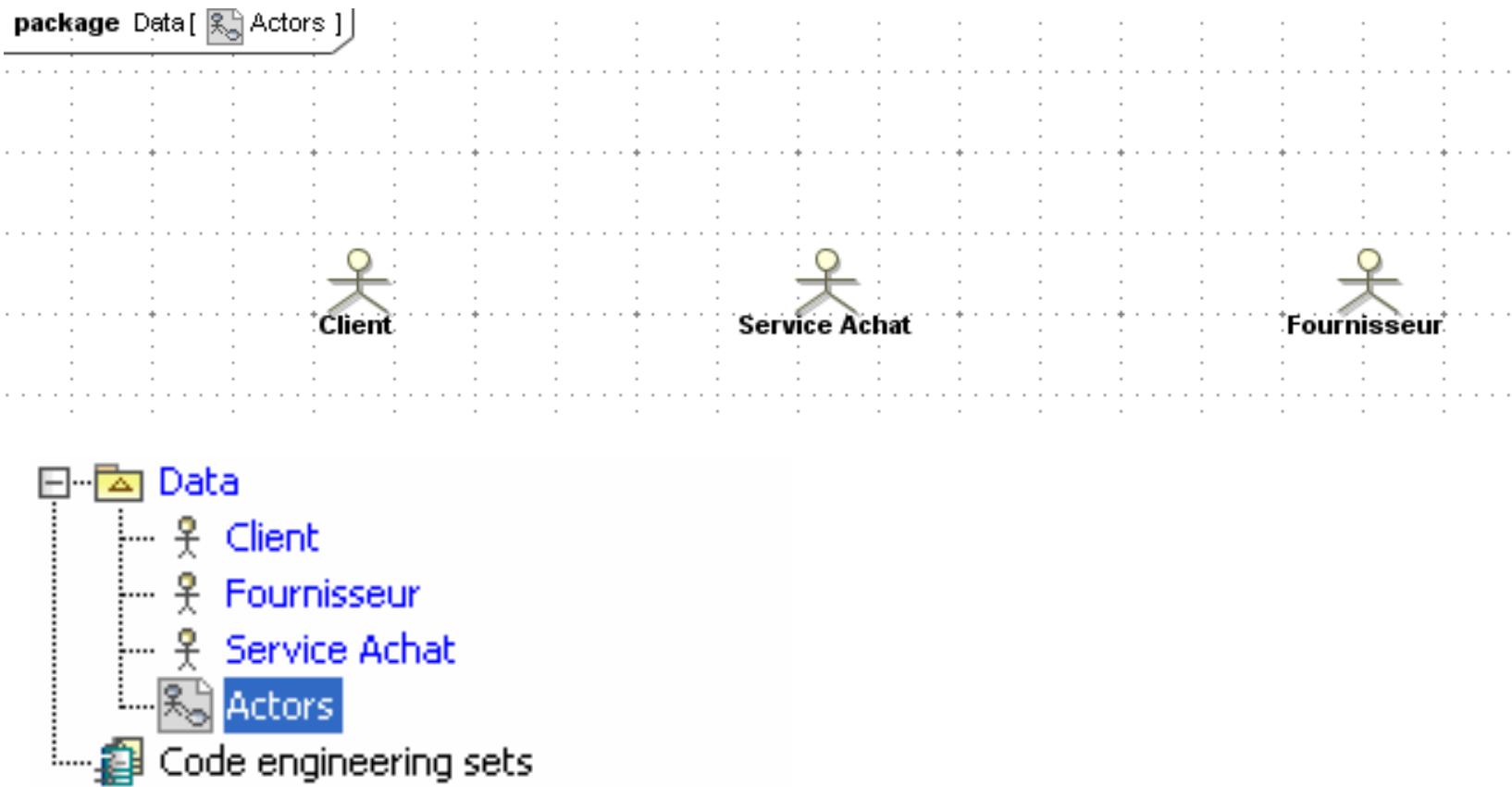
This screenshot shows a specific notebook titled 'Nicolas Figay TP UML'. The top part of the screen shows the Observable header with the user's profile and a 'Publish...' button. The notebook title is 'Nicolas Figay TP UML'. Below the title, there are two code cells. The first cell contains the code: 'import {plantuml} from "@karlerasmus/plantuml-in-observable"'. The second cell also contains the same code. The interface includes standard code editor features like copy, paste, and expand/collapse buttons.

Exemple de réalisation du cas

- Réalisé avec Magic Draw 17.0 comme atelier de meta-modélisation UML
- Alternatives pour ce type d'outil: Rational, Modélio 2, Topcased, Sparx Enterprise Architect, Papyrus, Visual Paradigm ...
- Pour l'introduction UML Université Lyon 1 2021-2022, l'outil utilisé a été PlantUML, sur l'environnement Observable.
- Motivation du choix: cours d'introduction, trop court pour la prise en main d'un atelier de modélisation
- Note: rien n'empêche de comparer votre expérience avec PlantUML et celle reflétée dans les pages qui suivent.
- Lien sur la transposition dans ce nouvel environnement. Cf. le document d'exemple à l'issu du TP: <https://observablehq.com/@506be501d75d5017/nicolas-figay-tp-uml>

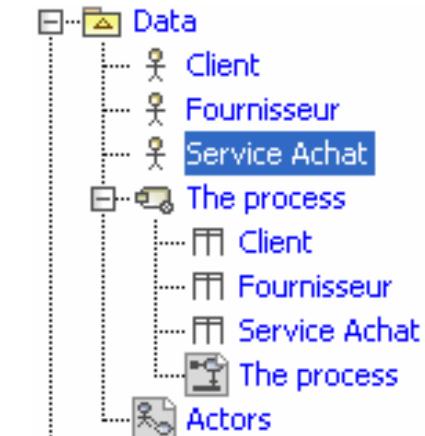
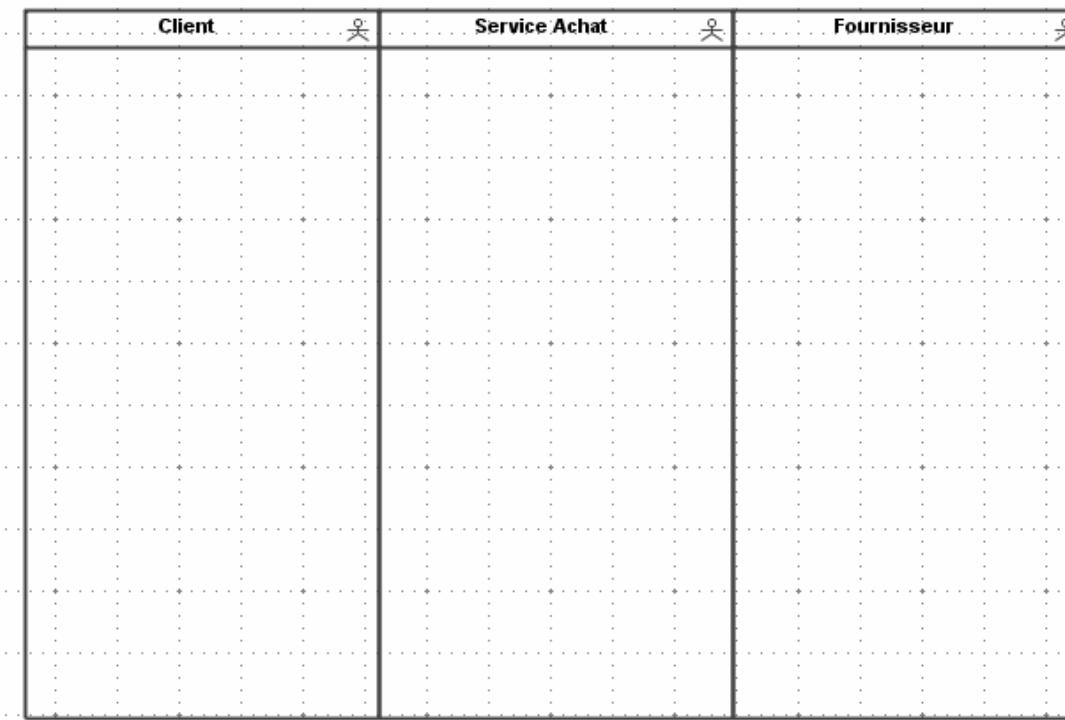
1 – identification des acteurs

- Diagramme de Use Case et modèle associé



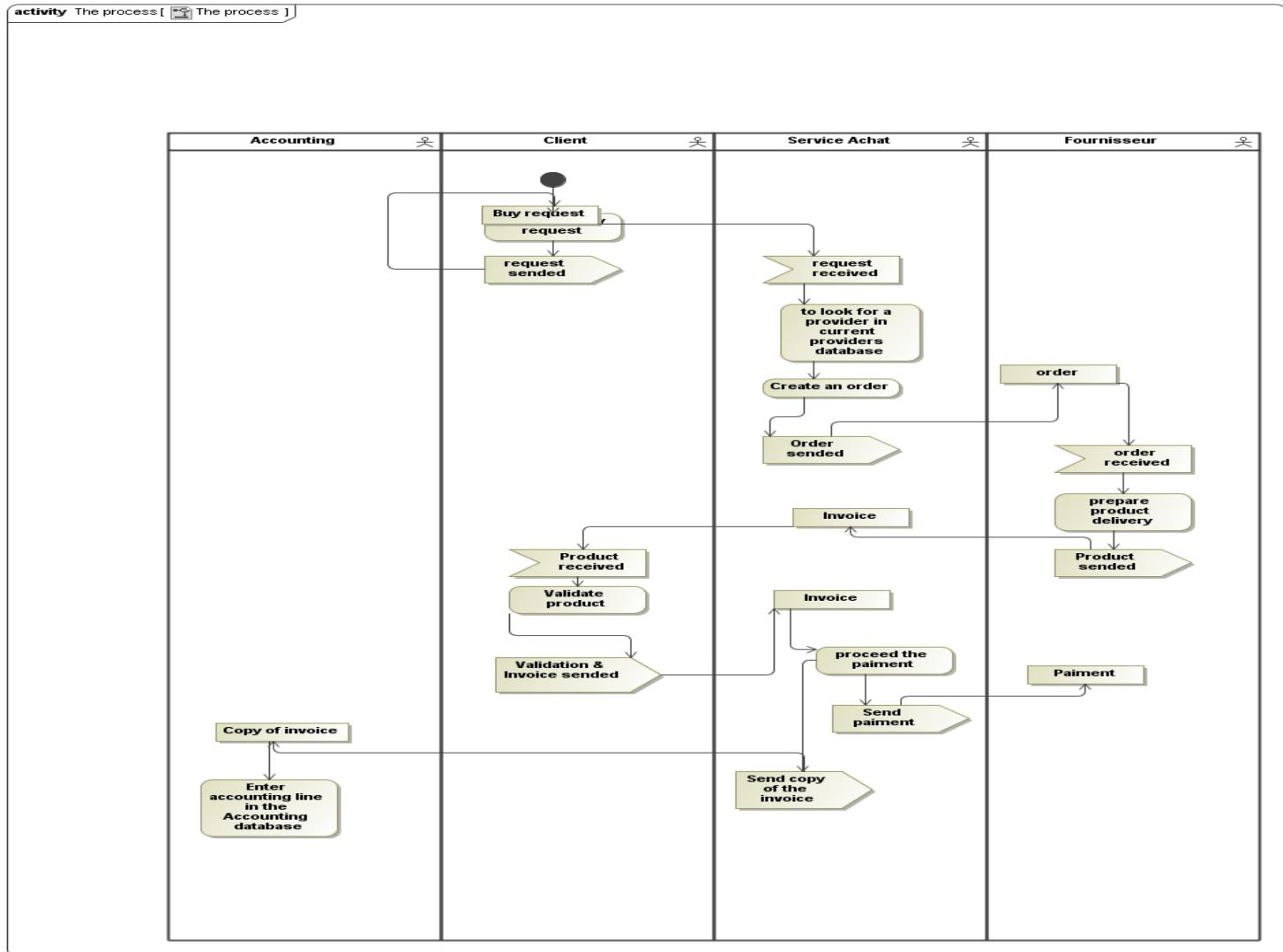
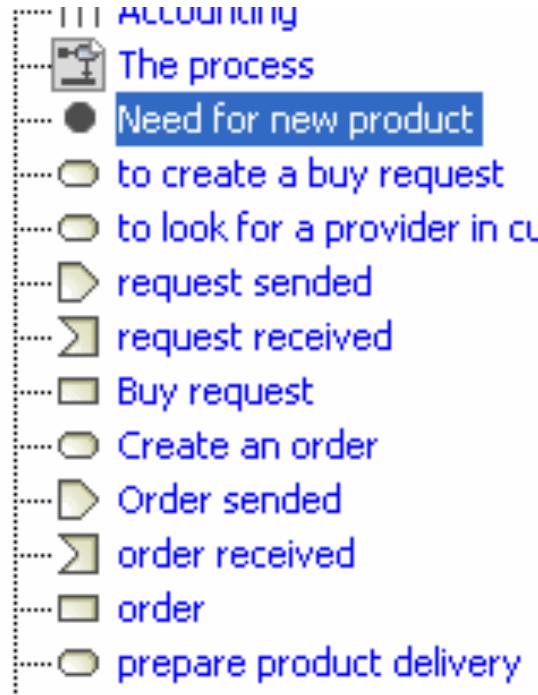
2 – Capture du processus (a)

- Sur un diagramme d'activité, création des « swim lanes », avec attribution des acteurs à chaque ligne



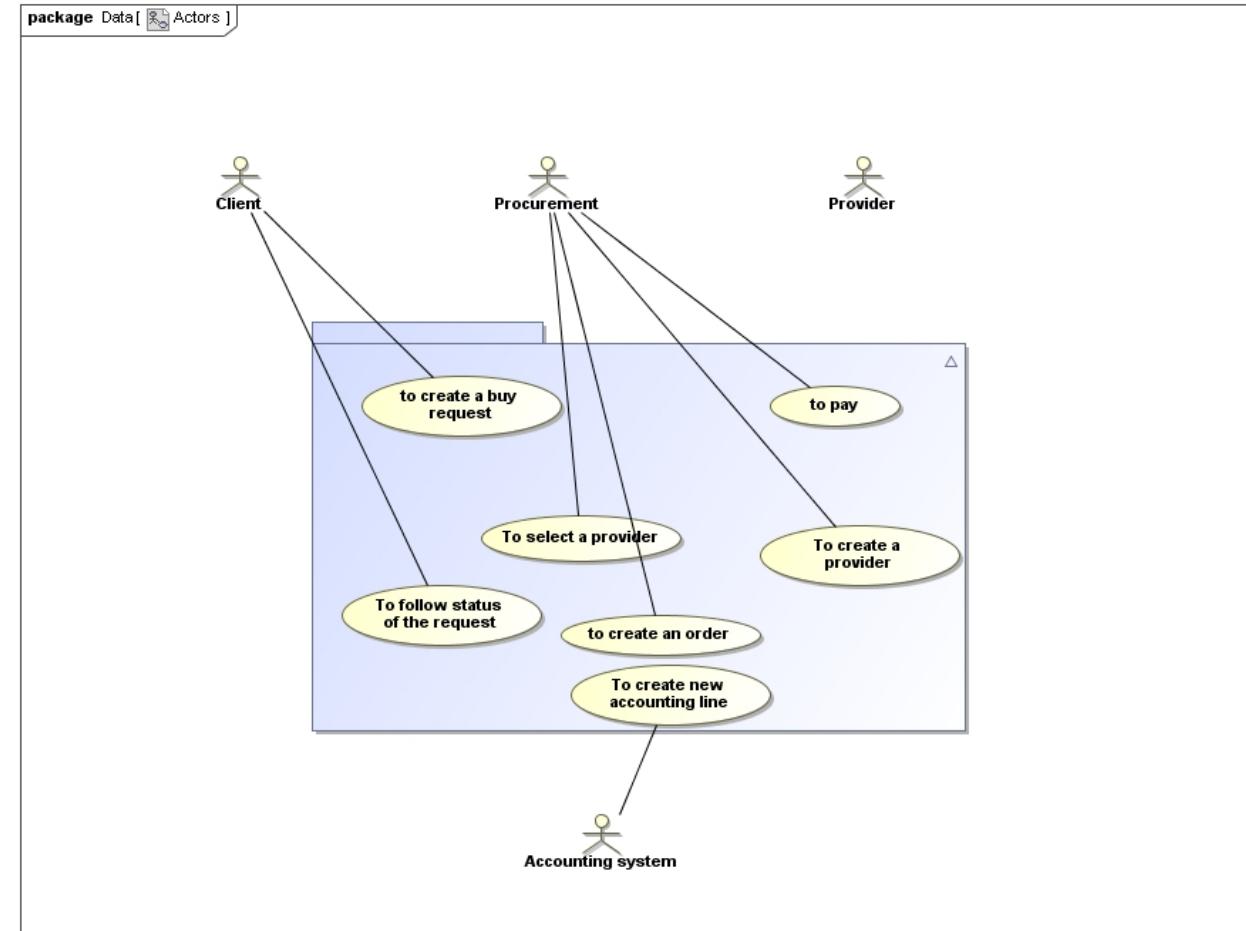
2 – Capture du processus (b)

- Sur un diagramme d'activité
- (cf. commentaires)



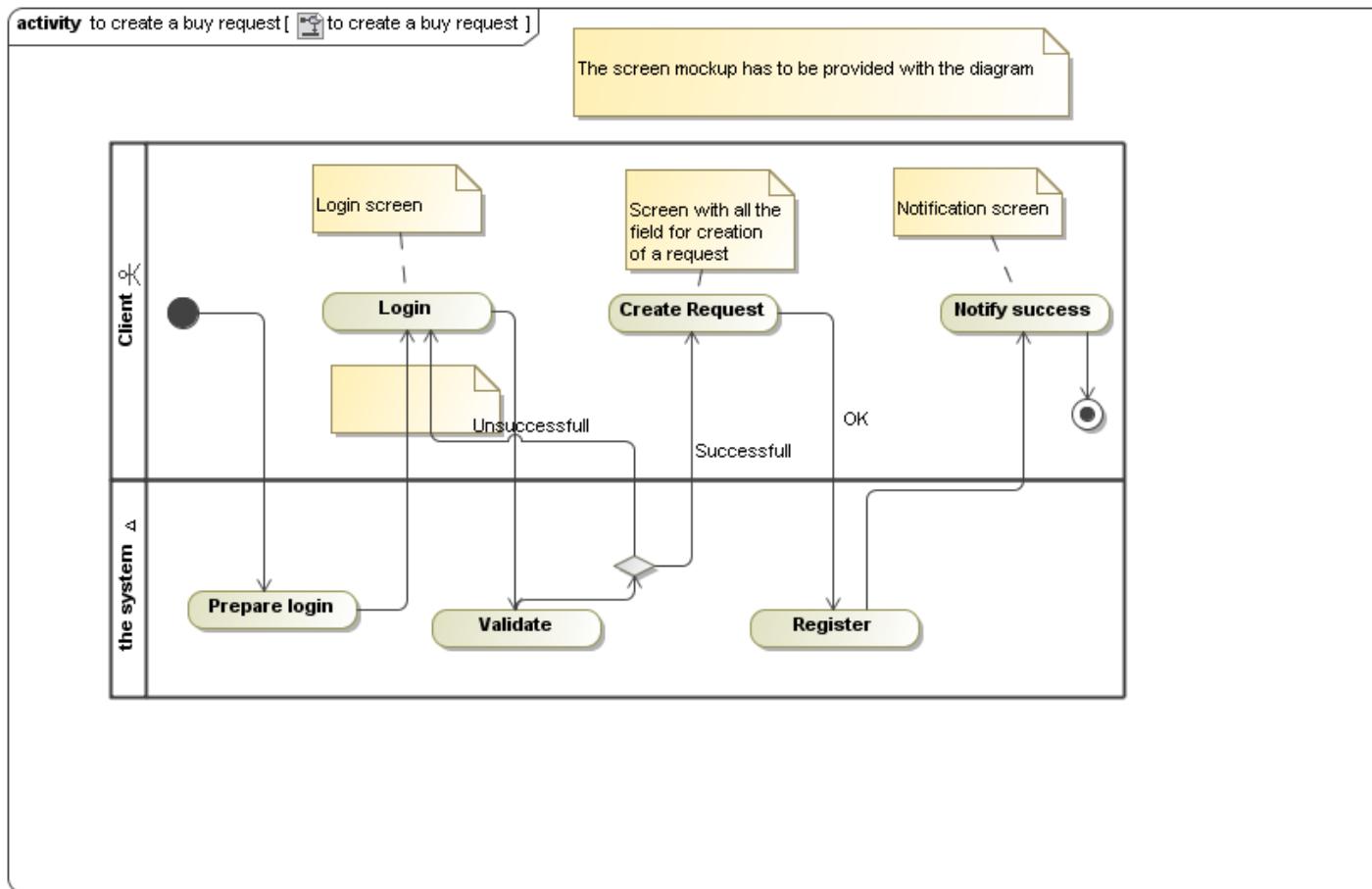
3 – Capture des cas d'utilisation

- Sur un diagramme de cas d'utilisation pour validation avec le client de ce que doit supporter le nouveau système



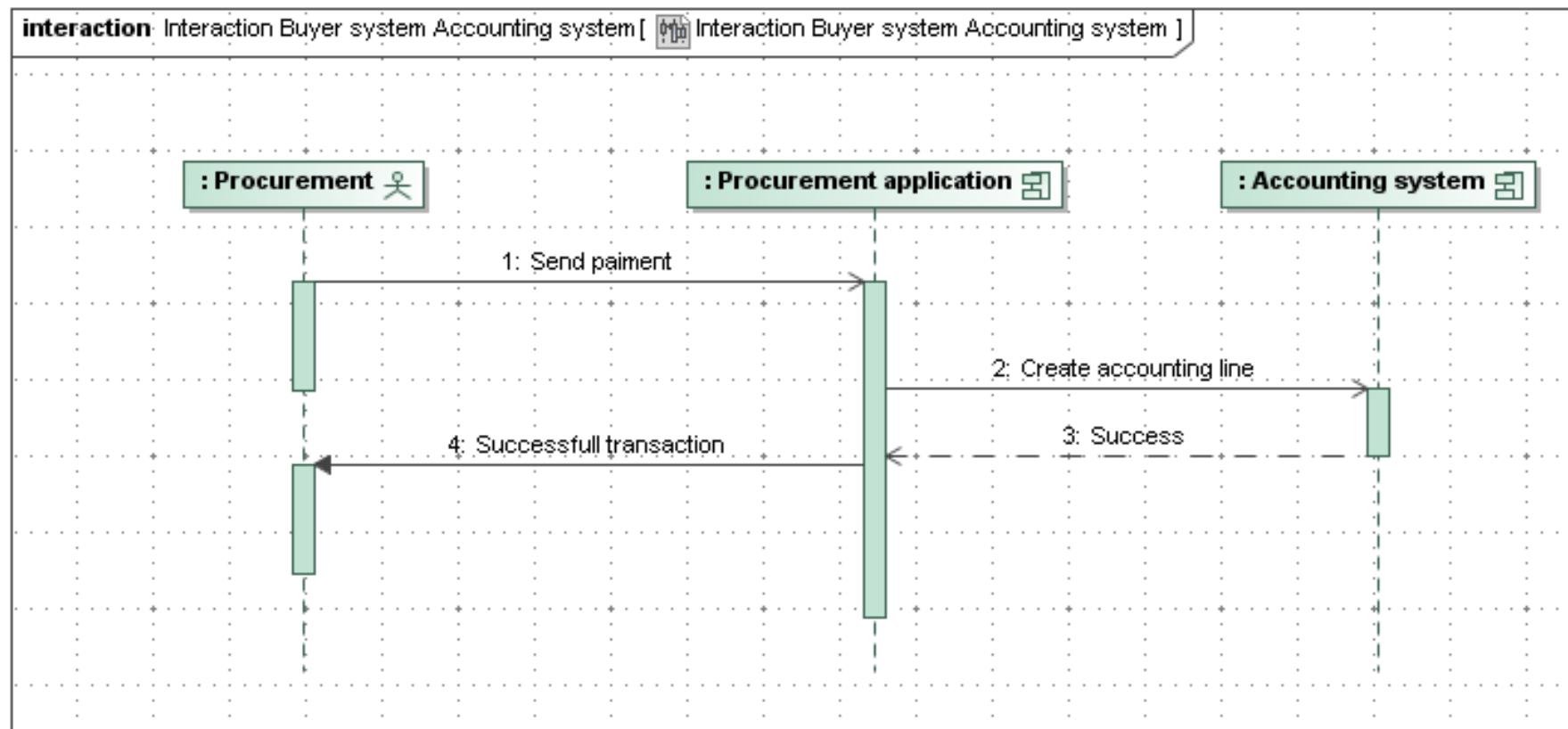
4– Détailer chaque cas d'utilisation

- ❑ Afin de définir les diagrammes d'interaction homme machine



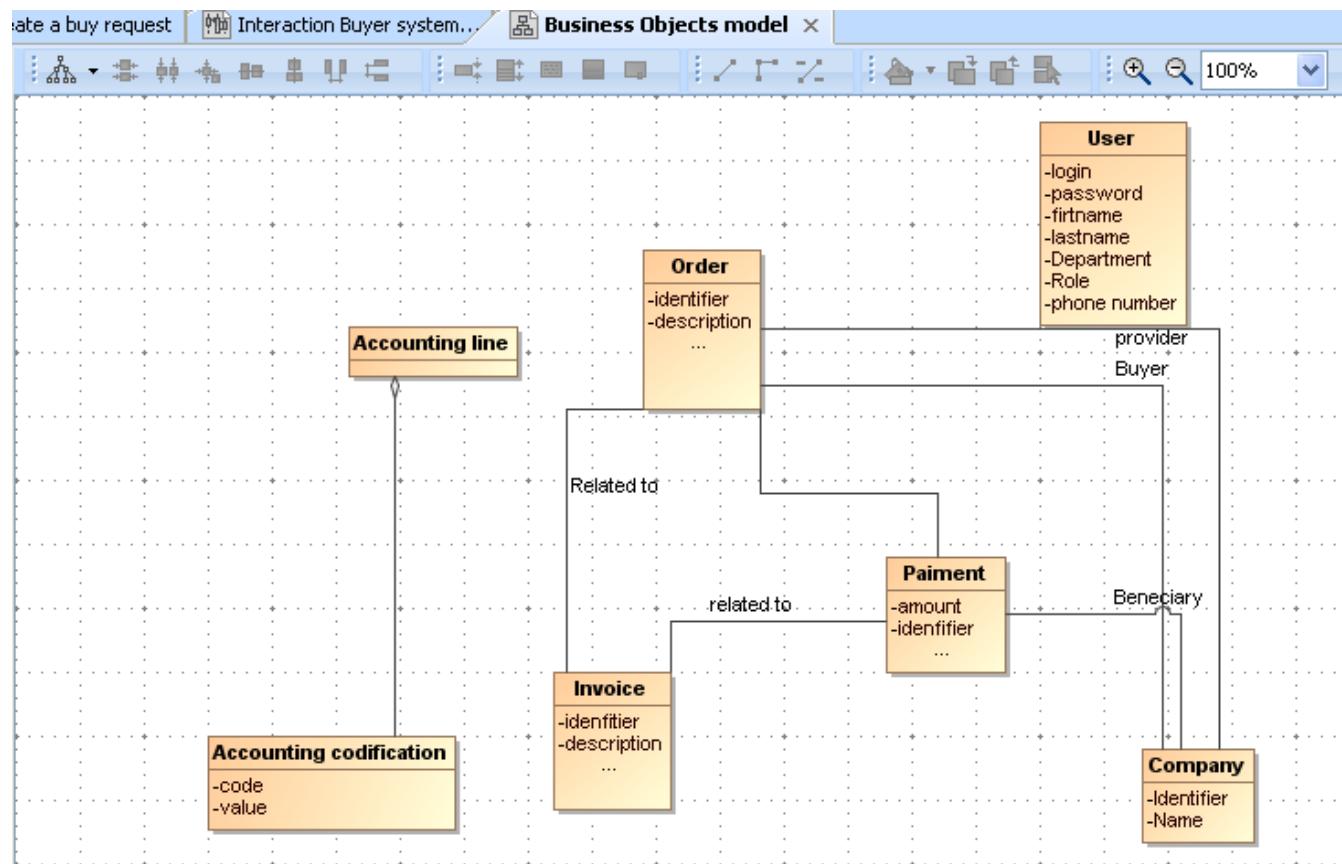
5– Collaboration entre composants

- ❑ Afin de capturer les besoins de communication entre le nouveau système et l'application de comptabilité – diagramme de séquence



6– Capture des entités métier

- ❑ Diagramme de classes – sans les méthodes!!!!
- ❑ Business Object Model
- ❑ Plus lié à définir ce qui apparaitra sur l'interface homme machine – et la documentation utilisateur - qu'à ce qui sera dans la base de données (sera adressé par conception et implémentation)



Avantages de la modélisation

- Représentations visuelles avec une notation partagée par divers acteurs
- Pour les ateliers mais pas pour plantUML
 - Un élément du modèle sur plusieurs diagrammes ou plusieurs fois sur un seul diagramme => mise à jour systématiques sur tous les diagrammes quand modification de l'objet du modèle
 - Capture des besoins de façon plus méthodique rendu possible et partageable (on s'appuie sur des méthodologies de type « eXtreme Programming », « Unified Process », « SCRUM », etc.
 - L'outil contrôle que les règles du langages sont respectées

Quelques difficultés

- ❑ Tous les types d'objets ne peuvent apparaître ensemble sur un diagramme
- ❑ 250 constructions de modélisation
- ❑ Hétérogénéité des outils sur les éléments du modèles qui peuvent apparaître sur un diagramme
- ❑ Tous ce que permet de représenter PlantUML, ou tout outil UML, n'est pas toujours conforme à la norme. Il existe des différences notamment en termes d'ergonomie et de contraintes imposées par les environnements qui facilite la vie de l'utilisateur, à partir de contraintes non transposée à l'identique dans tous les environnements (problème d'interchangeabilité)
- ❑ Certains éléments peuvent être modéliser de plusieurs manières: acteur automate, composant
- ❑ Plusieurs manières de modéliser – quelles sont celles à adopter?
- ❑ Formaliser une méthode, et utiliser les ateliers pour contraindre et valider la modélisation, est possible
- ❑ Difficulté similaire à l'apprentissage des langages de programmation – courbe d'apprentissage importante, règles de modélisation à établir, pratiques communes à établir. On peut partir d'approches méthodologiques qui capturent des meilleures pratiques. De même, les environnements de programmation n'implémentent pas toujours de la même manière les langages normalisés (couverture, contrainte, etc.).
- ❑ Il faut connaître les particularités des divers outils pour être performants.

Cours sur le site ...

- www.eads-iw.net/web/training/uml
- Adresse de contact: Nicolas.figay@gmail.com
- Références: <http://www.uml-diagrams.org>

Phase de conception (pour l'architecte - fait partie d'UML avancé)

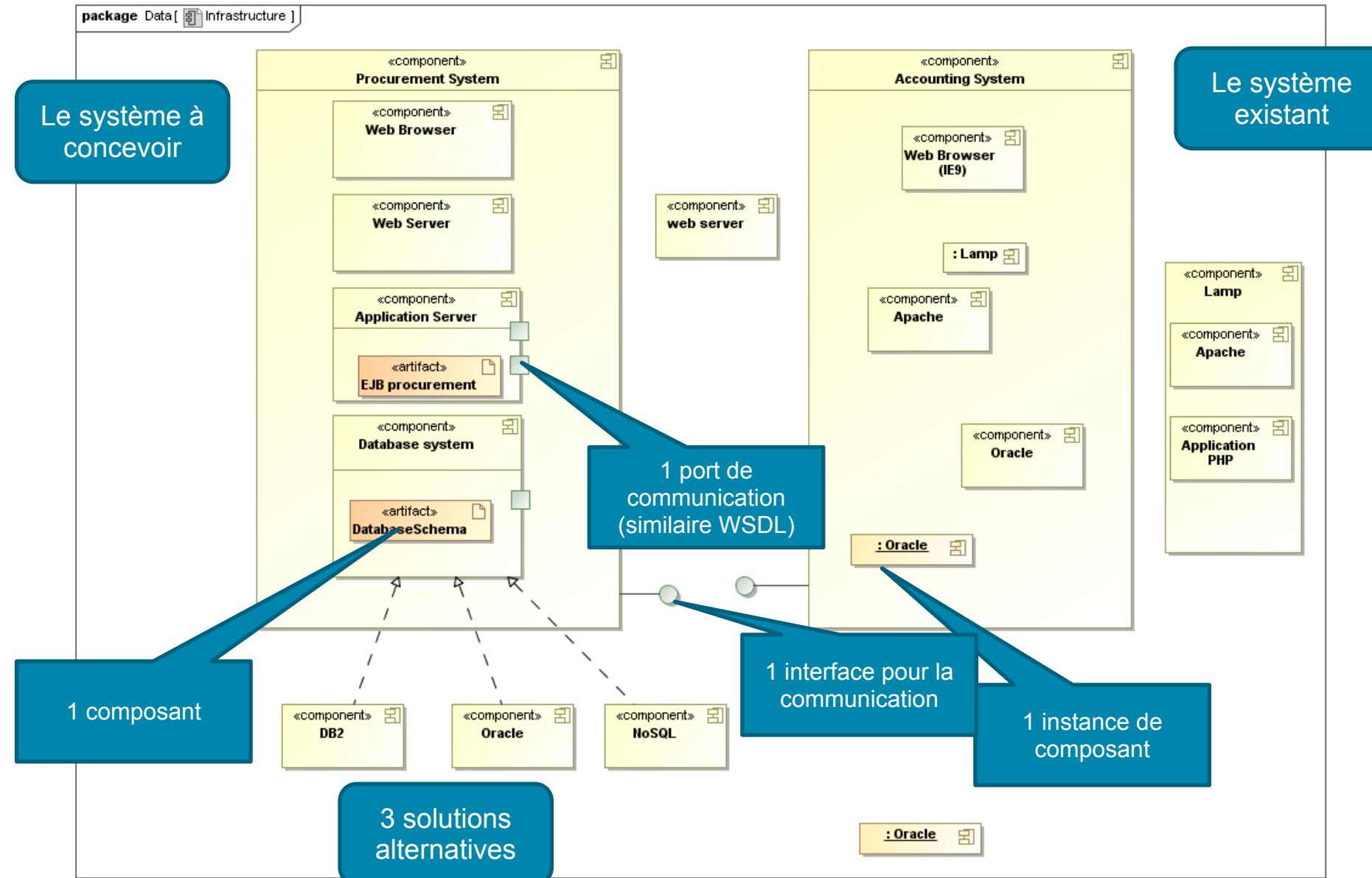
- ❑ 1 – capturer les diverses solutions techniques possibles et leur lien avec le système existant
=> Modèle d'architecture technique à partir d'un diagramme de composant
- ❑ 2- Définir l'architecture détaillée de la solution retenue
 - Les composants à intégrer et à déployer
 - Les composants à développer
 - o Modèle d'architecture à partir de diagrammes:
 - De composants
 - De déploiement
 - De collaboration
 - o Lien avec l'architecture fonctionnelle – use cases

Capture des solutions techniques

- ❑ Il s'agit, à partir de besoins non fonctionnel (le comment le système fait ou est fait – et non ce qu'il permet de faire), de pouvoir faire le choix entre plusieurs solutions techniques de haut niveau avant de commencer la conception détaillée.
- ❑ Exemples de contraintes:
 - Application accessible via navigateur internet pour des utilisateurs distribués dans plusieurs sites et plusieurs pays
 - Volumétrie de données très importante + temps de réponse de l'ordre de quelques secondes: plusieurs téraoctets
 - Utilisations d'applications déjà disponibles dans l'entreprises, ou étant des standards pour l'entreprise -ex: Oracle est le SGBD préconisé pour l'entreprise X
- ❑ Ces contraintes impactent le dimensionnement du projet, et amènent à devoir faire dès le début le choix de blocs technologiques/ de solutions techniques, assez rapidement avant de passer à la conception détaillée.
- ❑ Outre que ces choix vont dimensionner le travail à suivre et les ressources à mettre en œuvre, les solutions techniques possibles peuvent calibrer le projet en terme de ressources à acquérir, et doivent être valider au niveau du pilotage de projet pour vérifier que le budget du projet est réaliste

Illustration phase 1

Solution technique sous forme de diagramme de composant



Un cas pratique

- Dans le cadre de l'informatisation des achats, un certain nombre de contraintes techniques ont été données:
- Le service des achats est distribués sur plusieurs établissements, distribués dans plusieurs pays. Tous doivent pouvoir accéder à l'outil à partir du navigateur internet
- On ne veut pas acheter un progiciel des achats, mais on souhaite s'appuyer sur des environnements de développement de solutions d'entreprise
- L'application des achats est réalisée sur la base d'une application n-tiers, accessible à partir d'IE9, et s'appuyant sur un serveur Apache, Tomcat et Oracle 8i
- On souhaiterais un serveur d'applications pour l'application des achats, pour la flexibilité, l'agilité et la robustesse de ces technologie.
- Proposer deux ou trois solution en terme d'architecture générale, en utilisant un diagramme de composant.
- Il faut utiliser des composants, des instances de composant, des interfaces, la composition
- Il faut vérifier avec l'outil de modélisation que vous utiliser si le modèle et le diagramme sont alignés en ce qui concerne la modélisation et la visualisation des composants.
- Avec les mêmes groupe que la dernière fois, faire le modèle et réaliser une capture des diagrammes avec quelques notes explicatives

Quelques difficultés

- Faire la distinction en composant et instance de composant, entre composant/instance qui existe et qui va exister
- Les diagrammes et le modèle ne sont pas alignés
- Tous les outils ne permettent pas de dessiner des composants imbriqués (ex Modelio 2 qui par contre assure la cohérence entre le modèle et le diagramme)
- Pratiquer la modélisation de diagramme de composants en utilisant les diverses constructions UML et en comprenant pour vous utilisez telle ou telle construction du langage
- Qualifier l'outil utilisé, repérer les problèmes et donner des règles de validation des modèles. Elles peuvent éventuellement être contraintes par les atelier UML avec des profiles, des templates ou des règles OCL si l'outil peut les valider.
- Evaluer les outils avant de les choisir après un parcours complet de ce que l'on attend, sur les capacités de modélisation visuelle, la consistance, etc.

Définition de l'architecture du système à concevoir

- ❑ Il s'agit de structurer le système en sous éléments inter-reliés:
 - o Structurellement
 - o Dynamiquement
- ❑ La décomposition peut être:
 - o Fonctionnelle
 - o Physique
- ❑ Les composants attendus seront spécifiés.
- ❑ Les composants disponibles seront évalués et qualifiés.
- ❑ Les composants manquant seront développés puis testés au travers de tests unitaires
- ❑ L'ensembles de composants seront intégrés et des tests d'intégration seront réalisés
- ❑ La validation fonctionnelle sera ensuite réalisé, par le fournisseur du système puis par le client.

Outils de l'architecte: les patrons d'architecture et de conceptions(design patterns)

- ❑ Architecture
- ❑ Structuration globale en paquetages <= couches, Model/View/Controller, client serveur, multi-tiers, ...
- ❑ Conception: structuration détaillée en classes
- ❑ Attributs et opérations des classes: qui doit savoir, qui doit faire?
- ❑ Relation entre classes: délégation, héritage, réalisation,...?
- ❑ Description d'un patron de conception
 - o Nom (lié au vocabulaire de conception)
 - o Problème: description du sujet à traiter et de son contexte
- ❑ Solution: description des éléments, de leur relations/coopérations et de leurs rôles dans la résolution du problème
 - o Description générique
 - o Illustration par des exemples
- ❑ Conséquences: effets résultants de la mise en œuvre du patron
 - o Complexité en temps/mémoire, impact sur la flexibilité, portabilité...

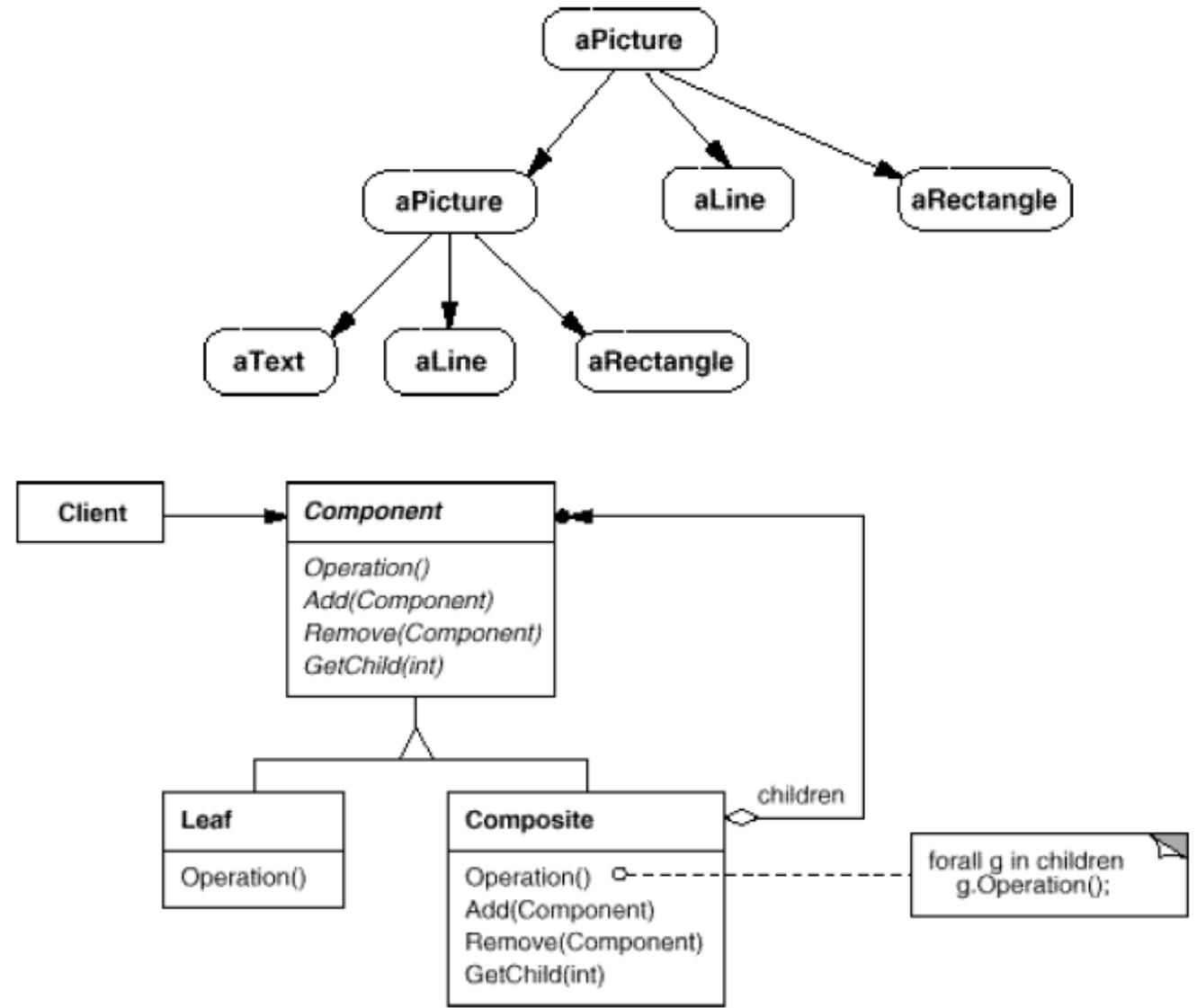
23 patrons du gang des quatres (Gang of Four: E.Gamma, R.Helm, R.Johson, J. Vlissides)

- ❑ Patrons de création : abstract factory, factory method, singleton, Prototype, Builder
- ❑ Patrons comportementaux: Iterator, Strategy, State, Observer, Command, Visitor, Chain of responsibility, Interpreter, Mediator, Memento, Template method

- ❑ Les patrons peuvent être décrits en UML
- ❑ Les outils UML peuvent fournir des bibliothèques de patron

Exemple de représentation de patron: composite

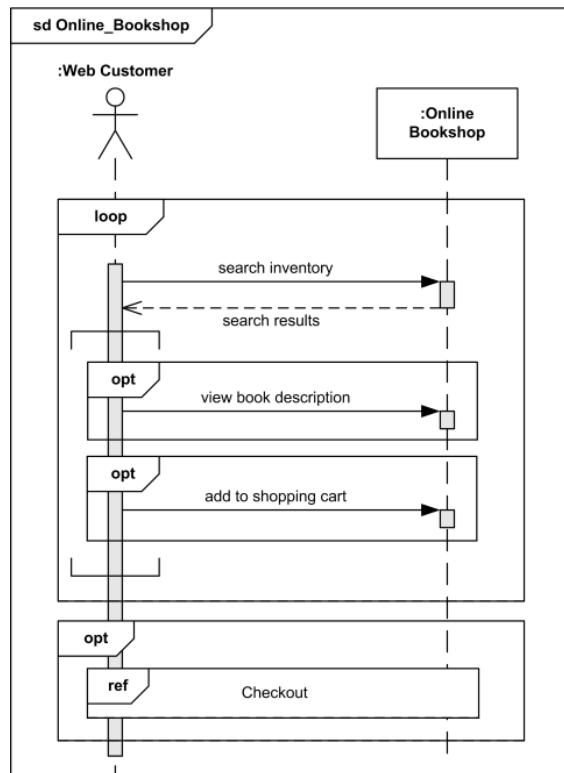
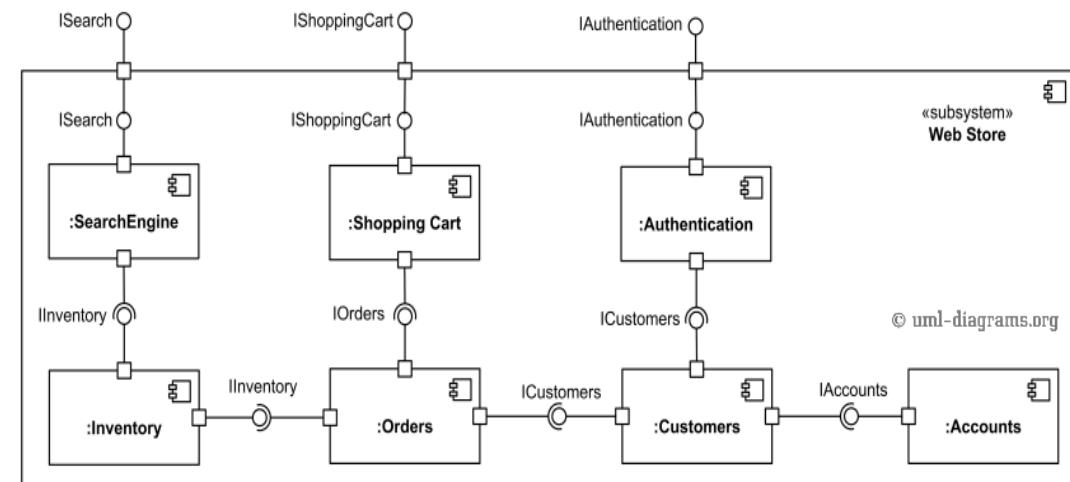
- Intention
 - Composition d'objets en structures arborescentes pour représenter des structures composant/composé.
 - Permettre au client de traiter de la même manière les objets atomiques et les combinaisons de ceux-ci.
- Indications
 - On souhaite représenter des hiérarchies d'individus à l'ensemble
 - On souhaite que le client n'ait pas à se préoccuper de la différence entre combinaisons d'objets et objets individuels



Diagrammes pour les patrons d'architecture

Composants et Séquences

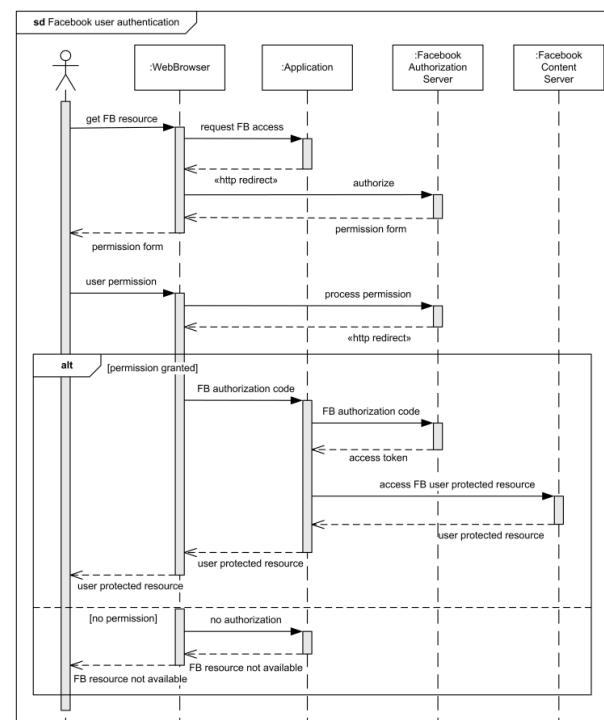
- Un exemple de diagramme de composants UML pour les achats en ligne sur un site Web de vente au détail.
- Le composant du moteur de recherche utilise l'interface d'inventaire pour permettre aux clients de rechercher ou de parcourir des articles. Le composant Panier d'achat utilise le composant Commandes pendant le processus de paiement. Le composant d'authentification permet au client de se connecter et lie le client au compte.



Un exemple de diagramme de séquence qui montre comment l'utilisateur de Facebook (FB) pourrait être authentifié dans une application Web pour permettre l'accès à ses ressources FB. Facebook utilise le cadre de protocole OAuth 2.0 qui permet à l'application Web (appelée "client"), qui n'est généralement pas le propriétaire de la ressource FB mais agit au nom de l'utilisateur FB, de demander l'accès aux ressources contrôlées par l'utilisateur FB et hébergées par le serveur FB . Au lieu d'utiliser les informations d'identification de l'utilisateur FB pour accéder aux ressources protégées, l'application Web obtient un jeton d'accès.

L'application Web doit être enregistrée par Facebook pour avoir un identifiant d'application (client_id) et un secret (client_secret). Lorsqu'une demande à certaines ressources Facebook protégées est reçue, le navigateur Web ("agent utilisateur") est redirigé vers le serveur d'autorisation de Facebook avec l'ID d'application et l'URL vers laquelle l'utilisateur doit être redirigé après le processus d'autorisation.

L'utilisateur reçoit en retour le formulaire de demande d'autorisation. Si l'utilisateur autorise l'application à récupérer ses données, le serveur d'autorisation Facebook redirige vers l'URI qui a été spécifié auparavant avec le code d'autorisation ("chaîne de vérification"). Le code d'autorisation peut être échangé par application Web contre un jeton d'accès OAuth.



Un cas pratique

- TP sur « Travaux Pratiques UML », fichier S1.pdf
- Faire l'exercice 5
- Mettre en avant les capacités des outils utilisés pour créer des diagrammes de séquences et les diverses constructions proposées par UML 2.5, notamment les séquences alternatives ou les options

Retour d'expériences sur le cas pratique

- Tous les outils utilisés, Rational ou Modelio, semblent ne pas permettre de capturer les alternatives ou les options
- Les diagrammes de séquences capturent principalement les flux de message; les activités ou les services n'apparaissent pas.
- Pour la cohérence, il faudrait être sûr que chaque acteur/composant interagissant fournit/consomme des services avec des opérations dont les entrées et sorties correspondent aux messages échangés.
- Pour les séquences initiés par un acteur, il faut faire correspondre les diagrammes de séquences aux use case définis: un diagramme d'interaction homme machine doit correspondre à un diagramme d'interaction où on illustre les interactions entre les composants internes découlant d'une action de l'utilisateur et allant jusqu'au retour vers l'utilisateur des résultats.
- **Illustration d'un outil assez complet pour dessiner des diagrammes de séquences mais ne permettant malheureusement pas de modéliser: plantUML (<http://www.plantUML.org>)**

Un cas pratique

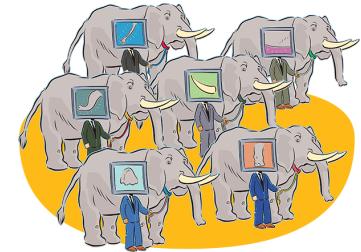
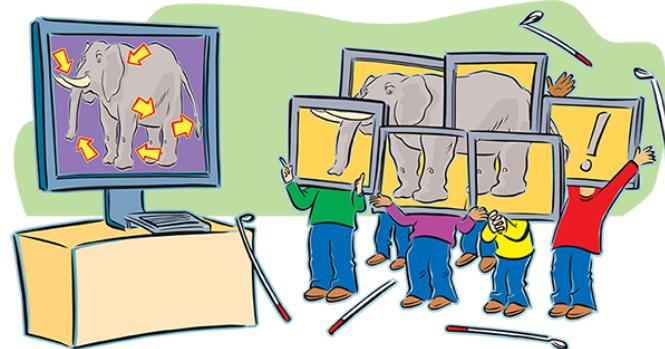
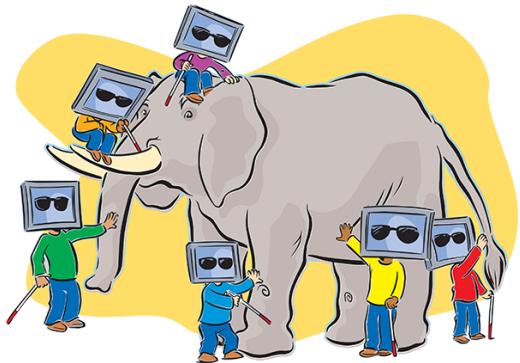
- ❑ Dans le cadre de l'informatisation des achats, l'architecte va distinguer les composants suivants:
 - o Répertoire des fournisseurs et des produits
 - o Recherche de fournisseurs
 - o Recherche de produit
 - o Composant Création des fournisseur
 - o Composant recherche de produit
 - o Composant enregistrement de produit
 - o Composant Achat de produit
 - o Composant Client Création de ligne comptable, qui invoque un service de l'application comptabilité pour créer une ligne pour l'achat réaliser
- ❑ Créer un diagramme de composant pour décrire une proposition d'architecture
- ❑ Créer des diagrammes de séquences à partir des use cases et montrant comment les composants internes interagissent pour répondre aux sollicitation de l'utilisateur
- ❑ Vérifier la cohérence globale
- ❑ Fournir sur un document office des copies des diagrammes créés, avec quelques lignes d'explication.

Les étapes suivantes

- La conception détaillée, avec diagrammes de classes et éventuellement certains diagrammes complémentaires à fournir (interaction, états, etc.) selon les besoins.
 - Les composants d'architecture sont détaillés pour pouvoir être spécifiés à des développeurs, testés et réceptionné.
 - Pas de réalisation de travaux pratique pour cette étape.
- L'intégration, les tests et la livraison
 - Avec la livraison du produit, les guides d'utilisation, d'exploitation, d'administration et du développeur sont à fournir. Ils reprennent les modèles mis à jour, soit sous forme d'images, soit comme un modèle.
 - Tous les artefacts doivent être à jour et consistants. La documentation technique, sous forme de document ou de modèle, doit être fournie en fournissant les vues utiles par ceux qui sont censés les consommer: utilisateur, administrateur métier, administrateur technique, architecte de système d'information, intégrateur d'application d'entreprise, urbaniste, développeur devant étendre ou faire évoluer le système, etc.
 - Les modèles à fournir sont dérivés des modèles précédents, en prenant en compte les modifications apportées lors du développement pour que les modèles décrivent ce qui va être livré, non pas ce qui a été initialement conçu. Les liens avec les besoins (use cases, etc.) doivent être capturé tout du long, pour vérifier que l'on répond bien à la demande du client.
- Support opérationnel et gouvernance
 - Des modèles de haut niveau (déploiement, composants, cas d'utilisation, lien avec processus métier) peuvent être réutilisés pour support er le support opérationnel et la gouvernance du système d'information.

En complément

Le modèles et les vues (diagrammes) sur le modèle au travers de l'allégorie sur l'éléphant et les 6 aveugles



- L'éléphant, c'est un peu le modèle et les diagrammes, les vues destinées à chaque type de parti prenante qui a un point de vue particulier. Sa vision de l'ensemble peut être restreinte juste à la partie qui l'intéresse, sans voir comment cela s'inscrit dans l'ensemble. Sans être aveugles, dans certains cas, ces partis prenantes fonctionneront avec des oeillères.
- Le fait de disposer du modèle sous jacents, c'est de pouvoir vérifier que les diverses vues sont cohérentes, et éventuellement de construire des vues de réconciliation entre celles des divers parti-prenante, pour permettre l'alignement et la cohérence globale