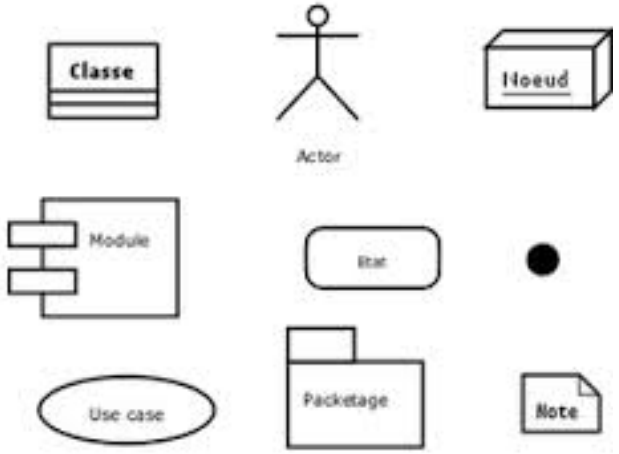
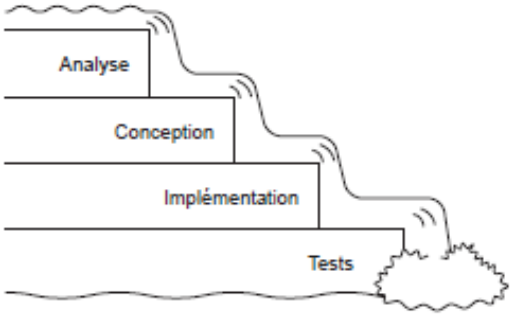
Programmation orientée Objet et C++ pour Eai

Unified Modeling Language (UML)

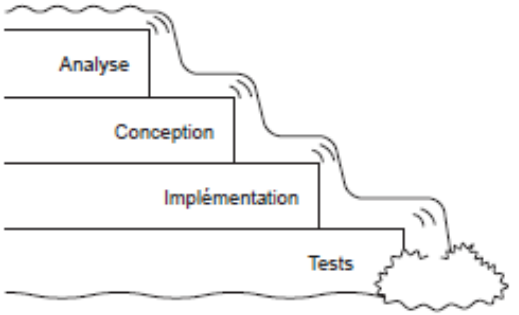


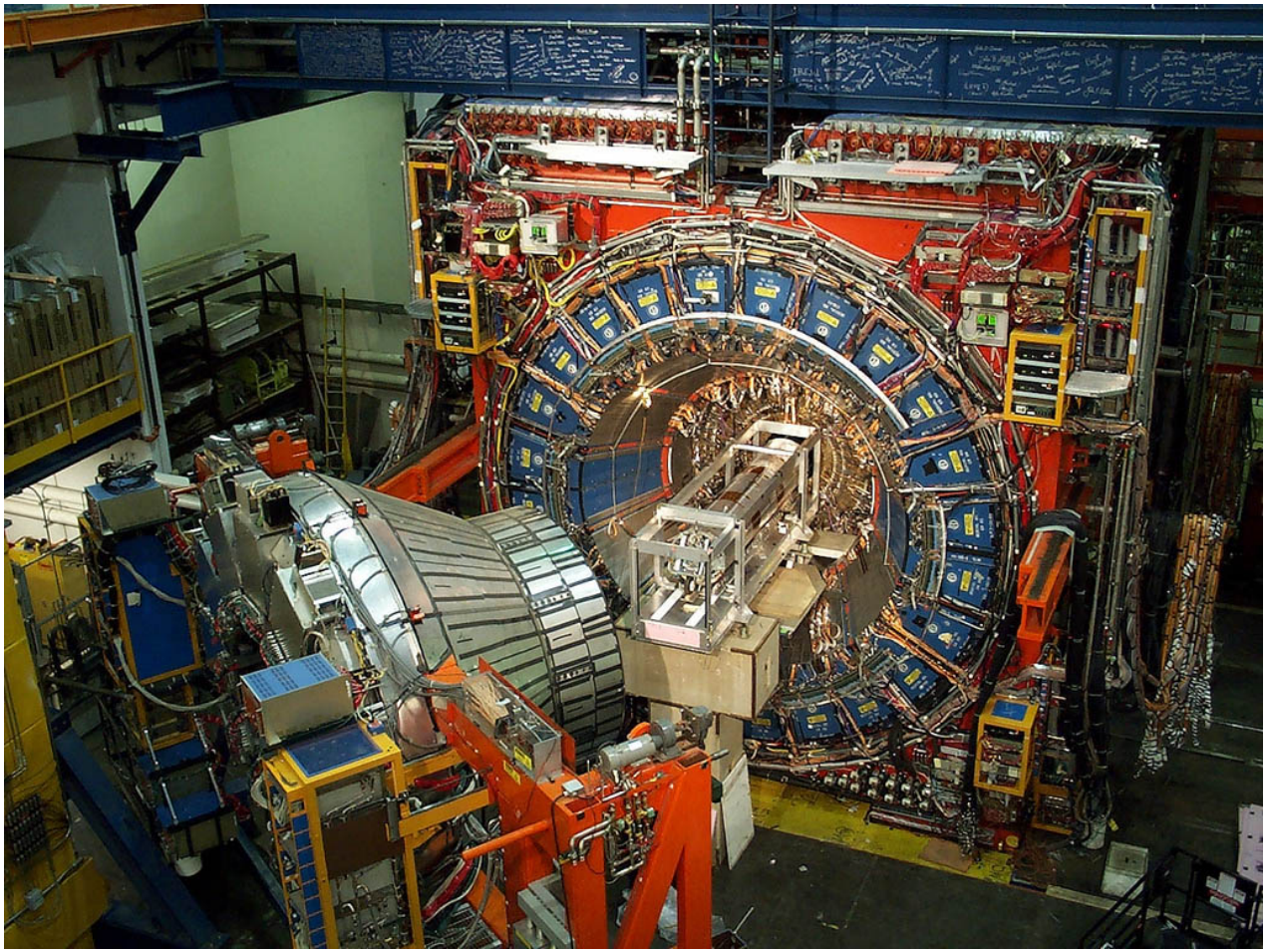
Analyse et conception orientée objet



- A force de se concentrer sur la syntaxe du langage C++, on risque de perdre de vue l'importance de ces techniques de construction des programmes
- Avant de foncer tête baissée dans le codage d'un projet, il faut:
  - Comprendre les besoins des clients
  - Créer l'architecture de son programme
- Et après le codage, il faut bien sûr
  - Tester son programme

La complexité logicielle



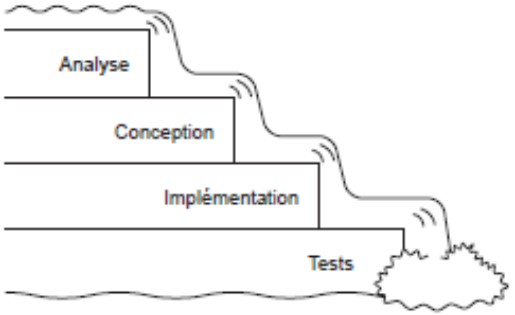


Exemple d'application très complexe

Le Large Hadron Collider (LHC, ou Grand collisionneur de hadrons en français) est un accélérateur de particules au CERN

heig-vd

La complexité logicielle



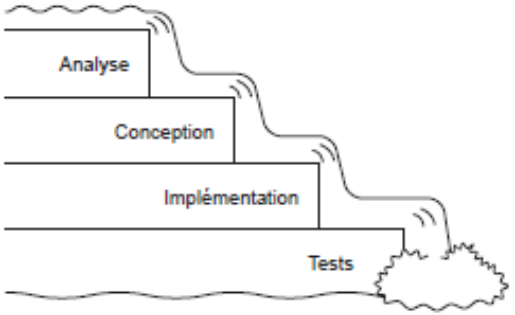
- Complexité des problèmes à traiter
  - Myriades d'exigences fonctionnelles
  - Exigences non fonctionnelles, souvent contradictoires
  - Difficultés de communication
- Complexité de la réalisation
  - Difficultés techniques de la programmation
  - Développement en parallèle à plusieurs
  - Besoin de créer une solution simple d'emploi
- Divergence des systèmes discrets
  - petite erreur => conséquences énormes

Organiser la connaissance

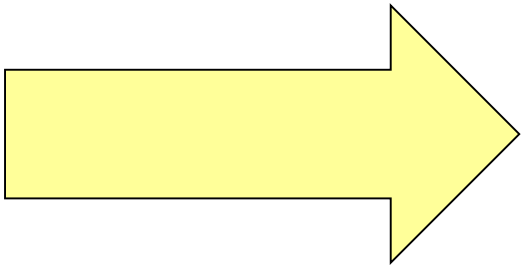
- Systèmes complexes
  - Trop d'information pour pouvoir les mobiliser simultanément
  - Nécessité de structurer pour rendre compréhensible

- Plusieurs axes d'organisation de la connaissance (principes de base de l'orienté objet)
  - Abstraction
  - Encapsulation
  - Modularité
  - Hiérarchie

Objectif du développement logiciel



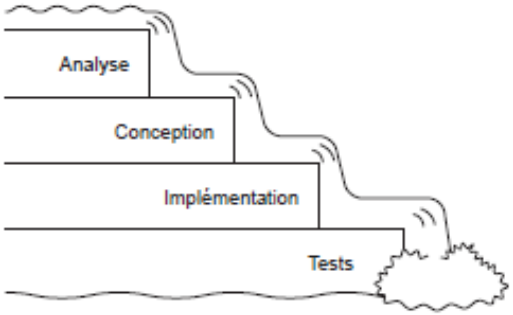
Exigences complexes



Solution déployée

heig-vd

Processus d'analyse et de conception



Exigences complexes

Solution déployée

Spécification claire formalisée

Solution validée

Analyse

Déploiement

Architecture logicielle

Solution testée

Conception générale, détaillée

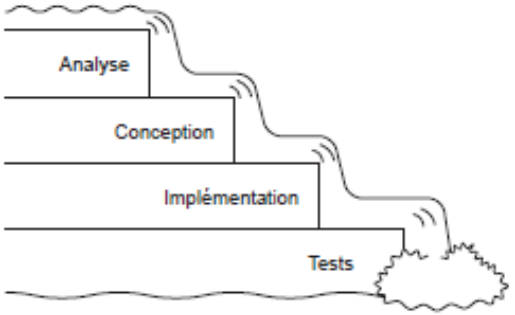
Acceptation

Code source C++, C#, java, …

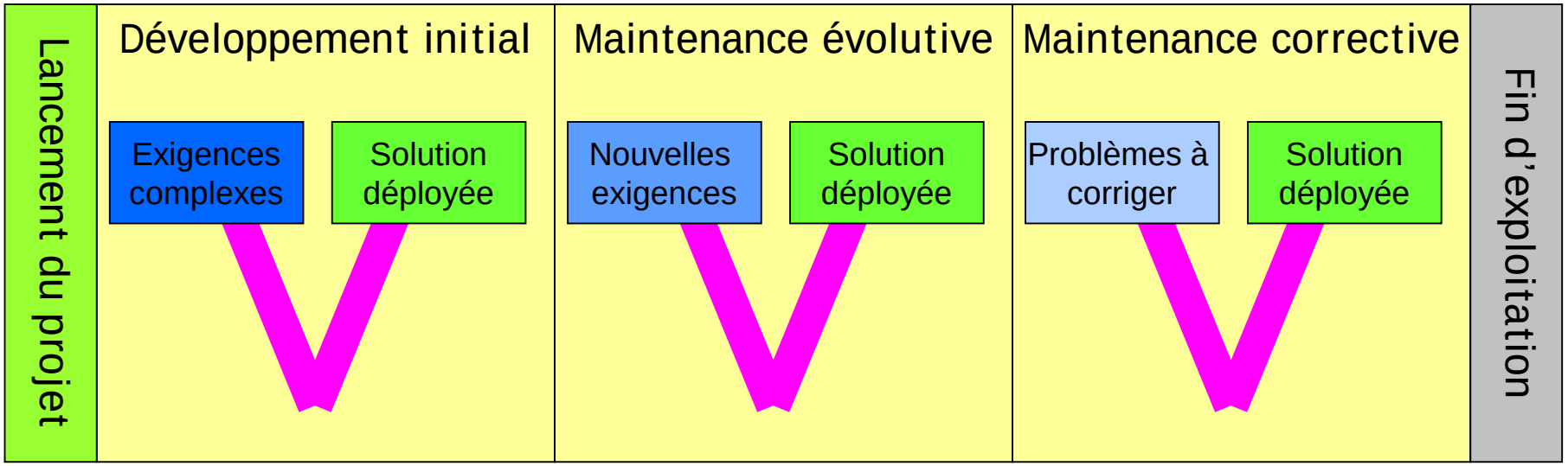
Programmation

Test unitaire, d'intégration

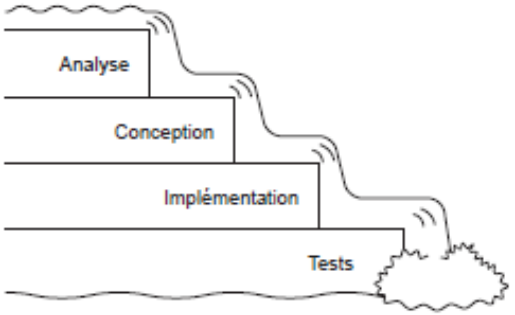
Le cycle de vie du logiciel

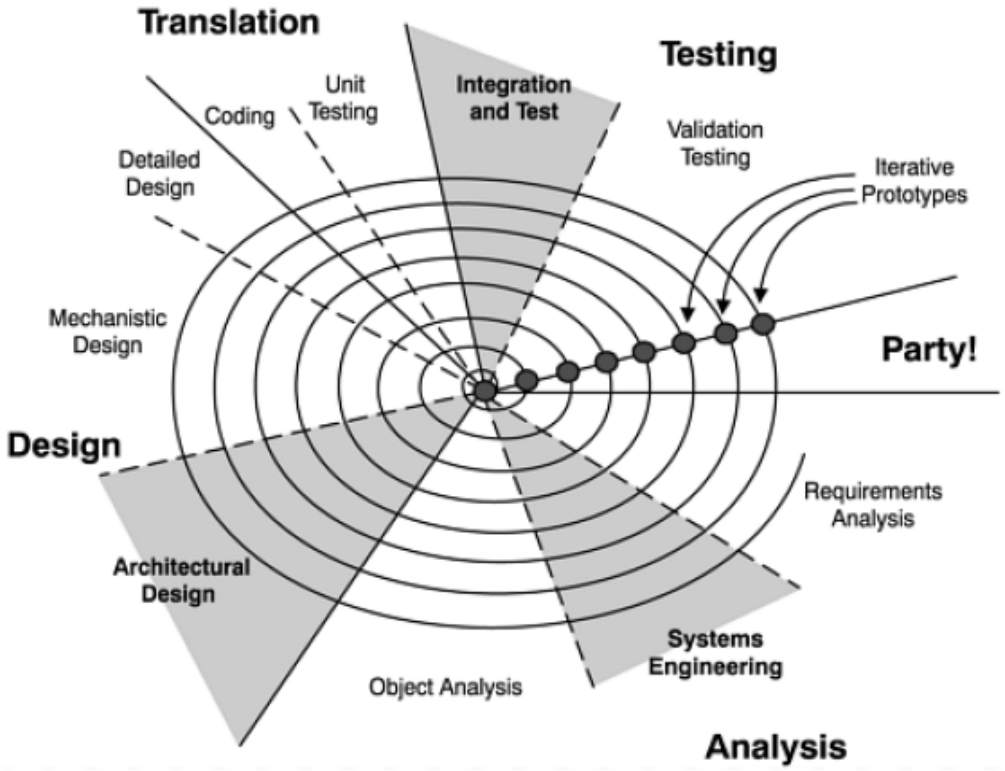


- Le cycle de vie du logiciel ne s'arrête pas à la fin du développement !



Le cycle de vie du logiciel

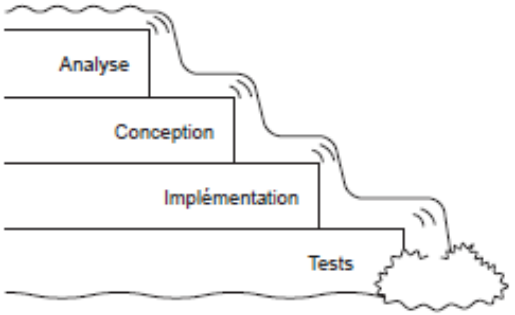




Les phases de développement peuvent être itératives (développement en spirale)

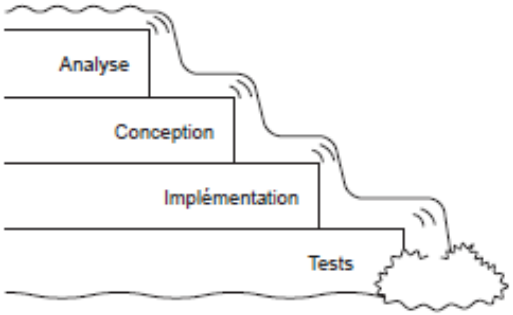
heig-vd

Critères d'évaluation de la qualité logicielle



- Jugement sur l'aspect visible par les utilisateurs
  - Le logiciel rend il tous les services attendus ?
  - Est-il simple et agréable d'utilisation ?
  - Est il suffisamment rapide ?
  - Peut il traiter les volumes de données rencontrés dans la réalité ?
  - Est il suffisamment fiable ?
  - Son coût est il maîtrisé ?
- Jugement sur l'aspect visible par les informaticiens
  - Les erreurs de programmation sont elles vite localisées et corrigées ?
  - La structure permet elle de faire évoluer le logiciel sans risque ?
  - La maintenance peut elle être confiée à d'autres personnes ?

Comment répondre aux objectifs de la qualité logicielle ?



- Bien analyser et comprendre les besoins
- Concevoir une structure adaptée et évolutive
- Ecrire un code maintenable
- Tester exhaustivement
- Suivre/tracer les demandes de correction

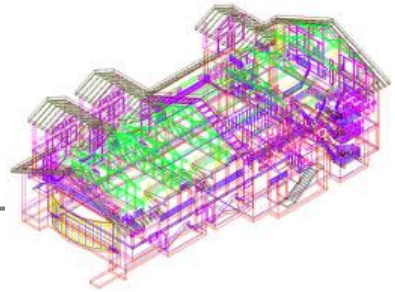
Langage d'analyse et de conception

- **UML**
- Qu'est ce qu'UML ?

- UML : Unified Modeling Language
  - Langage de **Modélisation** Unifié
  - Utilisé pour l'analyse et la conception des logiciels
  - Forte coloration orientée objet
  - Langage essentiellement graphique
  - Facile à lire et à comprendre

- En clair
  - UML: norme qui définit les diagrammes et les conventions à utiliser lors de la construction de modèles décrivant la structure et le comportement d'un logiciel
  - Les modèles sont des diagrammes constitués d'éléments graphiques et de texte
  - UML n'est pas une méthode, mais un langage

Pourquoi modéliser ?



- Modèle
  - Vue simplifiée de la réalité
  - Permet de comprendre synthétiquement le système à développer
- Le modèle permet de
  - Visualiser le système
  - Spécifier la structure et le comportement du système
    - comme il est ou comme il devrait être
    - ce qu'il fait ou ce qu'il devrait faire
  - Valider le modèle vis à vis des clients
  - Fournir un guide pour la construction du système
  - Documenter le système et les décisions prises

Les modèles

- L'objectif d'un modèle est de créer une abstraction significative du monde réel
- L'abstraction doit être plus simple que la réalité, mais aussi la refléter fidèlement afin que le modèle puisse servir à prévoir le comportement des *choses* dans la réalité
- L'abstraction permet d'extraire des concepts généraux sur lesquels raisonner, puis instancier les solutions sur les cas particuliers

UML – Quelques modèles

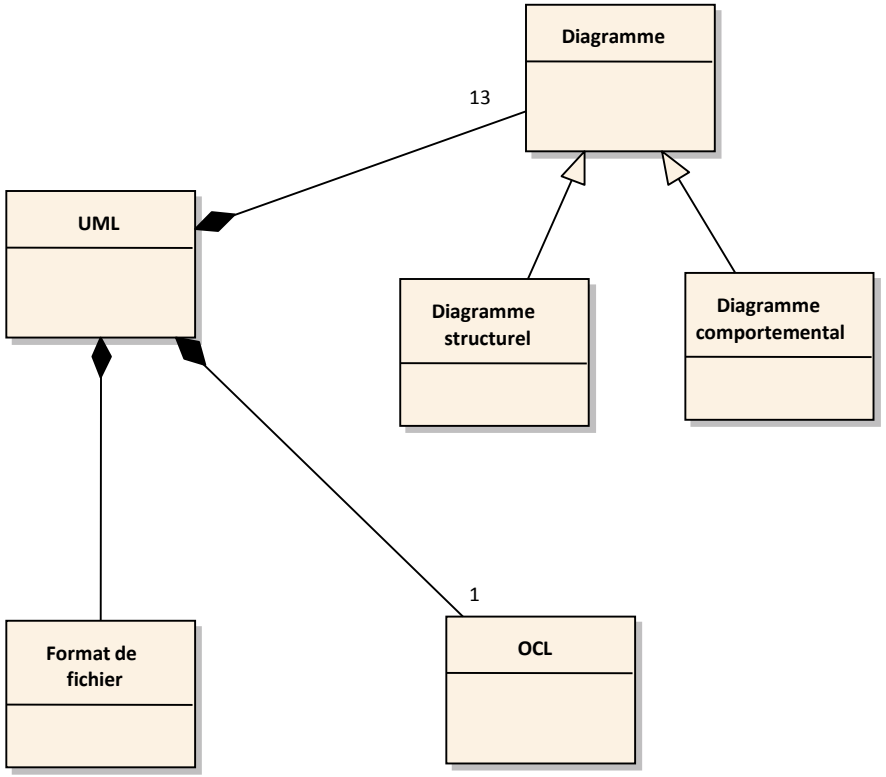
- Le modèle des cas d'utilisation qui décrit les besoins des utilisateurs
- Le modèle des classes qui capture la structure statique
- Le modèle des états qui exprime le comportement dynamique des objets
- Le modèle d'interaction qui représente les scénarios et les flots de messages

UML – Les différents modèles

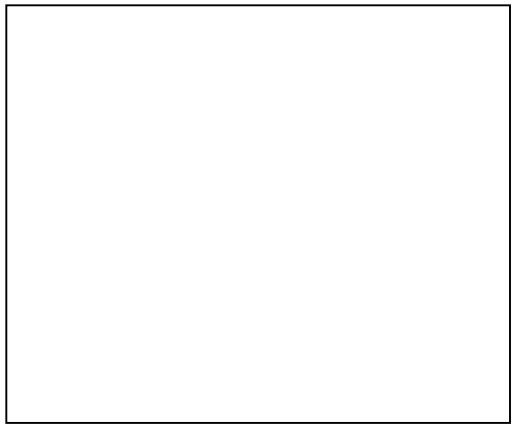
- Les modèles sont regardés et manipulés par les utilisateurs au moyen de diagrammes
- Les modèles contiennent un ou plusieurs diagrammes
- Un diagramme représente un certain aspect ou une partie d'un modèle

UML – Les diagrammes

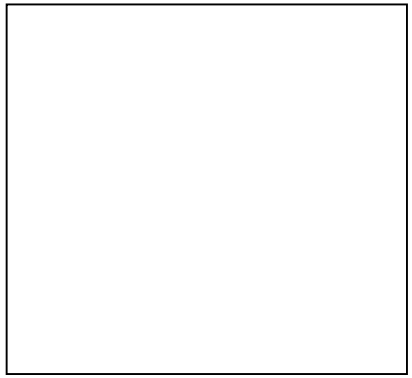
- 13 types de diagrammes
- Classés en 2 catégories
  - 7 diagrammes comportementaux
  - 6 diagrammes structurels
- Un langage de spécification de contraintes
  - OCL : Object Constraint Language
- Une spécification de format de fichier



UML - Diagrammes structurels



Class



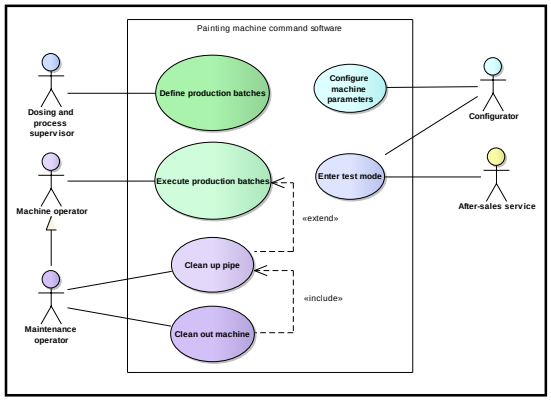
Component

Object

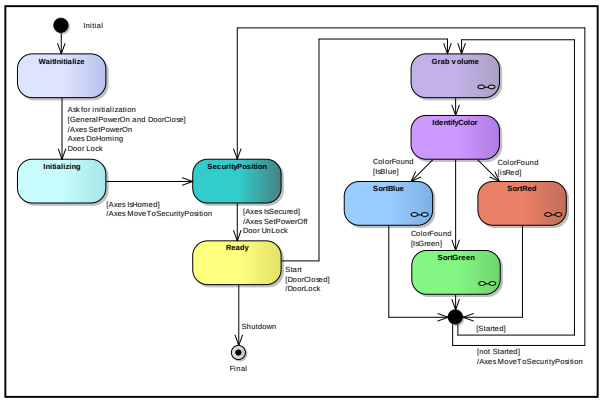
Package

Composite Structure

UML - Diagrammes comportementaux



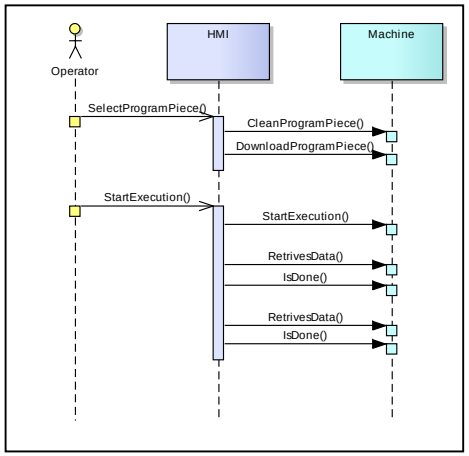
Use case



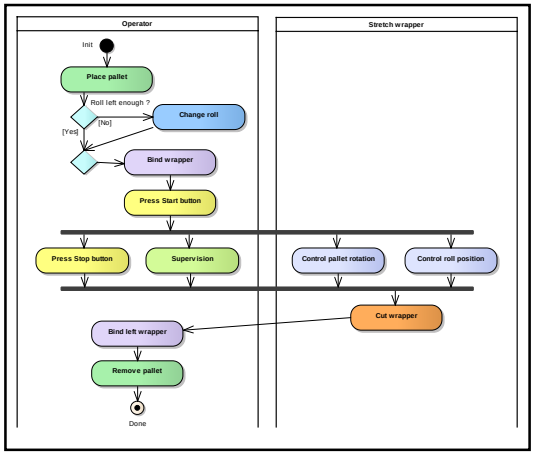
State machine

Timing

Interaction Overview



Sequence



Activity

Communication

UML – Un langage pour:

- Visualiser
  - chaque symbole graphique a une sémantique
- Spécifier
  - de manière précise et complète, sans ambiguïté
- Construire
  - les classes, les relations SQL
- Documenter
  - les différents diagrammes, notes, contraintes, exigences seront présentés dans un document

UML – Les diagrammes les plus utilisés

- Analyse de la fonctionnalité
  - Cas d'utilisation
  - Diagrammes d'activité
- Conception de la structure
  - Diagramme de classes
- Conception du comportement
  - Diagramme de séquence

UML – Comment analyser la fonctionnalité ?

- Avec les cas d'utilisation
- Définition
  - Un cas d'utilisation décrit une séquence d'interactions entre le système et son environnement extérieur
  - Aboutit à un résultat concret pour l'utilisateur
  - A cet effet, on définit des acteurs, censés être initiateurs d'actions, et le cas d'utilisation lui-même
  - L'utilisateur d'un système n'est pas forcément un humain
    - Ex : système de pilotage des moteurs d'un robot : l'utilisateur est l'automate ou le PC qui envoie des ordres
- Utilisation
  - Analyse de la fonctionnalité
  - Identifier les façons d'utiliser le système

UML – Cas d'utilisation - Symboles

UML – Cas d'utilisation

- Système de gestion de la bibliothèque

heig-vd

UML – Cas d'utilisation

- Application d'assemblage avec un robot

heig-vd

UML – Cas d'utilisation

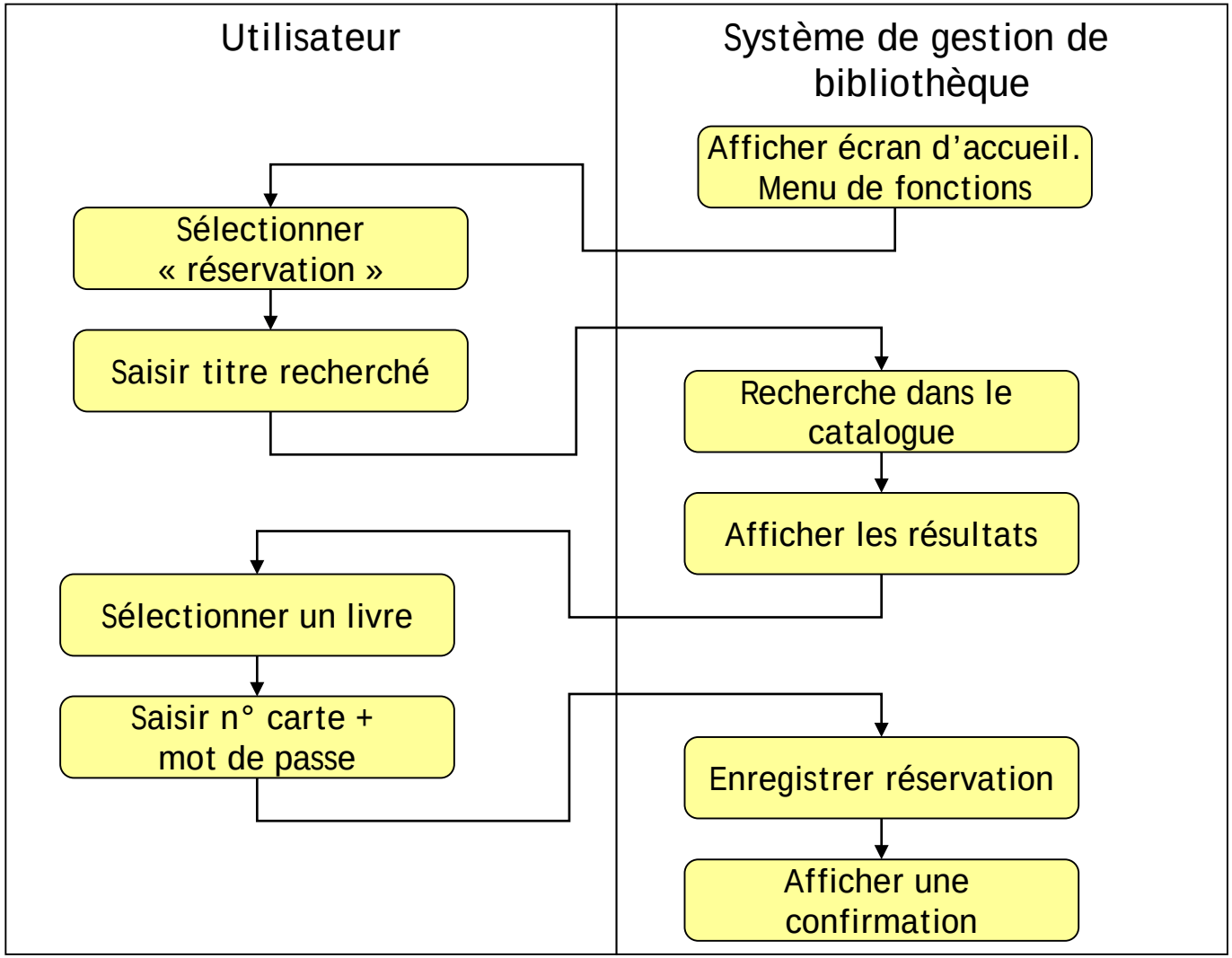
- Mélange de produits

UML – Comment analyser la fonctionnalité ?

- Les diagrammes d'activité
- Définition
  - Description d'un flot d'activité
  - Permet de montrer le déroulement d'un cas d'utilisation
  - En colonne, on place les acteurs du cas d'utilisation
  - Dans les colonnes, on décrit l'enchaînement des activités
- Bien adaptés
  - Pour décrire des enchaînements d'opérations
  - Pour clarifier les interactions avec l'utilisateur
  - Pour identifier la logique du système à programmer
  - Analyse de la fonctionnalité

UML – Diagrammes d'activité - Symbole

UML – Diagrammes d'activité – Exemple1



heig-vd

UML – Diagrammes d'activité – Exemple2

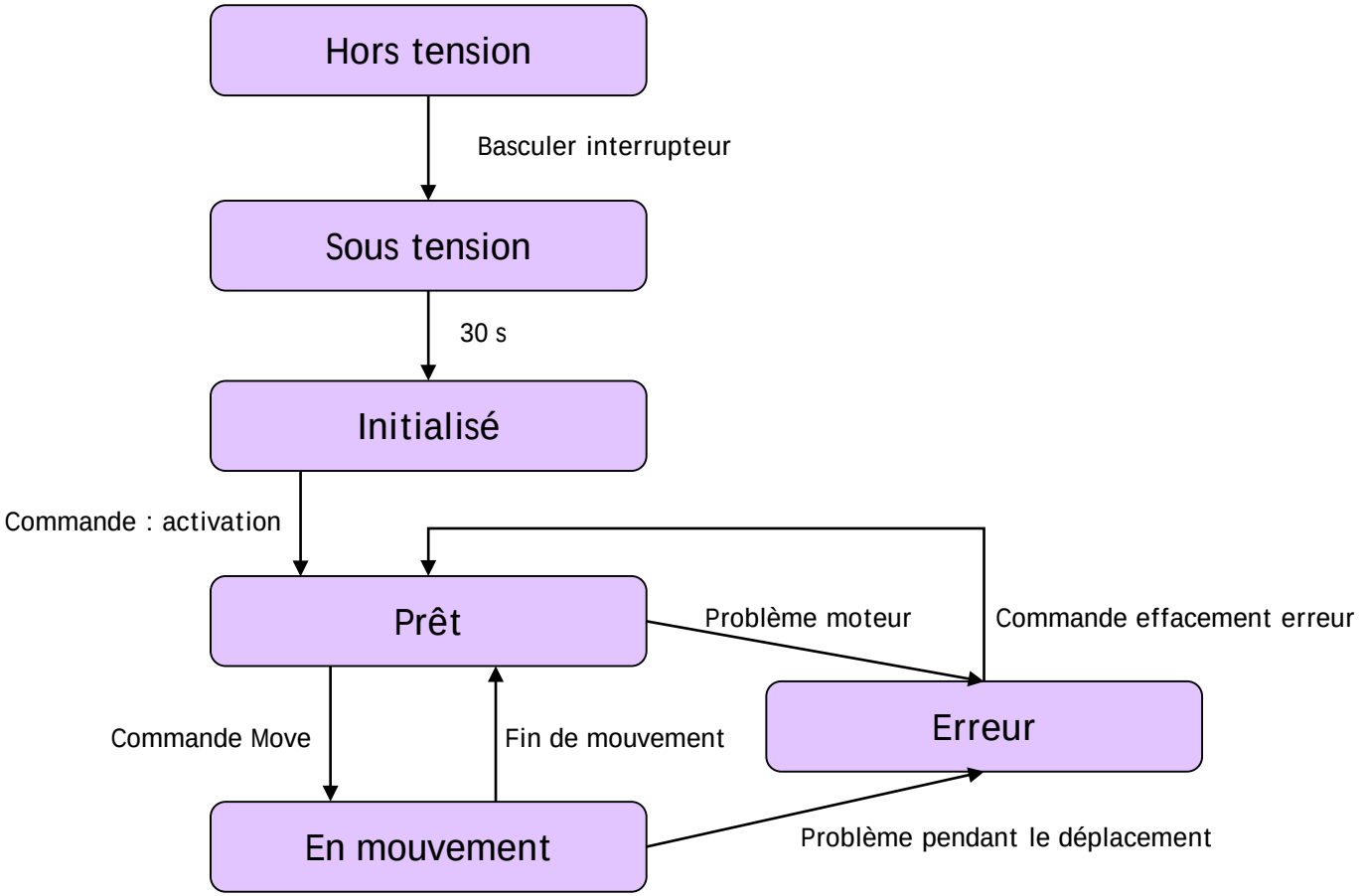
UML – Diagrammes d'activité – Exemple3

UML – Comment analyser la fonctionnalité ?

- Les diagrammes d'état
- Définition
  - Un diagramme d'état est une représentation graphique
  - Utilisée pour montrer les états que peut prendre un système
  - Les états sont représentés par des rectangles arrondis
  - Les transitions possibles sont représentées par des flèches
- Bien adaptés
  - Pour décrire l'évolution de l'état d'un système
  - Pour identifier la logique du système à programmer
  - Pour mettre en évidence des contraintes temporelles
  - Pour identifier les variables d'état

UML – Diagrammes d'état

Etat d'une commande numérique



UML – Diagrammes de classes - Principes

- Définition
  - Vue structurelle du logiciel
  - Représentation de classes et de leurs relations
  - Proche de l'implémentation

- Utilisation
  - Très utilisé pour la conception de la structure du logiciel
  - Préférer des vues simples explicitant quelques aspects du logiciel
  - Outil de travail itératif

UML – Diagrammes de classes - Symboles

Le nom des classes avec méthodes virtuelles pures sont en *italiques (interface)*
Le nom des méthodes virtuelles sont en *italiques*

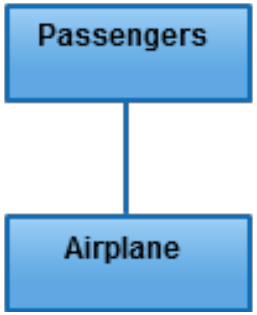
UML – Diagrammes de classes – Exemple1

- Structure du contrôle d'une machine

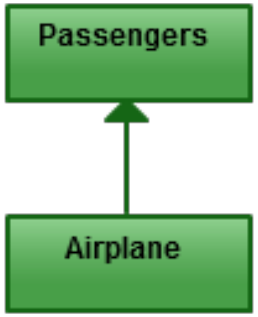
UML – Diagrammes de classes – Exemple2

- Définition des séquences

Types de relations entre classes

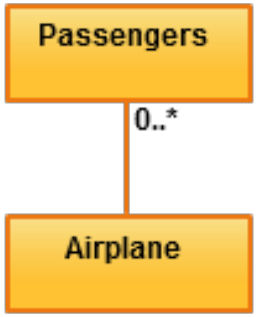


Utilisation ou Association

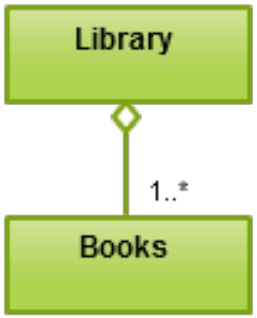


Association dirigée

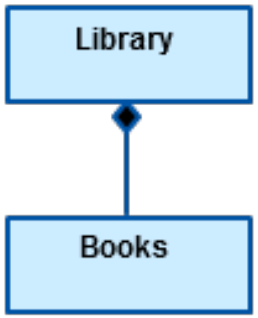
Association réflexive



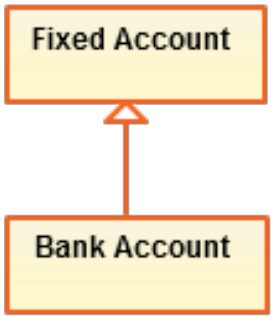
Multiplicité



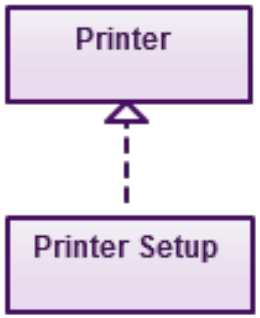
Agrégation



Composition



Héritage



Réalisation

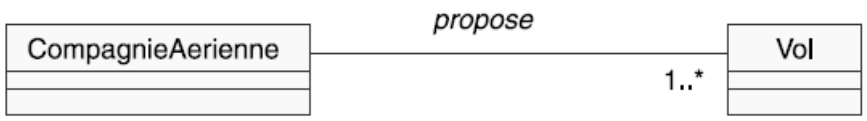
Relations entre classes
Lien simple : relation d'utilisation ou d'association

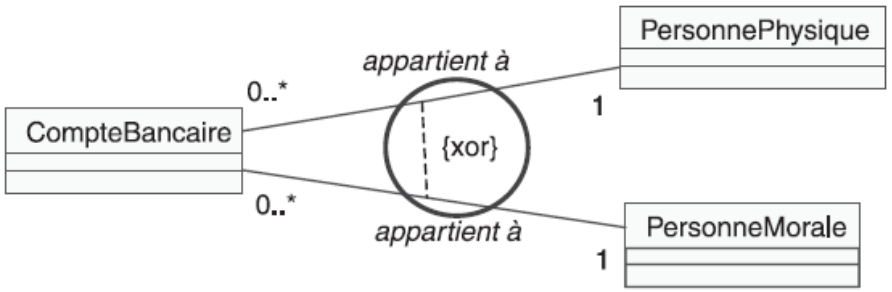
- Une association est une relation entre deux classes
- Les **associations peuvent être affinés** pour modéliser un type plus restrictif de relation appelée **agrégation**
- **L'agrégation peut** également **être affinée** pour modéliser une relation encore plus restrictive appelé **composition**

Relations entre classes

- Chaque association peut être définie avec un facteur de multiplicité.
- Une multiplicité est exprimée soit par un couple de valeur N..M soit par une seule valeur lorsque N est M sont égaux. Par défaut, la multiplicité est 1.

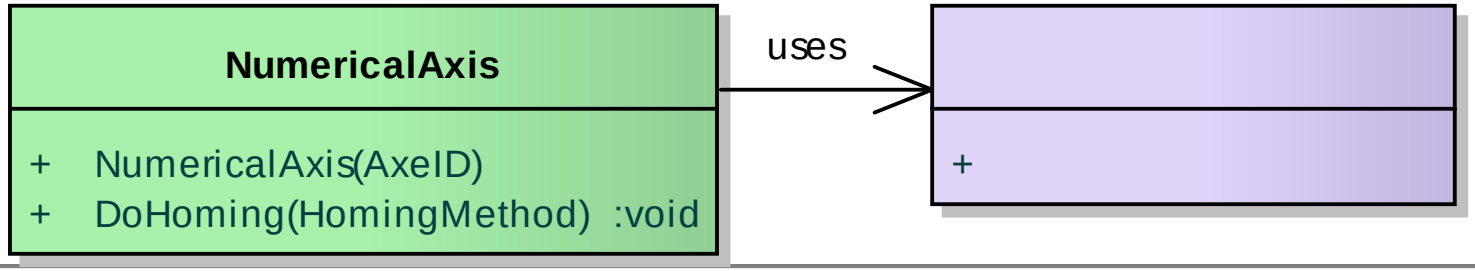
Exemple	Interprétation
1..1 ou 1	Un et un seul
0..1	zéro ou un seul
0..* ou *	zéro à plusieurs
3..4	trois à quatre
4	quatre et seulement quatre





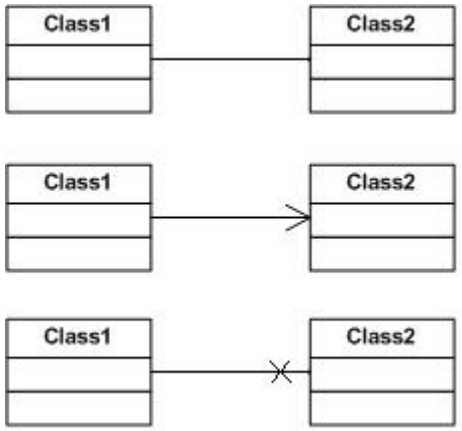
Relations entre classes
Lien simple : relation d'utilisation ou d'association

- Définition
  - Un objet utilise les services d'un autre
  - Concrètement, **il appelle ses méthodes**
- Test de validité
  - Un objet de ClasseA ***utilise*** un objet de ClasseB
- Exemple
  - L'objet « Contrôleur d'axe numérique » utilise l'objet « Homing» pour aller en position de référence



Relations entre classes – Utilisation

– **Navigabilité** : indique si on pourra accéder d'une classe à l'autre. Si la relation est entre les classes A et B et que seulement B est navigable, alors on pourra accéder à B à partir de A mais pas réciproquement. Par défaut, la navigabilité est dans les 2 sens.



Navigabilité des associations

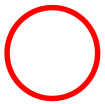
1- Bidirectionnelle

2- Mono-directionnelle, Invocation de méthode

3- Interdit une association

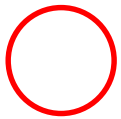
Relations entre classes - composition

- Définition
  - Un objet est composé de sous-objets
- Test de validité
  - Le sous-objet a exactement la même durée de vie que le contenant
  - Un objet de ClasseA *est composé (has a)* d'un objet de ClasseB
- Exemples
  - Une personne est composé d'une tête, d'un corps, de bras, etc.
  - Le « système de commande de la machine » *est composé* de 4 « contrôleurs d'axe numérique », de 20 « entrées TOR » et de 10 « sorties TOR ».



Relations entre classes - agrégation

- Définition
  - Un objet peut contenir des sous-objets.
  - Concrètement, un objet contient des références vers des sous-objets.
  - Une agrégation est juste un cas spécial de la composition
- Test de validité
  - Le sous-objet peut avoir une durée de vie différente du contenant.
    (Cela se traduit par le fait que le sous-objet n'est pas créé par l'objet.)
  - Un objet de ClasseA *peut contenir* un objet de ClasseB.
- Exemple
  - Un « workflow » contient des « actions ».



Relations entre classes - exemple

- Les camions continuent d'exister si l'entreprise fait faillite => agrégation.
- Le moteur part à la ferraille avec le camion (en général) => composition.

Relations entre classes – Agrégation
Exemple (1/2

heig-vd

Relations entre classes – Agrégation
Exemple (2/2

Relations entre classes – Héritage
(généralisation / spécialisation)

- Définition
  - Une sous-classe est une spécialisation de sa classe parent
  - Une classe est une généralisation de ses sous classes
- Test de validité
  - Un objet de la sous classe ***est un*** objet de la classe
- Exemples
  - Une « opération de fraisage » est une « opération d'usinage »
  - Une « sortie TOR Beckhoff » est une « sortie TOR ».

Relations entre classes – Réalisation

- Définition
  - Relation entre deux éléments de modèle, dans laquelle un élément de modèle (le client) réalise le comportement que l'autre élément de modèle (le fournisseur) spécifie (flèche composée d'un trait discontinu et d'une pointe creuse qui part du client)
  - Relation dans laquelle une interface définit un contrat garanti par une classe d'implémentation

UML – Diagrammes de séquences - Principes

- Définition
  - Vue comportementale du logiciel
  - Représentation des appels de méthodes entre objets
  - Proche de l'implémentation

- Utilisation
  - Très utilisé pour décrire un comportement du logiciel
  - Préférer des vues simples explicitant une séquence particulière
  - Ne l'utiliser que lorsqu'il y a un vrai travail d'analyse de séquence
    - On ne documente pas tous les enchaînements d'appels !

UML – Diagrammes de séquences - Symboles

heig-vd

UML – Diagrammes de séquences – Exemple1

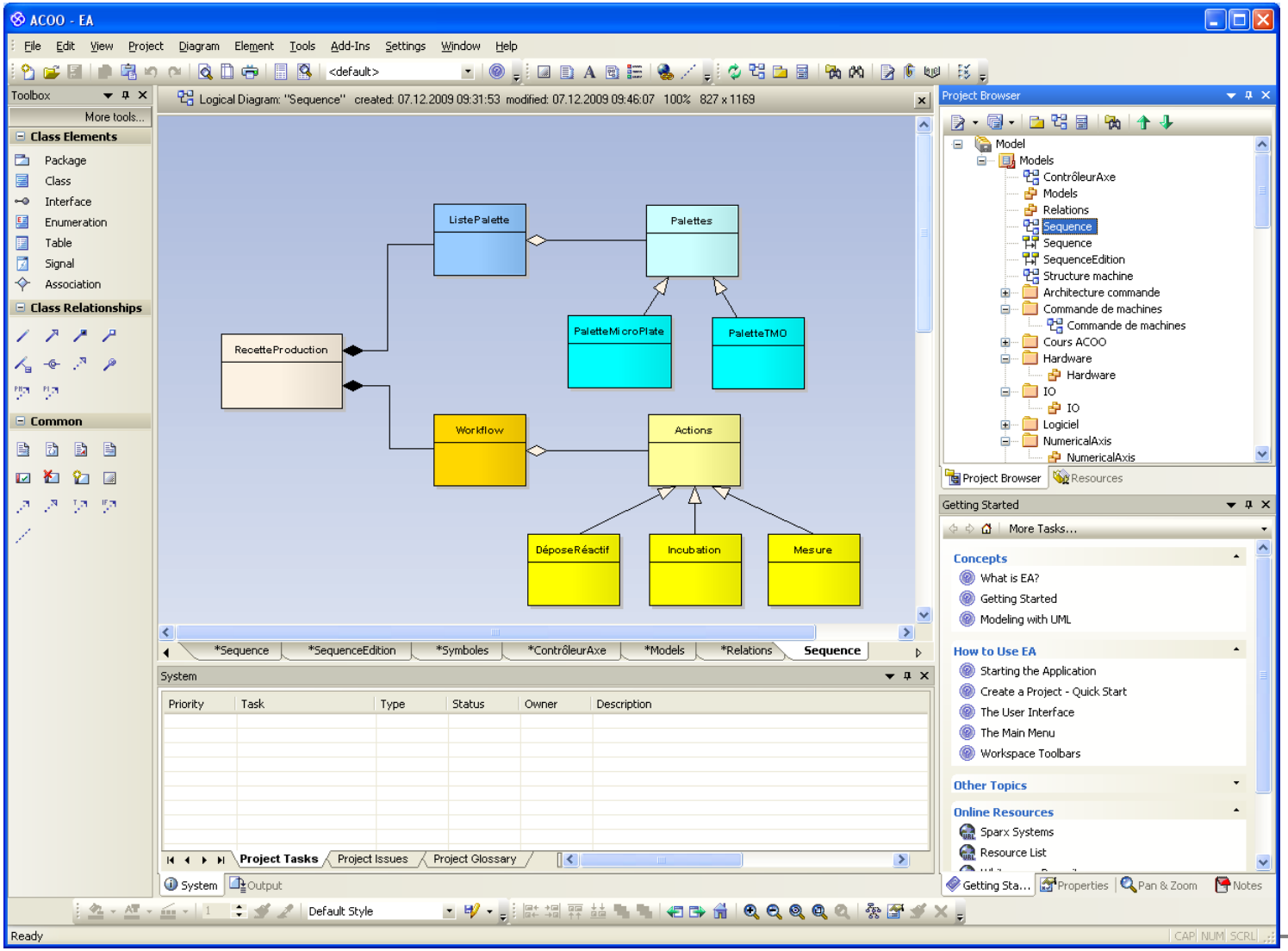
- Edition

UML – Diagrammes de séquences – Exemple2

- Edition

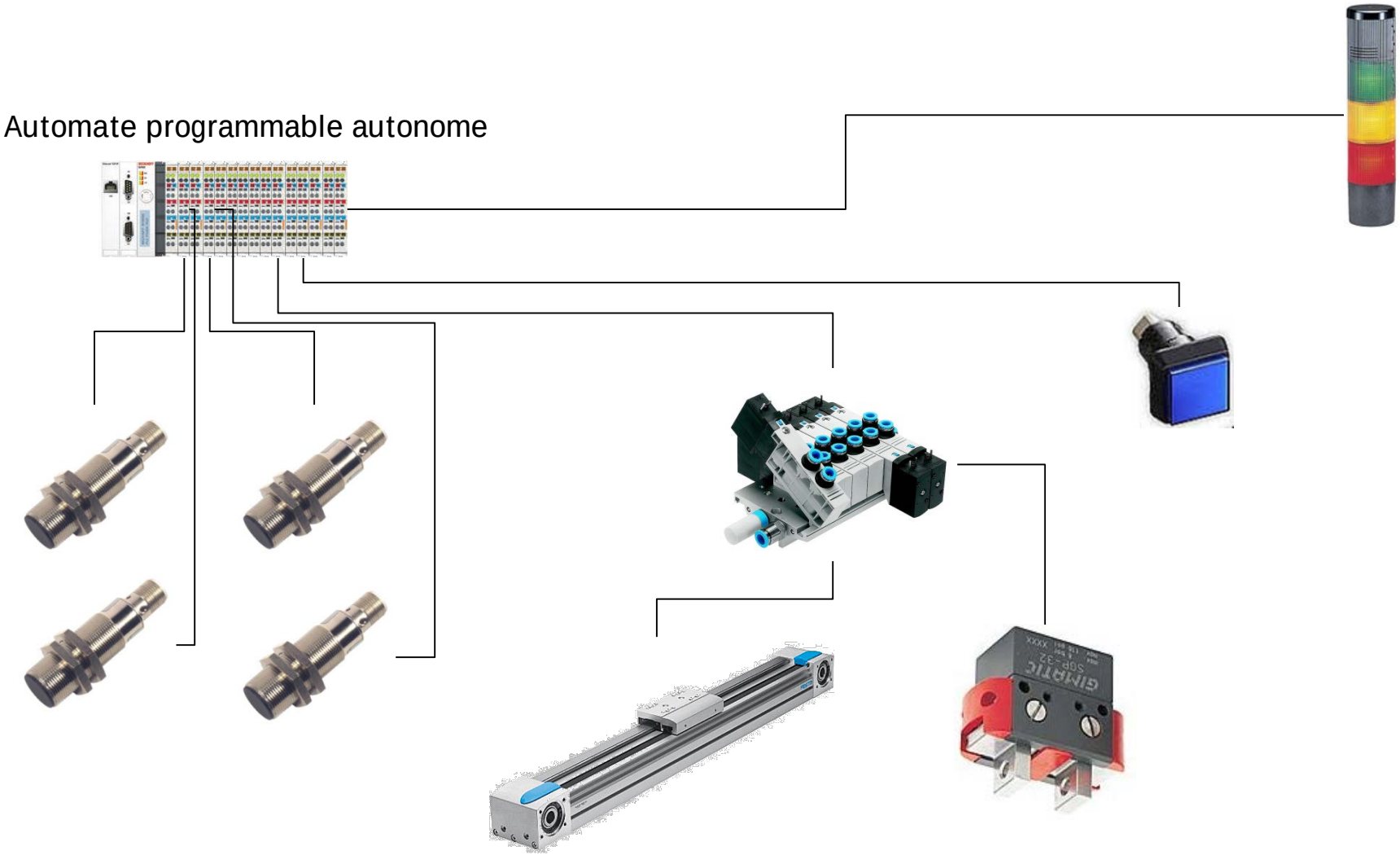
heig-vd

UML – Outils "EnterpriseArchitect"



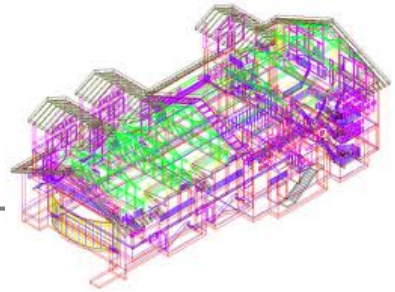
Approche orientée objet

Modélisation naturelle – Illustration sur un cas concret



Approche orientée objet

Démarche intellectuelle – travailler sur la langue naturelle



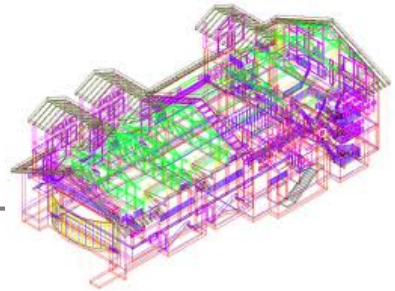
- Reconnaitre classes-objets et méthodes
  - **Classes et objets : noms communs**
  - **Méthodes : verbes**

- **Avec l'interface homme-machine, l'opérateur édite une séquence en ajoutant des actions, pour lesquelles il doit saisir les paramètres d'usinage.**

- **L'exécution d'une séquence d'actions consiste à verrouiller les portes de sécurité, à mettre sous tension les axes numériques, puis à exécuter les actions l'une après l'autre.**

Approche orientée objet

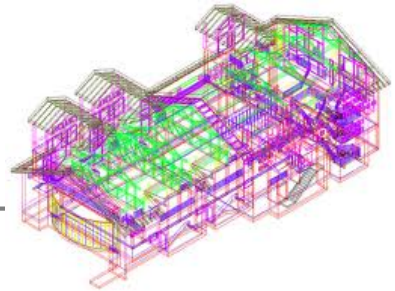
Démarche intellectuelle – travailler sur la langue naturelle



- Reconnaitre les relations entre classes
  - Les compositions
    - est composé de….
    - La machine est composée de 4 axes numériques et d'une pince.
  - Les agrégations
    - peut contenir des…
    - Une séquence peut contenir de nombreuses opérations.
  - Les généralisations / spécialisations
    - Est un type de…
    - Le *fraisage* est un type d'opération réalisable sur ces machines.
    - *L'opérateur et technicien de maintenance* sont des types d'utilisateurs

Approche orientée objet

Quelle relation ?



- Il est parfois difficile de choisir entre association classique et agrégation. Pour cela, il faut s'interroger sur le cycle de vie des objets concernés. Voici quelques questions qui vous permettront peut-être de trouver une réponse :

- Les objets concernés sont-ils indépendants les uns des autres ? C'est à dire, la mort de l'un entraine-t-il la mort d'autres ? Si NON alors ASSOCIATION.
- Y a-t-il des méthodes d'une classe qui s'appliqueraient aux autres ? Si oui, cette classe est le composé et sera reliée aux autres par une AGREGATION
- Demandez-vous si cela paraît naturel de dire mon objet TITI est une partie de mon autre objet TOTO ? Si oui, TOTO est composé de TITI (composant), et nous avons là une COMPOSITION.

Comment procéder ?

L'analyse et la conception

- **Phase d'analyse** : **clarifier les objectifs**
  - Clarifier exhaustivement les exigences à remplir.
  - Formaliser le comportement attendu du logiciel.
  - Le faire valider par le client.
- **Phase de conception** : **clarifier la structure**
  - Définir la structure du logiciel permettant de couvrir les exigences.
  - La valider par rapport aux exigences à remplir.
- **Démarche générale**
  - Abstraction : créer des vues simples à comprendre.
  - Décomposition hiérarchique vers plus de détail.
  - Utilisation d'un langage facile à comprendre par le client.

Principe de base

L'analyse et la conception

- Une bonne conception doit être fermée autant que possible aux modifications sur le code existant!
  (pas de modification des prototypes, mais correction et ajouts permis)

- Une bonne conception est la clef de voute pour du code réutilisable, extensible et robuste.

- L'objectif de ce cours n'est pas de vous apprendre à utiliser "bêtement" le langage C++, mais d'apprendre à concevoir d'une meilleure manière.

heig-vd

Vos questions


Exercices
UML