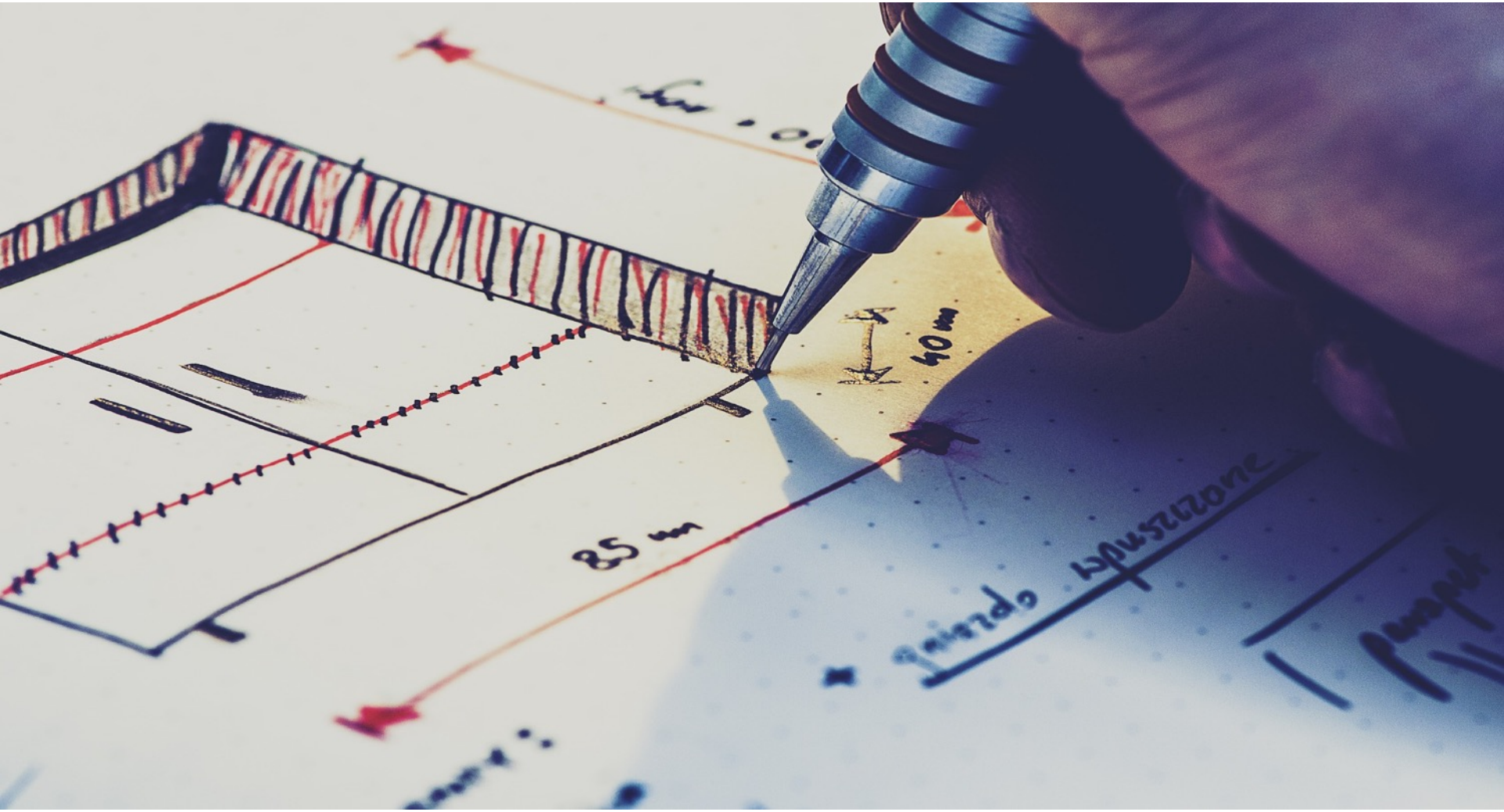
Conception logicielle

Université Nice Sophia Antipolis | POLYTECH NICE-SOPHIA

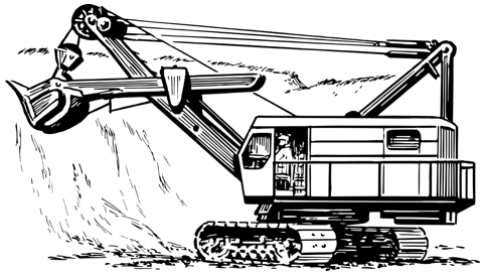
Membre de UNIVERSITÉ CÔTE D'AZUR



Objectifs

- Savoir analyser et concevoir « objet »
  - Connaître l'essentiel de la notation UML
- Apprendre les User Stories
  - et les tests comportementaux associés
- Connaître les patrons de conception les plus courants
  - savoir quand les appliquer et ne pas les appliquer

Mise en œuvre



- En équipe projet dès le début (dans 2h…)

- Première série de TD sur un cas que l'on complexifie progressivement
  - Modélisation en **UML** uniquement

- Deuxième série de TD sur un cas plus complexe, et que l'on complexifiera encore plus
  - Première approche en **UML** réduit (pour défricher le terrain)
  - Développement progressif en Java avec tests unitaires, **User Stories** et **Behavior-Driven Development avec tests comportementaux**
  - Application de **patrons de conception** lors du développement

Contrôle des connaissances

- Note (individualisée) du premier sujet en équipes : 20 %
- Note (individualisée) du second sujet en équipes : 40 %
- Examen écrit final : 40 %
- Absences injustifiées en TD ? Baisse des notes individualisées de TD long et/ou projet

Communication

- Diffusion des supports, sujets, etc. sur Moodle :
  - https://lms.univ-cotedazur.fr/course/view.php?id=4314
- Canal de communication par défaut : **SLACK**
  - **#si4-conception**
- Question sur un des sujets de TD/projet : sur la chaine, pas de mp
  - Tout le monde profite de la réponse
  - Vous vous répondez entre vous

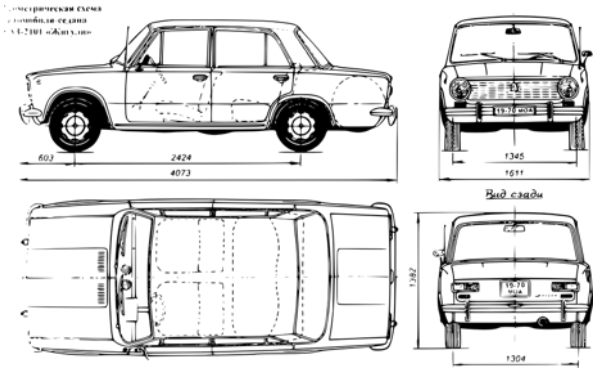
Formation des équipes

- 5 personnes par équipe (4 sinon, exceptionnellement)
  - 2 personnes au moins de 2 provenances différentes en entrée de 3A
  - Possibilité d'être dans des TDs différents (sauf si contraintes COVID sur la présence sur le campus)

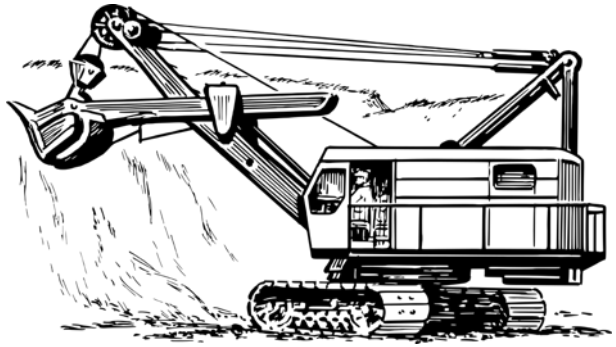
- Liste des étudiants de l'équipe à publier sur le slack (chaîne #si4-conception) avant ce soir 22h (lundi 1er septembre !)
  - Nom prénom pour chacun
  - 1 par ligne

Pourquoi modéliser ?

Spécifier la structure et le comportement d'un système



Aider à la construction d'un système



Visualiser un système existant

Documenter les décisions

UML un peu d'histoire

- Plein de notations et de méthodes différentes à la fin des années 90 : OMT, Booch, etc.
- Besoin d'un langage « standard »
- Plusieurs auteurs de méthodes orientées objets s'allient et créent l'entreprise Rational
  - Grady Booch, Ivar Jacobson, James Rumbaugh

UML, avec un U comme Unified

UML : un mélange de plusieurs notations

- Développé par Rational puis standardisé par l'OMG (Object Management Group à partir de 1997
- Tous les grandes compagnies entrent dans l'OMG : Oracle, HP, Microsoft, IBM…

Dernière version : UML 2.5.1, Dec. 2017 – 800 pages de spécification…

UML : standard industriel !

- En 2005 déjà, 97% des développeurs connaissaient UML et 56% l'utilisaient dans leurs projets
  - http://www.prweb.com/releases/2005/04/prweb231386.htm

- Mais c'est quoi ? Une notation visuelle…
  - Descriptions graphiques et textuelles
  - Syntaxe et sémantique (presque pas ambiguë)
  - Architecture et comportement
  - Génération ou rétro-ingénierie de code

UML : des points forts

- Un langage normalisé
  - Précision
  - Stabilité
  - Facilite l'outillage
- Un support de communication
  - Cadrage de l'analyse
  - Compréhension d'abstractions complexes
  - Universalité

UML : des points faibles

- Période d'adaptation nécessaire
- Lourdeur de la spécification
- UML n'est qu'une notation, l'utiliser ne donne pas de « méthode » pour bien concevoir

Organisation d'UML

- 13 diagrammes réalisés à partir des besoins utilisateurs
- Des aspects fonctionnels
- Des aspects liés à l'architecture

Intérêts d'UML

- Beaucoup moins dans l'utilisation complète de la notation

- Beaucoup plus dans l'utilisation de certains éléments en fonction du contexte :
  - Rétro-ingénierie de code
  - Documentation
  - Zoom sur un problème, décision avant refactoring
  - Création et évolution d'architectures logicielles
  - Génération de code dans des parties système

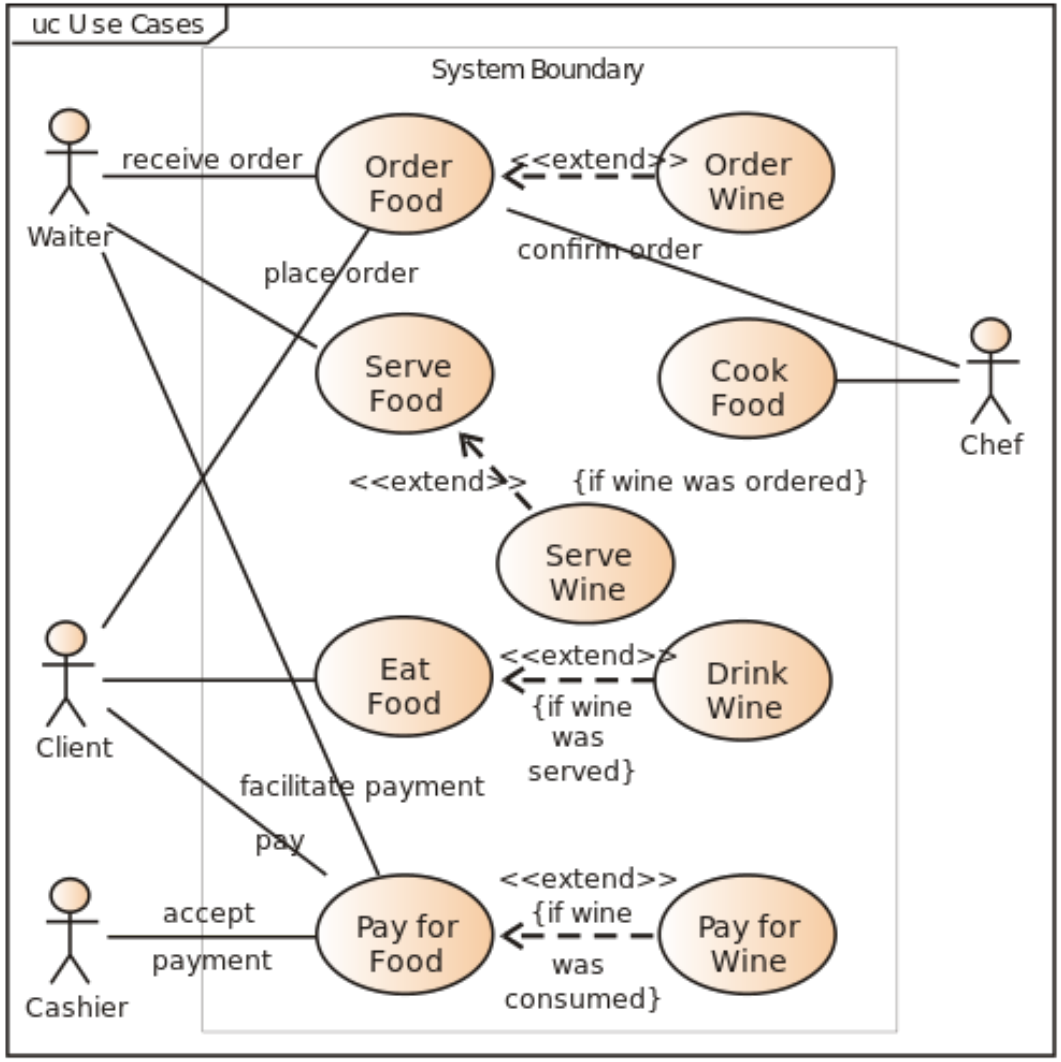
Les vues d'un système

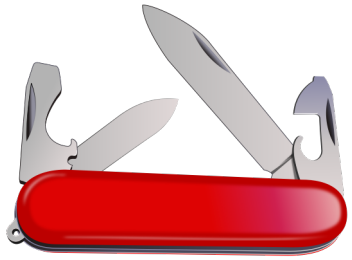
Fonctionnel = interactions utilisateurs/systèmes
- Cas d'utilisation
- Activités
- Séquence

Statique = structurelle, identifications des objets et Composants, et de leurs relations
- Classes
- Objets
- Packages
- Composants
- Déploiement

Dynamique = évolution des objets dans le temps
- Etats
- Activités
- Séquence
- Communication

Diagramme de cas d'utilisation





- Périmètre d'un système (pas forcément OO)
- Discussion utilisateur

Diagramme d'activités

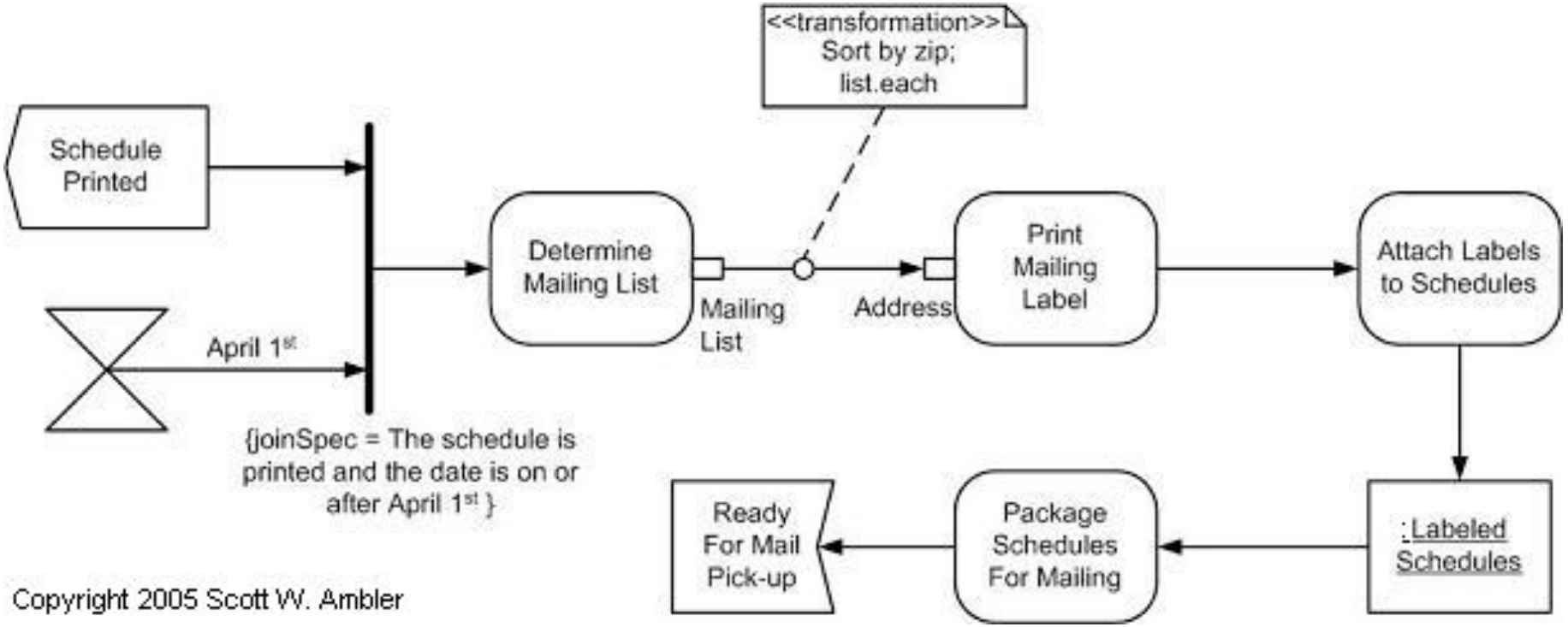


Diagramme de séquences

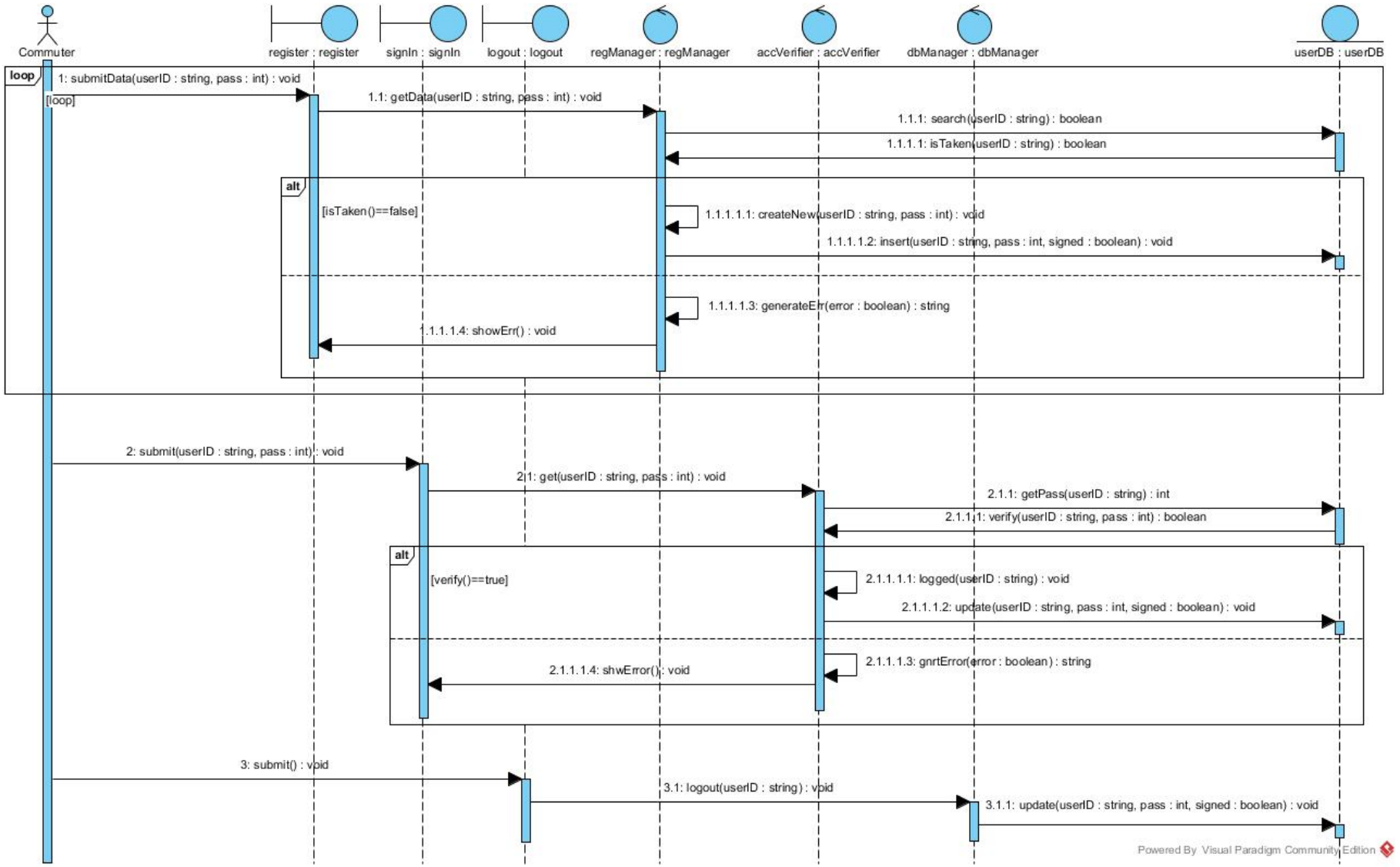


Diagramme de classes

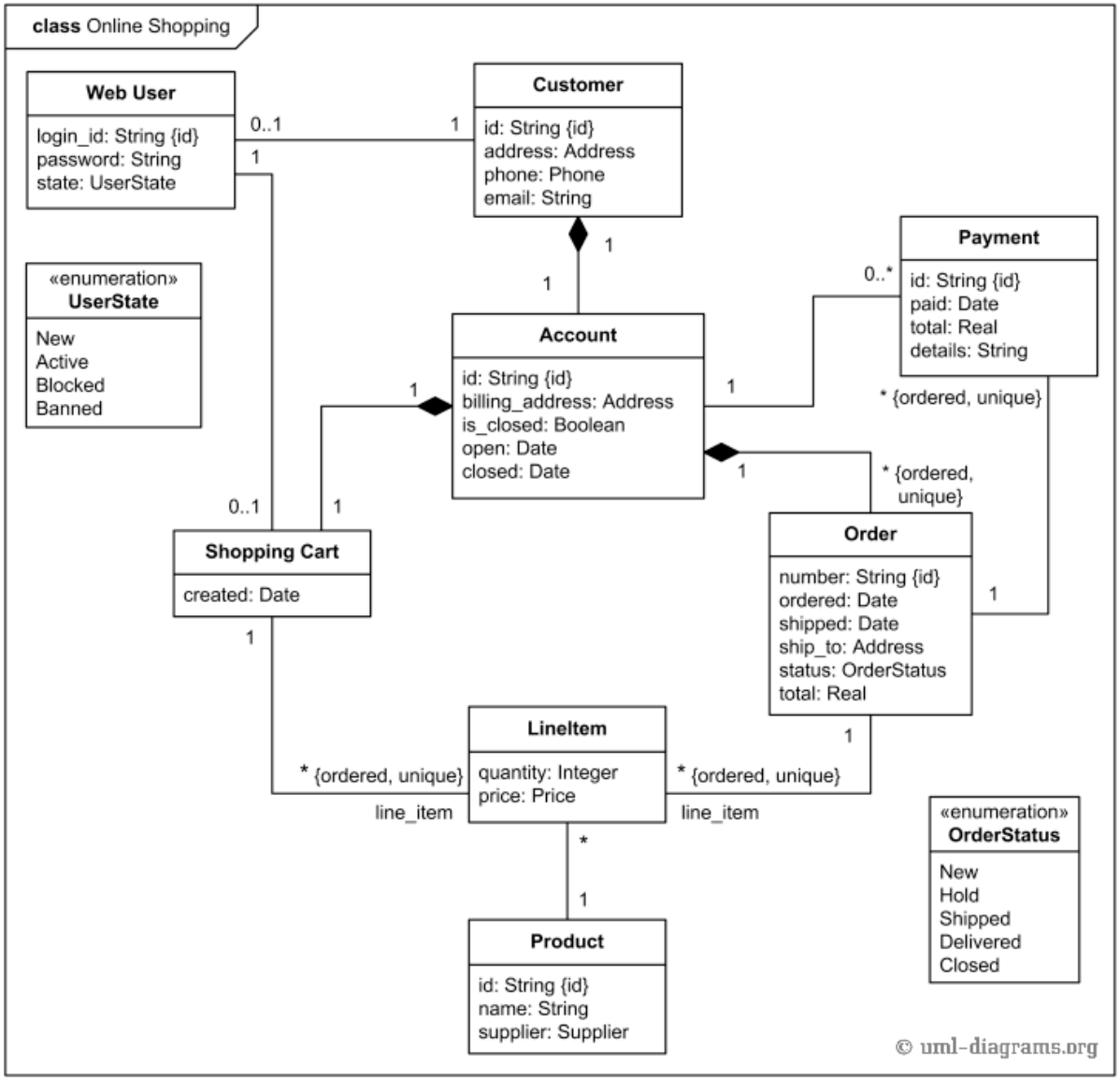
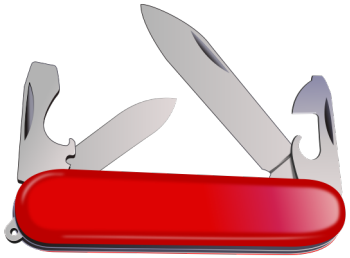
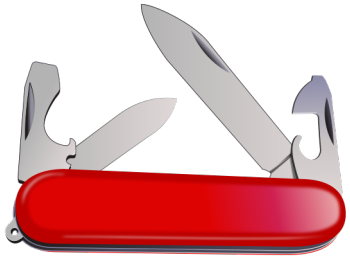


Diagramme de classes



- Rétro-ingénierie d'un système OO
- Modèle d'un métier (même si on ne va pas l'implémenter en OO)
- Modèle métier dans des architectures N-tiers (pour mapping object-relationnel automatique) -> cours ISA-DEVOPS au S8
- Modèle métier dans en *Domain-Driven Design* -> cours Micro-Services – SOA en SI5

Diagramme de séquences



- Rétro-ingénierie ou documentation d'appels entre objets, composants, services, systèmes distribués, etc.

Diagramme de packages

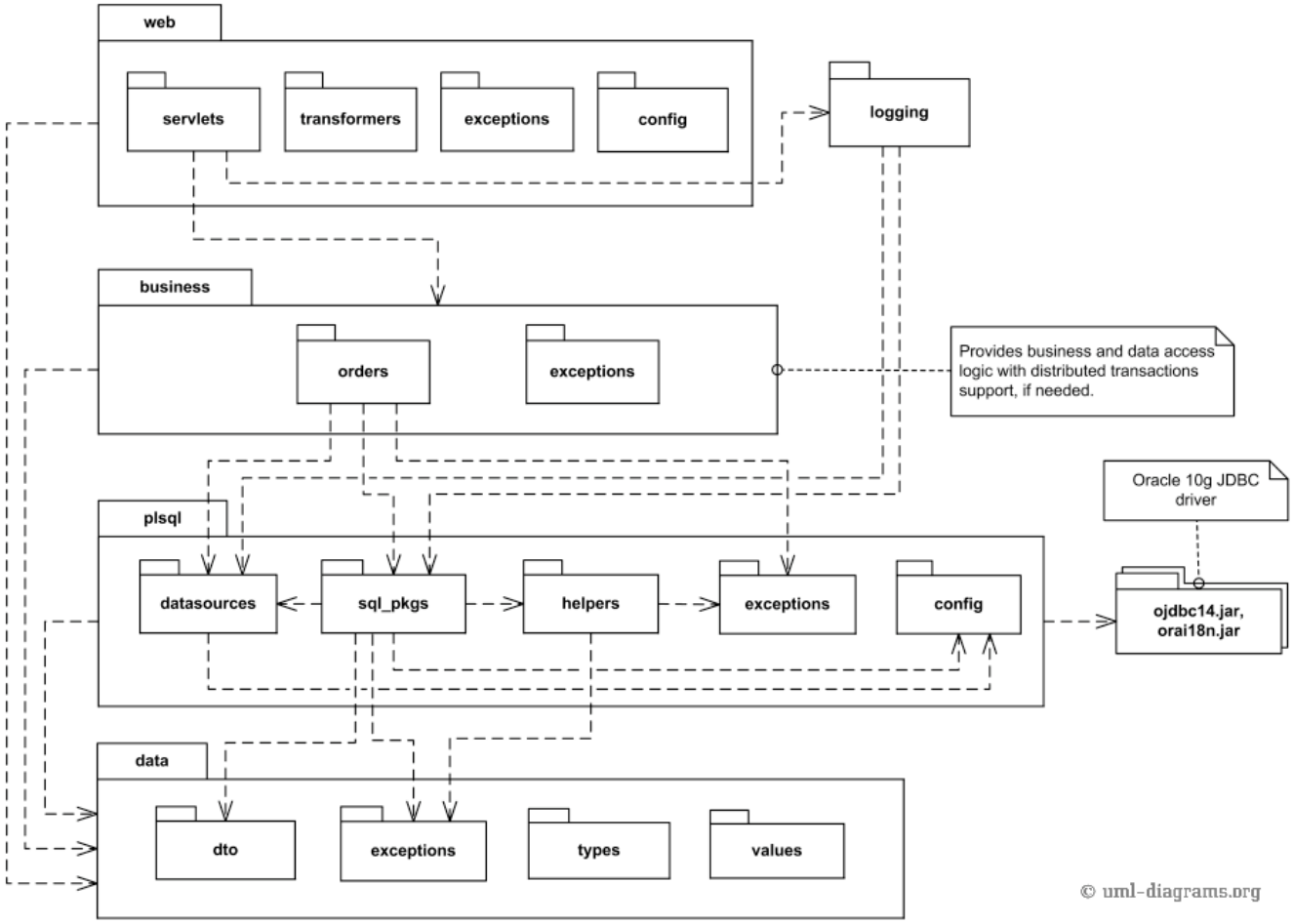
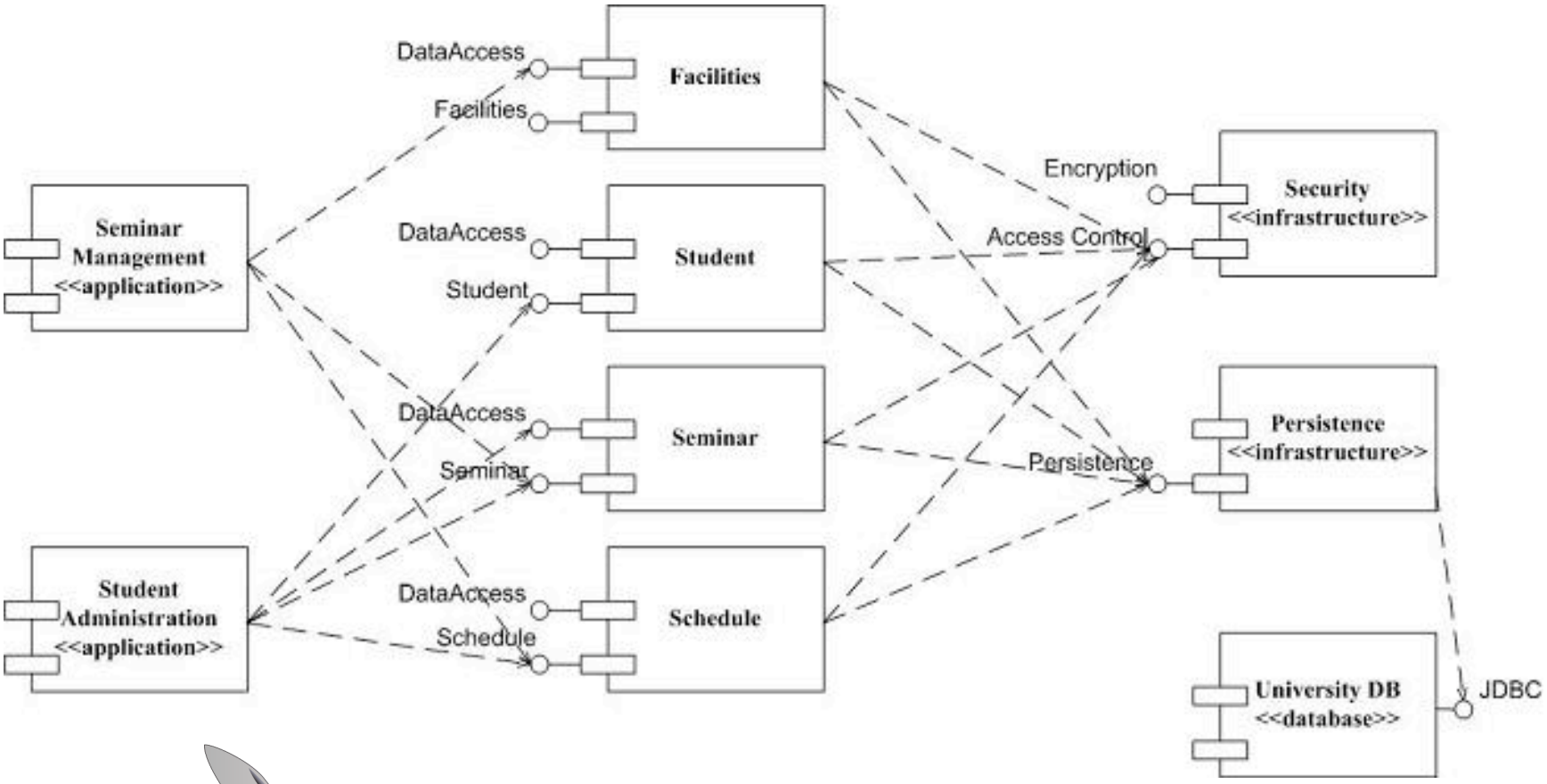
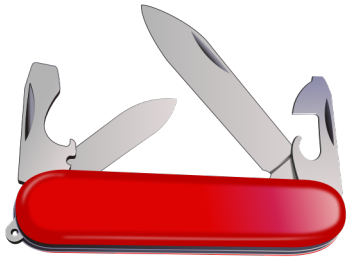


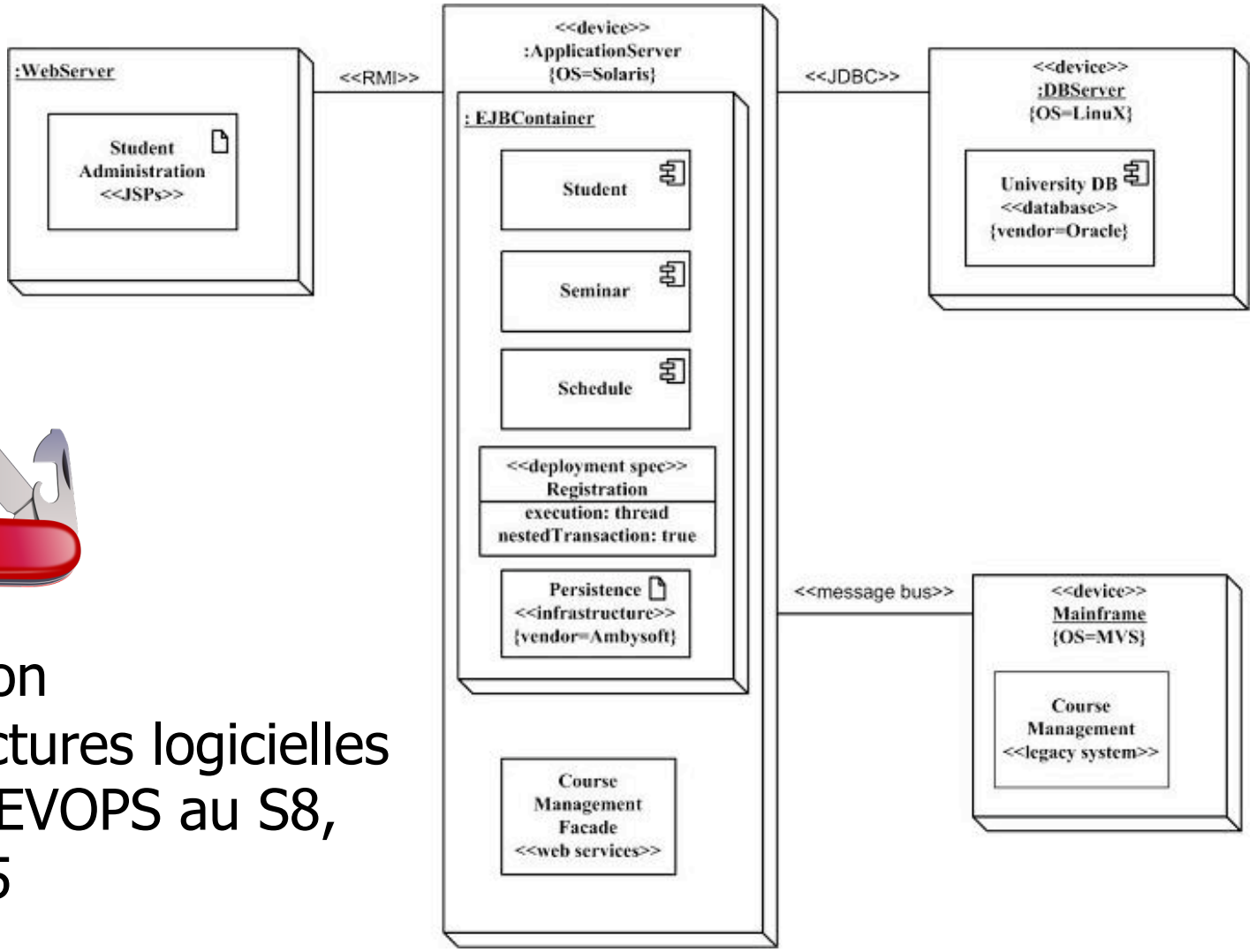
Diagramme de composants

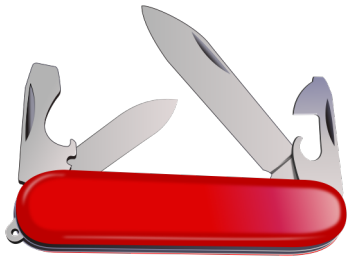




Description d'architectures logicielles -> ISA-DEVOPS au S8, AL en SI5

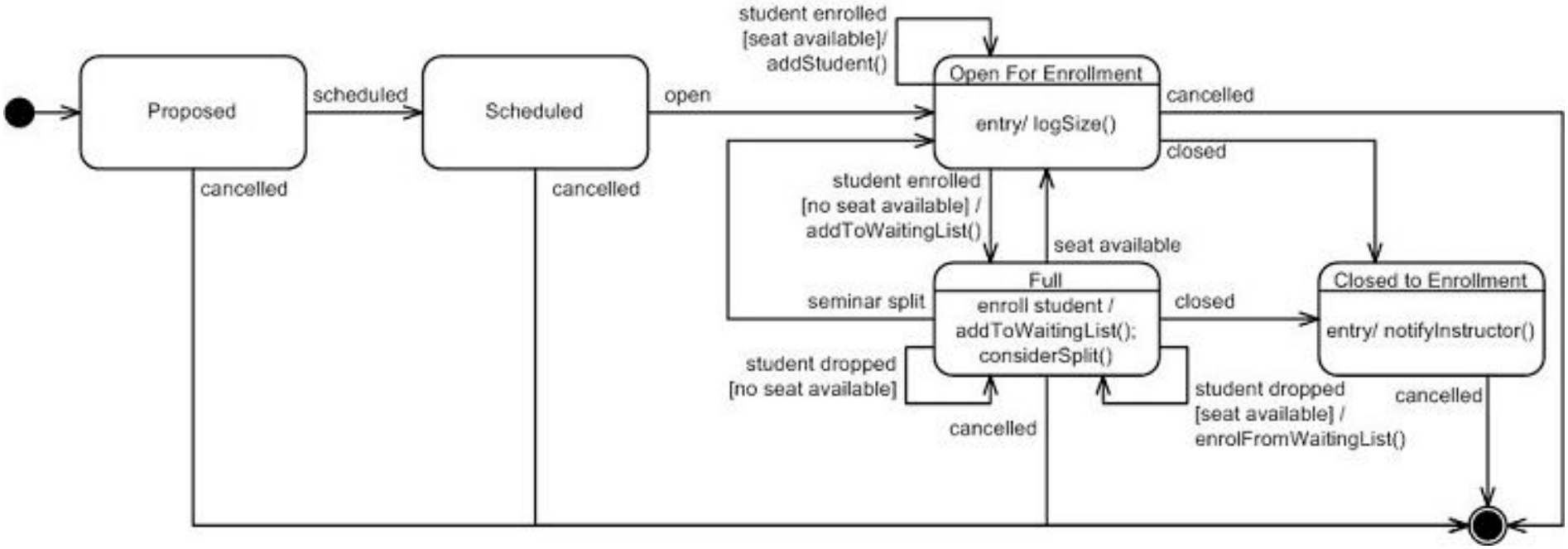
Diagramme de déploiement

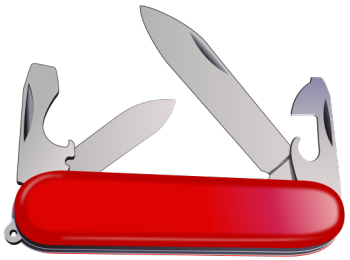




Description d'architectures logicielles -> ISA-DEVOPS au S8, AL en SI5

Diagramme d'états
(cf. cours Finite State Machines)





- Partout ou on a besoin de spécifier une machine a états (processus métier, système « bas niveau », etc.)