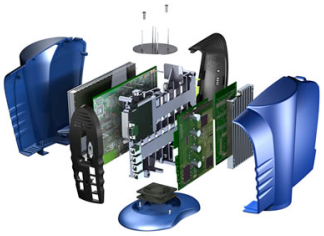
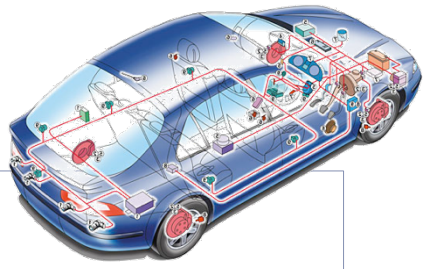
Vers une conception orientée objet des systèmes embarqués





Jérémie Guiochet
Université Toulouse 3, LAAS-CNRS
guiochet@laas.fr
www.laas.fr/~guiochet

Support de Cours ENSEEIHT– 3ème année AII

Plan du cours

- Partie 1 – Les systèmes embarqués – Une introduction
  - Chapitre 1 : Introduction aux SE (définition, exemples, caractéristiques générales)
  - Chapitre 2 : Les éléments de la conception matérielle et logicielle
  - Chapitre 3 : La sûreté de fonctionnement
  - Chapitre 4 : Le temps réel
- Partie 2 – Le développement des systèmes (embarqués) : vers un processus de développement orienté objet
  - Chapitre 5 : Généralités génie logiciel
  - Chapitre 6 : *Unified Modeling Language*
  - Chapitre 7 : Le Processus Unifié - Une Démarche Orientée Modèle

Crédits

- Cours de Mario Paludetto (Conception SE + temps réel)
  - http://www.laas.fr/~mario
- Cours de Jean Claude Laprie (Sûreté de fonctionnement)
  - http://www.laas.fr/laas/1-4287-Groupe-TSF.php
- Cours de Pierre Alain Muller (UML)
  - http://www.irisa.fr/triskell/members/pierre-alain.muller

PARTIE 1 : Les systèmes embarqués

Une introduction

Chapitre 1

Introduction aux systèmes embarqués

Les systèmes embarqués

- Définition
  - Système qui utilise conjointement matériel et logiciel pour réaliser une fonction spécifique) pour réaliser une fonction spécifique
  - Fait partie d'un système beaucoup plus vaste qui n'est pas nécessairement un ordinateur (système enfoui, *embedded system*)
  - Travaille à temps contraint et de façon réactive avec l'environnement
- Fortes contraintes :
  - Consommation, encombrement, poids, dissipation thermique, etc.
  - Robustesse, sécurité, fiabilité, etc.
  - Temps de réponse
  - Coût, maintenance
- S'oppose à l'informatique généraliste (Multi-usages et multi utilisateurs)
- Informatique embarquée :
  - Dès les années 80
  - Possibilité d'embarquer de l'intelligence -> souplesse et évolution
  - **Aujourd'hui : 5 % des CPU pour PC et 95 % pour embarqué**

Systèmes embarqués: Domaines d'application

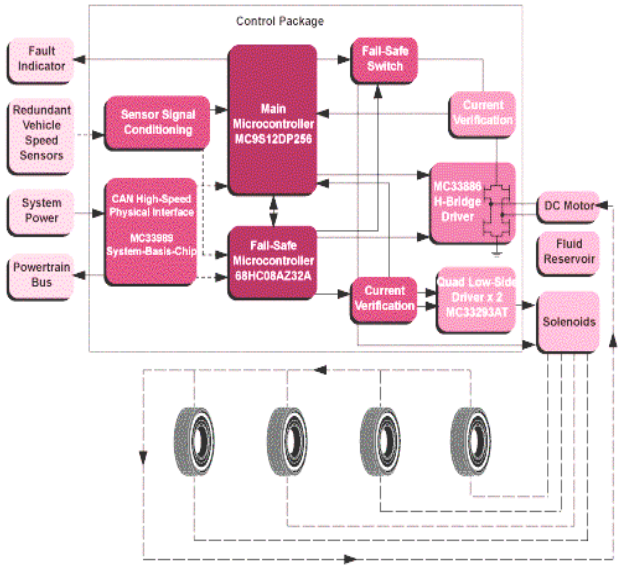
- Systèmes orientés calcul
  - Caméras, appareils photo et TV numériques, etc.
  - Multimédia, consoles de jeu, etc.
  - Télécommunications (téléphonie, GPS, etc.)
- Systèmes orientés contrôle
  - Electro-ménager
  - Automobile: ABS, Commandes moteur, navigation, etc.
  - Chaînes de fabrication et de montage, trains de laminoirs, traitements de surface, etc.
  - Avionique, spatial et défense: Calculateurs de bord, Commandes de vol, régulation moteurs, guidage, maintenance, etc.
  - Production, gestion et contrôle de l'énergie

Premier exemple : Automobile

- 65 microcontrôleurs embarqués aujourd'hui dans une Mercedes-Benz classe S
  - moteur (richesse du mélange, injection, allumage, pollution, etc.)
  - freins, tableau de bord, airbags, autoradio, vidéo, péage automatique, système de navigation, climatisation, régulateur de vitesse, direction, stabilisation, contrôle de trajectoire, détection d'obstacles, boîte de vitesse
  - détection de sommeil, stop & go, etc.

Automobile (2)

- Ex.: Système ABS



9

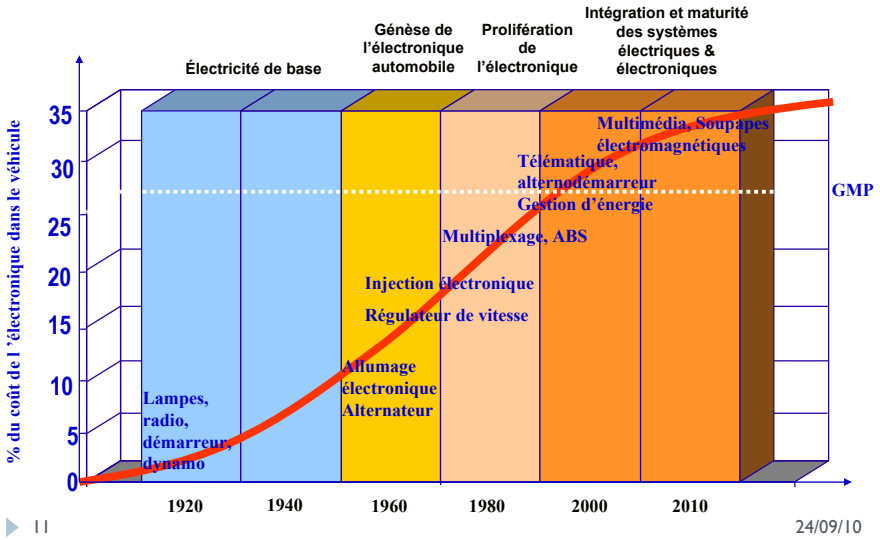
Automobile (3)

- Dans un proche avenir
  - Coordination véhicule avec infrastructure routière
  - Coordination entre véhicules
  - Par exemple pour faciliter le trafic ou éviter les collisions
- Disparition des liaisons mécaniques traditionnelles (colonne de direction, commande hydraulique, câble, etc.) est déjà programmée.
- Elles seront remplacées par des réseaux de modules électroniques contrôlant des actionneurs et des émulateurs (pour remplacer volant, pédales, etc.). On parle de technologie *X-by-Wire*.

10 24/09/10

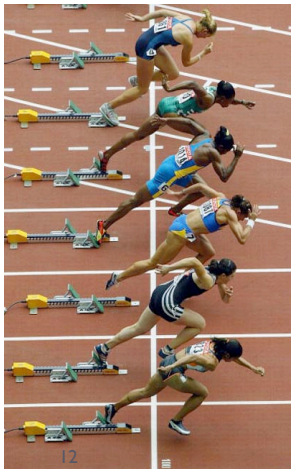
Automobile (4)

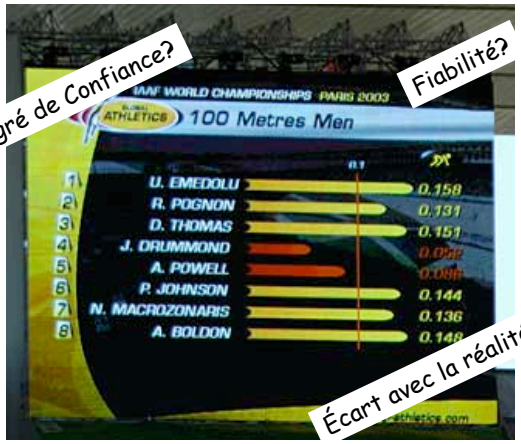
Extrait de la présentation de Joseph Beretta / PSA - *16 et 17 Juin 2003* – http://www.systemes-critiques.org/SECC/



Deuxième exemple : faux départs

- Systèmes Embarqués et performances temporelles

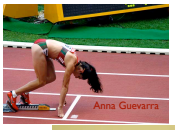


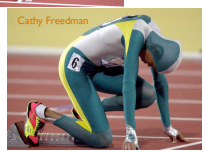


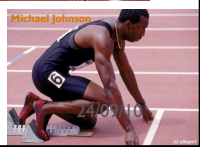
Images IAAF – championnat du monde 2003 au stade de France

Faux départs (2)

- Le système ou Les systèmes?







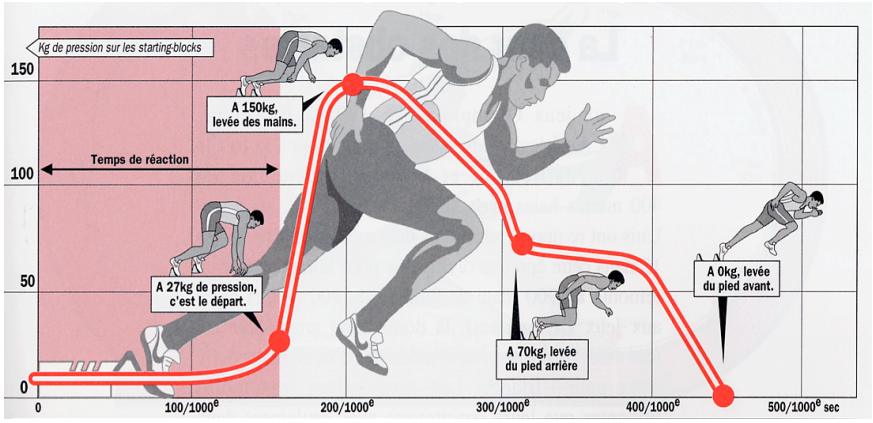
Images IAAF

Faux départs (3)
Exigences fonctionnelles

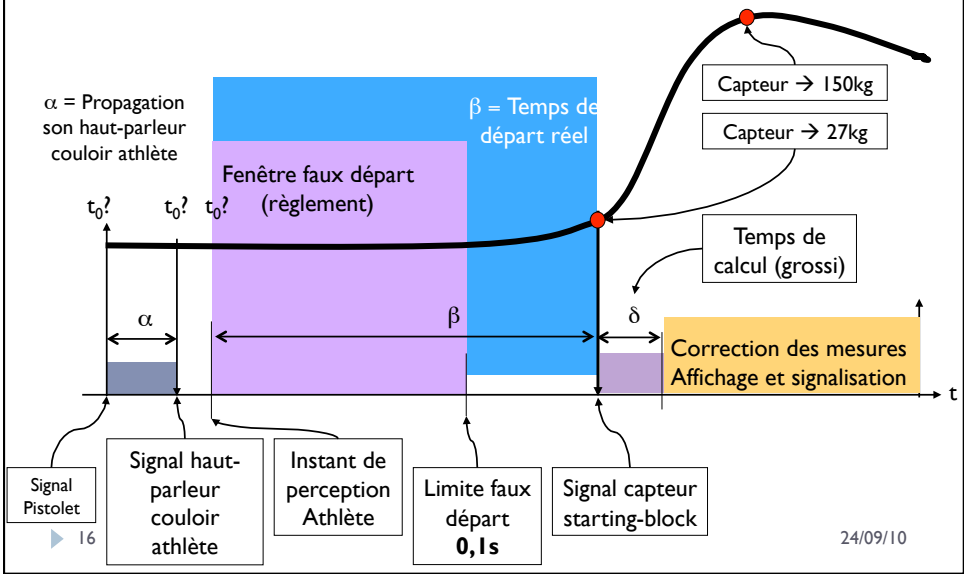
- Le signal de départ doit être perçu au même moment par tous les athlètes.
  - => Déclenchement par un pistolet relié au système
  - => Haut-parleur pour signal de départ acoustique dans chaque starting-block
- Starting-blocks avec capteurs de départ intégrés.
  - => Capteurs d'efforts précis et identiques sur les 8 couloirs
- Calcul des faux départs:
  - Au moment du départ, l'athlète agit sur les capteurs.
  - Le Système collecte les temps de départ de chaque athlète et les compare au temps de seuil règlementaire (0,1s).
  - Si un athlète a réagi trop vite, le "starter" reçoit un beep dans son casque téléphonique.
  - Affichage et imprimante désignent la (es) piste(s) fautive(s).

Faux départs (4)
La vue utilisateur

- Technique et dynamique du départ 100m



Faux départs (5)
Contraintes temporelles et exigences système

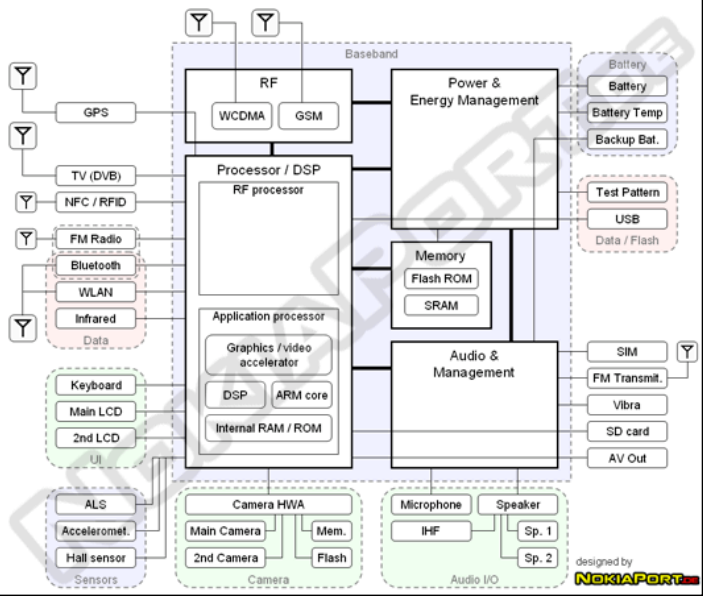


Architecture Systèmes embarqués

- Lister (et/ou dessiner) rapidement les éléments d'un téléphone portable (le votre par exemple) sans regarder le transparent suivant :

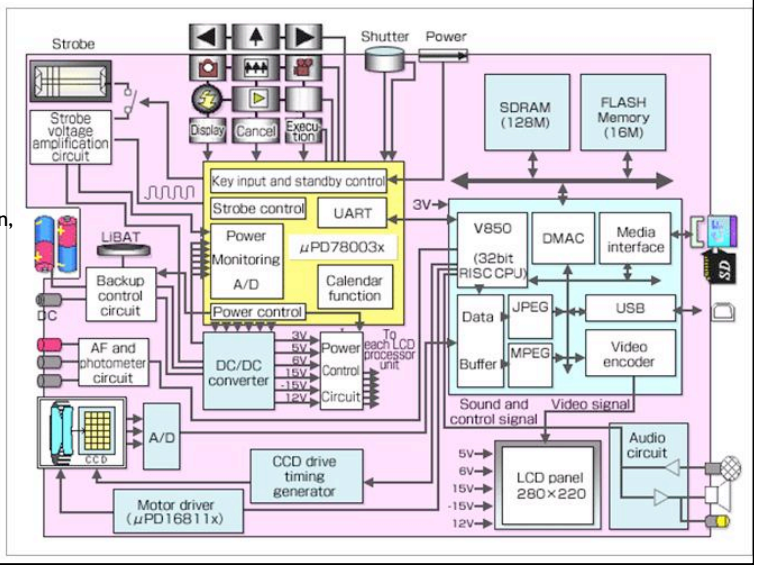
Exemple d'architecture: System on Chip (SoC)
Le téléphone portable

- Contraintes: taille, poids, consommation, performance télécom, disponibilité



Exemple d'architecture: System on Chip (SoC)
Appareil photo numérique

- Contraintes: taille, poids, consommation, temps d'acquisition, stockage, etc.



19

Systèmes embarqués: particularités

- Système Embarqués => Systèmes Enfouis (Embedded Systems)
  =>Systèmes C3I (Commande, Contrôle, Communication, Intelligence)
  - **Réactif**
    - En constante interaction avec l'environnement et au rythme de celui-ci
  - **Intelligent**
    - Capable d'analyser la situation/environnement et de prendre des décisions
  - **Temps réel**
    - Réactions perçues comme immédiates par l'environnement
    - Respect des temps et des échéances opératoires
  - **Sûr de fonctionnement**
    - Robuste, fiable
    - Tolérant aux fautes

- Problème:

  Modélisation, Conception et Implémentation des SE et/ou STR?

20 24/09/10

Chapitre 2

Les éléments de la conception matérielle et logicielle

Electronique ou informatique ?

- Implémentation matérielle :
  - Plus rapide mais plus chère
  - Obsolescence rapide des composants et des techniques
- Implémentation logicielle :
  - Plus lent mais plus flexible et évolutif
- Evolution vers toujours plus d'intégration (System On Chip : système sur une puce)
- Frontière de plus en plus floue avec par exemple les circuits de logique programmable qui offrent rapidité et flexibilité

Electronique ou informatique ? (2)

- Paramètres du choix :
  - Coût global, fonctionnalités & performances, fiabilité, robustesse, contraintes matérielles (poids, encombrement, consommation,…), environnement, durée de vie de l'application, délai de mise sur le marché, etc.
  - Exemple du spatial : composants éprouvés mais de production de une ou quelques unités (obsolescence pas un problème)
- Approche traditionnelle de développement d'un système embarqué
  - Choix et développement de l'architecture matérielle : composants, capteurs, processeur, etc.
  - Réalisation du logiciel pour réaliser l'application avec le matériel

Electronique ou informatique ? (3)

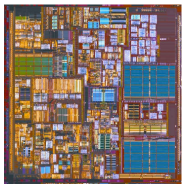
- Complexité croissante
  - Systèmes de plus en plus complexes
  - Optimisation globale en plusieurs itérations
  - Correction des erreurs, maintenance et évolution des systèmes
- Temps de développement
  - De plus en plus réduit
  - Participe au coût
  - Concurrence (appel d'offre, délai de mise sur le marché, …)
  - Exemple de l'industrie automobile : 1 à 2 ans aujourd'hui pour 3 à 5 ans avant
  - Il ne faut pas sacrifier la fiabilité/sécurité/disponibilité/etc.

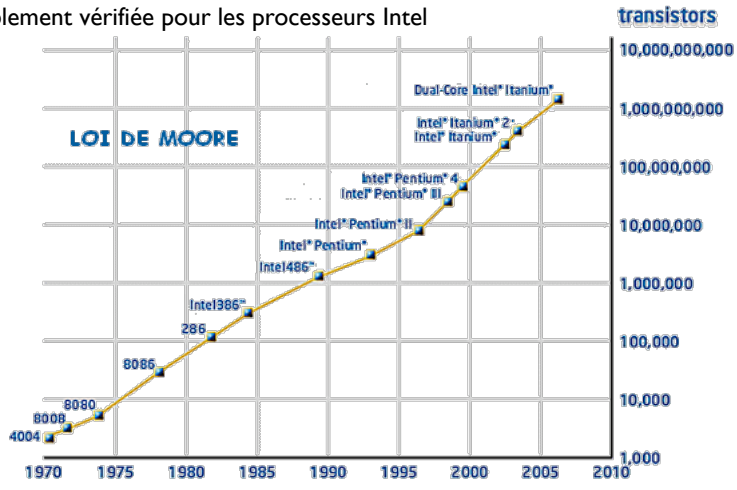
Technologie clé: Processeurs

- Grande variété d'architecture de processeurs
- Un processeur n'est pas nécessairement programmable
- On distingue généralement
  - Les processeurs à usage généraux (GPP)
  - Les processeurs spécifiques à certaines applications (Application Specific Processor, ex: DSP)
  - les processeurs dédiés à une tache (single purpose processor, ASIC)
- NB : Orientation «Système sur Composant »
  - pSoC (Programmable System-On-Chip)
  - Mixte de technologies
    - Association microcontrôleur –DSP
    - Association microcontrôleur –logique programmable
    - Autre…

Loi de Moore

- Le nombre de transistors des processeurs (circuits complexes) double tous les deux ans (1975)
- Remarquablement vérifiée pour les processeurs Intel





Processeurs à usage général (1)

- Processeur programmable utilisé pour de nombreuses applications (microprocesseur)
- Caractéristiques
  - Une mémoire pour le programme
  - Un chemin de donné généraliste comprenant un unité arithmétique et logique (ALU) puissante et un gros banc de registre
- Intérêt :
  - Time to market et coût
  - Flexibilité
- Exemple: Pentium, PowerPC, ARM, MIPS, etc.

Processeurs à usage général (2)
Les CPUs utilisés pour l'embarqué

32-bit Family	Number Sold
ARM	151 million
Motorola 68k	94 million
MIPS	57 million
Hitachi SuperH	33 million
x86	29 million
PowerPC	10 million
Intel i960	8 million
SPARC	3 million
AMD 29k	2 million
Motorola M-Core	1 million

source: T. R. Halfhill. Embedded Market Breaks New Ground. Microprocessor Report, January 2000

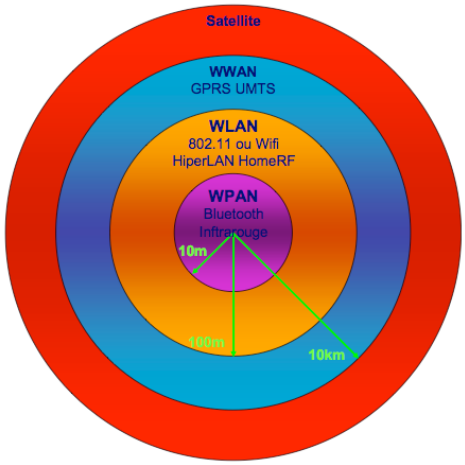
Processeurs dédiés

- Circuits intégrés destinés à exécuter exactement un programme: coprocesseur, accélérateur matériel ou périphérique.
- Caractéristiques:
  - Contient seulement les composants nécessaires à l'exécution du programme concerné
  - en général pas de mémoire de programme
- Intérêt :
  - Rapidité
  - Faible consommation
  - Surface
- Exemple: unité de calcul flottant, contrôleur USB, PCMCIA, decoder MPEG, etc.

Processeurs spécifiques

- Processeur programmable optimisé pour une classe particulière d'applications (ASIP: Application Specific Integrated Processor).
- Caractéristiques:
  - Mémoire de programme
  - Chemin de donnée optimisé
  - Unités fonctionnelles spécifiques
- Intérêt :
  - Flexibilité
  - performances: surface, rapidité, consommation
- Exemple: DSP, microcontrôleur (processeur 4bits, 8bits).

Les moyens de communications embarqués
(Quelques exemples)



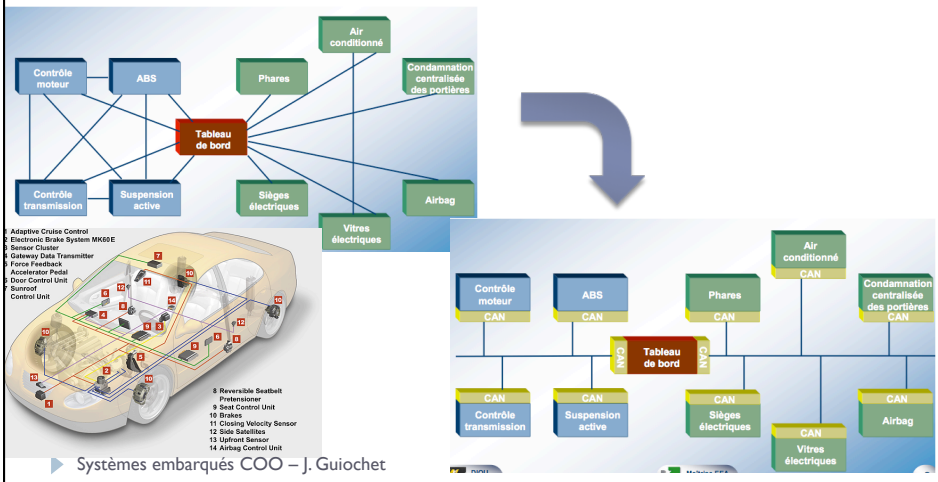
Les moyens de communications embarqués (2)

Bluetooth

- Faible portée Faible consommation
- Faible coût Réseaux réduits
- Communication de périphériques dans un espace d'opération personnel

Les moyens de communications embarqués (3)

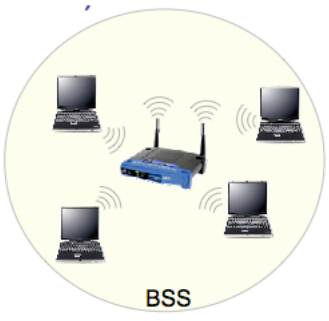
- Le bus CAN (Controller Area Network)



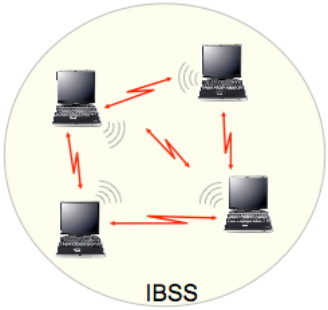
Les moyens de communications embarqués (4)

Wi Fi™ : Communication sans fils

- 2 modes : infrastructure et ad-hoc



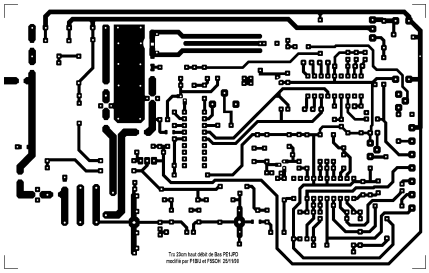
Basic Service Set



Independent Basic Service Set

Les moyens de communications embarqués

I²C BUS : faire communiquer entre eux des composants électronique très divers grâce à seulement 3 fils (Signal de donnée ,horloge et masse)



La géolocalisation

- Fournir la position, la vitesse, l'orientation, ... d'un objet (informatique) àune application
- Ces données sont des entrées de l'application
- Technologies d'acquisition
  - OutDoor : Satellite (GPS), Cellulaire (GSM), …
  - InDoor : GSM, WiFi, Bluetooth, RFID, …

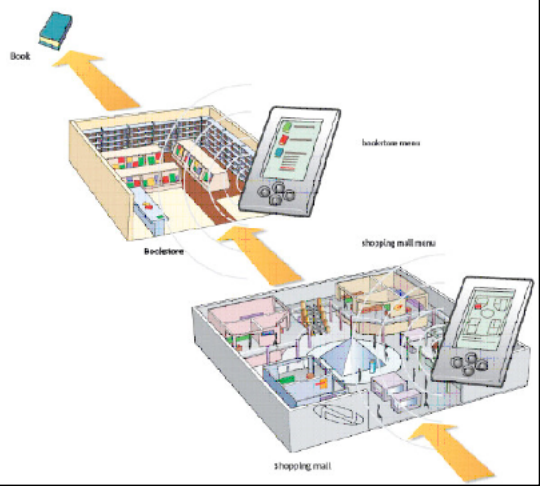
La géolocalisation : applications outdoor (innombrables)

- Civils
  - Géomètre, Cadastre, BTP, Agriculture, …
  - Transport (Assistance àla navigation, …)
  - Urgence (Guidage des secours, Quel est le véhicule de patrouille le +proche …)
  - Loisirs (Alpinisme, Randonnée, Voile, …)
  - Traçabilité(e-Track) (Conteneurs, Courrier rapide, Flot de véhicule,
  - Force commerciale, Flamme olympique pour Atlanta 1996, …)
  - Sécurité des biens (vol de véhicule, de conteneurs, …)
  - Réalité augmentée et Aide au Handicap (non voyant)
  - Commerce (quel est notre magasin le plus proche de chez vous ?)
- Militaires
  - Guidage d'armement (missile de croisière, …)
  - Assistance des troupes

37 24/09/10

La géolocalisation : applications indoor

- Aide à la navigation
  - Supermarché,
  - Musée, Parc à thème
  - Bibliothèque
  - Hôpital



38

La géolocalisation : GPS Terminals

- Modules GPS intégrables
- Terminaux embarqués
  - Dans un véhicule, un téléphone cellulaire, un appareil photo, …
- Terminaux portables
  - Avec/sans écran
  - Journal de positions intégré
    - Les positions peuvent être «infalsifiables»(signature électronique ?)
    - Fonctions «Homme à la mer»

Systèmes d'exploitation pour l'embarqué

- Grande hétérogénéité
- Mémoire et µProcesseurs limitées
- Capacité de communication
- Consommation énergie
- Contraintes temps réel
  - Non TR, « soft » RT, « hard » RT

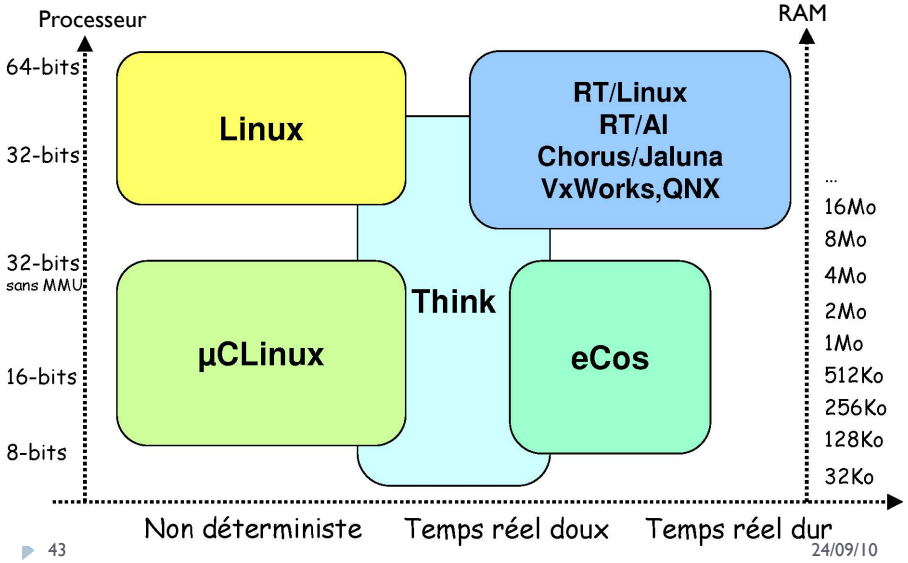
Plusieurs approches pour un OS pour l'embarqué

- Plus d'une centaine d'OS embarqués
- Noyau élagué :
  - Dur d'élaguer dans la dénomination (Linux, RTLinux, μ CLinux, etc.)
- OS Commercial
  - Cible un marché large donc fonctionnalités en trop (VxWorks, QNX, pSOS, etc.)
- OS Domaine
  - PalmOS, Symbian
- OS Custom
  - Couteux en temps. Difficilement portable et maintenable
- OS modulaire et flexible
  - Think, eCos
- Middleware/Intergiciel
  - Intermédiaire entre application et système
  - Souvent trop gros mais inévitable dans un système distribué

OS pour l'embarqué

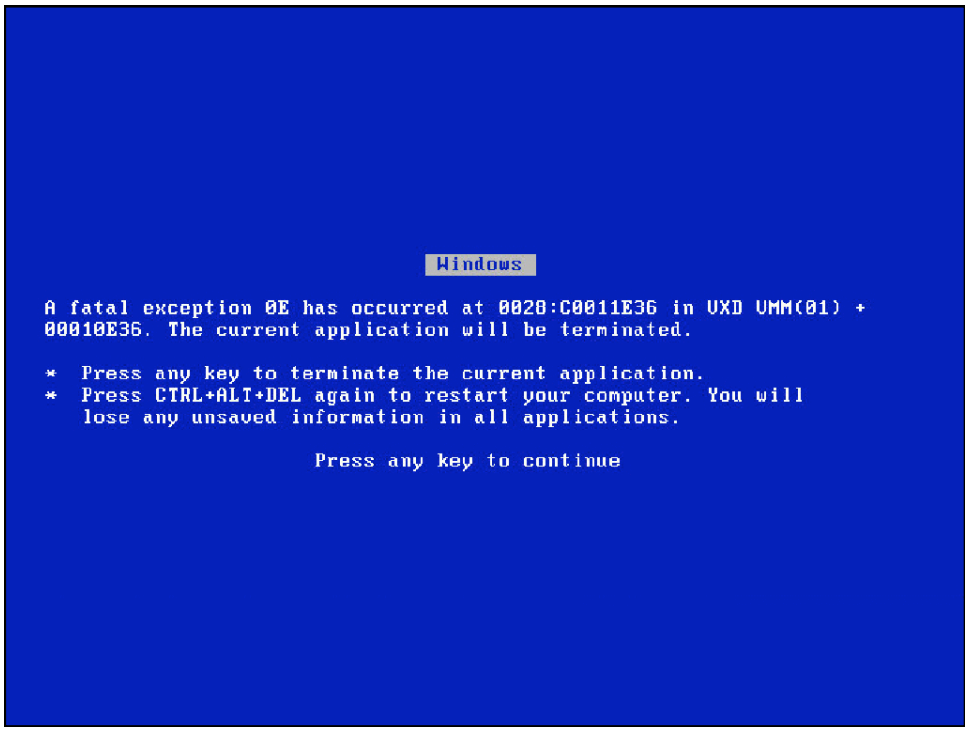
- Académique :
  - ExoKernel, SPIN, Think
  - Choices, OSKit, Coyote, PURE,2K
- Commercial
  - VxWorks, QNX, pSOS, WindowsCE
  - JavaOS, Jbed, MMLite, icWORKSHOP,Pebble
  - Open Source de qualité industrielle : Linux, µClinux,eCos
- LIRE : L. FFriedrich, J. Stankovic, M. Humphrey, M. Marley, J. Haskins, Survey of Configurable, Component-Based Operating Systems Embedded Applications" ,IEEEMicro, May-June 2001, pp54-68

Comparaison OS / HW



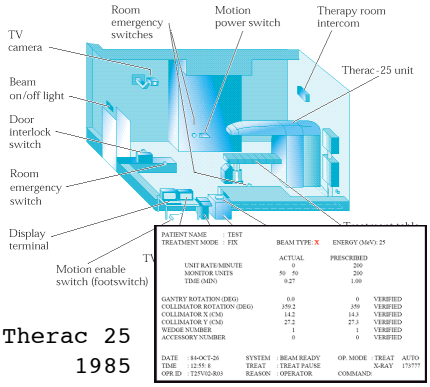
Chapitre 3

La sureté de fonctionnement

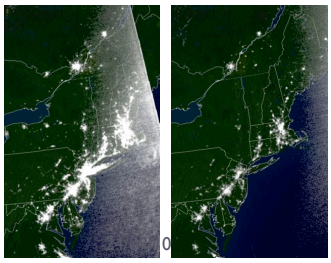


Systèmes critiques : Ariane 5 – vol 501

Ariane 5
1996



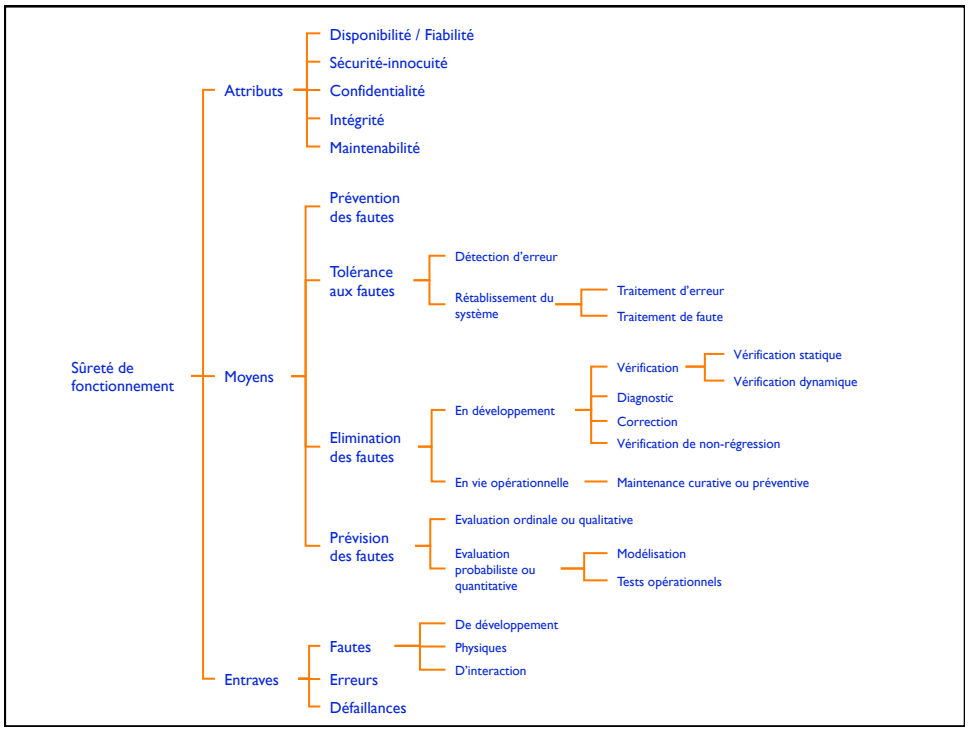
Therac 25
1985



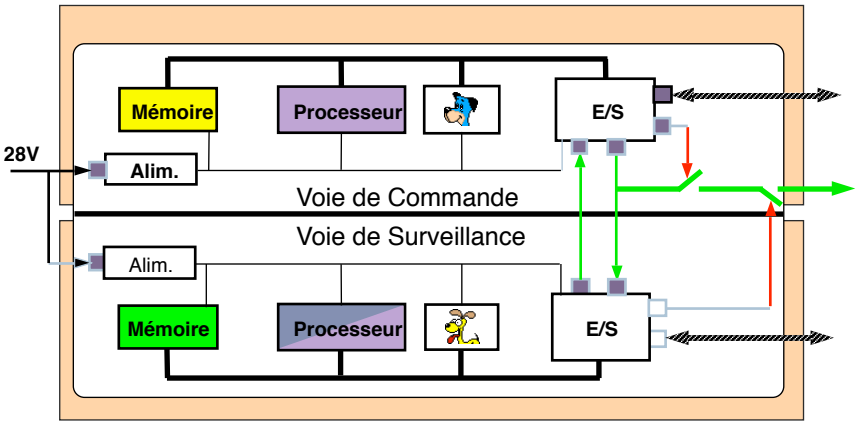
USA Blackout
2003

Systèmes embarqués critiques

- Critique : la plupart du temps du point de vue de la sécurité innocuité mais aujourd'hui de plus en plus d'importance de la sécurité-confidentialité
- Systèmes embarqués de plus en plus communicants sécurité des données ou protection contre attaques
- On distingue plusieurs niveaux de criticité selon qu'un dysfonctionnement met en jeu:
  - la sécurité des personnes (par ex: accident d'avion) ;
  - la réussite d'une mission (par ex: mauvaise trajectoire d'un satellite);
  - la réussite économique du produit (par ex: mise sur le marché tardive d'une console de jeux);
- Que faire : évaluer les risques et prendre les mesures nécessaires pour les réduire
- Utilisation de techniques de la sûreté de fonctionnement
  - Redondance de certains composants
  - Mode de fonctionnement dégradé mais sûr qui permettra d'attendre une intervention



Exemple : Architecture des calculateurs Airbus (1)



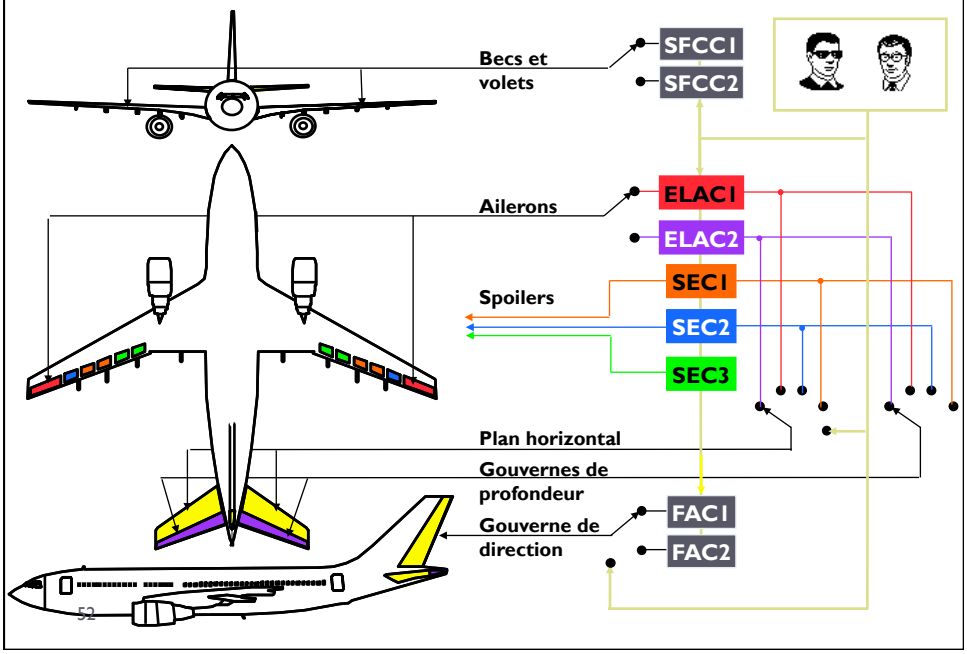
Protection foudre, EMI, sur/sous tension

Exemple : Architecture des calculateurs Airbus (2)

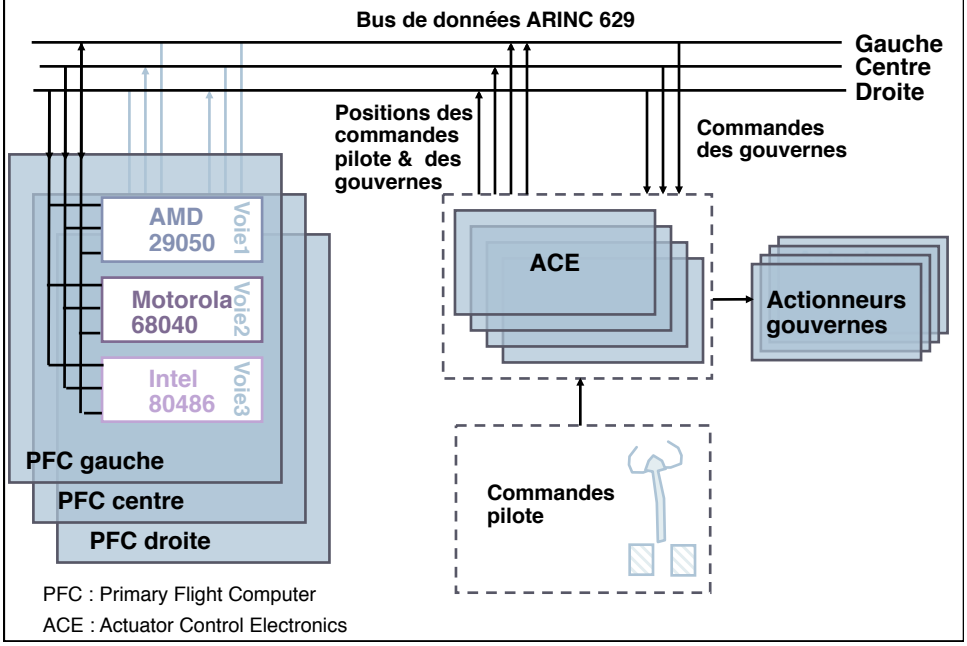
- Chaînes de production de logiciel différentes
  - équipes de conception différentes (Paris / Toulouse)
  - langages différents
  - outils de programmation différents
- Règles d'amplification de la diversification
  - organisation différente des programmes
  - allocations mémoires différentes
  - algorithmes différents
    - trigonométrie avec coordonnées polaires vs. coordonnées cartésiennes
    - fonctions numériques tabulées vs. calcul par équation
  - optimisations avec des buts différents *(temps vs. taille programme)*
  - précisions de calcul différentes *(12 bits vs. 8 bits)*
- Diversification au niveau système
  - calculateurs de fabricants différents
  - processeurs de types différents

Exemple : Architecture des calculateurs Airbus (3)

Airbus 320



Exemple 2 : Boeing 777 (1)

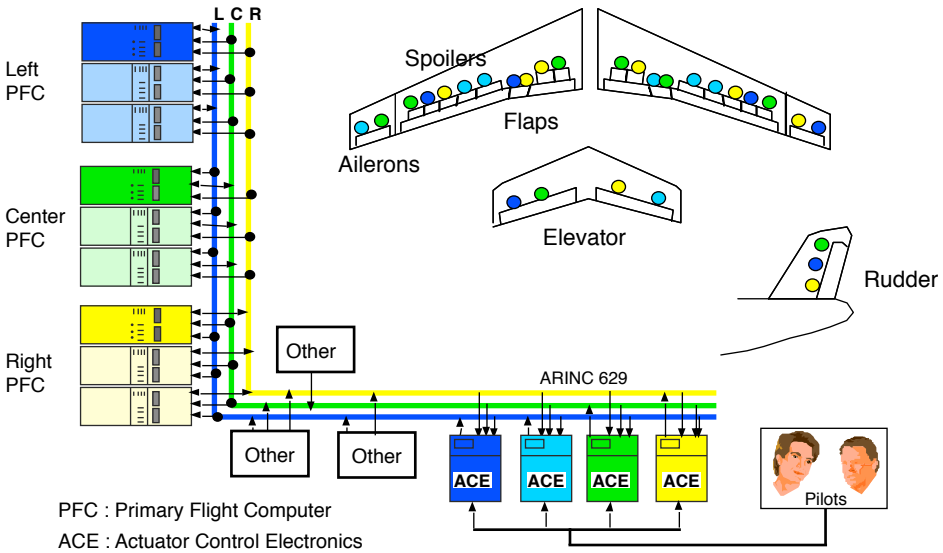


PFC : Primary Flight Computer
ACE : Actuator Control Electronics

Exemple 2 : Boeing 777 (2)

- Processeurs et compilateurs des PFC
- Fonctions commande et moniteur des ACE
- Données inertielles par dissimilarité des unités
  - ADIRU (Air Data Inertial Reference Unit)
  - SAARU (Secondary Attitude and Air Data Reference Unit)
- Matériel doublé et dissimilaire dans l'AFDC (autopilote)

Exemple 2 : Boeing 777 (3)

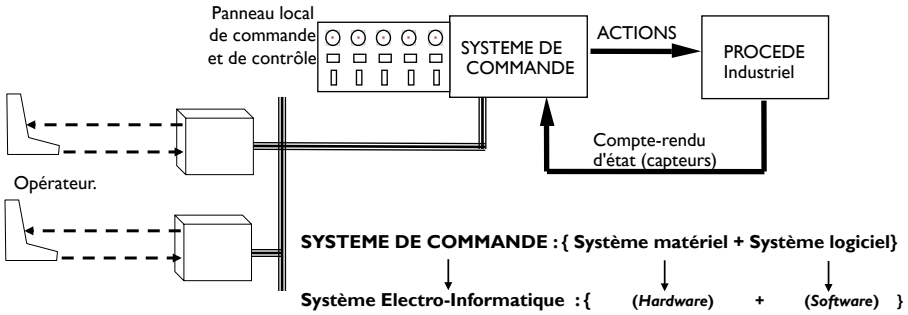


Chapitre 4

Le temps réel

Systèmes embarqués: configuration de base

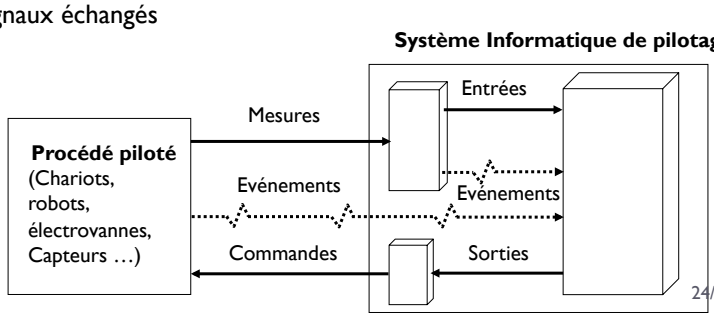
- Exemple de configuration classique
  - Commande en Temps Réel d'un Procédé Industriel



Systèmes embarqués: Temps réel

- Définitions d'un système temps réel
  - Système qui doit satisfaire des contraintes temps de réponse explicites et bornées.

    Si les bornes temporelles ne sont pas respectées, des dégradations des performances et des mauvais fonctionnements apparaissent.
  - Système dont *l'exactitude* dépend non seulement des résultats logiques mais également des instants où ces résultats sont fournis.
- Signaux échangés



Systèmes embarqués: Les besoins dits "temps réel"

- Respect des contraintes temporelles
  - Maîtrise des temps d'exécution (approximation du pire cas)
  - Ordonnancement des différents traitements sur la ressource processeur
  - Détection des Violations Temporelles: Délais et Chiens de garde.
- Diversité des entrées/sorties
- Comportements // et concurrents
- Support d'opérations distribuées et fiables
- Sûreté de fonctionnement
- Déterminisme
- Prédiction

Systèmes embarqués: classes de systèmes temps réel

- **Systèmes à temps réel strict/dur** (Hard Real-Time) dits *critiques: le non respect d'une échéance temporelle a des conséquences graves (humaines, économiques, écologiques, …)*
  - Commande et contrôle dans l'avionique, le spatial, le nucléaire, la production de matériaux toxiques, …
- **Systèmes à temps réel souple/mou** (Soft Real-Time): tolérance du non respect d'une contrainte temporelle ou d'un défaut système (fonctionnement en mode dégradé, ou cas d'une garantie probabiliste)
  - ex.: dégradation des performances temporelles de l'injection électronique
  - mais un défaut de périodicité relative est de type critique (détérioration moteur possible)
- De nombreux systèmes ont des contraintes strictes imposées par les opérations de l'environnement

Systèmes embarqués:
Développement des STR - Intro (1)

- Le passé
  - Méthodes *maison* (aucun formalisme, aucune vérification)
  - Programmation de bas niveau
  - Recherche de rapidité à tout crin (politique du qui peut le plus, peut le moins)
  - Évaluation des performances à posteriori
  - Pas vraiment de méthodologie couvrant le cycle de vie
- Et les fausses idées
  - Il faut exécuter le plus vite possible
  - Après les tests le logiciel ne peut plus faillir

⇒ Nombreuses tâches sur-estimées

⇒ L'ordonnancement devient le Problème majeur des STR classiques
  - Ordonnancement par heuristiques ⇒ Explosion combinatoire ⇒ Systèmes non déterministes
  - Ne peut s'appliquer qu'aux systèmes simples (temps de calcul bornés et mesurables, ordonnancements faisables – tâches périodiques- …)

Systèmes embarqués:
Développement des STR - Intro (2)

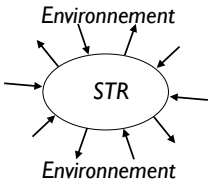
- Le présent et le futur
  - Besoins:
    - automatismes fiables, sûr, puissants, rapides, maintenables et peu coûteux
  - Réactions (aux erreurs du passé)
    - Évaluation fine des besoins non fonctionnels
    - Fini les Méthodes et outils *maison*
    - Développement sur des bases formelles
    - Développement en faisant abstraction du support matériel (codesign en VHDL)
    - Cycle de développement composé de nombreux **cycles complets itératifs** intégrant en priorité:
      - Vérification (preuves)
      - Validation (tests fonctionnels, simulation et prototypage virtuel)

⇒ Intégration de techniques formelles dans les méthodes

⇒ Ordonnancement
  - Intégré aux modèles et aux langages
    - SA-RT, UML, … + Automates + RdP …
    - Ada, Occam … Esterel, Lustre, …
  - A défaut, noyaux réduits à politiques déterministes

Systèmes embarqués: Développement des STR - Intro (3)

- Vue Globale
  - L'environnement est asynchrone
    - Les instants d'arrivée des données sont quelconques et non connus d'avance
    - Les sorties sont à fournir à des échéances qui peuvent être variables
  - Deux types de traitement intimement liés
    - Aspect réactif
      - Interaction avec l'environnement
      - Maîtrise du temps et des échéances
      - Contrôle des états des processus
    - Aspect transformationnel
      - Calculs et traitements propres à l'application
        - Fonctions de transfert, équations différentielles, trajectoires, …
      - Calculs de données internes inhérentes à la structuration



Développement des Systèmes embarqués - Les courants de pensée

STR

Dichotomie

Partie réactive ⟺ Partie transformationnelle

Modèles déterministes

Algorithmes à temps constant (*any time*)

Vision synchrone
- Hypothèse synchrone forte: temps de calcul et de communications = 0 (diffusion, *broadcasting*)
- Langages formels: Statecharts, Esterel, Lustre, Signal, …

Vision asynchrone
- Modèles peu formels: SA-RT, HOOD, UML
- Modèles formels : automates, réseaux de Petri
- Langages peu formels: Assembleur, Ada, Occam
- Exécutifs temps réel

- *Réalisme de l'hypothèse Synchrone forte?*
- *Dichotomie avec l'environnement asynchrone*
- *Moyens de structuration des systèmes complexes?*

- *Sémantique temporelle faible*
- *Attention aux exécutifs à politiques non déterministes*
- *Le concepteur doit vérifier le non déterminisme à tous les niveaux*

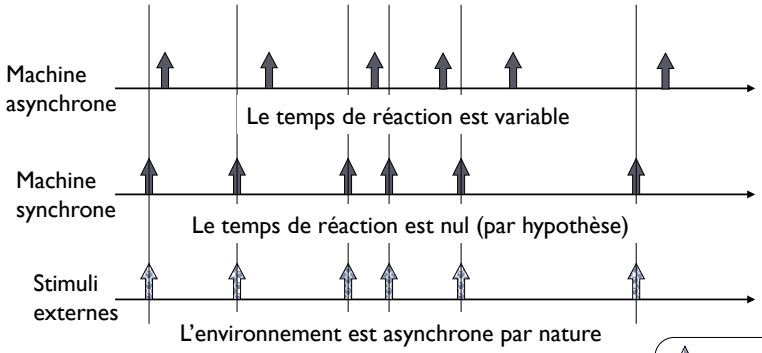
Développement des Systèmes embarqués: Approches Synchrone / Asynchrone (1)

- SYNCHRONE: Qui se produit dans le même temps. […]
  - [*En parlant d'un phénomène*] Qui se produit au même moment qu'un autre ou à intervalles réguliers **par rapport à un autre**. Mouvement, oscillation synchrone. […]
  - 1. [*En parlant d'un mode de transmission informatique*] "Dans lequel l'instant de l'émission de chaque élément binaire est déterminé par des signaux régulièrement espacés dans le temps" (SCOM *Informat.* 1977).
  - 2. [*En parlant d'un mécanisme*] Qui produit des mouvements synchrones. Calculateur synchrone. "Calculateur numérique ou ordinateur rythmé par une horloge mère dans lequel chaque opération élémentaire démarre sur un top horloge et dure un nombre fini de tops d'horloge" (Morvan 1980). […].

[Le TLF informatisé, http://zeus.inalf.cnrs.fr/tlf.htm]

Développement des Systèmes embarqués: Approches Synchrone / Asynchrone (2)

- Approches synchrone et asynchrone
  - Réactions à des stimuli externes



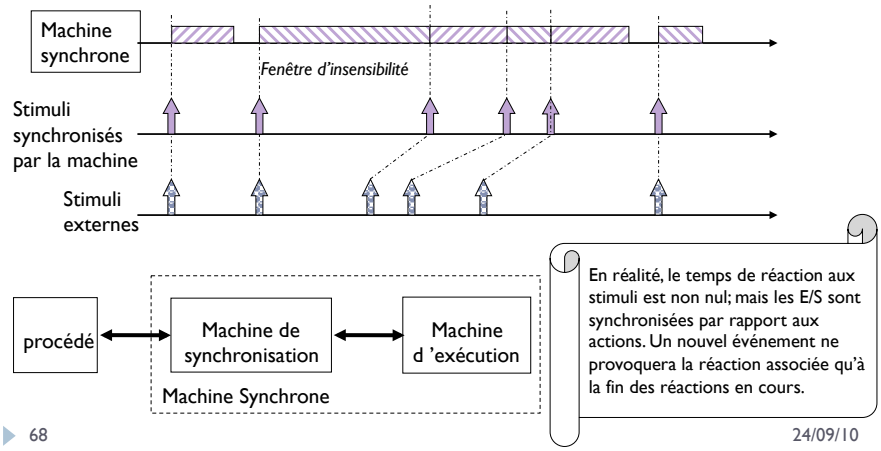
Fonctionnement correct si pour chaque couple stimuli-réponse:
le temps de réaction << temps de réponse

Développement des Systèmes embarqués Approches Synchrone / Asynchrone (3)

- Hypothèse Synchrone:
  - Temps de réaction du système négligeable ↔ dynamique de l'environnement
  - ⇒ Le Système est toujours "en phase" avec son environnement, toujours prêt à accepter des entrées.
  - ⇒ Pas besoin de préemption (interruption / reprise)
  - ⇒ Pas de "pseudo-parallélisme". Si parallélisme toutes les opérations se font en phase. Pas d'ordonnancement. Déterminisme par construction.
  - Langages Synchrones: Lustre, industrialisé dans l'Outils SCADE de Telelogic (ex-Verilog, www.telelogic.com). Airbus, Eurocopter, …
- Problèmes:
  - Interactions avec l'extérieur Asynchrone.
  - Couplage très fort entre les modules. Peu flexible.

Développement des Systèmes embarqués Approches Synchrone / Asynchrone (4)

- L'approche synchrone et le temps
  - Synchronisation des entrées



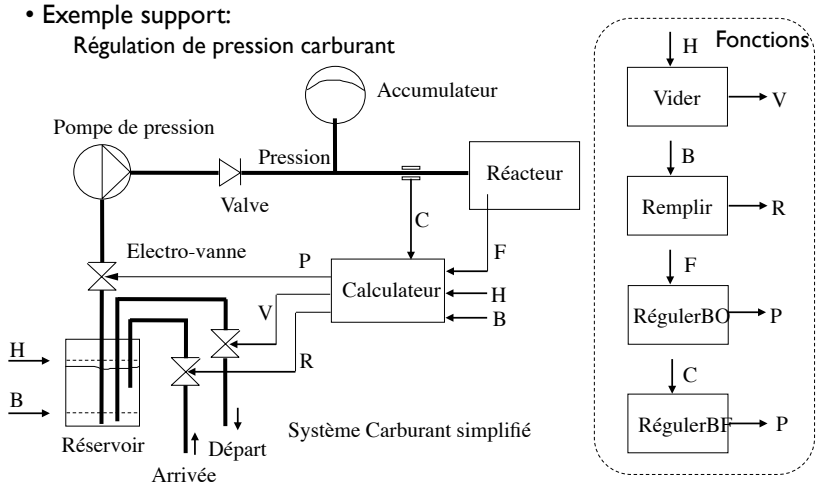
Développement des Systèmes embarqués Approches Synchrone / Asynchrone (5)

- Hypothèse Asynchrone:
  - La majorité des environnements sont asynchrones ➔ nécessité de dialogue
  - Absence de simultanéité d'événements
  - Simultanéité des traitements (tâches //)
  - Concurrence des ressources
  - Le temps de réaction du système n'est plus négligé ➔ une action possède un *début*, une *durée* de travail et une *fin*
  - Nécessité de réagir de manière dynamique aux urgences externes (gestion des modes de fonctionnement).
  - Préemption nécessaire. Qui préempte qui? ⇒ Ordonnancement.

Développement des Systèmes embarqués Approches Synchrone / Asynchrone (6)

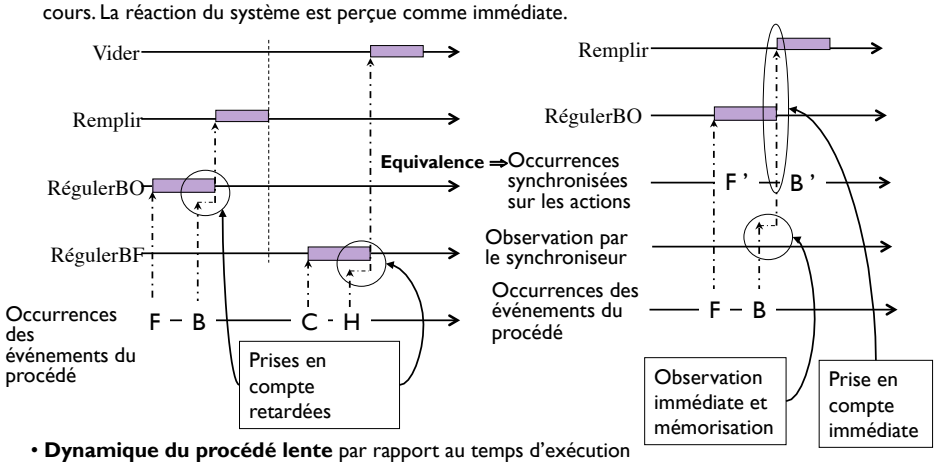
- Exemple support:

Régulation de pression carburant



Développement des Systèmes embarqués: Approches Synchrone / Asynchrone (7)

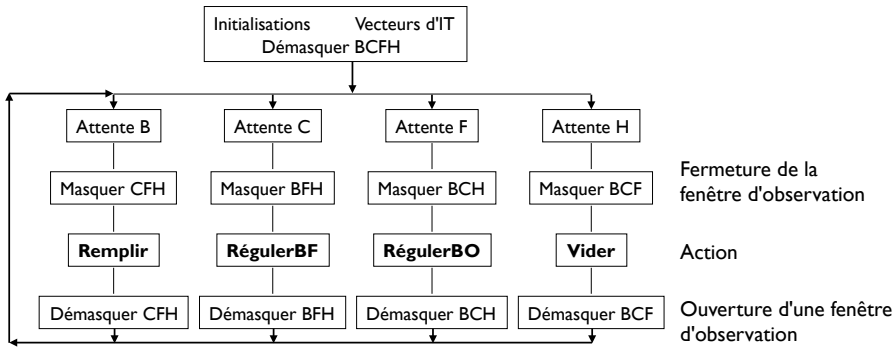
• Commande réactive **synchrone**: les entrées sont synchronisées sur la fin de l'action en cours. La réaction du système est perçue comme immédiate.



• **Dynamique du procédé lente** par rapport au temps d'exécution
• Commande **non préemptive**

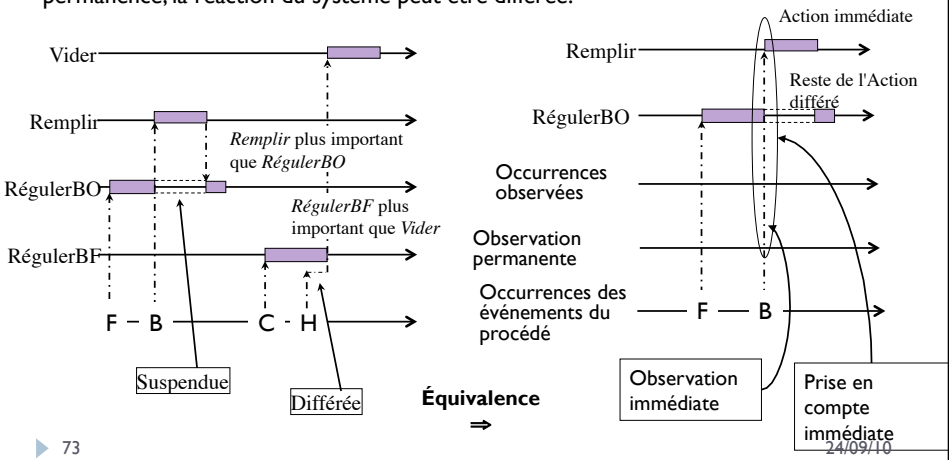
Développement des Systèmes embarqués: Approches Synchrone / Asynchrone (8)

• Réalisation d'une commande synchrone



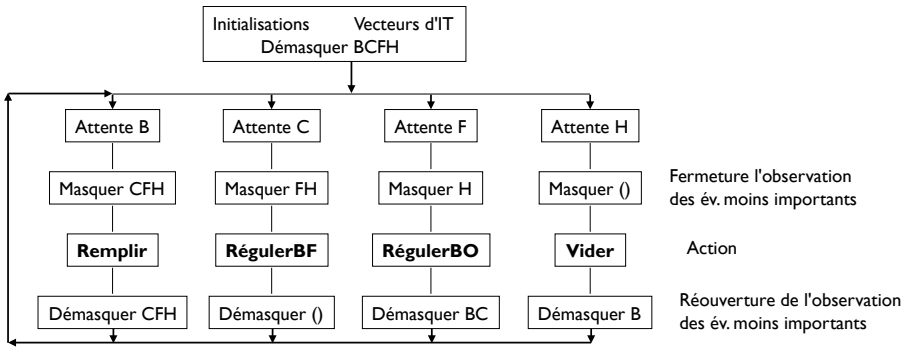
Développement des Systèmes embarqués: Approches Synchrone / Asynchrone (9)

Commande réactive **asynchrone** : les évènements du système sont observés en permanence, la réaction du système peut être différée.



Développement des Systèmes embarqués: Approches Synchrone / Asynchrone (10)

- Réalisation d'une commande asynchrone



Remplir > RégulerBF > RégulerBO > Vider

PARTIE 2 : Le développement des systèmes (embarqués)

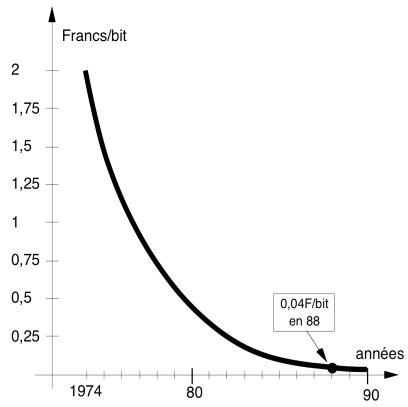
Un processus de développement Orienté Objet

Chapitre 5

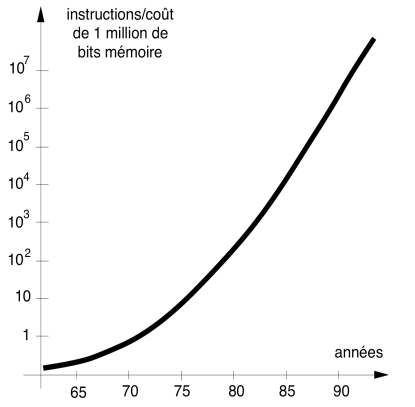
Généralités en Génie Logiciel

Contexte (1) : Technologies

Coûts mémoires

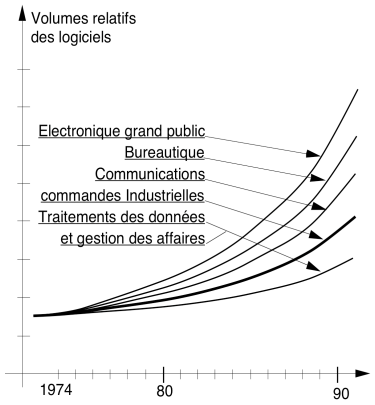


Rapidité des calculateurs/coûts mémoires

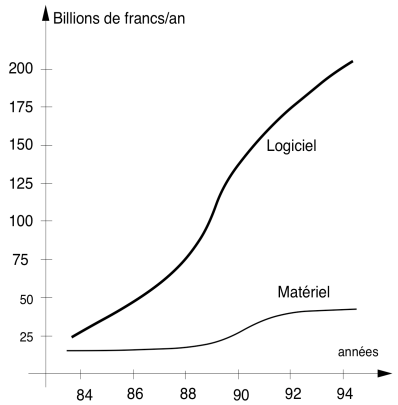


Contexte (2) : Hard contre Soft

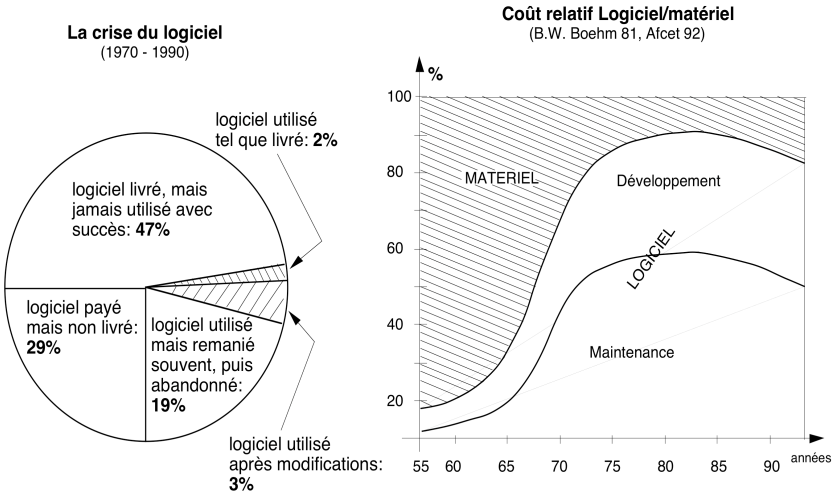
Evolutions des besoins en logiciel par secteur d'activités



Comparaison des dépenses matérielles /logicielles du Département de la défense (U.S.A.)



Contexte (3) : La "crise" du Logiciel



Nature du Logiciel

- Immatériel
  - Pas de coût de production, uniquement de développement
  - Peut être copié à l'identique, à l'infini
  - Réutilisation très avantageuse
    - (mais attention au coût d'adaptation !)
  - Pas d'usure. Les fautes sont là dès le début.
- Très sensible aux erreurs.
  - Pas de continuité du comportement
  - Impossible à tester complètement (pas d'interpolation possible)
- Pas de structure naturelle. Amorphe. Le logiciel doit être explicitement et volontairement structuré.

Qualité du Logiciel (1)

- Point de Vue de l'Utilisateur
  - Utilité
    - Validité (Exactitude du Comportement par / au Service attendu)
    - Complétude du Comportement
    - Efficacité / Performance
    - Coûts d'Utilisation
  - Ergonomie
    - Facilité de Compréhension pour l'Utilisateur
    - Qualité de la Documentation Utilisateur
    - Simplicité d'Utilisation
  - Sûreté de Fonctionnement (≈ Robustesse)
    - Disponibilité
    - Fiabilité
    - Sécurité-Innocuité
    - ...

Qualité du Logiciel (2)

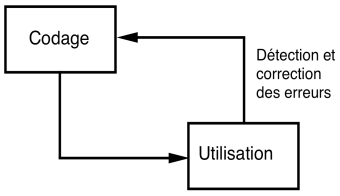
- Point de Vue du Développeur
  - Facilité de Validation
    - Complétude des spécifications
    - Localité
    - Testabilité
  - Facilité d'Adaptation (Extension, Correction)
    - Modularité / Structuration
    - Simplicité de Code (Algorithme, Structure)
    - Concision
    - Lisibilité
  - Portabilité, Compatibilité (Réutilisation)
    - Complétude
    - Indépendance vis à vis d'un plate-forme
    - Utilisation de Standards

Qualité du Logiciel (3)

- Répondre aux besoins fonctionnels / non fonctionnels
  - Fonctionnels: Directement liés à la résolution du problème pour lequel le logiciel est conçu. (Le quoi.)
  - Non Fonctionnel: Sûreté de fonctionnement (Sécurité, Confidentialité, Robustesse, …), performances (temporelles, consommation), autres contraintes de développement (ergonomie, compatibilité, etc.), etc.

Projet Logiciel (1)

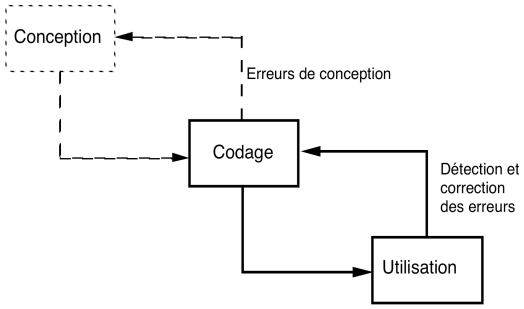
- **Notion de Projet Logiciel**



Petits Projets : Méthode descendante orientée programme

Projet Logiciel (2)

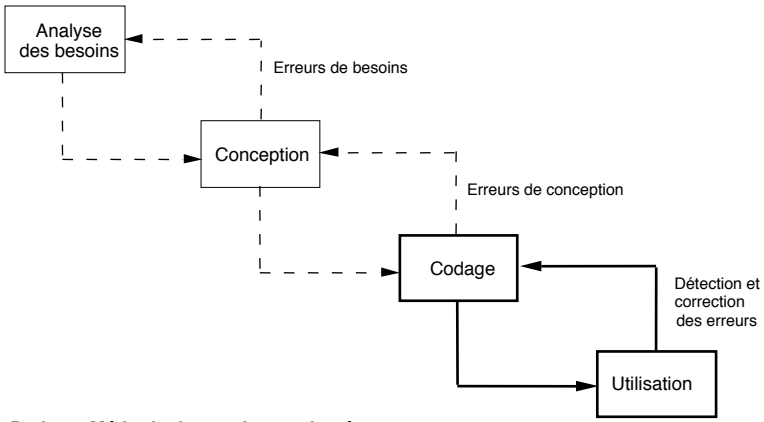
- **Notion de Projet Logiciel**



Petits Projets : Méthode descendante orientée programme

Projet Logiciel (3)

- **Notion de Projet Logiciel**



Petits Projets : Méthode descendante orientée programme

Projet Logiciel (4)

- Un projet logiciel = suite d'étapes
- Produit de l'information. Mais pas uniquement code final.
- Englobe documentation, partie très importante.
  - Documentation externe: pour l'utilisateur
  - Documentation interne:
    - Retour d'expérience, projets suivants
    - Maintenance: mémoire sur l'intérieur du logiciel
    - Mémoire de ce qui a été fait, pourquoi, comment.
  - Partie du Code non livrée, mais essentielle
    - Banc de test, jeux de test
- Chaque étape: transforme de l'information.
- Une grande partie: forme écrite = documentation.

Processus de Développement (1)

Particularités des petits et grands projets

	Petits Projets	Grands Projets
Utilisateur	L'utilisateur est le réalisateur	L'utilisateur est différent du réalisateur
Documentation utilisateur	minimale (aide mémoire)	Importante (éducative)
Validation	Minimale (optionnelle)	Importante et indispensable
Coût des modifications	Faible	Elévé
Spécification des besoins	Minimale, optionnelle (inutile?)	Nécessaire et importante
Informations sur l'état du projet	Toutes dans la tête d'une seule personne	Très répartie sur les équipes et les personnes
Erreurs courantes	de programmations : - algorithmiques, syntaxiques, logiques	- de spécification des besoins hypothèses incompatibles, incohérentes - de programmation (les mêmes)
Protection contre les erreurs d'utilisation	Optionnelle, minimale	Indispensable et importante
Interactions avec l'environnement	Peu	Nombreuses
Contrôle des versions	Informel	Nécessaire et complet

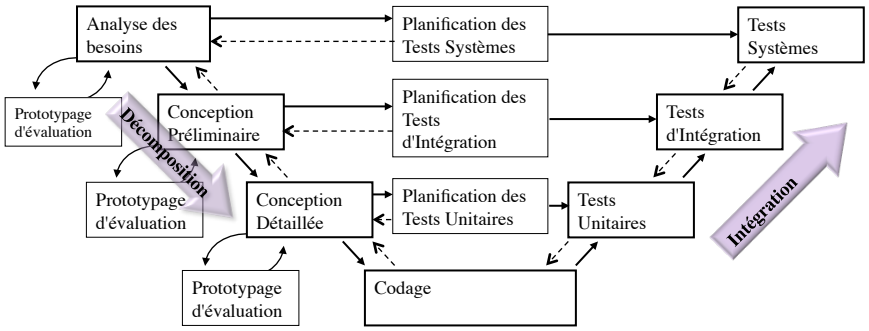
Processus de développement (2)

- Cycle de Vie d'un Logiciel
  - " **Étapes par lesquelles passe le processus de développement d'un logiciel.** "
    [H. Abrias, Dict. Encyclopédique du Génie Logiciel]

Analyse des besoins	Conception Architecturale	Conception Détaillée	Codage	Vérification et maintenance

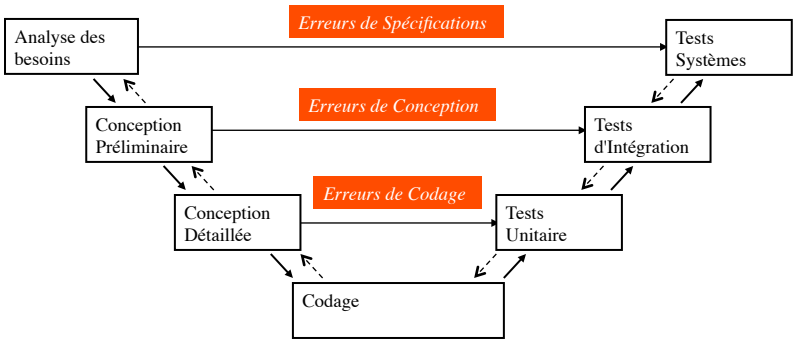
Processus de Développement (6)

- Cycle de Vie en V avec tests et prototypes
  - Le plus utilisé ces 20 dernières années



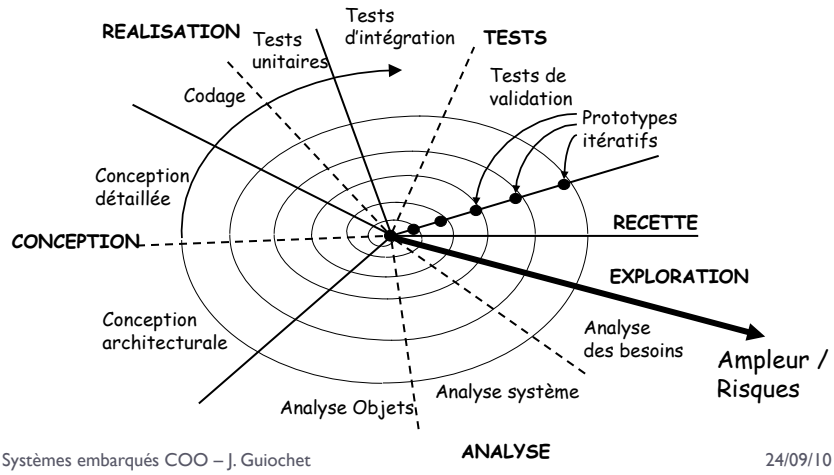
Processus de Développement (7)

- Propagation des erreurs dans le cycle de vie en V



Processus de Développement (8)

- Un Processus Itératif: Cycle de Vie en Spirale



Processus de Développement (9)
méthodes, techniques, outils

- *Analyse (Sw et Hw)* : DFD, CFD, SADT, SA-RT... OOA, UML
- *Conception* : SA-RT, SASS ... OMT, HOOD, UML, PDL, VHDL, VHDL-AMS...
- *Comportements* : Logique, MEF, StateCharts, Réseaux de Petri, ...
- *Langages* : Macro-Assembler, C, Pascal, Modula, Ada, C, C++, Java ...
- *Outils* : Select-Yourdon, Statemate, Design/IDEF, Rose, Rapsody
  Design/CPN, STOOD (HOOD), Select-OMT, Umbrello, ArgoUML
  Rapsody, Rational Rose, ...
- *Prototypage virtuel (simulation et co-simulation): Mentors (VHDL), PTOLEMY, …*

Chapitre 6

Conception orientée objet avec *l'Unified Modeling Language (UML)*

Plan

1 – D'où vient l'approche objet ?

Modélisation objet

- Problématique
  - Gestion de la complexité
  - Approches de modélisation
- Unification des méthodes objet
  - Genèse d'UML

Complexité des logiciels

- La problématique du domaine
- Le processus de développement
- L'inhérente flexibilité du logiciel

Conséquence de la complexité

- Le risque de défaillances graves est élevé
- La mise au point est lente et chaotique
- La maintenance est disproportionnée
- Le coût est astronomique
- Crise du logiciel (70)

Gestion de la complexité

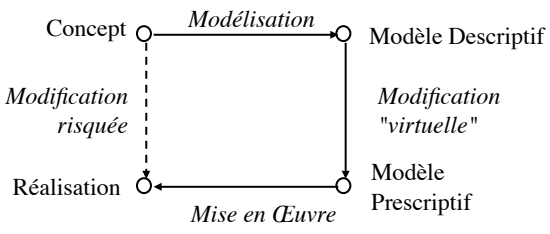
- A défaut de réduire la complexité il faut la maîtriser
  - Donner une illusion de simplicité => **modéliser**
  - Application de critères de partitionnement => **décomposer**

Notion de Modèle (1)

- L'activité de développement d'un système → besoin d'un représentation des concepts manipulés (du logiciel, de l'environnement, du matériel, …)
- Exemple de Modèles pour des personnes:
  - Photo d'identité, statistiques démographiques, bonhomme "fil de fer" …
- Notion de Modèle / de Modélisation
  - Peut représenter quelque chose (l'original du modèle) qui **existe déjà**, mais aussi quelque chose **qui n'existe pas encore**.
  - Est une **abstraction** de l'original. Seuls certains aspect, certaines propriétés de l'original sont pris en compte.
  - Sert un **objectif**, a un but. C'est uniquement par rapport à ce but que l'on peut juger de la qualité d'un modèle.

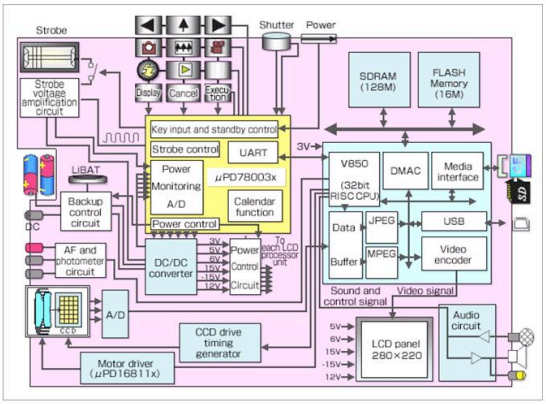
Notion de Modèle (2)

- On utilise un modèle quand:
  - La réalité est trop complexe pour pouvoir directement raisonner sur elle (simplification)
  - On veut opérer une mesure, qui dans l'absolu n'est pas possible (abstraction)
  - Une opération dans la réalité s'avère risquée, est difficilement réversible (représentation)

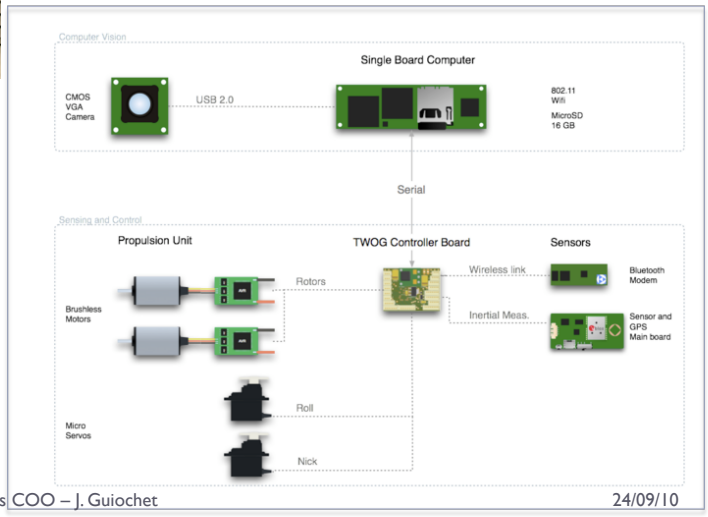


Notion de modèle

- Un modèle = une vision
- L'hégémonie du bloc et de la flèche



Notion de modèle



Notion de modèle (3)

- Lister les langages de modélisation que vous connaissez :
  - Modélisation comportementale
  - Modélisation statique/structurelle

Rôle de la décomposition

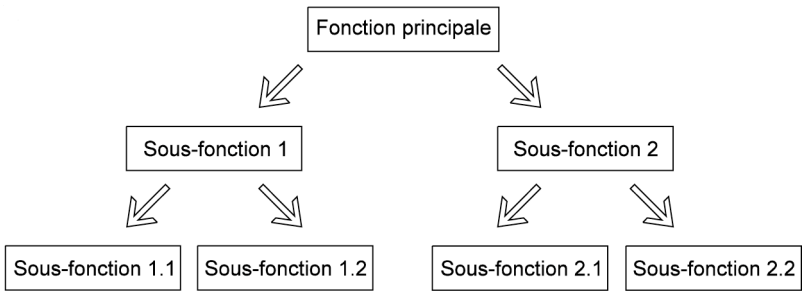
- Diviser pour régner
- Raffinage récursif jusqu'à obtention d'éléments compréhensibles
- Division intelligente de l'espace des états d'un système

Décomposition algorithmique
(aussi appelée fonctionnelle)

- Approche traditionnelle
- Chaque module représente une étape du processus global
- Décomposition fonctionnelle depuis le cahier des charges jusqu'au sous-programme

Décomposition algorithmique

- La fonction donne la forme du système

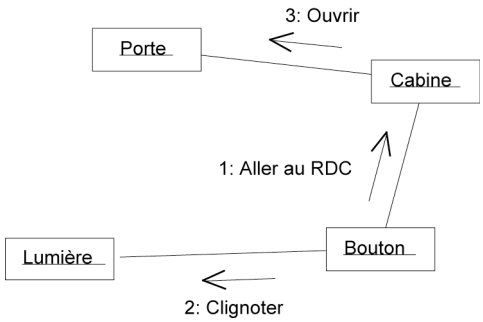


Décomposition objet

- Approche plus récente (en informatique)
- Chaque module représente un objet du domaine de l'application
- Les objets sont des entités autonomes qui collaborent afin de réaliser un projet global

Décomposition objet

- La fonction est réalisée par des objets collaborants



Décomposition / Composition

- Le terme décomposition objet est réducteur
- L'approche objet n'est pas seulement descendante
- Approche descendante, ascendante, récursive, itérative, incrémentale...

Comparaison fonctions / objets

- Les deux techniques de décomposition sont intéressantes
- Elles sont très différentes
- Il faut en choisir une pour commencer à décomposer
- Puis se servir du résultat comme cadre pour exprimer l'autre point de vue

Approche fonctionnelle

- Semble intuitive
- Mise en avant du «FAIRE»
- Bien adaptée lorsque tout est connu
- MAIS
  - Architecture rigide
  - Extension difficile
  - Peu adaptée à la découverte

Approche objet

- Mise en avant de l'«ETRE»
- Simple (petit nombre de concepts de base)
- Raisonnement par abstraction (objets du domaine)
- Adapté pour l'exploration et l'évolution
- MAIS
  - Déroutant pour les habitués de la démarche fonctionnelle

Avantages de l'objet

- Conduit à des modèles plus stables
  - Basés sur le monde réel
- Structure indépendante des fonctions
  - Evolutivité
- Encapsule la complexité
  - Facilite la réutilisation

La complexité des logiciels : conclusion

- Le logiciel est complexe par nature
- Il faut gérer cette complexité
- Les systèmes peuvent être décomposés selon ce qu'ils font ou selon ce qu'ils sont
- L'approche objet gère plus efficacement la complexité (que l'approche fonctionnelle)
  - Réutilisabilité, Evolutivité, Stabilité

Unification des méthodes objet

- Appel aux propositions de l'OMG (groupe international de standardisation)
- Démarche d'unification
- UML (*Unified Modeling Language*)

De quoi a-t-on besoin ?

- Un langage de modélisation
  - Notation claire
  - Sémantique précise
- Un processus de génie logiciel

Méthode = Langage + Processus

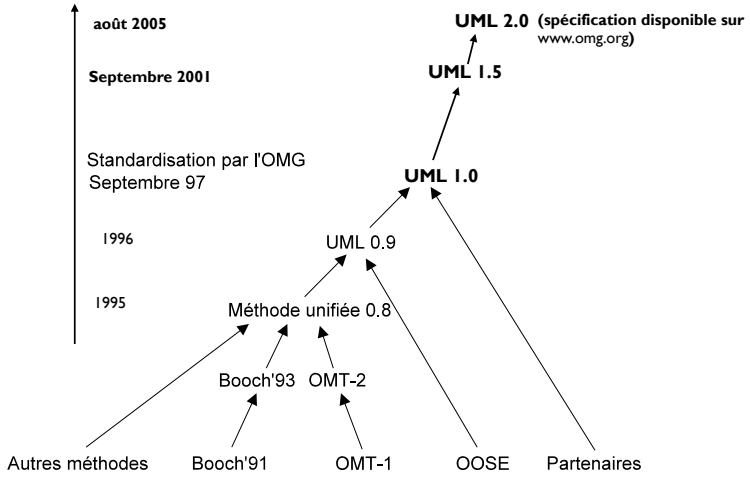
Langage de modélisation

- Générique
- Expressif
- Flexible (configurable, extensible)
- Syntaxe et sémantique
- Unification par convergence

La notation unifiée UML

- Basée sur les méthodes de BOOCH, OMT et OOSE
- Influencée par les bonnes idées des autres méthodes
- Mûrie par le travail en commun

Evolution d'UML



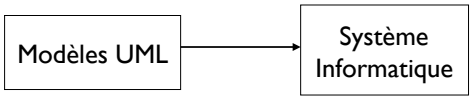
En résumé

- UML est une notation, pas une méthode
- UML est un langage de modélisation objet
- UML convient pour toutes les méthodes objet
- UML est dans le domaine public

UML est **la notation** pour documenter les modèles objets

En résumé (suite)

- À quoi sert la modélisation ? :
  - Meilleure compréhension du problème
  - Communiquer entre développeurs
  - Trouver des erreurs et des incohérences
  - Assurer la cohérence entre les besoins et l'implémentation
  - Générer du code



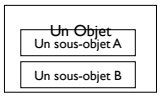
2. Les Principaux diagrammes d'UML

Les diagrammes d'UML

- *UML définit neuf sortes de diagrammes pour représenter les différents points de vue de modélisation.*
- *Un diagramme contient des attributs de placement et de rendu visuel qui ne dépendent que du point de vue.*
- *Les diagrammes peuvent montrer tout ou partie des caractéristiques des éléments de modélisation, selon le niveau de détail utile dans le contexte d'un diagramme donné*

Deux types de diagramme : structurels et dynamiques

- Un objet a une représentation structurelle (ou statique) :
  - structure interne (de quoi est il composé) et externe (ses relations structurelles avec les autres objets)



- Et dynamique :
  - Son comportement dans le temps: les messages qu'il envoie/ reçoit, ses changements d'états, etc.

Les diagrammes d'UML (UML1.5)

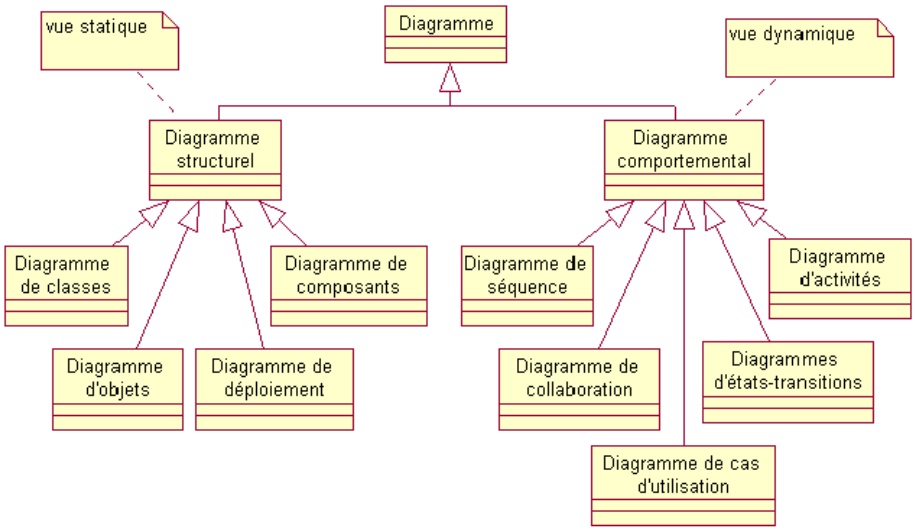


Diagramme d'objets

- *représente les objets et leurs relations (correspond à un diagramme de collaboration simplifié, sans représentation des envois de messages).*

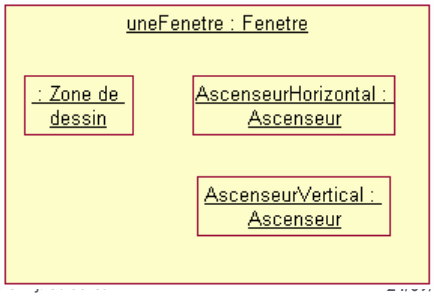


Diagramme de classes

- *Représente la structure statique en terme de classes et de relations.*

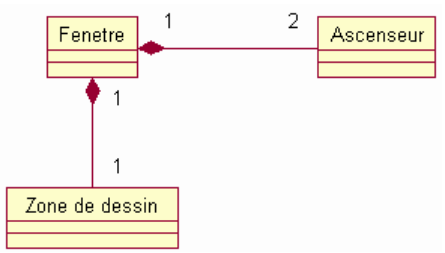


Diagramme de composants

- *Représente les composants physiques d'une application.*

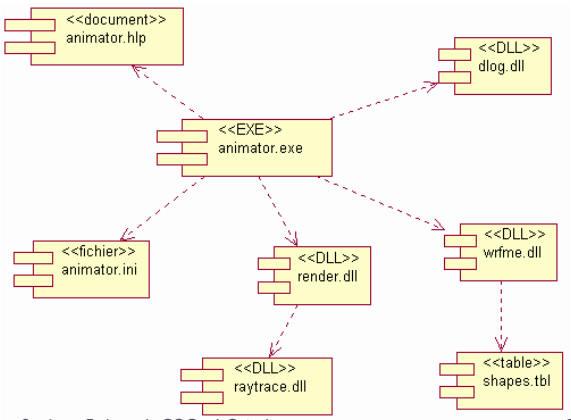


Diagramme de déploiement

- *Représente le déploiement des composants sur les dispositifs matériels.*

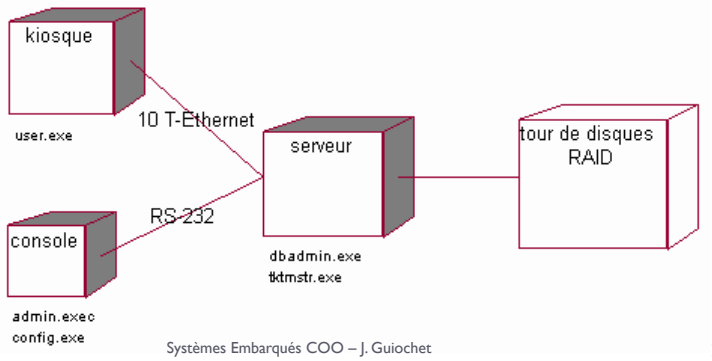


Diagramme de cas d'utilisation

- Représente les objectifs en terme de fonctionnalités du système du point de vue de l'utilisateur.

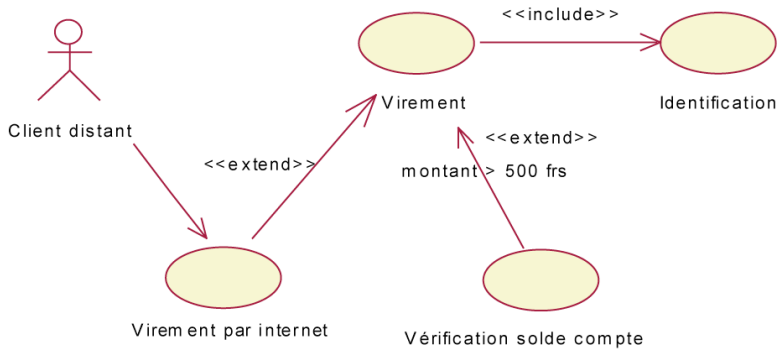


Diagramme de séquence

- *Représentation temporelle des objets et de leurs interactions.*

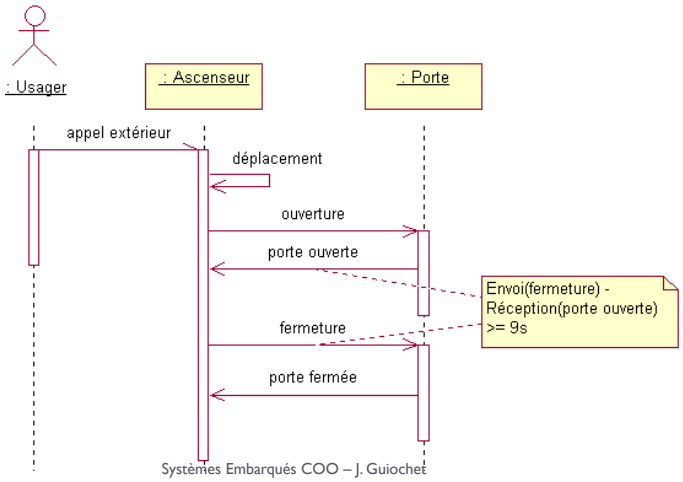


Diagramme de collaboration

- *Représentation spatiale des objets, des liens et des interactions.*

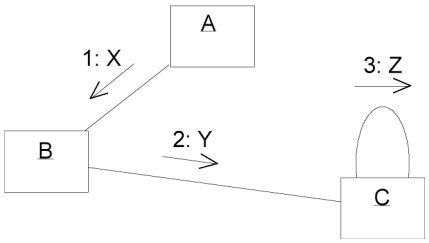


Diagramme d'états-transitions

- *Représente le cycle de vie d'un objet*

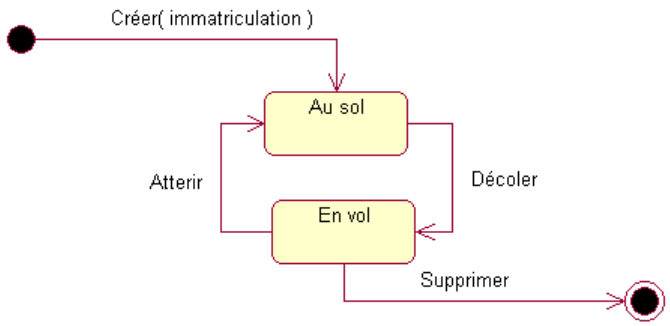
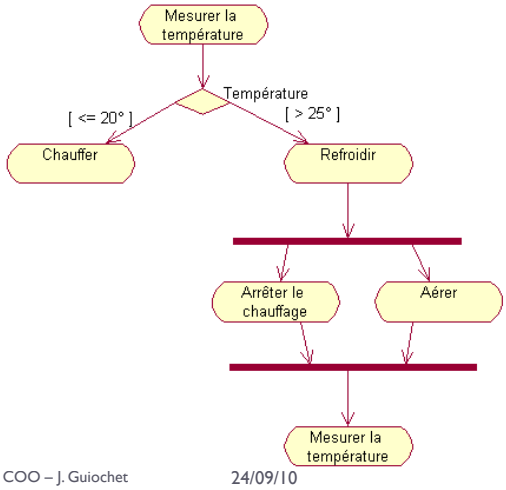


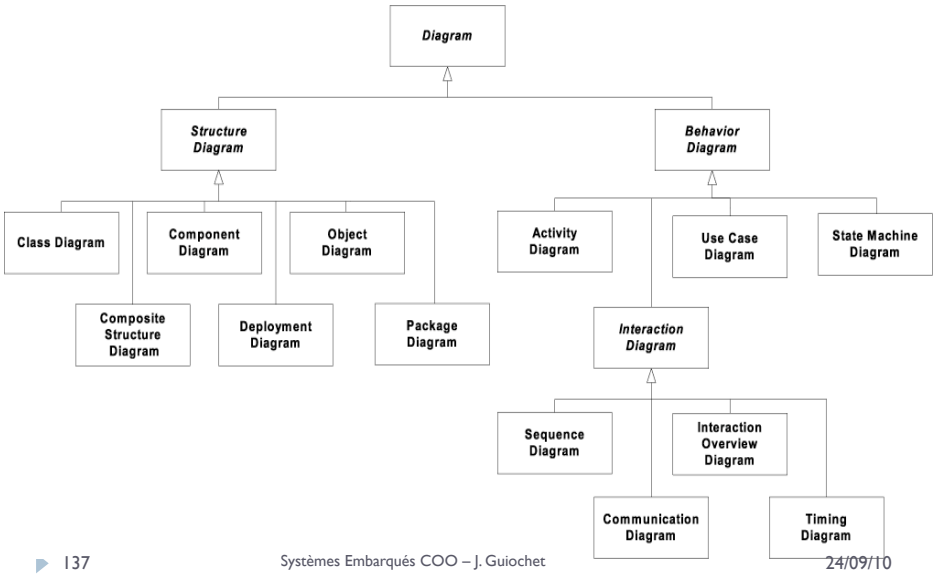
Diagramme d'activité

- *Représente un enchaînement d'activités au sein d'une opération, d'un cas d'utilisation ou d'un processus métier.*



[*PAM-00 p94*]

Diagrammes d'UML2.0

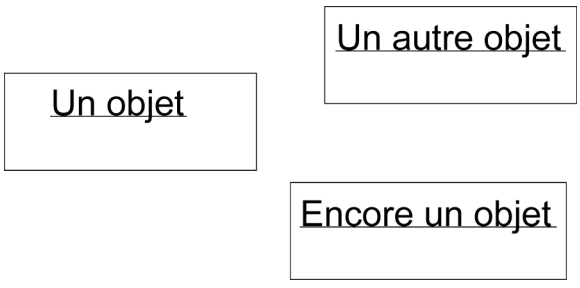


3. Objets et classes

Les objets

- Les objets du monde réel nous entourent, ils naissent, vivent et meurent
- Les objets informatiques définissent une représentation simplifiée des entités du monde réel
- Les objets représentent des entités concrètes (avec une masse) ou abstraites (concept)

Représentation graphique des objets



Les objets sont des abstractions

- Une abstraction est un résumé, un condensé
- Mise en avant des caractéristiques essentielles
- Dissimulation des détails
- Une abstraction se définit par rapport à un point de vue

Exemples d'abstractions

- Une carte routière
- Un nombre complexe
- Un téléviseur
- Une transaction bancaire
- Une porte logique
- Une pile
- Un actionneur
- Un capteur

Caractéristiques fondamentales des objets

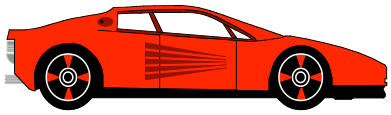
- L'état
- Le comportement
- L'identité

L'état

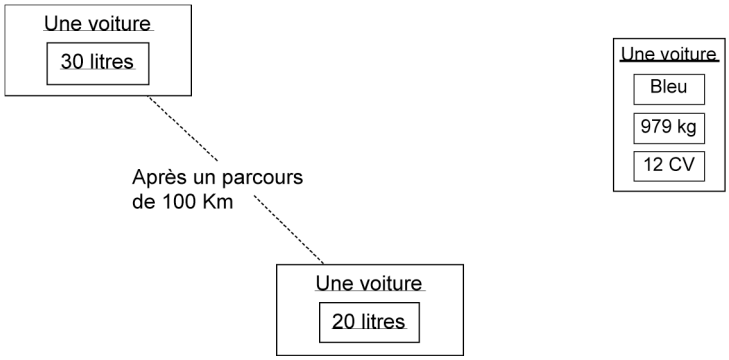
- L'état regroupe les valeurs instantanées de tous les attributs d'un objet
- L'état évolue au cours du temps
- L'état d'un objet à un instant donné est la conséquence de ses comportements passés

Exemples

- Pour un signal électrique : l'amplitude, la pulsation, la phase, ...
- Pour une voiture : la marque, la puissance, la couleur, le nombre de places assises, ...



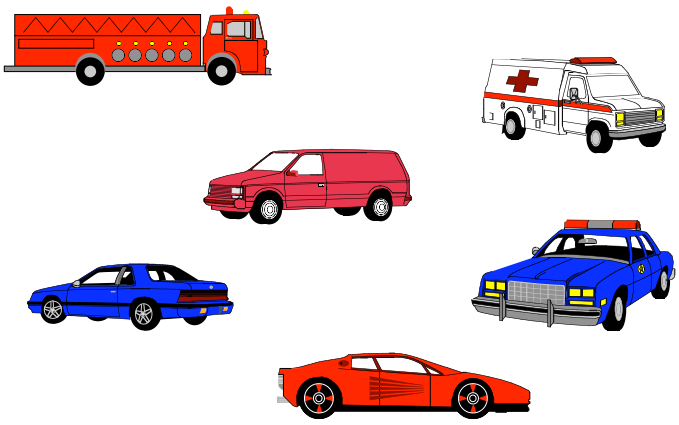
Exemples (suite)



Le chaos des objets

- Le monde qui nous entoure est constitué de très nombreux objets
- Pour comprendre le monde, l'être humain a tendance à regrouper les éléments qui se ressemblent
- Regrouper des objets suivants des critères de ressemblance s'appelle classer
- Les humains ont classé les animaux, les plantes, les champignons, les atomes, ...

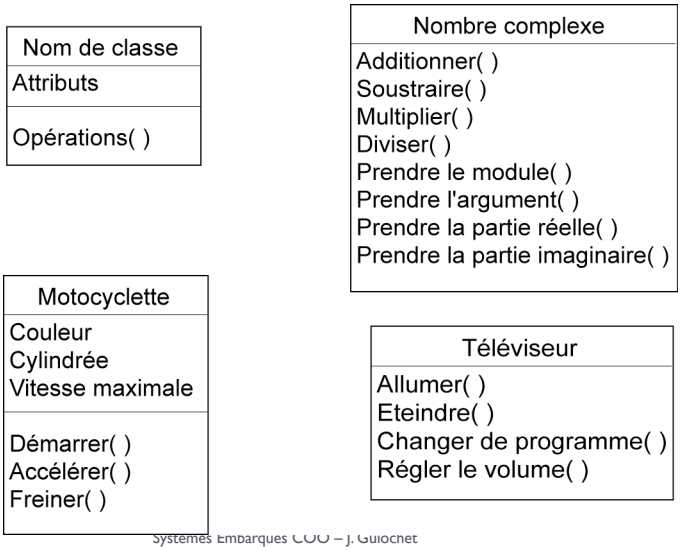
Le chaos des objets (suite)



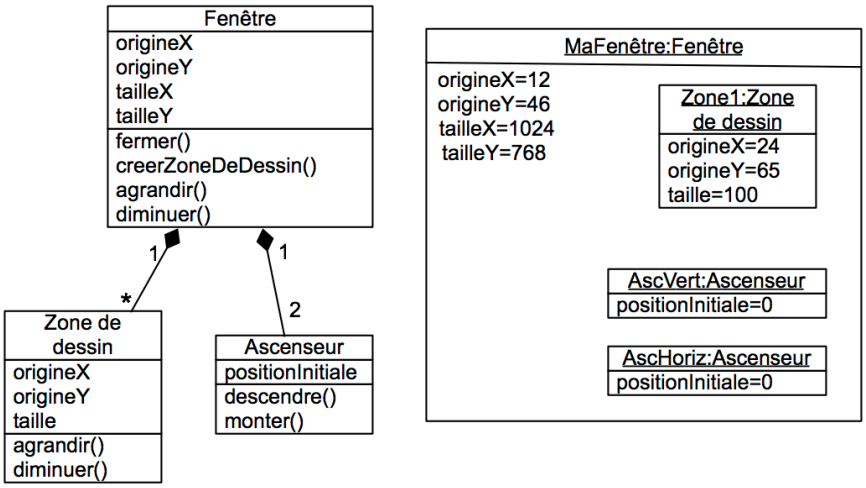
Les classes

- La classe est une description abstraite d'un ensemble d'objets
- La classe peut être vue comme la factorisation des éléments communs à un ensemble d'objets
- La classe décrit le domaine de définition d'un ensemble d'objets

Représentation graphique des classes



Classes et objets

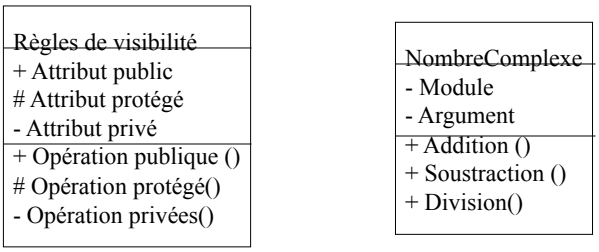


L'encapsulation

- Chaque objet encapsule des données
- Le genre des données encapsulées et les opérations applicables à un objet sont décrites dans la classe
- Les classes permettent de décrire des contrats qui seront valables pour tous les objets issus de ces classes

Encapsulation (Suite)

- La spécification d'une classe définit la partie visible des objets, le reste est caché dans la réalisation



Bénéfices de l'encapsulation

- L'encapsulation présente deux avantages
  - les données encapsulées sont protégées des accès intempestifs, ce qui permet de garantir leur intégrité
  - les clients d'une abstraction ne dépendent pas de la réalisation de l'abstraction, mais seulement de sa spécification

Conclusion sur les objets et les classes

- Les objets naissent, vivent et meurent
- Les objets interagissent entre eux
- Les objets sont regroupés dans des classes qui les décrivent de manière abstraite
- La classe intègre les concepts de type et de module

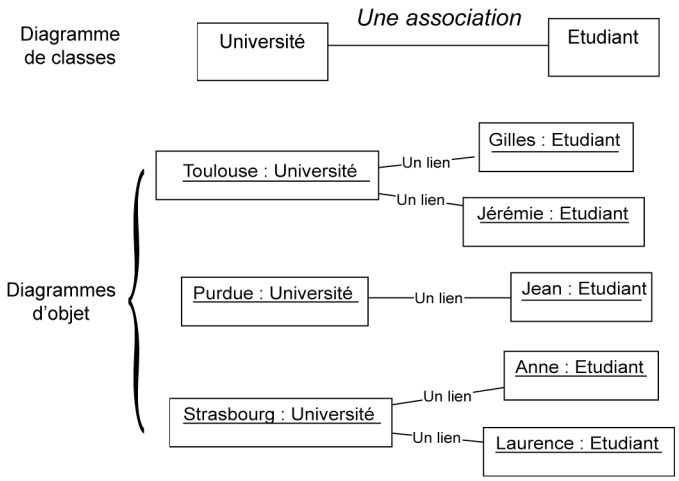
Les relations entre classes

- L'association
- L'agrégation
- La composition
- La généralisation

L'association

- L'association exprime une connexion sémantique bidirectionnelle entre classes
- Une association est une abstraction des liens qui existent entre les objets instances des classes associées
- Les associations se représentent de la même manière que les liens.
  - Distinction opérée en fonction du contexte

Exemple

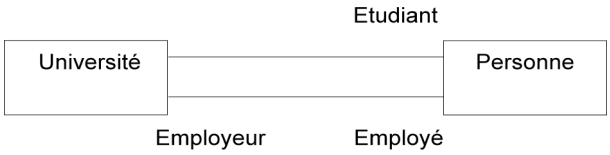


Nommage des associations

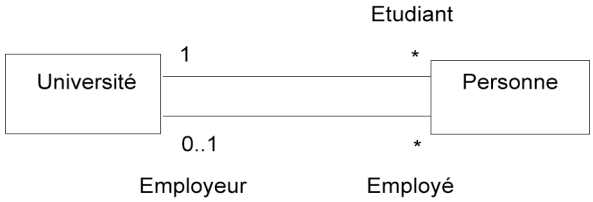
- Indication du sens de lecture

Nommage des rôles

- Le rôle décrit une extrémité d'une association



Multiplicité des rôles

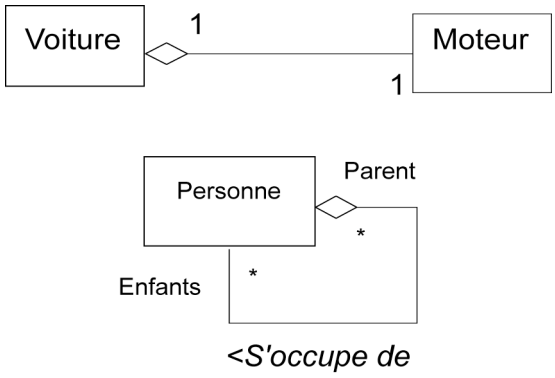


1	Un et un seul
0..1	Zéro ou un
M .. N	De M à N (entiers naturels)
*	Plusieurs
0 .. *	De zéro à plusieurs
1 .. *	D'un à plusieurs

L'agrégation

- Connexions bidirectionnelles dissymétriques
- L'agrégation EST une association mais qui exprime un couplage plus fort entre classes
- Représentation des relations
  - maître et esclaves
  - tout et parties
  - composé et composant.

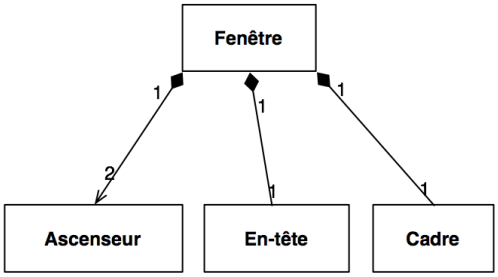
Exemples



La composition

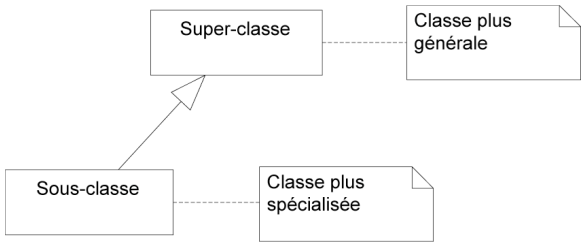
- Connexions bidirectionnelles dissymétriques
- La composition EST une association exprimant un couplage plus fort entre les classes que l'agrégation
- Représentation des relations
  - tout et parties
  - composé et composant.
- Destruction du composé entraîne destruction composant

Exemple



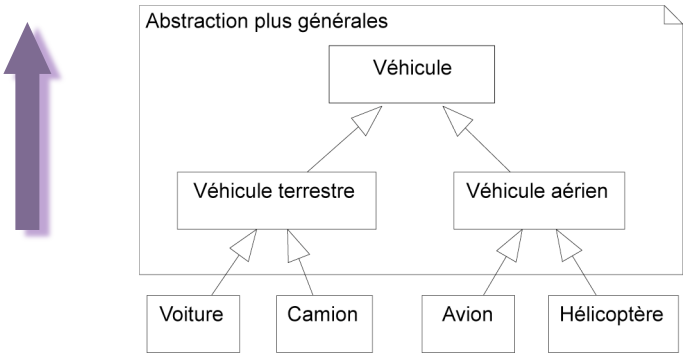
Hiérarchies de classes

- Gérer la complexité
  - Arborescences de classes d'abstraction croissante
- Généralisation
  - Super-classes
- Spécialisation
  - Sous-classes



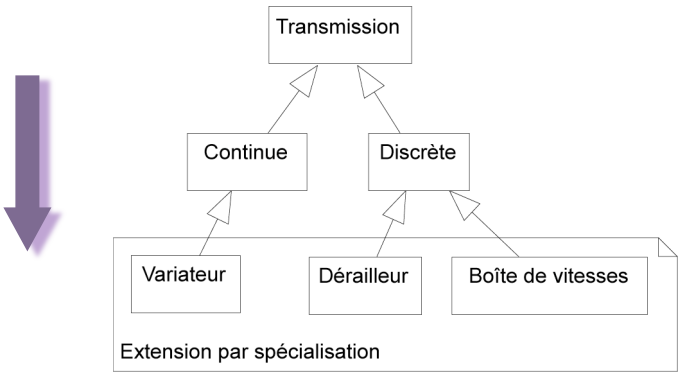
Généralisation

- Factoriser les éléments communs
  - attributs, opérations et contraintes



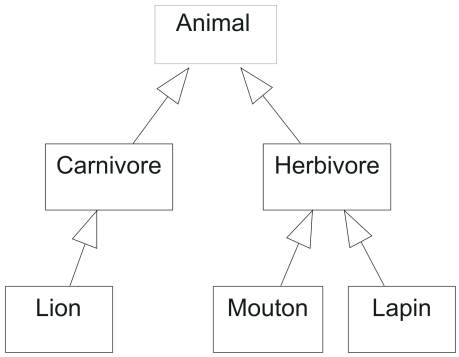
Spécialisation

- Extension cohérente d'un ensemble de classes



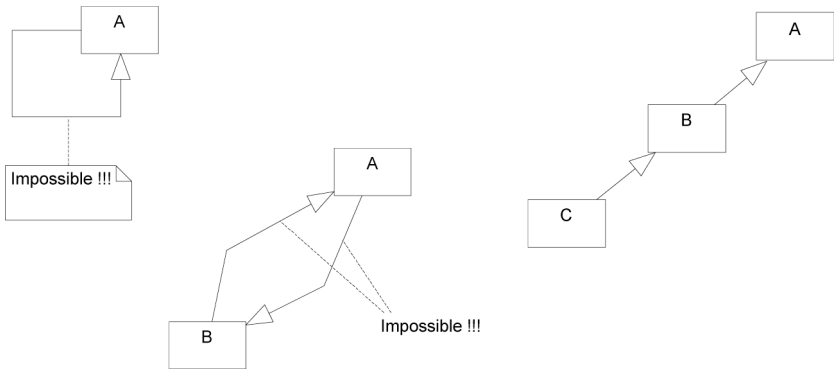
Propriétés de la généralisation

- Signifie toujours : *est un* ou *est une sorte de*

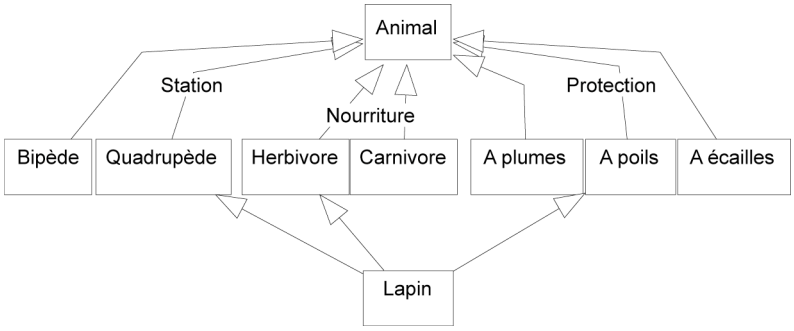


Propriétés de la généralisation

- Non-réflexive, non-symétrique, transitive



Généralisation multiple



L'héritage

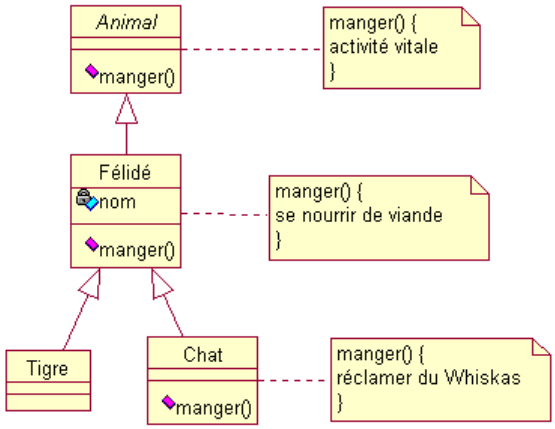
- Technique la plus utilisée pour réaliser la généralisation
- Construire une classe à partir d'une ou plusieurs autres classes, en partageant des attributs, des opérations et parfois des contraintes, au sein d'une hiérarchie de classes.

Héritage et polymorphisme

Polymorphisme :

• Une même méthode peut avoir plusieurs implémentations (voir rédéfinition et surcharge en java)

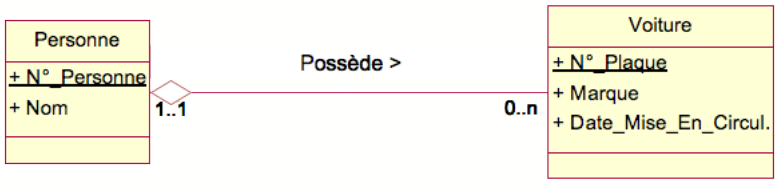
• Un objet de type Chat peut également être vu comme un *Félilé* ou comme un *Animal*



Exercice 1:

- construire le diagramme de classe relatif à l'énoncé suivant
  - Soient un ensemble de personnes et un ensemble de voitures. Une personne est caractérisée par un numéro qui l'identifie, son nom et par les voitures dont elle est l'unique propriétaire. Une voiture est caractérisée par un numéro de plaque, une marque et une date de mise en circulation.

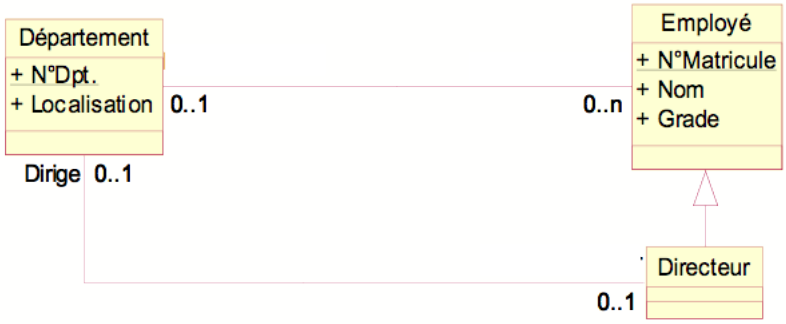
Exercice 1: solution possible



Exercice 2 :

- construire le diagramme de classe relatif à l'énoncé suivant
  - Les différents départements d'une entreprise occupent des employés. Un employé est décrit par son numéro matricule (unique dans l'entreprise), son nom, son grade et le département dans lequel il travaille. Un département est décrit par son numéro dans l'entreprise et sa localisation. Un département est dirigé par un directeur qui doit être un de ses employés.

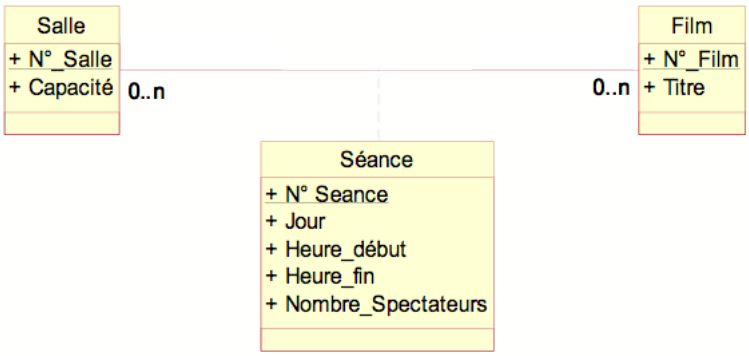
Exercice 2 : solution possible



Exercice 3 :

- construire le diagramme de classe relatif à l'énoncé suivant
  - Un exploitant possède plusieurs salles de cinéma. Un film fait généralement l'objet de plusieurs séances par jour. Décrire un schéma qui permette à l'exploitant d'obtenir des renseignements sur le chiffre d'affaire d'un film, d'une salle, d'une séance où d'un jour déterminé.

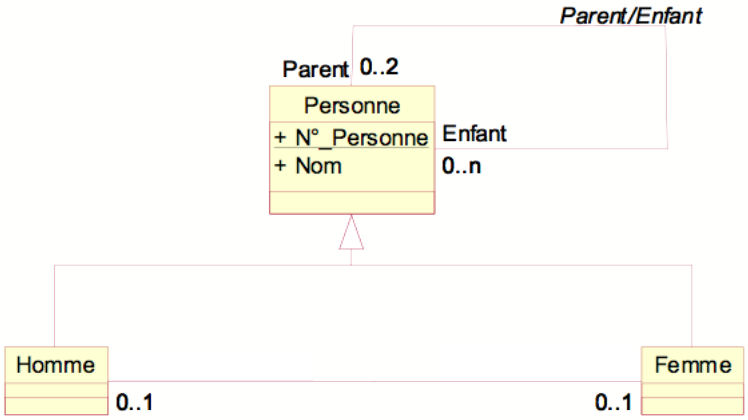
Exercice 3 - solution possible : la classe d'association



Exercice 4:

- construire le diagramme de classe relatif à l'énoncé suivant
  - Définir un schéma décrivant les liens familiaux (mariage, parent/enfant) d'une population de personnes identifiables par un numéro.

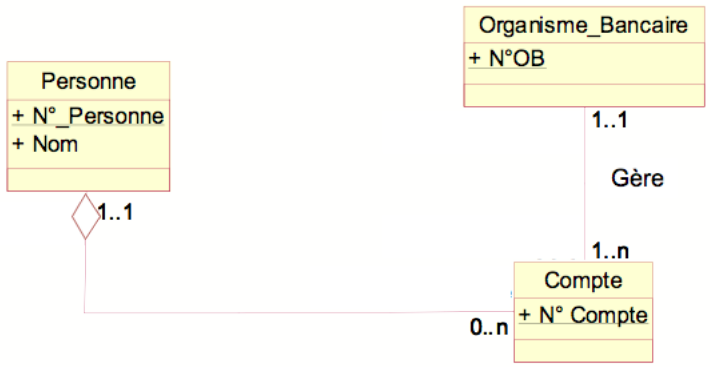
Exercice 4: solution possible



Exercice 5 :

- construire le diagramme de classe relatif à l'énoncé suivant
  - Soit un ensemble de personnes (identifiées par un numéro et caractérisées par un nom) et un ensemble d'organisme bancaires (identifiés par un numéro); une personne peut ouvrir un ou plusieurs comptes dans un organisme bancaire; chaque organisme bancaire affecte à chacun de ses comptes un numéro identifiant pour lui seul.

Exercice 5 : solution possible



Exercice 6 : MonAuto

Une réparation est toujours relative à un véhicule. La facture est envoyée au propriétaire (qui est toujours un client) du véhicule ou à une compagnie d'assurance en cas d'accident; une compagnie d'assurance est un client pour le garage. En cas de réparation en garantie, aucune facture n'est envoyée.

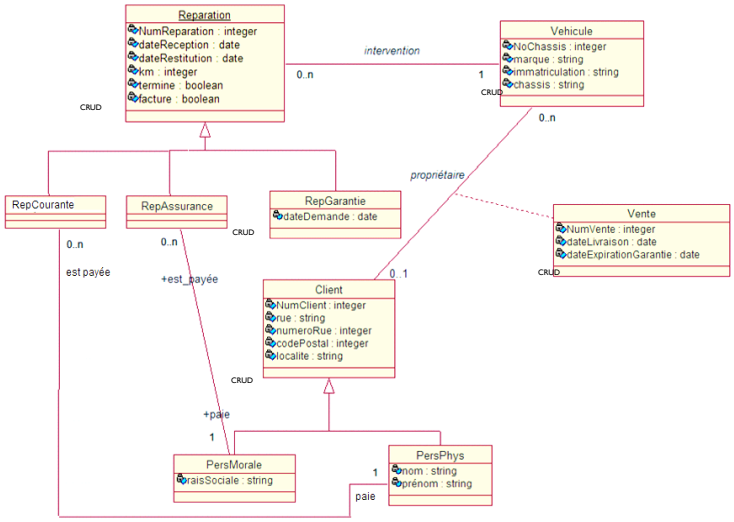
Le modèle doit contenir les renseignements qui permettent de faire la facture, selon les règles suivantes :

- Un véhicule vendu par MonAuto bénéficie d'une année de garantie à partir de la date de livraison.
  Pour bénéficier d'une réparation sous garantie, le client doit amener son véhicule à l'atelier avant l'expiration du délai de garantie. En fin de période de garantie, l'atelier peut être surchargé et le Chef d'atelier ne pourra pas toujours effectuer la réparation avant la date d'expiration. Pour résoudre ce dilemme et éviter toute réclamation, lorsqu'un client prend un rendez-vous pour effectuer une réparation en garantie le Chef d'atelier prépare une fiche de réparation "garantie" et y indique la date de la demande de rendez-vous du client, en plus des 2 dates de réception et restitution du véhicule pour la réparation; cette date de demande de rendez-vous sera utilisée comme critère de réparation en garantie.

Quelques précisions :

Nous ne gérons pas l'historique des changements de propriétaires des voitures; chaque fois qu'une voiture change de propriétaire, un nouveau véhicule sera créé avec indication de la nouvelle immatriculation, du nouveau propriétaire et de la date de livraison s'il s'agit d'une vente de MonAuto.

Exercice 6 : MonAuto



Conclusion

- Les classes sont connectées par des relations
- L'association exprime une connexion sémantique
- L'agrégation est une forme d'association plus forte
- La généralisation permet d'ordonner les objets au sein de hiérarchies de classes

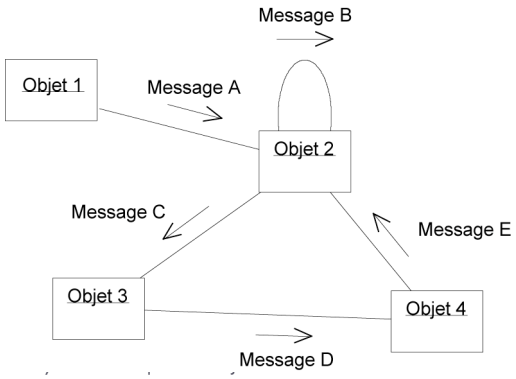
4. La dynamique des objets

Communication entre objets

- Application = société d'objets collaborants
- Les objets travaillent en synergie afin de réaliser les fonctions de l'application
- Le comportement global d'une application repose donc sur la communication entre les objets qui la composent

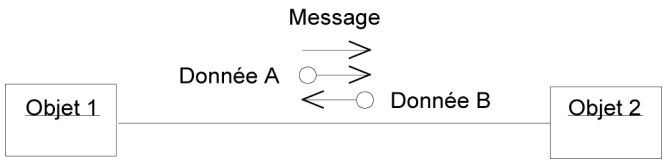
Les objets ne vivent pas en ermites

- Les objets interagissent les uns avec les autres



Communication entre objets

- Les objets communiquent en échangeant des messages



Le concept de messages

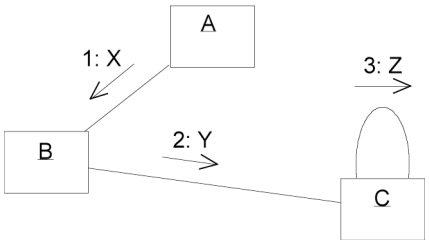
- L'unité de communication entre objets
- Concept très général pouvant être mis en oeuvre suivant de nombreuses variantes
- Regroupe les flots de contrôle et les flots de données
- Représente également les événements

Les diagrammes de collaboration

- Des objets dans une situation donnée
- Des liens relient les objets qui se connaissent
- Les messages échangés par les objets sont représentés le long de ces liens
- L'ordre d'envoi des messages est matérialisé par un numéro de séquence

Exemple

- *Un objet **A** envoie un message **X** à un objet **B**, puis l'objet **B** envoie un message **Y** à un objet **C**, et enfin **C** s'envoie un message **Z**.*

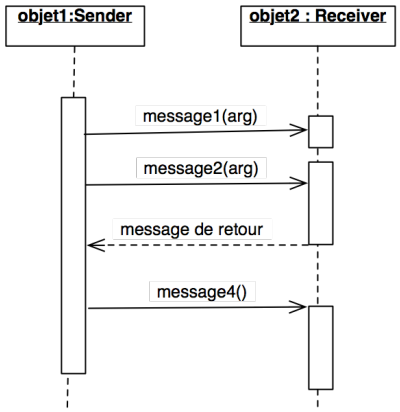


Les diagrammes de séquence

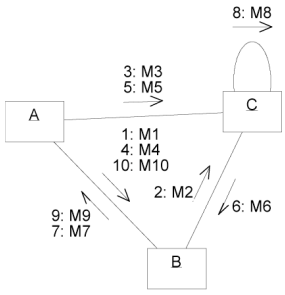
- L'accent est mis sur la communication, au détriment de la structure spatiale
- Chaque objet est représenté par une barre verticale
- Le temps s'écoule de haut en bas, de sorte que la numérotation des messages est optionnelle.

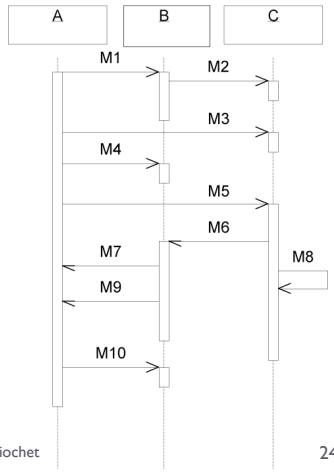
Objets et messages

- Des lignes verticales pointillées représentent des **objets** (pas des classes!)
- Le nom de l'objet (ex : objet1) est optionel (on peut noter juste :Sender).
- ⟶ représente un message
- – -> représente un retour explicite de message (return)
- La classe Sender devrait avoir une association avec la classe Receiver dans le diagramme de classes

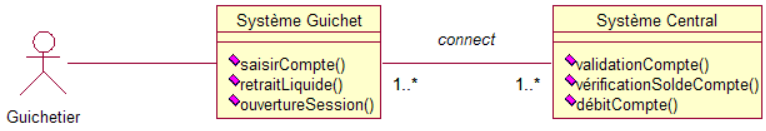


Comparaison entre diagramme de séquence et de collaboration





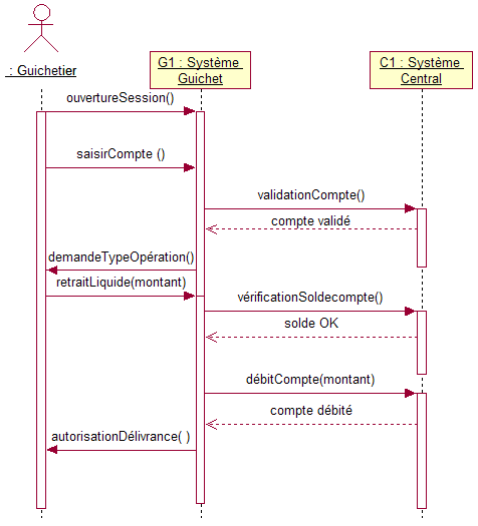
Exemple : Retrait en espèce



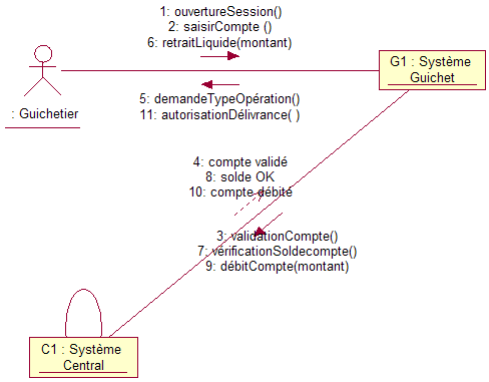
Rédigez un diagramme de séquence basé sur l'énoncé suivant:

- Le guichetier ouvre une session
- Le guichetier saisit le numéro de compte du client.
- Le système guichet valide le compte auprès du système central.
- Le système guichet demande le type d'opération au guichetier
- Le guichetier sélectionne le montant du retrait
- Le système guichet interroge le système central pour s'assurer que le compte est suffisamment approvisionné
- Le système guichet demande au système central de débiter le compte
- Le système notifie au guichetier qu'il peut délivrer le montant demandé

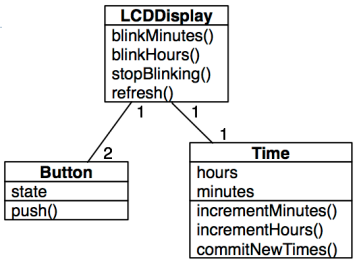
Retraite en espèce – Diagramme de Séquence

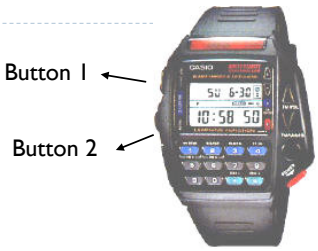


Retrait d'espèce – Diagramme de Collaboration



Exercice : Simple Watch





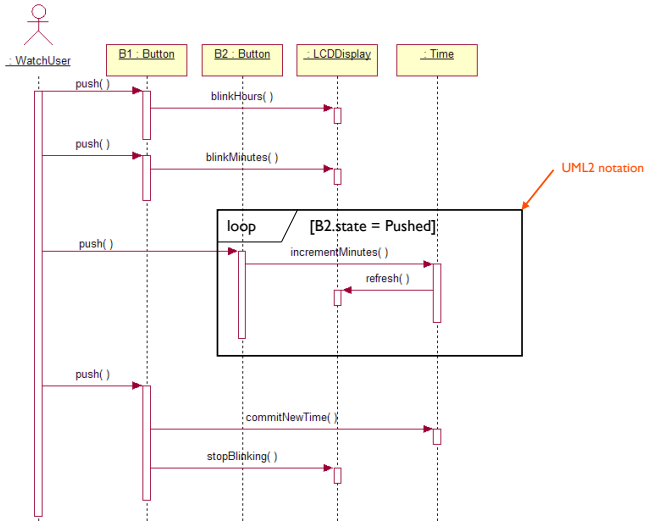
A partir du diagramme de classe ci-dessus

1. Rédigez un diagramme de séquence pour modéliser un scénario où un utilisateur voudrait régler l'heure (particulièrement les minutes) sur sa montre.

En appuyant 2X sur le bouton 1 il accède au réglage des minutes (heure clignote puis minute clignote). Ensuite avec le bouton 2 (sans relâcher le bouton) il incrémente les minutes, le LCD display est rafraîchi. En appuyant sur le bouton 1 un autre fois l'heure est enregistrée et l'affichage s'arrête de clignoter.

2. Rédigez un diagramme de collaboration à partir du diagramme de séquence obtenu

Simple watch: Diagramme de Séquence



Simple Watch: Diagramme de Collaboration

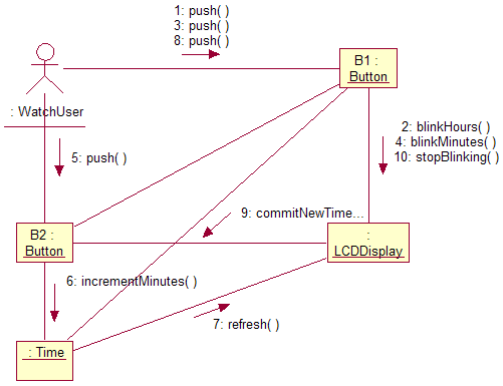


Diagramme d'états-Transitions

- Un Diagramme d'états-transitions est utilisé pour montrer
  - le cycle de vie d'un objet instance d'une classe,
  - les événements qui provoquent une transition d'un état à un autre,
  - les actions qui résultent d'un changement d'état

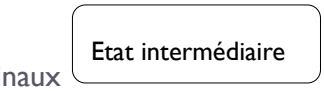
États

- L'état d'un objet est l'une des conditions possibles dans lequel un objet peut exister:
- Un objet est toujours dans un état donné, pour un certain temps
- Un état = {valeurs des attributs de l'objet} + liens vers d'autres objets
- Un état possède un nom qui l'identifie

États (suite)

- Caractéristiques :
  - un état initial

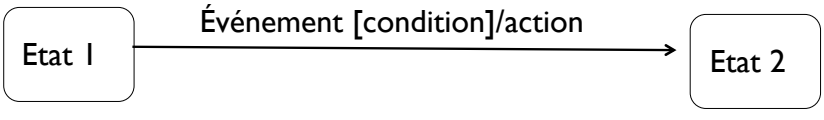
  - des états intermédiaires



  - des états finaux

Transitions

- Connexions unidirectionnelles reliant des états, formant un graphe dirigé
- Traduisent le passage d'un état à un autre état (instantané *car pas d'état inconnu*)
- Sont déclenchées par un **événement** si la **condition** est vraie, et engendrent une **action**



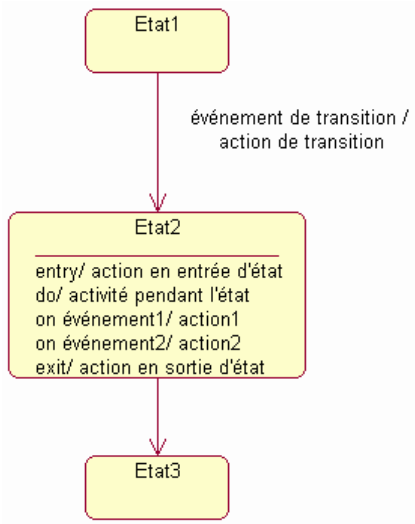
Événements, condition et actions

- Les événements :
  - éléments déclencheurs de la transition
  - Généralement 4 types :
    - Signaux : une occurrence asynchrone d'un événement externe à la machine à états (Ex : Bouton STOP enfoncé)
    - Appel : une notification explicite d'un autre objet
    - Changement : la modification d'un attribut peut être vu comme un événement (NiveauEssence=0)
    - Temporel : délai expiré, ou arrivée d'un temps absolu (ex: delai=30ms)
- Les conditions : condition booléenne validant ou non le déclenchement d'une transition à l'arrivée d'un événement

Les actions

- correspondent généralement à des opérations déclarées dans la classe
- opérations attachées à une transition
- instantanées et ne pouvant être interrompues
- ont accès aux paramètres de l'événement et aux attributs de l'objet
- les états peuvent aussi contenir des actions pouvant être exécutée en entrée ou en sortie de l'état ou lors de l'arrivée d'un événement alors que l'objet est dans l'état

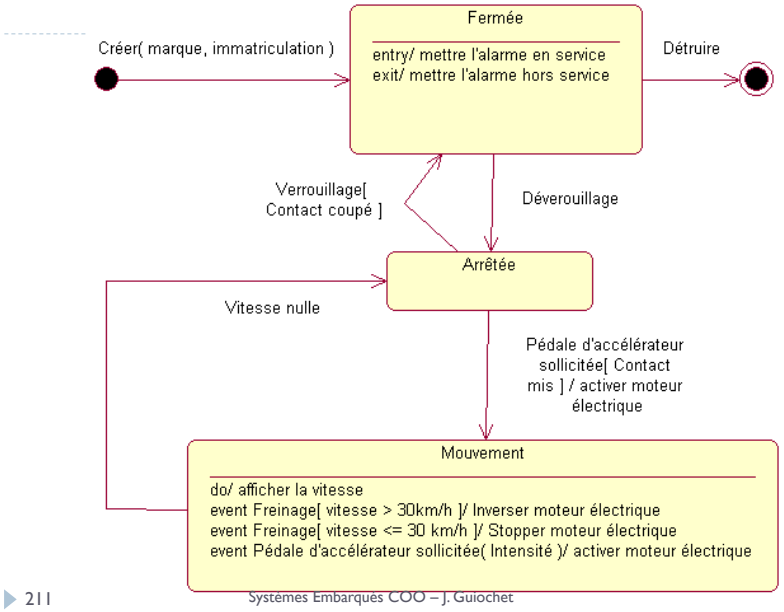
Les actions et les activités



Les activités

- opération attachée à un état
- a une certaine durée
- peut être interrompue, dès qu'une transition de sortie de l'état est déclenchée
- l'arrêt de certaines est soumis au déclenchement d'une transition (cycliques), d'autres s'arrêtent d'elles mêmes (séquentielles), démarrant à l'entrée dans l'état
- L'arrêt d'une activité séquentielle peut être suivi d'une transition automatique

Exercice 3 - 1



Les états composites

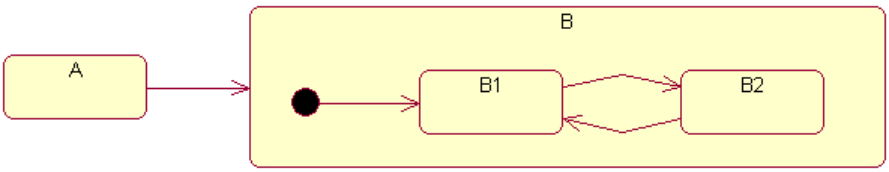
Un état composite (ou état englobant) est décomposé en sous-états.

- États disjoints
- Etats concurrents

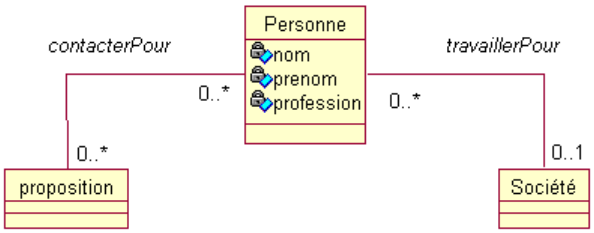
Cette approche d'abstraction procède de la même démarche que la décomposition hiérarchique; elle facilite la représentation et permet d'occulter les détails selon le niveau hiérarchique choisi.

Les états disjoints

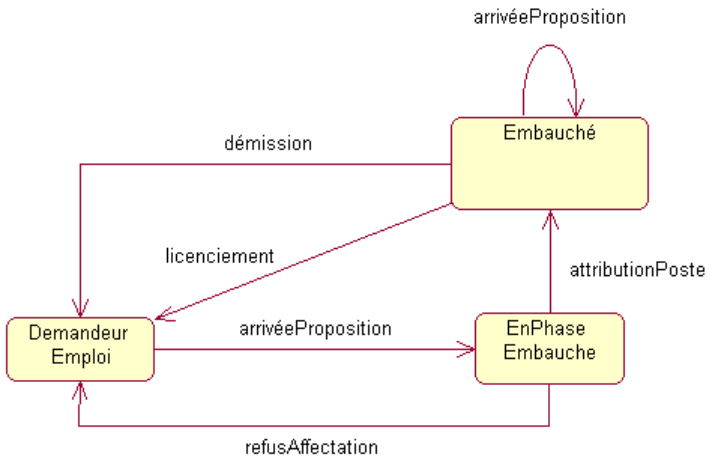
Un état peut se décomposer en plusieurs sous-états disjoints; les sous-états héritent des caractéristiques de leur état composite, en particulier des transitions externes. La décomposition en sous-états est également appelée décomposition disjonctive (décomposition de type ou-exclusif) car l'objet doit être dans un et un seul sous-état à la fois.



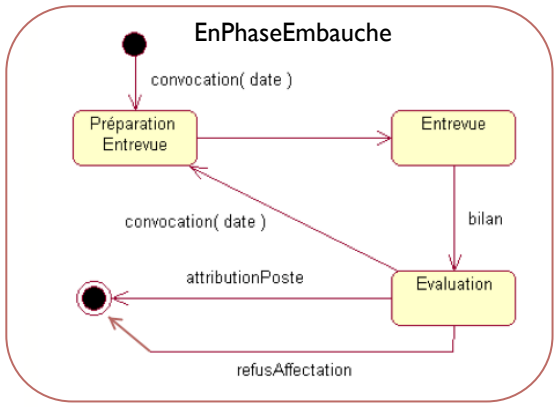
Exemple



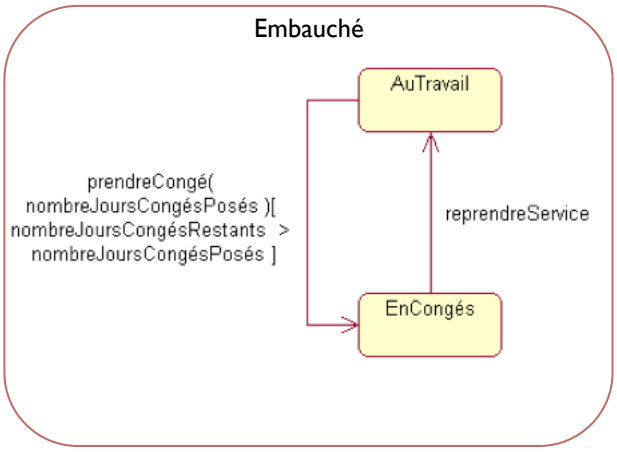
Exemple (suite)
Diagramme d'état-transitions de la classe Personne



Sous-états de EnPhaseEmbauche



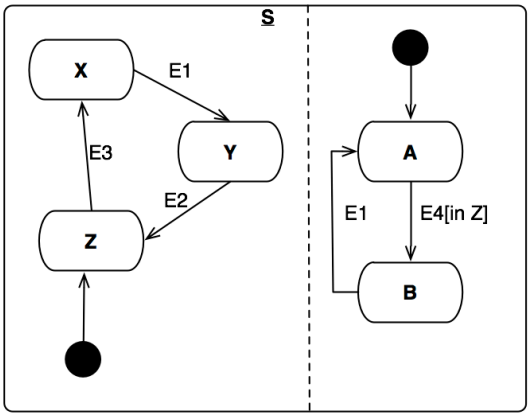
Sous-états de « Embauché »



Les états concurrents

Un état peut être composé de plusieurs sous-états concurrents. Les sous-états concurrents sont alors appelés régions. Cette composition est de type conjonctive (composition de type et) ce qui implique que l'objet doit être simultanément dans tous les états composant l'agrégation d'états. La conjonction d'états représente une forme de parallélisme entre états.

Exemple



Evénement	Etat	Synchronisation
Start	Z,A	Indépendance
E3	X,A	Simultanéité
E1	Y,B	Indépendance
E2	Z,B	Dépendance
E4	Z,A	Indédendance

La communication entre objets

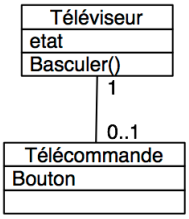
Les envois de messages entre deux objets sont visualisés de manière abstraite dans le formalisme des diagrammes d'états-transitions par l'envoi d'un événement entre les automates d'états-finis des classes d'objets concernés.

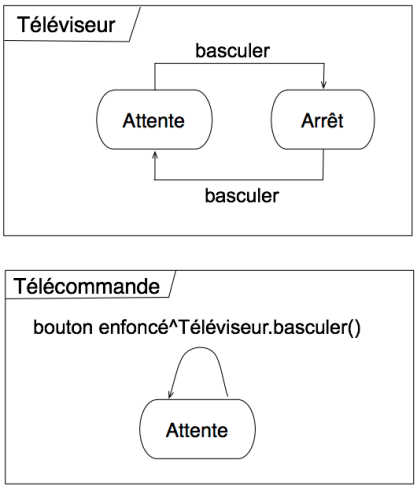
La syntaxe d'un envoi d'événement vers une classe est:

^Cible.Evénement (Arguments)

où la cible désigne la classe des objets destinataires de l'événement.

Exemple

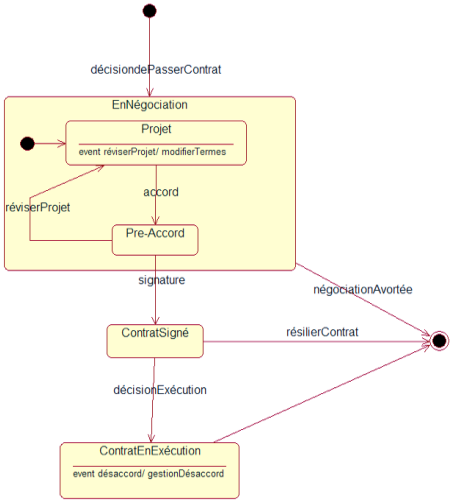




Exercice 1 : formation d'un contrat

Dessinez un diagramme d'état/transition résumant les états possibles d'un objet "contrat" tel que décrit dans l'énoncé suivant.
Un ensemble de personnes décident d'établir un contrat. Pour ce faire elles rédigent un projet par itération successive. Le contrat est ensuite informellement accepté par les parties, et devient ce que l'on appelle un pré-accord. A ce stade il peut toujours être l'objet de modification et revenir à l'état de projet. Une fois le pré-accord définitivement établi, le contrat est signé par les parties. Dès ce moment les partenaires sont liés. Une fois signé le contrat peut être rendu exécutoire par une décision d'une des parties. Un contrat en exécution peut faire l'objet de discussions qui sont réglées par un arbitre désigné à cet effet. Le contrat une fois exécuté prend fin.

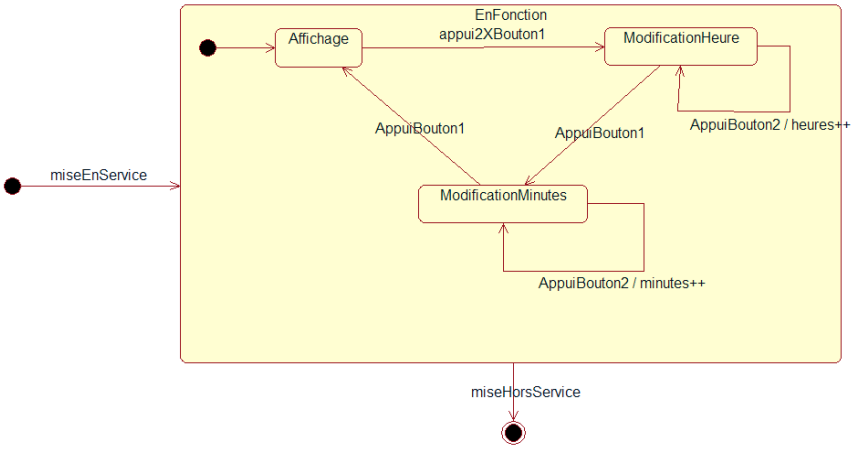
Proposition Solution 1



Exercice 2 : montre digitale

- Ma montre affiche l'heure, si j'appuie 2X sur le boutton 1, la montre passe en mode "modification". Chaque pression sur le boutton 2, incrémente l'heure d'une unité. Si j'appuie encore un fois sur le boutton 1, je peux régler les minutes de la même façon que les heures. Si j'appuie une quatrième fois sur le boutton 1, la montre affiche à nouveau l'heure courante.

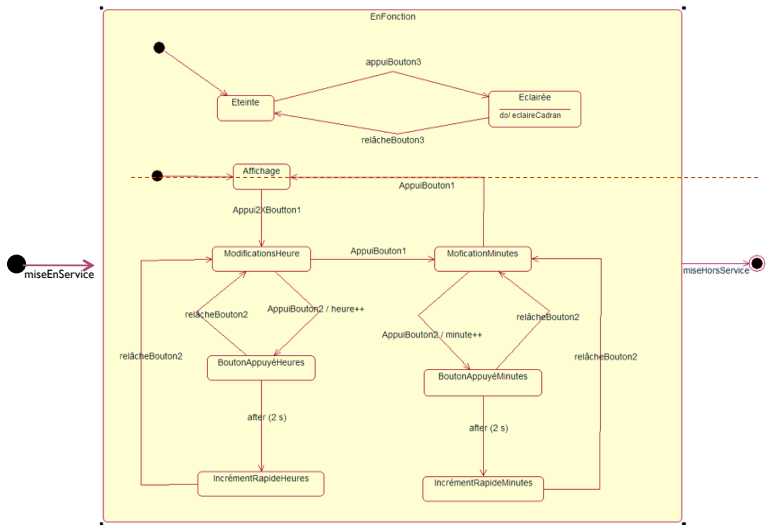
Proposition Solution 2



Exercice 3 : montre digitale plus avancée

- Lors du réglage de l'heure ou des minutes lorsque j'appuie sur le bouton 2 plus de deux secondes, les heures ou les minutes avancent très rapidement jusqu'à ce que je relâche la pression
- On ajoute un bouton 3 qui permet de rétro-éclairer l'écran LCD

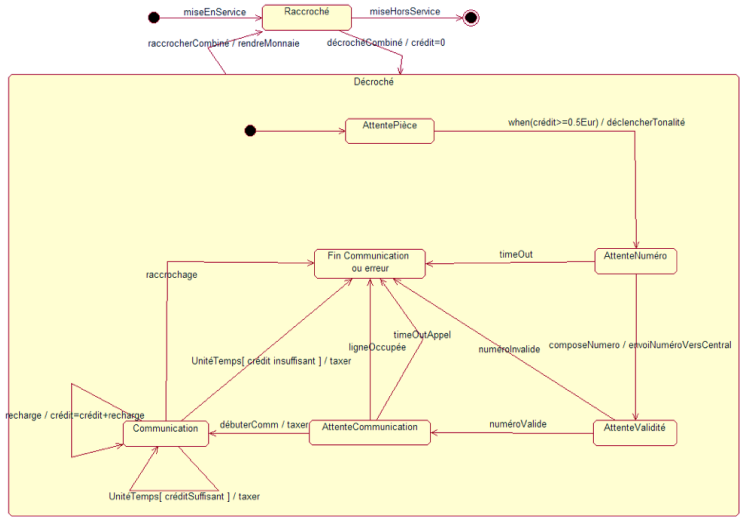
Proposition Solution 3



Exercice 4 : cabine téléphonique

- Modélisez le fonctionnement d'une cabine téléphonique

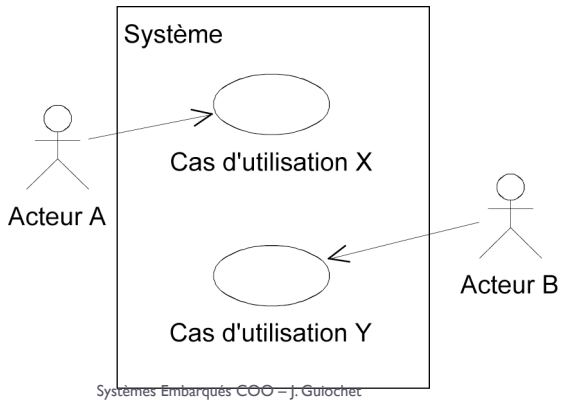
Proposition de Solution 4



5. Les Cas d'utilisation

Les cas d'utilisation

- Expression des besoins



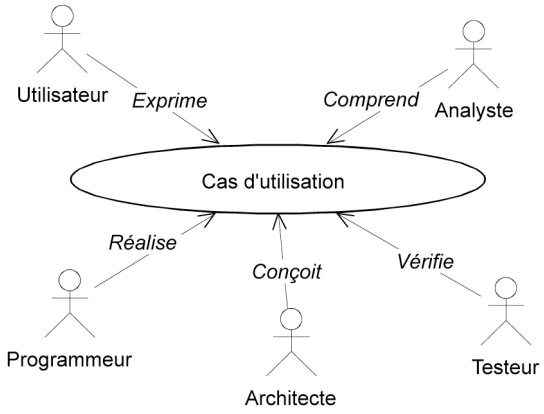
Les cas d'utilisations

- Formalisés par Ivar Jacobson : Object-Oriented Software Engineering (Addison-Wesley, 1992)
- Expression du comportement du système (actions et de réactions), selon le point de vue de l'utilisateur
- Décrivent le système et les relations entre le système et l'environnement

Intérêt des cas d'utilisations

- Constituent un moyen de déterminer les besoins d'un système
- Utilisés par les utilisateurs finaux pour exprimer leur attentes et leur besoins
- Permettent d'impliquer les utilisateurs dès les premiers stades du dévelopement
- Constituent une base pour les tests fonctionnels

Fil conducteur du projet



Les acteurs

- Un acteur est une personne ou un système qui interagit avec un système, en échangeant de l'information (en entrée et en sortie)
- On trouve les acteurs en observant les utilisateurs directs du système, ceux qui sont responsable pour sa maintenance, ainsi que les autres systèmes qui interagissent avec le système

Les utilisateurs

- Un acteur représente un rôle joué par un utilisateur qui interagit avec le système
- La même personne physique peut jouer le rôle de plusieurs acteurs (vendeur, client)
- D'autre part, plusieurs personnes peuvent également jouer le même rôle, et donc agir comme le même acteur (tous les clients)

Acteurs et cas d'utilisations

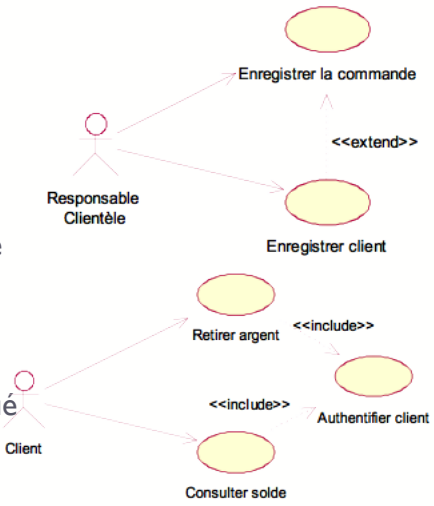
- Les cas d'utilisations représentent le dialogue entre l'acteur et le système de manière abstraite
- Ensemble de scénarios au sein d'une description unique
- Les cas d'utilisations doivent être vus comme des classes de scénarios

Détermination des cas d'utilisations

- Quelles sont les tâches de l'acteur ?
- Quelles informations l'acteur doit-il créer, sauvegarder, modifier, détruire ou simplement lire ?
- L'acteur devra-t-il informer le système de changements externes ?
- Le système devra-t-il informer l'acteur de conditions internes au système ?

Structurer les cas d'utilisation

- Les relations "extend" et "include"
  - "Extend" : un cas d'utilisation X étend un cas d'utilisation Y lorsque le cas d'utilisation X peut être appelé au cours de l'exécution du cas d'utilisation Y
  - "Include" : un cas d'utilisation est constitué de "sous" cas d'utilisation



Exemple de canevas pour une description textuelle

Cas d'utilisation		
Acteurs		
Point de vue		
Préconditions		
Déroulement normal	Description détaillée	
	Postconditions	
Déroulement alternatif	Exceptions	
	Variantes	

Exercice 1 : Le cas MonAuto

MonAuto est une entreprise qui fait le commerce, l'entretien et les réparations de voitures.

MonAuto désire exploiter un logiciel de gestion des réparations; elle dispose déjà d'un logiciel comptable. Les factures de réparations seront imprimées et gérées par le logiciel comptable.

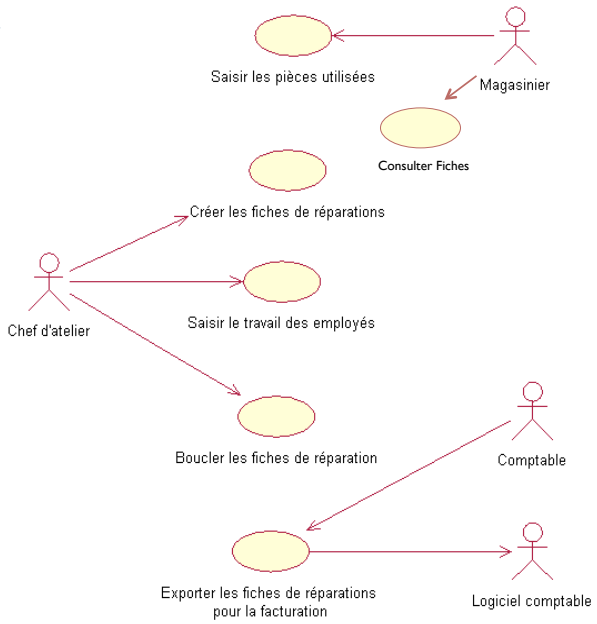
Le logiciel de gestion des réparations devra communiquer avec le logiciel comptable pour lui transmettre les réparations à facturer.

Exercice 1 : Le cas MonAuto (suite)

Le logiciel de gestion des réparations est destiné en priorité au chef d'atelier, il devra lui permettre de saisir les fiches de réparations et le travail effectué par les divers employés de l'atelier.
Pour effectuer leur travail, les mécaniciens et autres employés de l'atelier vont chercher des pièces de rechange au magasin. Lorsque le logiciel sera installé, les magasiniers ne fourniront des pièces que pour les véhicules pour lesquels une fiche de réparation est ouverte; ils saisiront directement les pièces fournies depuis un terminal installé au magasin.

Lorsqu'une réparation est terminée, le chef d'atelier va essayer la voiture. Si tout est en ordre, il met la voiture sur le parc clientèle et bouclera la fiche de réparation informatisée. Les fiches de réparations bouclées par le chef d'atelier devront pouvoir être importées par le comptable dans le logiciel comptable.

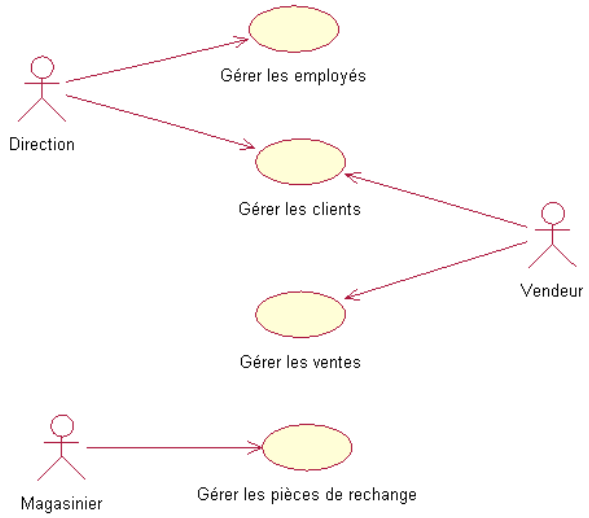
Solution possible



Exercice 2: Le cas MonAuto

- **Découvrez les besoins implicites**
- Nous n'avons pas explicité la manière dont les employés et leur qualification sont gérés, tout comme pour le stock de pièces de rechange, les clients, les employés, les ventes de voitures...
  Nous pouvons imaginer que le logiciel de gestion des réparations offre les fonctionnalités implicites de gestion des clients, employés, ventes de voitures, pièces de rechange...

Solution

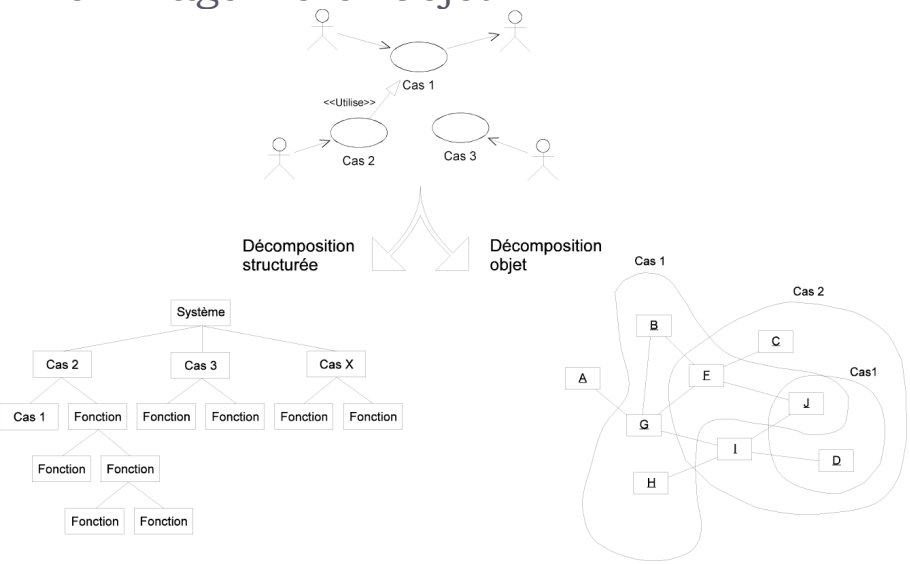


Acteurs et objets

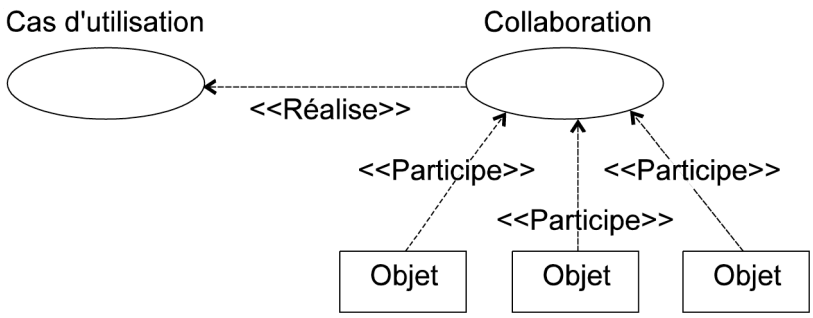
- Les acteurs sont des objets particuliers du système MAIS :
  - Ils n'apparaîtront pas forcément en tant qu'objet dans les diagrammes de classes (à quoi servirait un objet de type « Direction » ?, voir transp. précédent)
  - Ils n'y apparaissent dans le diagramme de classes que s'ils sont à la fois acteur et ressource du système (exemple : la classe Étudiant dans un système d'enseignement à distance)

- 6 -
Un Processus de développement ultraléger piloté par les cas d'utilisation

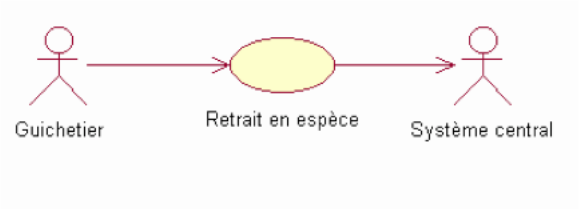
Le «virage» vers l'objet



Le «virage» vers l'objet



Exemple / Le cas d'utilisation « retrait en espèce »

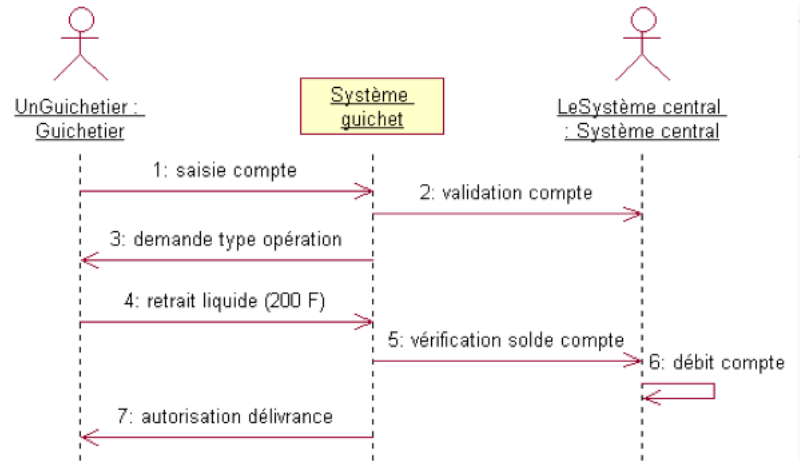


Exemple / représentation textuelle

Cas d'utilisation		Retrait en espèce
Acteurs		Guichetier (principal) et Système central (secondaire)
Point de vue		Analyste système
Préconditions		Le client est inscrit et possède un numéro de compte
Déroulement normal	Description détaillée	1. Le guichetier saisit le numéro de compte du client. 2. L'application valide le compte auprès du système central. 3. L'application demande le type d'opération au guichetier. 4. Le guichetier sélectionne un retrait de 200 euros. 5. Le système « guichet » interroge le système central pour s'assurer que le compte est suffisamment approvisionné. 6. Le système central effectue le débit du compte. 7. Le système notifie au guichetier qu'il peut délivrer le montant demandé.
	Postconditions	Une fois le montant débité, le compte doit présenter le solde précédent moins la somme retirée
Déroulement alternatif	Exceptions	a) Le numéro de compte client n'est pas valide : recommencer en 1 b) Approvisionnement insuffisant : message d'erreur, aller en 4 ou abandon
	Variantes	

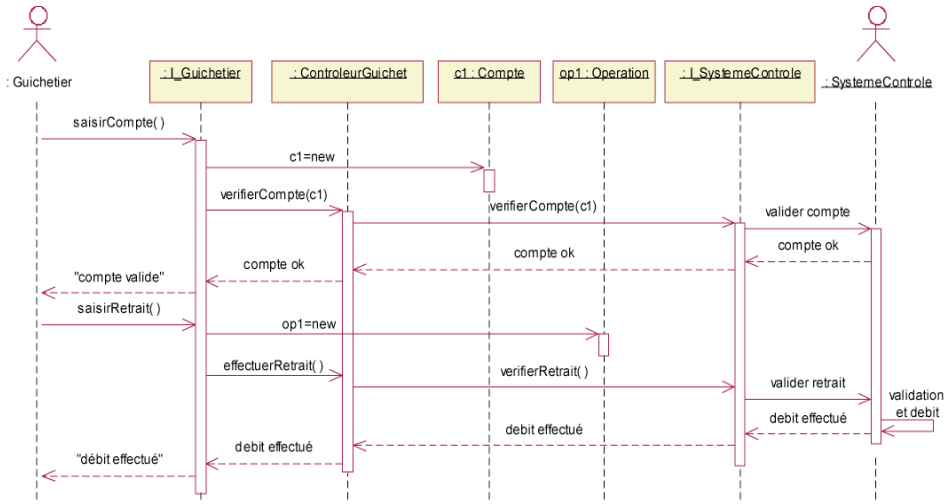
Exemple / Diagramme de séquence « système »

du CU « retrait en espèce »



Exemple (suite) - Diagramme de séquence de conception (incomplet)

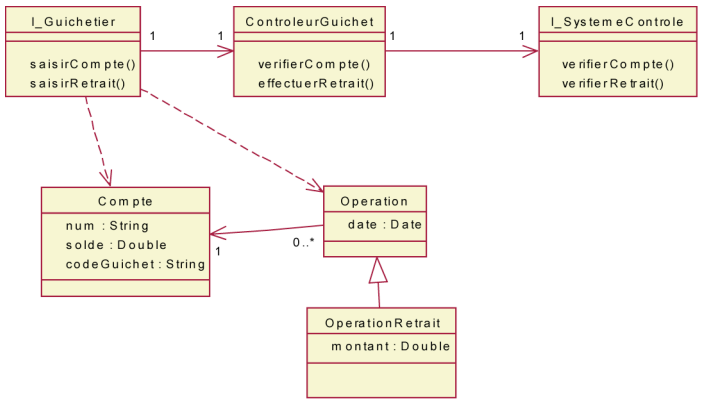
du CU « retrait en espèce »



Exemple (suite)

Diagramme de classes de conception

(classes collaborantes du CU « retrait en espèce »)



Synthèse des étapes d'un processus de développement simplifié (miniprojet solo)

1. Détermination des acteurs et de leurs tâches respectives
2. Détermination des cas d'utilisation et définition des frontières du système
3. Pour chaque CU développement des diagrammes de séquence "système" (le système entier est vu comme un seul objet)
4. Choix des cas d'utilisation critique (entre 3 et 5)
5. Développement des diagrammes de séquence faisant apparaître les objets et réalisation du diagramme de classes.
6. Programmation des cas d'utilisation critiques
7. Développement des autres cas d'utilisation

Références bibliographiques

- P. A. MULLER et N. GAERTNER, *Modélisation objet avec UML*, Eyrolles, 2000
- G. BOOCH, J. RUMBAUGH et Y. JACOBSON, *Le guide de l'utilisateur UML* , Eyrolles, 2000
- E. GAMMA et al., *Design Patterns,* Thomson, 1996

Liens utiles

- Norme UML disponible sur le site de l'OMG :
  - http://www.omg.org/technology/documents/modeling_spec_catalog.htm
- Autres cours en ligne (parmi beaucoup d'autres) :
  - http://lgl.isnetne.ch/uml/
  - http://www.isys.ucl.ac.be/etudes/cours/geti2101/tutorialslides/
  - http://deptinfo.cnam.fr/Enseignement/CycleSpecialisation/MAI/
- Logiciels de modélisation UML gratuits multi plate-formes
  - argoUML : http://argouml.tigris.org
  - PoseidonCE : http://www.gentleware.com
  - Plugin eclipse : http://www.omondo.org
- Logiciels windows gratuits :
  - http://staruml.sourceforge.net/en/index.php

Chapitre 8

Le Processus Unifié - Une Démarche Orientée Modèle

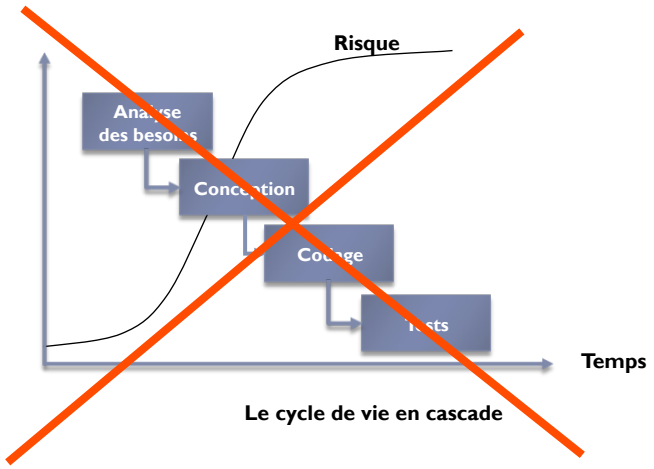
Sommaire

- Présentation générale du Processus Unifié
- Artefacts
- Enchaînement d'itérations
- Rôles
- Gestion de la configuration et des changements

Présentation générale du Processus Unifié

Développer le logiciel de façon itérative (1/2)

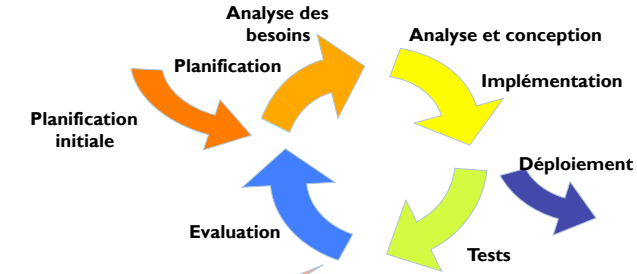
Bonne pratique n°1



Le cycle de vie en cascade

Développer le logiciel de façon itérative (2/2)

Bonne pratique n°1



Chaque itération a pour résultat une version exécutable

Un processus itératif et incrémental

Gérer les exigences

Bonne pratique n°2

- Problématiques
  - Modifications des exigences au cours du développement
  - Identification et intégration de l'ensemble des besoins au cours du processus
- Propositions
  - Organiser et décrire les fonctionnalités et les contraintes du système *(complétude et cohérence)*
  - Evaluer les changements à apporter au cahier des charges et estimer leur impact *(modifiabilité)*
  - Décrire les différentes décisions prises et en faire le suivi *(traçabilité)*

Utiliser des architectures à base de composants

Bonne pratique n°3

- L'architecture repose sur des décisions concernant :
  - L'organisation du système
  - Les choix des éléments structurels et de leurs interfaces
  - Leur comportement
  - Leur composition en sous systèmes
  - Les contraintes non fonctionnelles (réutilisation, robustesse, fiabilité, etc.)

(Exemples : Composants Com, Corba, EJB, classes .Net, etc.)

Modéliser graphiquement le logiciel

Bonne pratique n°4

Utilisation des diagrammes UML, et plus globalement de la notion de *modèle*.

- « Tout » n'est pas modèle
- « Tout modèle » n'est pas UML
- « Tout » n'est pas modèle UML !

Vérifier la qualité du logiciel

Bonne pratique n°5

- Fonctionnalités, fiabilité et performances
  - Tests (de « benchmark », de configuration, de fonctionnement, d'installation, d'intégrité, de charge, de performance, de stress, …)
  - Rapports de conception (ou revues)

Exercice

Décrivez en quelques lignes l'organisation du processus de développement que vous avez pu observer au cours de vos stages (précisez la taille des équipes de développement).

Déterminez si ce processus intègre à votre connaissance les cinq pratiques vues précédemment.

Qu'est ce que le Processus Unifié ? (1/3)

- **Le Processus Unifié ou UP (*Unified Process*) est une méthode générique de développement de logiciel développée par les concepteurs d'UML**
  - Générique signifie qu'il est nécessaire d'adapter UP au contexte du projet, de l'équipe, du domaine et/ou de l'organisation.
  - Il existe donc un certain nombre de méthodes issues de UP comme par exemple RUP (Rational Unified Process), 2TUP (Two Track Unified Process)
  - Il existe d'autres approches (qui ne sont en général pas complètement antinomique), comme par ex. les méthodes « agile », beaucoup moins orientées modèle, comme XP (eXtreme Programming)

Qu'est ce que le Processus Unifié ? (2/3)

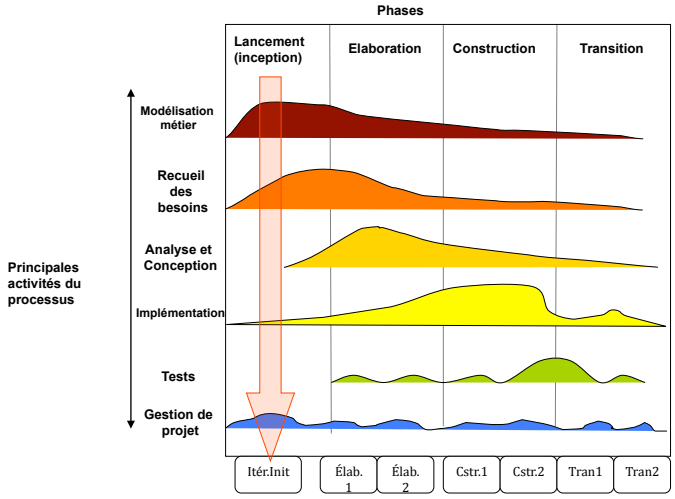
- Le processus unifié permet d'affecter des tâches et des responsabilités au sein d'une organisation de développement logiciel
  - Approche disciplinée pour des gros projets (chef de projet, analystes, intégrateur, intervenants, etc.)
  - Approche à adapter pour des petits projets
  - Pas particulièrement conçu pour le développement de systèmes embarqués

Qu'est ce que le Processus Unifié ? (3/3)

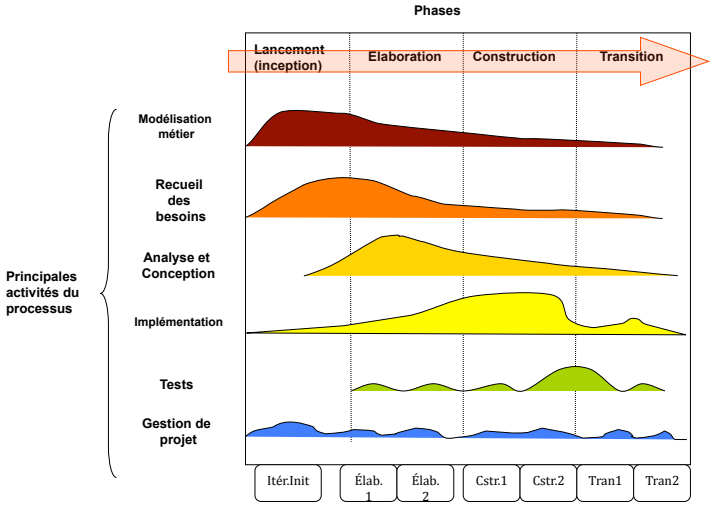
- Le processus unifié utilise le langage UML
- Le processus unifié est piloté par les cas d'utilisation
- Centré sur l'architecture
- Itératif et incrémental

La structure logique du Processus

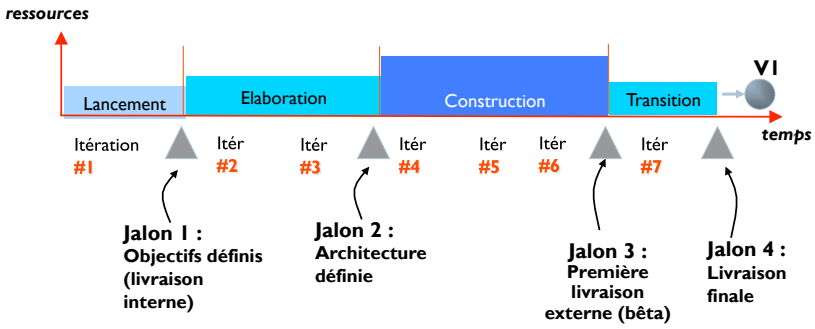
[Rational UP]



Les quatre phases

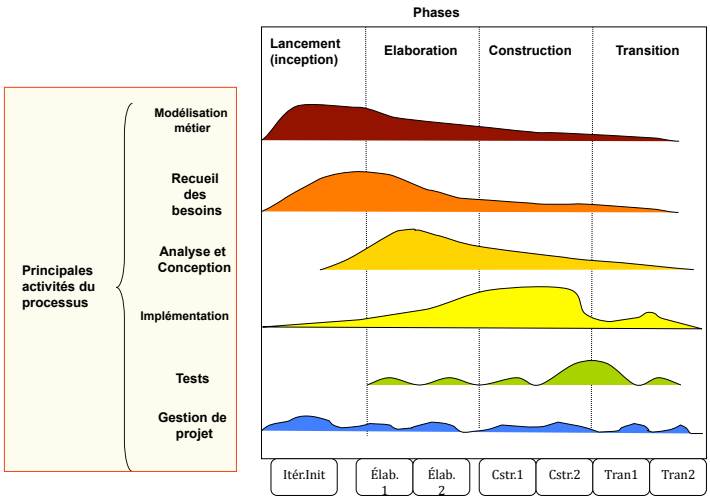


Les phases ▬, les itérations (#n) et les jalons ▲



➡ Les Jalons correspondent à des étapes d'évaluation de la phase terminée, et de lancement de la phase suivante

Les enchaînements d'activités



Gestion de projet

- « La **gestion de projet** est la mise en œuvre de connaissances, de ressources, de compétences, d'outils et de techniques qui permettent le lancement, la planification, la réalisation, le pilotage et la clôture d'un projet dans un cadre temporel et budgétaire, pour atteindre des objectifs précis. »
- De nombreux aspects ne sont pas couverts dans cette présentation (gestion personnel, budget, contrats, etc.)
- Objectifs
  - Planifier/évaluer un projet itératif
  - Gérer les risques
  - Contrôler les progrès (délais, coûts, qualité, efforts, satisfaction client, productivité, etc.)

Modélisation métier

- Compréhension de la structure et la dynamique de l 'organisation
- Comprendre les problèmes posés dans le contexte de l 'organisation
- Conception d 'un glossaire

Recueil et expression des besoins

- Auprès des clients et parties prenantes du projet
- Ce que le système doit faire
- Expression des besoins dans les cas d 'utilisation (exigences fonctionnelles)
- Spécification des cas d'utilisation en scénarios
- Limites fonctionnelles du projet et exigences non fonctionnelles (utilisabilité, fiabilité, performances)
- Planification et prévision de coût
- Ebauche de l 'IHM (prototypage et validation par l'utilisateur)

Analyse et conception

- Transformation des besoins utilisateurs en modèles UML
- Unified Modeling Language (pas une méthode mais un langage)
- Modèle d'analyse et modèle de conception, (éventuellement modèle de données)
- Étape importante :
  - de l'analyse à la conception : assigner des responsabilités aux classes ➡ Les patrons GRASP (General Responsability Assignment Software Patterns) [Présentation ultérieure]

Implémentation

- Développement incrémental
- Découpage en couches
- Composants d'architecture
- Retouche des modèles et des besoins
- Unités indépendantes, équipes différentes

Tests

- Etapes (unitaire, d'intégration, système, acceptation)
- Types :
  - De « benchmark » (comparaison)
  - De configuration (différentes config matérielles et logicielles)
  - De fonctionnement (vérification des CU)
  - D'installation
  - D'intégrité (fiabilité, robustesse, résistance)
  - De charge (conditions opérationnelles plus lourdes = nb utilisateurs, transactions,…)
  - De performance (en modifiant les configurations)
  - De stress (conditions anormales opérationnelles)

Exercice

- Décrivez en quelques lignes les phases que vous suivez lorsque vous êtes seul à développer un logiciel.
- Est ce que vous retrouvez certains éléments du Processus Unifié ?
- Quels tests effectuez vous lorsque vous développez seul ?

Les artefacts (ou livrables, ou délivrables)

Les artefacts

- Eléments produits lors du développement (documents, modèles, fichiers compilés, composants, etc.)
- ➡ Nous nous intéresserons ici aux artefacts de documentation et à leur gestion (création, modification, livraison)

Préambule

- Les parties des documents sont présentées ici à titre d'exemple.
- Suivant la taille du projet, il faut étendre ou réduire ces documents (voire les supprimer)
- Dans tous les cas, les documents doivent être mis régulièrement à jour, avec une gestion des changements (tableau de mise à jour dans l'en-tête de chaque doc)

Artefact ou pas ? Règles de base

- Objectif des artefacts : produire un meilleur logiciel
- Trop d'artefacts produit perte de temps, dispersion de l'équipe, augmentation des coûts
- ➡ Restez concentré sur le logiciel exécutable
- ➡ En cas de doute sur l'opportunité d'un artefacts, abstenez vous

Template des documents

<nom du projet>
Nom du document
Version <1.0>

Historique des révisions

Date	Version	Description	Auteur
<jj/mm/aa>	<x.x>	<details>	<name>
…	…	…	..

Sommaire

1. Introduction
   1. Objectifs
   2. Définitions, acronymes, et abréviations
   3. Références
2. <Sections spécifiques au document>

Ensemble d'artefacts

- L'ensemble de gestion
- L'ensemble des exigences
- L'ensemble de conception
- L'ensemble d'implémentation
- L'ensemble de déploiement

L'ensemble de gestion de projet

- Document de vision
- Plan de développement logiciel (planning des phases,rôles, responsabilités, jalons, activités)
- Liste des risques
- Planning d'itération
- Étude de rentabilité
- Points en suspens
- Glossaire
- Données sur des anomalies

Document de vision

[En-tête]

Sommaire

1. Problème

[motivations pour la création du logiciel]

2. Énoncé de la vision

[description du concept du système et intérêts]

3. Principaux acteurs concernés

[acteurs du développement et utilisateurs]

4. Caractéristiques du logiciel

[liste des principaux CU, performances, etc.]

Exemple de sommaire du document de vision étendu

1. Introduction
   - 1.1 Purpose
   - 1.2 Scope
   - 1.3 Definitions, Acronyms, and Abbreviations
   - 1.4 References
   - 1.5 Overview
2. Positioning
   - 2.1 Business Opportunity
   - 2.2 Problem Statement
   - 2.3 Product Position Statement
3. Stakeholder and User Description
   - 3.1 Market Demographics
   - 3.2 Stakeholder Summary
   - 3.3 User Summary
   - 3.4 User Environment
   - 3.5 Stakeholder Profiles
     - 3.5.1 <Stakeholder Name>
   - 3.6 User Profiles
     - 3.6.1 <User Name>
   - 3.7 Key Stakeholder or User Needs
   - 3.8 Alternatives and Competition
     - 3.8.1 <aCompetitor>
     - 3.8.2 <anotherCompetitor>
4. Product Overview
   - 4.1 Product Perspective
   - 4.2 Summary of Capabilities
   - 4.3 Assumptions and Dependencies
   - 4.4 Cost and Pricing
   - 4.5 Licensing and Installation
5. Product Features
   - 5.1 <aFeature>
   - 5.2 <anotherFeature>
6. Constraints
7. Quality Ranges
8. Precedence and Priority
9. Other Product Requirements
   - 9.1 Applicable Standards
   - 9.2 System Requirements
   - 9.3 Performance Requirements
   - 9.4 Environmental Requirements
10. Documentation Requirements
    - 10.1 User Manual
    - 10.2 Online Help
    - 10.3 Installation Guides, Configuration
    - 10.4 Labeling and Packaging

Plan de développement

Distribution dans le temps des ressources humaines (rôles et responsabilités) et matérielles

1. Structure organisationnelle
2. Rôles et responsabilités
3. Planning du projet
   1. Planning des phases
   2. Objectifs des itérations
   3. Délivrables
   4. Découpage en phases, identification des jalons et objectifs des itérations

Nb : Utilisation de tableaux, diagrammes (Gant par ex. pour des projets importants)

Liste des risques

1 <Identificateur de risque> *[nom descriptif ou numéro]*
- 1.1 Niveau de risque
- 1.2 Description
- 1.3 Conséquences
- 1.4 Indicateurs
- 1.5 Stratégie de traitement du risque

2 <Risque suivant>
[…]

Planning d'itération

[en-tête]

Sommaire

1. Planning
   [Description temporelles des jalons intermédiaires, versions beta, demos, etc.]
2. Ressources
   [Matérielles, humaines, et financières]
3. Cas d'utilisation
   [C.U. et scénarios qui seront développés dans cette itération]
4. Critères dévaluation
   [Fonctionalités, performances, mesures de qualité, etc.]

L'ensemble des exigences

- Demandes des intervenants
- Modèle des CU
- Spécifications supplémentaires
- Modèle métier

Demandes des intervenants

- Document permettant de noter et référencer les demandes des intervenants (développeurs, sous-traitants, clients, etc.), concernant des modifications d'exigences (fonctionnelles ou non)
- Date / Nom / Description / Prise en compte ou non

Modèle des cas d'utilisation

(Ensemble des fiches de C.U. et du diagramme des C.U.)

1. Nom
2. Résumé
3. Acteurs
4. Description des scénarios
   1. Scénario nominal
   2. Scénarios alternatifs
5. Pré conditions
6. Post conditions
7. Enchaînements alternatifs (description textuelle ou tableau)
8. Autres spécifications :
   1. Besoins d'IHM, ergonomie
   2. Robustesse, confidentialité, performances (temps de réponse, charge), disponibilité, etc.

Action de l'acteur principal	Action du système	Action de l'acteur secondaire
1. action1	2. Réponse du système et traitement	3. Réception d'une information

Spécifications supplémentaires

1. Introduction
2. Fonctionnalités
   2.1 <Exigence fonctionnelle 1>
3. « Utilisabilité » (anglicisme de usability)
   3.1 <Exigence d'utilisabilité1>
4. Sûreté de fonctionnement
   4.1 <Exigence SdF 1>
5. Performance
   5.1 <Exigence de performance 1>
6. Contraintes de conception
   6.1 <Contrainte 1>
7. Aide en ligne et système d'aide
8. Composants achetés
9. Interfaces
   9.1 Interfaces utilisateur
   9.2 Interfaces matérielles
   9.3 Interfaces logicielles
   9.4 Interfaces de communication
11. Exigences de licence
12. Copyright
13. Normes applicables

Modèle métier

- Diagramme de classes métier
- Diagramme de C.U. métier
- Diagramme d'activités métier

L'ensemble de conception

- Modèle de conception (Diagrammes UML)
- Description de l'architecture (privilégier un représentation graphique UML ou non)
- Plan de test
- Cas de test

Plan de test

1. Éléments ciblés par les tests
2. Vue d'ensemble des tests planifiés
   - 2.1 Descriptif des test inclus dans le plan de dev.
   - 2.2 Descriptif des autres tests exclus du plan de dev.
   - 2.3 Descriptif des tests candidats à une inclusion dans le plan de dev.
3. Types et techniques de test
   - 3.1 Test d'intégrité
   - 3.2 Test fonctionnel
   - 3.3 Test de "benchmark"
   - 3.4 Test de l'interface graphique
   - 3.5 Test de performance
   - 3.6 Test de charge
   - 3.7 Robustesse
   - 3.8 Test de stress
   - 3.9 Test de sécurité et de contrôle d'accès
   - 3.11 Test de configuration
   - 3.12 Test d'installation

➡ Il faut documenter les tests envisagés pour l'application

Plan de test

Objectifs de la technique :	*[Description des objectifs]*
Technique:	*[Description de la technique utilisée, par ex. « Exécuter chaque scénario et vérifier que… »]*
Oracles:	*[Stratégies utilisées par la technique pour observer précisément le succès ou la défaillance du test]*
Outils requis :	*[Besoins en terme de scripts, fichiers de log, etc…]*
Critère de succès :	*[Condition pour que l'ensemble du test soit positif]*
Considerations particulières :	*[Contraintes liées au type de test, recommandations, remarques, etc…]*

Cas de test

Cas de test : *<Nom du cas de test>*
Numéro d'identification : *<Identifiant>*
Catégorie : *<Catégorie de cas de test>*
[Les valeurs possibles de catégorie sont Fonctionnel et Performance]

1. Introduction

[Cette section décrit brièvement le rôle et l'objectif du Cas de Test]

2. Enchaînement des événements

[Décrire les étapes que le testeur doit suivre. Les réponses (R) sont les résultats attendus par le test.]

1. Étape 1
   R. Réponse 1.
2. Étape 2
   R. Réponse 2
3. Étape 3
   R. Réponse 3

3. Besoins techniques

[Besoins techniques liés à ce cas de test, par. Ex. un simulateur d'environnement, une machine sous windowsNT, etc.]

3.1 < Premier besoin >

[…]

L'ensemble d'implémentation

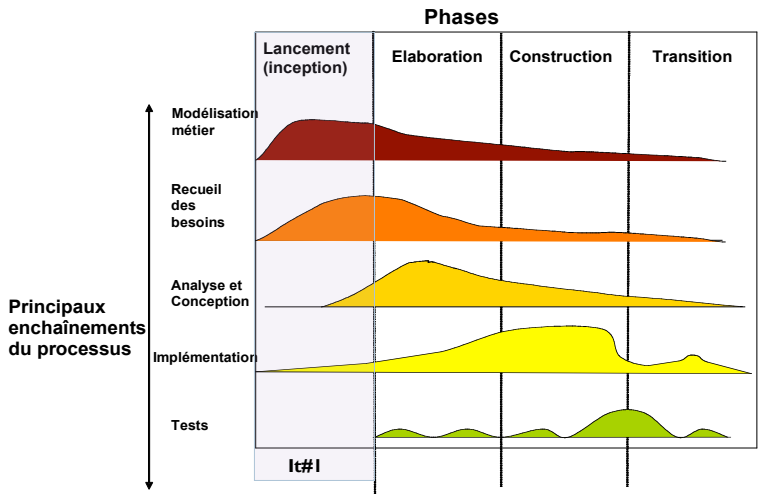
- Le code source et les exécutables
- Les fichiers de données associés (par ex. Javadoc) ou les fichiers permettant de les produire

L'ensemble de déploiement

- Scripts et guide d'installation
- Documentation utilisateur
- Matériel de formation

Les enchaînements d'itération

Phase de lancement



Phase de lancement

- Comprendre les objectifs et la portée du projet
- Objectifs :
  1. Comprendre le système à construire
  2. Identifier la fonctionnalité principale du système
  3. Déterminer au moins une solution (architecture) possible
  4. Décider du processus à appliquer et des outils à utiliser
  5. Evaluer les coûts et planning
  6. Les principaux risques

Nb : en général une seule itération, sauf pour projets importants, innovants, ou très risqués

Objectif 1 : Comprendre le système

- Produire une « vision » (artefact:document de vision), i.e. opportunités apportées par l'application, les utilisateurs cibles, quelques cas d'utilisation clés (un ou deux), certains besoins non fonctionnels (artefact:spécifications supplémentaires)
- Générer une description large et superficielle (élaboration du diagramme de contexte (artefact: modèle d'analyse) et identification des acteurs et des C.U. principaux (artefact: modèle des C.U.), décrire les C.U., créer un artefact:glossaire et/ou un artefact:modèle métier, développer des prototypes d'interface jetables si besoin)
- Comprendre les besoins en terme de contrôle et identifier les contraintes (compléter artefact:modèle d'analyse avec diagrammes de séquence système et diagramme temporels)

Objectif 2 : Identifier la fonctionnalité essentielle du système

- Collaboration chef de projet, architecte, et client (artefact: demandes des intervenants) pour déterminer les C.U. les plus critiques
  - La fonctionnalité est le noyau de l'application
  - Elle DOIT être livrée

Objectif 3 : Déterminer au moins une solution possible

- Architecture (client-serveur, centralisée, distribuée, etc.)
- Choisir technologies et éventuellement faire des tests d'implémentation pour estimer les risques liés à une technologie
  1. Effectuer un partitionnement matériel/logiciel du système
  2. Compléter l'architecture du substrat matériel
  3. Choisir les technologies matérielles de la partie matérielle (FPGA/VHDL)
  4. Choisir les technologies matérielles de la partie logicielle (Microcontrôleurs)
  5. Choisir les technologies logicielles de la partie logicielle (langage, compilateurs, etc.)

Partitionnement HW/SW

- connaissance et l'étude du marché des technologies matérielles et logicielles utilisables.
- La veille technologique et le maintien à jour des connaissances techniques des ingénieurs sont la base indispensable à l'efficacité de l'activité «Etudes d'implémentation et partitionnement logiciel/matériel ».

Choisir les technologies matérielles de la partie logicielle

- Processeurs / SoC / Mémoire / Bus / etc.
- + critères spécifiquement logiciels comme la disponibilité et/ ou la performance de tel système d'exploitation ou de tel middleware sur la plateforme matérielle envisagée

Choisir les technologies logicielles de la partie logicielle

- Les langages de programmation :
  - le choix d'un langage de programmation ne doit se faire que sur des critères purement techniques fondés sur
    - la disponibilité
    - la qualité des compilateurs
    - les performances attendues
    - la maturité industrielle du langage et des outils associés.

Choisir les technologies logicielles de la partie logicielle

- Les systèmes d'exploitation :
  - La problématique liée au choix du système d'exploitation pour une plateforme matérielle donnée s'articule autour de trois points principaux:
    - Les facilités de développement, d'intégration et d'évolution
    - Les performances (CPU, réseau, temps réel)
    - Le coût (licences de développement et exécutifs)

Choisir les technologies logicielles de la partie logicielle

- Lors du développement logiciel il est aujourd'hui quasi impossible de ne pas intégrer au moins une des technologies suivante :
  - Bibliothèques
  - COTS (Components On The Shelf)
  - Des environnements et des cadres de développement (*frameworks*)
  - Des générateurs de codes
  - Des *middleware*
- Problématique de maintenance, coût, performances, validation, évolutivité, etc.
- Quelques pistes : certification, support, communauté

Objectif 4 : Comprendre les coûts, les délais et les risques

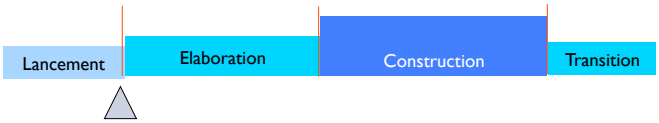
- Examiner la faisabilité du projet (étude de rentabilité, artefact: liste des risques)
- Établir le plan du projet (artefact : plan de développement du logiciel)

Objectif 5 : Décider du processus à appliquer et des outils à utiliser

- Faire des choix :
  - Processus de développement
  - Outils de développement
  - Compléter le plan de développement avec les choix, mettre à jour liste de risques, coûts et délais

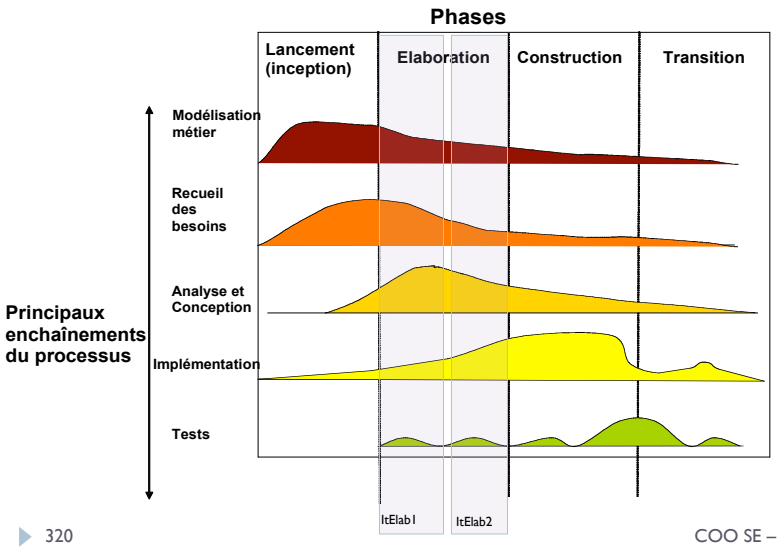
Revue de projet : Jalon fin de lancement

- Les différents intervenants valident:
  - le périmètre du projet, coûts et délais
  - la liste des exigences
- Les risques initiaux sont identifiés et il existe une stratégie de réduction pour chacun d'eux
- L'ensemble des artefacts produit est complet, à jour et cohérent



Exercice : Lister ci dessous l'ensemble des artefacts produits pendant la phase de lancement

Phase d'élaboration



La phase d'élaboration

- Construction d'un squelette d'architecture en intégrant les risques majeurs et affiner les plans de projet
- Objectifs :
  1. Comprendre en détail les exigences
  2. Concevoir, implémenter, valider l'architecture (proto architectural exécutable)
  3. Réduire les risques essentiels et estimer plus exactement le budget
  4. Affiner le plan de développement

NB: en général une à trois itérations (artefact : plan d'itération)

Objectif 1 : comprendre les exigences en détail

- Mettre à jour tout au long de cette phase :
  - Le modèle des C.U. (certains C.U. très simples et ne présentant aucun risque ne doivent pas être formalisés)
  - Le glossaire
- Hiérarchiser et faire des packages de C.U.

Objectif 2 : Concevoir, implémenter, et valider l'architecture

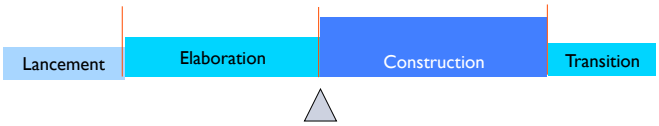
- Architecture: définir les sous-systèmes, les composants, et leurs interfaces (utiliser au maximum des *frameworks* d'architecture)(artefact : description de l'architecture)
- Déterminer les C.U. significatifs du point de vue architectural
- Concevoir les C.U. critiques
- Regrouper en packages les classes identifiées (artefact : modèle de conception)
- Réévaluer la couverture architecturale par les scénarios critiques
- Concevoir la base de données
- Décrire la concurrence, les processus, les *threads*, et la distribution physique
- Identifier les patterns
- Implémenter et tester les scénarios critiques (artefact : plan de test, cas de test)

Objectifs 3 & 4

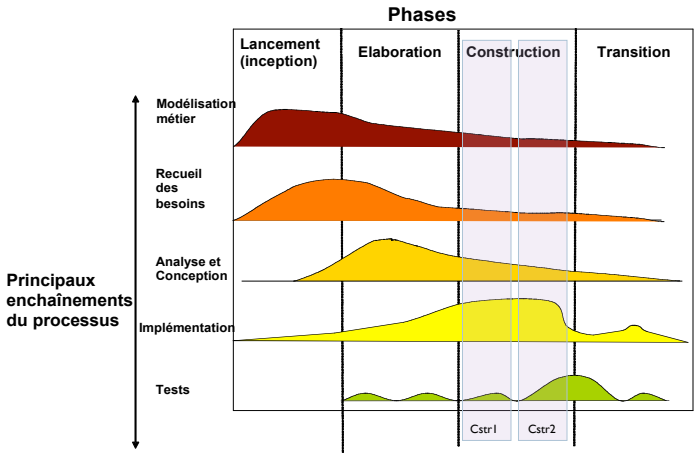
- Réduire les risques essentiels et estimer au mieux les délais et coûts et raffiner le plan de développement
  - Utiliser toutes les informations provenant des activités de conception pour mettre à jour les risques, délais et coûts.

Revue de projet : jalon fin d'élaboration

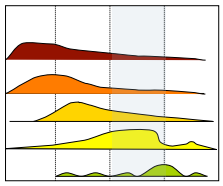
- Évaluer :
  - Stabilité vision et exigences
  - Stabilité architecture
  - Crédibilité du plan d'itération
- En cas de doute : refaire une itération



Phase de construction



Phase de construction (3/4)



- Gestion des ressources matérielles (installation sur plateformes choisies)
- Optimisation du processus pour réduire les coûts de développement
- Améliorer la qualité
- Compléter les modèles suivant les besoins d 'implémentation
- Produit : le logiciel installé sur les plate-formes choisies, les manuels d'utilisation, description de la version mise en place

La phase de construction

- Phase concentrée sur la conception, l'implémentation et les tests
- Objectifs :
  1. Minimiser les coûts de développement et obtenir un certain degré de parallélisme
  2. Développer de façon itérative un logiciel prêt à la transition vers les utilisateurs

NB : même pour de petits projets, il faut plusieurs itérations (entre 2 et 4)

Objectif 1 : Minimiser les coûts de développement et obtenir un certain degré de parallélisme

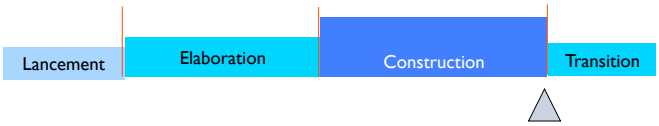
- S'organiser autour de l'architecture
- Gestion de la configuration
- Gestion des demandes de changement
- Appliquer l'architecture
- Assurer une progression continue

Objectif 2 : Développer de façon itérative un logiciel prêt à la transition vers les utilisateurs

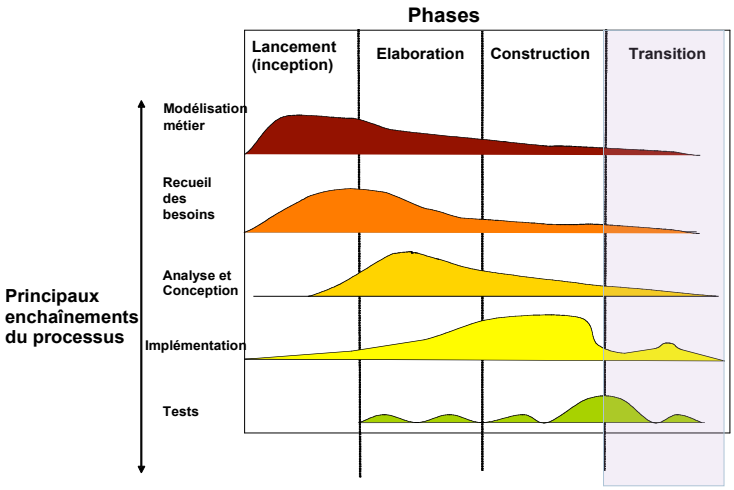
- Décrire les C.U. restants et les spécifications supplémentaires
- Terminer la conception
- Concevoir la base de données
- Coder et exécuter les tests unitaires
- Effectuer les tests d'intégration et système
- Premiers déploiements et boucle de rétroaction

Revue de projet : le jalon fin de Construction

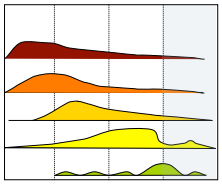
- Évaluation :
  - Version du logiciel est elle stable ?
  - Tous les intervenants sont prêts ?
  - Dépenses réelles/prévisionnelles acceptables ?



Phase de transition



Phase de transition (4/4)



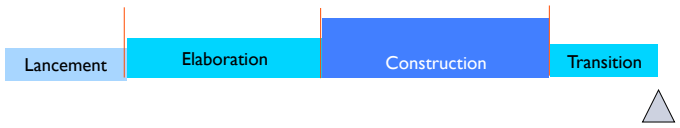
- Déployer
- Tester
- Valider (réponse aux attentes des utilisateurs)
- Accompagner l'utilisateur final (packaging, documentation, formations, manuels …)

Phase de transition

- Objectifs :
  - Exécuter les tests bêta
  - Former les utilisateurs
  - Préparer le site de déploiement
  - Préparer le lancement
  - Obtenir l'accord des intervenants
  - Améliorer les performances futures

Revue de projet : Jalon Fin de la Transition

- Évaluation :
  - Satisfaction des utilisateurs
  - Bilan sur les ressources réellement consommées



Bibliographie

P. Kroll et P. Kruchten, *Guide pratique du RUP*, CampussPress, 2003

P. Kruchten, *Introduction au Rational Unified Process*, Eyrolles, 2000

Craig Larman, *Applying UML and patterns - An introduction to object oriented analysis and design and the Unified Process*, Prentice Hall, 2004

Exemple de projet entièrement RUP : http://jdbv.sourceforge.net/