1.4 Software Design (La conception logicielle - تصميم البرمجيات)

1.4.1 Définition et objectifs

La définition utilisée est celle de IEEE qui dit que la conception logicielle est à la fois :

> ***« Le processus de définition de l'architecture, des composants, des interfaces et autres caractéristiques d'un système ou d'un composant ».***
>
> ***« Le résultat de ce processus »***

Donc, la conception peut être vue autant comme :

> ***un produit*** = description de l'architecture du système, comment il est décomposé en composants indépendants, l'interface et le comportement de ces composants
>
> ***un processus*** = étape du cycle de vie où les exigences et spécifications sont analysées dans le but de produire une description de l'organisation interne d'un système

En termes d'un processus classique de développement, on retrouve ainsi les étapes suivantes :

> ***1. Conception architecturale :*** *C'est le développement d'une structure modulaire et la représentation des relations de contrôle existant entre les modules. En d'autres termes, elle décrit comment le logiciel est décomposé et organisé en composants.* Cette étape prend en entrée *l'expression des besoins et comme sortie solution informatique en termes :*
>
> - ✓ *des fonctions*
> - ✓ *des données*
> - ✓ *des interfaces*
>
> ***2. Conception détaillée :*** *Elle décrit chaque composant.*

Comme objectifs, la conception logicielle doit analyser les exigences pour produire une description de la structure interne du logiciel qui servira à la construction du logiciel :

- Doit décrire l'architecture du logiciel.
- Doit décrire les composants et les interfaces entre les composants avec suffisamment de détails pour construire ces composants
- Sert à faire les liens entre les exigences et la construction (code et test).
- Produire en quelque sorte un plan du logiciel.

Le rôle clé de la conception est de générer des modèles qui :

- décrivent la solution avant qu'elle soit mise en œuvre
- pouvant être analysés et évalués
- pouvant servir comme base pour les tests
- sont utilisables pour la planification des étapes subséquentes

Dans SWEBOK, la conception logicielle est structurée comme suit :

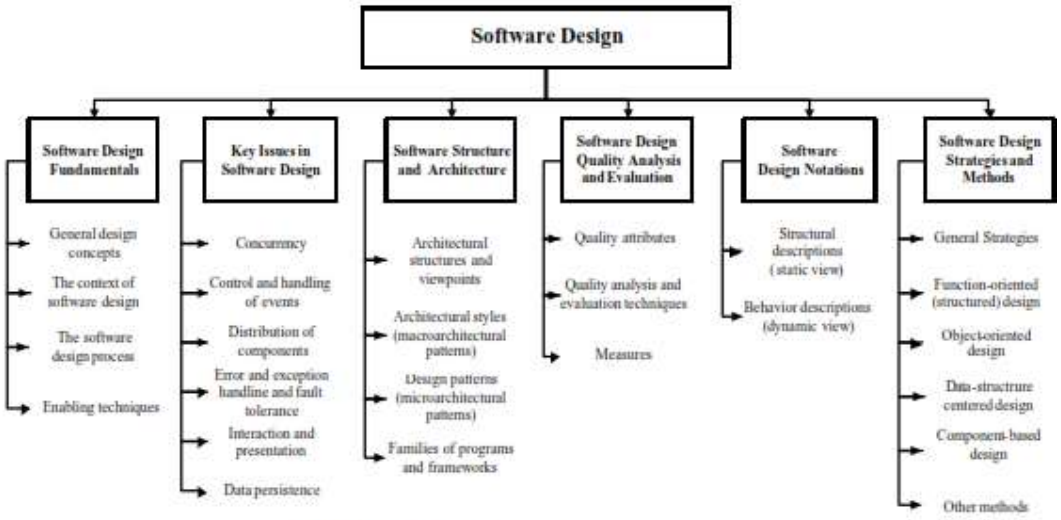


Figure 1 Breakdown of topics for the Software Design KA

1.4.2 Software Design Fundamentals

(Principes fondamentaux de la conception logicielle - أساسيات تصميم البرمجيات)

Cette section introduit 4 concepts qui offrent une base pour la compréhension du rôle et de la portée du software design :

- ✓ Concepts généraux de conception :
  - Le « design » peut être vu comme une forme de résolution de problème
  - Par exemple un problème sans solution définitive
  - est intéressant pour comprendre les limites du « design »
  - Sert à comprendre le « design » dans le sens général : buts, contraintes, alternatives, représentations et solutions
- ✓ Contexte de la conception logicielle :
  - Sert à comprendre le rôle et la place de Software Design
  - Il est important de comprendre le contexte dans lequel le design s'harmonise, i.e., le cycle de vie logiciel.
  - Les caractéristiques majeures des exigences vs le Software Design vs la construction (code) vs les tests doivent être comprises.
- ✓ Processus de la conception logicielle :
  - Se divise en 2 parties importantes : conception architecturale et conception détaillée
  - Le résultat de ce processus sera un ensemble de modèles et d'artéfacts qui décrit les décisions majeures qui auront été prises.
- ✓ Techniques permettant la conception logicielle :
  - Ce concept décrit les « principes » sous-jacents aux différentes approches du Software Design :
    - Abstraction
    - Couplage et cohésion
    - Décomposition et modularité
    - Encapsulation (data abstraction and information hiding)
    - Séparer l'interface de l'implantation
    - Complétude, suffisance et « primitivité »

1.4.3 Key issues in Software Design

(Enjeux clés de la conception de logiciels - القضايا الرئيسية في تصميم البرمجيات)

Un certain nombre d'éléments clés doivent être considérés :

- ✓ Concurrence des composants.
- ✓ Contrôle et manipulation des événements
- ✓ Distribution des composants
- ✓ Manipulation des erreurs et des exceptions et la tolérance aux fautes.
- ✓ Interaction et présentation
- ✓ Persistance des données

1.4.4 Software Structure and Architecture

(Structure et Architecture logicielle – تركيبة و بنية البرامج)

Cette section introduit 4 parties :

- ✓ Structure et points de vue architecturaux :
  - La conception logicielle doit être décrite et documentée selon différents points de vue. Ex:
    - Vue logique (point de vue des exigences)
    - Vue de processus (concurrence)
    - Vue physique (distribution)
    - Vue de développement (découpage des composants)
- ✓ Styles architecturaux :
  - Un style d'architecture est « Un ensemble de contraintes qui définissent un ensemble ou une famille d'architectures » Ex:
    - Tubes et filtres (Pipes and Filters)
    - Client-Serveur (Client-Server)
    - A base de règles (Rule-Based)
- ✓ Patron de conception :
  - Un patron de conception est « une solution commune à un problème commun dans un contexte donné » Ex:
    - Singleton
    - Decorator
    - Mediator
- ✓ Familles de programmes et frameworks :
  - Une possibilité pour la réutilisation des conceptions logicielles est de faire des familles de logiciels (Ligne de production logicielle) ; lesquelles sont faites en identifiant les points communs et en utilisant des composants réutilisables qui satisfont les membres des différentes familles.
  - Un « framework » est un sous-système partiellement complet qui peut être étendu en «instanciant» correctement quelques « plug-ins » spécifiques.

1.4.5 Software Design Quality Analysis and Evaluation

(Analyse de qualité et évaluation de la conception logicielle –

تحليل الجودة وتقييم تصميم البرمجيات)

Cette section comporte 3 parties :

- ✓ Les attributs de qualité :

- Différents attributs de qualité sont considérés pour obtenir une conception logicielle de qualité.
  - Détectables à l'exécution
  - Non-détectables à l'exécution
  - Reliés à la qualité de l'architecture

✓ Les outils d'analyse de qualité et d'évaluation :
- On peut décomposer les outils et techniques qui peuvent assurer la qualité du design en plusieurs catégories :
  - Les revues de conception logicielle
  - Analyse statique
  - Simulation et prototypage

✓ Les mesures :
- Il existe deux grandes catégories de mesures quantitatives :
  - Orientées-fonction
  - Orientées-objet

1.4.6 Software Design Notations

(Notations de conception logicielle – ترميزات تصميم البرمجيات)

Différentes notations sont utilisées autant dans la partie architecturale que dans la partie détaillée :

✓ Les descriptions structurelles :
- C'est une vue statique souvent graphique qui décrit et représente l'aspect structurel de la conception logicielle. Elle permet aussi de décrire les composants et leurs relations. Ex:
  - Les diagrammes de classes et d'objets
  - Les diagrammes de composant
  - Les diagrammes Entité-Relation
  - ADL (Architecture Description Languages);

✓ Les descriptions comportementales :

✓ C'est une vue dynamique utilisant notations, graphiques et langages pour décrire le comportement des composants et du logiciel. Ex:
- Les diagrammes d'activité
- Les diagrammes de collaboration
- Les diagrammes de séquence
- Les diagrammes d'état et de transition
- Les diagrammes de flux de données

1.4.7 Software Design Strategies and Methods

(Stratégies et méthodes de conception logicielle – استراتيجيات وطرق تصميم البرمجيات)

Une stratégie aide à guider le processus de conception, souvent de façon assez générale (heuristiques).

Une méthode est plus spécifique et fournit habituellement :

1. Un ensemble de notations à utiliser avec la méthode ;
2. Une prescription du processus à suivre pour utiliser la méthode ;
3. Un ensemble de règles et d'heuristiques qui aident à utiliser la méthode ;

Cette section comporte 3 parties :

✓ Les stratégies générales :
- Les exemples les plus souvent citées sont :
  - diviser pour régner

        - raffinement successif
        - approche descendante et ascendante
        - utilisation de patterns
        - utilisation d'approche incrémentale et itérative
- ✓ Conception orientée fonction (conception structurée) :
    - Il y a eu plusieurs stratégies et heuristiques proposées :
        - analyse des transformations vs. des transactions
        - stratégie basée sur la portée de l'effet vs. du contrôle
        - etc.
- ✓ Conception orientée objet :
    - Il y a eu plusieurs stratégies proposées telles que :
        - abstraction de données
        - approche bas »e sur les responsabilités
        - etc.
- ✓ Conception basée sur les structures de données :
    - La moins populaire des stratégies
        - La définition des entrées/sorties du système est faites en premier.
        - Les structures de contrôle du programme sont définies à partir des structures de données.
    - Exemple : La méthode de Jackson
- ✓ Autres méthodes :
    - Autres stratégies telles que :
        - méthode rigoureuse de développement
        - méthode transformationnelle
        - etc.

1.4.8 Quelques compléments d'informations

A- La modélisation en général

A-1 Les modèles :

- ✓ **Définition :**
  - Un modèle est une représentation abstraite de la réalité qui exclut certains détails du monde réel.
- ✓ **Utilité :**
  - Reflète ce que le concepteur croit important pour la compréhension et la prédiction du phénomène modélisé, les limites du phénomène modélisé dépendent des objectifs du modèle.
  - Permet de réduire la complexité d'un phénomène en éliminant les détails qui n'influencent pas son comportement de manière significative.

A-2 La modélisation :

- ✓ **Définition :**
  - Processus par lequel on arrive à élaborer un modèle décrivant un système réel (ou un phénomène du monde réel).

A-3 Types de modélisation :

- ✓ **Modélisation à priori :**
  - ➢ Modéliser un système avant sa réalisation (le système n'existe pas encore).
  - ➢ **Objectifs :**
    - Comprendre le fonctionnement du futur système.
    - Mesurer et Maîtriser sa complexité.
    - Assurer sa cohérence.
    - Pouvoir communiquer au sein de l'équipe de réalisation.
- ✓ **Modélisation à posteriori :**
  - ➢ Modéliser un système après sa réalisation (le système existe déjà).
  - ➢ **Objectifs :**
    - Corriger les erreurs dans l'ancien système.
    - Faire évoluer l'ancien système.

B- Approches de modélisation pour le logiciel :

B-1 Approche fonctionnelle :

- Approche traditionnelle basée sur l'utilisation des procédures et des fonctions.
- Les grands programmes sont décomposés en sous-programmes.
- La modélisation fonctionnelle est une ***approche descendante*** :
  - La modélisation du système se base sur les fonctions, et non pas sur les objets.
  - On commence par déterminer la fonction globale du système.
  - Puis, on décompose la fonction globale du système en plusieurs sous-fonctions jusqu'à obtenir des fonctions élémentaires simples à programmer.
  - Il s'agit donc d'une démarche descendante.

- **Avantages :**
  - Adéquate pour les petits logiciels et les systèmes peu complexes.
  - Démarche ordonnée et organisée.

- **Inconvénients :**
  - Pose des problèmes de structuration de données, car elle est orientée fonctions.

- Produit des logiciels non réutilisables.
- Produit des logiciels très difficile à les faire évoluer ou corriger.

B-2 Approche orientée objets :

- On identifie les éléments du système et on en fait des objets.
- On cherche à faire collaborer ces objets pour qu'ils accomplissent la tâche voulue.
- C'est une approche fondée sur les objets :
  - Le modèle à produire est décrit en termes d'objets et non pas en termes de fonctions.
  - On peut partir des objets du domaine (briques de base) et remonter vers le système global.
  - On définit également les interactions et les collaborations entre les objets du système.
  - Il s'agit essentiellement d'une ***approche ascendante***.

- **Avantages :**
  - Démarche naturelle et logique.
  - Raisonnement par abstraction sur les objets du domaine.

- **Inconvénients :**
  - Parfois moins intuitive que l'approche fonctionnelle.
  - Rien dans les concepts de base objets ne précise comment modéliser la structure objet d'un système de manière adéquate.

- **Différences entre COO (Conception orientée objet) & POO (Programmation orientée objet):**
  - La conception objet doit être dissociée de la programmation objet.
  - Les langages à objets ne présentent que des manières particulières d'implémenter certains concepts. Ils ne valident en rien la conception.
  - Il est même possible de concevoir objet et de coder en fonctionnelle (Langage C, par exemple).
  - Connaître des langages de programmation objet comme C++ ou Java n'est pas suffisant.
    - Il faut savoir s'en servir en utilisant un moyen de modélisation, d'analyse et de conception objet.
    - Sans les langages de haut niveau (donc de modélisation), il est difficile de comparer plusieurs solutions de modélisation objet.

- **Utilisation efficace de l'approche objet :**
  - Niveau démarche de modélisation :
    - Il faut suivre une démarche d'analyse et de conception objet.
    - Ne pas effectuer une analyse fonctionnelle, et se contenter d'une implémentation objet.
  - Niveau langage de modélisation :
    - Il faut utiliser un langage de modélisation qui permet de représenter les concepts abstraits, de limiter les ambiguïtés et de faciliter l'analyse.

C- Langages de modélisation :

- ✓ **Objectifs :**
  - Définir la sémantique des concepts.
  - Définir une notation pour la représentation des concepts.
  - Définir des règles d'utilisation des concepts et de construction.

- ✓ **Choix du langage de modélisation :**
  - Adaptés aux systèmes procéduraux (MERISE, …).
  - Adaptés aux systèmes temps réel (ROOM, SADT, …).
  - Adaptés aux systèmes à objets (OMT , Booch, UML, …).

- ✓ **Ateliers de génie logiciel :**
  - Présenter des outils pour la modélisation.
  - Le rôle de ces outils est primordial pour l'utilisabilité en pratique des langages de modélisation

D- Conception générale & Conception détaillée

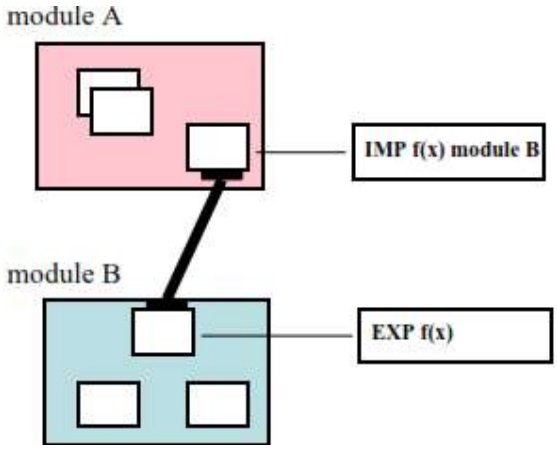
D-1 Conception générale (Les spécifications) :

- ***Passage :***
  - De l'expression des besoins à la solution informatique
  - Spécifications informatiques
  - des fonctions
  - des données
  - des interfaces
  - Phase précédant la conception détaillée
- ***Document des spécifications***
  - destiné aux développeurs
- **Exemple : spécification objet**
  - ✓ Représentation des entités et des relations entre entités
  - ✓ Les entités sont des objets
  - ✓ Un objet a:
    - Un nom
    - Des attributs
    - Des méthodes
  - ✓ Les propriétés des relations
    - L'héritage: relation « est un »
    - L'agrégation : relation « est composé de »
    - L'association: relation « père-fils »
  - ✓ on construit les spécifications en opérant par :
    - Généralisation: d'une ou plusieurs classes définies on crée une classe mère plus générale
    - Spécialisation: à partir d'une classe définie on crée une ou plusieurs autres classes plus détaillées
    - Extension: on crée une classe supplémentaire et si besoin une classe mère par généralisation
    - Décomposition : une classe est décomposée en sous classes
    - Composition : une classe est composée à partir d'autres sous classes

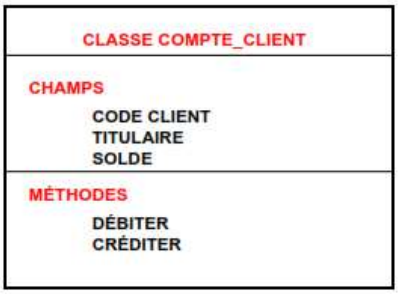
✓ **D-2 Conception détaillée**

- ***Passage :***
  - De la De la spécification à la conception détaillée
    - ✓ transformation en unités de programmes des fonctionnalités du logiciel
      - conception fonctionnelle descendante
      - conception orientée objet
    - ✓ éléments de conception
      - concepts de structuration
      - méthodes de conception
      - langages de programmation
  - Conception impérative
    - ✓ fonctionnelle descendante
    - ✓ modulaire
  - Conception applicative
    - ✓ orientée objets
    - ✓ héritage
- **Conception fonctionnelle descendante (Conception impérative)**
  - Module
    - ✓ unité fonctionnelle reliée aux données
    - ✓ langages: Ada, C, Modula 2
  - décomposition d'un module:

- ✓ l'interface: spécifications des composants exportés
- ✓ le corps: composants réalisant les fonctionnalités



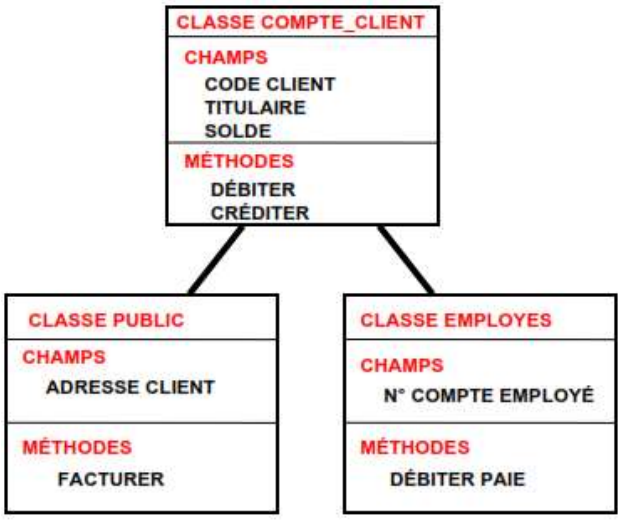
- **Conception orientée objet (Conception applicative)**
  - les fonctions changent ; les objets restent
  - penser réutilisation
  - unité de conception = classe d'objets
  - langages: Java, C++
  - Donne une vue ***"Sur quoi le système agit-il?"***
  - ***Un objet, c'est:***
    - ✓ Un ensemble d'attributs soumis à des actions spécifiques
      - les attributs -> état de l'objet
      - les actions -> les méthodes
  - Un objet a un comportement:
    - ✓ déterminé par ses actions
    - ✓ avec des contraintes sur les actions ou attributs
  - ***Classe d'objets***
    - ✓ ensemble d'objets
      - mêmes attributs statiques
      - mêmes attributs dynamiques
    - ✓ une instance est un objet particulier de la classe
      - crée avec les attributs d'états de la classe
      - se comporte selon les méthodes de la classe



  - ***Héritage***
    - ✓ des sous-classes: organisation hiérarchique
    - ✓ héritage des champs et des méthodes
      - extension des champs et méthodes de la classe dans la sous-classe
      - évite la duplication de code
    - ✓ héritages:
      - ***simple***: nouvelle classe issue d'une classe origine

- de toutes les propriétés
- de certaines propriétés

– ***multiple***: nouvelle classe issue de plusieurs classes



- **L'encapsulation**
  - Opacité du fonctionnement interne d'un objet
  - Accès aux seules propriétés et services nécessaires
- ***Le polymorphisme***
  - Traitement l'adaptant aux diverses versions des objets
- ***La surcharge***
  - Réécriture d'un traitement hérité: ajout ou suppression de fonctionnalités
- ***L'abstraction***
  - Le traitement (abstrait) est défini en fonction des objets inférieurs

- **Récapitulation dur la conception détaillée**
  - éclatement des fonctions en unités de programme
  - interfaces entre les modules
  - description des données en entrée (origine, format, contraintes,…)
  - description des données en sortie (format, présentation, stockage, …)
  - description des traitements

E- Généralités sur UML

E-1 Diagrammes de cas d'utilisation

Les cas d'utilisation sont utilisés pendant explicitation et l'analyse des exigences pour représenter le fonctionnement du système. Les cas d'utilisation se concentrent sur le comportement du système à partir d'un point de vue externe. Ce diagramme permet donc de faire le point sur les besoins de l'utilisateur. Aucune compétence informatique n'est exigée.

Un cas d'utilisation décrit une fonction fournie par le système qui donne un résultat visible pour un acteur. C'est une action qu'il est possible de réaliser et que le système doit savoir gérer. Elle se représente avec une ellipse et utilise un verbe à l'infinitif.

Acteur : Il décrit une entité qui interagit avec le système (par exemple, un utilisateur, un autre système, ...). C'est donc une personne qui va interagir avec le système. Les acteurs sont à l'extérieur de la limite du système, tandis que les cas d'utilisation sont à l'intérieur de la limite du système.

Système: C'est un rectangle qui délimite acteurs et actions. Toutes les actions sont incluses dans le système.

Exemple :

Définir un diagramme de cas d'utilisation UML décrivant la fonctionnalité d'une simple montre. L'utilisateur de la montre peut soit consulter l'heure sur sa montre ou régler l'heure. Toutefois, seul le réparateur de montres peut changer la pile de la montre.

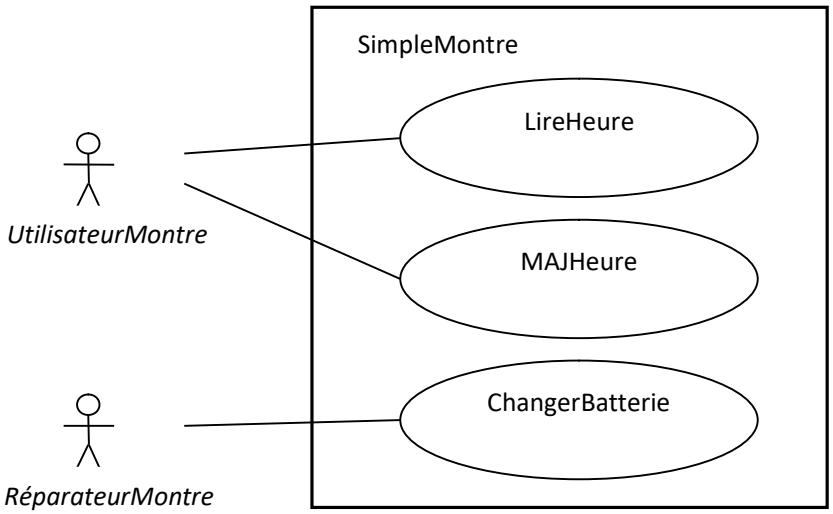


Diagramme des cas d'utilisation pour une simple montre

E-2 Diagrammes de classes

Nous utilisons les diagrammes de classes pour décrire la structure du système. Les classes sont des abstractions qui spécifient la structure et le comportement commun d'un ensemble d'objets. Les objets sont des instances de classes qui sont créées, modifiée, et détruites lors de l'exécution du système. Les objets ont un état qui comprend les valeurs de leurs attributs et leurs relations avec d'autres objets.

Les diagrammes de classes décrivent le système en termes d'objets, de classes, d'attributs, d'opérations et de leurs associations.

Exemple :

Elaboration d'un diagramme de classes décrivant les éléments de toutes les montres de la classe SimpleMontre. Ces objets montres ont tous une association vers un objet de la classe AppuiBouton, un objet de la classe Afficheur, un objet de la classe Heure, et un objet de la classe Batterie.

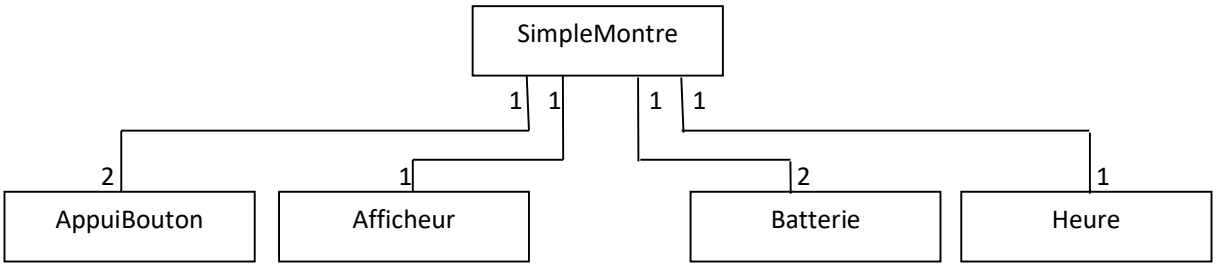


Diagramme de classe UML représentant les éléments de Simple montre

Les numéros sur les extrémités des associations indiquent le nombre de liens que chaque objet SimpleMontre peut avoir avec un objet d'une classe donnée. Par exemple, une SimpleMontre a exactement deux boutons, un afficheur, deux batteries, et une heure. De même, tous les objets AppuiButton, Afficheur, Heure, et Batterie sont associés à un seul objet SimpleMontre.

E-3 Diagrammes de séquence

Les diagrammes de séquence sont utilisés pour formaliser le comportement du système et de visualiser la communication entre les objets. Ils sont utiles pour l'identification d'objets supplémentaires qui participent dans les cas d'utilisation. Nous appelons les objets impliqués dans un cas d'utilisation des objets participant. Un diagramme de séquence représente les interactions qui ont lieu entre ces objets.

Exemple :
Soit le diagramme de séquence pour le cas d'utilisation mise à jour de l'heure (MAJHeure) de notre simple montre.

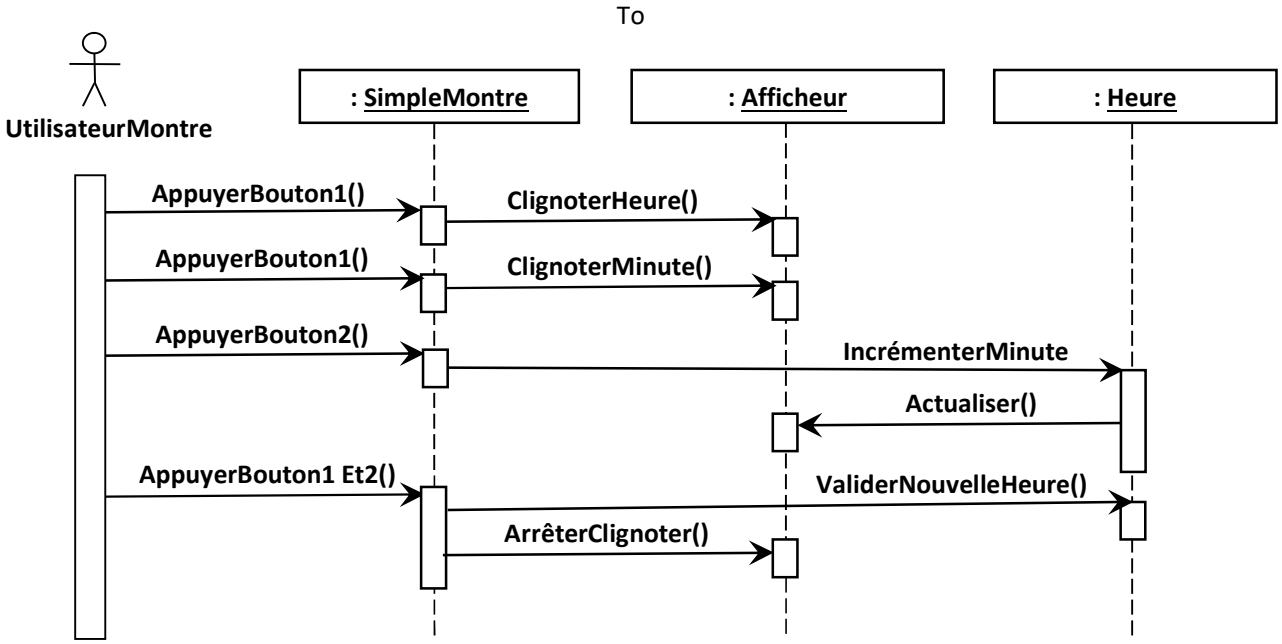


Diagramme de séquence

La colonne de gauche représente le scénario de l'acteur UtilisateurMontre qui initie le cas d'utilisation. Les autres colonnes représentent la chronologie des objets qui participent à ce cas d'utilisation. Les noms d'objets sont soulignés pour indiquer qu'elles sont des exemples (par opposition aux classes). Les flèches marquées représentent les stimuli qu'un acteur ou un objet envoie à d'autres objets. Dans ce cas, l'UtilisateurMontre appuie deux fois sur le bouton et une fois et sur le bouton 2 pour régler sa montre d'une minute d'avance. Le cas d'utilisation de MAJHeure se termine lorsque l'UtilisateurMontre appuie sur les deux boutons simultanément.

E-4 Diagrammes d'état-transition

Les diagrammes d'état-transition décrivent le comportement d'un objet individuel par un certain nombre d'états et de transitions entre ces états. Un état représente un ensemble de valeurs d'un objet. Une transition représente un état futur auquel l'objet peut se déplacer et les conditions associées à la modification de l'état.

Soit le diagramme d'état-transition pour le cas MAJHeure :

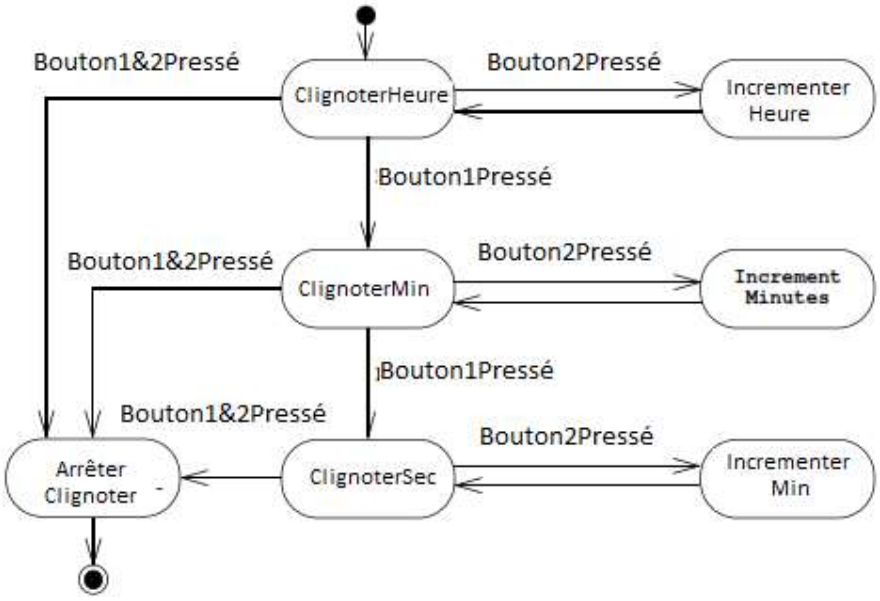


Diagramme d'Etat-Transition pour le cas d'utilisation MAJHeure

Ce diagramme représente une information différente du diagramme de séquence précédant. Le diagramme de séquence se concentre sur les messages échangés entre les objets à la suite d'événements extérieurs créés par des acteurs. Le diagramme d'état-transition se concentre sur les transitions entre états à la suite des événements externes pour un objet individuel.

Certains diagrammes peuvent comporter des transitions alternatives.

Exemple : Soit le diagramme d'état-transition relatif à une commande.

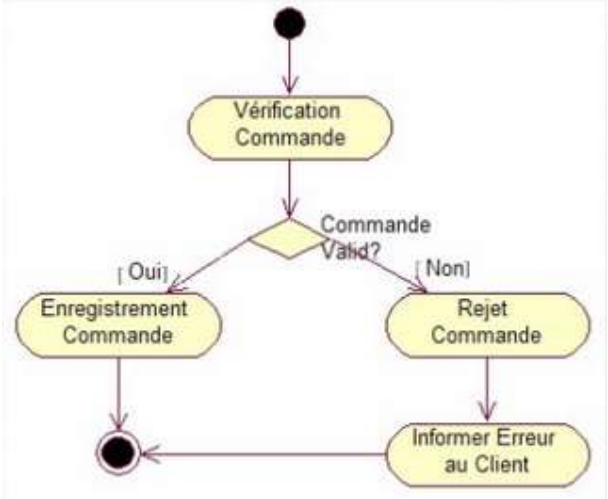


Diagramme d'Etat-Transition pour Commande

E-5 Diagramme d'activités

Un diagramme d'activités décrit un système en termes d'activités. Les activités sont les Etats qui représentent l'exécution d'un ensemble d'opérations. L'achèvement de ces opérations déclenche une transition vers une autre activité. Les diagrammes d'activités sont similaires aux diagrammes de flux en ce sens qu'ils peuvent être utilisés

pour représenter les flux de contrôle (c'est à dire, l'ordre dans lequel les opérations se déroulent) et les flux de données (c'est à dire, les objets qui sont échangés entre les opérations).
Exemple : Soit le diagramme d'activités représentant les activités liées à la gestion d'une livraison. Les rectangles arrondis représentent les activités; les flèches représentent les transitions entre les activités; les barres épaisses représentent la synchronisation des flux de contrôle. Le diagramme d'activité illustre que Contrôle-Quantité et Contrôle-Qualité ne peuvent être engagée qu'après que l'activité Réception-Livraison a été achevée. De même, l'activité Enregistrement-Livraison ne peut être lancée qu'après l'achèvement de Contrôle-Quantité et Contrôle-Qualité. Cependant, ces deux dernières activités peuvent se produire simultanément.

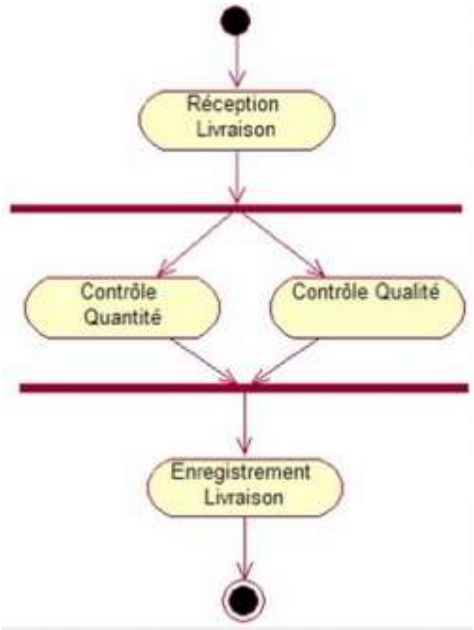


Diagramme d'activités UML de Livraison.

Les diagrammes d'activités représentent le comportement en termes d'activités et de leurs contraintes de précédence. La réalisation d'une activité déclenche une transition de sortie, ce qui à son tour peut déclencher une autre activité.

E-6 Diagramme de classes
Un diagramme de classe est utilisé pour représenter la structure d'un système en termes d'objets, de leurs attributs et leurs relations. Un diagramme de classes est donc une collection d'éléments de modélisation statiques qui montre la structure d'un modèle. Il fait abstraction des aspects dynamiques et temporels.

Pour un modèle complexe, plusieurs diagrammes de classes complémentaires doivent être construits.

On peut par exemple se focaliser sur :
- les classes qui participent à un cas d'utilisation (cf. collaboration),
- les classes associées dans la réalisation d'un scénario précis,
- les classes qui composent un paquetage,
- la structure hiérarchique d'un ensemble de classes.

Pour représenter un contexte précis, un diagramme de classes peut être instancié en diagrammes d'objets. Une association entre classes exprime une connexion sémantique bidirectionnelle entre deux classes.

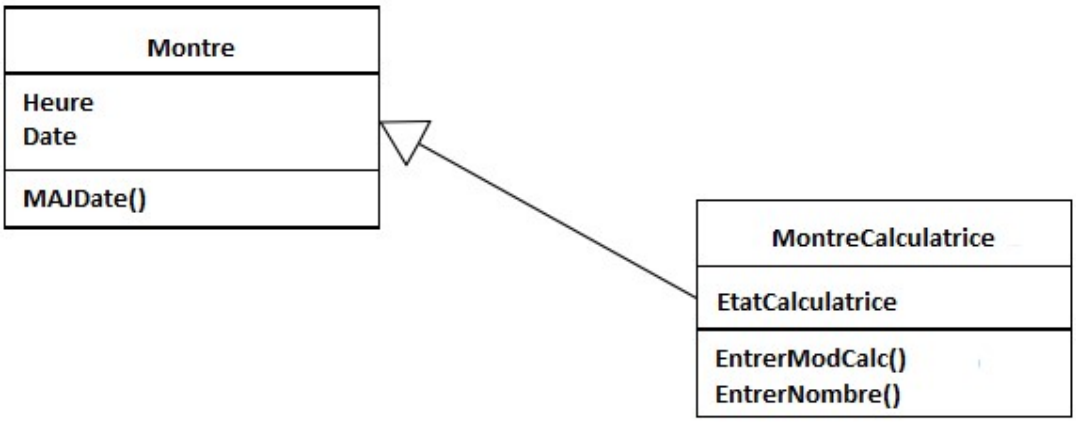


Diagramme de classes UML représentant deux classes: Montre et MontreCalculatrice.

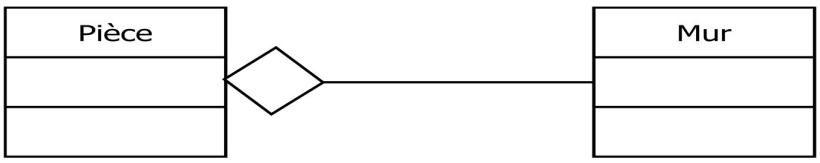
MontreCalculatrice est un raffinement de Montre, fournissant des fonctionnalités de la calculatrice normalement pas existantes dans les montres normales. Dans un diagramme de classes UML, les classes et les objets sont représentés comme des boîtes avec trois parties : La première partie représente le nom de la classe, la deuxième représente ses attributs et la troisième ses opérations. La deuxième et troisième parties peuvent être omises par souci de concision (brièveté).
Une relation d'héritage est schématisée par une ligne terminée par un triangle. Le triangle pointe vers la super-classe, et dont l'autre extrémité est fixée à la sous-classe.
Quand une généralisation ne sert qu'à des fins de modélisation d'attributs et d'opérations partagés, c'est à dire si la généralisation n'est jamais instanciée, elle est appelée ***classe abstraite***.

Un ***objet*** est une instance d'une classe. Un objet a une identité et stocke les valeurs d'attributs. Chaque objet appartient exactement à une classe. En UML, une instance est représentée par un rectangle avec son nom souligné.

Agrégation :
Une agrégation peut notamment (mais pas nécessairement) exprimer :

- qu'une classe (un "élément") fait partie d'une autre ("l'ensemble"),
- qu'un changement d'état d'une classe, entraîne un changement d'état d'une autre,
- qu'une action sur une classe, entraîne une action sur une autre.

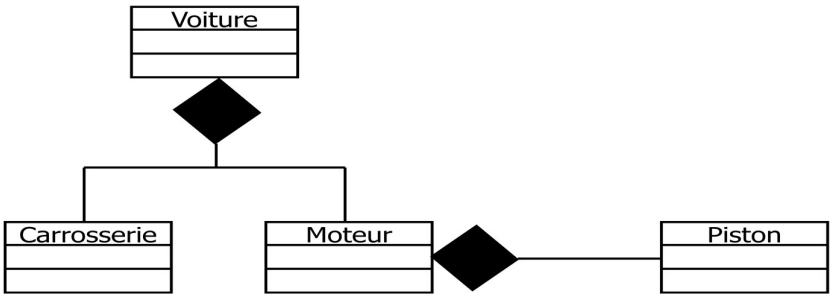
Exemple :



Composition :

- Cas particulier d'agrégation : contenance physique
- Représente une relation de type "composé / composant"
- Les cycles de vies des composants et du composé sont liés : si le composé est détruit (ou copié), ses composants le sont aussi
- A un même moment, une instance de composant ne peut être liée qu'à un seul composé

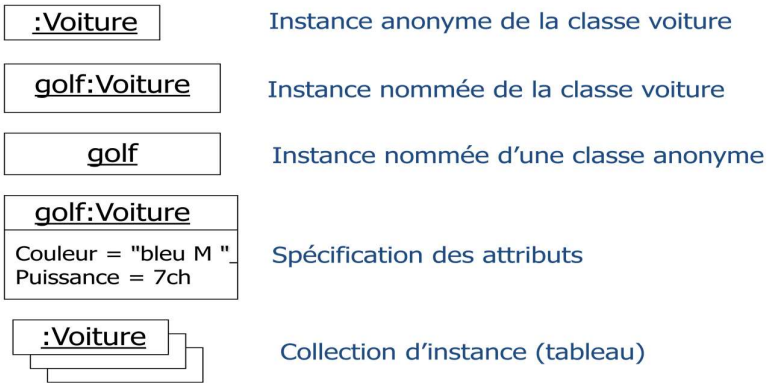
Exemple :



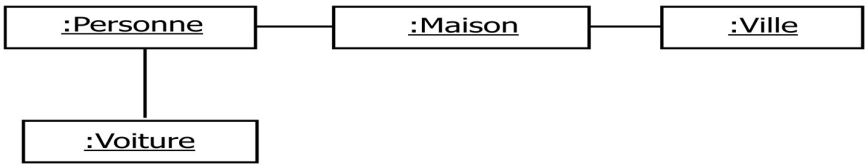
E-7 Diagramme d'objets

Un diagramme d'objets représente un ensemble d'objets et leurs liens. C'est une instance d'un diagramme de classes.

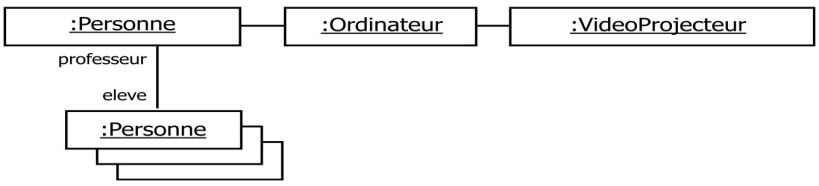
Les diagrammes d'objets sont des vues statiques des instances des éléments qui apparaissent dans les diagrammes de classes. Ils présentent la vue de conception d'un système, exactement comme les diagrammes de classes, mais à partir de cas réel ou de prototypes.



Exemple d'objets



Exemple 1 : Diagramme d'objets



Exemple 2 : Diagramme d'objets