Ministère de l'Enseignement Supérieur

****

Université de Carthage

****

Institut des Hautes Etudes Commerciales de Carthage

Atelier de Génie Logiciel : Document de conception détaillée du Domotique Casacontrol

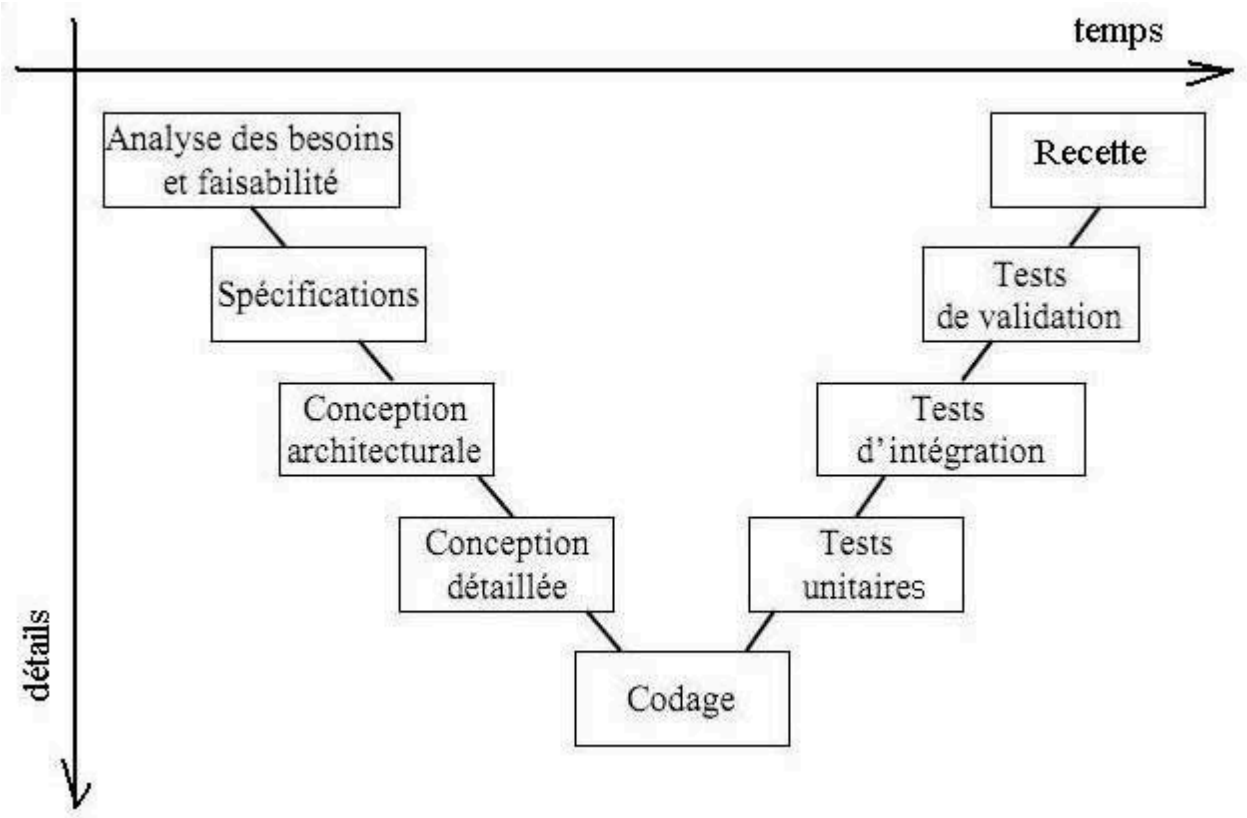
Réalisé par :Nouha ben nasr ,Salima Boudinar et Ines Mazgar

Date limite du rendu : 19/04/2024

Année universitaire 2023/2024

La conception détaillée est un document essentiel qui, s'il est bien étayé, nous permettra de restituer une réalisation de qualité.

L'objectif de ce document est de fournir la feuille de route exhaustive et détaillée nécessaire à la réalisation technique de votre projet.



1-Raffinement du diagramme de classe préliminaire :

- ***Identifier les classes et les relations :*** Vérifier et valider les classes déjà identifiées dans le diagramme de classe initial. Identifier de nouvelles classes si nécessaire en fonction des détails émergents.
- ***Spécifier les attributs et les méthodes :*** Examiner les attributs et les méthodes de chaque classe pour vous assurer qu'ils sont complets et précis. Ajouter des détails supplémentaires si nécessaire.
- ***Préciser les relations entre les classes :*** Affiner les relations entre les classes en définissant plus précisément le type de relation (agrégation, composition, association) et en spécifiant les cardinalités si elles ne sont pas claires.
- ***Gérer l'héritage :*** Si des hiérarchies de classes sont présentes, il faut clarifier les rôles et les responsabilités de chaque classe héritée. Il faut assurer que l'héritage est justifié et que les relations de sous-classe/super classe sont correctement définies.

- ***Documenter les contraintes et les règles métier :*** Ajouter toute contrainte ou règle métier importante qui doit être respectée dans le diagramme de classe.
- ***Vérifier la cohérence et la complétude :*** Assurer que le diagramme de classe est cohérent avec les autres documents de conception et qu'il est complet, couvrant tous les aspects importants du système.
- ***Finaliser la documentation :*** Une fois que le diagramme de classe a été raffiné et validé, assurer de documenter toutes les décisions prises et les détails ajoutés pour référence future.

```
@startuml DiagrammeDeClasse1

class Utilisateur {
  + idUtilisateur: integer
  + nom: string
  + prenom: string
  + login: string
  + motDePasse: string
  + email: string
  + Constructeur()
  - ajouterEquipementElectrique()
    - configurerEquipementElectrique()
    - supprimerEquipement()
    - consulterEtatEquipementElectrique()
    - piloterEquipementElectrique()
    - activerEquipement()
    - desactiverEquipement()
    - sAuthentifier()
    - afficherListeEquipements()
}

class actionneur {
  + Id: integer
  + Nom: string
  + Position: string
  + Description: string
  + Constructeur()
}

abstract class Capteur {
  - idCapteur: integer
  - nomCapteur: string
  - positionCapteur: string
  - descriptionCapteur: string

  - EnvoyerSignal()
  - DebutExecution()
```

```
SMART_HOME "1"   -- "1.." Piece : posséder
SMART_HOME "1"   -- "0.." Equipement : contrôler
SMART_HOME "1..*"  -- "1.." Utilisateur : posséder
SMART_HOME "1"   -- "1.." Box_Domotique : posséder
SMART_HOME "1"   -- "1.." Alerte : gérer
Utilisateur "1..*"  --> "0..*" Alerte : recevoir



Box_Domotique "1"  -- "1.." Capteur : côntroler
Capteur "1" --> "1..*" Alerte : Déclencher
Box_Domotique "1" -- "1.." actionneur : côntroler
Box_Domotique "1" --> "1..*" Equipement : Contrôler
Capteur "1..*" --> "1..1" Equipement : Envoyer des données environnementales
Equipement "1..*" --> "1..1" Piece : Se situer dans
actionneur "1..* " --> "1..1" Equipement : agir sur
@enduml
```

```
  - FinExecution()
}

class Cap_Conso extends Capteur {
  - DetecterConsommation()
  + Constructeur()
}

class Cap_Eclairage extends Capteur {
  - DetecterEclairage()
  + Constructeur()
}

class Cap_Temperature extends Capteur {
  - DetecterTemperature()
  + Constructeur()
}

class Detecteur_Gaz extends Capteur {
  - DetecterGaz()
  - Constructeur()
}

class Cap_Incendie extends Capteur {
  - DetecterIncendie()
  + Constructeur()
}

class Cap_Mouvement extends Capteur {
  - DetecterMouvement()
  + Constructeur()
}

class Box_Domotique {
  - idBoxDomotique: integer
  - nomBoxDomotique: string
  + Constructeur()
```

```
    - gererEquipements()
    - traiterDonneesEnvironnementales()
    - envoyerCommandes()
    - gererAlertes()
    - communiquerAvecUtilisateur()
    - gererStatistiques()
}

class SMART_HOME {
  + idUtilisateur: integer
  + nom: string
  + personne: Personne
  + actionneur: actionneur
  + capteur: Capteur
  + boxDomotique: Box_Domotique
  -  ControlerLacces()
  - ControlerLeclairage()
  - ControlerVocale()
  - ControlerVolume()
  - ControlerTemperature()
  - SurveillerLaMaison()
  - EconomiserEnergie()
}

abstract class Equipement {
  - idEquipement: int
  + nomEquipement: string
  + typeEquipement: string
  + marqueEquipement: string
  + modeFonctionnement: string
  + piece: Piece
  toString(): string
}

class Lampe extends Equipement {
  + puissance: int
  + couleur: string
```

```
  - allumer(): void
  - eteindre(): void
  - reglerTemperature(nouvelleTemperature: float): void
}

class SystemeAlarme extends Equipement {
  + codeAlarme: string
  + etatAlarme: boolean
  - activer(): void
  - desactiver(code: string): void
}

class Porte extends Equipement {
  + etatPorte: boolean
  - ouvrir(): void
  - fermer(): void
}

abstract class Piece {
  - idPiece: int
  + nomPiece: string
  + etage: int
  + superficie: float
  + equipements : array
  - afficher_equipements(): void
}



class Alerte {
  - idAlerte: integer
  - nomAlerte: string
  - dateAlerte: date
  - typeAlerte: string
  + Constructeur()
}
SMART HOME "1"    -- "1.." Piece : posséder
```

Figure 1 : code PlantUML à travers VS Code

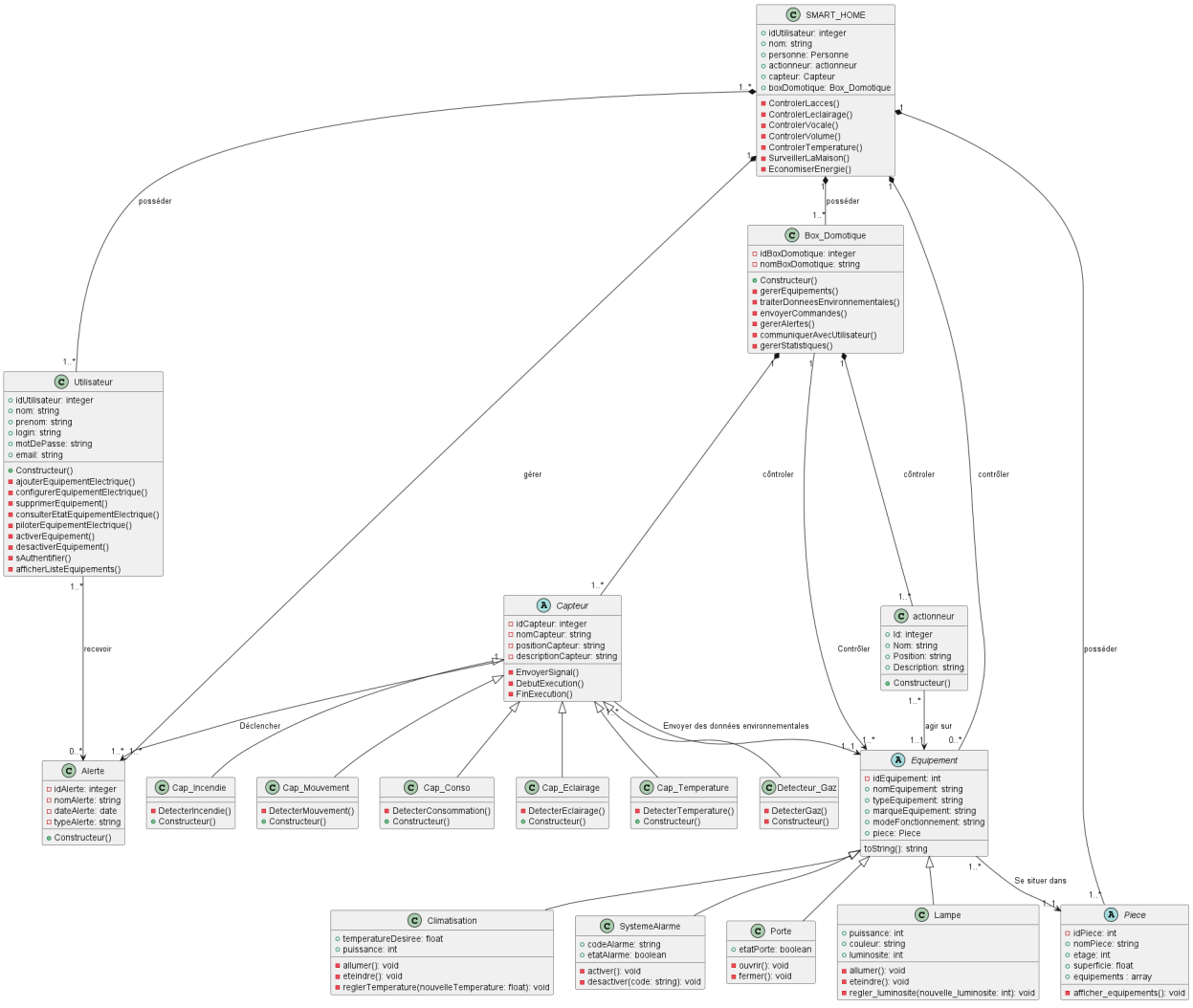


Figure 2 : Diagramme de classe raffiné généré

2. Le diagramme de machines à état :

Ce diagramme d'états-transitions modélise le comportement d'un système de gestion d'un système domotique, en particulier le passage entre différents états de fonctionnement d'un équipement.

- **Éteint :**

  C'est l'état initial du système lorsque celui-ci est éteint.

  Le système peut rester dans cet état jusqu'à ce qu'une action externe déclenche le démarrage.

- **Démarrage :**

  Lorsque la fonction *Debut_execution()* est invoquée, le système passe de l'état Eteint à l'état Démarrage.

  Dans cet état, le système attend généralement une action de programmation de la part de l'actionnaire.

- **EnMarche :**

  Lorsque l'actionnaire a programmé le système et que la fonction *Programmer_actionnaire()* est appelée, le système entre dans l'état En Marche.

  Dans cet état, le système est opérationnel et peut exécuter ses fonctions habituelles, telles que l'envoi de signaux.

- **Fin :**

  Lorsque la fonction *Fin_execution()* est appelée, le système passe de l'état En Marche à l'état Fin.

  Cet état indique généralement la fin de l'exécution du système ou l'achèvement d'une tâche spécifique.

Après avoir terminé ses tâches, le système peut être réinitialisé en revenant à l'état initial, Éteint.

diagrammedeclasse.plantuml | diagrammeetattransition.plantuml ●

diagrammeetattransition.plantuml > {} Diagramme d'état de transition

```
@startuml Diagramme d'état de transition


state Eteint  #Pink {
}
state Demarrage #SkyBlue  {

}

state EnMarche #SkyBlue {

}

state Fin #pink {
}



Fin : Fin_Execution()
Demarrage : Debut_Execution()
EnMarche : Envoyer_Signal()

[*] --> Eteint
 Eteint->Demarrage : Démarrer
 Demarrage->Eteint : Annuler
  Demarrage -> EnMarche : Programmer actionnaire
  EnMarche -> Fin : Mise à jour de l'état
Fin --> [*]
@enduml
```

Figure 3 : code PlantUML à travers VS Code

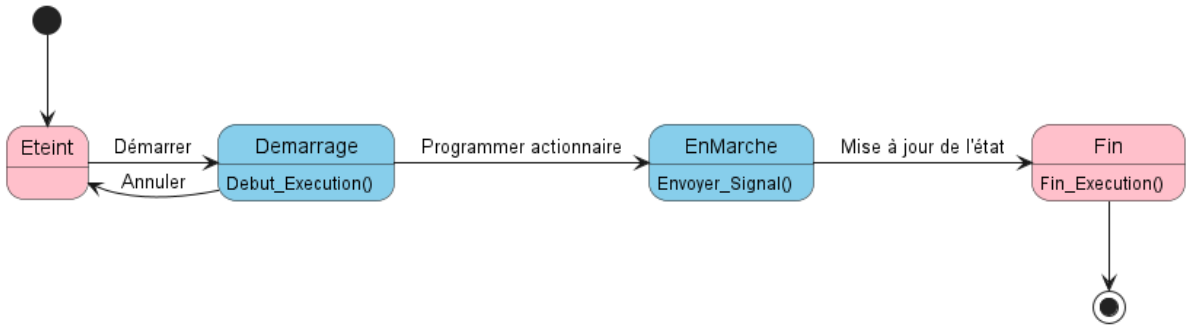


Figure 4 : Diagramme d'état de transition