CAHIER DES CHARGES

SUR LA REALISATION D'UN LOGICIEL DE DOMOTIQUE

OBJET DU DOCUMENT

HOMESOLUTION est une jeune entreprise caennaise née en 2012 suit au regroupement de huit ingénieurs prometteurs de l'ENSICAEN.

Son domaine d'application correspond à la technologie qui vise à piloter les différents équipements d'une maison et d'automatiser certains processus quotidien d'une maison ou d'un appartement. Cette technologie est appelé la « domotique ». HomeSolution par sa qualité de prestation et son innovation dans le domaine fait partie des leaders européens.

Le logiciel devra avoir répondu aux différentes exigences du client.

DOMAINE D'APPLICATION

Le présent Cahier des Charges a pour objet de définir les conditions dans lesquelles la société s'engage à réaliser les fonctionnalités du logiciel définies dans le présent Cahier des Charges.

SIGNATAIRES

Rédacteur(s) : MOISSON Florent, HINKATI Privel, MARGINIER Julien, COURTIOL Alexandre, CARITTE Micky, ALI FDAL Yousra, MICHELS Antoine, DUBOIS Jérémy – Ensicaennais

Vérificateur(s) : VERNOIS Sylvain – Client

Table des matières

PRÉ ETUDE 1

OBJET 2

EXIGENCES FONCTIONNELLES MINIMALES ATTENDUES DU FRAMEWORK 2

LES CONTRAINTES SUR LA GESTION DU PROJET 2

MOYENS MIS EN ŒUVRE POUR LE PROJET 2

ETUDE DU PROJET 4

PRESENTATION DU PROJET 5

CINEMATIQUE ET AUTOMATISME 5

CONTROLE, SUPERVISION ET INTELLIGENCE DU SYSTEME 5

SPECIFICITES FONCTIONNELLES 5

SCHEMA DE LA BASE DE DONNEES 5

ORGANISATION DU PROJET 4

REPARTITION DES TACHES 5

OUTILS DE GESTION DE PROJET 5

DEFINITION DES BESOINS 5

DESCRIPTION DE LA SOLUTION 5

MODELISATION DU FRAMEWORK 5

ANNEXES 4

ANALYSE DE LA QUALITE DU CODE PAR SONAR 5

Pré-étude

A. OBJET

Le présent Cahier des Charges (CDC) a pour but de définir les conditions dans lesquelles le logiciel SMARTHOME s'engage à répondre.

B. EXIGENCES FONCTIONNELLES MINIMALES ATTENDUES DU FRAMEWORK

Un logiciel de domotique est destiné à contrôler les équipements d'une maison ou d'un appartement. Il devra présenter au minimum les fonctionnalités suivantes :

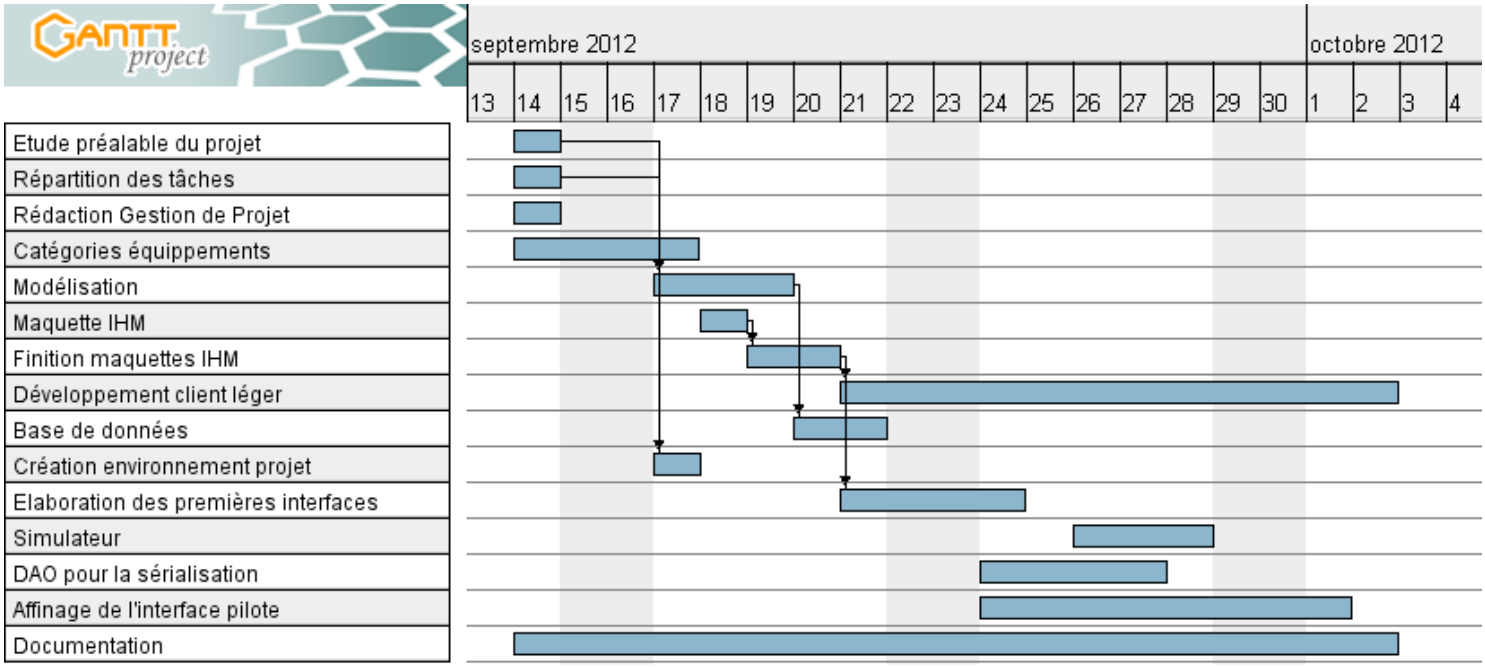
- Configurer une maison particulière.
- Visualiser la configuration de la maison.
- Contrôler les équipements individuellement : fermer la porte d'entrée, allumer un lecteur DVD.
- Consulter l'état des équipements : la lumière est bien éteinte, la porte du garage est bien fermée.
- Créer, modifier ou supprimer des scénarios.
- Ajouter ou supprimer un nouvel équipement.

C. LES CONTRAINTES SUR LA GESTION PROJET

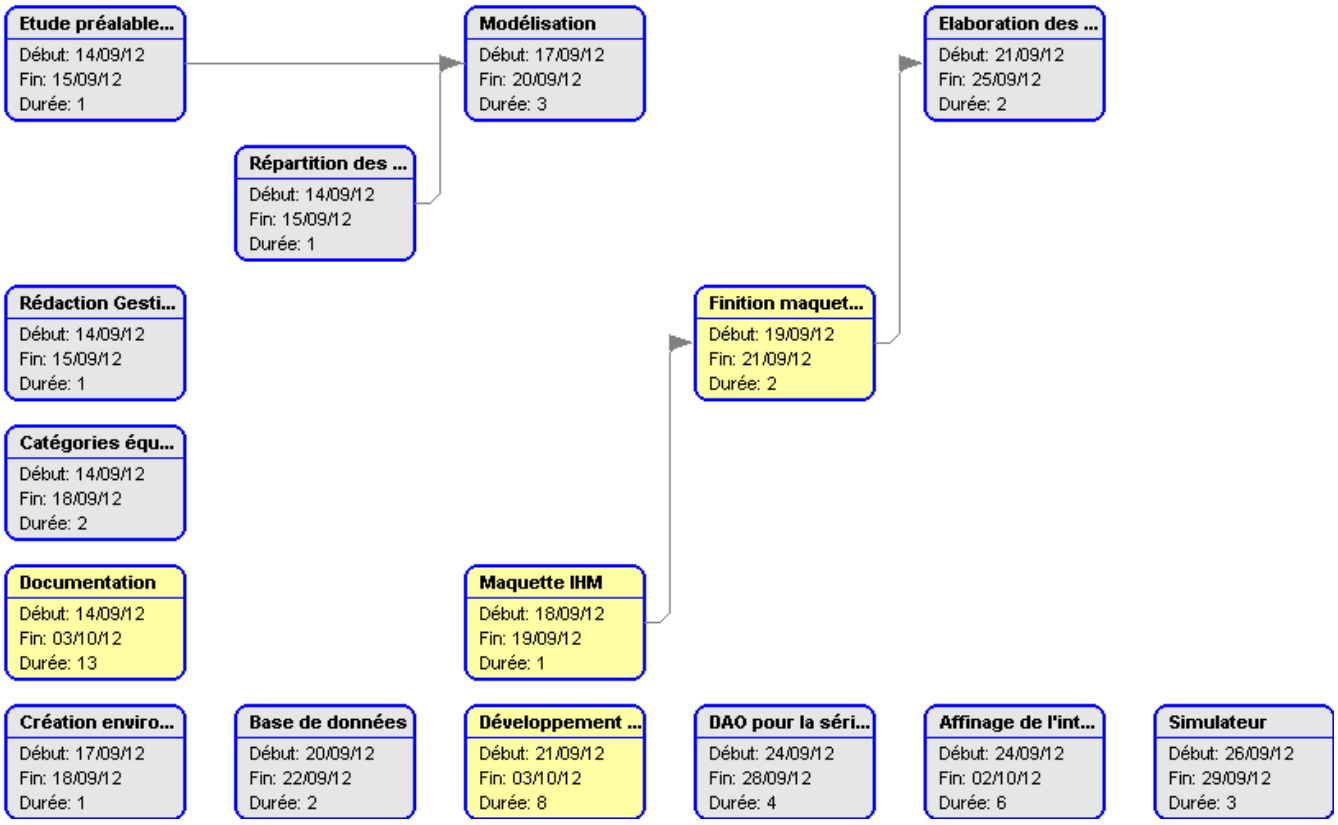
La réalisation du projet est soumise à quelques restrictions. La première correspond à l'utilisation d'une méthode de développement de type processus unifié. Nous avons donc choisi de mettre en place notre projet en respectant une phase d'**Etude d'opportunité** dans un premier temps, dans un second une phase d'**Elaboration** qui consistera à la conception de l'architecture du projet. S'ensuivra une phase de **Construction** de l'application en elle-même et on finira avec une phase de **Transition** qui nous permettra de vérifier la validité de l'application et ensuite de l'ensemble du projet.

D. MOYENS MIS EN ŒUVRE POUR LE PROJET

1. Diagramme de GANTT



2. Diagramme de PERT



Etude du projet

A. PRESENTATION DU PROJET

1. Rappel de la domotique

"La domotique est l'ensemble des techniques de l'électronique, de physique du bâtiment, d'automatisme, de l'informatique et des télécommunications utilisées dans les bâtiments et permettant de centraliser le contrôle des différents applicatifs de la maison (système de chauffage, volets roulants, porte de garage, portail d'entrée, prises électriques, etc.)." d'après wikipédia.
Elle peut être câblé ou par ondes radio et coordonne le fonctionnement de différents types d'équipements ménagers, de travail, de loisir…

2. Définition de l'UC et ses composants avec leurs fonctions

L'unité centrale constitue le cœur du système, notamment car il est le composant qui contrôle les différents équipements connectés (Portails, alarme, éclairage...).
Cette unité est déployée sur un serveur lié à Internet où tous les équipements utilisés seront branchés.
Les rôles de l'unité centrale sont tels que:
- Piloter individuellement chaque équipement.
- Définir des enchaînements d'actions choisis par l'utilisateur.
- Programmer des exécutions d'enchaînement d'actions

B. CINEMATIQUE ET AUTOMATISME

Notre système sera composé de deux parties:

- Partie commande

- Partie opérative

L'utilisateur qui fera fonctionner le système donnera des consignes à la partie commande, matérialisée par l'unité centrale, qui traduira celles-ci afin qu'elles soient exécutées par la partie opérative (les équipements branchés).
Une fois le scénario exécuté, les équipements utilisés enverront un signal à l'unité centrale afin qu'elle communique à l'utilisateur la bonne réalisation de l'action.

C. CONTROLE, SUPERVISION ET INTELLIGENCE DU SYSTEME (INTERACTION PAR INTERCONNEXION A DES SERVICES)

Le but de SmartHouse est d'assurer une gestion optimale du confort des utilisateurs, en leur permettant d'interagir (ou de lancer un scénario), de n'importe où, avec les différents appareils branchés sur le réseau.
L'interface de pilotage est une application liée au réseau local et connectée directement à l'unité centrale qui permet de créer des maisons et positionner les équipements installés.

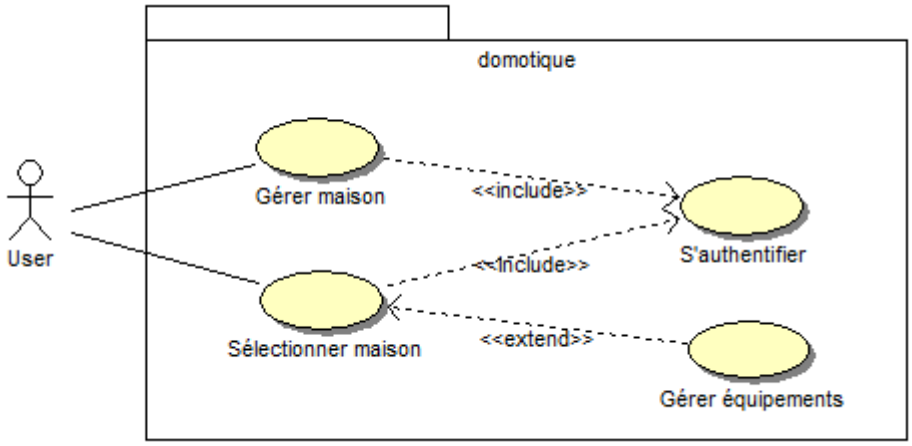
D. SPECIFICITES FONCTIONNELLES

Le rôle de notre système de domotique est la gestion et le contrôle des appareils branchés à l'unité centrale afin d'offrir un confort d'utilisation au client.
Les spécifications fonctionnelles requises sont telles que :

- la configuration et la visualisation d'une maison.
- Le contrôle des équipements séparément.
- Surveillance des états des appareils.
- Ajout des scénarios (Suite d'actions)
- Ajout/Suppression des équipements.

1. Diagrammes de cas d'utilisation

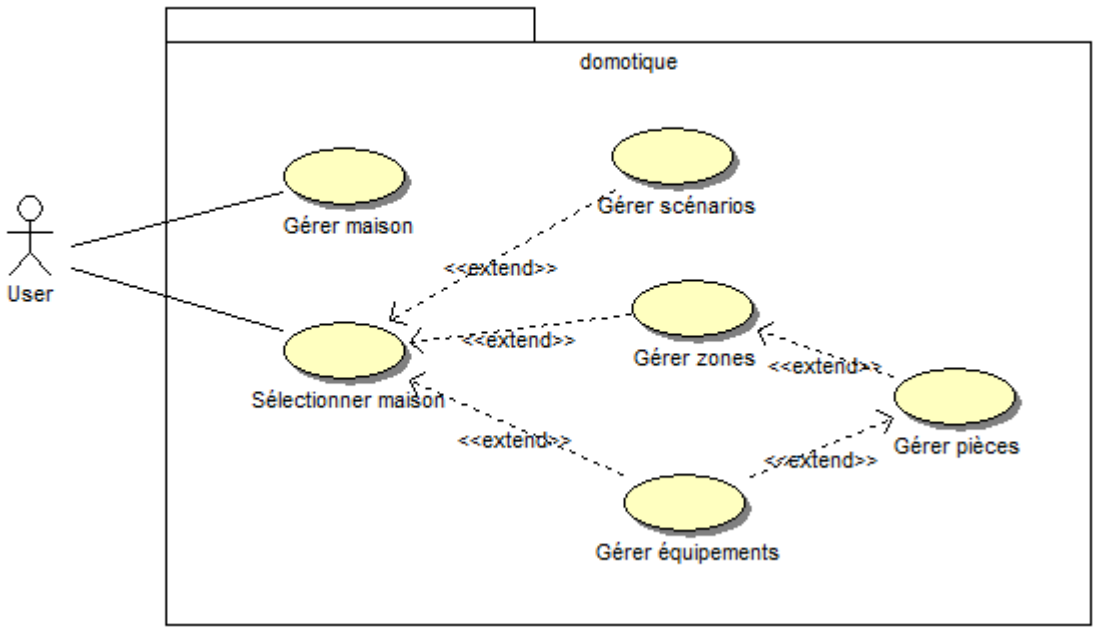
1.1 Cas d'utilisation haut niveau : Point d'entrée du système



Commentaire :

L'utilisateur par le biais d'une interface peut accéder au système SmartHouse après authentification. Cette authentification lui permet d'accéder à la gestion de la maison (Gestion des scénarios et des équipements).

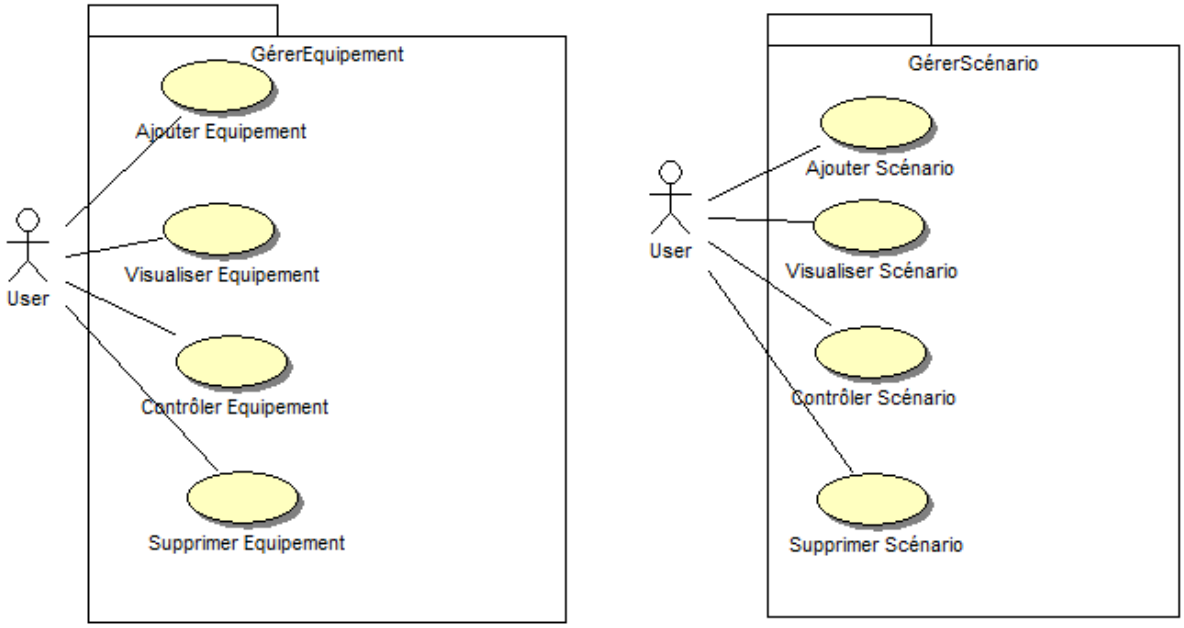
1.2 Cas d'utilisation : dépendances des actions de gestion



Commentaire :

Ce cas d'utilisation correspond à toute la gestion qui entoure une ou des maisons. Un utilisateur gère ses maisons (ajout / suppression) ou sélectionne une maison déjà enregistrée. Cette sélection permet de consulter, piloter et configurer la maison en gérant ses zones et ses équipements. La gestion des zones permet d'accéder à la gestion des pièces ainsi que la gestion des équipements de celle-ci (aussi accessible depuis la gestion de la maison pour tous les équipements). La gestion des scénarios est aussi disponible depuis la gestion de la maison.

1.3 Cas d'utilisation : Gérer équipements/ Gérer Scénarios



Commentaire :

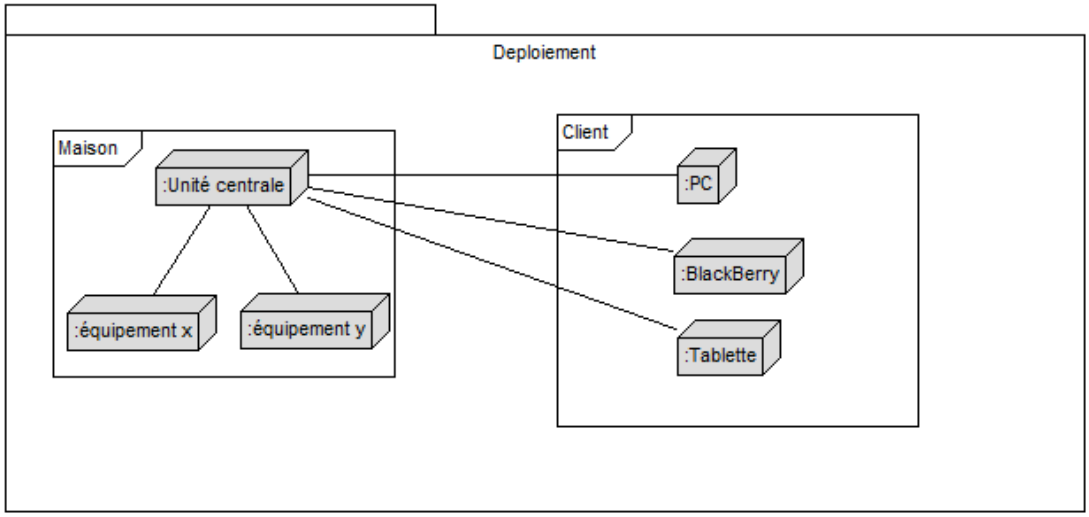
Gestion des équipements

Ce cas d'utilisation correspond aux différentes actions disponibles pour l'utilisateur en termes d'équipement. Grâce à l'application finale, il peut donc ajouter un équipement, visualiser, contrôler et supprimer un équipement de son choix.

Gestion des scénarios

Ce cas d'utilisation correspond aux différentes actions disponibles pour l'utilisateur en termes de scénarios. Grâce au logiciel, il peut donc ajouter un scénario, le visualiser, le contrôler ou le supprimer.

2. Diagramme de déploiement

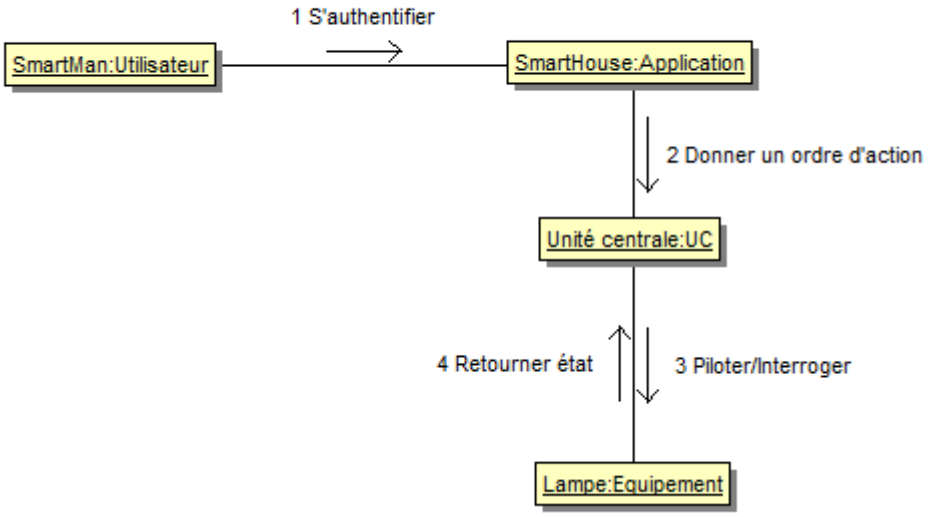


Commentaire :

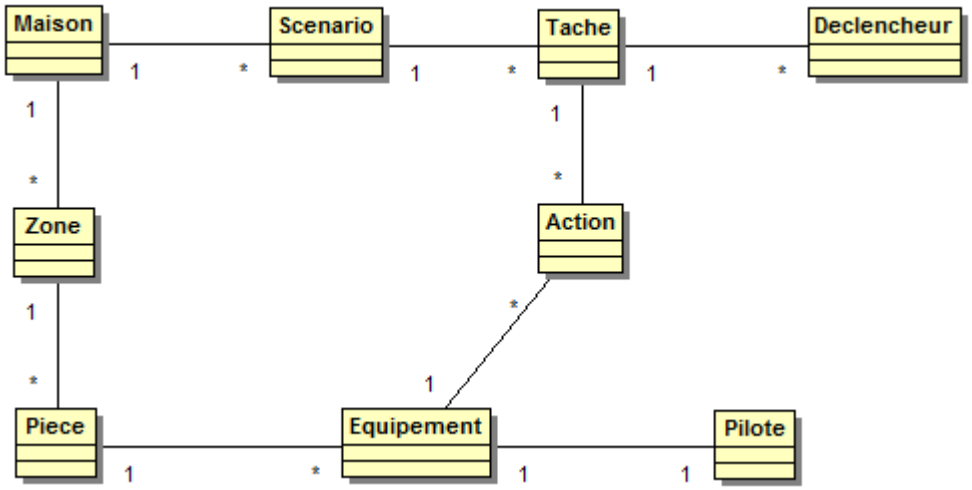
Nous avons regroupé les entités de notre système SmartHouse selon leur rôle dans l'architecture de l'application tel que :

Le fragment Serveur contient la base de données et le serveur web qui est relié au fragment Maison et Client.

3. Diagramme de collaboration



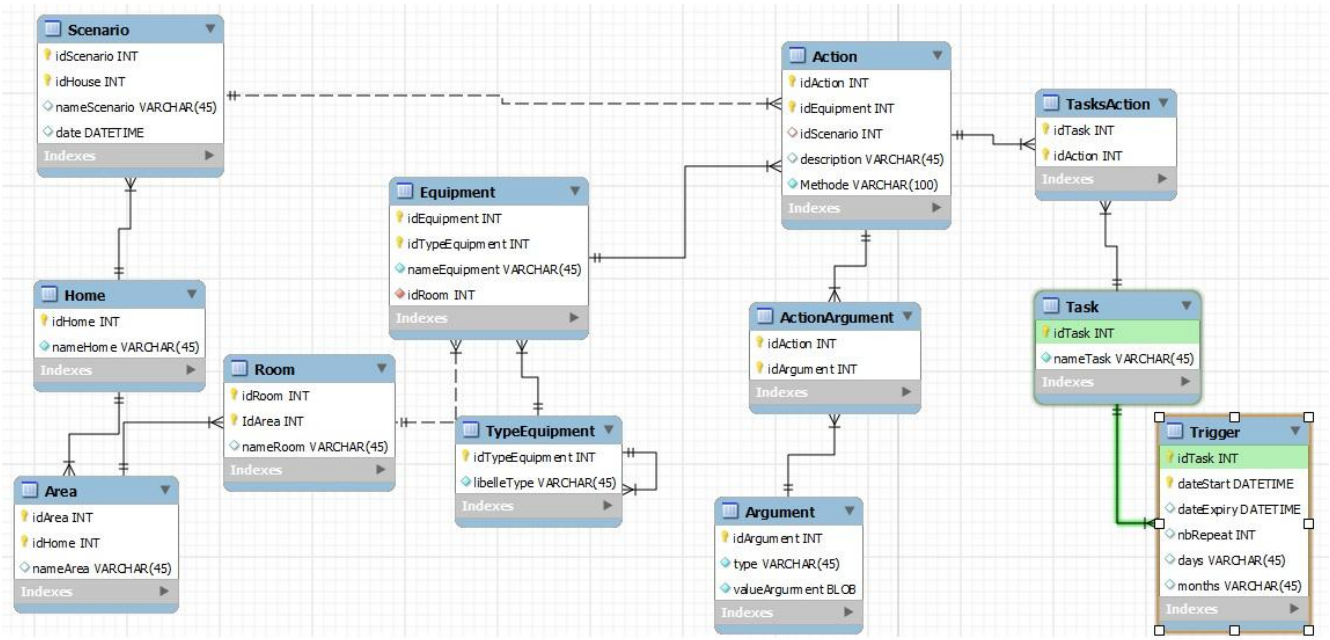
4. Diagramme de classe du domaine



Commentaire :

Ce diagramme de classes comporte les entités principales recensées dans le modèle au début du projet. Ce diagramme a servi de base pour la définition du framework lors de la phase de développement.

E. *SCHEMA DE LA BASE DE DONNEE*



Une réflexion sur la conception d'une base de données a été entreprise dès le début du projet. Cette base de données nous offrait une couche de persistance performante et devait tourner sur un moteur SQLite pour des questions pratiques. Au vu d'un risque de retard du projet, nous avons arrêté de développer cette couche de persistance au bénéfice de la "sérialisation" Java. A travers ce qui suit, nous tenons à expliquer notre réflexion pour présenter une autre implémentation possible de notre pattern DAO (Data Access Object) qui nous permet de nous abstraire de notre couche de persistance dans le framework.

Au vue des différentes exigences structurelles émises par le client à travers son cahier des charges, des tables représentant les objets suivants ont été créées : Home (pour les maisons), Area (pour les zones), Room (pour les pièces) et Equipment (pour les équipements).

Pour accompagner ces éléments structuraux, le pilotage s'intègre par la table **"Action"** pour répertorier toutes les actions réalisées sur chaque équipement. A chaque action correspond une méthode Java avec des arguments ; ceci est représenté par la table **"Argument"** et **"ActionArgument"** (pivot entre les tables Action et Argument afin d'associer 1 à n arguments à 1 action). Le pilotage pour des scénarios se complète par les éléments suivants :

- La table **"Task"** permet définir un ensemble d'actions.
- La table **"TaskAction"** permet de faire le lien entre la table **"Task"** et la table **"Action".**
- La table **"Trigger"** nous permet d'affecter une date/heure de début et date/heure de fin à une tâche. Elle nous permet également de déterminer la fréquence de répétition d'une tâche.

Organisation du projet

A. DIVISION DES TACHES

Après une lecture minutieuse du projet, nous avons tout d'abord déterminé un chef de projet pour mener et diriger l'équipe. Ensuite, nous nous sommes convenus d'une organisation par rapport aux différentes tâches au sein de l'équipe. Nous avons choisi d'affecter les personnes par domaine de compétence dans un premier temps et dans un second sur les tâches qu'ils restaient à effectuer.

Voici l'organisation que nous avons mise en place :

Rôle et tâches attribuées	Personnes affectées
Chef de projet	MOISSON Florent
Réalisation et rédaction du cahier des charges	HINKATI Privel, ALI FDAL Yousra
Réalisation des différents diagrammes de conception	MARGINIER Julien, MICHELS Antoine, HINKATI PRIVEL, ALIFDAL Yousra
Développement du framework	MARGINIER Julien, MOISSON Florent
Développement du client	MICHELS Antoine, COURTIOL Alexandre, DUBOIS Jérémy
Développement du simulateur	DUBOIS Jérémy, MOISSON Florent
Organisation et mise en place de la base de données	CARITTE Micky, HINKATI Privel
Étude sur les IHM	COURTIOL Alexandre,MICHELS Antoine
Rédaction et réalisation des tests d'intégration	DUBOIS Jérémy
Planification (GANTT & PERT)	ALI FDAL Yousra

A la suite des attributions des différentes tâches, nous avons réalisé un diagramme de

Gantt afin d'établir un échéancier des étapes du projet par personne et pour toute l'équipe.

B. OUTILS DE GESTION DE PROJET

1. Les moyens de collaboration utilisés

- Google Drive avec sa fonction de partage en temps réel et Google Document pour la collaboration en direct nous a permis d'être organisé. Toutes les personnes du groupe étant en possession des dernières versions des différents documents, cela nous a permis d'être efficace sur le travail à réaliser. Ces types d'outil nous permettant en plus d'avoir une collaboration participative, chacune des personnes du groupe pouvait donner son avis et donc apporter ses améliorations.
- Google Agenda pour le report des tâches effectuées.
- Subversion, un gestionnaire de sources, nous a permis de développer en collaboration tout en ayant un historique des créations, modifications et suppressions.
- La communication d'après le modèle de Shannon et Weaver nous a permis de dialoguer par le plus simple concept de l'homo sapiens qui est la voix.

2. Les outils de travail utilisés par les différentes tâches

- L'organisation des tâches et l'épluchage du sujet de projet ont été réalisé avec le logiciel MindView.
- Pour la base de données nous avons choisi d'utiliser l'outil MySQL Workbench car c'est un outil pratique et puissant et maîtrisé par la majorité de l'équipe.
- Pour toute la partie programmation (framework, simulateur et client), le client imposant le langage Java, nous avons utilisé l'environnement de développement Eclipse avec différentes extensions (Subclipse, WindowBuilder, Maven). La capacité Maven a été intégrée à tous nos projets pour simplifier l'importation des dépendances.
- Bouml a été utilisé pour la réalisation des différents diagrammes par conception direct ou par reverse-engineering.
- Extension Chrome Balsamiq Mockups pour les maquettes IHM.

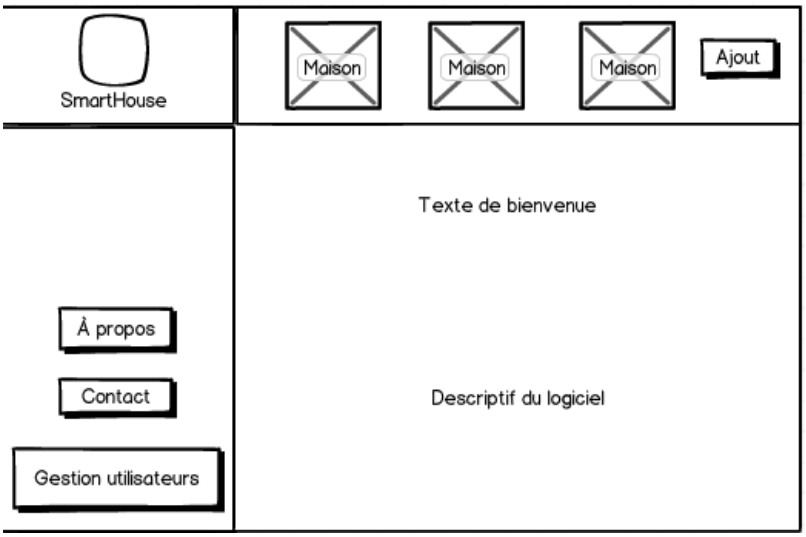
- Gantt Project pour la réalisation des diagrammes Gantt.

C. DEFINITION DES BESOINS

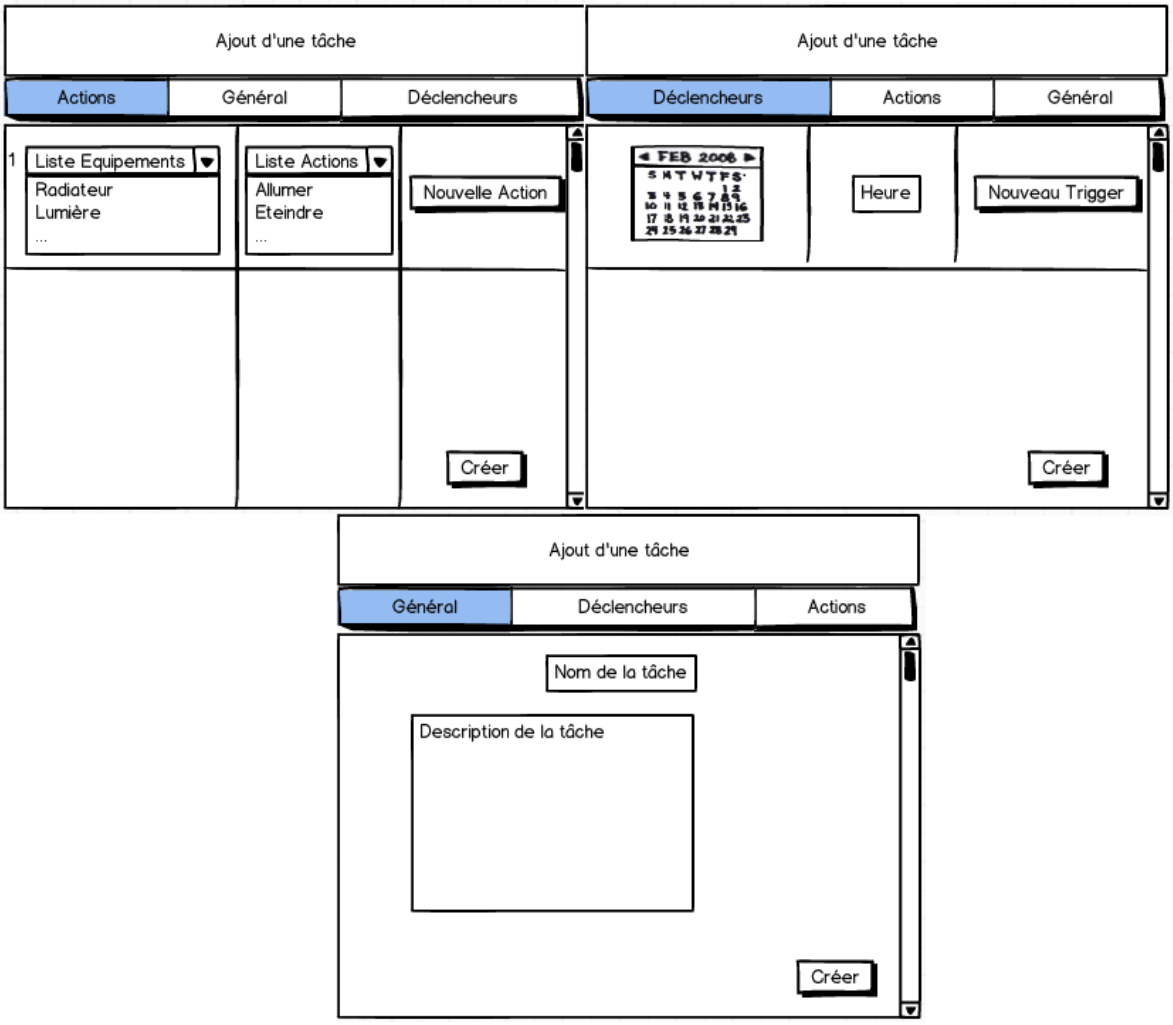
1. Maquette d'IHM Logiciel Client

Voici les principales maquettes réalisées en amont du développement afin de se donner des idées plus ou moins fixes sur l'IHM du logiciel client.

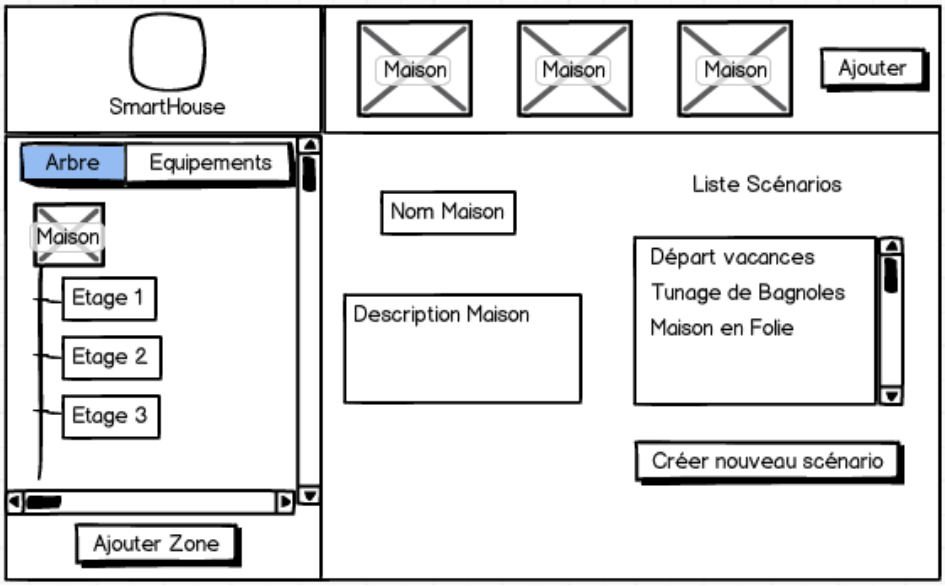
Fenêtre d'accueil de l'application



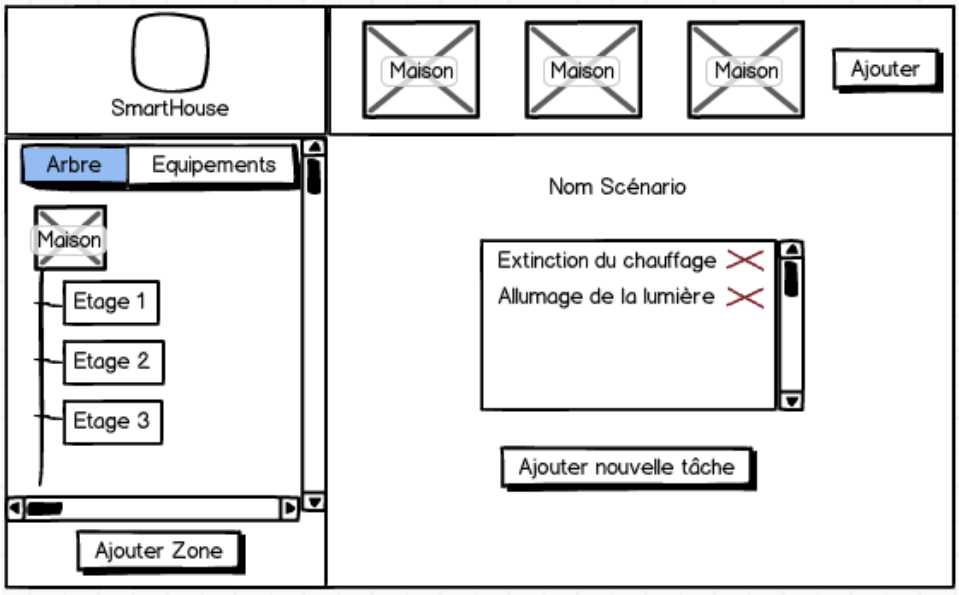
Fenêtre d'ajout d'une tâche à un scénario (onglet Général, onglet pour la liste des actions, onglet pour les déclencheurs)



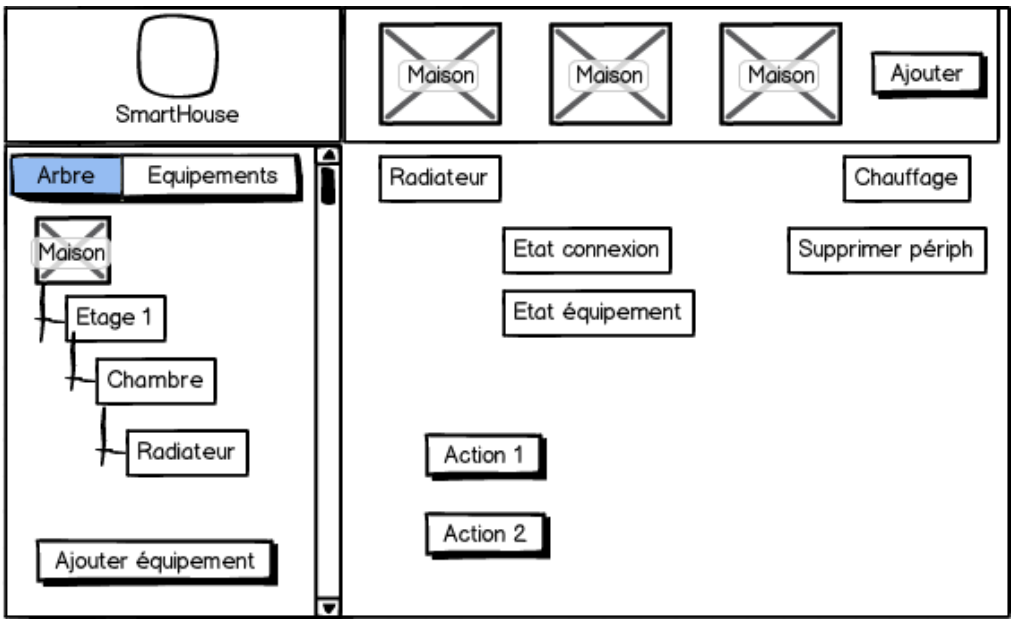
Fenêtre de sélection d'une maison (arbre à gauche, description maison, liste des scnéraios)



Fenêtre de sélection d'un scénario pour une maison donnée



Fenêtre de sélection d'un équipement



Ces fenêtres n'ont pas été implémentées telles quelles dans le développement qui s'en est suivi, mais ont fortement servi (gain de temps dans le développement).

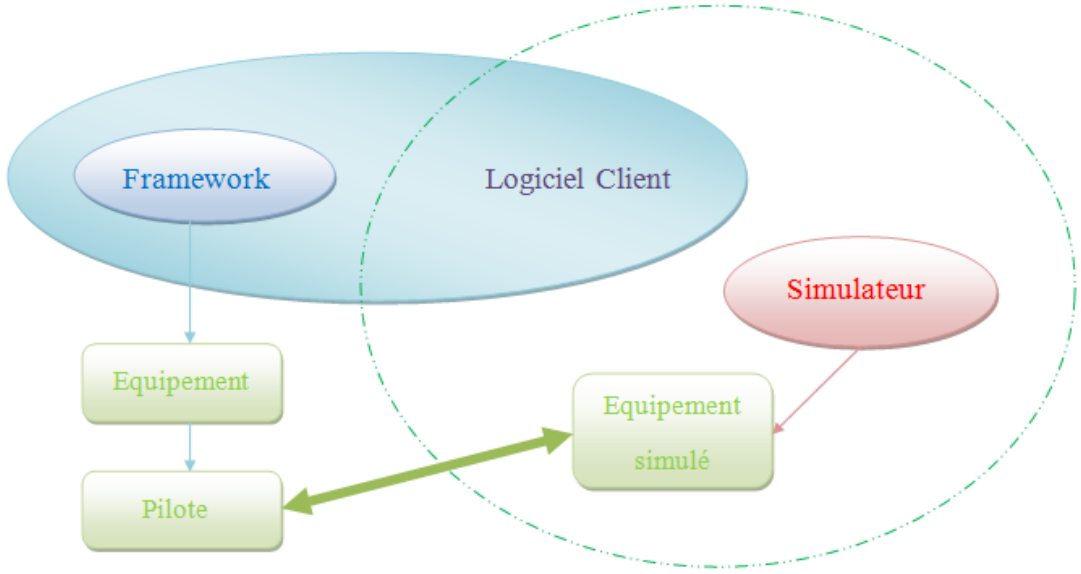
D. DESCRIPTION DE LA SOLUTION

1. Présentation de l'architecture générale

L'architecture de la solution développée se divise en trois parties : le **framework** qui représente le modèle (matérialisation du diagramme de classes du domaine), c'est-à-dire, la base commune sur laquelle s'appuient les autres éléments architecturaux, le **logiciel client** qui est en fait une interface graphique se basant sur le framework et permettant d'en exploiter les diverses fonctionnalités et enfin le **simulateur** qui permet de simuler des équipements. Cette simulation permet de tester le logiciel client en donnant l'illusion au client de se trouver dans une situation réelle avec de vrais équipements contrôlables à distance.

Concrètement, cela se matérialise de la manière suivante : lorsque le client ajoute un nouvel équipement dans le logiciel client, il devra lui associer un pilote existant (prévu pour communiquer avec un équipement simulé du simulateur), ce qui lui permettra une véritable interaction avec cet équipement simulé.

Illustration d'architecture générale de la solution



2. Rôle et constitution de chaque entité

Le **framework** représente le modèle de tout le système. On va globalement y retrouver toutes les classes décrites dans le diagramme de classes à savoir tout ce qui caractérise la gestion d'une maison (maisons, zones, pièces), la gestion des équipements (équipements, drivers) et la gestion des scénarios (scénarios, tâches, actions, déclencheurs). Cet acteur du projet fournit donc un ensemble de données et de services génériques relatifs à la domotique, pouvant être utilisés simplement par une application d'une couche plus haut niveau (par exemple un logiciel graphique).

De plus, le framework contient une couche d'abstraction permettant la gestion de l'accès aux données (sauvegarde/chargement des données, etc.).

Le **logiciel client** représente, lui, une implémentation graphique d'un logiciel de domotique de test. Celui-ci pourrait, s'il était plus abouti, être le véritable logiciel final de

domotique. Cependant, dans ce projet, il sert principalement à montrer comment on peut exploiter le framework et le simulateur pour répondre aux besoins du client.
Ce logiciel propose les diverses fonctionnalités exprimées dans le cahier des charges, que nous allons voir dans la partie suivante.

Le **simulateur** est la partie applicative permettant, comme son nom l'indique, de simuler des équipements afin de pouvoir donner un caractère cohérent au logiciel client. En effet, ne disposant pas d'équipements réels contrôlables à distance et de leurs pilotes associés, il s'est avéré nécessaire de simuler des équipements et de coder des pilotes "virtuels" leur étant associés (les pilotes étant conceptuellement en dehors du simulateur). Cet acteur du système permet donc de tester virtuellement l'interaction à distance avec les équipements proposés dans le logiciel client. Pour cela, il suffit, lors de l'ajout d'un équipement dans le logiciel client, d'y associer un pilote existant, prévu pour communiquer avec un équipement simulé. L'utilisateur, quand il interagira avec l'équipement qu'il aura créé, entraînera en fait une communication entre le pilote choisi et l'équipement simulé dans le simulateur.

3. Cinématique du logiciel

3.1 Fonctionnalités

Le logiciel répond globalement à toutes les spécifications énoncées dans le cahier des charges.
Le logiciel offre les fonctionnalités suivantes :

- Création/Modification d'une maison,
- Création/Modification d'une zone (dans une maison),
- Création/Modification d'une pièce (dans une zone),
- Création/Modification d'un équipement (dans une pièce).

On peut notamment renseigner le nom de l'équipement et y associer un pilote. Une fois le pilote associé, on peut voir dans un tableau les caractéristiques associées à un périphérique.

Une fois l'équipement ajouté, on peut opérer des actions dessus (définies par le pilote).

- Création/Modification d'un scénario (dans une maison).
- Création/Modification d'une tâche (dans un scénario).
- Création/Modification d'une action sur un équipement sélectionné (dans une tâche).
- Création/Modification d'un déclencheur (dans une tâche).

3.2 Interface Homme Machine

D'un point de vue IHM, le logiciel ne respecte pas rigoureusement les maquettes préalablement établies, mais s'en inspire cependant fortement. Il n'y a qu'une fenêtre principale divisée en deux parties (gauche-droite).

La partie gauche contient trois onglets :

- **Vue globale** : cet onglet affiche dans la partie gauche une vue en arbre de chaque maison renseignée dans le logiciel. À partir d'une maison, on peut donc descendre jusqu'à chacun de ses équipements, en passant par les zones et les pièces.
- **Équipements** : cet onglet affiche dans la partie gauche par le biais de listes déroulantes les listes d'équipements pour chaque maison renseignée.
- **Scénarios** : cet onglet affiche dans la partie gauche une vue en arbre de chaque scénario créé pour chaque maison. À partir d'une maison, on peut descendre jusqu'aux actions et déclencheurs de chaque tâche, en passant par les scénarios et les tâches.

La partie gauche permet donc de sélectionner les éléments avec lesquels on veut interagir.

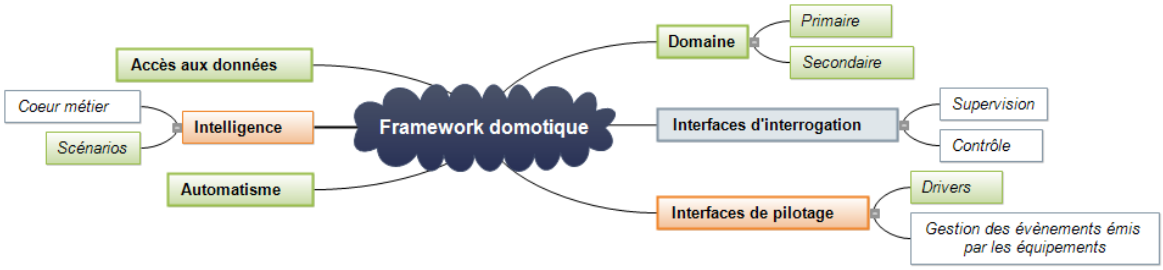
La partie droite, en fonction de ce qui a été sélectionné dans la partie gauche, va donc permettre de réaliser toutes les actions qu'on veut (afficher du contenu, ajouter du contenu, modifier du contenu, réaliser des opérations, etc...).

Par exemple, si on se trouve dans l'onglet **Scénarios** de la partie gauche et qu'on sélectionne un scénario, on va alors pouvoir, dans la partie droite, compléter le descriptif du scénario.

E. MODELISATION DU FRAMEWORK

1. Organisation du Framework

Le schéma suivant présente la composition générale du framework avec l'état d'avancement des tâches :



Légende

- En vert, les composants et sous-composants terminés.
- En orange, les composants incomplets ; les fonctionnalités principales ont été cependant développées.
- En bleu, les composants n'ayant pas été développés et qui restent en évolutions possibles du framework.

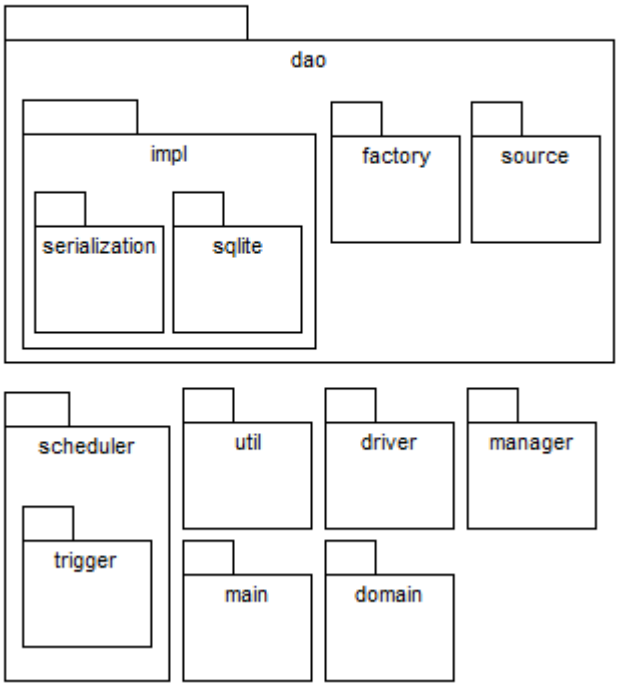
Détails des composants

- **Domaine :** cette partie correspond au regroupement des différents objets nécessaires au fonctionnement du framework comme support des données. On distingue les objets formant le domaine primaire (maison, zone, equipement, …) et le domaine secondaire qui est une réutilisation du domaine primaire pour formaliser des objets plus avancés (scénarios, tâches, ...).
- **Accès aux données :** ce composant correspond à l'abstraction de la couche de persistance. Notre framework repose sur différents objets pour son fonctionnement. Afin de sauvegarder et restaurer l'état de ces objets, nous avons opté dans un premier temps pour l'utilisation d'un moteur SQL léger et pratique (SQLite). Dans un

second temps, nous nous sommes repliés sur l'utilisation de la sérialisation Java. Le patron de conception DAO (Data Access Object) ayant été appliqué en amont, le passage à une nouvelle implémentation a été rapide ; l'efficacité de ce patron de conception a donc été observé et son utilisation justifiée. Si le framework est utilisé pour une utilisation à grande échelle (hotel, entreprise, …), le client pourra donc opter pour passer à un système de persistance plus performant (MySQL, Orale, …).

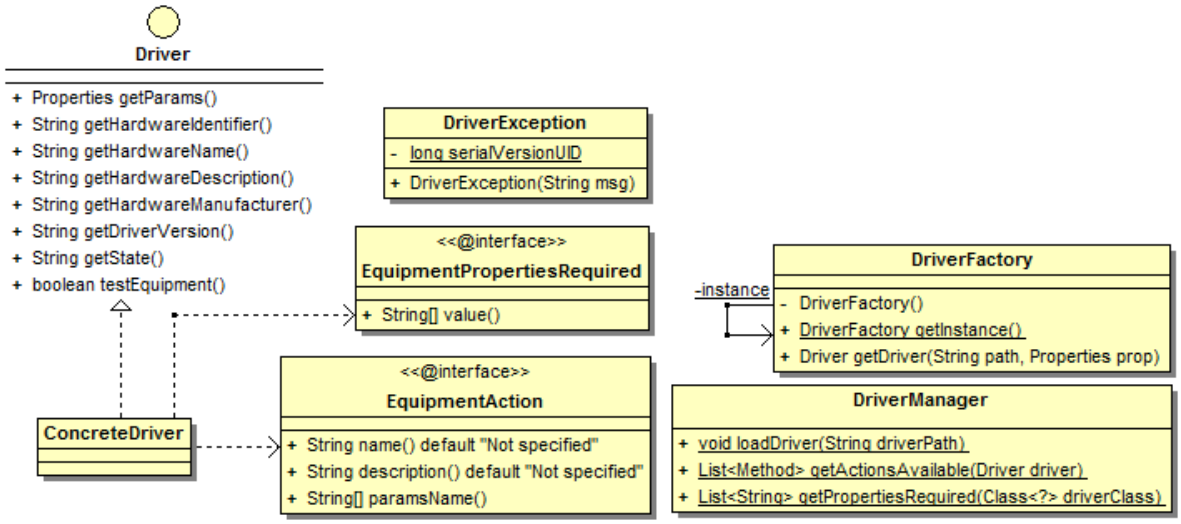
- **Intelligence :** cet aspect du framework correspond la couche métier. Elle se compose de la gestion des scénarios et de l'intelligence globale répartie dans le framework. Une gestion des erreurs centralisées aurait pu être envisagée pour avoir un diagnostic temps réel d'un système.
- **Automatisme :** cette composante est étroitement liée aux scénarios. Elle gère le déclenchement des différentes tâches planifiées.
- **Interfaces de pilotage :** ce module englobe toute l'abstraction de la communication avec les équipements distants. Le principal élément le composant est Driver (pilote), il définit le squelette à utiliser pour définir chaque implémentation correspondant à chaque modèle d'équipement (deux lampes Lapeyre n'auront pas forcément le même Driver). Une explication plus détaillée sur les pilotes est disponible dans la suite du rapport. L'aspect invocation provenant d'un équipement n'est actuellement pas géré (cas des alarmes).
- **Interfaces d'interrogation :** cette composante n'a pas été développée. Elle correspond aux différentes façades d'interrogation à distance dans le cas d'un client léger distant qui se connecterait pour interrogerer l'unité centrale. Dans le cadre du temps imparti au projet, nous laissons la charge au client utilisant le framework de réaliser cette partie avec la gestion des utilisateurs.

Le diagramme de paquetages ci-dessous présente l'organisation du framework en pratique :



2. Abstraction de la communication avec les équipements

Le diagramme ci-dessous présente les classes formant l'abstraction des actions avec l'implémentation du protocole de communication pouvant être réalisées sur un équipement.



Un driver (ou pilote) est spécifique à un modèle d'équipement. Il peut être utilisé par plusieurs équipements (par plusieurs instances du pilote en question). A l'aide des interfaces Driver, DriverException, EquipementPropertiesRequired et EquipmentAction, un fabricant peut émettre des drivers indépendamment de la société proposant le système de domotique.

Un pilote se développe en étendant l'interface Driver proposant des méthodes obligatoires pour se décrire pour permettre à une IHM d'avoir des informations formalisées sur ce dernier. L'interface propose de même une méthode de test de l'équipement permettant de tester une configuration de pilote ou d'exécuter un auto-diagnostique de l'équipement.

Notre abstraction se base sur l'API Réflexion de Java. Nous ne connaissons pas les différentes actions pouvant être exécutées sur un équipement par son pilote à l'avance mais nous pouvons les découvrir dynamiquement. L'instanciation d'un pilote nécessite un tableau associatif clé-valeur (Properties) permettant de configurer le pilote par rapport à l'équipement. Les clés nécessaires à un pilote sont décrites à travers une annotation.

Les annotations Java apporte à nos pilotes une capacité à se décrire par eux-mêmes :

- *EquipmentPropertiesRequired* permet de définir la liste des clés à passer obligatoirement à l'instanciation du driver. Pour exemple, l'adresse IP de l'équipement distant, son port de communication, son mode de fonctionnement, … nous sommes totalement abstraits des données passées.

- EquipmentAction permet de définir explicitement une action (méthode) sur un équipement afin de différencier des autres méthodes non concernées. Cette annotation contient en plus 3 attributs à compléter :
  - *name* : Nom de l'action
  - *description* : Description de l'action
  - *paramsName* : Tableau des noms correspondants aux paramètres de la méthode

  Ces paramètres d'annotation ont été ajoutés dans le cadre de l'IHM et de l'internationalisation. Ainsi, au lieu de récupérer le nom des arguments ou le nom brut de la méthode, nous utilisons ces informations. Couplé à un système d'internationalisation, un fabricant allemand pourra coder son pilote en russe, polonais,... tout en proposant un pilote multi-langage (en interface finale).

Les exceptions d'un driver doivent utiliser la classe DriverException. Pour exemple, une action avec un paramètre dont il y a une limite ; on pourra imaginer le pilote lever une exception pour avertir de l'erreur de l'utilisateur.

Pour terminer, la classe DriverFactory permet d'instancier un pilote et la classe DriverManager permet de charger une archive de type jar contenant des pilotes et en plus quelques outils de manipulation d'un pilote.

4. Patrons de conception utilisés

Dans le cadre de notre framework, un ensemble de patrons de conception de haut niveau ou bas niveau ont été utilisés. Ces patrons peuvent être des applications volontaires de l'intention ou bien simplement la reconnaissance de ces derniers après développement.

Les patrons de conception de haut niveau :

DAO

Ce pattern permet de faire le lien entre la couche d'accès aux données et la couche métier d'une application. Il permet de mieux maîtriser les changements susceptibles d'être opérés sur le système de stockage des données, donc, par extension, de préparer une migration d'un système à un autre (BDD vers fichiers XML par exemple...).
Dans notre cas, on l'utilise pour la persistance des données de type : maison, structure de la maison, scénarios, tâches...

Consommateur/Producteur

Ce pattern correspond au principe d'un PIPE.
Dans notre cas, on l'utilise pour gérer les actions à exécuter. En effet, les scénarios produisent des actions (producteurs) et une autre classe va se charger d'exécuter les actions (consommateur et gestion de la file d'attente).

Les patrons de conception de bas niveau :

Factory

Ce pattern permet que les instances se fassent à un seul endroit. Ainsi, si nous devons faire des changements, ils ne se feront qu'à un seul endroit.
Dans notre cas, nous l'avons couplé avec le pattern DAO. Ainsi, toutes les actions d'appels à notre DAO sont centralisées dans la factory.

Singleton

Ce pattern permet de s'assurer que l'objet en singleton ne soit instancier qu'une et une seule fois. On peut, via se pattern, gérer les accès concurrents de l'objet « singletoné » (pour cela on utilise un singleton "thread-safe").
Dans notre cas, on l'utilise pour plusieurs classes afin de s'assurer qu'on utilise toujours la même.

Facade

Utiliser de façon logique pour proposer des mécanismes un peu lourds à travers une seule méthode statique pour simplifier et permettre une réutilisation. Ce patron est utilisé à plusieurs endroits dans le framework avec des degrés de complexité ou de longueur de code variables.

Callback

A travers l'utilisation de l'API Reflexion de Java pour l'exécution de nos actions, nous avons la même idée que le patron de conception Callback. Pour rappel, une action correspond à une méthode du pilote, cette méthode est stockée par arguments voir sur support dans le cas des scénarios par exemple avant d'être exécutée.

F. CONTROLE DE QUALITE

Gestion des risques

Un audit de code a été réalisé avec le logiciel Sonar. Ainsi, les risques liés au codage du programme qui est susceptible d'être à l'origine d'incidents, ont été identifiées.

Tests unitaires

JUnit a été utilisé pour réaliser des fragments de code. Notamment, la DAO (avec la serialisation) pour l'objet 'Maison' et pour l'objet 'Equipement' a été testé. Ainsi, la cohérence et l'intégralité des données ont été vérifiées dans des cas passants et critiques.

Tests d'intégration

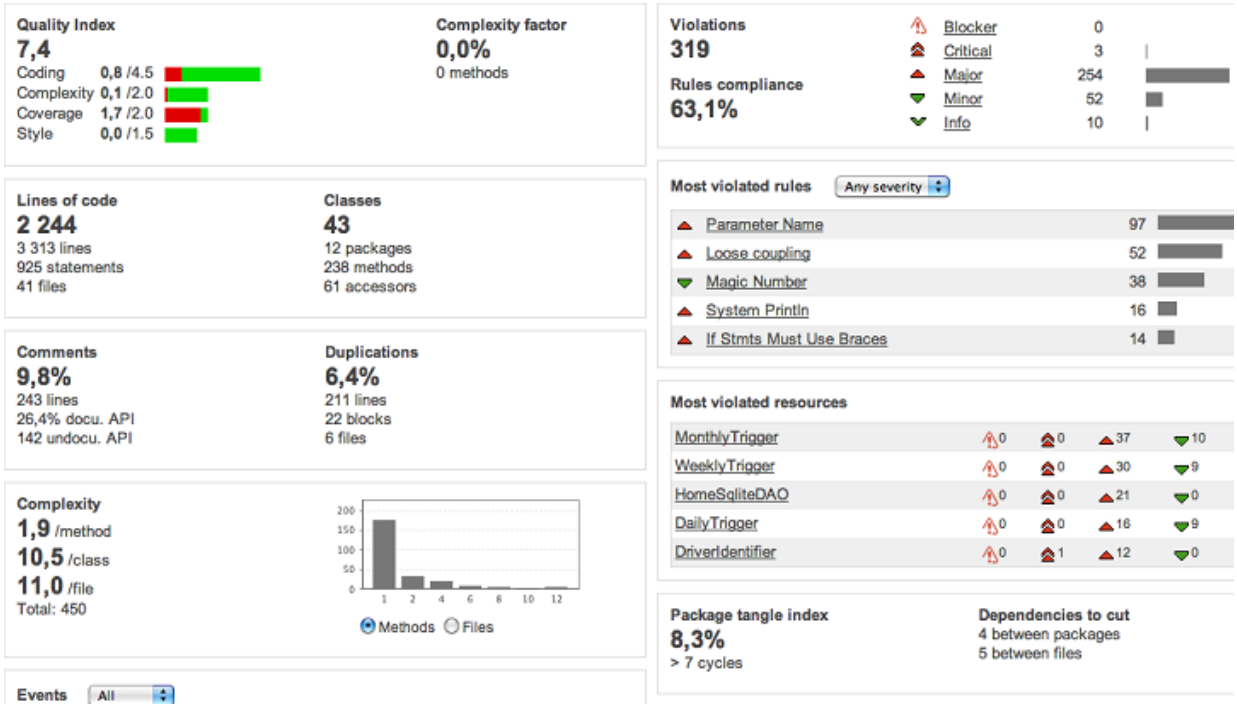
Plusieurs scénarios de tests ont été envisagés :

- la configuration d'une maison particulière en ajoutant/supprimant/modifiant tous les éléments possibles : maisons, zones, pièces, équipements, scénarios, tâches (actions sur les équipements et déclencheurs).
- la visualisation de la configuration de la maison à travers l'arbre (partie gauche de l'écran).
- le pilotage de chaque équipement après les avoir ajouter, lié à leur driver.
- la vérification des états de chaque équipement en temps réel.
- L'exécution de scénarios définis par l'utilisateur mettant en action plusieurs équipements et plusieurs déclencheurs simultanément.

Annexes

A. ANALYSE DE QUALITE DU CODE PAR SONAR

L'analyse effectuée par l'outil Sonar a donné les métriques suivantes :



Name	Rules compliance	Coverage	Build time	Links
SmartHouse	63,1%	16,3%	12:35	

Name	Rules compliance	Coverage	Build time	Links
dao	86,4%	100,0%	12:35	
dao.factory	92,5%	50,0%	12:35	
dao.impl			12:35	
dao.impl.serialization	94,4%	18,9%	12:35	
dao.impl.sqlite	63,3%	0,0%	12:35	
dao.source	72,7%	46,4%	12:35	
domain	62,2%	18,8%	12:35	
driver	65,8%	15,3%	12:35	
main	51,4%	48,0%	12:35	
manager	74,4%	0,0%	12:35	
scheduler	51,7%	0,0%	12:35	
scheduler.trigger	7,2%	0,0%	12:35	
util	62,1%	59,1%	12:35	

Nous avons constaté que les règles et normes de codage Java non respectées ne compromettent pas l'intégrité du projet. Les métriques fournies concernent uniquement le framework. Nous avons analysé une version du code tardive ce qui ne nous a pas permis d'éradiquer les erreurs facilement corrigibles.