Regis.Clouard@ensicaen.fr
Julien.Mahier@ensicaen.fr
Sylvain.Vernois@ensicaen.fr

Logiciel de domotique

Ce projet vise à la mise en œuvre des techniques de génie logiciel pour la réalisation d'un projet concret. Une attention particulière sera donc accordée à la qualité de la conception de votre logiciel. Il est difficile dans le cadre de séances de travaux pratiques de proposer un projet d'une taille suffisante pour se rendre compte qu'une conception mal structurée conduit inévitablement à un logiciel rigide, fragile et immobile qui vire à l'usine à gaz. Compte-tenu des contraintes de temps, nous ne pouvons que proposer un projet modeste et qui pourrait être conçu sans compétences particulières en génie logiciel. Néanmoins, dans un but pédagogique, il vous est demandé de bien vouloir jouer le jeu de la gestion de projet et de livrer un logiciel démontrant des qualités de robustesse, extensibilité et réutilisabilité, accompagné de ses tests unitaires. Un projet ne présentant pas toutes les fonctionnalités demandées mais conçu selon les règles de l'art sera beaucoup mieux évalué qu'un projet complètement fonctionnel mais bricolé.

1 Contexte

Vous venez de créer une entreprise dans le domaine du développement de logiciels de domotique. Un client (rôle joué par votre encadrant de TP) qui représente un célèbre fabricant de matériel domotique vous demande de concevoir un logiciel permettant d'interfacer leur matériel et qui sera distribué gracieusement sur leur site à titre promotionnel.

1.1 Architecture fonctionnelle d'un système domotique

Un système domotique est basé sur une architecture fonctionnelle organisée autour d'une unité centrale qui contrôle les équipements connectés, d'une ou de plusieurs interfaces homme-machine qui permettent la gestion du système par un utilisateur et de pilotes pour les équipements connectés à l'unité centrale. La Figure 1 résume l'architecture globale d'un tel système.

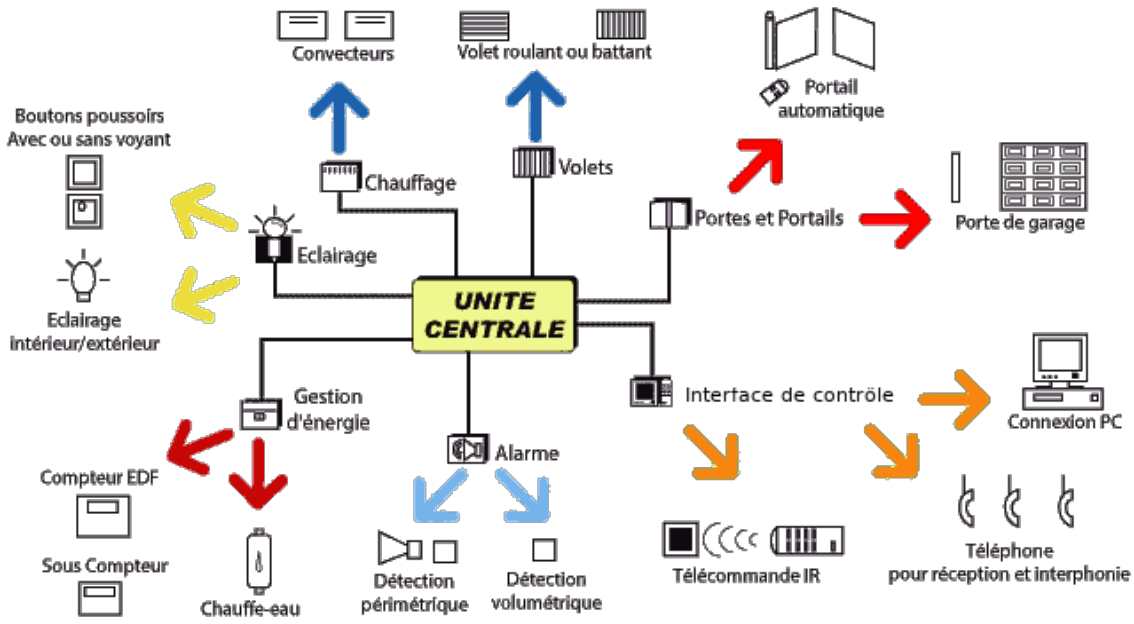


Figure 1. Architecture fonctionnelle globale du système.

L'unité centrale

L'unité centrale est évidemment le cœur du système. On supposera ici qu'elle est résidente sur un serveur bénéficiant d'une connexion à internet. C'est sur ce serveur que seront branchés les équipements choisis. L'unité centrale regroupe l'ensemble des programmes qui permettent la gestion et le contrôle du parc d'équipement. Elle doit permettre au minimum de :

- Piloter chaque équipement individuellement ; par exemple faire couler un bain, moduler l'éclairage ou baisser les volets roulants.

- Exécuter des scénarios particuliers définis par l'utilisateur mettant en action simultanément plusieurs équipements, par exemple :
    - Le scénario « départ au travail » enchaîne les actions : l'éclairage s'éteint, le garage s'ouvre, le chauffage se met en veille, les volets et le garage se ferment.
    - Le scénario « rentrée du travail » effectue les actions inverses : les volets s'ouvrent et le chauffage passe en mode confort.
    - Le scénario « vacance » ou « nuit ».

Interface de contrôle

L'interface permet à un utilisateur de contrôler graphiquement le système. L'interface est un client du serveur et peut être portée par :

- un PC équipé d'une connexion Internet ;
- une tablette tactile ;
- un téléphone mobile.

Les différents utilisateurs du système seront identifiés et pourquoi pas authentifiés par le système.

Équipements

Les équipements intégrables au système doivent disposer d'un pilote logiciel pour les connecter au serveur. Quelques exemples d'équipements possibles (non exhaustifs) :

- Chauffage, volets, portail et porte du garage, éclairage, alarme, lave-linge, lecteur de DVD, cinéma à domicile…

Définition d'une habitation

Une *habitation* est faite de plusieurs *zones*. Chaque zone regroupe des *pièces*. Une zone peut être un étage ou une sphère d'activités telles qu'une sphère de jour avec la cuisine et le salon ; une sphère de nuit avec l'ensemble des chambres ; la sphère jardin ou la sphère garage. Une même pièce peut être incluse dans plusieurs zones.

2 Travail à réaliser

Dans le cadre de ce projet, il s'agit avant tout de développer un **framework** qui constitue la base de la réalisation d'un système domotique réel. Ce que l'on va chercher à maximiser sont les capacités d'**extensibilité**, de **réutilisation** et de **maintenance** de l'architecture, plus que son ergonomie et la liste des fonctionnalités. Elle doit pour cela, être simple (KISS[1]) et respecter les principes SOLID. La qualité du framework sera notamment jugée sur sa capacité à traiter les questions suivantes :

- comment est représentée une habitation ?
- comment est ajouté un nouvel équipement ?
- comment est piloté un équipement ?
- Comment est construit un scénario ?

Pour aider au développement de ce framework, il vous est aussi demandé de développer un système de simulation de système domotique construit sur la base du framework. Dans ce système, tous les équipements seront simulés et l'interface de contrôle sera limitée à un PC sur le réseau.

2.1 Exigences fonctionnelles minimales du client

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

- ☑ Configurer une habitation particulière.
- ☑ Contrôler les équipements individuellement, p. ex. fermer la porte d'entrée, allumer un lecteur de 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.
- ☑ Contrôler le serveur via une interface client sur un PC relié au serveur central par le réseau.

1 Keep It Simple and Stupid

☑ Afficher la météo courante du lieu de résidence sur l'interface client.
Le site https://www.prevision-meteo.ch/uploads/pdf/recuperation-donnees-meteo.pdf fournit les informations météorologiques sous la forme d'un fichier Json. Si l'on suppose que l'habitation se trouve à Caen : https://www.prevision-meteo.ch/services/json/caen.

1.1 Exigences non fonctionnelles

☑ Réseau : La communication entre le serveur et les clients utilise une socket.

☑ Base de données : La persistance des données est réalisée dans une base de donnés SQLite.

1.2 Contraintes pédagogiques

☑ **JAVA** : Le langage d'implémentation sera le Java.

☑ **JAVAFX** : L'interface graphique se basera intégralement sur la bibliothèque JavaFX.

☑ **Test** : Le code source du projet doit s'accompagner de **tests unitaires** et de **tests d'intégration** développés avec JUnit et potentiellement Mockito.

☑ **Gradle** : le projet sera géré par le moteur de production Gradle.

☑ **GIT** : La gestion des versions sera réalisée avec Git. Le dépôt central sera localisé sur gitlab.ecole.ensicaen.fr. La récupération des livrables finaux sera faite par votre encadrant directement à partir du dépôt Gitlab dont vous lui communiquerez l'adresse[2].

☑ **UML :** Les modélisations seront réalisées avec le langage UML. La rédaction des diagrammes UML devra se faire avec un atelier de génie logiciel installé sur les machines ou un autre atelier de votre choix[3].

☑ **Patrons de conception** : L'architecture devra, bien entendu, faire appel aux patrons de conception, dont la pertinence devra être justifiée dans le rapport et la soutenance.

2 Vous n'oublierez pas d'ajouter le client à votre dépôt avec le rôle développeur afin qu'il puisse suivre votre progression.
3 Voire les suggestions en fin de document.

3 Organisation du projet

1.3 Méthode de gestion de projet

Le projet sera réalisé par équipe de 8 personnes et se déroulera sur **8 séances de 2 heures**.

Il vous est demandé d'utiliser une méthode de gestion de projet à base de **cycles de développement itératifs**.

- Dans notre cadre, une itération correspond à deux séances.
- Le projet commence par une première séance de définition des fonctionnalités du futur logiciel et une répartition globale sur les itérations prévues. Cette phase associe normalement pleinement le client ; ici, votre client viendra simplement valider ce que vous aurez prévu.
- Chaque itération commence par la définition d'un sous-ensemble de tâches à réaliser (de granularité assez fine) qui correspondent aux fonctionnalités identifiées pour cette itération.
- Au cours de l'itération, les tâches sont réalisées par les développeurs de l'équipe avec une étanchéité la plus élevée possible entre les développeurs.
- Chaque itération se termine par une **démonstration** du prototype opérationnel montrant à votre client les fonctionnalités développées jusque-là. Cette démonstration est un moment d'évaluation par votre encadrant de votre capacité à concevoir le logiciel demandé. Elle doit donc être préparée et réalisée de manière formelle. Le trop fréquent « effet démo » est ici une erreur qui sera sanctionnée.
- La démonstration s'accompagne ensuite d'une rétrospective qui vise à réviser le plan de développement, en supprimant, ajoutant ou changeant les fonctionnalités de la liste prévue.

1.4 Rôles et fonctions dans chaque équipe

Chaque équipe doit répartir les responsabilités en nommant des personnes aux postes suivants :

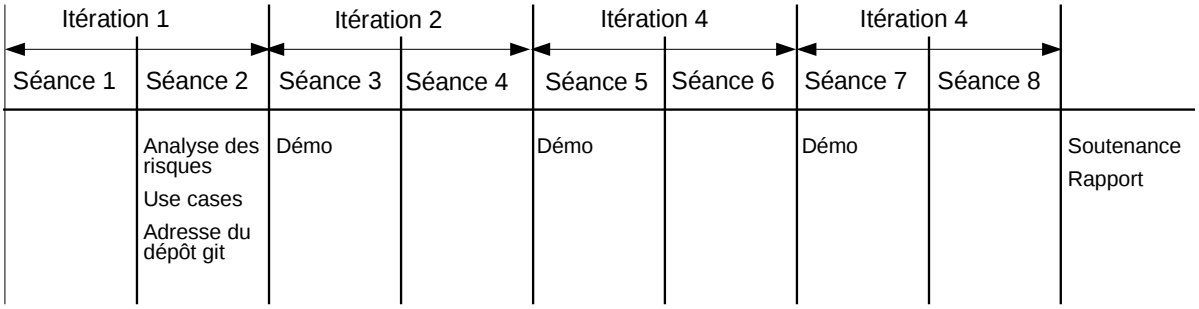
☑ **Chef de projet** : son rôle est de coordonner les activités de

l'équipe, de superviser les travaux et d'interagir avec le client.

- ☑ **Architecte** : son rôle est de concevoir l'architecture du logiciel, l'organisation de la conception en paquets et de prendre les décisions tactiques relatives à cette réalisation. Il a aussi en charge les tests d'intégration.
- ☑ **Développeur** : son rôle est de concevoir les classes et leur organisation pour réaliser les fonctionnalités qui lui sont attribuées. Le développeur doit aussi écrire les tests unitaires correspondants (*pourquoi pas en TDD*). Seules les classes réellement graphiques ne sont pas accompagnées de tests unitaires.
- ☑ **Responsable de version** : cet ingénieur est le responsable du dépôt sur Gitlab. C'est lui qui fait l'intégration continue du travail des développeurs et réalise le prototype de démonstration qui sera présenté à l'issue de chaque itération. Il est aussi le garant de la propreté du code et de la présence des tests dans les contributions des développeurs.

Il est à noter qu'une même personne peut endosser plusieurs rôles et qu'un rôle peut être assumé par plusieurs personnes.

3.1 Livrables



Premier rendu au début de la séance 2

- ☑ **Analyse des risques** : la liste des risques majeurs de ne pas parvenir à développer le logiciel et les moyens de les prévenir ou y remédier.
- ☑ **Cas d'utilisation** : l'analyse des besoins sur la forme d'un diagramme des cas d'utilisation.

☑ **Dépôt git** : l'adresse gitlab du projet.

Rendu final

Différents documents, regroupés dans un rapport, devront être produits. Le rapport devra au moins contenir les documents suivants :

☑ **Analyse des risques** : reprise de l'analyse faite en première séance.

☑ **Cas d'utilisation** : le diagramme des cas d'utilisation du logiciel.

☑ **Conception UML** : La modélisation du logiciel en UML où les patrons de conception seront mis en exergue.

☑ **Diagramme de paquet** : uniquement les paquets et leurs dépendances.

☑ **Code source** : Le projet Gradle avec le code Java du logiciel et le code des tests unitaires et des tests d'intégration.

Cette liste n'est en aucun cas exhaustive et pourra être complétée avec d'autres documents qui vous paraîtront pertinents dans la limite du raisonnable (au sens Agile du terme).

1.5 Soutenance

À l'issue des huit séances de travail, une soutenance sera organisée pour la validation finale du logiciel organisée en 10 minutes de présentation, 5 minutes de démonstration et 10 minutes de question. La soutenance devra au moins faire apparaître clairement :

☑ L'organisation de l'équipe et la gestion de projet conduite.

☑ L'architecture de votre système.

☑ La réponse de votre conception à la robustesse, la réutilisabilité et la maintenance.

Si la planification de votre projet s'est avérée irréaliste et que l'avancement réel en est très éloigné, il vous est demandé de prendre du recul et d'analyser les causes du dysfonctionnement. Le travail d'une équipe qui saurait analyser des dysfonctionnements et proposer des solutions sera bien mieux apprécié

que celui d’une équipe qui chercherait à dissimuler d’éventuelles déconvenues derrière un produit fini, même spectaculaire.

Lors de la présentation orale, tous les membres de l’équipe doivent présenter l’aspect du projet dont ils sont responsables. Il ne sera pas accepté qu’une seule personne présente l’ensemble du projet.

1.6 Évaluation

La note de TP de chaque membre d'une équipe sera la moyenne des notes obtenues pour chacune des quatre évaluations suivantes :

- ☑ La perception du client sur votre capacité à produire le logiciel demandé.
- ☑ La pertinence de la conception.
- ☑ La qualité du rapport.
- ☑ La propreté du code et la présence des tests automatiques.
- ☑ La qualité de la soutenance.

4 Ressources

Afin de vous aider dans la réalisation de votre projet vous trouverez sur la plateforme pédagogique plusieurs ressources documentaires et matérielles.

L'ensemble des ressources proposées sont conformes à l'IDE IntelliJ Idea, que nous vous encourageons à utiliser.

1.7 Exemples de code Java

- ☑ Un projet Gradle avec un exemple d'interface graphique en JavaFX basée sur le patron d'architecture MVP[4].
- ☑ Un exemple en Java de communication client-serveur utilisant les sockets.

1.8 Documentation

- ☑ Le travail de développement de projet avec Git (workflow github pour Gitlab).
- ☑ Les tests unitaires avec JUnit.
- ☑ Les doublures avec Mockito.

4 Model View Presenter