Université Ibn Zohr Faculté des Sciences à Agadir

Centre d'Excellence IT

Master d'Excellence

Ingénierie Informatique et Systèmes Embarqués - IISE

Module : Systèmes répartis et distribués

Cahier de charges

Système distribué pour la gestion de la consommation énergétique d'une maison

Présenté par :

EL GOUMIRI Imane

Ben-kari Fatima Ezzahra

Mariam kazaz

Maryam lamahad

Encadré par :

Pr. IDRAIS

Année universitaire : 2024/2025

Sommaire

Liste de figures

I. Introduction

1. Contexte du projet

Notre projet vise à implémenter un système distribué pour la gestion de la consommation énergétique d'une maison. Ce système distribuera les différentes parties de l'application sur plusieurs serveurs et permettra la communication entre eux pour accomplir des tâches spécifiques.

2. Objectifs

- Suivre en temps réel la consommation énergétique de chaque appareil.
- Analyse historique des données.
- Gérer et optimiser l'utilisation des ressources via des appels distans.
- Envoyer des alertes automatiques si un appareil dépasse un seuil critique.

II. Architecture logicielle et physique

1. Description générale de l'architecture du système

Le système est conçu selon une architecture distribuée orientée services, où chaque composant assure une fonction précise, répartie sur des serveurs distincts. L'interconnexion entre les serveurs se fait via des appels de méthodes distantes (Java RMI), assurant une communication efficace. Chaque serveur joue un rôle spécifique dans le traitement, l'analyse ou la gestion des données énergétiques collectées en temps réel. Le client (utilisateur) interagit avec le système via une interface Web qui affiche les données, génère des graphiques dynamiques et permet de contrôler les appareils connectés.

1.1 Architecture en 3 niveaux

L'architecture sera basée sur une approche en trois niveaux :

- ***Niveau 1 : Serveur de Surveillance***

- Fonctionnalité : Collecte en temps réel les données de consommation des appareils de la maison.

- Technologie : Java RMI pour la communication distante entre le serveur et les appareils connectés.

- Responsabilité : Ce serveur recueille les informations concernant la consommation énergétique de chaque appareil. Il sera en contact direct avec les dispositifs matériels (via des capteurs).

- ***Niveau 2 : Serveur d'Analyse***

- Fonctionnalité : Stocke les données historiques et effectue des analyses sur la consommation.

- Technologie : Java RMI pour recevoir les données du serveur de surveillance et MySQL pour le stockage.

- Responsabilité : Ce serveur gère la base de données MySQL où toutes les données énergétiques sont stockées. Il analyse les tendances de consommation sur des périodes données.

- ***Niveau 3 : Serveur de Recommandation et Gestion***

- Fonctionnalité : Envoie des alertes, recommandations et permet de contrôler à distance les appareils de la maison.

- Technologie : Java RMI pour la communication entre les serveurs.

- Responsabilité : Ce serveur analyse les données reçues et génère des alertes si la consommation dépasse un seuil critique. Il propose des recommandations pour optimiser la consommation et permet de gérer les appareils à distance.

1.2 Serveur RMI

Chaque serveur (surveillance, analyse, gestion, recommandations) sera développé en Java, et la communication entre ces serveurs se fera via des appels distants RMI. RMI permet aux objets situés sur différents serveurs de communiquer comme s'ils étaient locaux.

1.3 Client web

- Fonctionnalité : Interface graphique pour l'utilisateur avec un tableau de bord interactif.

- Technologie : HTML, CSS, JavaScript pour l'interface, avec Java EE pour le backend. Les utilisateurs pourront voir des graphiques de consommation, recevoir des alertes et envoyer des commandes à distance.

- Responsabilité : Cette interface affiche les données des serveurs (par exemple, consommation actuelle, historique, alertes) via une connexion sécurisée avec les serveurs d'analyse et de gestion.

1.4 Communication entre Serveurs

- Les serveurs communiquent entre eux via des appels RMI, garantissant une séparation claire des responsabilités. Chaque serveur est indépendant et gère ses tâches spécifiques.

- Le serveur de surveillance envoie les données au serveur d'analyse pour stockage. Le serveur de gestion peut interagir avec le serveur de recommandations pour contrôler les appareils.

1.5 Base de données (MySQL)

- La base de données stocke toutes les données historiques de consommation énergétique, les seuils définis pour chaque appareil, et les alertes envoyées aux utilisateurs.

- Une architecture de base de données relationnelle permet une gestion efficace des données et leur récupération pour analyse.

2. Déploiement distribué

2.1 Environnement de Déploiement

Le système sera déployé dans un environnement distribué hétérogène combinant :

- Machines virtuelles (VMs) pour les serveurs RMI (1 VM par serveur spécialisé).
- Cloud privé (ou local) pour la base de données MySQL.
- Clients légers (navigateurs web) pour l'interface utilisateur.

2.2 Topologie du réseau

Le système repose sur une topologie en étoile dans laquelle :

- le client (interface web) communique avec le serveur web via le protocole HTTP.
- le serveur web communique avec les serveurs métiers (surveillance, analyse, recommandation, gestion) via RMI (Remote Method Invocation).
- les serveurs accèdent à une base de données MySQL partagée via JDBC.

2.3 Communication

La communication entre composants s'effectue de manière asynchrone ou synchrone, selon le besoin :

- RMI (Java) est utilisé pour les appels distants entre serveurs (analyse, gestion, alerte…).
- Les sessions utilisateurs sont gérées côté serveur web pour maintenir les connexions entre l'interface web et les modules métiers.
- Les serveurs échangent des messages via un réseau local sécurisé ou via VPN dans le cloud.

III. Diagrammes UML

Pour faciliter la compréhension du fonctionnement de l'application, nous avons utilisé l'**UML** (Unified Modeling Language). C'est un langage de modélisation, constitué de diagrammes qui servent à visualiser et décrire la structure et le comportement des objets qui se trouvent dans un système. Il permet de présenter des systèmes logiciels complexes de manière plus simple et compréhensible qu'avec du code informatique.

1. Diagramme de cas d'utilisation

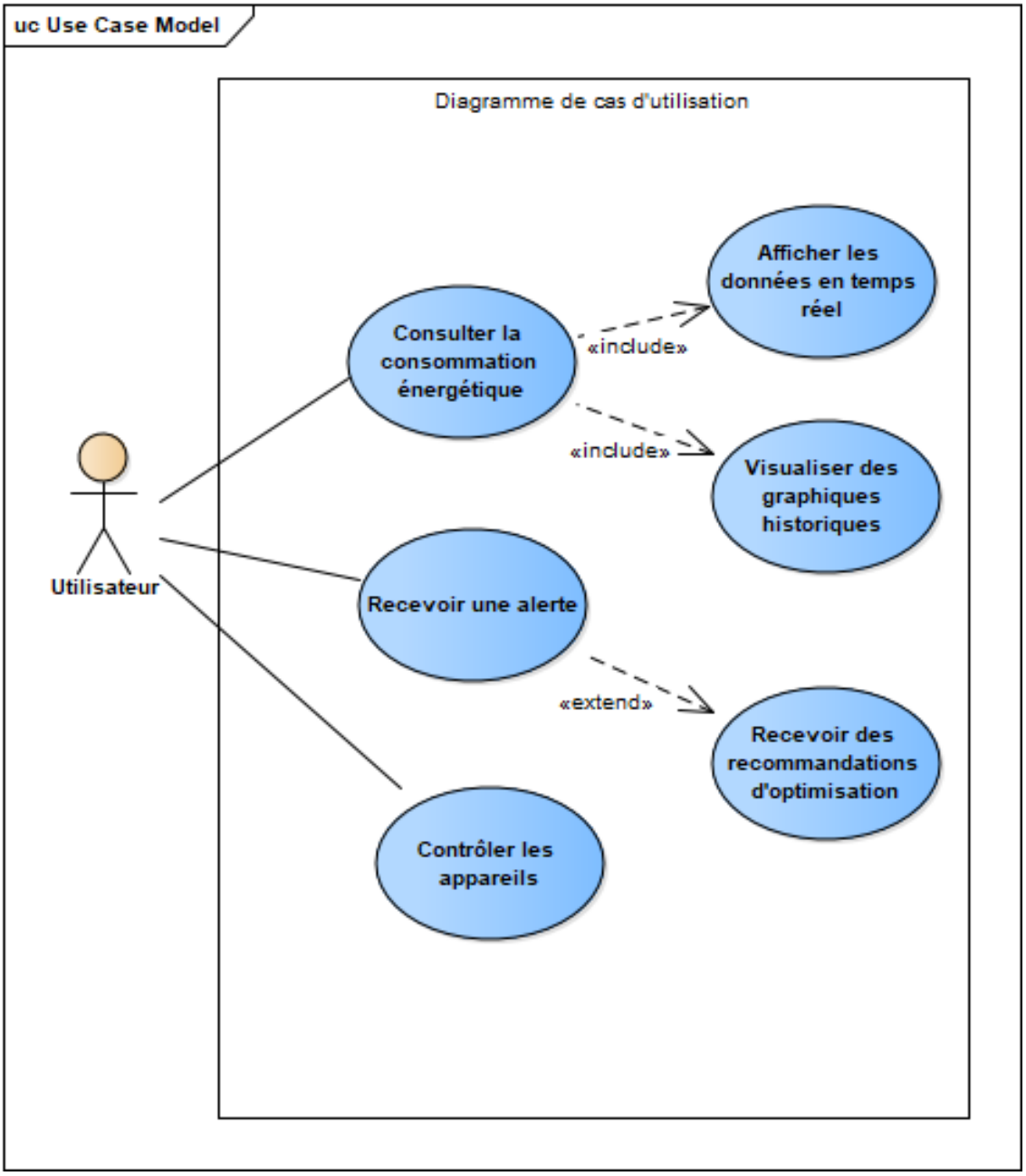

Figure 1 : Diagramme de cas d'utilisation

Ce diagramme montre l'utilisateur interagissant avec l'interface Web, qui agit comme passerelle vers les serveurs. Chaque cas d'utilisation représente une fonctionalité métier. Les serveurs exécutent les traitements associés en arrière-plan via RMI.

2. Diagramme de séquence

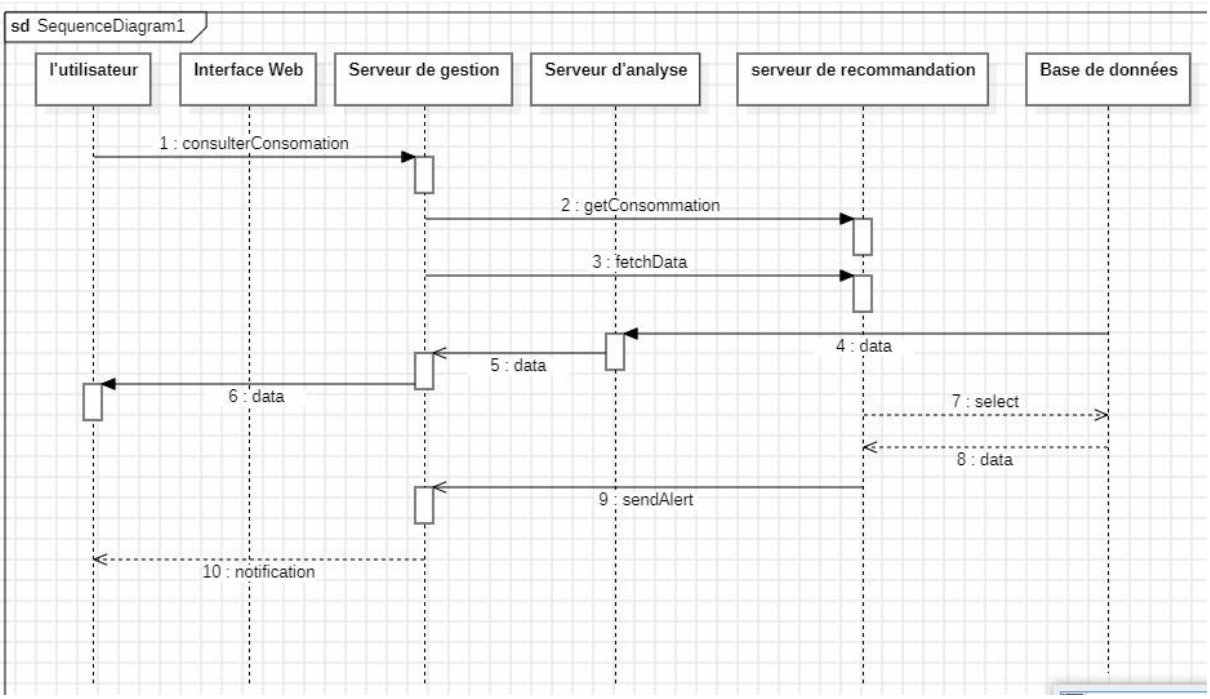

Figure 2 : Diagramme de séquence

Ce diagramme permet de visualiser le fonctionnement d'un service distribué, avec appels RMI entre les serveurs et interaction avec l'interface web.

3. Diagramme de classe

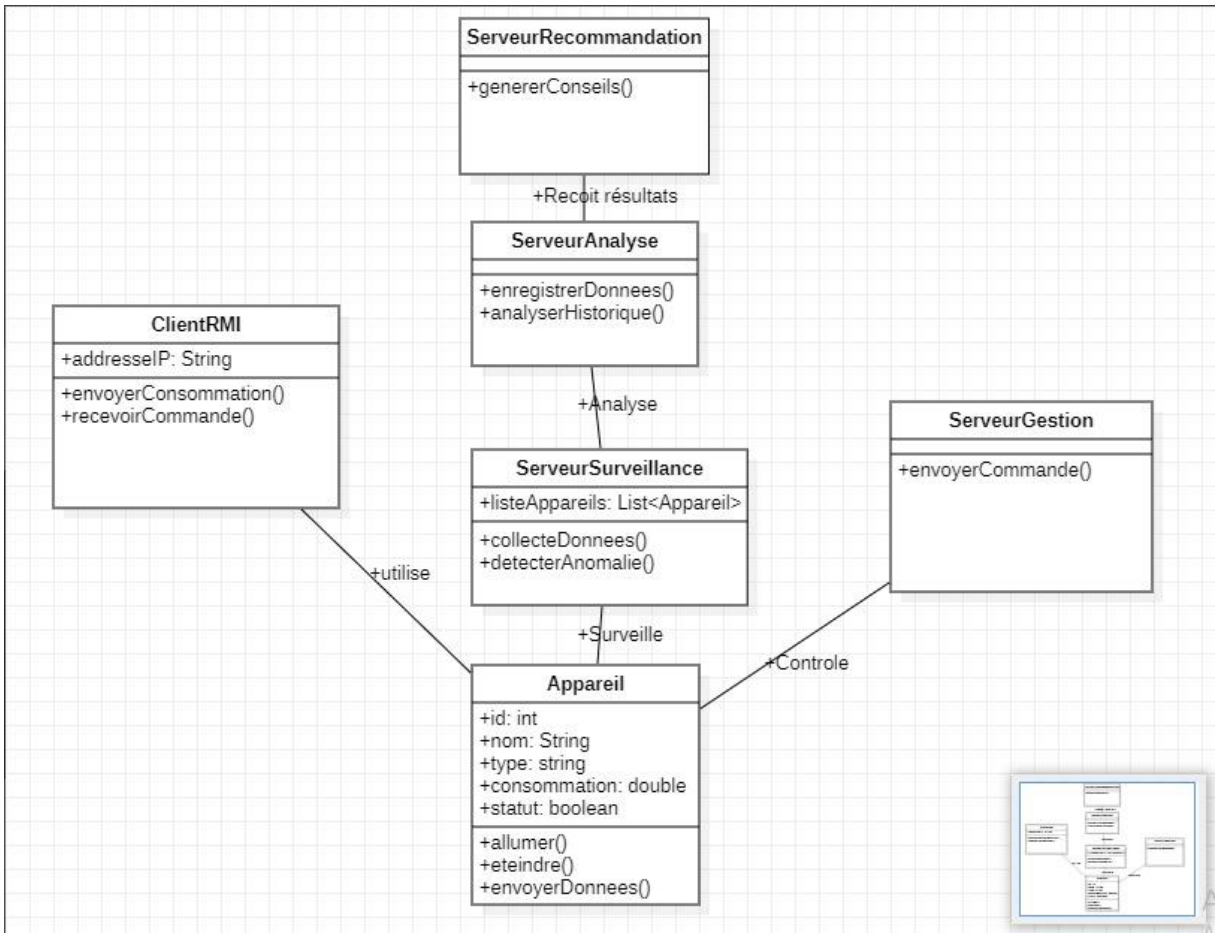

Figure 3 : Diagramme de classe

Chaque serveur RMI est représenté par une classe avec ses méthodes exposées à distance. Les entités comme Appareil et Utilisateur sont reliées aux serveurs via des associations.

4. Diagramme de déploiement

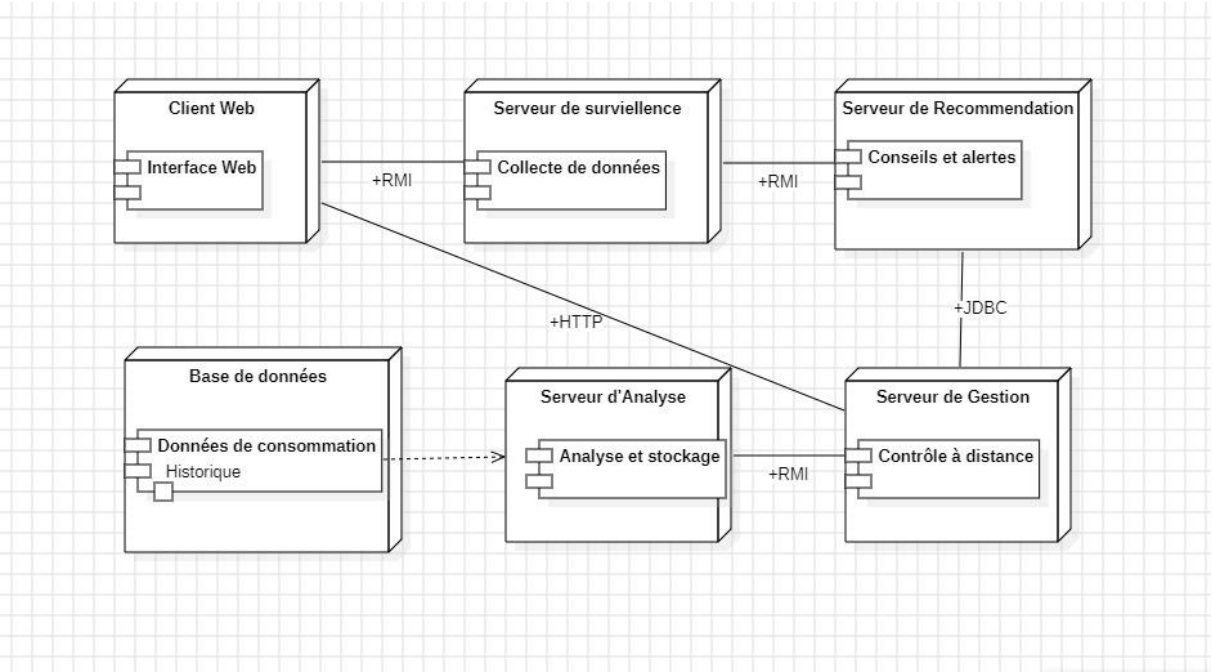

Figure 4 : Diagramme de déploiement

Ce diagramme montre la distribution physique du système, les connexions réseau (RMI/HTTP), et comment les composants sont déployés sur différentes machines, locales ou cloud.

IV. Techniques utilisées

- **Backend (Serveur RMI) :**

Java & C++ : Le système distribuera les différentes parties de l'application sur des serveurs utilisant Java pour les services RMI et C++ pour certaines parties de traitement des données, assurant une communication distante efficace.

- **Frontend (Interface Web) :**

Java EE : Pour la gestion des processus côté serveur, avec JSP/Servlet pour générer des pages dynamiques côté client.

HTML/CSS/JavaScript : Pour la structure de l'interface utilisateur et la gestion des interactions côté client.

- **Base de données :**

MySQL : Pour le stockage des données de consommation des appareils, l'historique des alertes et des rapports d'analyse.

- **Gestion de version :**

GitHub : Pour le contrôle de version du code source, la gestion de la collaboration et la mise à jour du code sur un repository central.