

République Tunisienne
Ministère de l'Enseignement Supérieur et
de la Recherche Scientifique

Université de Sfax
Ecole Nationale d'Ingénieurs de Sfax
Département de Génie Informatique et de
Mathématiques Appliquées



Cycle de Formation d'Ingénieurs
dans la discipline
Génie Informatique

Projet de fin d'études

N° d'ordre: 2013-DGIMA-037

MEMOIRE

présenté à

L'Ecole Nationale d'Ingénieurs de Sfax

(Département de Génie Informatique et de Mathématiques Appliquées)

en vue de l'obtention

du Diplôme National d'Ingénieur en Génie Informatique

par

Khalil HAMDI

**Etude et développement d'une solution pour le
contrôle domotique à distance**

Soutenu le 21 Juin 2013, devant la commission d'examen :

Mr. Adel M. ALIMI

Président

Mme. Nouha BACCOUR

Rapporteur

Mme. Boudour AMMAR

Encadreur académique

Mlle Sonda BOUSNINA

Encadreur industriel

Dédicaces

C'est grâce Dieu que tout a commencé,

Et c'est à lui que je rends grâce.

Le reste n'est que dédicaces.

*A mon cher père Hassen qui a sacrifié ses plus belles années pour
embellir les miennes,*

*Aucun mot ne serait assez pour témoigner de l'étendue des sentiments
que j'éprouve à son égard,*

*A ma chère mère khadija, que nulle dédicace ne puisse exprimer ce
que je lui dois, pour sa bienveillance, son affection et pour les
sacrifices qu'elle avait consentis,*

A mon cher frère Mohamed,

A ma chère sœur Fatma,

A toute ma famille,

Pour leurs encouragements et leurs amours.

A mes professeurs et encadrants

*A tous mes amis et tous ceux que j'aime et qui me sont chers
Je dédie ce travail en témoignage de mon grand amour, ma profonde
reconnaissance et ma gratitude infinie.*

Remerciements

C'est avec un grand plaisir que nous réservons cette page en signe de gratitude et de reconnaissance à tous ceux qui ont assisté ce travail.

Nous exprimons nos vifs remerciements à M. Mohammed Adel ALIMI, et à Madame Nouha BACCOUR, pour l'honneur qu'ils nous ont donné en acceptant de juger notre mémoire de fin d'études.

Nous tenons à exprimer notre gratitude à notre encadreur Madame Boudour AMMAR pour sa gentillesse, ses encouragements et ses conseils Judicieux,

Nous remercions aussi Mademoiselle Sonda BOUSNINA, notre encadreur industriel pour son aide précieuse dans la réalisation de ce projet.

Enfin, nous adressons nos remerciements à tout le personnel de la société Cynapsys pour leur accueil chaleureux et leur esprit de collaboration.

khalil

Sommaire

Introduction Générale.....	1
Chapitre 1 : Etat de l'art	3
Introduction	4
1 Présentation du projet.....	4
1.1 Présentation de l'entreprise.....	4
1.2 Travail demandé.....	5
2 Etude préliminaire	5
2.1 Plan du travail	5
2.2 Diagramme de GANTT	7
3 Concepts de base	8
3.1 Domotique.....	8
3.1.1 Définition	8
3.1.2 Avantages de la domotique	8
3.2 La plateforme Androïde	9
4 Etude l'existant.....	10
4.1 Les moyens de commande domotique existants	10
4.1.1 Moyens de Commande en local	10
4.1.2 Moyens de commande à distance :.....	11
4.2 Applications domotiques existantes.....	13
Conclusion	14
Chapitre 2 : Spécifications des besoins	15
Introduction	16
1 Problématique.....	16
2 Spécification des besoins	16
2.1 Besoins fonctionnels	16
2.2 Besoins non fonctionnels	17

3	Choix technologiques :.....	17
3.1	OSGI: Open Service Gateway Initiative.....	17
3.1.1	Présentation de la technologie OSGI	17
3.1.2	Domaines d'application.....	18
3.1.3	Architecture du Framework.....	18
3.1.4	Services	19
3.1.5	Description d'un composant OSGI	20
3.1.6	Cycle de vie d'un composant OSGI.....	21
3.2	Utilisation d'OSGI dans Androïde.....	22
3.3	Avantages d'un serveur OSGI	23
4	Architecture de l'application.....	24
4.1	Architecture physique	24
4.2	Architecture logicielle.....	25
	Conclusion	28
	Chapitre 3 : Analyse et conception.....	29
	Introduction	30
1	Choix du langage de modélisation	30
1.1	Présentation de l'UML (Unified Modeling Language).....	30
1.2	Relation entre UML et SOA	30
2	Méthodologie suivie : RUP	31
3	Diagrammes structurels.....	32
3.1	Diagramme de déploiement	32
3.2	Diagramme de composants	33
4	Etude des cas d'utilisation.....	35
4.1	Diagramme des cas d'utilisation.....	35
4.2	Description des cas d'utilisation	36
4.2.1	Cas d'utilisation : s'authentifier	36
4.2.2	Cas d'utilisation : vérifier l'état d'un dispositif	38
4.2.3	Cas d'utilisation : Commander un dispositif.....	40
4.2.4	Cas d'utilisation : Recevoir des notifications.....	41
5	Diagramme d'interactions	43

Conclusion	44
Chapitre 4: Réalisation	45
Introduction	46
1 Environnement de travail	46
1.1 Environnement matériel.....	46
1.2 Environnement logiciel.....	48
1.2.1 Eclipse	48
1.2.2 Le Framework Knopflerfish.....	49
1.2.3 Le Framework Bootstrap.....	50
2 Implémentation.....	52
2.1 Implémentation des services OSGI.....	52
2.1.1 Utilisation de l'API Android	52
2.1.2 Communication inter-composants.....	53
2.2 Implémentation Androïde	53
3 Réalisation de l'interface de contrôle.....	54
4 Configuration et installation.....	56
4.1 Intégration de knopflerfish sur la plateforme Androïde.....	56
4.1.1 Besoins pour la compilation de la source du framework	56
4.1.2 Etapes d'installation	57
4.2 Installation des composants OSGI.....	58
4.2.1 Configurations	58
4.2.2 Résultat.....	59
Conclusion	60
Conclusion et perspectives	61
Bibliographie	62
NETOGRAPHIE	63
Annexe A: Anatomie d'une application Android	64
Annexe B: Installation et utilisation d'Apache Felix sur Android en utilisant le shell.....	67

Table des figures

Figure 1.Compétences de la société	5
Figure 2 .Diagramme de GANTT	7
Figure 3 : Architecture d'OSGI [6]	19
Figure 4 .Exemple de fichier manifest	20
Figure 5. Cycle de vie d'un composant OSGI	21
Figure 6.Comparaison entre OSGI et Androide	22
Figure 7.Exécution d'application Android et OSGI [7].....	23
Figure 8 .Architecture physique	25
Figure 9.Architecture trois tiers	26
Figure 10.Architecture logicielle de l'application	27
Figure 11.Cycle de vie de RUP [8]	31
Figure 12.Diagramme de déploiement du système	32
Figure 13. Diagramme de composants	34
Figure 14 .Diagramme des cas d'utilisation global	36
Figure 15. Diagramme de séquence "authentification"	37
Figure 16.Diagramme cas d'utilisation « vérifier l'état d'un dispositif ».....	38
Figure 17.Diagramme de séquence « vérifier l'état d'un dispositif »	39
Figure 18.Diagramme cas d'utilisation "commander un dispositif"	40
Figure 19.Diagramme de séquence "commander un dispositif"	41
Figure 20.Diagramme cas d'utilisation "recevoir des notifications"	41
Figure 21. Diagramme de séquence "recevoir des notifications"	42
Figure 22. Diagramme d'interactions	43
Figure 23. La clé DiZik	47
Figure 24.Prototype d'un système domotique	47
Figure 25. knopflerfish	50
Figure 26. Communication entre deux composants OSGI.....	53
Figure 27. Interface d'authentification	54
Figure 28. Interface des chambres.....	55

Figure 29. Interface des dispositifs	56
Figure 30. Exécution du framework.....	58
Figure 31. Etapes de compilation des applications java.....	58
Figure 32. Accès au serveur domotique	60
Figure 33. Anatomie d'une application Android	65

Table des tableaux

Tableau 1. Planning Prévisionnel.....	6
Tableau 2. Les Boutons	11
Tableau 3. Unités de commande à distance	12
Tableau 4. Applications domotiques	13
Tableau 5. Description des états du cycle de vie d'un composant OSGI	21
Tableau 6. Différence entre un serveur OSGI et un serveur ijetty	24
Tableau 7. Comparaison des implémentations OSGI	49

Liste des acronymes

ADT	Android Development Tools
API	Application Programming Interface
CPU	Central Processing Unit
CSS	Cascading Style Sheets
EDI	Environnement de Développement Intégré
HTML	HyperText MarkupLanguage
HTTP	Hypertext Transfer Protocol
IP	Internet Protocol
IR	Infra Rouge
IT	Information Technology
JAR	Java Archive
JEE	Java Enterprise Edition
JSP	JavaServer Pages
JVM	Java Virtual Machine
NTIC	Nouvelles Technologies de l'Information et de la Communication
OS	Operating System
OSGI	Open Services Gateway initiative
PC	Personal Computer

RAM	Random Access Memory
SWT	Standard WidgetToolkit
TV	télévision
UML	UnifiedModelingLanguage
UPNP	Universal Plug aNd Play
UPMS	UML Profile and Meta model for Service
USB	Universal Serial Bus
XMPP	Extensible Messaging and Presence Protocol
XPL	eXtremely simPle protoCoL

Introduction Générale

De nos jours nous ne cessons pas d'entendre parler de l'explosion Internet, de NTIC (Nouvelles Technologies de l'Information et de la Communication), de télécommunications, de multinationales, de sociétés de services, etc.

Avec cette évolution, l'administration d'équipements domotiques à distance est devenue un enjeu majeur. En effet, de nos jours le contrôle à distance est possible puisque à partir des smartphones, tablettes ou ordinateurs, on peut surveiller et piloter notre habitat grâce à la connexion internet à la maison, au travail et même en dehors avec le réseau 3G.

Malgré cette évolution, l'utilisation des systèmes domotiques rencontre des difficultés concernant les technologies utilisées, les prix élevés et la compatibilité.

Il est alors important d'optimiser ces systèmes pour être moins cher, plus simple à utiliser et compatible avec la majorité des équipements. La société Cynapsys est partante pour adopter une solution avec un réseau Zigbee en local pour une consommation assez réduite en énergie, un serveur avec le système androïde et des applications de contrôle à distance compatible avec la plupart des outils de contrôle (Smartphone, tablette, ordinateur, etc.)

Dans notre projet de fin d'étude, nous participons à la réalisation de cette solution. Notre mission est de contrôler à distance les équipements de l'habitat. Pour ce faire, nous avons besoin d'étudier les solutions possibles de contrôle et fixer la meilleure architecture en respectant les choix précédemment choisis.

Afin de présenter le travail de notre projet, nous avons organisé le rapport en quatre chapitres comme suit :

Le premier chapitre introduit le contexte général du projet, la présentation de l'organisme d'accueil, le sujet avec ses différentes problématiques et un planning prévisionnel pour les différentes tâches. En outre, une étude et une comparaison entre différents outils de commande ont été effectuées et enfin la présentation de quelques applications existantes.

Le deuxième chapitre définit les besoins aussi bien fonctionnels que non fonctionnels. Dans ce chapitre, nous décrivons l'architecture des différents modules du projet et des choix technologiques.

Le troisième chapitre expose le choix conceptuel du système. Il mentionne les différents diagrammes adoptés pour la réalisation de notre projet.

Enfin, le quatrième chapitre présente l'environnement matériel et logiciel dans lequel le projet a été réalisé. Nous clôturons par la présentation de quelques interfaces de l'application ainsi que les contraintes de réalisation avant de clôturer ce mémoire par une conclusion et les perspectives de ce travail.

Chapitre 1 : Etat de l'art

Introduction

La Maison Intelligente est une résidence équipée de technologie informatique qui assiste ses habitants dans les situations diverses de la vie domestique en essayant de gérer de manière optimale leur confort et leur sécurité par action sur la maison.

Dans ce chapitre nous exposons le contexte général du projet de fin d'étude. Dans lequel, nous présentons l'organisme d'accueil et le travail demandé. Ainsi, nous décrivons les concepts en relation avec notre projet et l'étude du marché et des applications domotiques existantes.

1 Présentation du projet

1.1 Présentation de l'entreprise

Cynapsys est une société de service en ingénierie informatique et en IT Consulting, certifiée AFAQ ISO 9001 version 2008, accompagne ses clients dans le cadre du développement de leurs applications ou de leurs systèmes d'information. Les services de Cynapsys sont déployés autour du :

- Développement spécifique de logiciel
- Tierce Maintenance Applicative
- Conseil et Assistance technique en TIC
- Etude et Audit sécurité (AMOA, MOA, Schéma directeur)
- Test & Validation
- Formation

En proposant des solutions innovantes dans les domaines des Systèmes d'Information, Systèmes Embarqués et des Applications Mobiles, Cynapsys se positionne en véritable partenaire de services d'ingénierie informatique en Tunisie pour ses clients actifs dans les secteurs des télécoms, automobile, finance, industrie, santé, etc. [1].

Cultivant un sens aigu de l'anticipation et de l'innovation, les ingénieurs de Cynapsys ont une vision claire de l'évolution des technologies et une réelle compréhension des besoins des clients en vue d'offrir un service intégré et tirer parti des avantages des services de

développement informatique au niveau des différentes technologies présentés par la figure suivante:



Figure 1. *Compétences de la société*

1.2 Travail demandé

Dans son orientation vers les nouvelles technologies et vers l'esprit de l'innovation, la société Cynapsys adopte un projet complet de domotique. A cette occasion et afin d'offrir plus de sécurité et de confort, l'équipe du projet a eu l'idée de mettre en place une solution pour contrôler l'habitat à distance en utilisant le réseau internet. En fait, l'accès au web aujourd'hui est de plus en plus aisée et à la portée de tout le monde.

Notre solution visée prend en compte aussi la diversité des terminaux et des différents systèmes d'exploitation et navigateurs sur le marché pour offrir à son client le pouvoir d'accéder à l'interface de contrôle à partir de son ordinateur, tablette ou Smartphone.

2 Etude préliminaire

2.1 Plan du travail

Pour le bon déroulement et l'avancement contrôlé du développement d'un logiciel, un planning détaillé du suivi du projet est nécessaire. Pour cela nous avons estimé les durées de réalisation de chaque tâche, ainsi nous avons mis des dates limites pour les différentes étapes du projet. Nous avons illustré les actions, leurs durées et leurs dates limites dans le tableau 1.

Tableau 1. Planning Prévisionnel

Tâches de conception et développement de l'application			
Taches	Description	Durée	Deadline
Tache 1	Etude de l'existant et solutions possibles	30 jours	15/03
Tache 2	installation et test des outils de développement	15 jours	30/03
Tache 3	Conception :		
	1. Architecture générale de l'application et diagramme de cas d'utilisation 2. Conception détaillée	15 jours 40 jours	30/03 10/05
Tache 4	Implémentation	40 jours	20/05
Tache 5	Tests unitaires	30 jours	20/05
Tache 6	Réalisation de l'interface web	10 jours	30/05
Tache 7	Intégration	5 jours	05/06
Tache 8	Test et validation	10 jours	15/06
Rédaction du rapport			
Chapitres	Titres	Durée de rédaction	Deadline
Chapitre 1	Etat de l'art	30 jours	30/03
Chapitre 2	Spécifications	20 jours	20/04
Chapitre 3	Analyse et conception	30 jours	20/05
Chapitre 4	Réalisation	25 jours	15/06

2.2 Diagramme de GANTT

Pour faciliter le suivi des opérations à entreprendre, éviter les problèmes de communication, l'information doit parfaitement circuler.

Pour cela nous disposons d'outil : le diagramme de GANTT qui a pour objectifs :

- ✓ Coordonner les tâches.
- ✓ Déterminer les délais.
- ✓ Contrôler l'avancement des travaux.

Ce diagramme est un planning représentant graphiquement les différentes actions du travail. Il permet le suivi des différentes opérations mises en œuvre et leur réajustement compte tenu d'éventuels aléas (exemple : retard). Ce dernier a pour rôle de renseigner sur : la durée d'une tâche, le moment où elle débute et celui où elle s'achève au plus tôt et au plus tard.

Le plan de déroulement de la mise en place du planning détaillé est le suivant :

- ✓ Recensement des tâches à réaliser.
- ✓ Détermination des dépendances des tâches et leur ordonnancement.

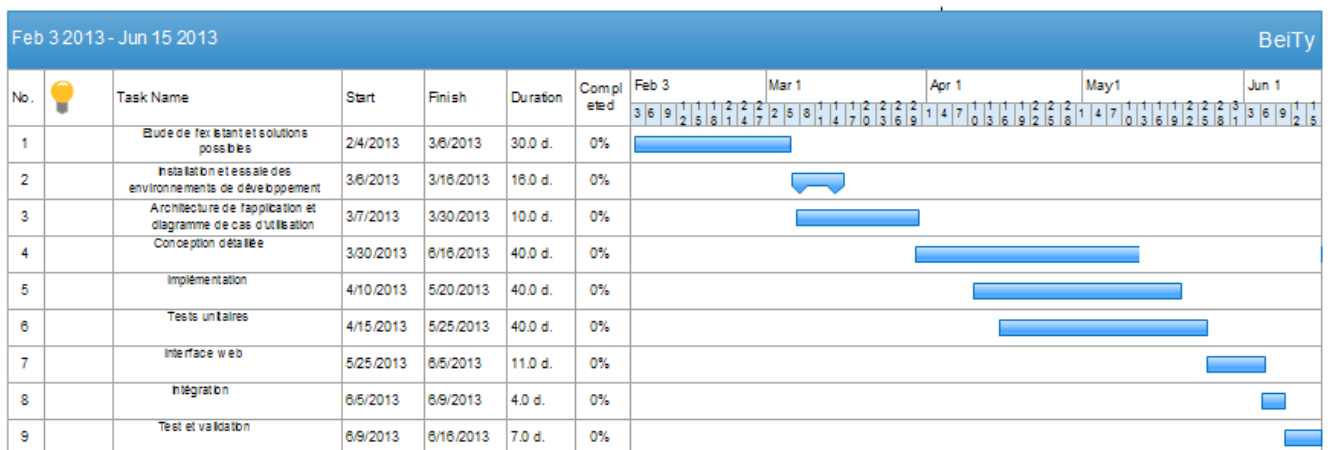


Figure 2 .Diagramme de GANTT

3 Concepts de base

3.1 Domotique

3.1.1 Définition

La domotique rassemble les technologies de l'informatique, des télécommunications et de l'électronique utilisées dans les domiciles. Elle vise à assurer des fonctions de sécurité, de confort, de gestion d'énergie et de communications.

Les appareils de la maison sont intégrés au sein des systèmes qui doivent communiquer entre eux afin de gérer des automatismes [2]

La domotique s'agit d'un système électrique qui permet de communiquer avec des télécommandes ou des boutons poussoirs afin de rendre le contrôle de la maison plus facile. Ils permettent de commander d'un simple geste une ou plusieurs actions (Exemples : baisse du chauffage, réglage de l'éclairage, descente des volets motorisés...). Une application domotique nécessite, pour son fonctionnement, de pouvoir capter une information et d'actionner par la suite une commande. Pour cela, il fallait transférer l'information entre différents dispositifs.

Le système permet de programmer des automatismes qui représente des sources de confort de sécurité et d'économie d'énergie comme l'ouverture et la fermeture automatique des volets, l'arrosage automatique en fonction du climat, la mise en sécurité des stores en cas de vent violent, la gestion de l'éclairage en fonction de la présence et de la luminosité, etc.

3.1.2 Avantages de la domotique

La domotique est l'automatisation des fonctions techniques à des fins d'économie d'énergie, d'augmentation du confort, de la sécurité et de la mobilité pour les constructions nouvelles comme pour les rénovations.

- **Le confort**

L'accroissement du niveau de confort des habitations a été le premier objectif de la domotique.

Les fonctions de commande à distance simples et qui agissent sur différents types d'appareils sont maintenant banalisées. En effet, il est possible d'activer à distance des fonctions qui ont pour but de recréer une ambiance ou un état prédéfinis dans la maison.

Il est donc facile d'imaginer un nombre illimité des fonctions qui pourraient faciliter le confort quotidien dans la maison (par exemple la cafetière s'allume et les volets s'ouvrent à 7h tous les matins) [1].

– La sécurité

En cas de menace pour la sécurité de la maison, tout composant domotique est capable d'émettre un message sur l'installation qui sera repris et traité par un module spécialisé pour la surveillance. Ce module peut alors déclencher n'importe quel composant présent dans l'installation afin de simuler une présence ou bien renforcer la sécurité. Ces actions peuvent se faire selon un choix particulier, selon une durée ou un nombre de détections ou bien directement par téléphone ou par un ordinateur à distance.

Ces actions peuvent être aussi diverses telles que :

- Enclenchement de certaines lumières intérieurs pour simuler une présence (des centaines de combinaisons aléatoires sont possibles)
- Enclenchement d'un signal acoustique destiné à décourager les « visiteurs »
- Création et envoi d'un SMS sur notre portable
- Composition d'un numéro de téléphone afin d'activer un service de sécurité

– La gestion d'énergie

Un système domotique peut diminuer de 40% à 70% la facture d'énergie du domicile sans toucher au confort de vie.

La domotique va influencer la dépense d'énergie sur deux éléments :

- La suppression de la consommation électrique inutile : l'économie d'énergie, c'est avant tout supprimer une énergie dépensée alors qu'elle n'est pas utilisée. En effet, le gaspillage d'énergie peut être limité avec des produits domotiques afin de ne pas avoir de lampe oubliée à la cave pendant plusieurs jours, un éclairage surdimensionné ou une lampe allumée en plein jour
- La distribution de chaleur dans les pièces : Une régulation « intelligente » de l'énergie dans les pièces est essentielle, non seulement pour les factures d'énergie, mais aussi pour le confort de vie. Le réglage des températures est simple et visuel, chaque pièce peut bénéficier d'un réglage qui lui est propre (absence/présence, jour/nuit) et une commande à distance par téléphone est possible afin de mettre la maison sur « confort » ou bien « économie » lors de l'absence des habitants [1].

3.2 La plateforme Androïde

Android est un système d'exploitation Open Source pour terminaux mobiles conçu par Android, une startup rachetée par Google en juillet 2005. Cet OS se différencie principalement de ses concurrents par le fait qu'il est ouvert. Le modèle économique de Google semble très

pertinent, l'adoption d'Android par les fabricants sera probablement rapide du fait de la gratuité d'utilisation pour le constructeur. Voilà pourquoi il nous semble pertinent de travailler sur cette plateforme.

Les applications Android sont développées en JAVA. Android dispose d'un set de bibliothèques qui inclut la plupart des fonctionnalités présentes dans JAVA ainsi que des fonctionnalités supplémentaires pour la gestion de l'interface graphique par exemple. Cependant, les applications ne s'exécutent pas dans la machine virtuelle java sun mais dans la "Dalvik VM". Il s'agit d'une machine virtuelle développée par Google pour Android et adaptée aux systèmes restreints en mémoire et puissance de processeurs. Les classes java doivent être converties au préalable au format dex (grâce à l'outil dx).

Enfin, une autre particularité importante est que chaque application s'exécute dans sa propre instance de Dalvik VM. En effet la Dalvik VM est conçue pour que le système puisse exécuter en même temps plusieurs instances de VM. C'est un avantage pour la stabilité du système, mais complique le partage des ressources et des classes entre les applications [15].

4 Etude l'existant

4.1 Les moyens de commande domotique existants

De nombreux procédés sont capables de commander les équipements de la maison. Il est important de décrire ces différents dispositifs disponibles sur le marché de la domotique, sans lesquels, aucune action de commande à distance ne serait possible.

4.1.1 Moyens de Commande en local

- ❖ **Boutons et interrupteurs** : Plusieurs types de boutons et interrupteurs prennent leur place dans le marché. Dans le tableau suivant, on a essayé d'illustrer les différents types de cet outil de commande.

Tableau 2. Les Boutons

Boutons	Fonctions
Bouton-poussoir	Commande, variation, sonnette, carillon,...
Platine de boutons poussoirs	Rassemblement de fonctions
Bouton inverseur	Commande d'automatismes (Montée / Descente de volets)
Bouton variateur	Gradation d'intensité d'éclairage ou sonore

- ❖ **Télécommande** : Deux types de technologies « traditionnelles » existent en ce qui concerne les télécommandes à savoir la radio (pour radiofréquence) et l'IR (pour Infrarouge).

L'avantage des télécommandes radio est de ne pas avoir besoin de viser l'appareil à commander pour le piloter. Les télécommandes infrarouge sont, quant à elles, beaucoup utilisées dans le monde du handicap (pilotage d'équipements de mobilité notamment) car elles ne sont en aucun cas perturbées par un autre signal infrarouge externe, ce qui octroie donc une certaine fiabilité de fonctionnement pour des personnes complètement dépendantes de ces équipements.

4.1.2 Moyens de commande à distance :

Il s'agit certainement des organes de commande qui se développent le plus actuellement car ils permettent, soit de pouvoir réutiliser un équipement technologique existant pour piloter ces équipements, soit d'investir dans un équipement aux fonctionnalités beaucoup plus étendues qu'une simple télécommande.

Tableau 3.*Unités de commande à distance*

Moyens	Fonctions
Téléviseur	Commande d'équipements sur l'écran de TV via la télécommande
Ordinateur de bureau	Programmation, supervision, commandes ponctuelles, contrôle à distance
Ordinateur portable	Supervision de commandes ponctuelles partout dans la maison, contrôle à distance
Tablette PC	Supervision, télécommande universelle, commande domotique, accès au réseau local, à internet et à toutes les applications informatiques
Écran tactile IR/radio	Commande tactile d'appareils audiovisuels/domotiques
Écran tactile IR/radio/IP	Commande tactile d'appareils audiovisuels/ domotiques et accès au réseau local et à internet
UMPC27	Commande tactile d'appareils audiovisuels et domotiques, accès au réseau local et à internet
Écran tactile propriétaire	Commande tactile d'appareils audiovisuels et domotiques propriétaires (Accès au réseau local et à internet si IP présent)
Assistant Personnel – PDA	Commande ponctuelle d'appareils audiovisuels ou domotiques (IR, Wifi, Bluetooth)
Téléphone mobile	Commande et supervision à distance (caméras, chauffage, sécurité, etc.)

Nous cherchons à adapter notre solution avec les différents moyens de commande à distance qui ont la possibilité de se connecter à internet.

4.2 Applications domotiques existantes

Les applications domotiques sont nombreux sur le marché et leur nombre continue de s'accroître de jours en jours. A l'heure actuelle les principaux équipements contrôlés :

- ✓ L'éclairage (extérieur aussi bien qu'intérieur),
- ✓ Les appareils ménagers (bouilloire, cafetière, réfrigérateur, four, etc.) et domestiques (téléviseur, système de son, etc.),
- ✓ La consommation de l'énergie (chauffe-eau, stores vénitiens, etc.),
- ✓ La sécurité personnelle et résidentielle (agression, rôdeurs, intrusion, feu, vol, etc.),
- ✓ Les équipements audio et vidéo, etc.

Ces applications se différencient essentiellement en système d'exploitation utilisé dans le serveur, protocoles et licence.

Nous illustrons quelques applications dans le tableau suivant pour distinguer les différents critères d'un système domotique :

Tableau 4. *Applications domotiques*

Applications	Licence	Système d'exploitation du serveur	protocoles	Technologies
OpenHAB	Open source	Linux, Windows ou MacOS	XMPP et KNX	OSGI (java)
Pouply	Open source	Linux	XPL	Perl
LinuxMCE	Open source	Linux (kubuntu)	Z-Wave, GSD	C++
Domotiga	Open source	Linux	X10	C
OpenRemote	Commercial	Windows, linux ou MacOS	KNX, X10, Z-Wave	-

Conclusion

Dans ce premier chapitre, nous avons présenté la société d'accueil Cynapsys dans laquelle nous avons élaboré notre projet de fin d'étude. De plus, nous avons présenté une étude préalable pour la réalisation de notre application.

Dans le chapitre suivant, nous procédons à la spécification des besoins ainsi que la précision de l'architecture du projet.

Chapitre 2 : Spécifications des besoins

Introduction

Dans tout système, les fonctionnalités doivent être mises en relation avec un ensemble de besoins utiles. Ces besoins définissent les services dont les utilisateurs s'attendent avoir fourni par le système. Dans ce chapitre, nous définissons les besoins fonctionnels et non fonctionnels. Puis, à partir de cette spécification, nous précisons les choix technologiques désignés pour gâter nos besoins.

1 Problématique

Après l'étude de l'existant, nous trouvons un besoin d'exécuter un API web services sur le serveur domotique afin de pouvoir contrôler l'habitat à distance. Cette implémentation doit répondre aux différents besoins fonctionnels.

Il faut alors installer un serveur HTTP sur la tablette androïde qui représente le serveur domotique dans la solution « Home Automation » pour pouvoir contrôler notre habitat. Ainsi, nous avons besoin de trouver le moyen pour communiquer avec les modules implémentés sur ce serveur pour exposer les services de contrôle à des clients distants via internet.

2 Spécification des besoins

2.1 Besoins fonctionnels

L'analyse des besoins fonctionnels consiste à définir les diverses fonctions de l'application à réaliser, ainsi que les informations manipulées d'une manière complète.

Il s'agit de traduire les besoins du client en fonctionnalités du système à concevoir. Nous proposons dans ce travail de réaliser un middleware pour exposer des services à utiliser par un client web. Ces services permettent à l'utilisateur de contrôler et surveiller son habitat. Nous illustrons dans la suite l'ensemble des fonctionnalités du système :

- **Authentification** : Chaque utilisateur doit s'authentifier pour pouvoir accéder à l'interface de contrôle.

- Visualisation de l'habitat : le système présente en premier lieu l'ensemble des chambres de l'habitat. En sélectionnant une chambre, le système permet aussi de visualiser tous les composants qu'on peut les surveiller ou contrôler.
- Exécution des commandes : Notre application permet à l'utilisateur de contrôler les équipements de sa maison en exécution des commandes (ON/OFF) par exemple pour : allumer ou éteindre la lumière, ouvrir ou fermer la porte, activer ou désactiver le clim, etc.
- Visualisation des états : Il est possible aussi de savoir l'état actuel d'un équipement : température, porte ouverte ou fermée, lampe allumée ou éteinte, etc.
- Envoie des notifications : le système envoie des alertes à l'utilisateur selon des scénarios prédéfinis.

2.2 Besoins non fonctionnels

Plusieurs besoins opérationnels sont à tenir en compte afin que notre application soit plus conviviale, plus simple à utiliser et plus performante.

Les besoins non fonctionnels sont cités ci-dessous :

- Ergonomie des interfaces : l'interface de l'application doit être simple, pratique et similaire à l'application Androïde déjà existante, installée sur la tablette, afin que l'utilisateur puisse le tenir sans se référer à des connaissances particulières. En d'autres termes, les informations doivent être lisibles et faciles à accéder par n'importe quel utilisateur.
- Flexibilité et évolutivité : le système doit s'adapter aux divers changements exigés par la société. Il doit être modulaire garantissant ainsi une évolutivité de la future solution.
- Sécurité : différents niveaux d'accès possibles au système pour les diverses catégories d'utilisateurs du système. Ce besoin est assuré via une authentification.

3 Choix technologiques :

3.1 OSGI: Open Service Gateway Initiative

3.1.1 Présentation de la technologie OSGI

Cette technologie a été définie et est maintenue par l'OSGI Alliance fondée en 1999 qui est un consortium de grandes entreprises informatiques dont le but est de fournir à l'industrie

des spécifications, des implémentations de référence concernant des standards ouverts et des jeux de tests permettant de certifier ou non les implémentations de la norme. Le problème constaté est le suivant : la complexité des logiciels actuels reposants sur de nombreuses bibliothèques, elles même liées à d'autres bibliothèques et dont l'évolution se fait de façon asynchrone au sein d'un même projet ne facilite ni la conception rapide d'applications mais encore moins leur maintenance. L'idée directrice est donc la réutilisation de composants à faible coût, de façon fiable mais surtout :

- une plate-forme de programmation favorisant l'indépendance des composants
- un format de livraison des composants standardisé
- une plate-forme d'exécution permettant le remplacement de composants à chaud

OSGI est une spécification décrivant des composants et leurs interactions et la spécification de la plate-forme d'exécution gérant ces composants (appelés bundles dans la spécification). Il s'agit d'une approche orientée composants dynamique au sein d'un environnement d'exécution Java. C'est aussi une approche orientée services car chaque module peut proposer ou requérir certains services exposés par d'autres modules. La gestion des dépendances est assurée par le système. La version actuelle des spécifications est la R4.2 [6].

3.1.2 Domaines d'application

La plate-forme OSGI est utilisée dans de multiples domaines. A l'origine, OSGI était prévue pour le réseau local d'un foyer. Elle permet d'englober les applications servant à automatiser les tâches ménagères, principe même de la domotique. Mais, elle eût un tel succès que son application a été étendue à d'autres domaines. De nos jours l'OSGI est utilisé dans:

- Les passerelles domotiques
- Le développement des logiciels tel qu'eclipse
- L'automobile
- La téléphonie mobile

3.1.3 Architecture du Framework

La vue en couche de l'environnement OSGI présentée dans la figure 4 montre que le Framework travaille au-dessus de la machine virtuelle Java qui, elle-même, est placée au-dessus du système d'exploitation. Le lien entre le système d'exploitation et le matériel est rendu possible par l'utilisation de pilotes. Ainsi, les bundles peuvent exploiter les ressources du

Framework, de la JVM et du système d'exploitation. Il est à noter que les pilotes du matériel peuvent aussi être représentés par des bundles OSGI.

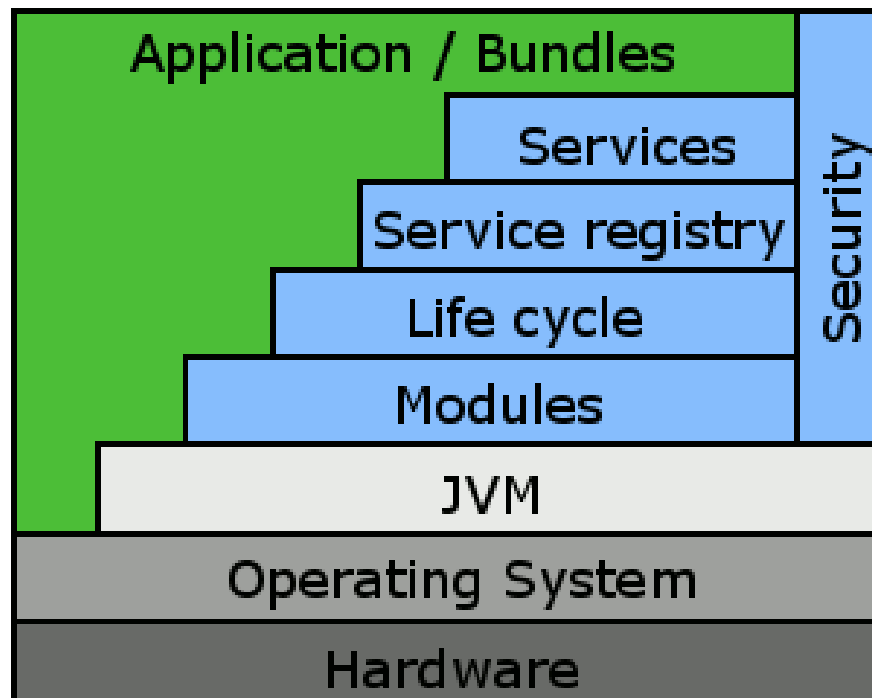


Figure 3 : *Architecture d'OSGI* [6]

3.1.4 Services

La spécification OSGI décrit une multitude de services utiles pour le déploiement et la gestion d'application. Cependant, les services proposés peuvent ne pas être implémentés dans certaines solutions. Nous illustrons quelques services natifs de cette plateforme :

- **SystemBundle** : Ce service représente le Framework. son cycle de vie correspond au démarrage et l'arrêt du Framework
- **PackageAdmin** : Ce service permet la communication entre les différents services
- **ServiceTracker** : Il permet l'appel des fonctions dans un au un autre composant OSGI
- **LogService** : Il permet de journaliser des traces.
- **HttpService** : Il permet à d'autres bundles de publier de servlets et ressources par http.

3.1.5 Description d'un composant OSGI

La spécification décrit comment les composants doivent-être implémentés pour être intégrés à la plate-forme OSGI. C'est l'unité de base au sein de la plate-forme d'exécution. Un composant contient les classes Java et tout autre ressource (image, librairie native, méta-données, aide, code source...), c'est en fait un fichier JAR.

Il doit contenir un fichier *manifest* décrivant son contenu et les informations nécessaires à la plate-forme d'exécution OSGI comme par exemple les dépendances liées en l'absence desquelles il ne pourra être lancé. Ce fichier, *META-INF/MANIFEST.MF* est habituellement présent dans un JAR pour par exemple déclarer la classe principale pour lancer l'application. Il faut donc lui ajouter des instructions pour décrire le composant OSGI. Parmi les instructions courantes pour décrire un tel composant, nous citons :

- Bundle-Activator : spécifie le nom de la classe utilisée pour démarrer et arrêter le composant.
- Bundle-Name : nom du composant
- Bundle-Description : description courte du composant
- Bundle-Vendor : nom du fabricant
- Bundle-Version : version du composant
- Import-Package : spécifie les packages Java requis
- Export-Package : spécifie les packages Java proposés
- Bundle-NativeCode : permet d'indiquer les librairies natives à utiliser et leur environnement

La figure 4 présente un exemple de fichier *manifest* décrivant un composant OSGI.

```
Manifest-Version: 1.0
Bundle-ManifestVersion: 2
Bundle-Name: DoRefresh
Bundle-SymbolicName: DoRefresh
Bundle-Version: 1.0.0.qualifier
Bundle-Activator: dorefresh.Activator
Bundle-Vendor: khalil
Bundle-RequiredExecutionEnvironment: JavaSE-1.7
Import-Package: org.osgi.framework;version="1.3.0"
```

Figure 4 .Exemple de fichier manifest

3.1.6 Cycle de vie d'un composant OSGI

La figure suivante illustre les différents états de gestion des *bundles* par le conteneur ainsi que leurs enchaînements possibles:

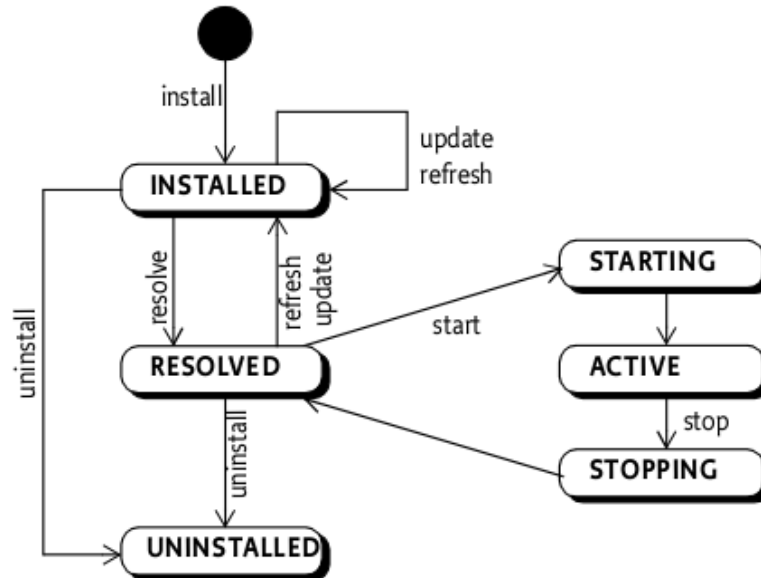


Figure 5. Cycle de vie d'un composant OSGI

Nous décrivons ces états ainsi que leurs principales caractéristiques dans le tableau suivant :

Tableau 5. Description des états du cycle de vie d'un composant OSGI

Etat	Descriptif
Installé (<i>installed</i>)	Etat dans lequel se trouve un <i>bundle</i> juste après avoir été installé, la résolution des dépendances n'ayant pas encore été réalisée.
Résolu (<i>resolved</i>)	Etat dans lequel se trouve un <i>bundle</i> après avoir été installé, la résolution des dépendances ayant juste été réalisée.
En train de démarrer (<i>starting</i>)	Etat dans lequel se trouve un <i>bundle</i> lorsqu'il est en train d'être démarré. Cet état correspond à un état transitoire entre les événements <i>Résolu</i> et <i>Actif</i> .
Actif (<i>active</i>)	Etat dans lequel se trouve un <i>bundle</i> lorsqu'il a été démarré avec succès. Le <i>bundle</i> ainsi que les services qu'il expose sont disponibles pour les autres <i>bundles</i> .
En train de s'arrêter (<i>stopping</i>)	Etat dans lequel se trouve un <i>bundle</i> lorsqu'il est en train d'être arrêté. Cet état correspond à un état transitoire entre les événements <i>Actif</i> et <i>Résolu</i> .
Désinstallé (<i>uninstalled</i>)	Etat dans lequel se trouve un <i>bundle</i> une fois qu'il a été désinstallé.

3.2 Utilisation d'OSGI dans Androïde

OSGI et Androïde définissent une plate-forme de développement orientée composants et fournissent une mise en œuvre de l'architecture orientée services, au sein de l'appareil mobile. La différence clé entre les deux plateformes provient du fait que toutes les applications Androïde sont exécutées dans des machines virtuelles séparées (une instance de VM par application), alors qu'OSGI exécute toutes les applications dans la même VM.

Dans la figure 6, nous précisons une comparaison entre OSGI et Androïde.

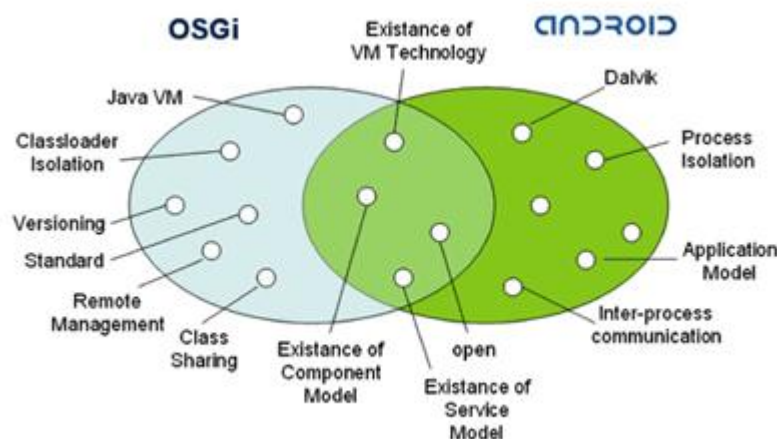


Figure 6. *Comparaison entre OSGI et Androïde*

En fait, exécuter les applications dans des VM séparés c'est-à-dire des processus d'OS séparés a l'avantage de procurer une gestion de ressources plus rigoureuse. Cependant, ce modèle exige plus de mémoire et le partage de services entre les composants est plus complexe et plus coûteux.

Avec OSGI, le partage de ressources entre les différents composants (bundle) se fait par une simple déclaration dans le fichier manifest. Durant l'exécution, tous les bundles partagent le même contexte (classe BundleContext) et peuvent accéder aux ressources auxquelles ils ont droit.

Une représentation de la différence d'exécution des applications entre OSGI et Androïde est exposée par la figure qui suit :

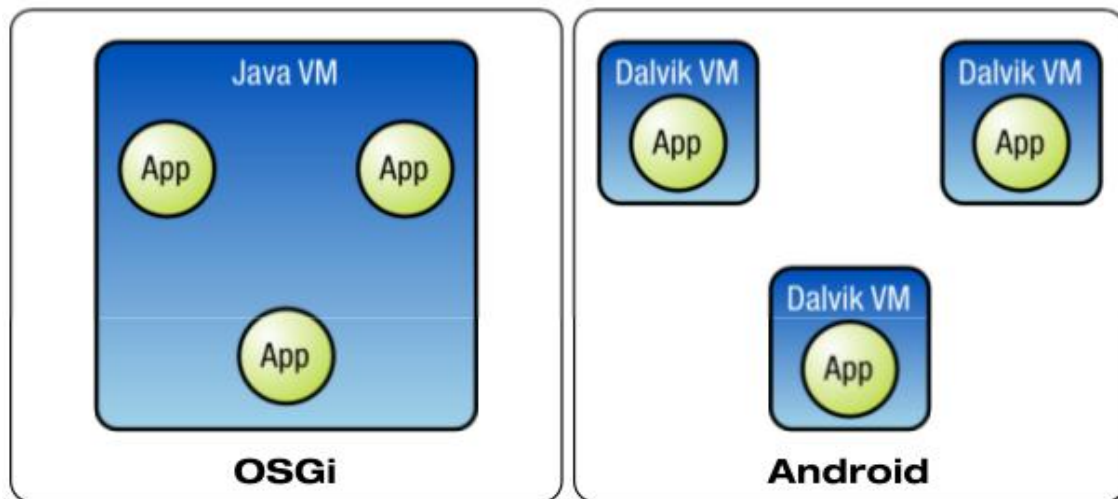


Figure 7.Exécution d'application Android et OSGI [7].

Les deux plates-formes donnent des solutions très différentes pour la même question. Le modèle de processus basé sur l'isolement dans Androïde est un avantage dans la gestion des ressources et le manque de capacité de classe partagé limite le partage des composants et rend la gestion de la plateforme très difficile. Le manque de versioning et d'API de gestion est une autre limitation. L'OSGI fourni des solutions à ces besoins importants, mais ceux-ci seront exclusifs et conduira à une fragmentation de la plate-forme. Bien qu'Androïde ait des caractéristiques prometteuses, il semble y avoir un long chemin à parcourir pour atteindre la maturité [7].

3.3 Avantages d'un serveur OSGI

Il existe sur le marché des serveurs http gratuite et payante pour la plateforme Androïde. Mais dans notre cas, nous avons besoin d'un serveur permettant la réalisation d'une application modulaire et extensible. Une comparaison est mise entre les critères d'un serveur existant tels que: i-jetty et un serveur OSGI :

Tableau 6.*Différence entre un serveur OSGI est un serveur ijetty*

Serveurs Critères	OSGI	i-jetty
Description	Plateforme de programmation favorisant l'indépendance des composants	Serveur http permettant à travers les sockets d'exposer les fonctionnalités de la tablette vers le web
Modularité	Modularité et couplage faible	Couplage fort
Compilation	Compilation à chaud	Nécessité de recompilation
Evolution	S'adapte avec plusieurs protocoles (Zigbee, KNX, UPNP, etc.)	Évolution limité de l'application
Compatibilité avec J2EE	Capable de compiler une application web JEE	N'accepte pas les fichiers JSP

Dans le tableau 6 une comparaison entre le serveur open source ijetty et un serveur intégré dans le Framework OSGI. Cette comparaison montre l'importance et le rôle de l'OSGI dans la domotique.

4 Architecture de l'application

4.1 Architecture physique

La partie réalisée permet à l'utilisateur d'une tablette équipée d'un système d'exploitation Androïde de contrôler des équipements domotiques. La partie manquante est d'avoir un accès de l'extérieur en utilisant l'internet puisque la tablette peut être connectée en wifi.

Afin d'offrir au client la possibilité de commander à distance via internet en utilisant différents moyens, nous présentons dans la figure 8 une architecture générale du projet qui affirme la nécessité d'une application générique indépendante de l'outil de commande à distance (ordinateur, tablette, smartphone, etc.).

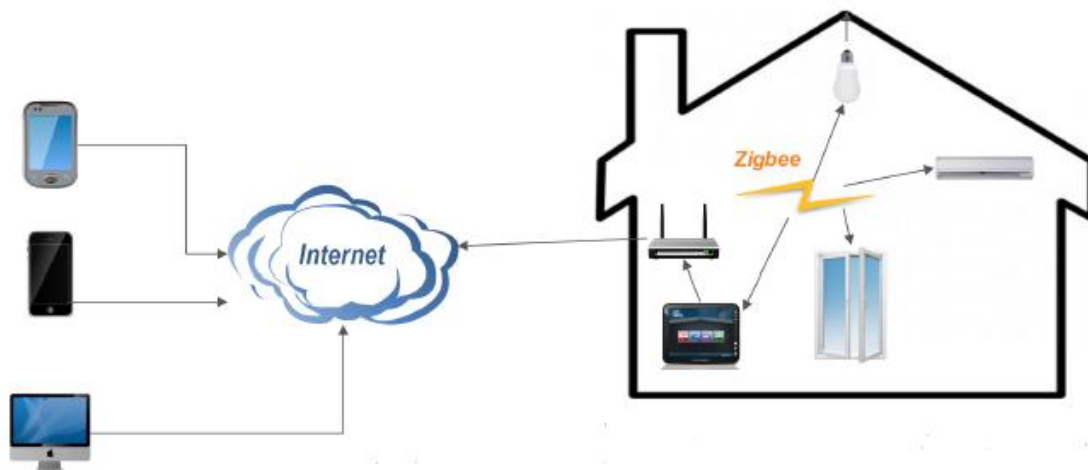


Figure 8 .Architecture physique

Cette figure décrit les différentes parties dans notre système :

- Des appareils de contrôle domotique, avec différents systèmes d’exploitations et différents navigateurs, qu’ils ont la possibilité de se connecter à internet.
- Un serveur Home Automation (tablette ‘Freescale’) équipé d’un système androïde.
- Des équipements pour la communication avec le serveur utilisant le protocole zigbee.

4.2 Architecture logicielle

Notre application est basée sur une architecture trois tiers. Cette architecture est un modèle logique d'architecture applicative qui vise à séparer très nettement trois couches logicielles au sein d'une même application ou système, à modéliser et présenter cette application comme un empilement de trois couches dont le rôle est clairement défini :

- **Couche présentation** : représente l'interface Homme/Machine et permet aux utilisateurs de dialoguer avec le système.
- **Couche objets métier ou traitement** : c'est une couche logicielle entre l'interface utilisateur et la base des données. Ce niveau représente l'ensemble des fonctionnalités

et des règles métier de l'application tels que les contrôles et la vérification du calcul qui s'exécutent au niveau du client.

- **Couche accès aux données** : constitue la base de cette architecture car elle offre tous les services nécessaires d'accès aux sources de données, ces services sont utilisés par la couche d'objets métier ou traitement de cette architecture.

Nous présentons dans la figure 5 une architecture défini les différents couches du système

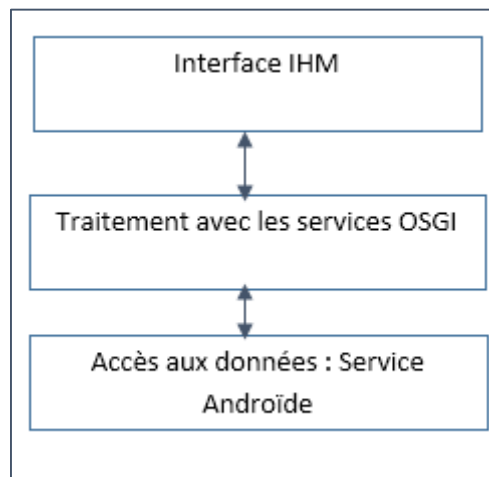


Figure 9. *Architecture trois tiers*

Les trois couches de notre application sont :

- Couche présentation : un portail web générique
- Couche de traitement : un middleware OSGI communique avec un service androïde. Le service androïde est utilisé pour communiquer avec la base des données et le port USB.
- Couche des données : une base des données SQLite

Dans notre travail, nous utilisons la base des données SQLite en mode lecture seulement. L'écriture dans cette base se fait à partir de l'application androïde installé sur la tablette. Cette application contient aussi un service pour échanger des messages avec un port USB ou on peut brancher une clé « Zigbee ». A partir de ce dernier, nous pouvons envoyer des commandes aux différents équipements de la maison.

Pour finaliser le travail nous :

- ajoutons les implémentations manquantes dans le service androïde (*broadcastreceiver*).

- Ainsi nous intégrons le Framework OSGI avec ces différents services à notre plateforme.
- Nous préparons des composants OSGI

Nous essayons de décrire notre application dans la figure suivante avec ces divers composants.

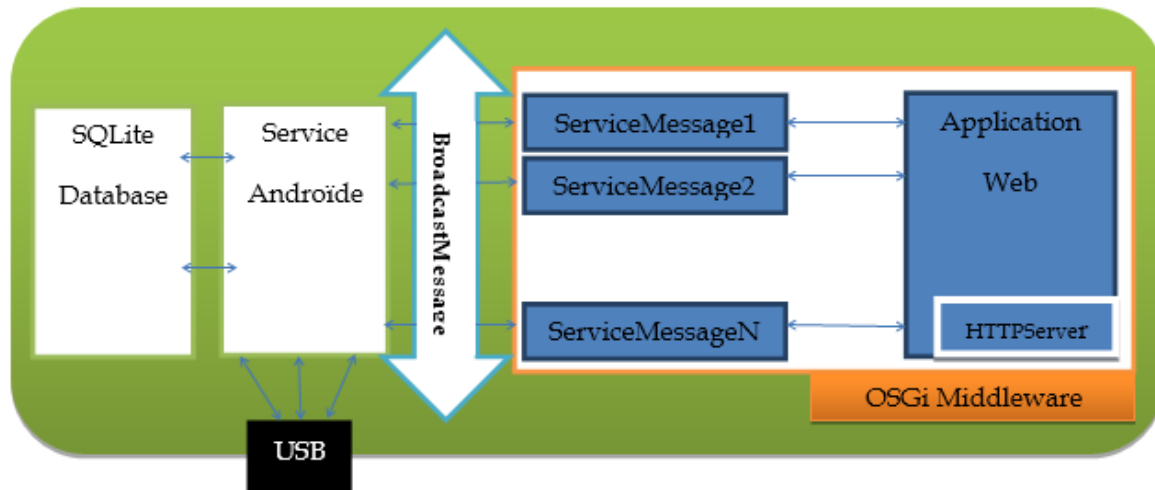


Figure 10. Architecture logicielle de l'application

Les différents composants sont alors :

- Application web : un composant OSGI qui intègre un serveur http.
- Services (Service Message) : des composants OSGI en communication avec le composant précédemment décrit.
- Un service androïde : un service en communication avec les services OSGI
- Une base de donnée SQLite : un registre de tous les détails (chambres, dispositifs, états, etc.)
- Un port USB : ou on peut brancher une clé zigbee pour accéder au réseau local. Ce port est en communication avec le service androïde.

Conclusion

A travers ce chapitre nous avons détaillé nos besoins. Ainsi nous avons justifié nos choix en décrivant la plateforme OSGI puis nous avons précisé l'architecture de notre application.

Dans le prochain chapitre nous procédons par la conception et la présentation de notre travail avec différents diagrammes d'UML.

Chapitre 3 : Analyse et conception

Introduction

L'analyse des besoins et la conception sont deux phases fondamentales dans le cycle de vie d'un logiciel. En effet, ces étapes permettent de mieux comprendre le fonctionnement du système également un bon moyen pour maîtriser sa complexité et pour assurer sa cohérence. Dans ce chapitre, nous allons, en premier lieu, justifier nos choix du langage de modélisation. Ensuite, nous présenterons les cas d'utilisation avec une description détaillée et le diagramme de séquences pour quelques cas.

1 Choix du langage de modélisation

1.1 Présentation de l'UML (Unified Modeling Language)

L'UML est un langage de modélisation graphique et textuel. Il unifie à la fois les notations et les concepts orientés objets. Il permet grâce aux différents diagrammes de comprendre et décrire des besoins, spécifier et documenter des systèmes, dessiner des architectures logicielles. Il est aujourd'hui un standard très largement utilisé dans l'analyse et la conception des systèmes informatiques. L'UML est un moyen d'exprimer des modèles objets en faisant abstraction de leur implémentation. En effet, le modèle élaboré par UML est valable pour n'importe quelles plateformes et langages de programmation.[9]

1.2 Relation entre UML et SOA

UML est la voix naturelle de modélisation pour la SOA. En effet, SOAml en cours de spécification par l'OMG, s'appuie sur UML 2 et est annoncée comme une spécification UPMS (UML Profile and Meta model for Service).

Autrement dit le choix de l'UML n'est pas une obligation. Dans le cas où nous choisissons la modélisation avec Merise, le diagramme d'activité UML sera plutôt le Modèle de Traitement, et le modèle de classe UML sera ainsi le Modèle Conceptuel de données Merise.

UML 2.x est souvent préféré pour des raisons de facilité de formalisme notamment au niveau des diagrammes de composants et des diagrammes de séquences [4].

2 Méthodologie suivie : RUP

Nous avons choisi RUP (*Rational Unified Process*) comme étant une méthodologie permettant de mettre notre application dans son cadre. Elle présente une solution pour remédier à la complexité de la mise en place des fonctionnalités du système.

RUP a pour but de spécifier les différentes phases d'un projet, de définir les tâches de chacun des intervenants, et de contrôler les coûts, les délais et la qualité de l'application logicielle produite.

RUP est qualifié par les caractéristiques suivantes :

- ❖ **Itératif et incrémental:** Le projet est découpé en plusieurs itérations de courte durée qui permettent de mieux suivre l'avancement. A la fin de chaque itération, une partie exécutable du système final est produite, de façon incrémentale.
- ❖ **Piloté par les cas d'utilisation:** Le projet est mené en tenant compte des besoins et des exigences des utilisateurs. Les cas d'utilisation du futur système sont identifiés et décrits avec précision et priorité.

Le processus unifié répète un certain nombre de fois une série de cycles. Comme il est présenté dans la figure 1:

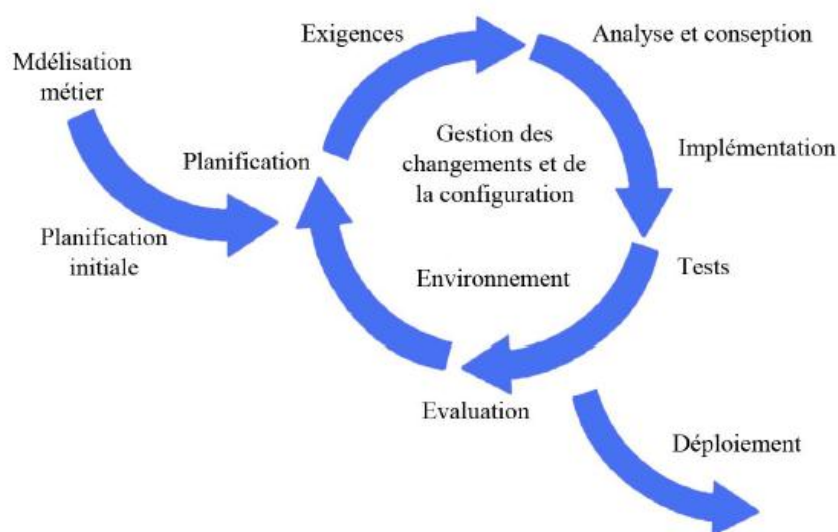


Figure 11. Cycle de vie de RUP [8]

3 Diagrammes structurels

3.1 Diagramme de déploiement

Un diagramme de déploiement est un diagramme de structure qui montre la configuration d'un ensemble des nœuds d'instances exécutables. Nous avons dans notre cas trois nœuds différents. Nous présentons ces trois nœuds ainsi que l'échange entre eux dans la figure 12.

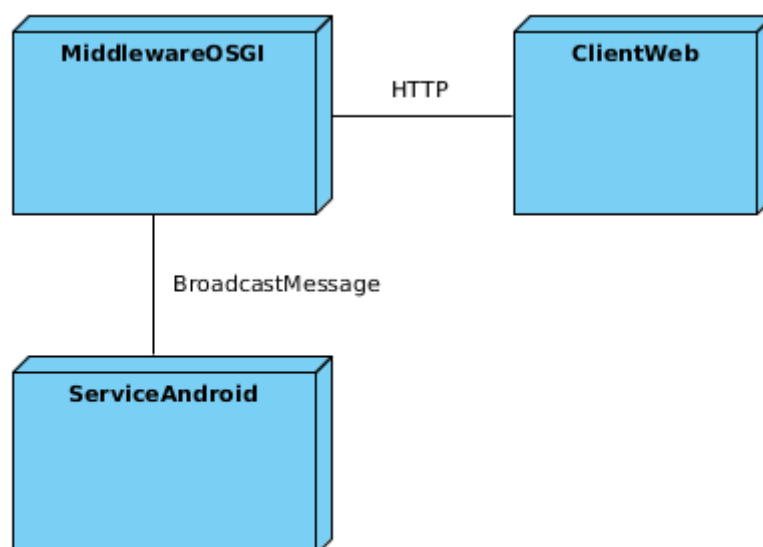


Figure 12. *Diagramme de déploiement du système*

Les trois nœuds dans notre application sont :

- MiddlewareOSGI : C'est un ensemble des services OSGI intermédiaire entre le client web et le service androïde
- ClientWeb : C'est une application web pour exploiter les services OSGI via des requêtes http.

- Service Android : Un service exploité par les services OSGI pour pouvoir accéder au réseau local et la base des données SQLite.

3.2 Diagramme de composants

La programmation orientée composant s'intègre très bien dans le contexte de la programmation orientée objet. En effet, l'utilisation de composants est assimilable à une approche objet, non pas au niveau du code, mais au niveau de l'architecture générale du logiciel.

Un composant doit fournir un service bien précis. Les fonctionnalités qu'il encapsule doivent être cohérentes entre elles et génériques (par opposition à spécialiser) puisque sa vocation est d'être réutilisable.

Un composant est une unité autonome représentée par un classeur structuré, stéréotypé «*component*», comportant une ou plusieurs interfaces requises ou offertes. Son comportement interne, généralement réalisé par un ensemble de classes, est totalement masqué. La seule contrainte pour pouvoir substituer un composant par un autre est de respecter les interfaces requises et offertes.

La relation de dépendance entre les différents composants est utilisée dans les diagrammes de composants pour indiquer qu'un élément de l'implémentation d'un composant fait appel aux services offerts par les éléments d'implémentation d'un autre composant [12].

Nous présentons dans la figure 3 les différents composants de l'application et les relations entre eux à travers un diagramme de composant :

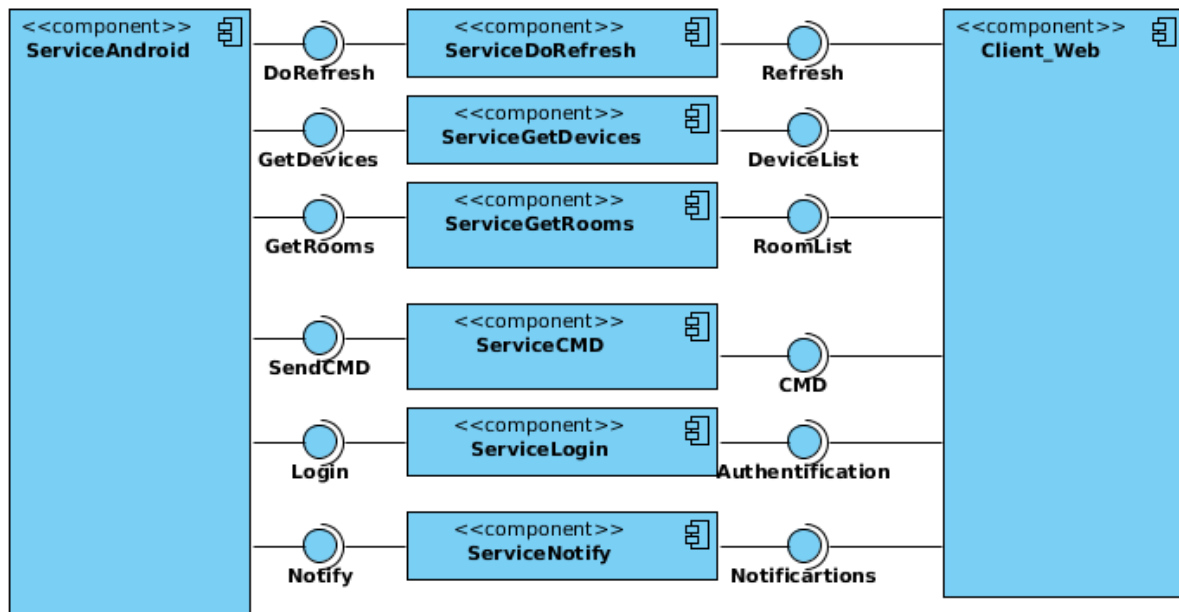


Figure 13. *Diagramme de composants*

Ces composants sont essentiellement :

- **ServiceAndroid** : Ce service est l'intermédiaire entre les services OSGI d'une part et la base des données et le réseau zigbee d'autre part. Ce service est consommé par les services OSGI
- **ServiceDoRefresh** : C'est un service OSGI toujours actif pour récupérer l'état actuel du dispositif
- **ServiceGetRooms** : C'est un service OSGI qu'on l'active dès l'authentification pour afficher les différentes chambres de l'habitat
- **ServiceCMD** : C'est un service OSGI qu'il regroupe les commandes de contrôle.
- **ServiceLogin** : C'est un service OSGI pour valider l'authentification de l'utilisateur.
- **ServiceNotify** : C'est un service OSGI pour notifier le client authentifié selon des scénarios prédéfinis
- **Client_Web** : C'est un Service OSGI sous forme d'une application web représentant l'interface de contrôle.

4 Etude des cas d'utilisation

4.1 Diagramme des cas d'utilisation

Les diagrammes des cas d'utilisation est un moyen simple pour exprimer les besoins fonctionnels. Ils permettent de recueillir, d'analyser et d'organiser les besoins, et de recenser les grandes fonctionnalités d'un système. Il s'agit donc de la première étape de conception UML pour l'analyse d'un système.

Un diagramme de cas d'utilisation présente les fonctionnalités d'un système, d'un sous-système, d'une classe ou d'un composant tel qu'un utilisateur extérieur l'observe. Il partage la fonctionnalité du système en unités cohérentes, les cas d'utilisation, ayant un sens pour les acteurs. Les cas d'utilisation permettent d'exprimer le besoin des utilisateurs d'un système, ils sont donc une vision orientée utilisateur de ce besoin au contraire d'une vision informatique.

Il ne faut pas négliger cette première étape pour produire un logiciel conforme aux attentes des utilisateurs. Pour élaborer les cas d'utilisation, il faut se fonder sur des entretiens avec les utilisateurs [12].

Notre application est dédiée pour un seul acteur comme un simple utilisateur. Il doit s'authentifier afin de :

- Consulter les états des dispositifs
- Commander un dispositif
- Recevoir des notifications

Le diagramme de cas d'utilisation global de la figure 14 représente les fonctionnalités précédemment présentés.

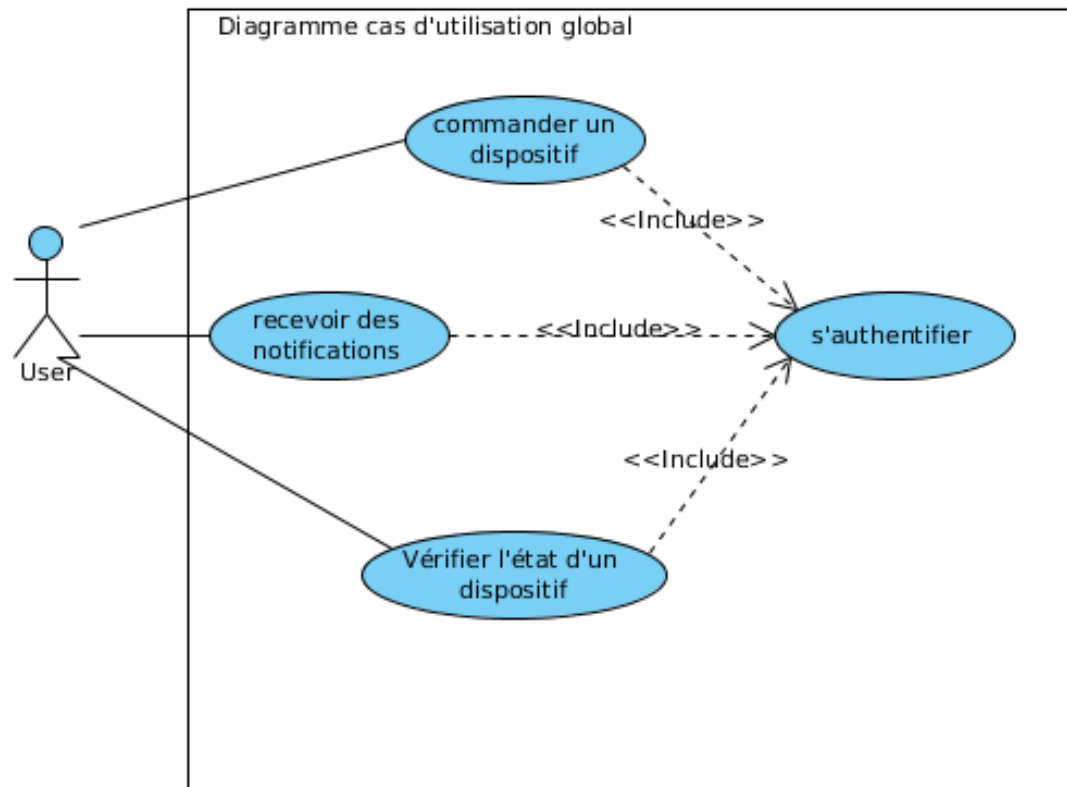


Figure 14 .Diagramme des cas d'utilisation global

4.2 Description des cas d'utilisation

4.2.1 Cas d'utilisation : s'authentifier

- **Scénario nominal**

1. Le client accède au serveur
2. Le client saisie son identifiant
3. Le système active le service login
4. Le service login envoie l'identifiant saisi au service Androïde
5. Le service androïde vérifie l'identifiant saisi
6. Le service androïde renvoie une réponse au service login
7. Le service login traite la réponse du service androïde
8. Si l'identifiant est valide, le système affiche la liste des chambres, si non le système renvoie un message d'erreur

- **Scénario alternatif**

- A1 : Erreur d'authentification**

1. L'identifiant n'est pas accepté
2. Le système renvoie un message d'erreur
3. Le système demande de saisir l'identifiant

- A2 : Identifiant valide**

1. L'identifiant est accepté
2. Le système active le service GetRooms
3. Le service GetRooms envoie une demande au service Androïde
4. Le service Androïde récupère la liste des chambres de la base des données
5. Le service Androïde envoie la liste des chambres au service GetRooms
6. Le Système affiche les chambres

- **Diagramme de séquence**

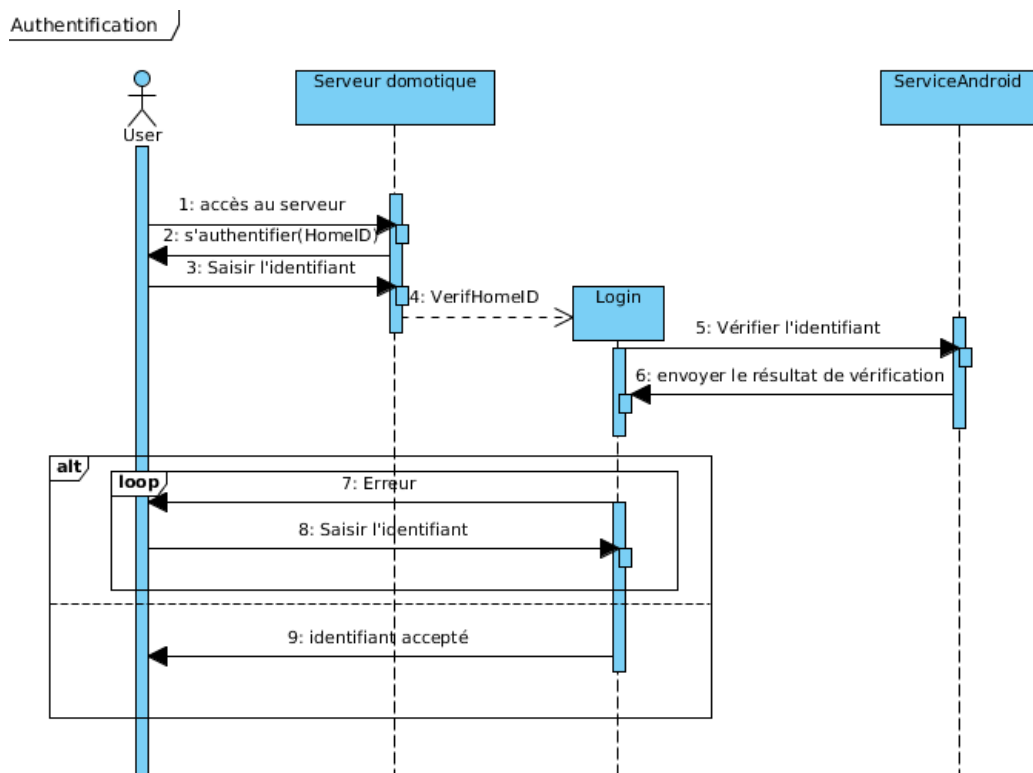


Figure 15. Diagramme de séquence "authentification"

4.2.2 Cas d'utilisation : vérifier l'état d'un dispositif

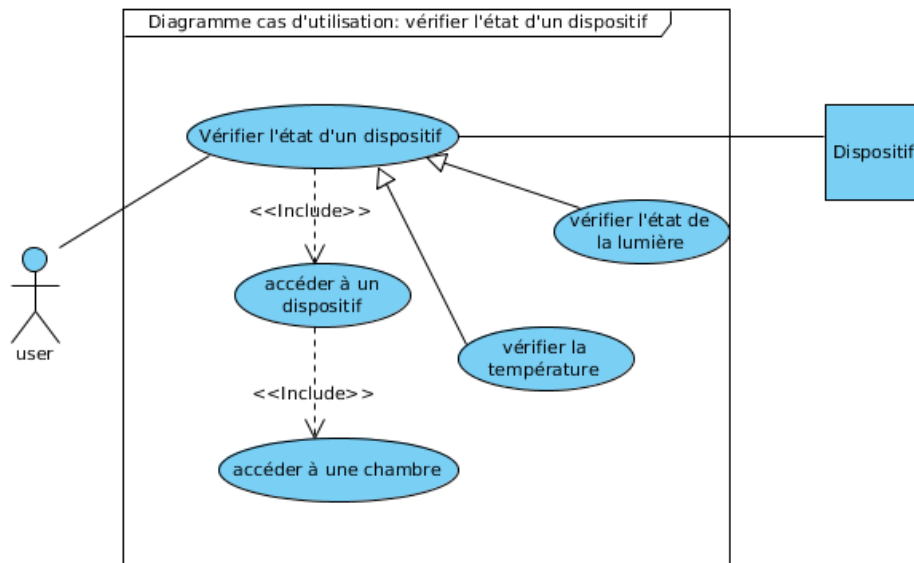


Figure 16.Diagramme cas d'utilisation « vérifier l'état d'un dispositif »

- **Scénario nominal**

1. Le client s'authentifie
2. Le client choisi une chambre
3. Le système active le service GetDevices
4. Le Service GetDevices envoie une demande au service Androïde
5. Le service Androïde récupère la liste des dispositifs de la chambre choisie
6. Le service Androïde envoie la liste des dispositifs au service GetDevices
7. Le système affiche la liste des dispositifs de la chambre
8. Le client choisi un dispositif
9. Le système affiche les propriétés du dispositif choisi

- Diagramme de séquence

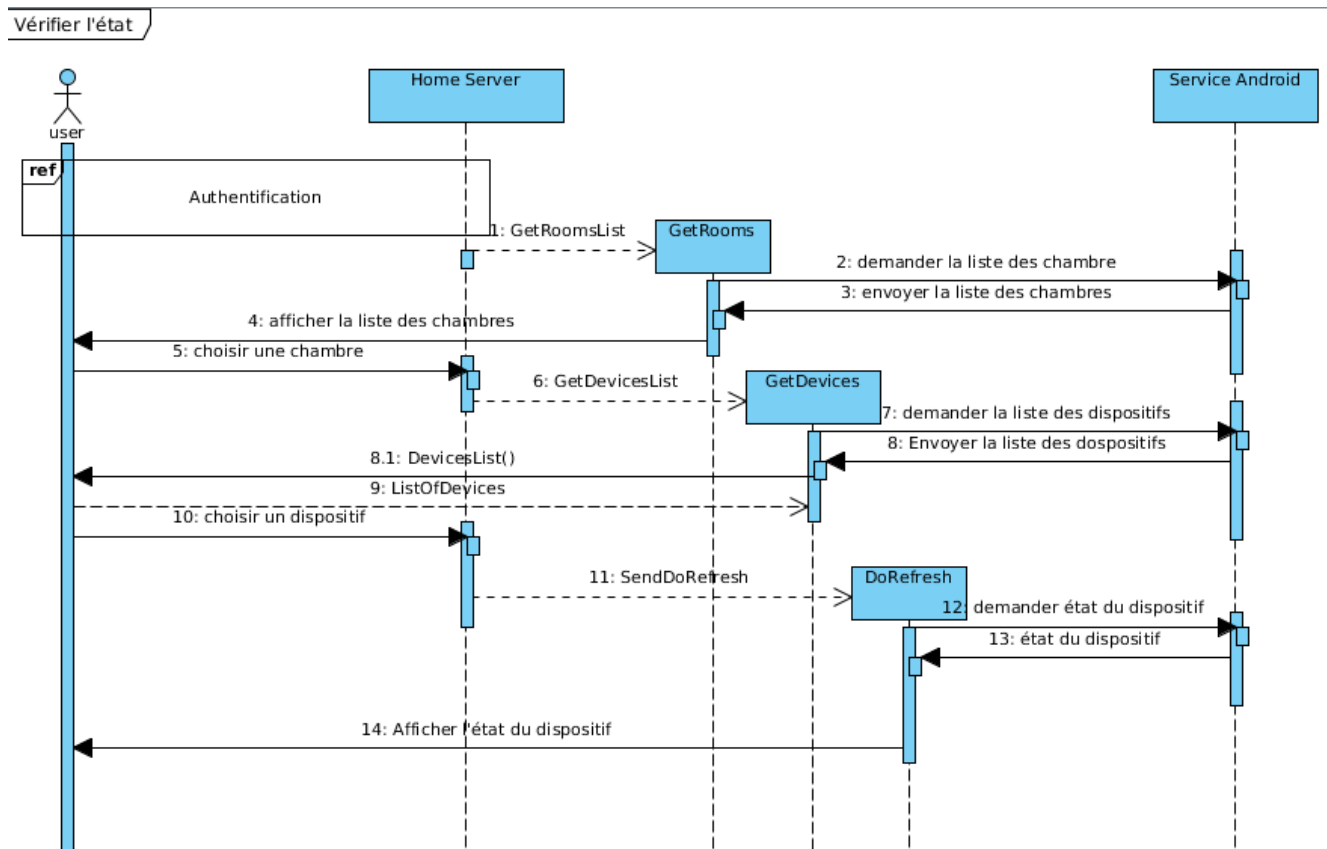


Figure 17.Diagramme de séquence « vérifier l'état d'un dispositif »

4.2.3 Cas d'utilisation : Commander un dispositif

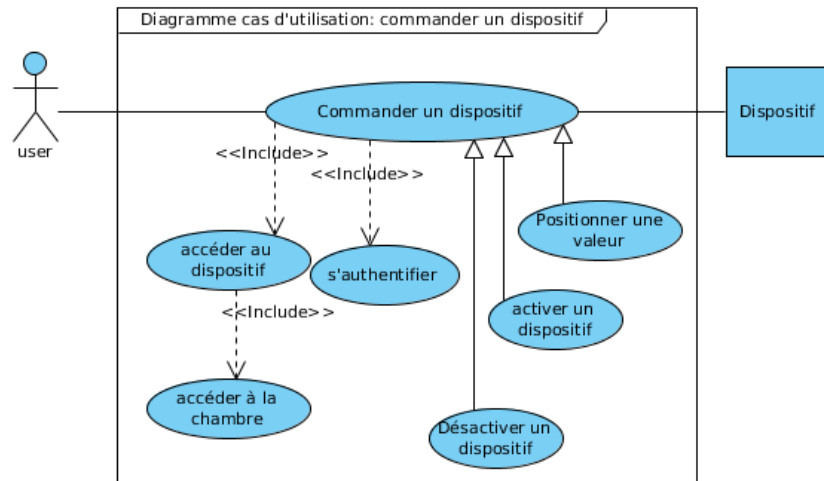


Figure 18.Diagramme cas d'utilisation "commander un dispositif"

Un utilisateur qui consulte l'application va prendre une idée sur les différents états des équipements et leurs envoyer des commandes de contrôle selon le besoin. Aux cas d'un besoin de pilotage (envoi des commandes) nous pouvons considérer le module de consultation d'états comme une étape du module de pilotage. Pour ce fait, nous, allons décrire par la suite le module de pilotage.

- **Scénario nominal**

1. Le client vérifie l'état d'un dispositif
2. Le client demande l'exécution d'une commande
3. Le système active le service CMD
4. Le service CDM envoie une demande au service Androïde
5. Le service Androïde exécute la commande
6. Le service Androïde envoie une réponse au service CMD
7. Le système affiche le nouvel état du dispositif

- **Diagramme de séquence**

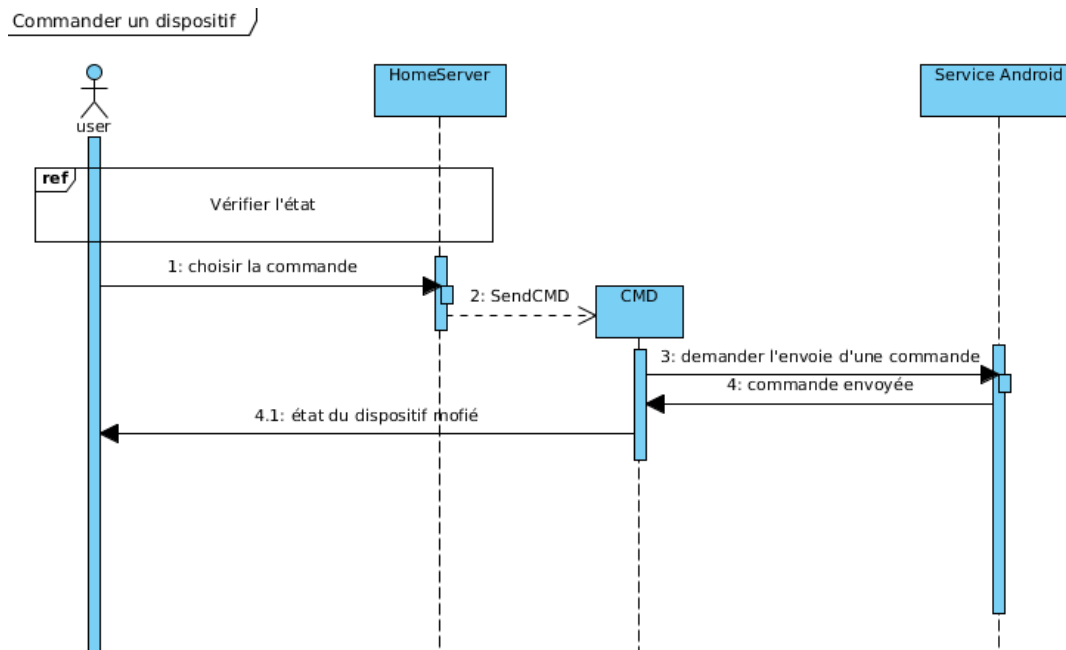


Figure 19.Diagramme de séquence "commander un dispositif"

4.2.4 Cas d'utilisation : Recevoir des notifications

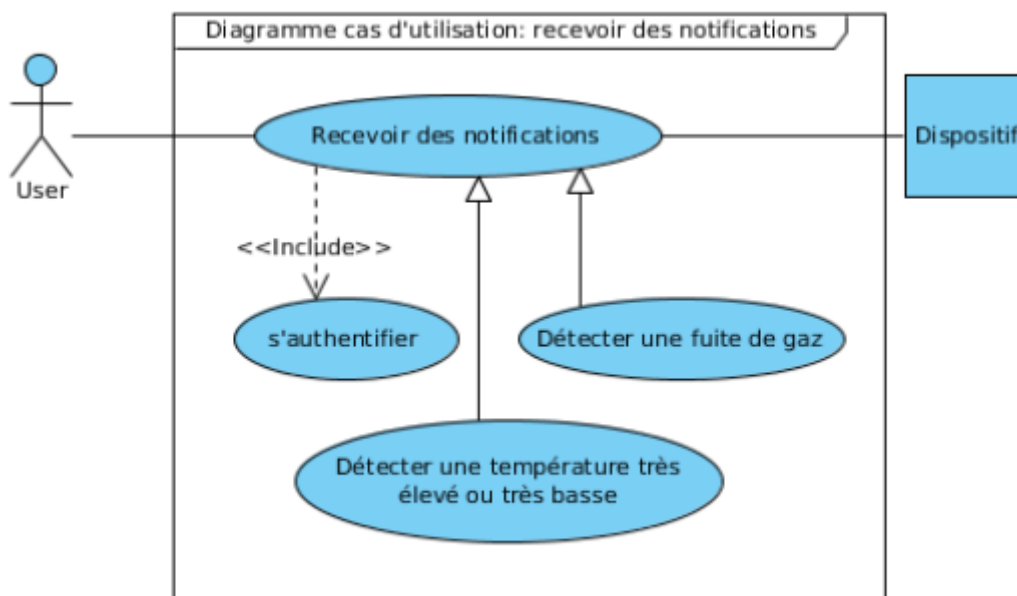


Figure 20.Diagramme cas d'utilisation "recevoir des notifications"

Pour améliorer les fonctionnalités, nous ajoutons un module pour notifier l'utilisateur. En fait un utilisateur authentifié peut recevoir des alertes lors de la détection à partir des scénarios prédéfinis tels que :

- ✓ La détection d'une fuite de gaz
- ✓ Une température qui dépasse la moyenne
- ✓ Alarme de sécurité
 - **Scénario nominal**
 1. Le client s'authentifie
 2. Le système active le service DoRefresh
 3. Le système récupère l'état des dispositifs
 4. Le système parcourt les scénarios de notifications
 5. Le système envoie une notification au client
 - **Diagramme de séquence**

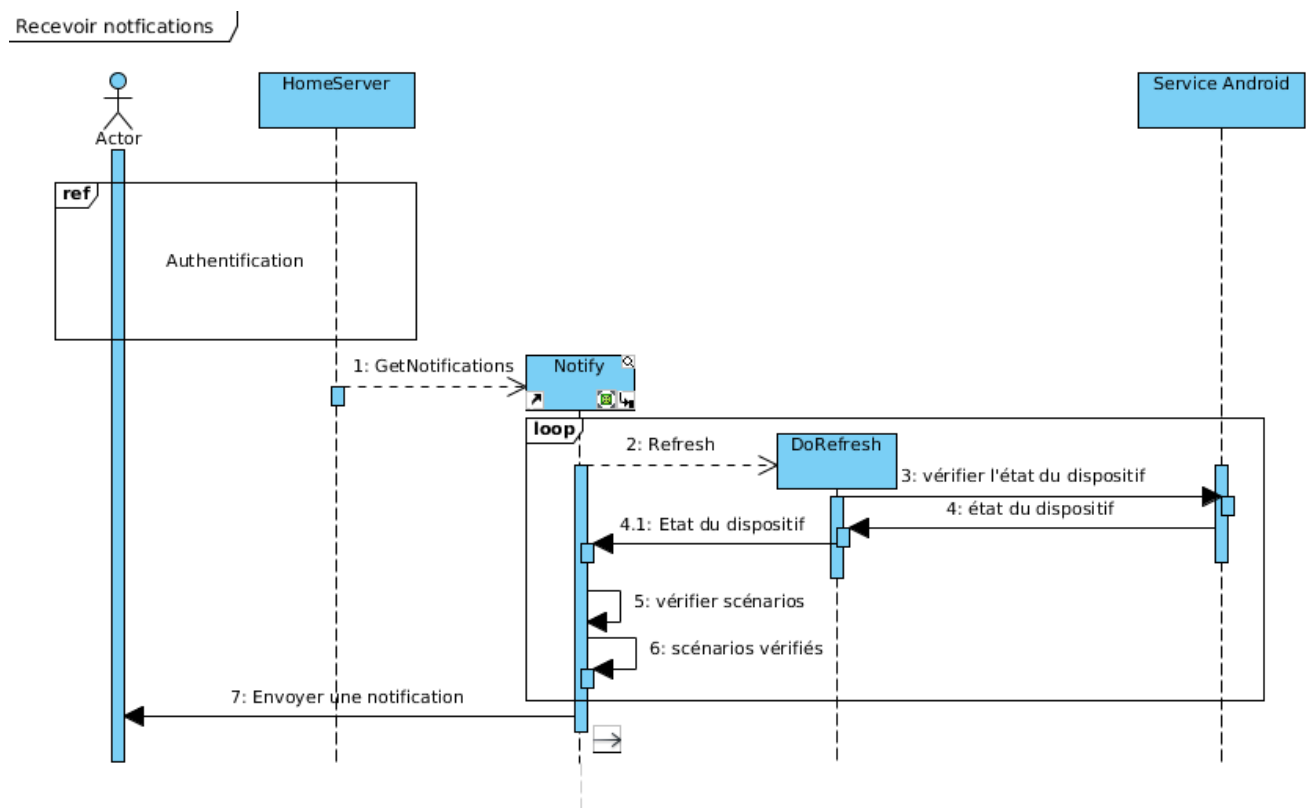


Figure 21. Diagramme de séquence "recevoir des notifications"

5 Diagramme d'interactions

Le diagramme d'interaction ou diagramme d'interactivité est un diagramme UML version 2.0 utilisé pour prendre compte de l'organisation spatiale des participants en interaction.

Dans la figure 11, nous présentons les interactions entre les différents cas d'utilisation de l'application :

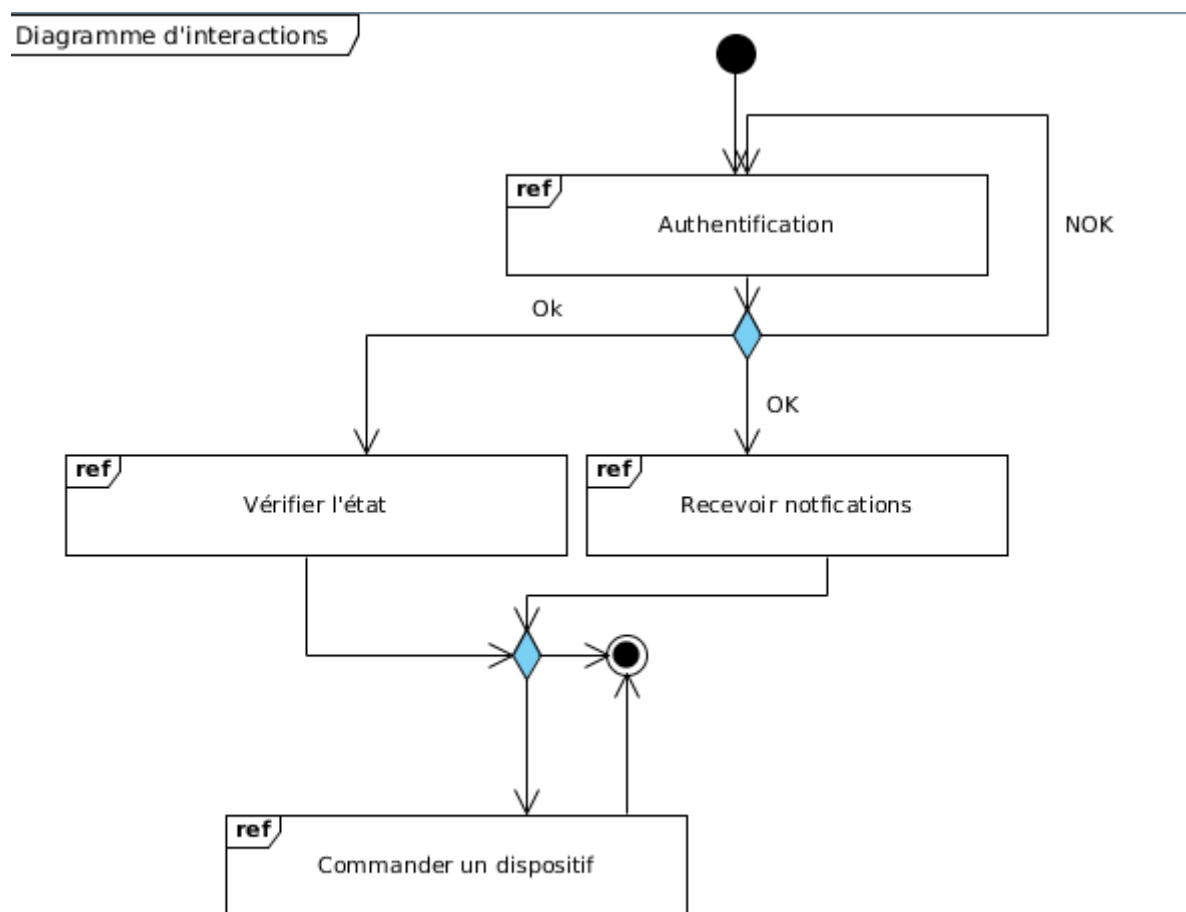


Figure 22. *Diagramme d'interactions*

En effet :

- Si l’identifiant n’est pas accepté, l’utilisateur doit s’authentifier de nouveau
- Si l’identifiant est accepté, il peut vérifier l’état d’un dispositif ou recevoir des notifications puis il peut :
 - Quitter l’application
 - Envoyer des commandes puis quitter l’application

Conclusion

Dans ce chapitre, nous avons présenté le langage de modélisation UML, le processus de développement RUP, l’architecture du système et le modèle conceptuel des données afin de réaliser l’analyse et la conception de notre projet.

Dans le chapitre suivant, nous présenterons les technologies utilisées dans notre application et nous décrirons la phase de réalisation.

Chapitre 4: Réalisation

Introduction

Dans ce chapitre, nous présentons l'environnement logiciel de notre application ainsi que les technologies utilisées.

De plus, nous allons faire une présentation détaillée des différentes phases de la réalisation du projet. Enfin, nous présentons des imprimes écran de quelques interfaces de notre application.

1 Environnement de travail

1.1 Environnement matériel

Du point de vue matériel, nous avons utilisé :

1. Un ordinateur portable possédant les configurations suivantes :
 - Processeur : Intel(R) Core (TM) 2 Duo CPU 2,10GHz;
 - Mémoire : 4 Go de RAM;
2. Une tablette ayant les configurations suivantes :
 - Système d'exploitation : Android 4.1
 - Mémoire : 1 Go de RAM
3. Un prototype d'un système domotique avec :
 - Des modules Zigbee : nous avons utilisés les modules DiZik avec une clé USB pour la connexion avec le réseau.



Figure 23. La clé DiZik

- Des capteurs : Ils sont utilisés pour différentes fonctionnalités telles que :
 - ✓ La détection d'une fuite de gaz
 - ✓ La gestion d'économie d'énergie avec un capteur de mesure niveau de luminosité
 - ✓ Le contrôle des volets électriques de fenêtres
 - ✓ Le contrôle de la variété de luminosité
 - ✓ Le contrôle d'un ensemble d'interrupteurs à distance

La figure 22 présente une image réelle de notre prototype.



Figure 24. Prototype d'un système domotique

1.2 Environnement logiciel

1.2.1 Eclipse

L'EDI Eclipse est un environnement de développement intégré, libre, extensible, universel et polyvalent, permettant de créer des projets de développement mettant en œuvre n'importe quel langage de programmation (Ruby, Cobol, Java, C, etc.). Il est écrit principalement en Java (à l'aide de la bibliothèque graphique SWT, d'IBM).

Eclipse diffère des autres EDI du fait que sa conception est complètement modulaire, basée sur des notions telles que micro noyau (depuis la version 3) ou encore les plug-ins, ce qui fait d'Eclipse une boîte à outils facilement améliorable, modifiable et extensible. De plus, Eclipse peut être exécuté sur plusieurs plateformes telles que Windows, Linux ou MacOS.[17]

Nous avons utilisé Eclipse pour l'implémentation d'un service androïde d'une part et l'implémentation des plug-ins OSGI d'autre part. Pour ce fait, nous avons utilisé la version *Eclipse Juno* et nous avons y installé :

– Plugin ADT

Le point fort d'Eclipse est qu'il repose sur la notion des plug-ins. Pour développer avec Androïde, on peut alors installer Eclipse puis le plugin ADT.

Le plugin ADT (*Android Development Tools*) ajoute des extensions puissantes à l'EDI Eclipse. Il permet de créer et déboguer des applications Androïde plus facilement et plus rapidement en donnant un incroyable élan au développement :

- Il permet une utilisation simplifiée de plusieurs outils fournis par le SDK Androïde tels que le DDMS.
- Il prévoit un assistant de création de nouveaux projets, qui permet rapidement de créer et de mettre en place tous les fichiers de base nécessaires au développement de toute application Androïde.
- Il fournit un éditeur de code qui permet d'écrire en XML.

– Plugin m2eclipse

Ce plugin est l'une des approches utilisées pour le développement des projets *maven* avec l'EDI Eclipse. Il est poussé par *Sonatype* (la société derrière Maven). Un tel projet peut générer un composant OSGI.

Par ailleurs, pour le développement des plug-ins OSGI, nous avons utilisé aussi un outillage spécifique de l'eclipse : le PDE (Plug-ins Development Environment).

















En effet, basé sur l'outillage Java, le PDE améliore l'environnement de développement Java avec des outils prenant en compte les étapes spécifiques au développement des plug-ins (édition des fichiers manifestes, lancement d'un environnement de test, ...).

1.2.2 Le Framework *Knopflerfish*

Il existe de nombreuses implémentations des composants OSGI, aussi bien commerciales qu'open source.

Dans le cadre de ce projet nous avons étudié les différentes implémentations qui sont les plus répandues : Apache Felix Framework, Eclipse Equinox, Knopflerfish et Equinox.

Tableau 7. Comparaison des implémentations OSGI

Détails	Felix	Oscar	Knopflerfish	Equinox
Open source				
Avec interface graphique				
Documentation				
Adaptation avec Androïde				

Nous avons choisi l'implémentation knopflerfish. C'est une implémentation open source des spécifications OSGI, riche en services. Son système de compilation (*ant Build*) possède un

support intégré pour produire des composants OSGI (fichier jar) exécutables sur la plateforme Androïde à l'aide des outils SDK [10].

La figure 24 présente l'interface de cet outil :

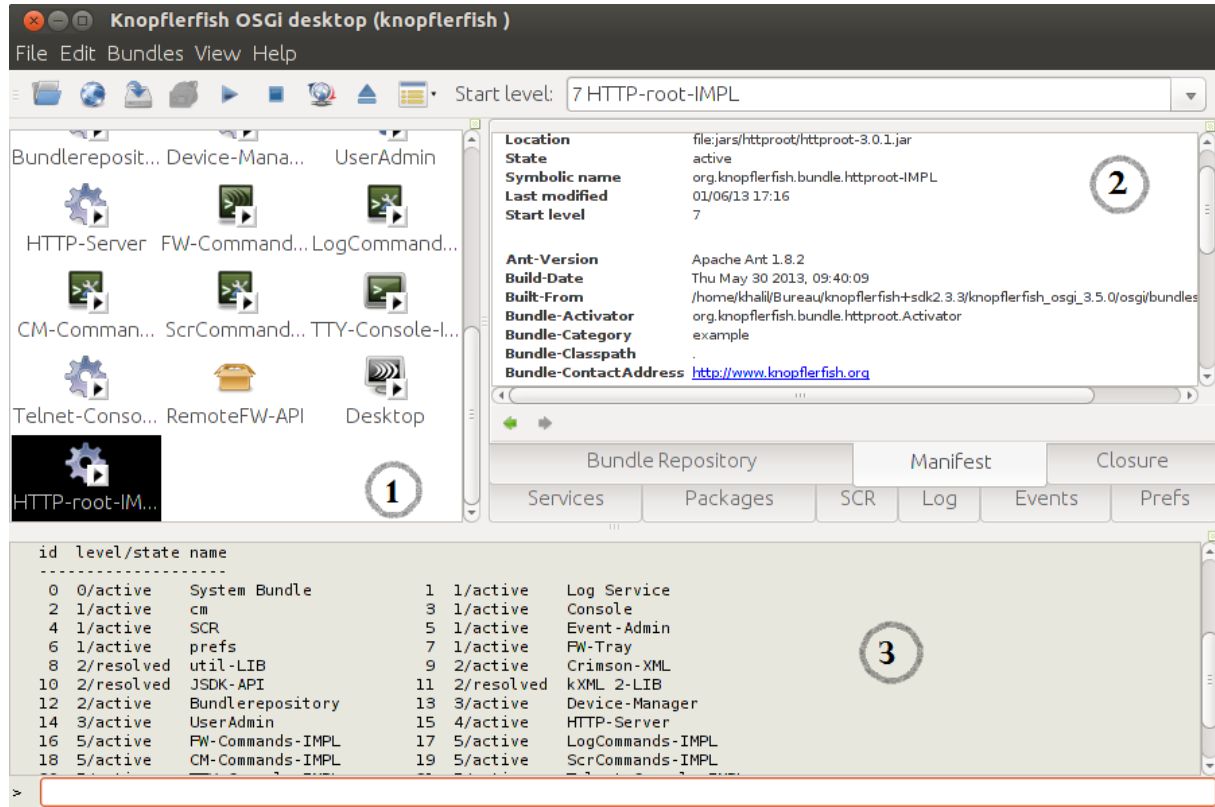


Figure 25. *knopflerfish*

L'interface du Framework contient trois zones :

1. La première zone présente l'ensemble des composants installés
2. Une deuxième zone dans laquelle apparaissent les propriétés d'un composant sélectionné de la première zone
3. En bas une zone pour lancer des commande tels que :
 - *lsb* pour afficher la liste des composants installés
 - *install* pour installer un composant
 - *start* pour activer un composant

1.2.3 Le Framework Bootstrap

Bootstrap permet un gain de temps énorme en proposant des fondations prédéfinies pour la présentation des interfaces web, une normalisation du code en respectant les standards et des

éléments graphiques loin d'être négligeables. En plus, Bootstrap permet de faire efficacement des structures de sites en *Responsive Web Design*, en d'autres termes de prendre en compte et d'adapter le contenu du site Web aux différents types d'appareils, mobiles ou non [13]. Ce Framework est basé sur :

- **JQuery**

Le JavaScript est un langage qui s'interprète directement par le navigateur du client. Il n'y a pas donc besoin de charger une page pour voir l'interaction de celle-ci. Il existe des dizaines de librairies (ou Framework) JavaScript (prototype, mootools, dojo...). Celles-ci permettent d'écrire le code de façon plus courte. JQuery est présenté comme une bibliothèque légère, rapide, simple et puissante c'est qui justifie sa large utilisation. Elle permet grâce à ses plug-ins d'intégrer un bon nombre d'effets assez époustouflants [14].

- **HTML5**

HTML5 est la prochaine révision majeure de HTML (format de données conçu pour représenter les pages web). HTML5 spécifie deux syntaxes d'un modèle abstrait défini en termes de DOM : HTML5 et XHTML5. Le langage comprend également une couche application avec de nombreuses API, ainsi qu'un algorithme afin de pouvoir traiter les documents à la syntaxe non conforme. HTML5 propose un ensemble de nouveaux tags qui permettent une structuration sémantique des contenus : nav, header, footer, section, article, aside. Chacun des nouveaux tags définis permet d'indiquer dans le code le rôle/sens du contenu à l'intérieur du document.

- **CSS3**

Le CSS3 est le nom employé pour caractériser l'ensemble des nouveautés depuis le CSS2.1. Il s'agit par exemple d'un ensemble de nouveaux effets à appliquer sur des éléments HTML. Mais le CSS3 c'est également un ensemble de nouveaux sélecteurs, de nouvelles manières de spécifier les couleurs, une détection des caractéristiques de l'appareil de l'utilisateur, des calculs dans les feuilles de style.

2 Implémentation

2.1 Implémentation des services OSGI

Pour l'implémentation des services OSGI, nous avons utilisé L'EDI Eclipse. Les besoins fonctionnels du projet nécessitent l'intégration de l'API androïde dans un composant OSGI et l'utilisation de quelques spécifications OSGI.

2.1.1 Utilisation de l'API Android

La communication entre un composant OSGI et le service Androïde se fait en utilisant :

- **Les *intents* :**

Ce sont les composants les plus importants de l'architecture Androïde. Les objets de type *Intent* permettent de faire communiquer des activités entre elles. Egalement, ils permettent de communiquer des services.

Plus encore, les objets *intents* permettent la communication des composants au sein de tout le système Androïde.

C'est un concept de la réutilisation de tous les composants des applications. La communication inter applications est possible grâce aux intents qui sont utilisés pour que:

- Une activité lance une seconde activité du programme (ou d'un autre programme Androïde).
- Une activité lance un service d'un programme
- Un *broadcastreceiver* reçoit un intent de la même application ou d'une autre application (type l'utilisateur vient de changer la langue ou le time zone).

Un intent est un objet qui est utilisé pour passer des paramètres (quelconques) d'un composant Androïde à un autre et ce, sans se pré-occuper des processus et de leur gestion mémoire [15].

- **BroadcastMessage**

Il faut importer les classes nécessaires de l'API Android dans un composant OSGI. La gestion des dépendances se fait en ajoutant le fichier android.jar du SDK utilisé.

Pour ajouter le paquetage de cet API, il est nécessaire d'importer le fichier `android.jar` comme bibliothèque supplémentaire nécessaire pour réussir la compilation du plugin sous forme d'un composant OSGI. C'est l'un des utilités du fichier *Manifest*.

En effet l'ajout se fait selon les étapes suivantes :

1. Importer `android.jar` comme un fichier système (*filesystem*) dans notre composant OSGI
2. Ouvrir le *Manifest* du composant et aller dans la fenêtre *build*
3. Ajouter l'API importé dans la zone *Extra ClasspathEntire*

2.1.2 Communication inter-composants

OSGI réalise une structure de chargeur de classes basée sur un réseau. Chaque composant OSGI a son propre chargeur de classe. Ce Framework sépare entre les différents types de paquetages : importés, exportés ou privés. Les classes de paquetages exportées peuvent être accessibles à partir d'autres classes. La figure 24 explique la communication entre deux composants OSGi [6].

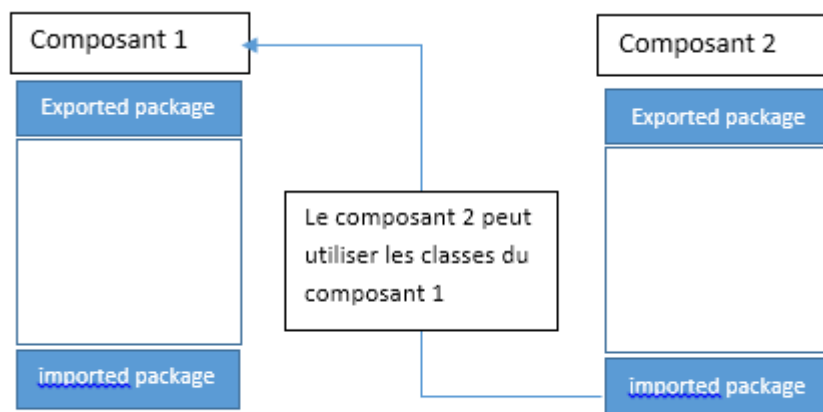


Figure 26. Communication entre deux composants OSGI

2.2 Implémentation Androïde

Le service Androïde est responsable à la communication avec le réseau zigbee d'une part et la base des données d'autre part. La communication est réalisée à travers des messages de

type *BroadcastMessage*. De ce fait, il est nécessaire de comprendre les notions en relation avec les services Androïde [11].

- **Les services**

Ce sont des tâches de fond, des objets qui une fois instanciés (via des intents), restent en 'background'. Ils peuvent survivre au composant qui les a lancés.

Nous pouvons démarrer un service ou l'arrêter, se lier à un service ou s'en séparer.

Il est aussi possible en l'associant à un *broadcastreceiver*, de lancer un service au boot d'Android.

- **Broadcastreceiver**

Ce sont des composants qui sont spécialisés pour recevoir des communications "broadcast" sous forme d'intents.

Les "receivers" vont donc écouter certains intents.

3 Réalisation de l'interface de contrôle

La création de l'interface de contrôle est l'un des éléments moteurs de la réussite du projet. Ainsi, elle doit être ergonomique, puissante et cohérente.

Nous mentionnons ainsi une liste de quelques interfaces de notre application. La première interface est celle de l'authentification.

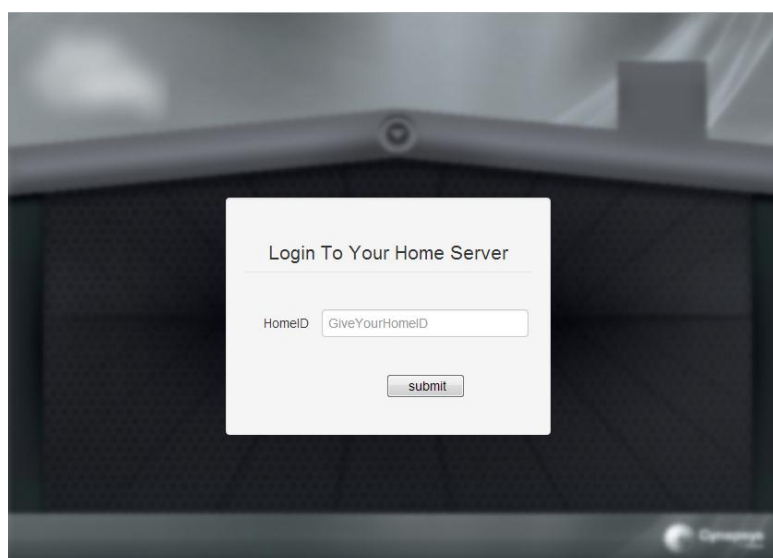


Figure 27. Interface d'authentification

Cette interface contient une zone pour saisir l'identifiant. L'authentification est une sorte de sécurité pour l'application.

Une fois l'identifiant est saisi, l'utilisateur clique sur le bouton apparaissant sous la zone de saisi qu'elle fait appel au service Login pour la vérification. Si l'identifiant saisi est accepté le système active le service GetRooms pour afficher les chambres que l'utilisateur peut surveiller ou contrôler, ainsi que les services DoRefresh et Notify. L'interface des chambres est similaire à celle dans la figure 28.



Figure 28. *Interface des chambres*

Après avoir à l'interface de la figure 27, il suffit que l'utilisateur choisisse la chambre et clique sur son image pour activer le service GetDevices et afficher la liste de ses dispositifs comme il apparaît dans la figure 29.



Figure 29.*Interface des dispositifs*

4 Configuration et installation

4.1 Intégration de knopflerfish sur la plateforme Androïde

En se basant sur le système de compilation du Framework knopflerfish comme implémentation choisie d'OSGI et à l'aide de la documentation de l'organisation *Makewave*. Nous avons réussi à intégrer les composants OSGI à la plateforme Androïde. [10]

4.1.1 Besoins pour la compilation de la source du framework

L'intégration du Framework knopflerfish sur la plateforme androïde nécessite :

1. La source du knopflerfish (la version 3.2 au minimum) : la source du Framework est disponible sur le site officiel de l'organisation *makewave*. Elle contient l'ensemble des spécifications OSGI ainsi que quelques exemples d'application.
2. Android SDK (3.1 au minimum)
3. Un appareil androïde rooté : le routage est nécessaire car l'installation du Framework n'est fonctionnel que sous le répertoire /data du système Android.

4.1.2 Etapes d'installation

Après avoir téléchargé la source du knopflerfish sous forme d'un fichier d'extension .jar, nous suivons les étapes suivantes pour l'installer sur la plateforme Android :

1. Installer le Framework avec la commande :

```
java -jar knopflerfish-3.5.0
```

2. Compilation de la source :

Dans cette étape nous allons compiler la source téléchargée en précisant le lien vers notre SDK android, avec la commande *ant* sous le répertoire de la source du framework :

```
ant -DANDROID_HOME="sdkpath"
```

Puis en se plaçant sous le dossier *tools/android* on lance la commande:

```
ant setup -DANDROID_HOME="sdk path"
```

3. Installation : Nous finissons cette tâche par l'installation du framework en utilisant la commande :

```
ant install
```

Cette commande se lance en se plaçant sous le répertoire *tools/android/* de la source du framework. Elle mène à la création du framework sous le répertoire */data/kf-<version>/* du système Android.

Il est possible ainsi de lancer le framework sous android après avoir se placer sous le répertoire du framework en utilisant la commande :

```
dalvikvm -classpath framework.jar \
```

```
org.knopflerfish.framework.Main -xargsdalvik.xargs
```

La capture de l'exécution du framework sous Android est illustrée par la figure 30.

```

khalil@khalil-PC:~/Bureau/KnF&SDK/sdk/platform-tools$ adb shell -s.2.0 & ls
root@android:/ # cd data/kf-3.3.0/
root@android:/data/kf-3.3.0 # ls
dalvik.xargs
framework.jar
jars
props.xargs
root@android:/data/kf-3.3.0 # dalvikvm -classpath framework.jar \
> org.knopflerfish.framework.Main -xargs dalvik.xargs
Knopflerfish OSGi framework launcher, version 5.3.3
Copyright 2003-2012 Knopflerfish. All Rights Reserved.
See http://www.knopflerfish.org for more information.

Created Framework: org.knopflerfish.framework, version=5.3.3.
> [stdout] Framework launched
lsb
  id  level/state name
  ---
  0  0/active  System Bundle
  2  1/active  cm
  4  1/active  Event-Admin
  6  2/resolved JSDK-API
  8  5/active  FW-Commands-IMPL
 10  5/active  CM-Commands-IMPL
 12  7/active  HTTP-root-IMPL
  1  1/active  Log Service
  3  1/active  Console-Impl
  5  2/resolved util-LIB
  7  4/active  HTTP-Server
  9  5/active  LogCommands-IMPL
 11  5/active  TTY-Console-IMPL

```

Figure 30. Exécution du framework

4.2 Installation des composants OSGI

4.2.1 Configurations

Après avoir implémenté les composants OSGI nécessaires, il est possible de les installer dans le framework par deux méthodes différentes :

– Première méthode

Il est possible de préparer un composant OSGI pour l'activer sur Androïde. En fait il faut adapter le fichier Jar qui fonctionne sur une machine *JVM* pour être fonctionnel sur la machine *Dalvik* de l'Androïde.

Cette technique nécessite la compréhension et le suivi des étapes de compilation des applications java comme elle présente la figure29.

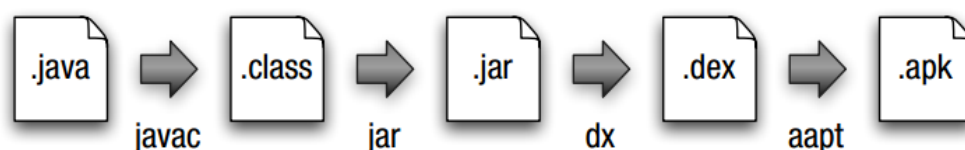


Figure 31. Etapes de compilation des applications java

L'adaptation de ces composants qui doivent être testés sur la machine JVM pour qu'ils fonctionnent sur Android se fait en lançant les deux commandes sur le fichier jar :

✓ `dx --dex --output=classes.dex Service.jar`

✓ `aapt add Service.jar classes.dex`

Après avoir adapté notre service, il suffit d'avoir une copie sur la plateforme et puis l'installer avec la commande :

Installfile://"Servicepath"

– Deuxième méthode

Cette méthode est basée sur la configuration des fichiers de type *XARGS* à la phase d'installation du framework. En effet, la source du framework contient l'ensemble des services OSGI sous forme des fichiers jar chacun dans un dossier. Nous pouvons procéder de la même manière et mettre chaque composant développé dans un dossier afin d'éviter les problèmes lors de la préparation de ces composants pour les fonctionner sur la plateforme Android.

Sous le répertoire de la source */tools/android*, le choix des services OSGI à installer sur la plateforme Android se fait à travers le fichier *template.dalvik.xargs*. Ce fichier contient deux types de commande :

- Install

Exemple : `-install @httproot-N.N.N.jar@`

- Start

Exemple : `-start @httproot-N.N.N.jar@`

4.2.2 Résultat

Nous arrivons enfin, à créer un serveur domotique qu'utilise le système d'exploitation Android et intégrant un serveur http. Il est devenu ainsi possible d'accéder aux services de contrôle à partir d'une interface de contrôle via le web.

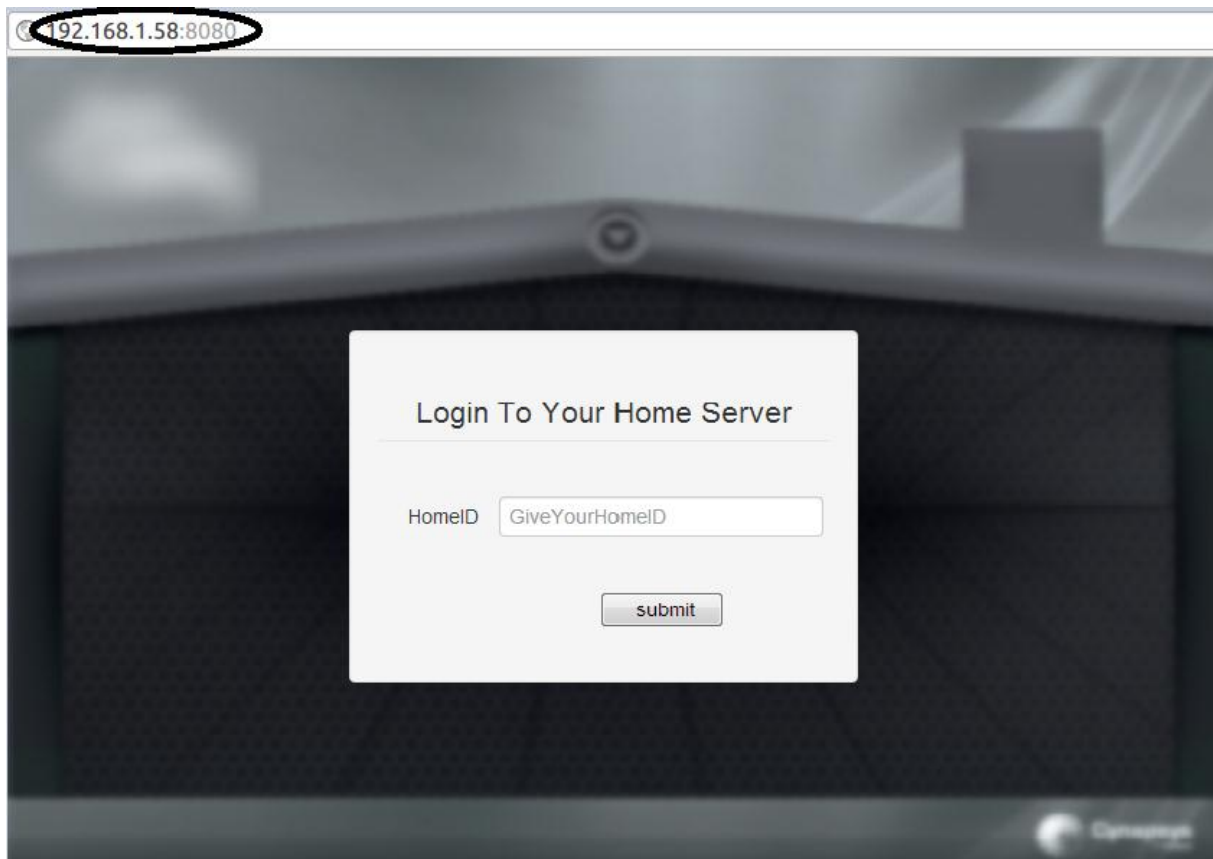


Figure 32. *Accès au serveur domotique*

Dans la figure 32, nous montrons l'exécution du serveur. Il suffit de fixer son adresse dans le réseau, l'accès et le contrôle sont devenus possibles.

Conclusion

Ce chapitre a été consacré pour la présentation détaillée de la réalisation des différentes parties du projet en commençant par la démonstration de l'environnement du travail matériel et logiciel. Ensuite, une illustration de quelques interfaces conçues de l'application avec explication des passages a été réalisée. Enfin, le chronogramme du déroulement du travail a été présenté.

Conclusion et perspectives

Notre travail consiste à l'étude, la conception et la réalisation d'une application de contrôle via le web d'une plateforme domotique. Ce travail s'effectue dans le cadre de la recherche et développement d'une solution « Home Automation » que Cynapsys est en train de mener sur les technologies domotiques.

L'apport de notre projet se résume dans le développement et l'intégration d'un middleware modulaire à la plateforme Androïde qui nous aura permis d'exposer des services pour le contrôle des équipements domotiques à des clients web. Pour ce faire, nous avons eu recours à des services du Framework OSGI pour gérer la plateforme et offrir la possibilité d'accès à distance à nos services. Nous avons intégré un serveur http à notre plateforme domotique puis nous avons procédé par implémenter des services permettant la communication entre les dispositifs de l'habitat et un client distant.

Grâce à son état extensible, cette application dans son état actuel peut être enrichie par d'autres fonctionnalités. Ainsi, il est possible d'exposer les services OSGI en web services afin d'assurer l'hétérogénéité des applications de contrôle à distance.

Bibliographie

- [1] Maxime TACHON et Patrice BARBEL, *Contribution à un état de l'art de la domotique orienté action publique*, 2009.
- [2] Marc-Antoine Boryczka, thèse, *Architecture logicielle pour l'habitat intelligent*, 2011
- [3] Michael Eckel, *Component Technologies on Google Android*.
- [4] Virginie ELIAS, mémoire de master, *Architecture SOA*

NETOGRAPHIE

[5] www.cynapsys.de

[6] <http://www.osgi.org>

[7] <http://www.adon-line.de/kunden/prosystBlog/?p=18>

[8] http://thieum22.free.fr/Quest_RUP.htm

[9] [<http://www.uml.org/>]

[10] <http://www.knopflerfish.org>

[11] <http://www.tutos-android.com/broadcast-receiver-android>

[12] <http://laurent-audibert.developpez.com/Cours-UML/html/index.html>

[13] <http://twitter.github.io/bootstrap/>

[14] <http://jquerymobile.com/>

[15] <http://www.android.com/>

[16] [http://developer.android.com/index.](http://developer.android.com/index)

[17] <http://www.eclipse.org/>

Annexe A: Anatomie d'une application Android

Il y a quatre éléments de base dans une application Android :

- Activity
- BroadcastReceiver
- Service
- Content Provider

Il n'est pas évident que toutes les applications sous Android contiennent obligatoirement ces quatre éléments, mais une application doit s'écrire avec une certaine combinaison de ces éléments. Chaque élément doit être déclaré dans l'AndroidManifest.xml avant d'être utilisé dans n'importe quelle application.

1. Activity

Une activité (Activity) est un écran d'une application. Chaque activité est implémentée par une classe unique qui étend la classe de base Activity (ou une de ses dérivées : ListActivity par exemple). Cette classe affiche une interface utilisateur composée d'éléments graphiques (Views) et répond à des événements. Une application est une succession d'écrans, donc d'activités.

Quand une nouvelle activité est créée, la précédente est mise en pause et est placée dans la pile d'activités, ainsi l'utilisateur peut y naviguer. Lorsque la mémoire vient à manquer, les activités du bas de la pile sont détruites.

Android utilise une classe spéciale pour passer d'une activité à une autre : Intent. Les deux éléments importants que contient un objet Intent sont les traitements et les données à traiter. Les données sont des ressources représentées sous formes d'URI.

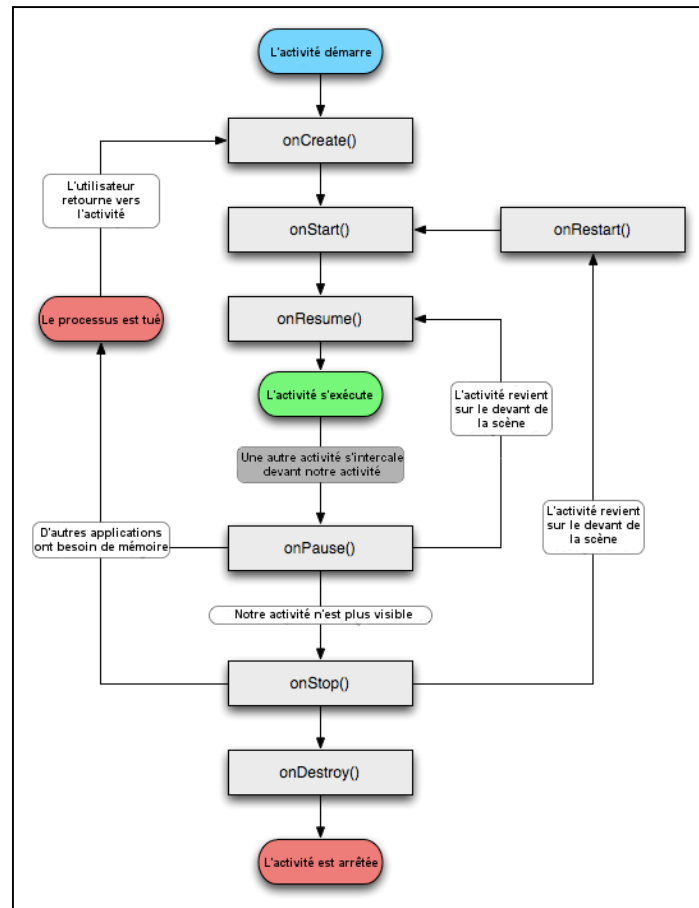


Figure 33. Anatomie d'une application Android

2. BroadcastReceiver

Les IntentReceivers peuvent être utilisés pour réaliser certaines opérations en réaction à un événement extérieur (par exemple quand le téléphone sonne, ou lorsqu'il est minuit). Ils n'affichent pas d'interface graphique, cependant ils peuvent utiliser le NotificationManager intégré à Android pour alerter l'utilisateur que quelque chose s'est produit.

Les IntentReceivers peuvent soit être enregistrés dans l'AndroidManifest.xml, soit être créés depuis le code Java.

Si l'application ne tourne pas lorsqu'un de ses IntentReceiver est déclenché, le système la démarrera. De plus, une application peut diffuser ses propres Intents aux autres applications du système.

3. Service

Un service est un programme qui tourne en arrière-plan (sans interface graphique). Un bon exemple d'utilisation est un antivirus : l'antivirus tourne en arrière-plan, et l'utilisateur effectue d'autres opérations.

4. Content provider

Le « fournisseur de contenu » laisse aux autres applications la possibilité d'accéder aux données de l'application courante (via une base de données SQLite, ou aussi le Shared Preferences). Des méthodes peuvent être implémentées dans l'application pour permettre aux autres programmes de lire et de manipuler les données de l'application. Le Content Provider est utile pour partager des données avec d'autres applications. Le système d'exploitation dispose de son propre Content Provider qui propose aux applications des informations telles que la liste des contacts, l'historique des communications téléphoniques, etc.

Annexe B: Installation et utilisation d'Apache Felix sur Android en utilisant le shell

Apache Felix est implémentation d'OSGI compatible Android depuis la version 1.0.3. De plus, plusieurs sources nous ont permis réaliser quelques exemples de bundles réalisant un affichage via l'API graphique proposé par Android. Il est possible de connecter un shell Android à notre émulateur grâce à la commande :

adbshell

Une fois connecté il suffit de se créer un répertoire de travail (/data/felix par exemple) dans lequel nous copieront les fichiers suivants:

./bin/felix.jar	archive jar du serveur Apache Felix
./conf/config.properties	configuration standard de Felix lançant

Les bundles ci-dessous au démarrage du serveur (mettre en annexe).

./bundle/org.apache.felix.shell-1.0.2.jar	Shell felix
./bundle/org.apache.felix.shell.tui-1.0.2.jar	Console pour le shell de felix

Pour copier les fichiers, on utilise la commande *adb push*: exemple

adb push c:\[chemin en local]\felix.jar /data/felix/bin/felix.jar

Tous les jars utilisés sous android doivent être adaptés pour pouvoir tournés sur la DalvikVM. Pour cela google fournit un utilitaire dx permettant de convertir les class au format dalvik :

`dx --dex --output=[chemin complet]/classes.dex [chemincomplet]JAR_file.jar`

`cd [chemin complet]`

`aaptadd JAR_file.jar classes.dex`

Il est important de se mettre dans le dossier de travail pour utiliser la commande `aapt add`, sinon `classes.dex` ne sera pas ajouté correctement à l'archive (l'arborescence complète sera insérée dans l'archive) et elle sera inutilisable pour la DalvikVM. Ensuite nous pouvons lancer notre serveur apache felix:

adbshell → pour lancer un shell

cd /data/felix

La commande `java` n'existe pas sous android, on utilise la commande suivante :

*/system/bin/DalvikVM -Xbootclasspath:/system/framework/core.jar *

-classpath bin/felix.jar org.apache.felix.main.Main

Nous avons alors un serveur felix OSGI et la possibilité de charger d'autres bundles.