Rapport final du TER HLIN601
du Licence Informatique 3 année,
effectué de Janvier à Mai 2015,
encadré par Abdelhak-Djamel SERIAI

Surveiller sa maison via son Smartphone sous Android

RAPPORT FINAL (MAI 2015)

Travail réalisé par l'équipe

AbdoulayeDIALLO
Quentin PHILIPPOT (Chef du projet)
Jihen FOURATI
Redoine EL-OUASTI

https://sourceforge.net/projects/ter2015/

Remerciements

Nous remercions notre référent, M. Seriari, pour ses conseils avisés et pour l'efficacité avec laquelle il nous a guidés tout au long du développement du projet. Nous tenons à remercier aussi toutes personnes présentes lors de notre soutenance et plus particulièrement les membres du Jury.

Table des matières

1 Étude bibliographique

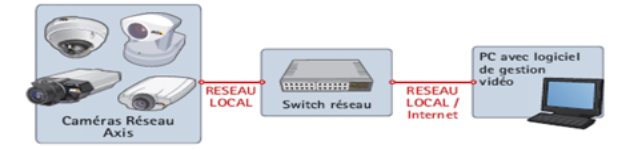
1.1 Télésurveillance

La télésurveillance est un système technique structuré en réseau permettant de surveiller à distance des locaux appartenant à des particuliers ou à des professionnels , des machines ou des individus.
Avec les techniques de vidéosurveillance et le développement d'Internet, la télésurveillance s'est répandue.
Elle a ouvert la voix à de nombreuses applications pour les entreprises et les collectivités telles que :
** L'évaluation du trafic routier grâce à des caméras et des capteurs
** La surveillance d'appareils ou de machines : état d'usure, épuisement de consommables, anomalies de fonctionnement, actes de malveillance subits par l'appareil
** La surveillance de personnes à distance : patients dans un hôpital, personnes âgées dans une maison de retraite
** La surveillance de lieux sensibles (banques, centrales nucléaires, etc.) et d'habitations, afin de prévenir les intrusions, les cambriolages et les actes de vandalisme

1.2 LA VIDEOSURVEILLANCE

La vidéosurveillance consiste à placer des caméras de surveillance dans un lieu public ou privé pour visualiser en un endroit centralisé tous les flux de personnes au sein d'un lieu ouvert au public et prévenir vols, agressions et mouvements de foule.

1.3 Système de vidéosurveillance sur IP avec caméras réseau



Il s'agit d'un système entièrement numérique dans lequel les images vidéos sont transmises sur réseau IP à l'aide de caméras réseau.
Une caméra réseau associe une caméra numérique et un ordinateur, et permet la numérisation et la compression vidéo.
Les images vidéos sont acheminées par réseau IP via les commutateurs réseau, pour être restituées et enregistrées sur un PC/serveur standard à l'aide d'outils de gestion vidéo.
Ce nouveau système de vidéosurveillance sur IP « tout numérique » permet flexibilité et évolutivité.

1.4 Qu'est-ce qu'une caméra IP ?

Une caméra IP ou caméra réseau est une caméra de surveillance utilisant le Protocole Internet pour transmettre des images et des signaux de commande via une liaison Fast Ethernet.
Certaines caméras IP sont reliées à un enregistreur vidéo numérique (DVR) ou un enregistreur vidéo en réseau (NVR) pour former un système de surveillance vidéo.
L'avantage des caméras IP est qu'elles permettent aux propriétaires et aux entreprises de consulter leurs caméras depuis n'importe quelle connexion internet via un ordinateur portable ou un téléphone 3G.

1.5 ANDROID c'est quoi ?

Android est un système d'exploitation Open Source pour smartphones, PDA et terminaux mobiles conçu par Android, une startup rachetée par Google et

annoncé officiellement le 15 novembre 2007.
Afin de promouvoir ce système d'exploitation ouvert, Google a su fédérer autour de lui une trentaine de partenaires réunis au sein de l'Open Handset Alliance. Ils ont comme principaux concurrents Apple avec iPhone OS , Microsoft et son Windows Mobile , et bien sûr OpenMoko, le projet dont les spécifications logicielles et matérielles sont ouvertes.
Android doit son nom à la startup éponyme spécialisée dans le développement d'applications mobiles.
L'opportunité d'intégrer un système d'exploitation puissant, gratuit et pouvant s'enrichir d'applications tierces à son matériel électronique a ouvert la route à plusieurs projets.
Android équipe maintenant des cadres photos, des tablettes tactiles, des netbooks, des voitures, des smartbooks et prochainement des Walkman Sony.

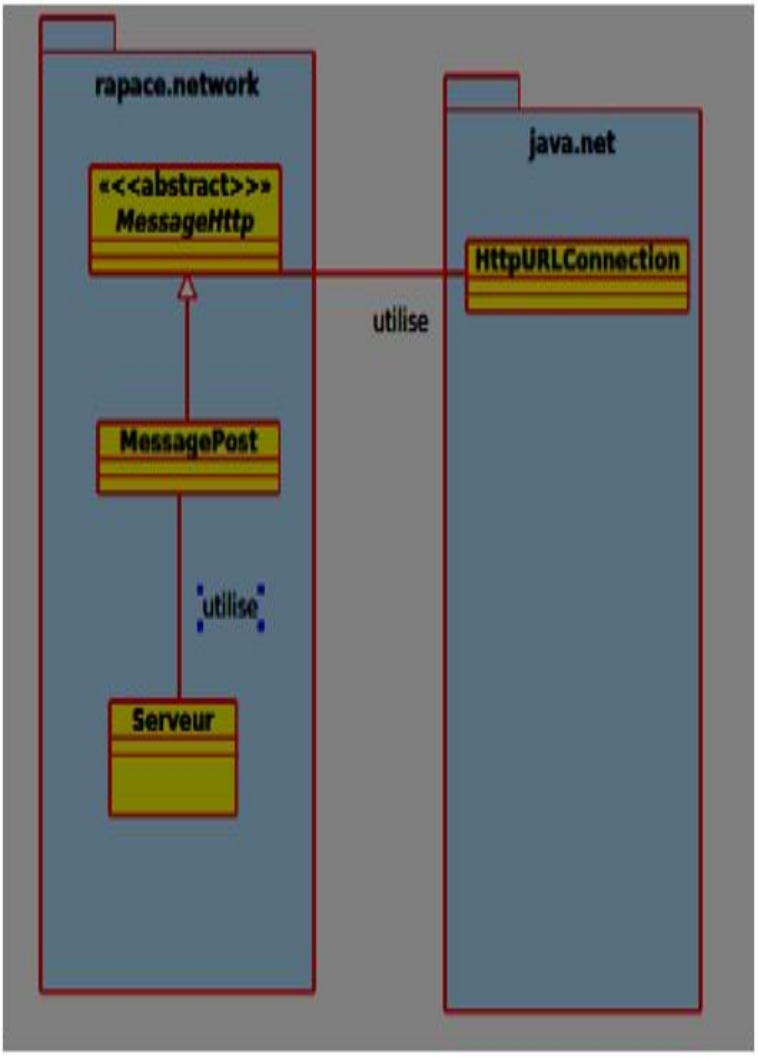
2 Présentation du problème

Il s'agit de développer une application en langage natif (java pour android), pour pouvoir surveiller à distance son domicile via son mobile.
Pour satisfaire cet objectif on doit passer par le développement d'une architecture client/serveur pour le streaming, sur un mobile, de vidéos ou images capturées à partir d'une ou de plusieurs caméras IP.
Cette application doit permettre :
- l'enregistrement d'un utilisateur et assurer son authentification.
- à l'utilisateur de regarder ce qui se passe en direct (streaming) au site surveillé à n'importe quel moment qu'il souhaite.
- Enregistrer la vidéo
Options supplémentaires :
l'application permet à l'utilisateur de recevoir une alerte quand il y a un intrus et offre aussi de choisir entre appeler la police et supprimer l'alerte dans le cas d'une fausse alerte,
L'application doit être constituée de trois activités :
1- Activité d'authentification :
Après avoir enregistré l'utilisateur dans la base de données, On lui donne un identifiant et un mot de passe qui lui permettront de s'authentifier sur la première activité,
l'application affichera des messages d'erreur dans les cas suivants :
-Le login ou le mot de passe sont très courts (moins de cinq caractères).
-Le mot de passe ou Le login ne sont pas les mêmes que ceux qu'on a enregistré dans la base de donnée.
SI tout se passe bien, on passe à l'activité suivante qui est :
2- Activité des sites surveillés :
Sur cette activité l'utilisateur pourra choisir parmi les sites surveillés par Rapace celui qu'il veut regarder,
l'interface de ce cet activité est dynamique c-à-d elle change selon le nombre et les noms des sites surveillés de chaque utilisateur.
La couleur des boutons des sites est par défaut en vert, mais elle change en rouge quand il y a une alerte pour un site.
Lorsque l'utilisateur choisit un site l'application lui dirige vers une troisième activité :
3- Activité d'affichage :
Sur cette activité l'application se connecte à la camera IP et diffuse sur l'écran du mobile ce qui se passe au site, En cas d'alerte il y a deux boutons qui s'affichent en dessous de la diffusion, qui donne la possibilité d'appeler la police ou supprimer l'alerte.

3 Solutions proposées

Nous avons à présent un aperçu du problème, il s'agit désormais de présenter les solutions développées au cours du projet.
1. Nativité d'une application :
Que l'on développe sous Android, iOS ou Windows, il existe trois types d'applications ayant chacune leurs avantages et leurs défauts.
• Les applications natives : Elles sont intégralement développées en langage natif (Java et XML pour Android). Ces langages ont l'avantage d'offrir de meilleures performances, il est compilé (en partie) et le système est optimisé pour les supporter. De plus, ils donnent la possibilité d'interagir avec le système d'exploitation.
• Les applications web : Elles récupèrent le contenu d'une page web. Sous Android cela se fait via la balise ouvrante XML <WebView> prenant en attribut l'url de la page encodée avec technologies web usuelles. L'avantage de ces applications est leur portabilité : il ne s'agit plus de compatibilité entre système d'exploitation mais entre navigateur web. Là où trois application en trois langages natifs différant sont requis pour les application natives, une page web suffit chez les applications web. Les performances sont moins importante, mais il faut noter que la majorité des applications ne requièrent pas de hautes performances. Le vrai désavantage réside dans la sécurité déployée par les systèmes d'exploitation :
Android, par exemple, interdit aux scripts d'interagir avec les composants du système (caméra, accès sms, répertoire, etc).
• Les application hybrides : De manière assez évidente, elles acceptent du code natif, et du code provenant d'une page web. Il s'agit d'un compromis entre application native et application web, permettant de tirer judicieusement parti de chacune.
À l'écriture du rapport, nous proposons une solution native. En effet, nous avons saisi l'opportunité de découvrir une nouvelle manière de programmer, la programmation mobile. De plus un impératif technique nous contraint à interagir avec l'os pour implémenter notre mécanisme d'alerte. Dans tous les cas, une application au moins hybride est requise.
2. Communication clients - serveurs :
Pour gérer l'authentification et déterminer les lieux qu'un utilisateur place sous surveillance, il convient d'utiliser un système de gestion de base de données, lequel conservera nos informations. Dans notre cas, il s'agira d'un serveur MySQL couplé à l'outil PhpMyAdmin ; ce sont des outils simples, et dont les performances suffisent largement à l'usage souhaité.
Nous utilisons les relations sont les suivantes :

Utilisateur (id, nom, prenom, psswd, email)
Site(id, nom, adresse, descriptif, url, en_alerte)
Surveille (id_utilisateur, id_site)
- Utilisateur.psswd est crypté avec la fonction PHP password_hash() conformément aux recommandations php.net.
- Sitr.url est l'url du flux vidéo, l'adresse de notre caméra ou l'ip de la machine à contacter dans notre cas.
- Site.en_alerte est un booléen vrai si le site est en état d'alerte, faux sinon.
L'accès à la base de données se fait par l'intermédiaire d'un serveur Apache, via des scripts PHP et l'API PDO. Afin de communiquer avec le serveur MySQL, le client Rapace doit alors envoyer un formulaire au serveur Apache, lequel traite les données et échange éventuellement avec le serveur MySQL.
Notre gestion des communications peut se décliner en deux points : d'une part côté client, l'application gère l'envoi de formulaire au serveur Apache et attend une réponse ; côté serveur, d'autre part, les scripts PHP du serveurs Apache manipulent le formulaire, contactent le serveur MySQL puis font suivre sa réponse au client Rapace.
2 - 1. Côté client Rapace :
Le package rapace.network régit l'ensemble des communications avec le serveur Apache. Sa structure est la suivante :



La classe abstraite MessageHttp implémente les fonctionnalités indispensables à l'envoi de message HTTP, indépendamment la méthode utilisée. Elle repose principalement sur la classe java.net.HttpURLConnection, recommandée par developer.android depuis la version 2.3 .
D'autres classes permettant l'envoi de requêtes HTTP existent, mais des problèmes de compatibilité apparaissent suivant la version du mobile.
MessagePost spécialise MessageHttp afin de prendre en charge la méthode POST.
Nous aurions pu créer une classe MessageGet spécialisant MessageHttp pour la méthode GET, mais nous n'avons pas eu besoin d'utiliser cette méthode dans notre projet.
En réalité HttpURLConnection, supporte à elle seule l'envoi de message HTTP, quelque soit la méthode. On peut alors se demander pourquoi avoir créé le

package rapace.network.
Le site developer.android nous donne un exemple d'utilisation de HttpURLConnection :

```
// Création de l'URL :
URL url = new URL("http ://www.android.com/") ;
// Ouverture de connection
HttpURLConnection urlConnection = (HttpURLConnection)
url.openConnection() ;
try {
// Configuration de la connection :
urlConnection.setDoOutput(true) ;
urlConnection.setChunkedStreamingMode(0) ;
// Création du flux d'émission :
OutputStream out = new
BufferedOutputStream(urlConnection.getOutputStream()) ;
// ecriture dans ce flux :
writeStream(out) ;
// Creation du flux de réception :
InputStream in = new BufferedInputStream(urlConnection.getInputStream()) ;
// Lecture du flux :
readStream(in) ;
finally {
// fermeture de connection :
urlConnection.disconnect() ;
}
}
```

Le code suivant pose plusieurs problèmes :
1 - Ce code génère un grand nombre d'exceptions :
on en compte pas moins de huit dans son adaptation à notre client Rapace.
Or l'implémentation ci-dessus ne permet pas d'identifier clairement les exceptions et de les traiter. Plusieurs méthodes, par exemples, jettent potentiellement un JavaNullPointerException.
* Une solution viserait à découper le bloc try en autant de sous-blocs try qu'il y a de méthodes « à risque ».
Mais avec un nombre important de méthodes à risque, notre code devient rapidement illisible.
2 - Ce code gère à la fois l'ouverture de connexion, la configuration, la création de flux (émission/réception) ainsi que leur entrées/sorties, plus la fermeture de connexion.
Or, il est généralement préférable de n'attribuer qu'un rôle à une méthode.(cf livre coder proprement)

* Une solution consiste à découper notre code en autant sous routines, qu'il y a de commentaires dans notre exemple.
En conjuguant nos deux solutions, on obtient la méthode principale de MessagePost :

```
public void run() {
ouvrir_connection();
emettre_requete();
setReponse_serveur(recevoir_reponse());
fermer_connection();
}
```

Les sous-routines ci-dessus sont elles mêmes découpées en sous-routines, nous renvoyons le lecteur vers le code source en annexe afin de mieux comprendre le découpage de notre programme.
En plus d'améliorer le maintient de notre programme, la modularité permet aussi d'associer une sémantique à une sous-routine.
Puisque notre classe MessageHttp réutilise les propriétés de HttpURLConnection, on peut se demander pourquoi nous avons choisi d'associer les deux classes, alors qu'une relation de spécialisation semblerait mieux indiqué pour représenter les liens sémantiques entre nos classes.
Tout simplement parce que Java ne permet pas l'héritage multiple, et que MessageHttp spécialise déjà la classe Java.lang.Thread.
En effet, la communication avec le serveur Apache peut prendre un certain temps durant laquelle l'application se fige. L'IHM n'est alors plus opérationnelle, et l'utilisateur peut ressentir un désagrément.
Pire encore, le système Android peut décider de fermer notre application et afficher un message d'erreur si l'application n'obtient pas rapidement de réponse !
La classe Thread permet de définir ce que l'on appelle en français un processus léger. Un processus léger n'est pas un processus au sens propre.
Ce sont des objets manipulable par le programme, des éléments rattachés à un processus.
Processus et Thread permettent une exécution en parallèle, mais deux processus ont leurs propres segments mémoire (pile, statique, dynamique), alors que les processus légers utilisent les mêmes segments, celui du processus père.
Cela rend la communication entre processus légers particulièrement agréable.
De plus, chacun possède sa propre pile d'appel, ce qui permet dans notre cas de détacher l'IHM de communication avec le serveur Apache.
En java, la création d'un processus léger se fait comme n'importe quelle instanciation d'objet.
L'exécution du processus léger démarre avec la méthode Thread.start() qui effectue le traitement défini dans la redéfinition de Thread.run().

Pour attendre la fin de la fonction Thread.start(), on utilise Thread.join().
L'envoi d'un message HTTP suivant la méthode POST se fait par :
MessagePost message = new MessagePost(url, parametres) ;
message.start() ;
message.join() ;
Quant à la classe Serveur, elle implémente plusieurs fonctions comme authentifier(), ou lever_alerte(), dont le rôle est simplement de renseigner l'url du script à appeler et formater les paramètres à transmettre à la classe MessagePost. Lorsque message passe à l'état terminé, la méthode lit la réponse renvoyée par le serveur Apache et effectue un traitement en conséquence.
2 -2 . Côté serveur Apache :
Nous utilisons trois scripts : authentification.php, demander_site_par_utilisateur.php, lever_alerte.php dont le rôle est assez explicite.
Les scripts sont formés de la façon suivante :
1 - On teste la présence des paramètres attendus :
if (!empty(trim($_POST['parametre1']))
and !empty(trim($_POST['parametre2'])) . . .
and !empty(trim($_POST['parametre_n']))){ /* traitement */ }
2 - On ouvre une connexion avec PDO, utilisons une requête préparée, et renvoyons le résultat avec une boucle sous la forme :
$answer = $query->fetch(PDO : :FETC_ASSOC) ;
foreach ($answer as $value) {
print $value . " " ;
}
L'importance d'utiliser PDO et ses requêtes préparées ?
L'API PDO propose un ensemble de fonctionnalités optimisées et disponibles indépendamment du système de gestion de base de données utilisé.
Son usage (fortement recommandée par le site php.net) permet une plus grande portabilité du code, mais pas seulement.
La faille par injection SQL :
Une faille connue consiste à envoyer du code SQL en tant que paramètres pour manipuler notre base de données sans en avoir les droit. (Voir exemples en annexe.)
En plus d'optimiser nos requêtes, l'utilisation de requêtes préparées permet de se prémunir contre ce type d'attaque.
L'encryptage du mot de passe :
Malgré ces précautions, que se passerait-il si un individu parvenait à récupérer le contenu de notre base de donnée ? Il posséderait le mot de passe et chacun des utilisateurs et aurait accès à toutes les caméras enregistrées !
Pour éviter ce genre de problème, il convient de crypter la base de donnée, et le

site php.net recommande l'utilisation de password_hash() pour crypter le mot de passe, et de password_verify() pour vérifier si le mot crypté et non-crypté correspondent.
Ainsi notre script d'authentification procède en deux étapes :
Il charge les informations de la table Utilisateur pour un id donné, et vérifie le mot de passe. Si les deux concordent alors on renvoie les information à l'application cliente.

```
$query = $bdd->prepare("SELECT * FROM Utilisateur WHERE
email= :email");
$query->bindParam(' :email', $email, PDO : :PARAM_STR);
$query->execute();
$answer = $query->fetch(PDO : :FETCH_ASSOC);
if ( password_verify($password, $answer['psswd']) ) {
foreach ($answer as $value) {
print $value . " ";
}
}
```

3 – Interface Homme Machine :
Sous Android les IHM sont étroitement liée à la notion d'Activité.
Les Activités sont des classes spéciales auxquelles est associée un fichier au format XML lequel définit une interface statique, comme on le ferait avec du HTML pour une page web, par exemple.
Ceci étant, sous Android on peut également définir l'IHM en utilisant du code Java.
Si d'emblée cette pratique semble moins agréable qu'une définition via XML, elle présente cependant la possibilité de changer certaines propriétés de notre interface à la volée, c'est ce que l'on appelle une interface dynamique.
Dans le cadre de notre application, un utilisateur peut surveiller plusieurs site. L'Activité Affichage_Liste_Site, comme son nom l'indique liste l'ensemble des sites placés sous surveillances. Ce nombre étant variable, il devient donc indispensable d'utiliser une interface dynamique.
Pour ce faire, nous écrivons toute la partie IHM statique dans le XML et à chaque fois que l'activité revient au premier plan (callback : onResume()), on reconstruit l'ensemble de la partie dynamique (c'est cette dynamicité qui permet de colorer un site en vert lorsque l'état d'alerte est faux, et en rouge dans le cas contraire).
Pour écrire une interface dynamique, notre solution est de laisser un LineaLayout identifiée que l'on remplira ensuite, exemple :
Dans le XML :

```
<LinearLayout
android :id="@+id/content2 "
```

... ></LinearLayout>
Dans le code Java :
// Récuperation Layout XML :
LinearLayout content = (LinearLayout) findViewById(R.id.content2) ;
// On le vide de son contenu :
content.removeAllViewsInLayout() ;
// Création d'une vue :
TextView no_site = new TextView(this) ;
no_site.setText("Vous ne surveillez actuellement aucun site.") ;
LinearLayout.LayoutParams layoutParam = new
LinearLayout.LayoutParams(LinearLayout.LayoutParams.WRAP_CONTENT,
LinearLayout.LayoutParams.WRAP_CONTENT) ;
// Insertion d'une vue dans notre layout :
content.addView(no_site, layoutParam) ;
(voir code source en annexe pour de meilleur exemples)
4 – Mécanisme d'alerte :
Notre application est capable d'alerter l'utilisateur en se lançant automatiquement au premier plan sous ordre du serveur.
Concrètement, le serveur enverrait un sms à l'utilisateur (lorsque nous écrivons ce rapport, le mécanisme n'est pas implémenté côté serveur), lors de la réception l'application se lancerait automatiquement en envoyant un message d'erreur.
Cela est avant tout possible car Android permet au développeur d'interagir facilement avec les composants du téléphone.
La classe SMSReceiver permet de détecter quand un message nouveau message est arrivé, vérifie si l'émetteur est notre serveur, et se lance le cas échéant.

4 Conclusion

Nos travaux ouvrent de nombreuses perspectives aussi bien au niveau proposition d'un rapprochement entre la vidéosurveillance et les techniques d'enregistrement de données biométriques. Cela permettrait, par exemple, d'analyser la démarche des personnes filmées ou leur attitude et ainsi de détecter des anomalies telles qu'une personne lourdement chargée ou un comportement suspect.

5 Bibliographie

*)*Livres Programmer en Java de Claude Delannoy*
Cours Développement d'applications Mobiles sous Android , Abdelhak Djamel Seriai , Université Montpellier
*)*Sites web https ://www.java.com*
https ://www.openclassrooms.com
http ://developer.android.com/
http ://fr.wikipedia.org/wiki/Vidésurveillance