



## REVUE DE PROJET FINALE

### MODULE DE DIFFUSION D'INFORMATIONS

PETRELLA Olivier

## WISMAS version 1.0

*Weather Information System Multi Activity Station*

## TABLE DES MATIÈRES

<b>ETUDE ET ANALYSE</b>	<b>4</b>
Présentation du projet	4
Expression du besoin	4
Ressources	5
Matérielles	5
Logicielles	5
Schéma du système	6
Cahier des charges	7
Module de diffusion d'informations	7
Tâches	7
Planification	8
Diagramme de Gantt - Analyse	8
Diagramme de Gantt - Conception	8
Diagramme de cas d'utilisation	10

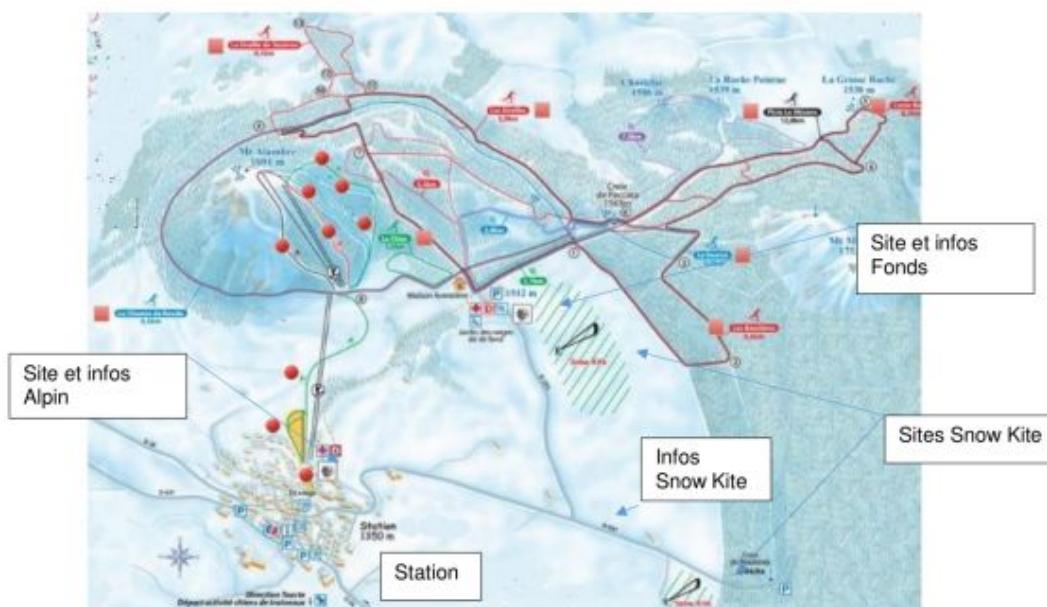
<b>CONCEPTION</b>	<b>11</b>
IHM	11
Écran	11
Panneau lumineux	12
Structure des données	13
Les données météo	13
Les données de fartage	14
Base de données	15
Architecture logicielle	17
Diagramme de classes	17
Communication	23
Stations météos	23
Schéma de communication	23
Caractéristiques XBee	24
Protocole WSP	26
Panneau lumineux	32
Schéma de communication	32
Caractéristiques panneau lumineux	32
Protocole DBC	33
Cas d'utilisation - Visualiser les informations de la station	35
Cas d'utilisation - Relever les mesures des sites	36
Cas d'utilisation - Enregistrer les données	37
Diffusion vidéo	38
Déploiement	39
Diagramme de déploiement	39
Fichier de configuration	40
<b>TESTS DE VALIDATION</b>	<b>45</b>

## ETUDE ET ANALYSE

### Présentation du projet

WISMAS (*Weather Information System Multi Activity Station*) est un système d'information météo pour une station multi activités.

La station de ski sur laquelle s'appuie l'étude est une station de moyenne montagne multi-activités dans laquelle on peut faire du ski de fond, du ski de piste et également du snowkite sur 3 sites voisins. Le système est doté de différents capteurs météo, afin de pouvoir les retransmettre aux clients de la station de ski. Les supports d'affichages sont un panneau lumineux et un écran de télévision. Les différents sites seront aussi équipés de caméras pour donner un aperçu quasi temps réel aux clients.



Plan de situation du système

### Expression du besoin

Il s'agit de développer :

- sur un système embarqué (carte ATMEL SAM4S-EK par exemple) les applications nécessaires à la mise en œuvre des différents capteurs et à la transmission sans fil des données (**Module de météorologie**)

- sur un PC d'acquisition, un logiciel de gestion des caméras implantées sur les différents sites (**Module d'acquisition « vidéo »**)

- sur un poste de diffusion, un logiciel de publication d'informations sur la station (**Module de diffusion d'informations**) : gestion du panneaux lumineux et d'un écran

## Ressources

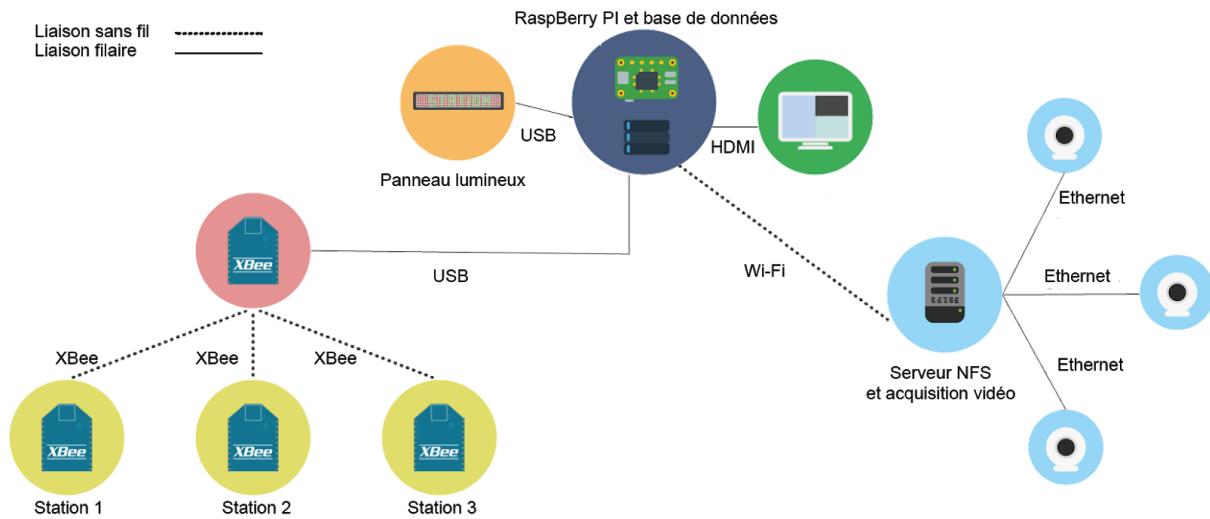
### Matérielles

- PC « Acquisition » PC HP sous GNU/Linux Ubuntu 12.04
- Nano-ordinateur Raspberry Pi modèle 3B à Broadcom BCM2837 64 bit à quatre cœurs ARM Cortex-A53 à 1,2 GHz équipé de 1GO de RAM et d'une carte SD 16GO (Raspbian OS)
- ECRAN Écran HDMI
- PANNEAU Panneau lumineux Mc Crypte 590996

### Logicielles

- Système d'exploitation du PC « Acquisition » : GNU/Linux Ubuntu LTS 12.04
- Système d'exploitation de la Raspberry Pi : Raspbian
- Environnement de développement (IR) : Qt Creator et Qt Designer
- API GUI PC « Acquisition » : Qt 4.8 (minimum)
- API GUI Raspberry Pi : Qt 4.8
- Compilateurs : GNU g++ pour Linux
- Système de gestion de bases de données relationnelles : MySQL
- Gestion et administration de bases de données : phpMyAdmin
- Atelier de génie logiciel (IR) : bouml 7.4
- Logiciel de gestion de versions (IR) : subversion
- Générateurs de documentation (IR) : Doxygen version 1.8.11
- Gestionnaire de projet (IR) : Planner

## Schéma du système



Les éléments du schéma reliés par des pointillés définissent une liaison sans fil. Les Stations, transmettent des trames de données (température, vitesse vent, direction vent, humidité...), vers le Raspberry Pi, qui lui va assurer le calcul et le traitement des données. Il possède une base de données pour stocker périodiquement les informations météo. Nous avons aussi un serveur NFS qui va stocker les fichiers vidéo de chaque sites, pour les afficher sur l'écran.

## Cahier des charges

Le système devra :

- Faire des mesures météorologiques sur plusieurs sites.
- Prendre des séquences vidéos à partir de plusieurs caméras IP.
- Afficher l'ensemble de ces renseignements sur les sites d'achat des forfaits.
- Enregistrer les données météo périodiquement en base de données.

## Module de diffusion d'informations

Le poste de diffusion, devra gérer un panneau lumineux et un écran afin de publier des informations sur la station via un Raspberry Pi :

- Conditions météorologiques suivant le site (Alpin, Fond ou Snow kite) avec la possibilité de visualiser une vidéo du lieu.
- Informations sur la station (accès, horaires, tarifs, piste ouverte).
- Conseils de partage en fonction des conditions météorologiques.

Pour cela, l'application logicielle « WISMAS\_Station » doit permettre :

- De relever les mesures des stations météorologiques installées sur les différents sites.
- D'enregistrer les mesures dans une base données.
- De diffuser les conditions météorologique d'un site (sur un panneau lumineux).
- De diffuser des informations (météo, vidéos, messages) sur un écran.

## Tâches

### Développement :

L'application logicielle « WISMAS\_Station »

### Installation :

L'environnement de développement, le panneau lumineux, l'écran

### Mise en oeuvre :

L'environnement de développement, la base de données

### Configuration :

Le panneau lumineux, l'écran, la liaison sans fil

### Réalisation :

Les diagrammes UML, L'IHM du module, Le code source de l'application

### Documentation :

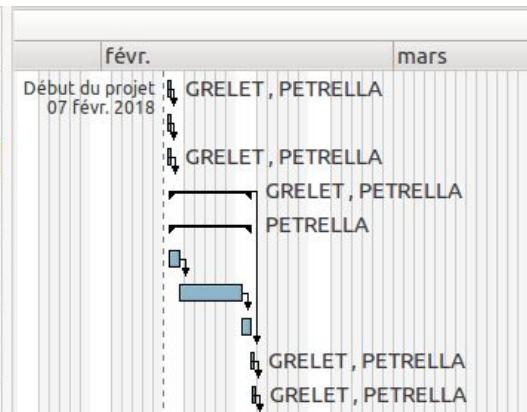
Le dossier technique et les documents relatifs au module

## Planification

### Diagramme de Gantt - Analyse

Le **diagramme de Gantt** est un outil utilisé (souvent en complément d'un réseau PERT) en ordonnancement et en gestion de projet et permettant de visualiser dans le temps les diverses tâches composant un projet.

TPÉ	Nom
1	Rédiger document tests et validation
2	Définir une application itérative
3	Diagramme de Gantt
4	▼ Diagramme UML
4.1	▼ Module de diffusion d'informations
4.1.1	Diagramme de cas d'utilisation
4.1.2	Diagramme de séquence
4.1.3	Diagramme de classe
5	Définir une planification itérative
6	Prototyper l'IHM

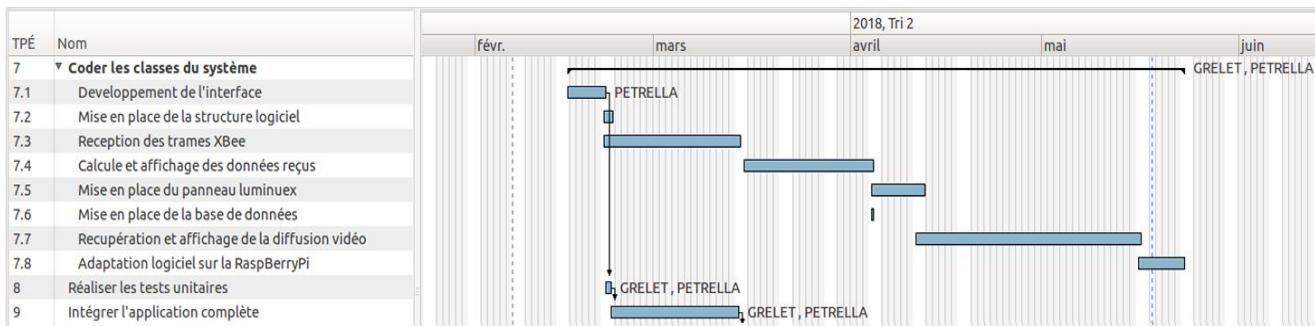


*Extrait du diagramme de Gantt de la partie analyse du projet WISMAS*

#### Activités :

- Etude du cahier des charges
- Création de tous les diagrammes UML
- Création du croquis de l'interface

## Diagramme de Gantt - Conception



*Extrait du diagramme de Gantt de la partie conception du projet WISMAS*

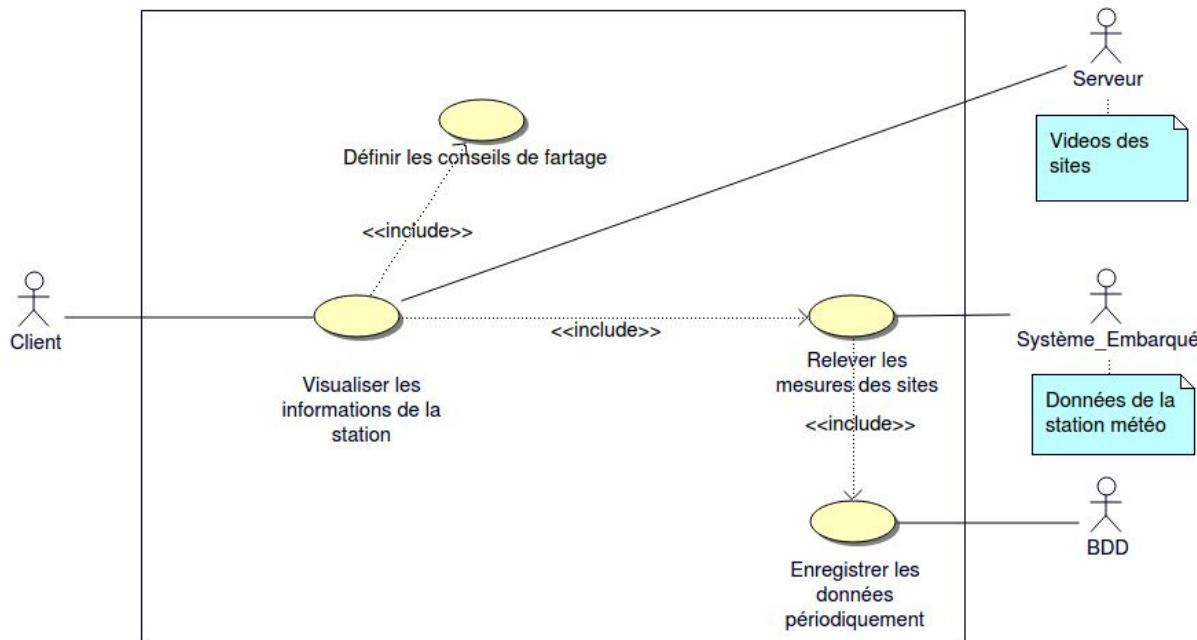
### Planification itérative :

Dans un premier temps, j'ai développé l'interface afin de pourvoir avoir un support pour afficher les valeurs. Par la suite la mise en place de la structure du logiciel, permet de mettre en place les classes de bases du système. Puis le développement des classes satellites qui vont récupérer les données et les traiter. Pour finir, une partie adaptation pour la Raspberry Pi et la correction des erreurs avec les tests unitaires.

## Diagramme de cas d'utilisation

Un **cas d'utilisation** (use case) représente une fonction offerte par le système et qui produit un résultat observable intéressant pour un acteur.

Un **acteur** (actor) représente un rôle joué par une entité externe qui interagit avec le système.



Le **client** peut visualiser des différentes informations affichées sur deux types de support : un écran de diffusion et un panneau lumineux.

Le cas d'utilisation “**Visualiser les informations de la station**” affiche :

- sur l’écran de diffusion (type écran télévision) :
  - l’ensemble des données météos des différents sites,
  - les vidéos des différents sites,
  - et optionnellement les **conseils de fartage**
- sur le panneau lumineux :
  - une synthèse des données météos des différents sites,
  - des informations complémentaires (date, heure, horaires d’ouverture, tarifs, ...)

Ce cas d’utilisations a besoin de **relever les mesures** des différentes stations météo (**système embarqué**) implantées sur les sites (Alpin, Fond, ...). Ces données sont enregistrées dans une base de données (**BDD**) pour utilisation externe (site web, statistiques, ...).

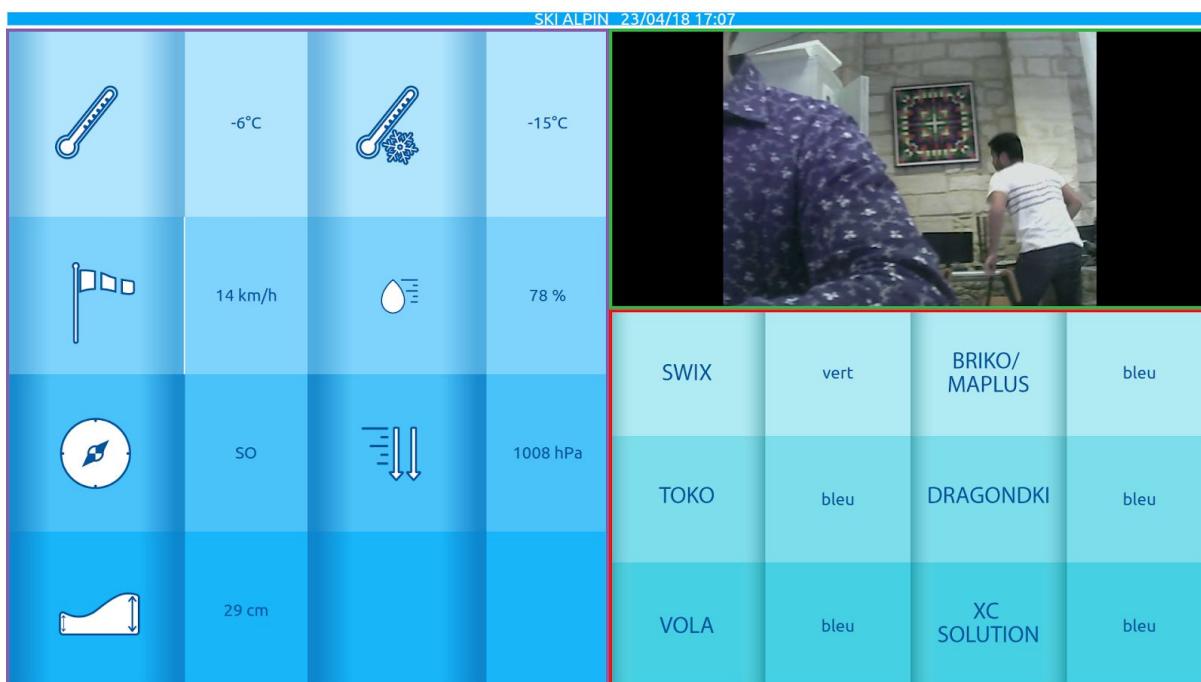
Les vidéos sont récupérées sur un **serveur** distant.

## I. CONCEPTION

### IHM

L'IHM (Interface Homme-Machine) définit les moyens et outils mis en œuvre afin qu'un humain puisse contrôler et communiquer avec une machine.

### Écran



Voici, une capture d'écran de l'interface WISMAS. Pour distinguer les parties de l'interface, des cadres de couleurs ont été rajouté.

**Blanc** : Cette partie présente la station courante dans notre cas nous sommes sur la station SKI ALPIN. Après avoir défini une période dans le fichier de configuration de l'application. Toute les périodes la station changera. Par exemple la station passera à SKI DE FOND, et cela changera toute les informations de l'interface en fonction de la station courante. Elle comporte aussi l'heure et la date.

**Violet** : Cette partie comporte les éléments météo reçu par la station courante. Température de l'air, température de la neige, vitesse et direction du vent, taux d'humidité de l'air, pression atmosphérique et hauteur de neige.

**Vert** : Cette partie affiche la dernière vidéo enregistrée par le site courant. Cette vidéo tourne en boucle jusqu'au changement du site.

**Rouge** : Cette partie est destinée aux professionnels des sport de glisse. Elle affiche les types de fartage qu'il faut utiliser en fonction des données météo.

## Panneau lumineux

Le panneau lumineux va afficher l'heure le nom du site, les données météo et les informations complémentaires (tarifs, horaire, ...).

Le panneau lumineux utilisé ici est un Mc Crypte de taille de 80x7. En utilisant une police de taille 6x7, il sera possible d'afficher 13 caractères. Pour afficher l'ensemble des informations, il faudra donc utiliser un mode déroulant structuré en pages.

Les pages seront :

- page A : Nom de la station
- page B : Heure
- page C : Date
- page D : Température
- page E : Vitesse vent
- page F : Hauteur de neige



*Panneau lumineux affichant la page A*

## Structure des données

### Les données météo

La station météo fournit les données suivantes :

#### Identifiant de la station

---

*L'identifiant station est un identifiant unique donné à chaque station météo lors de sa configuration. Il va permettre de les distinguer. Valeur entière.*

#### Type de site

---

*Le type de site, va permettre de d'identifier sur quel site est située la station. Il en existe trois, le snowkite, ski de fond et ski alpin. Valeur entière.*

#### Girouette

---

*La girouette est un capteur, qui va nous permettre d'indiquer l'orientation du vent. Qui a pour valeur un point cardinal exemple : "SE". Chaîne de caractères.*

#### Anémomètre

---

*L'anémomètre est un capteur servant à mesurer la vitesse du vent. Qui a pour unité le kilomètre par heure (km/h). Valeur entière.*

#### Sonde de température de l'air

---

*La sonde de température de l'air est un capteur permettant de capter le réchauffement et le refroidissement de l'air. Qui a pour unité le degré celsius (C°). Valeur réelle.*

#### Baromètre

---

*Le baromètre est un capteur de pression atmosphérique. Qui a pour unité le Pascal (hPa). Valeur entière.*

#### Hygromètre

---

*Le baromètre est une capteur de d'humidité de l'air. Qui a pour unité le pourcentage, c'est aussi une valeur entière comprise entre (0 et 100). valeur entière.*

## Sonde de température de la neige

*La sonde de température de la neige est un capteur permettant de capter le réchauffement et le refroidissement de la neige. Qui a pour unité le degré celsius (C°). Valeur réelle.*

## Nivomètre

*Le nivomètre est un capteur qui permet de calculer la hauteur de la neige. Qui a pour unité le centimètre (cm). Valeur entière.*

## Les données de fartage

Le fartage est un corps gras, ayant l'aspect de la cire, dont on enduit les semelles des skis, pour les empêcher d'adhérer à la neige et faciliter le glissement. Les conseils de fartages sont des indicateurs qui permettent de choisir un produit de fartage adapté aux conditions climatiques. Les conseils de fartage sont destinés aux professionnelles des sports d'hiver.

### Tableau des conseils de fartage :

NEIGE		Chaud humide	Standard	Froide	Très froide	Très froide et sec
SWIX	CH/LF/HF	0°/+10° jaune : 10	+4°/-4° rouge : 8	-2°/-8° violet : 7	-6°/-12° bleu : 6	-10°/32° vert : 4
TOKO	CH/LF/HF	0°/+6° jaune	-4°/-12° rouge		10°/-30° bleu	
VOLA	CH/LF/HF	+14°/-2° jaune	+0°/-5° rouge	-2°/-10° violet	-7°/-15° bleu	-10°/-25° vert
BRIKO / MAPLUS	BP/LP/HP	0°/-4° jaune ou orange	-4°/-8° rouge	-6°/-12° violet	-10°/-18° bleu	-10°/-30° vert : 4
DRAGON SKI	CH/LF/HF	0°/+10° jaune	+2°/-8° rouge	-5°/-15° bleu		
XC SOLUTION	HF	0°/+10° jaune	-3°/-7° rouge	-5°/-15° bleu		

### Légende des acronymes :

*BP ou SF ou CH = fart sans fluor.*

*LP ou LF = fart légèrement fluoré.*

*HP ou HF = fart hautement fluoré*

Le tableau ci-dessus sera réadapter pour être affiché sur l'application WISMAS. Dans la colonne neige les valeurs sont les marques qui proposent des cires. La colonne blanche indique le taux de fluor dans la cire. Les autres couleurs sont tout simplement la couleur de la cire. **Jaune** si la neige est chaude et humide entre ( $+14^{\circ}$  et  $-4^{\circ}$ ). **Rouge** si il y'a une neige Standard entre ( $+4^{\circ}$  et  $-7^{\circ}$ ). **Violet** si la neige et froide entre ( $-2^{\circ}$  et  $-12^{\circ}$ ). **Bleu** pour une neige très froide entre ( $-6^{\circ}$  et  $-18^{\circ}$ ). Et **Vert** pour une neige très froide et sec entre ( $-10^{\circ}$  et  $-32^{\circ}$ ). Ces données seront déterminés grâce à la sonde de température de la neige.

## Base de données

Une base de données relationnelle est une base de données où l'information est organisée dans des tableaux à deux dimensions appelés des **tables**. Les lignes de ces relations sont appelées des **enregistrements**. Les noms des colonnes sont appelés des **champs** ou des attributs.

Nous allons utiliser **MySQL** pour notre base de données.

La base de données est nommée **WISMAS\_2018**. Le schéma relationnel ci-dessous permet de visualiser les tables et leurs relations :

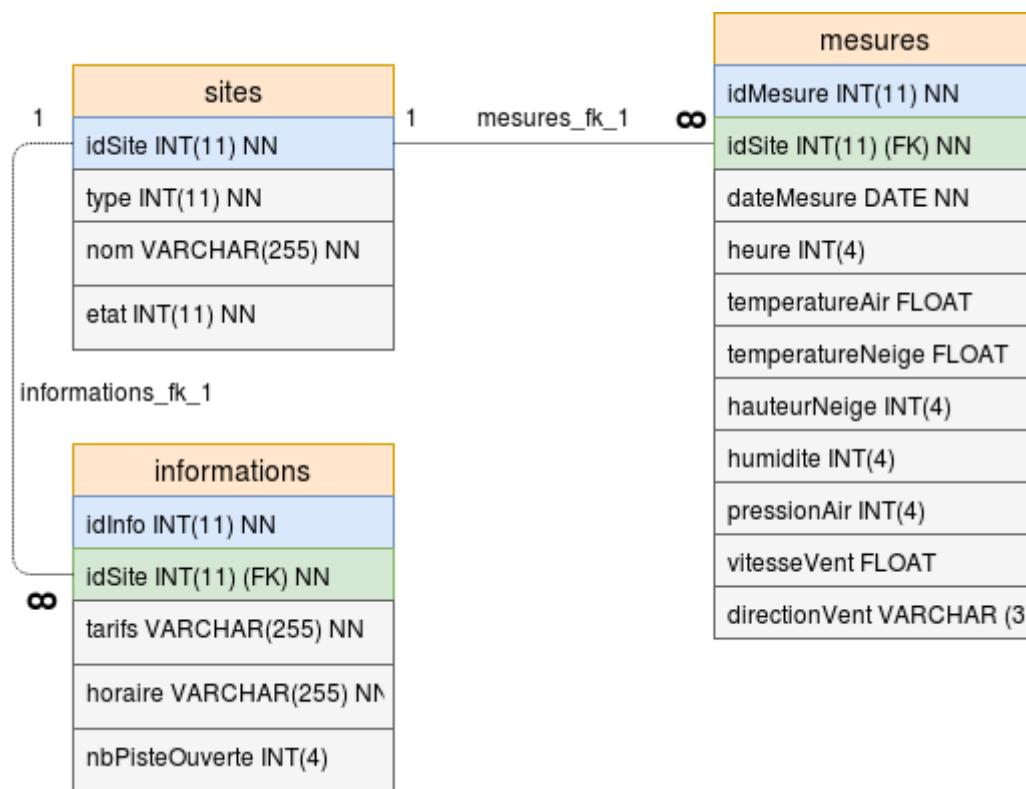


Table **sites** stocke les principales valeurs qui définissent un site.

Table **mesure** stocke les données météo. La relation est faire avec la table **sites** grâce à une **clé étrangère idSite**. Cela va faire l'association entre les données météo et un site.

Table **informations** stocke les informations complémentaires aux sites : tarifs, horaire et nombre de piste ouvertes. De la même façon, la relation est faite avec une **clef étrangère idSite**. Pour l'insertion des données de cette table, elle se fera depuis PHPMyAdmin. Des valeurs par défauts ont été fixées.

<b>idInfo</b>	<b>idSite</b>	<b>tarifs</b>	<b>horaire</b>	<b>nbPisteOuverte</b>
1	101	Adulte 14 euros / journée, Jeune/Etudiant 11 euros...	De 9h à 17h	10
2	201	Adulte 13 euros / journée, Jeune/Etudiant 10 euros...	De 8h30 à 16h30	8
3	301	Adulte 10 euros / journée, Jeune/Etudiant 8 euros ...	De 10h à 17h30	2

Valeurs de la table *informations*

Programmation Qt :

Avec Qt, l'accès à une base de données est réalisé par le module **QtSql**. Il fournit un ensemble de classes dont **QSqlDatabase** et **QSqlQuery**. Il faut ajouter dans le fichier de projet **.pro** :

**QT += sql**

On utilisera les services de la classe fournie **BaseDeDonnees** :

- `executer(QString requete)` : pour des requêtes INSERT, UPDATE et DELETE
- `recuperer()` : pour la requête SELECT

Exemple d'une requête d'insertion en base de données avec Qt :

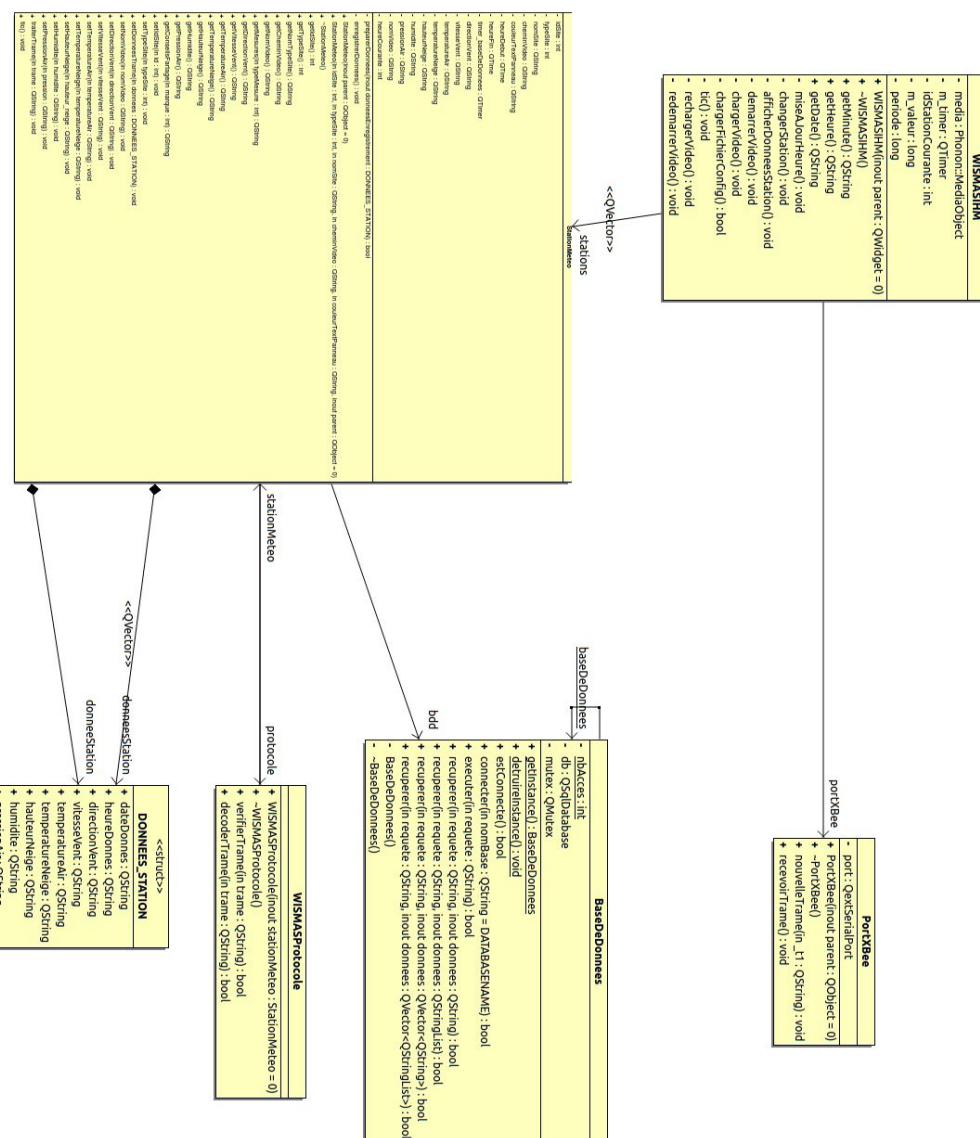
```
requete = "INSERT INTO mesures(idSite,dateMesure, heure,temperatureAir,
temperatureNeige,hauteurNeige,humidite,pressionAir,vitesseVent,directionVen
t) VALUES ('" + QString::number(this->getIdSite()) + "','" +
dateMesure.toString("yyyy-MM-dd") + "','" +
donneesEnregistrement.heureDonnes + "','" +
donneesEnregistrement.temperatureAir + "','" +
donneesEnregistrement.temperatureNeige + "','" +
donneesEnregistrement.hauteurNeige + "','" +
donneesEnregistrement.humidite +
"','" + donneesEnregistrement.pressionAir + "','" +
donneesEnregistrement.vitesseVent + "','" +
donneesEnregistrement.directionVent + "')";
```

La requête INSERT pour les mesures. Elle remplit les champs de la table mesures. Pour cela on lui fait passer une structure de données météo **donneesEnregistrement** de type **DONNEES\_STATION**

## Architecture logicielle

### Diagramme de classes

**Le diagramme de classes** fait partie de la partie statique d'UML car il fait abstraction des aspects temporels et dynamiques. Une classe décrit les responsabilités, le comportement et le type d'un ensemble d'objets. Les éléments de cet ensemble sont les instances de la classe (les objets).



La classe **WISMASIHM** est le coeur de notre programme c'est elle qui va charger le fichier de configuration afin de remplir les structures de données via une association avec la classe StationMeteo qui elle a une composition avec cette structure. Elle va afficher toutes les informations sur l'IHM (écran et panneau lumineux). Elle va aussi avoir une association avec la classe PortXBee pour récupérer les trames en provenance des stations.

WISMASIHM
<ul style="list-style-type: none"> <li>- media : Phonon::MediaObject</li> <li>- m_timer : QTimer</li> <li>- idStationCourante : int</li> <li>- m_valeur : long</li> <li>- pagePanneau_compteur : long</li> <li>- nb_page : int</li> <li>+ WISMASIHM(inout parent : QWidget = 0)</li> <li>+ ~WISMASIHM()</li> <li>+ getMinute() : QString</li> <li>+ getHeure() : QString</li> <li>+ getDate() : QString</li> <li>- afficherConditionsMeteoStation() : void</li> <li>- afficherDonneesMeteoStation() : void</li> <li>- afficherInformationsMeteoStation() : void</li> <li>- demarrerAffichagePanneau() : void</li> <li>- afficherConseilsFartage() : void</li> <li>- afficherHorodatage() : void</li> <li>- changerStation() : void</li> <li>- demarrerVideo() : void</li> <li>- recupererDerniereVideo() : void</li> <li>- chargerVideo() : void</li> <li>- chargerFichierConfig() : bool</li> <li>- rafraichir() : void</li> <li>- rechargerVideo() : void</li> <li>- redemarrerVideo() : void</li> </ul>

**WISMASIHM** : Classe principale, elle rythme notre application. Elle possède un Association avec StationMeteo et PortXBee. StationMeteo est placé dans un <<QVector>> pour y stocker toute les stations. Elle se charge de changer d'afficher les stations. De charger les vidéos des sites. Et d'initialiser le fichier de configuration.

<b>WISMASProtocole</b>
+ WISMASProtocole(inout stationMeteo : StationMeteo = 0)
+ ~WISMASProtocole()
+ verifierTrame(in trame : QString) : bool
+ decoderTrame(in trame : QString) : bool

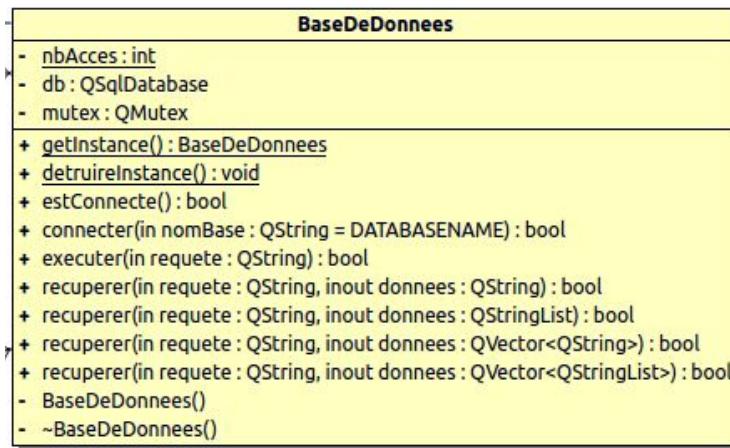
**WISMASProtocole** : Classe qui va se charger de traiter les trames reçues par la classe PortXBee afin de charger les données météo dans la classes StationMeteo grâce à une Association. La base de données est en Association avec la classe StationMeteo. La classe BaseDeDonnees est possède une relation sur elle même car c'est un Singleton. Cela restreint l'instanciation de celle-ci.

<b>PortXBee</b>
- port : QextSerialPort
+ PortXBee(inout parent : QObject = 0)
+ ~PortXBee()
+ nouvelleTrame(in _t1 : QString) : void
+ recevoirTrame() : void

**PortXBee** : Classe qui va gérer la connexion avec le XBEE et la réception des trames.

StationMeteo
<pre> - idSite : int - typeSite : int - nomSite : QString - cheminVideo : QString - couleurTextPanneau : QString - heureDebut : QTime - heureFin : QTime - timer_baseDeDonnees : QTimer - directionVent : QString - vitesseVent : QString - temperatureAir : QString - temperatureNeige : QString - hauteurNeige : QString - humidite : QString - pressionAir : QString - nomVideo : QString - heureCourante : int  - preparerDonnees(inout donneesEnregistrement : DONNEES_STATION) : bool - enregistrerDonnees() : void + StationMeteo(inout parent : QObject = 0) + StationMeteo(in idSite : int, in typeSite : int, in nomSite : QString, in cheminVideo : QString, in couleurTextPanneau : QString, inout parent : QObject = 0) + ~StationMeteo() + getIdSite() : int + getTypeSite() : int + getNomTypeSite() : QString + getCheminVideo() : QString + getNomVideo() : QString + getMesures(in typeMesure : int) : QString + getDirectionVent() : QString + getVitesseVent() : QString + getTemperatureAir() : QString + getTemperatureNeige() : QString + getHauteurNeige() : QString + getHumidite() : QString + getPressionAir() : QString + getConseilsPartage(in marque : int) : QString + setIdSite(in id : int) : void + setTypeSite(in typeSite : int) : void + setDonneesTrame(in donnees : DONNEES_STATION) : void + setNomVideo(in nomVideo : QString) : void + setDirectionVent(in directionVent : QString) : void + setVitesseVent(in vitesseVent : QString) : void + setTemperatureAir(in temperatureAir : QString) : void + setTemperatureNeige(in temperatureNeige : QString) : void + setHauteurNeige(in hauteur_neige : QString) : void + setHumidite(in humidite : QString) : void + setPressionAir(in pression : QString) : void + traiterTrame(in trame : QString) : void + tic() : void </pre>

**StationMeteo** : Classe qui gère le traitement des données météo. Elle va enregistrer les données météo mais aussi les renvoyer pour l'affichage via la classe WISMASIHM. Elle va gérer plusieurs associations. Composition avec la structure DONNEES\_STATION. Association avec WISMASProtocole et la BaseDeDonnees.



**BaseDeDonnees** : Classe fournie pour mettre la sélection, l'insertion, la modification et la suppression des données dans la base. Elle utilise des requêtes SQL.

Les structures de données



**DONNEES\_STATION** : Structure de données météo qui sera rempli pour enregistrer les données en base de données.

Autres structures :

```
struct CONFIG_LOGICIEL
{
    int nb_station; // défini le nombre de station
    long periode; // change l'affichage d'une station en s
};
```

```
struct PARAM_COMMUNICATION
{
    QString port; // fichier de périphérique XBee
    int baud_rate; // débit
    int data_bit; // nb de bits de données
    int stop; // nb de bits de stop
    int parity; // parité impaire, paire ou aucune
};
```

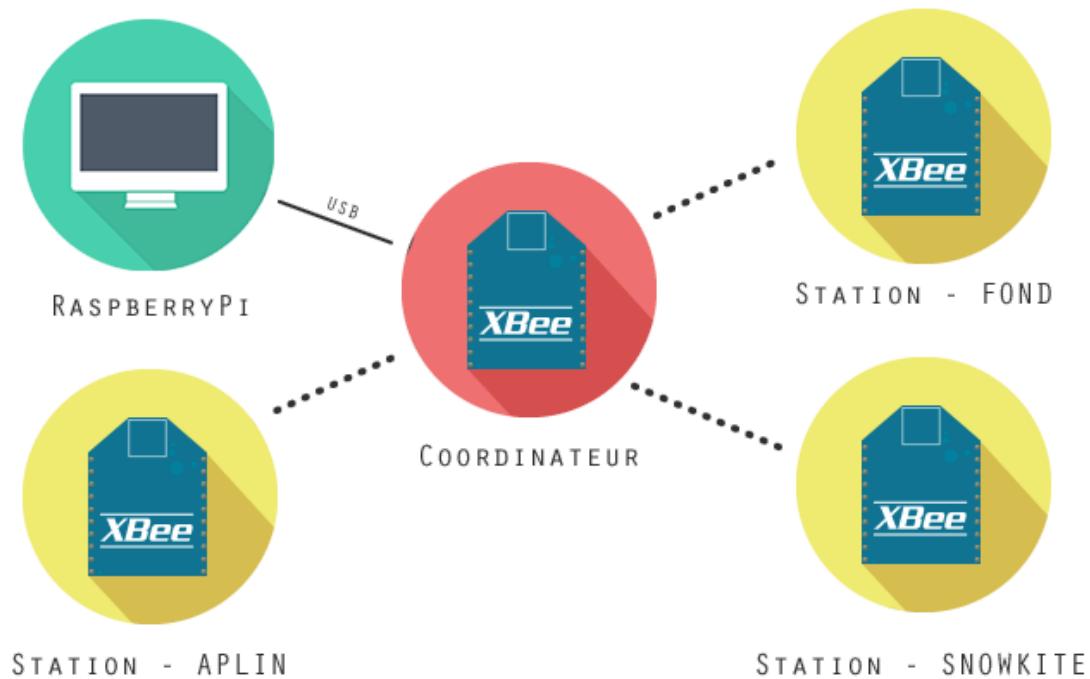
```
struct PARAM_PANNEAU
{
    QString port; // fichier de périphérique Panneau
    int baud_rate; // débit
    int data_bit; // nb de bits de données
    int stop; // nb de bits de stop
    int parity; // parité impaire, paire ou aucune
};
```

## Communication

### Stations météos

Le schéma de communication ci-dessous présente de façon simplifié la communication entre les stations météos des sites et les liaisons utilisées. La RaspberryPI embarquera un **XBee** coordinateur connecté en USB. Celui-ci recevra les trames envoyées sur son réseau. Les XBee en jaune représentent les stations.

### Schéma de communication



## Caractéristiques XBee

Tableau de comparaison de différent système sans file

	XBee Pro S1 802.15.4	Bluetooth 802.15.1	Wi-Fi 802.11b	GPRS/GSM 1XRTT/CDMA
Champ d'application	Monitoring et contrôle	Remplace les cable	Web, video, email	WAN, voix/Data
Ressources système	4KB-32KB	250KB+	1MB+	16MB+
Temps sur batterie (jours)	100-1000+	1-7	1-5	1-7
Largeur de bande (Kbps)	20-250	720	11000+	64-128
Portée (mètres)	1600	1-10+	1-100	1000+
points faibles / forts	faible consommation et prix bas	prix cher	Rapide et flexible	portée et qualité

Dans notre cas nous allons utiliser un module **Xbee Pro S1**

### Caractéristiques technique :

Alimentation: 3,3 Vcc (50 mA)

Xbee série 1

Débit HF: 250 kbps

Portée moyenne:

- 100 m en intérieur
- 1600 m en extérieur

Fréquence: 2,4 GHz

Dimensions: 28 x 25 x 10 mm

T° de service: -40 à +85 °C

Poids: 3 g

Configuration avec les commandes AT en passant par l'utilitaire proposé par linux "CuteCom"

Pour la configuration de nos XBee nous devons passer par des commandes AT pour cela on rentre 3 symboles + en mode (no end line) :

**+++ (no end line) durée avant la prochaine commande et de 10 secondes**

*Exemple de notre configuration :*

---

ATID=2018 #Le PAN ID c'est l'identifiant du réseau

ATMY=5001 #L'adresse de la carte : 5001

ATAP=0 #Mode de fonctionnement du module en utilisant la commande ATAP :

ATDH=0 #Les adresses sont sur 16bits

ATDL=FFFF #adresse de destination de Broadcast

Programmation Qt :

Le module XBee relié sur la Raspberry Pi est "vu" comme un port série virtuel. Avec Qt 4.8, on utilisera la bibliothèque **qextSerialPort** pour la communication série.

*Remarque : Avec Qt 5, on pourrait utiliser la classe QSerialPort à la place de la bibliothèque qextSerialPort.*

## Protocole WSP

**WSP** (*WISMAS Station Protocole*) est le nom du protocole. Utilisé pour la communication XBee, WSP est un protocole d'échange de type ASCII. Les trames sont envoyées périodiquement par les stations. Par défaut la période est définie à 1000ms.

Les trames commencent par un ‘\$’ afin de pouvoir déterminer le début et finissent par un ‘/r’ pour déterminer la fin. Les indicateurs sont précédés par un ‘.’ pour séparer les valeurs. Chaque bloc d'informations est séparé par des ‘,’ pour les isoler. Enfin les codes d'erreurs seront traités côté station avec ‘--’ si les informations sont erronées, et ‘\*\*’ si les capteurs n'existent pas ou si ils sont endommagés.

### Tableau d'indication :

Capteur	Préfixe	Valeurs
<b>Identifiant station</b>	ID.	Nombre entier
<b>Type de site</b>	TS.	Nombre entier de (1..3) qui spécifie si le site est un site de snowkite, ski de fond ou ski alpin.
<b>Girouette (Direction du vent)</b>	DV.	S,SE,SO,N,NO,NE,E,O
<b>Anémomètre (Vitesse du vent)</b>	VV.	Nombres entiers en km/h
<b>Thermomètre (Température air)</b>	TA.	Nombres décimaux à 10^-1 (00,0) en C°
<b>Thermomètre (Température neige)</b>	TN.	Nombres décimaux à 10^-1 (00,0) en C°
<b>Nivomètre (Hauteur de la neige)</b>	H.	Nombres entier en cm
<b>Hygrométrie (humidité)</b>	HY.	Nombre entier en %
<b>Baromètre (Pression atmosphérique)</b>	B.	Nombre entier en Pascale

## Gestion des erreurs

---

Tableau des codes erreur du protocole WSP :

Code	Préfixe	Message
100	--	Valeur introuvable
101	--	Valeur erronée
200	**	Capteur introuvable

Les codes erreurs commençant par le chiffre **1** correspondent à une erreur sur la valeur. Alors que les codes erreurs commençant par la valeur **2** correspondent à un problème sur un capteur (*Capteur débranché, absent ou endommagé*).

Exemple de trame valide :

**\$ID.1,TS.3,DV.SE,VV.20,TA.-10,TN.-15,H.75,HY.20,B.990/r**

Capteur	Préfixe	Valeurs
<b>Identifiant station</b>	ID.	<b>1</b>
<b>Type de site</b>	TS.	<b>3</b>
<b>Girouette (Direction du vent)</b>	DV.	<b>SE</b>
<b>Anémomètre (Vitesse du vent)</b>	VV.	<b>20</b>
<b>Thermomètre (Température air)</b>	TA.	<b>-10</b>
<b>Thermomètre (Température neige)</b>	TN.	<b>-15</b>
<b>Nivomètre (Hauteur de la neige)</b>	H.	<b>75</b>
<b>Hygrométrie (humidité)</b>	HY.	<b>20</b>
<b>Baromètre (Pression atmosphérique)</b>	B.	<b>990</b>

Exemple de trame comportant des erreurs :

**\$ID.4,TS.1,DV.NO,VV.50,TA.-15,TN.-10,H.250,HY.--100,B.\*\*200/r**

Capteur	Préfixe	Valeurs
<b>Identifiant station</b>	ID.	<b>4</b>
<b>Type de site</b>	TS.	<b>1</b>
<b>Girouette (Direction du vent)</b>	DV.	<b>NO</b>
<b>Anémomètre (Vitesse du vent)</b>	VV.	<b>50</b>
<b>Thermomètre (Température air)</b>	TA.	<b>-15</b>
<b>Thermomètre (Température neige)</b>	TN.	<b>-10</b>
<b>Nivomètre (Hauteur de la neige)</b>	H.	<b>250</b>
<b>Hygrométrie (humidité)</b>	HY.	<b>ERREUR valeur introuvable</b>
<b>Baromètre (Pression atmosphérique)</b>	B.	<b>ERREUR capteur introuvable</b>

Amélioration possible du protocole :

Une des améliorations à proposer est de scinder **ID.** et **TS.** Cela nous limite tout de même à 99 stations possible par Type de site.

Exemple :

\$ID.103,DV.SE,VV.20,TA.-10,TN.-15,H.75,HY.20,B.990/r

ID.103=> **1** étant le type de site, **03** l'identifiant de la station.

## Programmation Qt :

Avec Qt, l'accès à un port série virtuel réalisé par le module `QextSerialPort` :

Exemple d'une lecture de port avec Qt :

Cette implémentation est située dans le constructeur de la classe `PortXBee`.

```
//Création de l'objet port de type QextSerialPort
port = new QextSerialPort(QLatin1String(PORT_XBEE),
QextSerialPort::EventDriven, this);

//affectation du débit
port->setBaudRate(BAUD9600);

//affectation du nombre bits de données
port->setDataBits(DATA_8);

//affectation du nombre de bits de stop
port->setStopBits(STOP_1);

//ouverture du port
port->open(QIODevice::ReadWrite);

//vérification du port si il est ouvert on se connect au slot recevoirTrame
if(port->isOpen())
{
    connect(port, SIGNAL(readyRead()), this, SLOT(recevoirTrame()));
}
```

Exemple de décodage de la trame :

Cette implémentation est située dans la classe **WISMASProtocole**. C'est la méthode **decoderTrame()**.

```
// Extraire les champs de la trame
QStringList champs = trame.split(',');
// Extraire les valeurs ID et TYPE des champs de la trame
QStringList id = champs.at(0).split('.');
QStringList type = champs.at(1).split('.');
// Vérification si la station est bonne
if(id.at(1).toInt() == stationMeteo->getIdSite())
{
    // Extraire les données des champs de la trame
    QString direction_vent = champs.at(2).split('.')[1];
    QString vitesse_vent = champs.at(3).split('.')[1];
    QString temperature_air = champs.at(4).split('.')[1];
    QString temperature_neige = champs.at(5).split('.')[1];
    QString hauteur_neige = champs.at(6).split('.')[1];
    QString humidite = champs.at(7).split('.')[1];
    QString pression_air = champs.at(8).split('.')[1].split('/')[-1];
    pression_air.chop(0);
```

```
// Déclaration de la structure
DONNEES_STATION donneeStation;
QDate date = QDate::currentDate();
QTime heure = QTime::currentTime();

// Remplissage de la structure avec les données de la trame
donneeStation.dateDonnes = date.toString("dd/MM/yyyy");
donneeStation.heureDonnes = heure.toString("hh:mm:ss");
donneeStation.directionVent = direction_vent;
donneeStation.vitesseVent = vitesse_vent;
donneeStation.temperatureAir = temperature_air;
donneeStation.temperatureNeige = temperature_neige;
donneeStation.hauteurNeige = hauteur_neige;
donneeStation.humidite = humidite;
donneeStation.pressionAir = pression_air;

stationMeteo->setDonneesTrame(donneeStation);

}
```

Nous utilisons la fonction **split()**, elle permet de décomposer les chaînes de caractères. Une fois les données décomposées, elles sont stockés dans la structure **donneeStation** de type **DONNEES\_STATION**.

## Panneau lumineux

### Schéma de communication

Le panneau lumineux utilisé est un Mc Crypte :



Le panneau lumineux va afficher les données météo l'heure le nom du site et les informations complémentaires tarifs, horaire. Sur la même période du changement de l'interface.

### Caractéristiques panneau lumineux

Le panneau fait une taille de 80x7 soit 560 led de couleur rouge vert jaune.

Les caractères sont de type ASCII, et en utilisant une police de taille 6x7, il sera possible d'afficher 13 caractères.

### Méthode d'interface :

- Connexion RS232
- Bauds 9600
- Bits de données 8
- Bit de parité N
- Bit de stop 1

### Type d'alimentation d'entrée :

- Tension 12V
- Type de courant continu
- Courant 1.8A

## Protocole DBC

Le protocole de communication est défini dans le document suivant :  
[http://www1.produktinfo.conrad.com/datenblaetter/575000-599999/590996-da-01-en-Communication\\_protocol\\_LED\\_Displ\\_Board.pdf](http://www1.produktinfo.conrad.com/datenblaetter/575000-599999/590996-da-01-en-Communication_protocol_LED_Displ_Board.pdf)

Les délimiteurs de trame :

Le délimiteur de début “<IDXX>” définit le premier élément de la trame, qui est l’identifiant du panneau. Il est suivi d’un chiffre en Hexadécimal XX. Le délimiteur de fin “<E>” définit la dernière valeur. Les “<” et “>” font parti de la trame, ils sont obligatoires.

Les options :

Entre ces deux délimiteurs le protocole propose un certain nombre d’option. Dans le tableau ci-dessous, présentera une partie de ces options.

Option	Valeur	signification
<Ln>	n compris entre 1 et 8	Nombre de lignes de led
<Pn>	n compris entre A et Z	Pages du panneau
<Fn>	n compris entre A et S	Type d’animation d’entrée et de sortie.
<Wn>	n compris entre A et Z	Temps entre chaque période d’affichage de text. A = 0,5 secondes et Z = 25 secondes.
Checksum	Hexadécimal	Contrôle

Remarque : *Le checksum est une valeur hexadécimale, il calcule la taille de la trame entre les délimiteurs, et permet un contrôle d’erreur.*

Exemple de trame :

<ID01><L1><PA><FE><WC><FE>message1F<E>

Le panneau propose un système de pages afin d'afficher toutes les informations de manière cyclique. Pour résumer, je découpe toutes les informations à transmettre au panneau lumineux et les place dans des pages (<PA>, <PB>...<Pn>).

Pour communiquer avec le panneau, nous utilisons un connexion série USB. Le panneau est donc "vu" comme un port série virtuel et on utilisera la bibliothèque `QextSerialPort`.

Programmation Qt :

Cette implémentation est située dans le constructeur de la classe `PortPanneau`.

```
//Création de l'objet port de type QextSerialPort
port = new QextSerialPort(QLatin1String(PORT_PANNEAU),
QextSerialPort::EventDriven, this);

//affectation du débit
port->setBaudRate(BAUD9600);

//affectation du nombre bits de données
port->setDataBits(DATA_8);

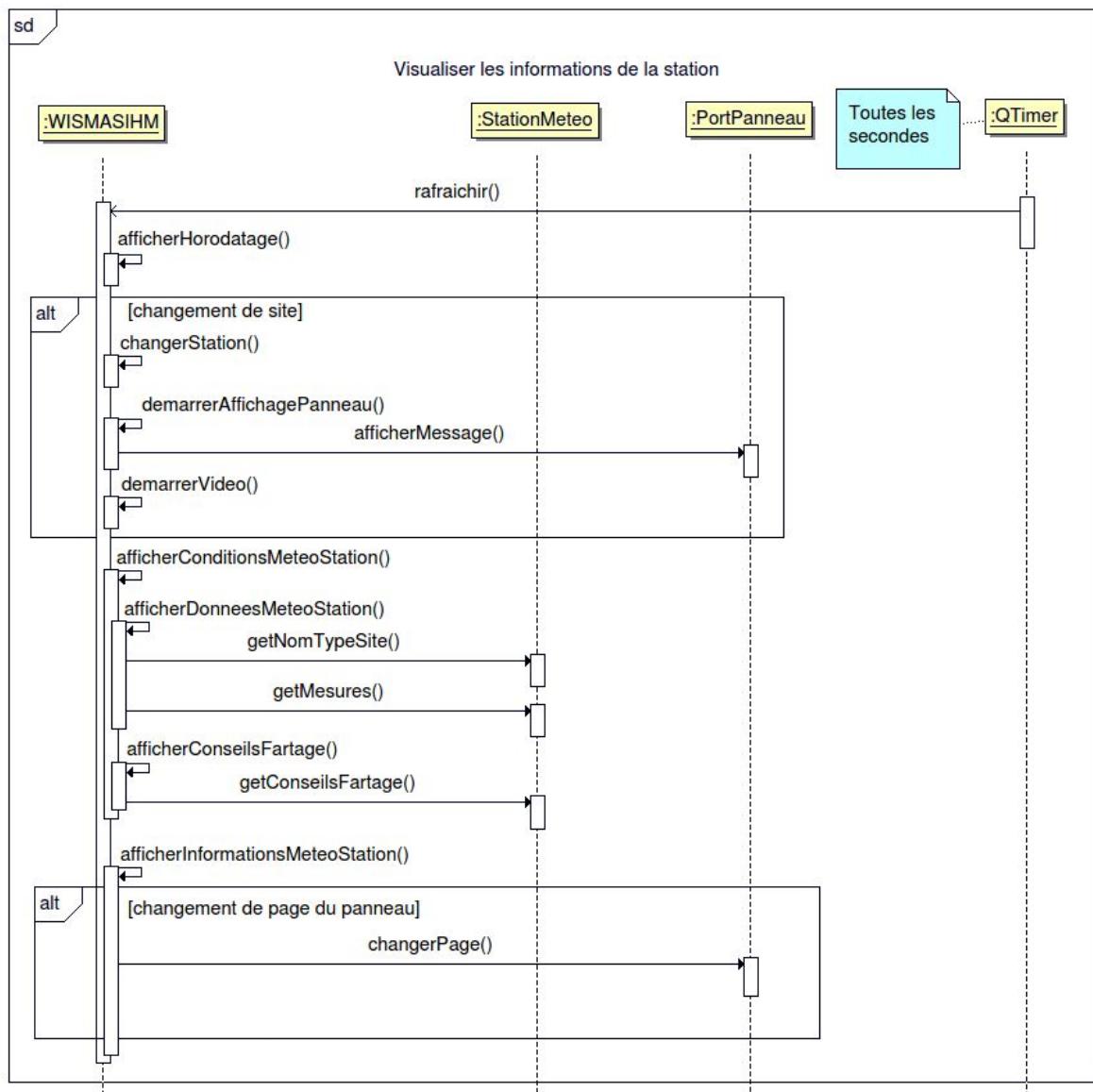
//affectation du nombre de bits de stop
port->setStopBits(STOP_1);

//ouverture du port
port->open(QIODevice::ReadWrite);

//verification du port si il est ouvert on se connect au slot recevoirTrame
if(port->isOpen())
{
    connect(port, SIGNAL(readyRead()), this, SLOT(lireTrame()));
}
```

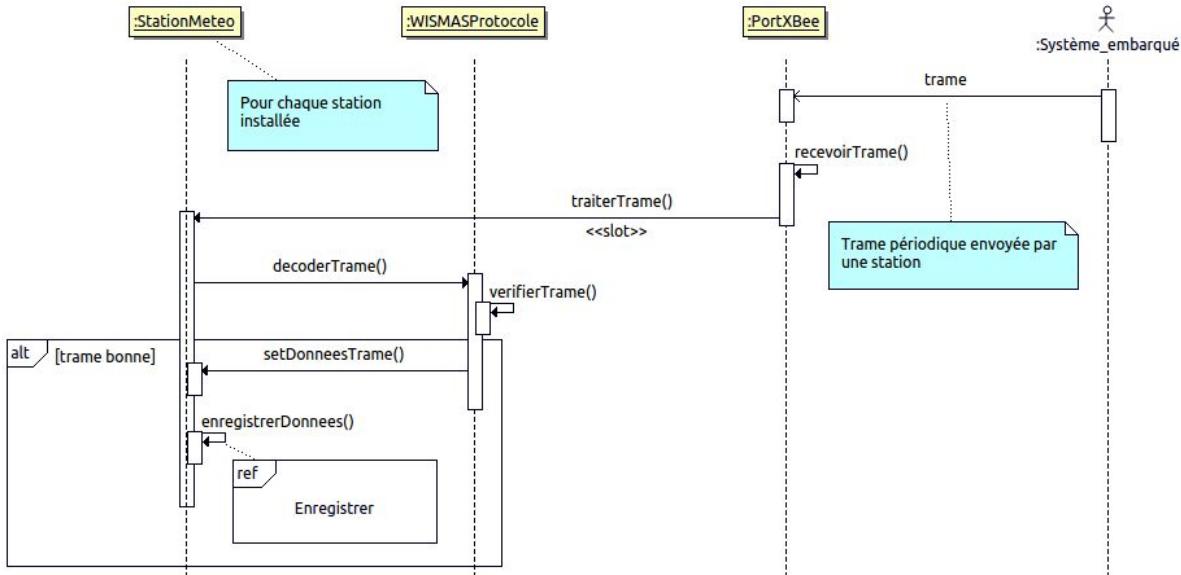
## Cas d'utilisation - Visualiser les informations de la station

Un **diagramme de séquence** représente les détails d'un cas d'utilisation UML. Il montre les interactions entre objets, en insistant sur la chronologie des envois de message.



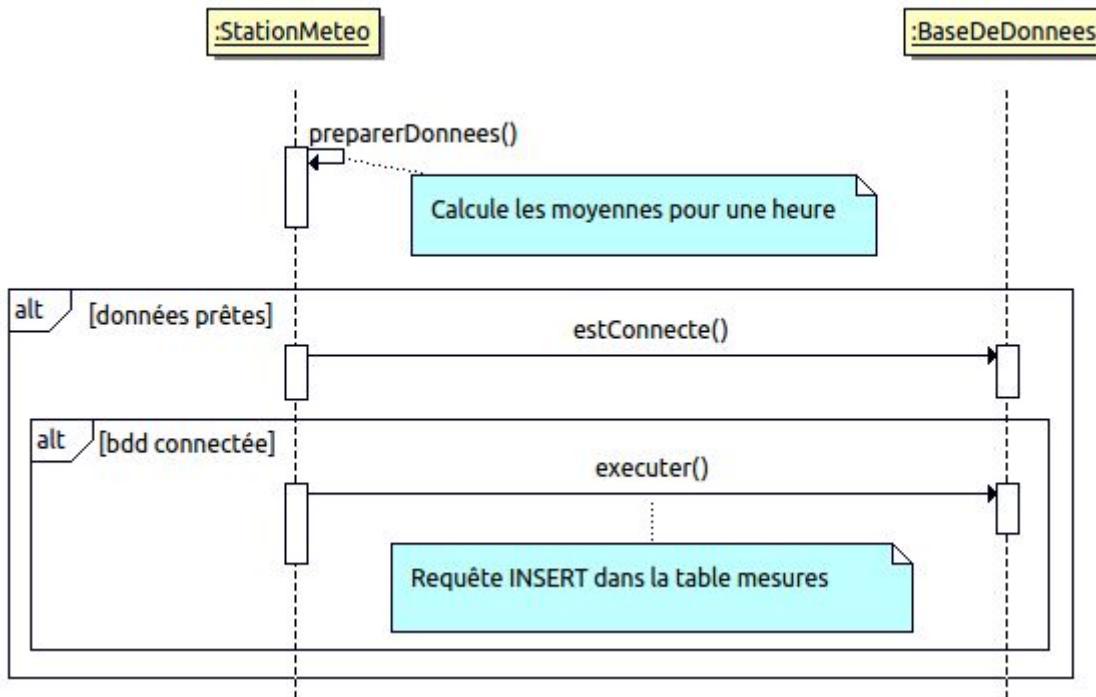
Décrit le cas qui visualise les informations (données météo, conseils de partage, vidéo) sur l'écran (Type Télévision) et visualiser les informations complémentaires sur le panneau lumineux.

## Cas d'utilisation - Relever les mesures des sites



Le diagramme de séquence ci-dessus déroule le fonctionnement de la réception et du traitement des trames envoyées par le système embarqué. L'acteur **Système\_embarqué**, va transmettre une trame via une communication XBee, la trame est transmise de manière périodique à notre objet **PortXBee**, qui lui va se charger de recevoir les trames. Une fois les trames reçus, nous allons les traiter dans l'objet **StationMeteo**, pour chaque station installées. Enfin, pour le traitement nous allons appeler l'objet **WISMASProtocole**, qui va décoder les trames, les vérifier. Et donc si la trame est bonne alors nous allons enregistrer les données depuis l'objet **StationMeteo**.

## Cas d'utilisation - Enregistrer les données



Le diagramme de séquence ci-dessus déroule le fonctionnement de l'enregistrement des informations météo en base de données. L'objet **StationMeteo** prépare les données et calcule leur moyenne. La moyenne des données météo permet d'obtenir une meilleure précision à l'enregistrement. Donc, si les données sont prêtes alors, nous allons nous connecter à la base de données, avec l'objet **BaseDeDonnees**. Si, la base de données est bel est bien connectée alors nous exécutons un requête d'insertion des données en base.

Diffusion vidéo

---

Nous avons utilisé Phonon, qui permet de manipuler des fichiers vidéo.

Exemple de code pour démarrer une vidéo.

```
//Déclaration de média dans le .h
Phonon::MediaObject* media;

//implémentation de la méthode demarrerVideo() dans le .cpp
void WISMASIHM::demarrerVideo()
{
    recupererDernierVideo();

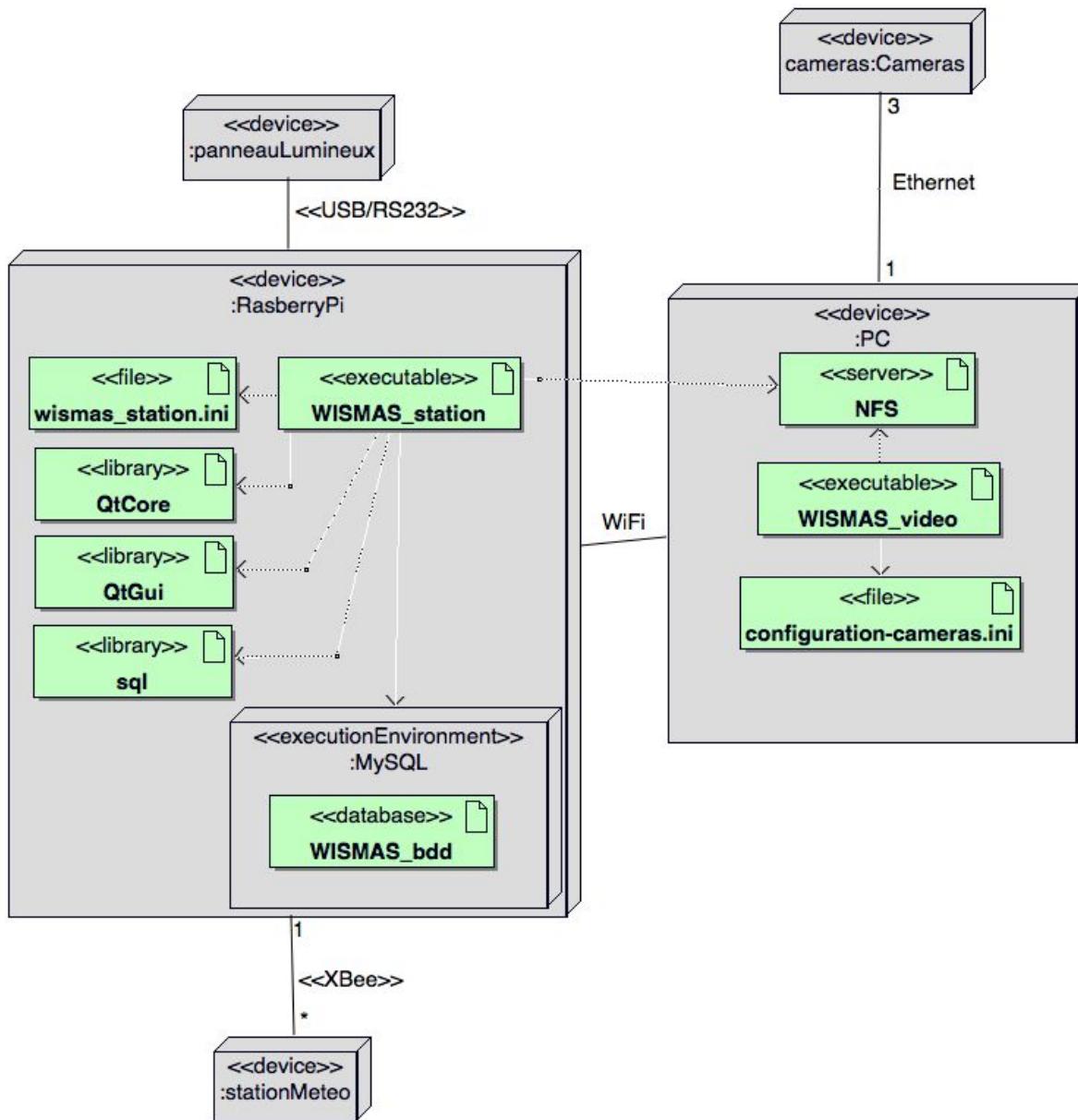
    media->stop();
    media->clear();
    chargerVideo();
    media->play();
}
```

Dans un premier temps on lance la méthode **recupererDernierVideo()**. Pour aller récupérer le chemin de la dernière vidéo enregistrée. Puis nous utilisons les méthodes proposées par Phonon. **stop()** pour stopper la vidéo au début et éviter des conflits. Ensuite **clear()** pour nettoyer le player. Et on charge une vidéo avec **chargerVideo()**. Et pour finir on lance la vidéo avec **play()**.

## Déploiement

### Diagramme de déploiement

En UML, un diagramme de déploiement est une vue statique qui sert à représenter l'utilisation de l'infrastructure physique par le système et la manière dont les composants du système sont répartis ainsi que leurs relations entre eux.



La base de données se situent sur la **Raspberry Pi**. La **Raspberry Pi** est en liaison USB avec le panneau lumineux et en liaison XBee avec le module météorologique **stationMeteo**. Les caméras sont en liaison Ethernet avec le système **WISMAS**. Une liaison WiFi permet de connecter la **Raspberry Pi** par le réseau avec le système **WISMAS**.

Le fichier **WISMAS\_station.ini** permet de paramétriser l'application **WISMAS\_station**.

**QtCore**, **QtGui**, **sql** sont les principales bibliothèques utilisées par l'application **WISMAS\_station**.

- **QtCore** : Classes non graphiques utilisées par d'autres modules
- **QtGui** : Classes Interface homme machine (IHM)
- **sql** : Classes de gestion pour les bases de données

## Fichier de configuration

Le fichier de configuration se situe à la racine de l'application et porte le nom : **WISMAS\_station.ini**

Le fichier de configuration va permettre de conserver une configuration de l'application et influer sur l'affichage du panneau lumineux et de l'écran de la station.

### Structure d'un fichier .ini

---

Les fichiers sont divisés en sections. Chaque sections comportent un certain nombre de paramètres de configuration. Chaque section commence par un titre placé entre crochets « [ » et « ] ». La valeur de chaque paramètre de configuration est indiquée par une formule : paramètre = valeur.

Les fichiers peuvent contenir des commentaires. Les commentaires sont souvent utilisés pour décrire les paramètres et les valeurs à introduire. Ils sont précédés d'un point-virgule.

*Exemple du fichier **WISMAS\_station.ini** par défaut :*

**[Configuration]**

nb\_station=2  
periode=5

**[Station\_configuration]**

port=ttyUSB0  
baud\_rate=9600  
data\_bit=8  
stop=1  
parity=0  
periode=1000

**[Panneau\_configuration]**

port=ttyUSB1  
baud\_rate=9600  
data\_bit=8  
stop=1  
parity=0

periode=1000

**[Panneau\_affichage]**

titre=Station WISMAS  
temperature\_air=true  
temperature\_neige=false  
hauteur\_neige=true  
humidite=false  
pression=false  
vitesse\_vent=true

**[Station1]**

id=101  
type=1

**[Station2]**

id=201  
type=2

Paramètre de configuration logiciel **[Configuration]**

---

*Pour la configuration de l'application on utilise le paramètre **[Configuration]**.*

Nombre de station

---

nb\_station = int #nb\_station est une valeur entière.

*Le nombre de station va permettre au programme de déterminer le nombre de trame qu'elle attend. Mais aussi le nombre de station qu'elle doit afficher.*

Période de rafraîchissement

---

periode = int #periode est une valeur entière en seconde.

*La période de rafraîchissement va permettre de réactualiser les informations entre chaque stations affichées.*

Paramètre de configuration du panneau / Station **[Station\_configuration]**  
**[Panneau\_configuration]**

---

*Pour la configuration de la station météo et du panneau lumineux on utilise les paramètres : **[Station\_configuration]** et **[Panneau\_configuration]**.*

Port

---

port = String #le port est une valeur text du port série.

*Le port permet de spécifier le nom d'un port série d'entrer. Par défaut placé à **ttyUSB0** sous linux.*

Baud rate

---

**baud\_rate = int** #Le baud rate est un valeur entière.

*Le Baud Rate définit la vitesse de transmission dans les liaisons séries. Ici par défaut il est placé à 9600.*

Bits de données

---

**data\_bit = int**

*Le nombre de bits de données dans chaque caractère peut être 5 (pour le code Baudot), 6 (rarement utilisé), 7 (pour le vrai ASCII), 8 (pour la plupart des types de données, car cette taille correspond à la taille d'un octet), ou 9 (rarement utilisé). 8 bits de données sont presque universellement utilisés dans les nouvelles applications.*

Bit de stop

---

**stop = int** #Le bit de stop est une valeur entière comprise entre (0 et 1).

*Le bit de stop est un signal de contrôle utilisé pour indiquer la fin d'un groupe de données lors d'une transmission série.*

Bit de parité

---

**parity = int** #La parité est une valeur entière comprise entre (0 pas de parité ou 1 parité).

*Le bit de parité est un bit généré par l'interface de communication et sert à la détection d'erreur de transmission série.*

Période

---

**periode = int** #La période est un nombre entier en milliseconde (ms).

*La période définit le temps de rafraîchissement des trames entre le panneau lumineux et le raspberry Pi.*

Affichage du panneau lumineux [**Panneau\_affichage**]

---

*Pour la configuration de l'affichage on utilise le paramètre [**Panneau\_affichage**].*

## Titre

---

titre = String #Le titre est une chaîne de caractère.

Le titre va s'afficher sur le panneau lumineux en première position avant les valeurs météo. Il est aussi possible de mettre aucune valeur dans ces cas là, le titre sera automatiquement désactivé de l'affichage.

## Paramètres à afficher

---

temperature\_air = bool #Température de l'air.  
temperature\_neige = bool #Température de la neige.  
hauteur\_neige = bool #Hauteur de la neige.  
humidite = bool #Humidité.  
pression = bool #Pression.  
vitesse = bool #Vitesse.

Les valeurs des paramètres sont pour type bool. Si la valeur est sur true alors on affiche l'information sur le panneau lumineux. Si la valeur est sur false alors on n'affiche pas l'information sur le panneau lumineux.

## Enregistrement station [StationX]

---

L'enregistrement station va charger les stations [StationX]. le X dans station est en réalité une valeur numérique entière que l'on incrémente à chaque nouvelle station.

## Identifiant station

---

id = 201 #L'identifiant est un valeur entière

## Type de station

---

type = 2 #Le type est une valeur entière comprise entre (1 et 3).

## API Qt

---

Pour manipuler les fichiers ini, Qt fournit la classe QSettings.

Exemple pour la lecture de fichier ini WISMASIH::chargerFichierConfig().

## Programmation Qt :

```
// Le nom du fichierINI : chemin-executable/nom-executable.ini
QString fichierINI = qApp->applicationDirPath() + "/" +
qApp->applicationName() + ".ini";

if(QFile::exists(fichierINI))
{
    QSettings parametres(fichierINI, QSettings::IniFormat);

    // Chargement des informations logiciel
    this->config.nb_station = parametres.value("Configuration/nb_station",
"0").toInt();
    this->config.periode      = parametres.value("Configuration/periode",
"10").toInt();

    // Chargement de la configuration du récepteur station
    ...
    // Chargement de la configuration du panneau lumineux
    ...
    // Chargement des paramètres d'affichage du panneau lumineux
    ...
    // Chargement des paramètres des stations
    ...

}

else
{
    return false;
}
```

Dans cet exemple on affecte à **fichierINI** le chemin absolu du fichier ini. Avec la bibliothèque **QFile** on teste si le fichier ini est présent, si oui on l'ouvre avec **QSettings**. Une fois le fichier ouvert nous avons juste à affecter les valeurs du fichier à nos variables afin de pouvoir les utiliser.

## II. TESTS DE VALIDATION

Module de diffusion d'informations	Oui	Non
Le protocole de communication est spécifié et mis en œuvre	V	
Les mesures météorologiques relevées sont enregistrées	V	
Les informations sur la station sont consultables sur l'écran	V	
Les conditions météorologiques d'un site sont consultables sur le panneau lumineux	V	
Remarques		



## REVUE DE PROJET FINALE SYSTEME D'ACQUISITION VIDEO - GRELET PIERRE

### WISMAS version 1.0

***Weather Information System Multi Activity Station***

# TABLE DES MATIÈRES

<b>ETUDE ET ANALYSE</b>	<b>3</b>
CAHIER DES CHARGES	3
Présentation générale du système	3
Analyse de l'existant	4
Expression du besoin	4
Tâches à réaliser	5
Contraintes techniques	5
OUTILS ET RESSOURCES	6
Logiciels	6
Matériels	6
Diagramme de déploiement	7
PLANIFICATION	8
<b>CONCEPTION</b>	<b>10</b>
DIAGRAMMES	10
Diagramme de cas d'utilisation	10
PARAMÉTRER LE SYSTÈME	10
DÉMARRER LE SYSTÈME D'ACQUISITION	13
Diagramme de classes	17
Diagramme de séquences	21
PARAMÉTRER LE SYSTÈME	21
DÉMARRER LE SYSTÈME D'ACQUISITION	26
ENREGISTRER UNE VIDÉO	27
SUPPRIMER LES ENREGISTREMENTS	30
<b>INTERFACE HOMME MACHINE</b>	<b>34</b>
<b>TEST DE VALIDATION</b>	<b>37</b>
<b>GLOSSAIRE</b>	<b>38</b>

# I. ETUDE ET ANALYSE

## A. CAHIER DES CHARGES

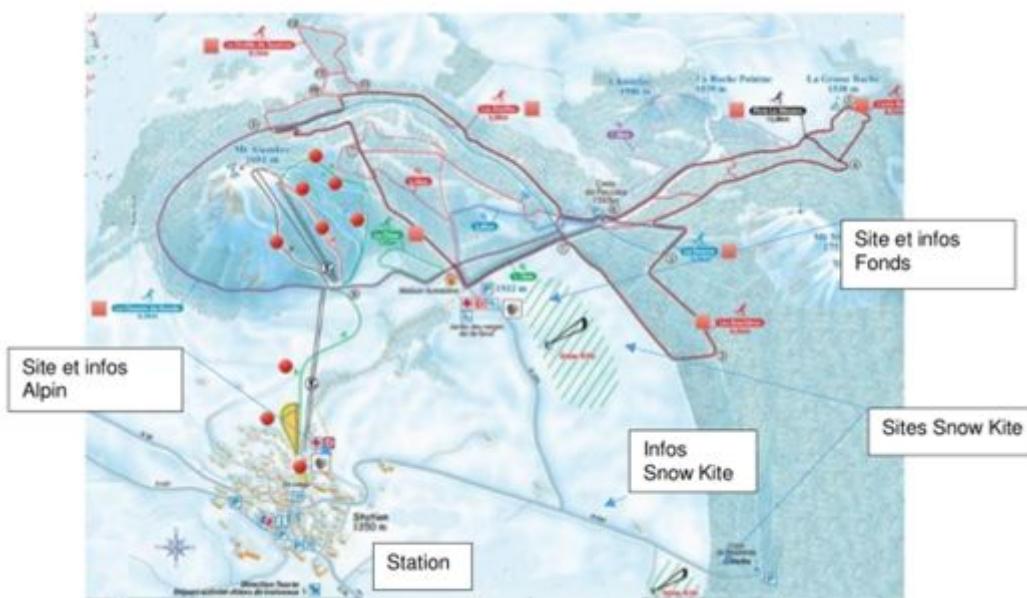
### 1. Présentation générale du système

Développer un système d'information météo pour une station multi activités nommé **WISMAS** (Weather Information System Multi Activity Station).

**WISMAS** est une station de ski située en moyenne montagne multi-activités. Le système opère sur 3 sites où l'on peut faire du ski de fond, du ski de piste et du snow kite.

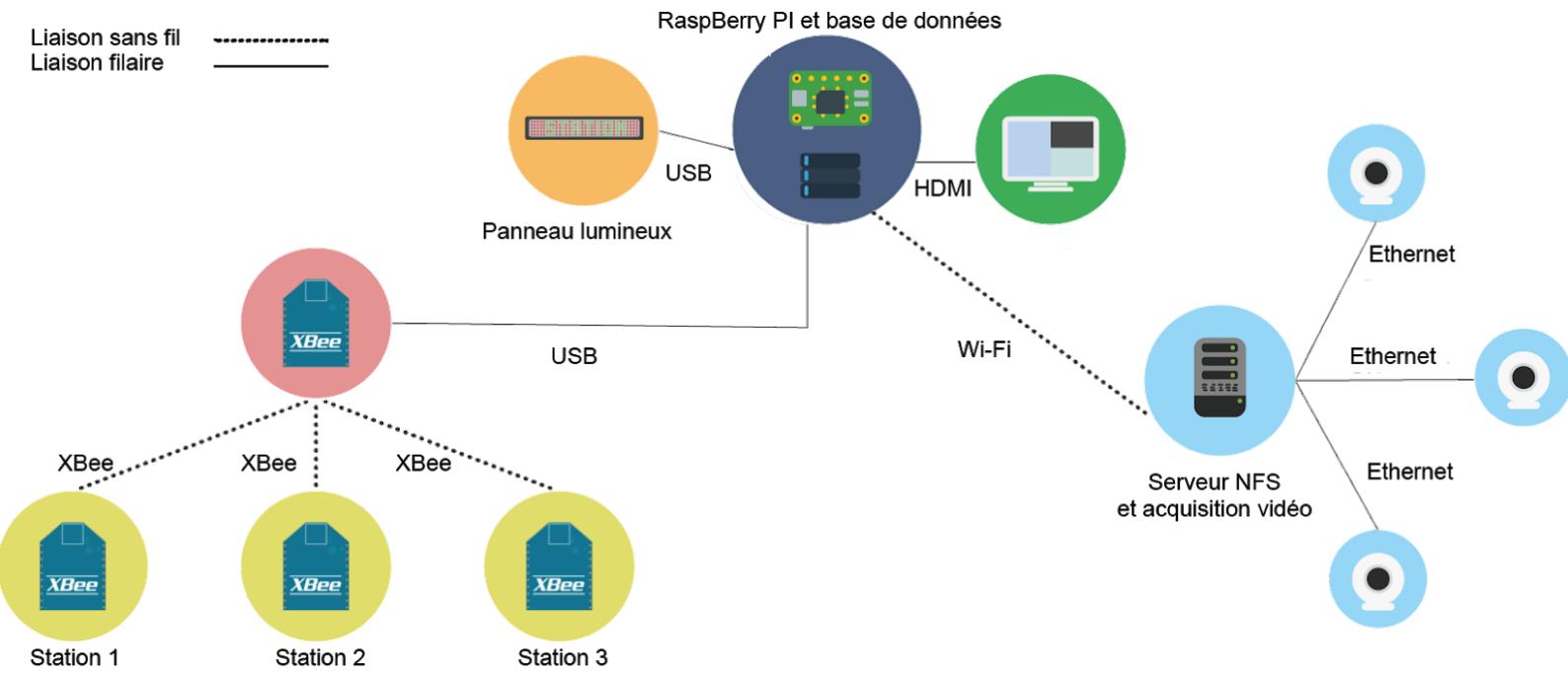
Son objectif sera donc de :

- Faire des mesures météorologiques sur plusieurs sites
- Prendre des séquences vidéo à partir de caméra
- Afficher l'ensemble de ces renseignements sur les sites d'achat des forfaits



Il se compose des modules suivants :

- Module de météorologie
- **Module d'acquisition « vidéo »**
- Module de diffusion d'informations



## 2. Analyse de l'existant

Le système existant sur le site est un ensemble de modules :

- Un panneau lumineux affichant les conditions météorologiques
- Un écran diffusant des informations (météo, vidéos, messages...)
- Des caméras capturant des séquences vidéo

## 3. Expression du besoin

L'objectif est de développer **WISMAS** avec un ensemble de fonctionnalités :

- Un système embarqué, les applications nécessaires à la mise en œuvre des capteurs et de la transmission sans fil des données
- Un PC d'acquisition doté d'un logiciel de gestion de caméras
- Un poste de diffusion doté d'un logiciel de publication d'informations météos

### Module de météorologie

- Capteur de température de l'air
- Capteur d'humidité de l'air
- Capteur de pression atmosphérique
- Capteur de température de la neige
- Capteur de hauteur de neige
- Capteur de vitesse de vent
- Capteur de direction de vent

### Module d'acquisition « vidéo »

- Acquérir et enregistrer des séquences vidéo pour une diffusion ultérieure
- Commander les déplacements d'une caméra pour une acquisition panoramique
- Paramétrier les caméras

### Module de diffusion d'informations

- Diffuser des informations sur la station (horaires, accès...) et les conditions météorologiques avec vidéo
- Afficher les conseils de partage en fonction des conditions météos
- Relever et enregistrer les mesures météorologiques sur une base de données

## 4. Tâches à réaliser

### Module d'acquisition « vidéo »

- Démarrer le module d'acquisition vidéo
- Acquérir et enregistrer une vidéo
- Déplacer une caméra
- Paramétrier les caméras

INSTALLATION	MISE EN ŒUVRE	CONFIGURATION	RÉALISATION	DOCUMENTATION
Environnement de développement Caméras IP Wanscam	Environnement de développement Caméras IP Wanscam	Caméras IP Wanscam	Diagrammes UML IHM du module Code source de l'application	Dossier technique Documents relatifs au module Guide de mise en route et d'utilisation du module

## 5. Contraintes techniques

L'environnement ne permet pas de filmer des sites différents, distants de plusieurs kilomètres.

Le module d'acquisition vidéo doit pouvoir :

- Afficher le flux de 3 caméras simultanément
- Enregistrer au moins 3 vidéos à un moment « t »

## B.OUTILS ET RESSOURCES

### 1. Logiciels

Ressources	Version
Système d'exploitation du PC « acquisition »	GNU/Linux Ubuntu LTS 12.04
Environnement de développement	Qt Creator ; Qt Designer
API GUI PC « acquisition »	Qt 4.8
Compilateurs	GNU g++ for Linux
Atelier de génie logiciel	BOUML 7.4
Logiciel de gestion de versions	Subversion RiouxSVN
Générateurs de documentation	Doxxygen 1.8.11

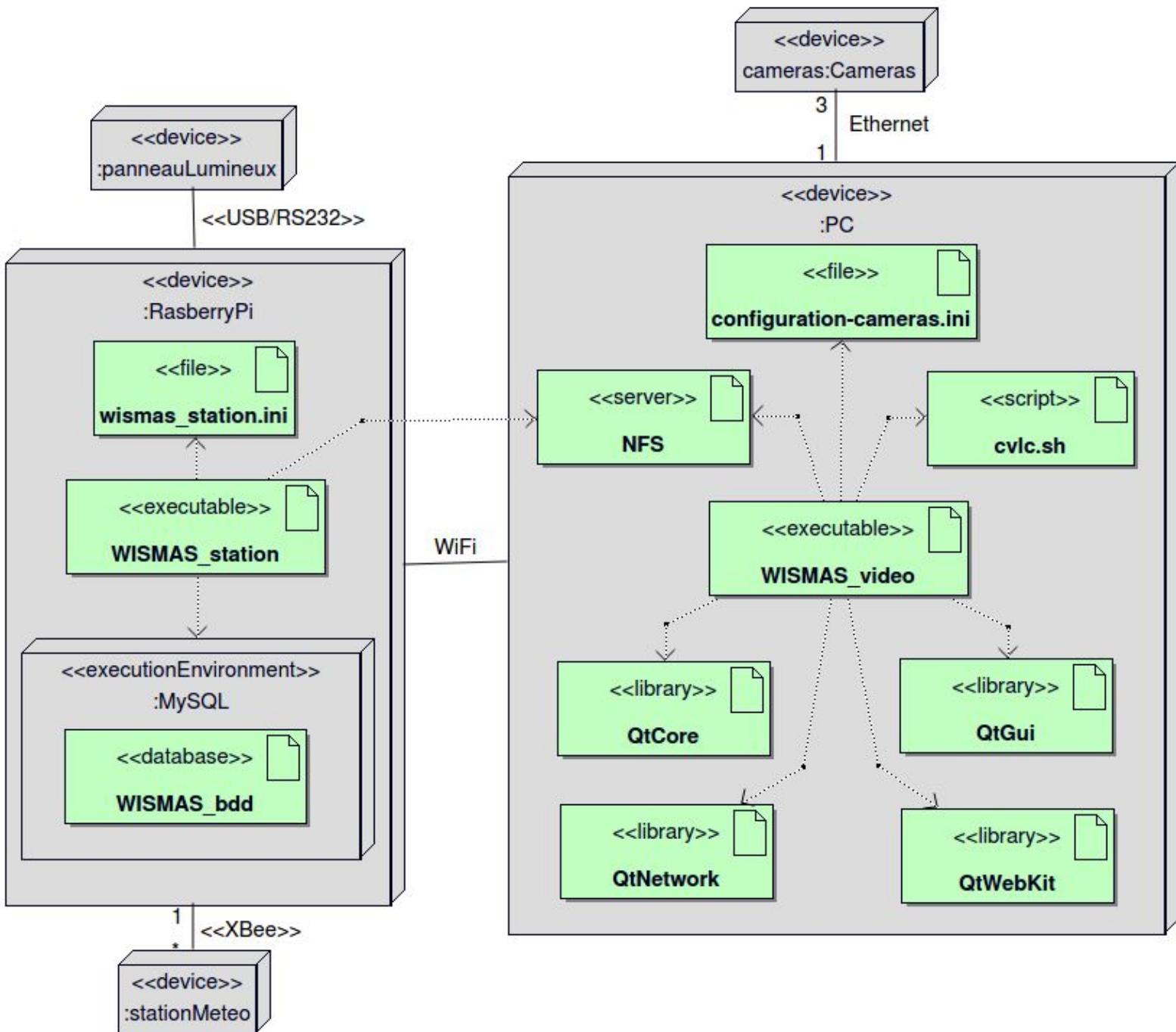
### 2. Matériels

	Caméra de surveillance IP	Caractéristiques
	<b>Standard WLAN</b> <b>Résolution</b> <b>Rayon de mouvement</b> <b>Compression vidéo</b> <b>Température de service</b>	<b>802.11b/g/n</b> <b>640 x 480 pixels</b> <b>Vertical : 105°, horizontal : 355°</b> <b>MJPEG</b> <b>5 à +40 °C</b>

**802.11** est en ensemble de normes qui spécifie l'implémentation de réseaux numériques locaux à liaison sans fil. Le module d'acquisition vidéo pourrait donc être sans fil pour faciliter une installation des caméras en montagne.

**MJPEG** est le codec vidéo qui compresse les images en **JPEG**. En partant de ce format, il est possible de transcoder les fichiers vidéos en format **MP4**, le format utilisé par le module de diffusion d'informations.

### 3. Diagramme de déploiement



L’application du module d’acquisition vidéo et le serveur **NFS** se situent sur un ordinateur de type PC. La base de données se trouve sur la **RaspberryPi**. La **RaspberryPi** est en liaison **USB** avec le panneau lumineux et en liaison **XBee** avec le module météorologique **stationMeteo**. La **RaspberryPi** est en liaison **WiFi** avec le PC. Les caméras sont en liaison **Ethernet** avec le système **WISMAS**.

Le fichier **configuration-cameras.ini** permet de paramétriser le module d’acquisition vidéo.

Le serveur **NFS** permet de partager les vidéos enregistrés avec le module de diffusion d'informations.

Le script **cvlc.sh** permet d'enregistrer les vidéos en format **MP4** en utilisant le logiciel **VLC**.

**QtCore**, **QtGui**, **QtNetwork** et **QtWebkit** sont les principales bibliothèques utilisées par l'application **WISMAS\_video**.

- **QtCore** : Classes non graphiques utilisées par d'autres modules
- **QtGui** : Classes Interface homme machine (IHM)
- **QtNetwork** : Classes programmation réseau
- **QtWebkit** : Classes utilisées pour interpréter le contenu web

## C.PLANIFICATION

Le diagramme de GANTT représente la planification du module de diffusion d'informations et du module d'acquisition vidéo.

Le projet est planifié de la manière suivante :

- **Analyse**  
L'analyse cerne les besoins du client, il faut donc rédiger un document « Tests et validation » qui met en évidence les tâches à effectuer.
- **Conception**  
La conception permet de concevoir les diagrammes UML qui donnent une vue d'ensemble sur la structure du code.
- **Implémentation**  
L'implémentation est le codage de l'application **WISMAS\_video** pour le module.
- **Test**  
Les tests détectent les bugs, ils seront alors corrigés pour la revue suivante.

La documentation technique est sans arrêt mis à jour pour assurer un suivi concret et continu du projet.

La station **WISMAS** est soumise à 3 livraisons.

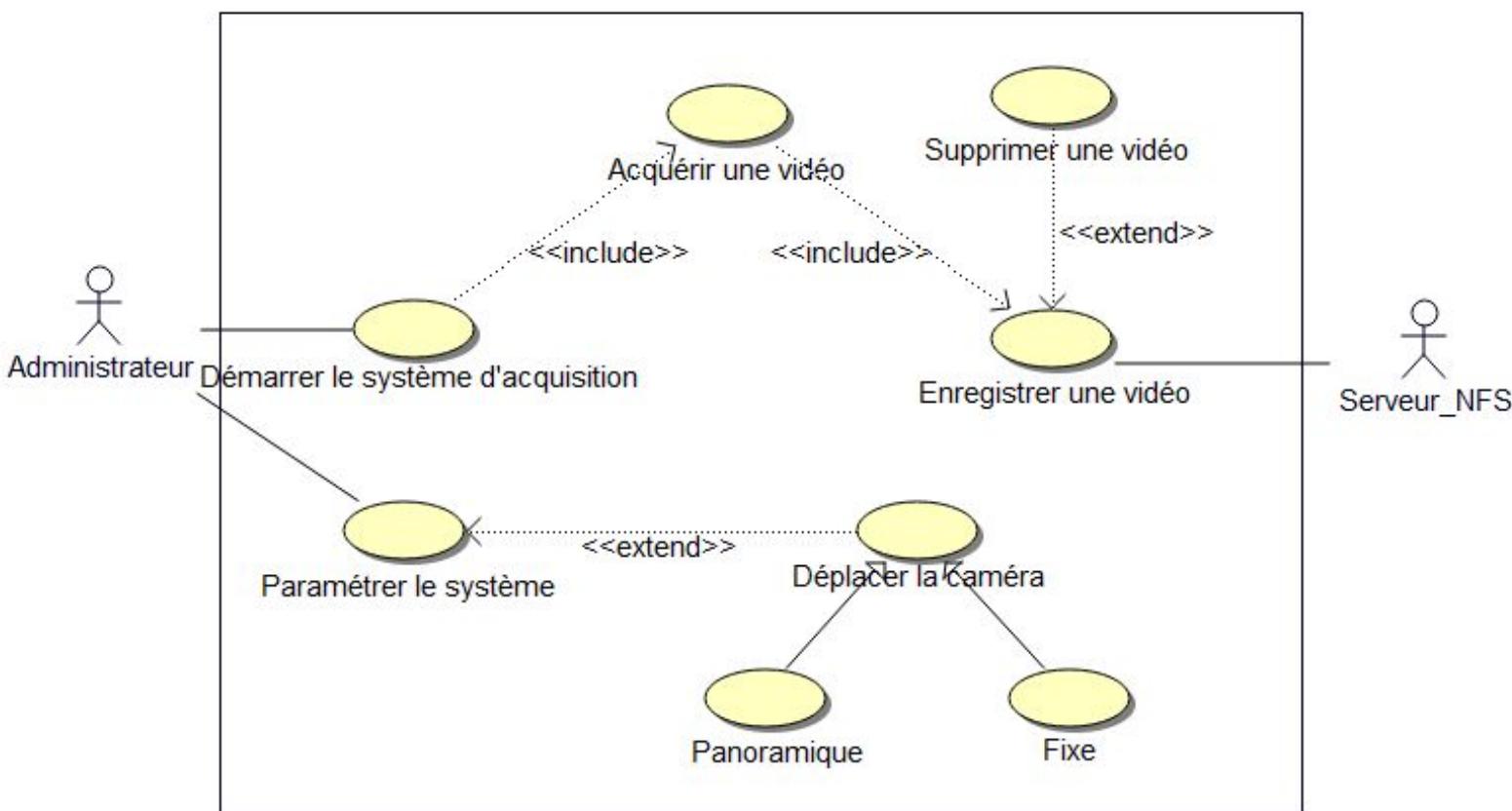
- *Revue n°0 : 21/02/2018*
- *Revue n°1 : 28/03/2018*
- *Revue n°2 : 25/05/2018*
- *Revue finale : 11/06/2018*

TPÉ	Nom	Travail	févr.	mars	avril	mai	juin
1	Rédiger document tests et validation	1h	Début du projet 07 Févr. 2018				
2	Définir une application itérative	1h					
3	Diagramme de Gantt	5h					
4	▼ <b>Diagramme UML</b>	5j					
4.1	▼ <b>Module d'acquisition vidéo</b>	2j 3h					
4.1.1	Diagramme de cas d'utilisation	6h					
4.1.2	Diagramme de séquence	6h					
4.1.3	Diagramme de classe	6h					
4.2	▼ <b>Module de diffusion d'informations</b>	2j 3h					
4.2.1	Diagramme de cas d'utilisation	6h					
4.2.2	Diagramme de séquence	6h					
4.2.3	Diagramme de classe	6h					
5	Définir une planification itérative	4h					
6	Prototyper l'IHM	4h					
7	▼ <b>Coder les classes du système</b>	22j					
7.1	▼ <b>Module d'acquisition vidéo</b>	8j					
7.1.1	IHM	2j					
7.1.2	Paramétriser les caméras	2j					
7.1.3	Gérer le déplacement des caméras	1j					
7.1.4	Enregistrer des vidéos	2j					
7.1.5	Supprimer des vidéos	1j					
7.2	Module de diffusion d'informations	14j					
8	Realiser les tests unitaires	1j					
9	Intégrer l'application complète	2j					
10	▼ <b>Documentation</b>	4j					
10.1	Rédiger le dossier technique et les documents relatifs	2j					
10.2	Produire un guide de mise en route et d'utilisation du système	2j					
11	▼ <b>Livraison</b>	4j 5h					
11.1	Revue n°0	1j					
11.2	Revue n°1	1j					
11.3	Revue n°2	1j					
11.4	Remise du dossier	5h 50min					
11.5	Soutenance finale	1j					

## II. CONCEPTION

### A. DIAGRAMMES

#### 1. Diagramme de cas d'utilisation



L'administrateur pourra effectuer deux actions, paramétrier les caméras vidéo ou bien démarrer le système d'acquisition vidéo.

#### ▷ PARAMÉTRER LE SYSTÈME

Lu au démarrage, le fichier de configuration « .ini » des caméras se caractérise de la manière suivante :

- Une section de paramétrage situé entre crochets [...]
- Un paramètre sous la syntaxe **[paramètre]=[valeur]**

Paramétrier depuis le fichier de configuration « .ini », l'administrateur a la possibilité de modifier le comportement du module d'acquisition vidéo :

- Le nombre de caméras
- Le nombre de vidéos pour une caméra
- Le niveau de suppression pour toutes les caméras

	Paramètres général	Description
<b>[General]</b> nb_cameras=3 nb_video=3 niveau_suppression=Aucun	<b>nb_cameras</b>	Le nombre de caméras installé sur WISMAS
	<b>nb_videos</b>	Le nombre de vidéos maximum sauvegardé
	<b>niveau_suppression</b>	La méthode de suppression des enregistrements vidéo

L'administrateur peut paramétrier les caméras (IHM/fichier de configuration « .ini ») :

- L'état de la caméra
- Le nom du site
- Le chemin vidéo
- L'adresses IP et le numéro de port
- L'identifiant et le mot de passe
- Le type de déplacement lors de l'acquisition vidéo
- La durée d'une capture vidéo
- La périodicité pour relancer la capture vidéo
- La résolution

Fichier de configuration	Paramètres caméra	Description
<b>[Cameral]</b> etat=1 nom=Alpin chemin_video=./videos/alpin/ adresse_IP=192.168.52.221 numero_port=99 identifiant=admin mot_de_passe= type_deplacement=Panoramique duree=15 periode=90 resolution=480	<b>etat</b>  <b>nom</b>  <b>chemin_video</b>  <b>adresse_IP</b>  <b>numero_port</b>  <b>identifiant</b>  <b>mot_de_passe</b>  <b>type_deplacement</b>  <b>duree</b>  <b>periode</b>  <b>resolution</b>	L'état de la caméra  Le nom du site où la caméra se situe  Le chemin où les enregistrements de la caméra sont sauvegardés  L'adresse IP  Le numéro de port  L'identifiant pour authentification  Le mot de passe pour authentification  Le type de déplacement; Panoramique/Fixe  La durée de capture vidéo  La période d'attente pour relancer une capture vidéo automatique  La résolution d'affichage

Les paramètres sont enregistrés dans la structure **Paramètres**.

```
struct Paramètres
{
    QString etat;
    QString nomSite;
    QString adresseIP;
    QString numeroPort;
    QString identifiant;
    QString motDePasse;
    QString type_deplacement;
    QString cheminVideo;
    QString duree;
    QString periode;
    QString resolution;
};
```

Certains paramètres (général et caméra) peuvent modifier le comportement de l'application en ayant une valeur différente.

Paramètres caméra	Valeur	Fonctionnalité
etat	0	Caméra activée
	1	Caméra désactivée
type_deplacement	Panoramique	Déplacement de la caméra
	Fixe	Aucun déplacement

**Paramètres caméra**

**Valeur**

**Fonctionnalité**

etat	0	Caméra activée
	1	Caméra désactivée

type_deplacement	Panoramique	Déplacement de la caméra
	Fixe	Aucun déplacement

Paramètres général	Valeur	Fonctionnalité
niveau_suppression	Tous	Tous les enregistrements sont supprimés
	Aucun	Aucun enregistrements supprimés
	Nb_enregistrements	L'enregistrement le plus récent est conservé pour chaque caméra

**Paramètres général**

**Valeur**

**Fonctionnalité**

niveau_suppression	Tous	Tous les enregistrements sont supprimés
	Aucun	Aucun enregistrements supprimés
	Nb_enregistrements	L'enregistrement le plus récent est conservé pour chaque caméra

## ▷ DÉMARRER LE SYSTÈME D'ACQUISITION

Les enregistrements vidéo sont partagés avec le module de diffusion d'information. Ils sont donc conservés sur un serveur **NFS** utilisant les protocoles TCP/IP, RPC et XDR.

Le serveur **NFS** désigne le système qui possède physiquement les ressources (enregistrements vidéo) et les partage sur le réseau avec d'autres systèmes. Il est configurable via le fichier **/etc(exports)**. Le fichier contient la liste des ressources partagées et possède le format suivant :

- [dossier\_partagé] [client1]([options]) [client2]([options]) ...

```
/srv/WISMAS 192.168.0.0/16(ro) 192.168.52.125(rw)
```

A chaque modification, il faut recharger la configuration du service :

```
$ /etc/init.d/nfs-kernel-server reload
```

Pour obtenir l'accès à la liste des ressources partagées :

```
$ showmount -e
```

Pour monter une ressource manuellement sur un **client** :

```
$ mount 192.168.52.125:/srv/WISMAS/ videos
Export list for iris-HP-Pro-3500-Series:
/srv/WISMAS 192.168.52.125,192.168.0.0/16
```

Pour monter une ressource automatiquement il faut ajouter la ressource dans le fichier **/etc/fstab** :

- [adresseIP\_serveur]:[dossier\_partagé] [point\_montage] nfs [options] 0

```
192.168.52.125:/srv/WISMAS/ /home/iris/Documents/wismas/trunk/WISMAS_video/bin/videos nfs rw 0 0
```

L'enregistrement des vidéos se fait par **VLC**. Il doit lire les flux réseaux de caméra et les capturer en format **MJPEG**. Cependant, le module de diffusion d'information lit uniquement le format **MP4**. Il faut donc transcoder les fichiers du format **MJPEG** en **MP4**.

Un script permet alors l'enregistrement des vidéos.

La fonction **arreter()** arrête le processus d'enregistrement.

```
arreter()
{
    echo "Arret du PID vlc $PID_VLC" >> ./cvlc.log

    kill $PID_VLC
}
```

Elle s'exécutera à la réception des signaux de terminaison :

- **1 SIGHUP** : Fin de session
- **2 SIGINT** : Interruption au clavier **Ctrl + C**
- **3 SIGQUIT** : Quitter au clavier **Ctrl + D**
- **9 SIGKILL** : Mort du processus (ne peut être intercepté)
- **15 SIGTERM** : Terminaison normale du processus

```
trap arreter 0 1 2 3 9 15
```

Si l'enregistrement de la caméra est bien lancé, le système met fin au processus lorsque la durée du **QTimer** est écoulé (**timeout()**) ou que l'administrateur arrête subitement l'application (**Ctrl + Q**).

```
void Video::arreter()
{
    if(enregistrement == true)
    {
        ...
        processus->terminate();
        ...
    }
}
```

La fonction **terminate()** envoie le signal de terminaison 15 (**SIGTERM**) au PID du processus.

L'enregistrement d'une vidéo nécessite la création d'un nouveau processus. En effet, il est nécessaire de paralléliser l'enregistrement de plusieurs caméras.

Les paramètres de la fonction **enregistrerVideo()** sont récupérés en argument.  
La fonction **start()** permet de créer et démarrer un nouveau processus. Le processus créé exécute le script (**cvlc.sh**) avec les arguments récupérés dans la variable **arguments** (**adresseIP**, **numeroPort...**).

```
void Video::enregistrerVideo(const int numeroCamera, const QString nomSite, const QString adresseIP, const QString numeroPort, const QString identifiant, const QString motDePasse, const int duree)
{
    ...

    // Format mp4
    programme = "./cvlc.sh";
    if(motDePasse.length() == 0)
    {
        arguments << adresseIP
                    << numeroPort
                    << identifiant
                    << nomFichier
                    << QString::number(duree);
    }
    else
    {
        arguments << adresseIP
                    << numeroPort
                    << identifiant
                    << motDePasse
                    << nomFichier
                    << QString::number(duree);
    }

    ...

    processus->start(programme, arguments);
}

...
```

Le lien **URL** pour enregistrer un flux vidéo nécessite :

- l'adresse IP de la caméra
- le numéro de port
- l'identifiant pour s'y authentifier
- le mot de passe (optionnel)
- le nom du fichier comporte aussi le chemin d'accès au dossier d'enregistrement
- la durée d'enregistrement du flux vidéo

Lance en arrière plan la commande d'enregistrement **VLC** sans interface graphique.

Pour 5 arguments (sans mot de passe) -

```
/usr/bin/vlc -I dummy
"http://[adresseIP]:[numeroPort]/videostream.cgi?user=[identifiant]&pwd=&resolution=32&rate=0"
--sout "#transcode{vcodec=h264} :file{mux=ts,dst=[nomFichier]}"

sleep [duree]
```

Pour 6 arguments (avec mot de passe) -

```
/usr/bin/vlc -I dummy
"http://[adresseIP]:[numeroPort]/videostream.cgi?user=[identifiant]&pwd=[motDePasse]&resolution=3
2&rate=0" --sout "#transcode{vcodec=h264} :file{mux=ts,dst=[nomFichier]}" &

sleep [duree]
```

Que ce soit dans un cas ou l'autre, le script lance l'enregistrement de la vidéo en utilisant le logiciel **VLC** : /usr/bin/vlc

**VLC** est lancé sans interface graphique : -I dummy

Les arguments permettent alors de compléter l'URL du flux vidéo de la caméra :

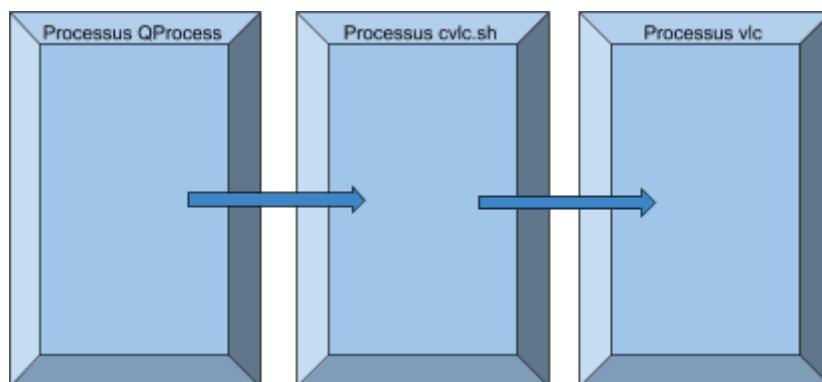
```
http://[adresseIP]:[numeroPort]/videostream.cgi?user=[identifiant]&pwd=[motDePasse]
&resolution=32&rate=0
```

Et d'attribuer une durée à l'enregistrement : sleep([duree])

Le fichier vidéo est encodé en **MP4** vers un fichier situé dans un répertoire :

```
#transcode{vcodec=h264}:file{mux=ts,dst=[nomFichier]}
```

Le module d'acquisition vidéo exploite donc la notion de processus lourd. Les processus du système sont isolés les uns des autres. Chacun possède un contexte qui lui est propre. Un processus pour les caméras (**WISMAS\_video**), un processus pour le script (**cvlc.sh**) et un processus pour l'enregistrement (**VLC**).



## 2. Diagramme de classes



IHMWISMAS	
- manager : QNetworkAccessManager*	
- fichierINI : QString	
- <<Qvector>> timerCamera : QTimer*	
- timerInitial : QTimer*	
- timerFinal : QTimer*	
- actionQuitter : QAction*	
- connecter() : void	
- creerCameras() : void	
- creerUrlCamera() : QUrl	
- demarrerTimers(numeroCamera : int, parametres : Parametres) : void	
- arreterTimers()	
+ IHMWismas(parent : QWidget*)	
+ ~IHMWismas()	
+ fini() : void	
+ parametrer() : void	
+ demarrer() : void	
+ gererBoutonEtat() : void	
+ gererTypeDeplacement() : void	
+ activerBoutonsCamera() : void	
+ desactiverBoutonsCamera() : void	
+ déplacerGauche() : void	
+ déplacerDroite() : void	
+ déplacerHaut() : void	
+ déplacerBas() : void	
+ sauvegarderParametres() : void	
+ afficherParametresCamera() : void	
+ rafraichirFluxVideo(identifiantCamera : QString, parametres : Parametres) : void	
+ afficherFluxVideo() : void	
+ sauvegarderPositionInitiale() : void	
+ sauvegarderPositionFinale() : void	
+ initialiserPosition() : void	
+ demarrerAcquisitionVideo() : void	
+ déplacerPositionInitiale() : void	
+ déplacerPositionFinale() : void	
+ arreterAcquisitionVideo() : void	
+ afficherMessage(message : QString) : void	

La classe **IHMWISMAS** est l'interface du module d'acquisition vidéo. Elle crée les caméras ainsi que leurs paramètres définis par le fichier de configuration «.ini» (adresse IP, numéro de port, durée...).

<b>Camera</b>
- identifiant : QString
- numeroCamera : int
+ Camera()
+ Camera(numeroCamera : int, identifiant : QString)
+ ~Camera()
+ setIdentifiant(identifiant : QString) : void
+ setNumero(numeroCamera : int) : void
+ getParametres() : Parametres
+ lireParametres() : void
+ seDeplacer() : QString
+ afficherMessage(message : QString)
+ progression(chargement : int)
+ enregistrer() : void
+ arreter() : void
+ progresser(pct : int) : void
+ terminer(chargement : int) : void
+ supprimer() : void

La classe **Camera** gère le déplacement panoramique des caméras de surveillance IP.

<b>Paramétrage</b>
- fichierINI : QString
+ Parametrage()
+ ~Parametrage()
+ setParametres(parametres : Parametres) : void
+ getParametres() : Parametres
+ lireParametres(identifiantCamera : QString) : void

La classe **Paramétrage** assure la lecture et l'écriture des paramètres caméras (adresse IP, numéro de port...) définis dans le fichier de configuration «.ini».

### Video

```

- processus : QProcess*
- timerProgression : QTimer*
- timerEnregistrement : QTimer*
- enregistrement : bool
- chargement : int
- numeroVideo : int
- duree : QString
- fichierINI : QString
- cheminVideo : QString
+ getNumeroVideo() : int
- creerCheminVideo(numeroCamera : int) : void
- demarrerTimers(duree : int) : void
- arreterTimers() : void
+ Video()
+ ~Video()
+ enregistrerVideo(numeroCamera : int, nomSite : QString, adresseIP : QString, numeroPort : QString, identifiant : QString, motDePasse : QString, duree : int) : void
+ arreterEnregistrements() : void
+ fini() : void
+ progression(chargement : int) : void
+ arreter() : void
+ progresser() : void
+ supprimerEnregistrements(nomSite : QString) : void
+ supprimerTousLesEnregistrements(repertoire : QDir) : void
+ supprimerNbEnregistrements(repertoire : QDir, nbEnregistrementsAConserver : uint, nomSite) : void

```

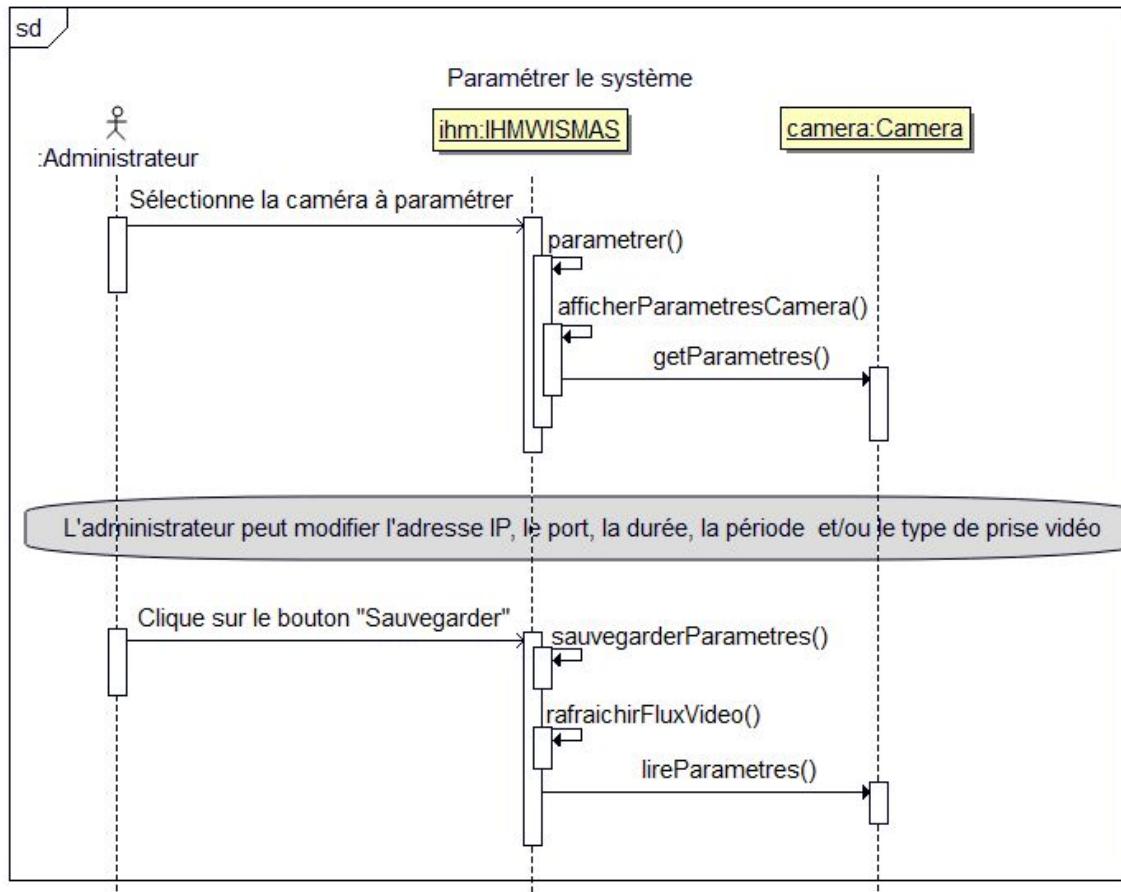
La classe **Vidéo** s'occupe de lancer, d'interrompre et de supprimer les enregistrements vidéo.

<<struct>> <b>Parametres</b>
<ul style="list-style-type: none"> <li>+ etat : QString</li> <li>+ nomSite : QString</li> <li>+ adresseIP : QString</li> <li>+ numeroPort : QString</li> <li>+ identifiant : QString</li> <li>+ motDePasse : QString</li> <li>+ typeDeplacement : QString</li> <li>+ cheminVideo : QString</li> <li>+ duree : QString</li> <li>+ periode : QString</li> <li>+ resolution : QString</li> </ul>

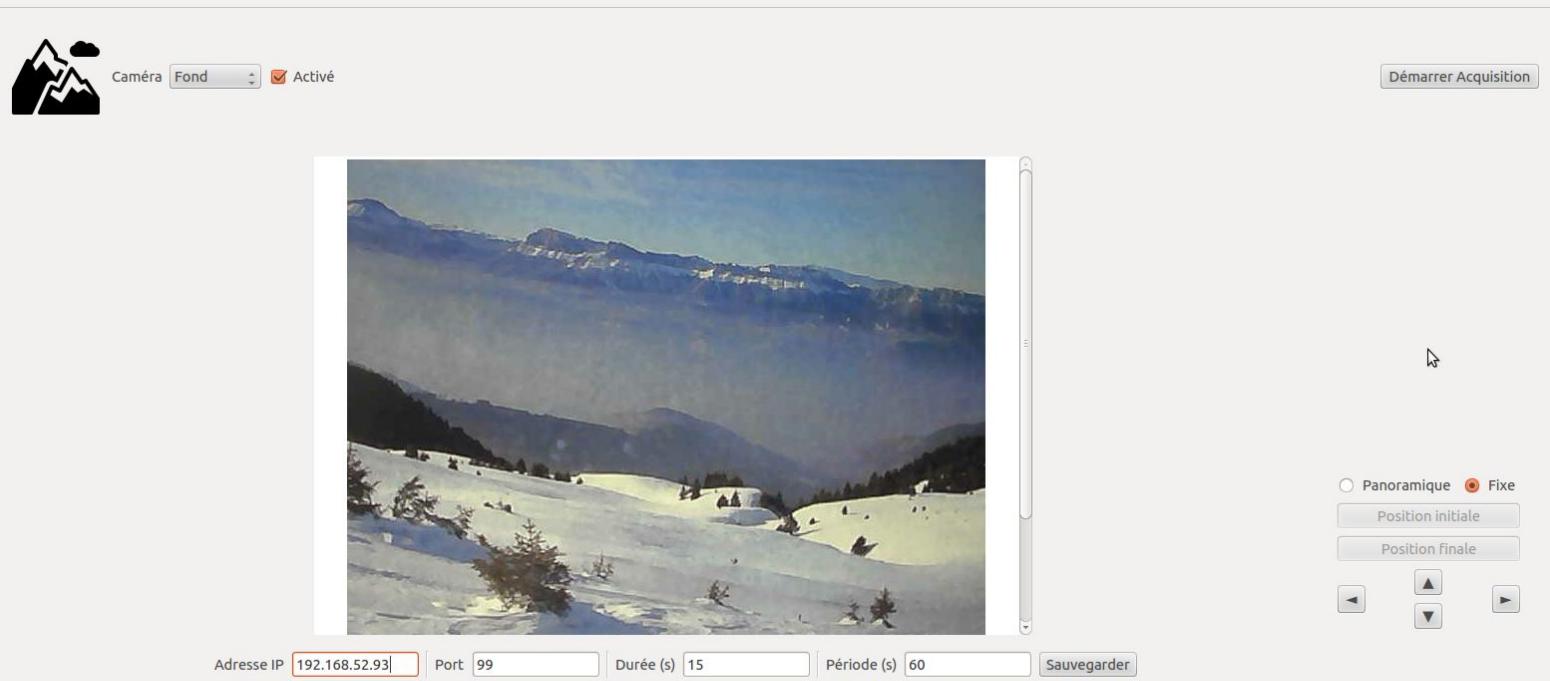
La structure **Parametres** comporte les paramètres caméras.

### 3. Diagramme de séquences

#### ▷ PARAMÉTRER LE SYSTÈME



Décrit la situation où l'administrateur décide de paramétriser la caméra d'un site. Les paramètres définis dans un fichier de configuration «.ini» sont alors lus.



Programmation avec Qt :

```
void IHMWismas::afficherParametresCamera() const
{
    #ifdef DEBUG
    qDebug() << Q_FUNC_INFO;
    #endif

    int numeroCamera = ui->listeConfigurationCameras->currentIndex();

    if(numeroCamera >=0 && numeroCamera < cameras.count())
    {
        Parametres parametres = cameras.at(numeroCamera)->getParametres();

        if(parametres.etat == "0")
        {
            ui->parametres->setDisabled(true);
            ui->acquisition->setDisabled(true);
            ui->visualisationParametrage->stop();
            ui->etat->setChecked(false);
            ui->etat->setText("Désactivé");
        }
        else
        {
            ui->parametres->setEnabled(true);
            ui->acquisition->setEnabled(true);
            ui->etat->setChecked(true);
            ui->etat->setText("Activé");
        }

        ui->champAdresseIP->setText(parametres.adresseIP);
        ui->champPort->setText(parametres.numeroPort);
        ui->champDuree->setText(parametres.duree);
        ui->champPeriode->setText(parametres.periode);

        if(parametres.typeDeplacement == "Panoramique")
        {
            ui->typeDeplacementPanoramique->setChecked(true);
            activerBoutonsCamera();

        }
        else if(parametres.typeDeplacement == "Fixe")
        {
            ui->typeDeplacementFixe->setChecked(true);
            desactiverBoutonsCamera();
        }
        else
        {
            ui->typeDeplacementFixe->setChecked(true);
            desactiverBoutonsCamera();
        }
    }
}
```

Si la caméra est désactivée, la méthode **setDisabled()** de la classe **QWidget** désactive le bouton « Démarrer Acquisition » et la modification des paramètres (adresse IP, déplacement...).

```
ui->parametres->setDisabled(true);
ui->acquisition->setDisabled(true);
ui->visualisationParametrage->stop();
ui->etat->setChecked(false);
ui->etat->setText("Désactivé");
```

Si la caméra est activée, la méthode **setEnabled()** de la classe **QWidget** et **QPushButton** active le bouton « Démarrer Acquisition » et la modification des paramètres (adresse IP, déplacement...).

```
ui->parametres->setEnabled(true);
ui->acquisition->setEnabled(true);
ui->etat->setChecked(true);
ui->etat->setText("Activé");
```

Si la caméra est paramétrée pour un déplacement panoramique, la méthode **setChecked()** de la classe **QRadioButton** enregistre le type de déplacement « Panoramique ».

```
if(parametres.typeDeplacement == "Panoramique")
{
    ui->typeDeplacementPanoramique->setChecked(true);
    activerBoutonsCamera();
}
else if(parametres.typeDeplacement == "Fixe")
{
    ui->typeDeplacementFixe->setChecked(true);
    desactiverBoutonsCamera();
}
else
{
    ui->typeDeplacementFixe->setChecked(true);
    desactiverBoutonsCamera();
}
```

La méthode **setText()** de la classe **QLineEdit** affecte les paramètres de la structure **Parametres** dans les champs de paramétrage.

```
ui->champAdresseIP->setText(parametres.adresseIP);
ui->champPort->setText(parametres.numeroPort);
ui->champDuree->setText(parametres.duree);
ui->champPeriode->setText(parametres.periode);
```

**sauvegarderParametres()** sauvegarde les paramètres modifiés par l'administrateur grâce à l'IHM.

```
void IHMWismas::sauvegarderParametres()
{
    int numeroCamera = ui->listeConfigurationCameras->currentIndex();
    QString identifiantCamera = "Camera" + QString::number(numeroCamera+1);
    Parametres parametres = cameras.at(numeroCamera)->getParametres();
    QSettings configuration(fichierINI, QSettings::IniFormat);

    ...

    configuration.setValue(identifiantCamera + "/adresse_IP", ui->champAdresseIP->text());
    configuration.setValue(identifiantCamera + "/numero_port", ui->champPort->text());
    configuration.setValue(identifiantCamera + "/duree", ui->champDuree->text());
    configuration.setValue(identifiantCamera + "/periode", ui->champPeriode->text());

    ...

    rafraichirFluxVideo(identifiantCamera, parametres);
    cameras.at(numeroCamera)->lireParametres();
}
```

La méthode **setValue()** de la classe **QSettings** écrit vers le fichier de configuration «.ini».

```
configuration.setValue(identifiantCamera + "/adresse_IP", ui->champAdresseIP->text());
configuration.setValue(identifiantCamera + "/numero_port", ui->champPort->text());
configuration.setValue(identifiantCamera + "/duree", ui->champDuree->text());
configuration.setValue(identifiantCamera + "/periode", ui->champPeriode->text());
```

**rafraichirFluxVideo()** rafraîchit le flux vidéo pour créer le nouveau **URL** en fonction des paramètres modifiés.

```
void IHMWismas::rafraichirFluxVideo(const QString identifiantCamera, Parametres &parametres)
{
    ...

    QSettings configuration(fichierINI, QSettings::IniFormat);
    parametres.adresseIP = configuration.value(identifiantCamera + "/adresse_IP", "").toString();
    parametres.numeroPort = configuration.value(identifiantCamera + "/numero_port", "").toString();

    QUrl URL("http://" + parametres.adresseIP + ":" + parametres.numeroPort + "/mobile.htm");
    URL.setUserName(parametres.identifiant);
    URL.setPassword(parametres.motDePasse);

    ui->visualisationParametrage->load(URL);
    ui->visualisationParametrage->update();
}
```

**URL** flux vidéo : "http://" + [adresseIP] + ":" + [numeroPort] + "/mobile.htm"

Pour accéder aux flux vidéo, un identifiant et un mot de passe sont nécessaires. La méthode **setUserName()** et **setPassword()** de la classe **QUrl** permettent l'authentification.

```
URL.setUserName(parametres.identifiant);  
URL.setPassword(parametres.motDePasse);
```

Le nouveau **URL** vidéo est alors chargé par la méthode **load()** de la classe **QWebView**.

```
ui->visualisationParametrage->load(URL);
```

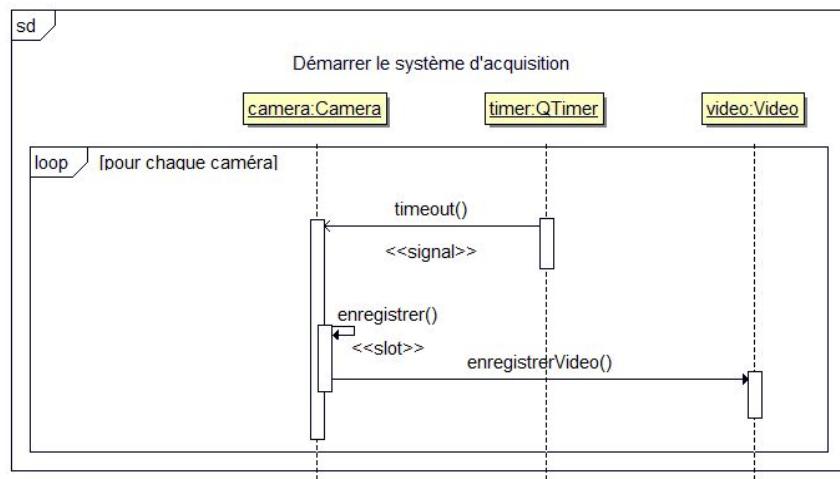
Le flux vidéo est ensuite mis à jour par la méthode **update()** de la classe **QWidget**.

```
ui->visualisationParametrage->update();
```

**lireParametres()** rafraîchit l'affichage des paramètres sur l'IHM.

```
void Camera::lireParametres()  
{  
    parametrage->lireParametres("Camera" + QString::number(numeroCamera));  
    parametres = parametrage->getParametres();  
}
```

## ▷ DÉMARRER LE SYSTÈME D'ACQUISITION



Décrit la situation où l'administrateur lance les enregistrements vidéo sur une durée définie par le fichier de configuration «.ini». Les enregistrements sont à nouveau lancés lorsqu'un timer initialisé à une période définie dans les paramètres de la caméra expire (**timeout()**).



Programmation avec Qt :

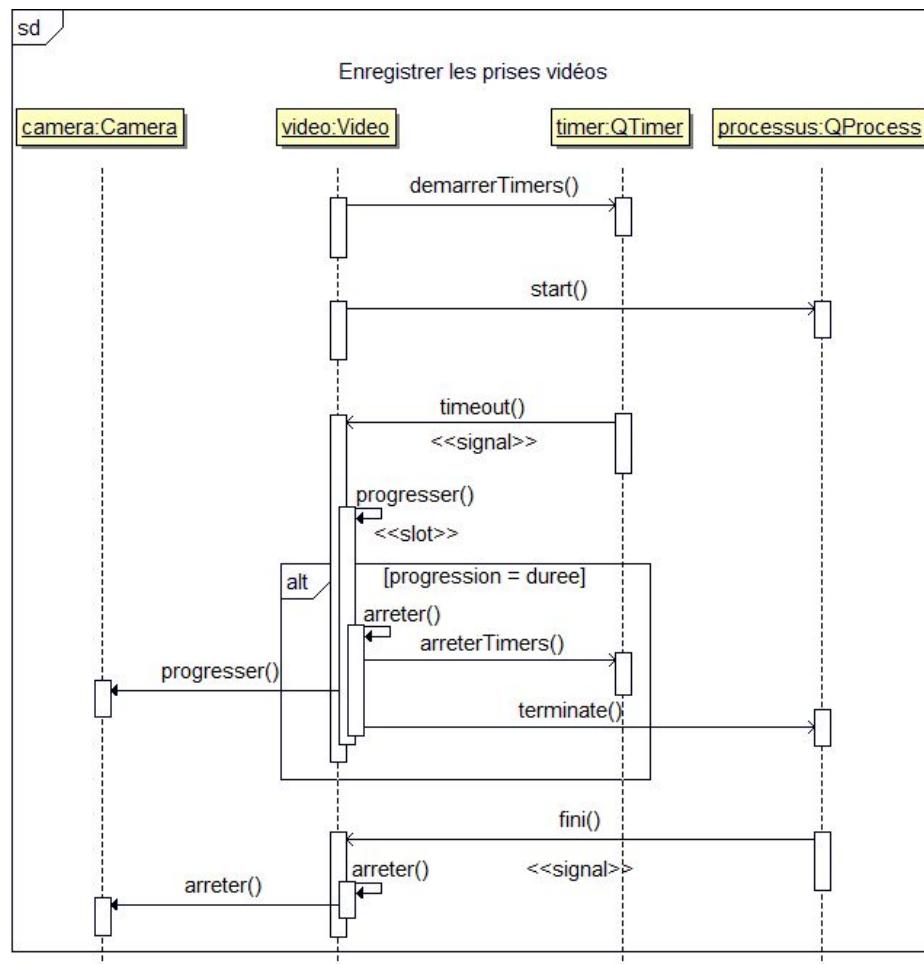
**enregistrer()** envoie un message vers les **logs**, **afficherMessage()** pour signaler la programmation d'un enregistrement, **enregistrerVideo()**.

```

void Camera::enregistrer() const
{
    if(parametres.etat == "1")
    {
        emit afficherMessage(QString::fromUtf8("Démarrage enregistrement caméra %1 (durée %2
s")).arg(parametres.nomSite).arg(parametres.duree));
        video->enregistrerVideo(numeroCamera, parametres.nomSite,
                               parametres.adresseIP, parametres.numeroPort,
                               parametres.identifiant, parametres.motDePasse,
                               parametres.duree.toInt());
    }
}

```

## ▷ ENREGISTRER UNE VIDÉO



Décrit la situation où l'administrateur lance l'acquisition vidéo. L'enregistrement



Paramétrier système

Programmation enregistrement caméra Alpin : toutes les 90 s  
Démarrage enregistrement caméra Alpin (durée 15 s)  
Programmation enregistrement caméra Fond : toutes les 60 s  
Démarrage enregistrement caméra Fond (durée 15 s)

Caméra Alpin : 7%      Caméra Fond : 7%      Caméra : 0%

Démarrer Arrêter

des caméras est lancé sur une durée définie dans le fichier de configuration «.ini» par les timers (**demarrerTimers()**). Lorsqu'une durée d'enregistrement

expire (**timeout()**), le processus donné par **QProcess** qui lance le script **cvlc.sh** avec les arguments (adresse IP, numéro de port...) est arrêté.

Programmation avec Qt :

**enregistrerVideo()** enregistre le flux vidéo de la caméra sélectionnée.

```
void Video::enregistrerVideo(const int numeroCamera, const QString nomSite,
                             const QString adresseIP, const QString numeroPort,
                             const QString identifiant, const QString motDePasse,
                             const int duree)
{
    ...

    if(duree != 0)
    {
        ...

        programme = "./cvlc.sh";
        if(motDePasse.length() == 0)
        {
            arguments << adresseIP << numeroPort << identifiant << nomFichier
                << QString::number(duree);
        }
        else
        {
            arguments << adresseIP << numeroPort << identifiant << motDePasse << nomFichier
                << QString::number(duree);
        }

        ...

        processus->start(programme, arguments);

        ...

        this->duree = duree;
        demarrerTimers(duree);
    }
}
```

Le script **cvlc.sh** est affecté à la variable **programme** puisqu'il permet l'enregistrement des vidéos avec le logiciel **VLC**. Les arguments sont affectés à la variable **arguments**. Elle permet de compléter le lien **URL** (adresse IP, numéro de port...) pour lancer la commande d'enregistrement.

```
programme = "./cvlc.sh";
if(motDePasse.length() == 0)
{
    arguments << adresseIP << numeroPort << identifiant << nomFichier
        << QString::number(duree);
}
else
{
    arguments << adresseIP << numeroPort << identifiant << motDePasse << nomFichier
        << QString::number(duree);
}
```

Le système donne alors un nouveau processus à l'application par la méthode **start()** de la classe **QProcess**.

```
processus->start(programme, arguments);
```

**demarrerTimers()** démarre les timers par la méthode **start()** de la classe **QTimer** pour une durée définie dans le fichier de configuration «.ini».

```
void Video::demarrerTimers(int duree)
{
    ...
    if(duree != 0)
    {
        timerProgression->start((duree*1000)/100);
        timerEnregistrement->start(duree*1000);
        connect(timerProgression, SIGNAL(timeout()), this, SLOT(progresser()));
        connect(timerEnregistrement, SIGNAL(timeout()), this, SLOT(arreter()));

        ...
    }
}
```

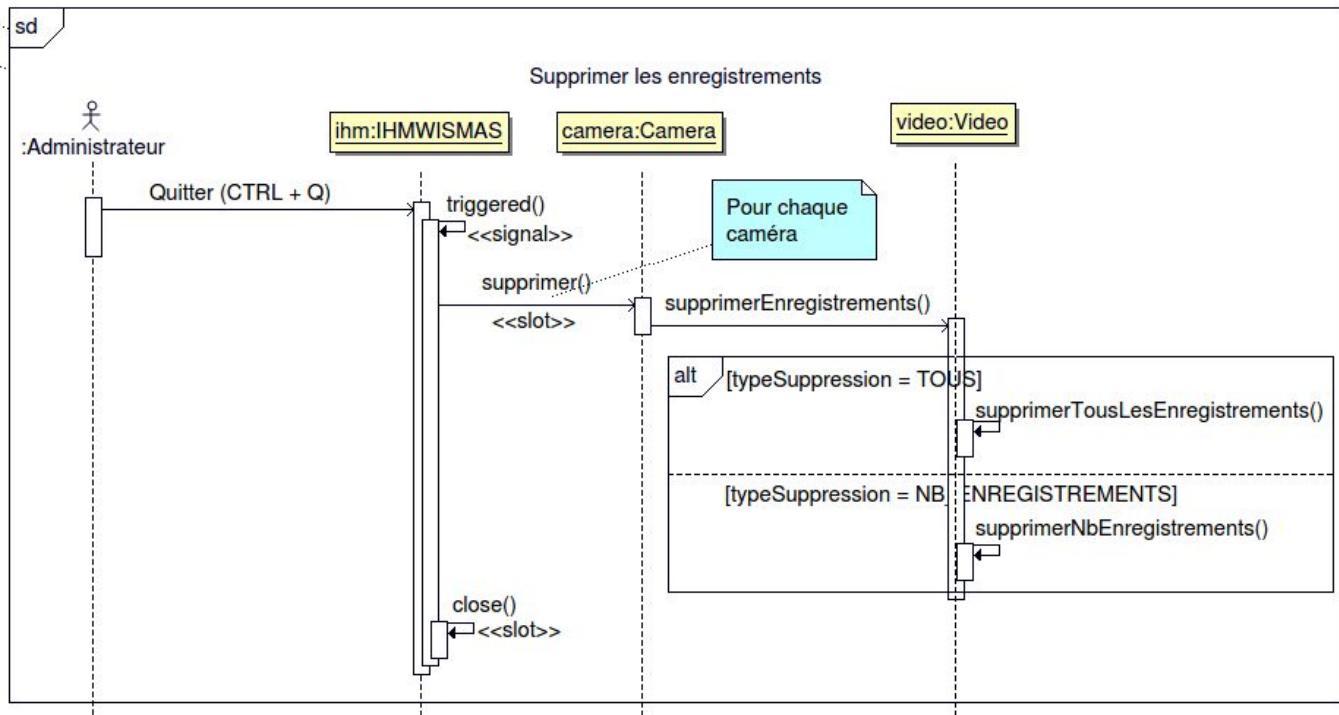
**timerProgression** qui indique l'état de l'enregistrement en pourcentage.  
**timerEnregistrement** pour le temps d'enregistrement de la vidéo.

```
timerProgression->start((duree*1000)/100);
timerEnregistrement->start(duree*1000);
```

Lorsque les timers expirent (**timeout()**) **timerEnregistrement** envoie un signal à la fonction **arreter()** pour arrêter les enregistrements et **timerProgression** met à jour les barres de progression.

```
connect(timerProgression, SIGNAL(timeout()), this, SLOT(progresser()));
connect(timerEnregistrement, SIGNAL(timeout()), this, SLOT(arreter));
```

## ▷ SUPPRIMER LES ENREGISTREMENTS



Lorsque l'application est fermée par le raccourci clavier **Ctrl + Q**, les enregistrements sont traités par la méthode **supprimerEnregistrements()**.

Si le type de suppression est défini sur :

- « **Tous** » lance la méthode **supprimerTousLesEnregistrements()**.
- « **Nb\_enregistrements** » lance la méthode **SupprimerNbEnregistrements()**.
- « **Aucun** » n'exécute aucune instruction.

Programmation avec Qt :

**supprimerEnregistrements()** supprime les enregistrements selon le niveau de suppression défini dans le fichier de configuration «.ini».

```
void Video::supprimerEnregistrements(const QString nomSite)
{
    QSettings configuration(fichierINI, QSettings::IniFormat);
    unsigned int nbEnregistrementsAConserver = configuration.value("nb_videos", "1").toInt();

    QStringList filtre;
    QString extension = ".mp4";
    QDir repertoire(cheminVideo);
    QString typeSuppression = configuration.value("niveau_suppression", "Aucun").toString();
    filtre << "*" + extension;

    if(repertoire.exists())
    {
        if(typeSuppression == "Tous") // Tous les fichiers seront supprimés
        {
            supprimerTousLesEnregistrements(repertoire);
        }
        else if(typeSuppression == "Aucun") // Aucun fichiers supprimés
        {
            //Ne rien faire
        }
        else if(typeSuppression == "Nb_enregistrements")
        {
            supprimerNbEnregistrements(repertoire, nbEnregistrementsAConserver, nomSite);
        }
    }
    else
    {
        repertoire.mkdir(cheminVideo);
    }
}
```

Le niveau de suppression défini dans le fichier de configuration «.ini» est affecté à la variable **typeSuppression** par la méthode **value()** de la classe **QSettings**.

```
QString typeSuppression = configuration.value("niveau_suppression", "Aucun").toString();
```

La méthode **supprimerTousLesEnregistrements()** traite le niveau de suppression « **Tous** » et **supprimerNbEnregistrements()** traite le niveau de suppression « **Nb\_enregistrements** »

```
if(typeSuppression == "Tous")
{
    supprimerTousLesEnregistrements(repertoire);
}
else if(typeSuppression == "Aucun")
{
    //Ne rien faire
}
else if(typeSuppression == "Nb_enregistrements")
{
    supprimerNbEnregistrements(repertoire, nbEnregistrementsAConserver, nomSite);
}
```

Un niveau de suppression défini sur « **Tous** » supprime tous les enregistrements par la méthode **supprimerTousLesEnregistrements()**.

```
void Video::supprimerTousLesEnregistrements(QDir& repertoire)
{
    QStringList filtre;
    QString extension = ".mp4";
    filtre << "*" + extension;

    foreach(QString enregistrement, repertoire.entryList(filtre, QDir::NoFilter, QDir::DirsLast))
    {
        repertoire.remove(enregistrement);

        ...
    }
}
```

Les noms de fichiers sont récupérés dans la variable **repertoire** par la méthode **entryList()** de la classe **QDir**.

Les enregistrements les plus anciens sont donc supprimés par la méthode **remove()** de la classe **QDir**.

```
foreach(QString enregistrement, repertoire.entryList(filtre, QDir::NoFilter, QDir::DirsLast))
{
    repertoire.remove(enregistrement);

    ...
}
```

Un niveau de suppression défini sur « **Nb\_enregistrements** » supprime les enregistrements les plus anciens et conserve le plus récent par la méthode **supprimerNbEnregistrements()**.

```
void Video::supprimerNbEnregistrements(QDir& repertoire,
                                         unsigned int nbEnregistrementsAConserver,
                                         const QString nomSite)
{
    QStringList filtre;
    QString extension = ".mp4";
    filtre << "*" + extension;

    if(repertoire.count() > nbEnregistrementsAConserver)
    {
        foreach(QString enregistrement, repertoire.entryList(filtre))
        {
            if(!(enregistrement == QString::number(getNumeroVideo()) + "_" + nomSite + extension))
                repertoire.remove(enregistrement);

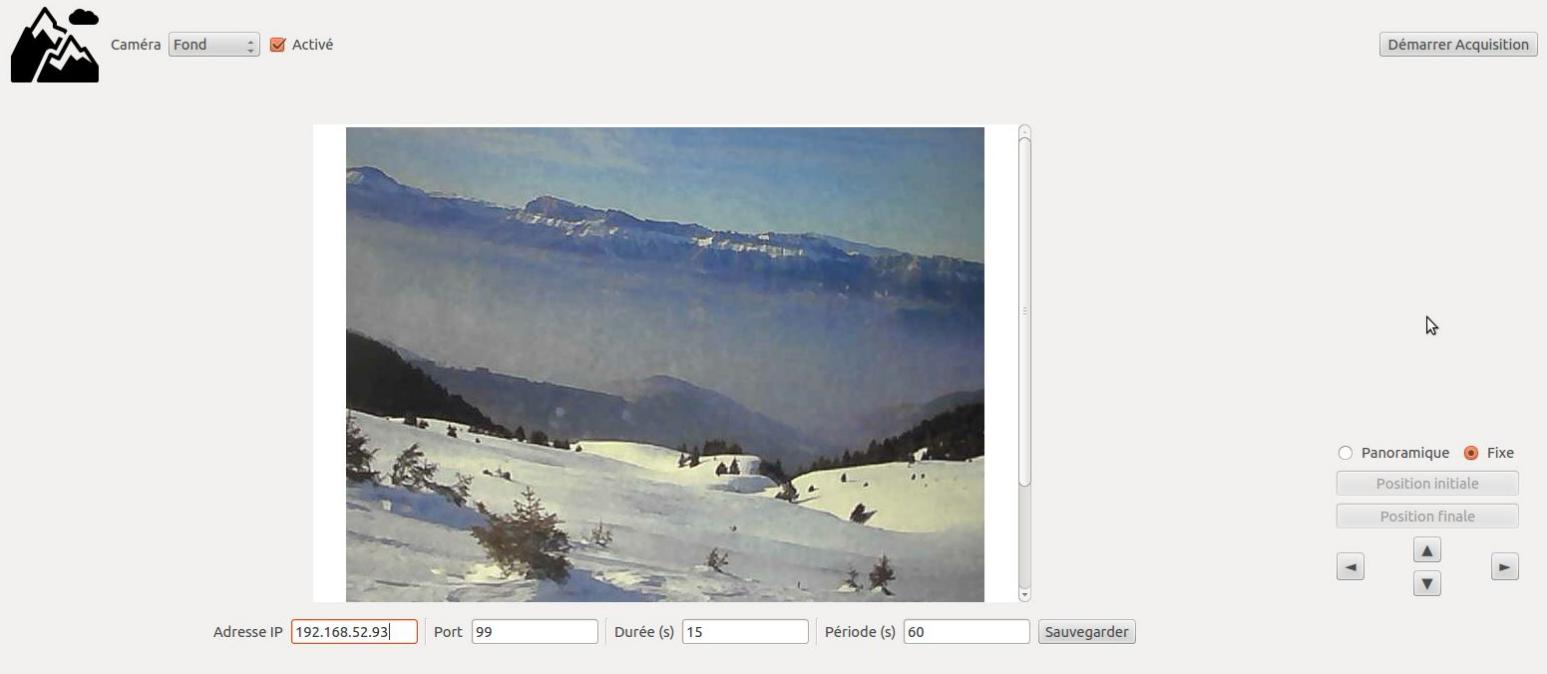
            ...
        }
    }
}
```

La méthode **getNumeroVideo()** permet de connaître l'enregistrement le plus récent.

```
if(!(enregistrement == QString::number(getNumeroVideo()) + "_" + nomSite + extension))
    repertoire.remove(enregistrement);
```

Un niveau de suppression défini sur « **Aucun** » conserve tous les enregistrements donc aucune instruction est exécutée.

### III. INTERFACE HOMME MACHINE



Au lancement de l'application, la fenêtre comporte une vue de la caméra sélectionnée et ses paramètres.

Une liste déroulante permet à l'administrateur de sélectionner une caméra.

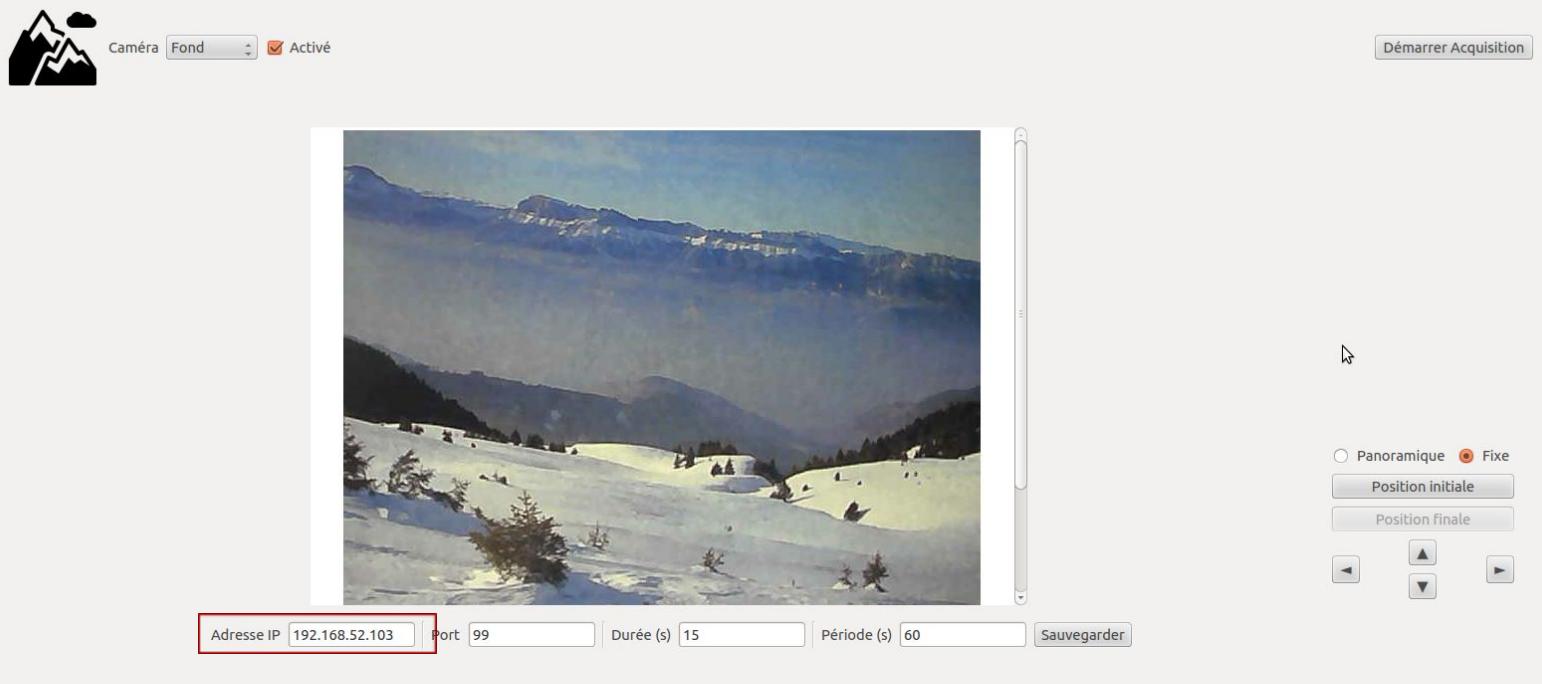
Les paramètres d'une caméra se caractérisent par une adresse IP, un numéro de port, une durée, une période et son type de déplacement pour l'acquisition vidéo.

Il doit pouvoir demander une acquisition vidéo sur une durée, champs **Durée**. Lorsque la période d'attente est écoulée, champs **Période**, l'enregistrement vidéo est à nouveau lancé.

Le déplacement de la caméra lors d'une séquence vidéo est modifiable à l'aide des types de déplacement, boutons radios **Panoramique** (d'un point à l'autre) et **Fixe** (ne rien faire).

Les champs **Adresse IP** et **Port** doivent correspondre à l'adresse IP et au numéro de port de la caméra de surveillance IP sélectionnée.

Les paramètres modifiés peuvent être sauvegardés (bouton « Sauvegarder »). Le fichier de configuration « .ini » sera donc mis à jour.



```
[Camera2]
etat=1
nom=Fond
chemin_video_=/videos/fond/
adresse_IP=192.168.52.103
numero_port=99
identifiant=admin
mot_de_passe=
type_deplacement=Fixe
duree=15
periode=60
resolution=480
```

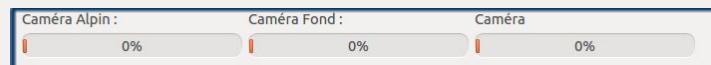
Modifier l'adresse IP de la caméra du site Fond provoque donc l'arrêt du flux vidéo car la nouvelle adresse IP « **192.168.52.103** » ne correspond pas à son adresse IP « **192.168.52.93** ». De plus, le fichier de configuration « .ini » est bien modifié puisque nous retrouvons l'adresse IP du site Fond modifié.

Il est alors possible de paramétriser les caméras depuis l'IHM ou depuis le fichier de configuration « .ini » (nécessite un redémarrage de l'application).



Paramétrer système

Programmation enregistrement caméra Alpin : toutes les 90 s  
Démarrage enregistrement caméra Alpin (durée 15 s)  
Programmation enregistrement caméra Fond : toutes les 60 s  
Démarrage enregistrement caméra Fond (durée 15 s)



Démarrer Arrêter

A l'acquisition vidéo, bouton « Démarrer acquisition », en cours d'enregistrement, il lui est impossible de paramétriser les caméras.

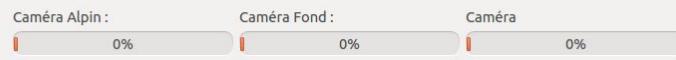
Un affichage des **logs** et des **barres de progression** permet à l'administrateur d'avoir un suivi sur les enregistrements vidéo.

- I.1 : Signale que les caméras sont prêtes à l'enregistrement pour chaque période
- I.2 : Indique que le flux vidéo est actuellement en enregistrement sur une durée
- Indique le taux d'enregistrement de la vidéo



Paramétrer système

Programmation enregistrement caméra Alpin : toutes les 90 s  
Démarrage enregistrement caméra Alpin (durée 15 s)  
Programmation enregistrement caméra Fond : toutes les 60 s  
Démarrage enregistrement caméra Fond (durée 15 s)  
Fin de la programmation des enregistrements caméra Alpin  
Fin de la programmation des enregistrements caméra Fond



Démarrer Arrêter

Lorsque l'acquisition est arrêtée manuellement (bouton « Arrêter »), l'administrateur peut à nouveau paramétriser les caméras ou relancer une acquisition (bouton « Démarrer »).

## IV. TEST DE VALIDATION

MODULE D'ACQUISITION VIDÉO	Démarches	Résultat	Oui	Non
Afficher un flux vidéo	- Au démarrage de l'application, une vue de la caméra sélectionnée	Le module reçoit un flux vidéo	X	
Enregistrer une vidéo	- Démarrer l'acquisition vidéo - Arrêter la capture en cours de la vidéo <b>OU</b> - Attendre que l'acquisition vidéo soit terminée (durée expirée)	L'enregistrement doit se situer sur le serveur NFS	X	
Déplacer une caméra	- Sélectionner le type « <i>Fixe</i> » - Déplacer de gauche à droite et de haut en bas <b>OU</b> - Sélectionner le type « <i>Panoramique</i> » - Définir la position initiale et la position finale	La caméra doit effectuer les mouvements envoyés par l'administrateur	X	
Paramétriser le système	- Modifier les paramètres d'une caméra - Valider les paramètres en appuyant sur « <i>Sauvegarder</i> »	Les paramètres modifiés sont écrits vers le fichier de configuration « <i>.ini</i> »	X	
Supprimer un enregistrement	- Quitter par le raccourci clavier <i>Ctrl + Q</i>	Selon le niveau de suppression défini dans le fichier de configuration « <i>.ini</i> » : - <b>Tout supprimer</b> - <b>Aucune suppression</b> - <b>Conserver le plus récent</b>	X	
Remarques				

## V. GLOSSAIRE

- **API (Application Programming Interface)** : Bibliothèque de fonctions destinées à être réutilisée dans diverses applications afin de réaliser des traitements de données
- **WISMAS (Weather Information System Multi Activity Station)**: Station météo de moyenne montagne
- **Fichier INI** : Fichier de configuration qui permet de modifier le comportement d'un logiciel afin de l'adapter à un environnement et de lui faire réaliser des fonctionnalités particulières.
- **Qt** : API orientée objet et développée en C++ par Qt Development Frameworks. Qt offre des composants d'interface graphique, d'accès aux données, de connexions réseaux... grâce à ses nombreuses bibliothèques logicielles multiplateformes.
- **SVN/RiouxSVN** : Outil de gestion de configuration qui permet de centraliser les sources afin de les rendre disponibles à l'ensemble des développeurs et acteurs d'un projet et d'assurer un suivi des fichiers.
- **Doxxygen** : Générateur de documentation capable de produire une documentation logicielle à partir du code source d'un programme.
- **Bouml** : Logiciel de création de diagrammes UML
- **NFS (Network File System)** : Système de fichiers en réseau et protocole réseau de partage d'un système de fichiers. Il permet de partager des fichiers entre systèmes UNIX.
- **WLAN (Wireless Local Area Network)** : Désigne un type de réseau local sans fil. Il s'agit d'un réseau capable d'établir une connexion entre plusieurs systèmes à distance.
- **URL (Uniform Resource Locator)** : Désigne une ressource présente sur le web (page web, vidéo, son, image...).
- **VLC (VideoLAN Client)** : Lecteur multimédia capable de lire des flux réseaux.