Hes·so VALAIS WALLIS
Haute Ecole Spécialisée de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences Western Switzerland

# Filière Systèmes industriels

## Orientation Infotronics

# Diplôme 2009

*Johann Seydoux*

*Plateforme de mesure embarquée avec publication web*

Professeur Dominique Gabioud

Expert Frédéric Revaz

Sion, le 6 juillet 2009

**HES-SO Valais**

| SI | TV | EE | IG | EST |
|---|---|---|---|---|
| X | X | | | |

# Données du travail de diplôme
## *Daten der Diplomarbeit*

FO 1.2.02.07.DB
mam/30/06/2008

| | | |
|---|---|---|
| ☒ FSI<br>☐ FTV | Année académique / *Studienjahr*<br>**2008/09** | No TD / *Nr. DA*<br>**it/2009/25** |
| Mandant / *Auftraggeber*<br>☒ HES—SO Valais<br>☐ Industrie<br>☐ Etablissement partenaire | Etudiant / *Student*<br>**Johann Seydoux** | Lieu d'exécution / *Ausführungsort*<br>☒ HES—SO Valais<br>☐ Industrie<br>☐ Etablissement partenaire |
| Professeur / *Dozent*<br>**Dominique Gabioud** | Expert / *Experte* *(données complètes)* | |
| Travail confidentiel / *vertrauliche Arbeit*<br>☐ oui / *ja* ☒ non / *nein* | | |

Titre / *Titel*

**Plateforme de mesure embarquée avec publication web**

Description et Objectifs / *Beschreibung und Ziele*

L'objectif du projet est de concevoir et de développer un environnement de publication web pour des capteurs embarqués.

Les éléments suivants sont déjà disponibles :

- Matériel : carte à processeur avec capteurs et interface réseau Ethernet (ARM EBS 3)
- Système d'exploitation et middlleware : Linux, système de gestion de bases de données relationnelles SQLite, pilote pour l'acquisition des données des capteurs.

L'environnement de publication web doit répondre aux exigences suivantes : performance (code compilé et non interprété), souplesse (paramétrisation uniquement par la base de données), publication des données pour des personnes (HTML) et pour des machines (RDF).

Les objectifs concrets à atteindre sont les suivants :

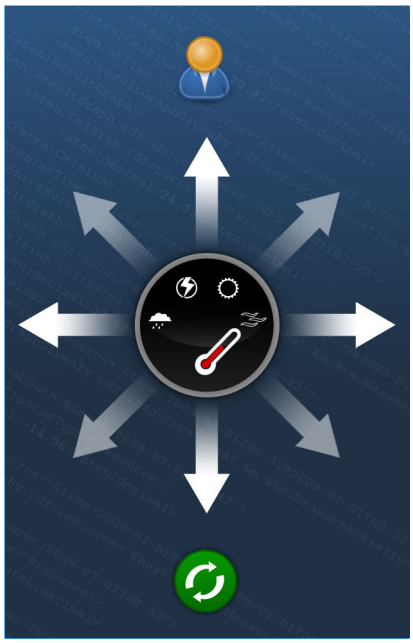
— Conception d'une structure de navigation HTML et RDF
— Design de documents HTML et RDF génériques
— Génération à la volée de documents HTML et RDF
— Réalisation d'un démonstrateur.

| Signature ou visa / *Unterschrift oder Visum* | Délais / *Termine* |
|---|---|
| Resp. de la filière<br>*Leiter des Studieng.:* ..... | Attribution du thème / *Ausgabe des Auftrags:*<br>18.02.2009 |
| Etudiant / *Student*: ..... | Remise du rapport / *Abgabe des Schlussberichts:*<br>06.07.2009, 12:00 |
| | Exposition publique / *Ausstellung Diplomarbeiten:*<br>04.09.2009 |
| | Défense orale / *Mündliche Verfechtung:*<br>Semaine / *Woche* 35 |

Rapport reçu le / *Schlussbericht erhalten am* ..... Visa du secrétariat / *Visum des Sekretariats* .....

Haute Ecole Spécialisée de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences Western Switzerland

# Plateforme de mesure embarquée avec publication web



Diplômant Johann Seydoux

## Objectif du projet

L'objectif de ce projet est de concevoir une plateforme permettant de publier des mesures physiques sur le Web. Ces mesures peuvent être notamment utilisées pour l'analyse environnementale.

## Méthodes | Expériences | Résultats

Les mesures publiées doivent être interprétables autant par un client humain derrière un navigateur que par un client machine. La publication ne peut donc pas se faire uniquement en HTML. En effet, un client humain est capable d'interpréter les données contenues dans un document HTML contrairement à un client machine qui nécessite que les données soient définies explicitement.

Les nouvelles technologies du Web rendent possible cette définition explicite de l'information grâce au langage RDF (*Resource Description Framework*). Ce dernier a pour but de lier explicitement toute l'information contenue dans le Web.

Le système réalisé permet de publier des données en HTML et en RDF à travers un serveur Web. Ces données décrivent toute l'information relative à la plateforme et aux mesures qu'elle effectue.

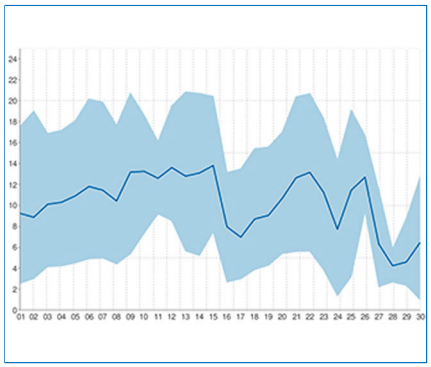
.

Travail de diplôme
| édition 2009 |

Filière
*Systèmes industriels*

Domaine d'application
*Infotronics*

Professeur responsable
*M. Dominique Gabioud*
*dominique.gabioud@hevs.ch*



Représentation HTML des mesures à l'aide d'un graphique

```
<sten:ResultGraph rdf:about="&host;&m;/2
      <sten:during rdf:resource="&time;2
      <sten:Celsius rdf:parseType="Resou
            <sten:decimal>-14.958200</st
      </sten:Celsius>
      <sten:resultGraphGivenBy rdf:resou
</sten:ResultGraph>
<sten:ResultGraph rdf:about="&host;&m;/2
      <sten:during rdf:resource="&time;2
      <sten:Celsius rdf:parseType="Resou
            <sten:decimal>12.887803</ste
      </sten:Celsius>
      <sten:resultGraphGivenBy rdf:resou
</sten:ResultGraph>
<rdf:Description rdf:about="">
```

Représentation RDF des mesures dans un document sérialisé avec XML

HES-SO Valais
Route du Rawyl 47
1950 Sion

Tél. 027 606 85 11
URL www.hevs.ch

# Table des matières


**1 Introduction** **8**
1.1 Objectifs . . . 9

**2 Web sémantique** **10**
2.1 Définition . . . 10
2.2 RDF . . . 10
2.3 Ontologies S-TEN et EMEP . . . 13
2.4 Structure de la plateforme . . . 13
2.4.1 Ressources . . . 13
2.4.2 Parcours de la structure . . . 15

**3 Architecture** **16**
3.1 Navigation . . . 16
3.1.1 Documents RDF . . . 17
3.1.2 Documents HTML . . . 18
3.2 Plateforme . . . 20

**4 Environnement de la plateforme** **21**
4.1 Serveur web . . . 21
4.2 Système de gestion de base de données . . . 21
4.3 Langage de programmation . . . 21
4.4 Bibliothèque logicielle Qt . . . 21
4.5 Interfaces CGI et FastCGI . . . 21
4.6 CGI . . . 22
4.7 FastCGI . . . 22
4.8 Choix de l'interface . . . 23
4.8.1 Environnement des tests . . . 23
4.8.2 Tests . . . 23
4.8.3 Résultats . . . 23

**5 Conception de la base de données** **25**
5.1 Indroduction . . . 25
5.2 Configuration de la plateforme . . . 26
5.3 Mesures . . . 26
5.4 Configuration des graphiques . . . 29
5.5 Configuration des processus d'acquisition . . . 29

**6 Application de publication** **30**
6.1 HTTP request parsing . . . 30
6.2 Content negociation . . . 31
6.3 URI parsing . . . 32
6.4 RDF/HTML generation . . . 33
6.4.1 Mécanisme de génération de fichiers . . . 33
6.4.2 Liste des fichiers et leurs fonctions . . . 34
6.4.3 Liste des fonctions de traîtement de l'information . . . 36
6.5 Error handler . . . 41

**7 Classe d'acquisition** **43**
7.1 Utilisation . . . 45

**8 Optimisation** **46**
8.1 Utilisation des ressources . . . 46
8.1.1 Passage de paramètres par référence . . . 46
8.1.2 Optimisation des documents RDF . . . 46
8.2 Performances . . . 46
8.2.1 Base de données . . . 46

**9 Tests et résultats** **48**
9.1 Tests de limitations . . . 48
9.1.1 Emprunte mémoire . . . 48
9.1.2 Restitution de la mémoire . . . 49
9.2 Tests de performance . . . 50
9.3 Tests fonctionnels . . . 50
9.3.1 Test de la publication des données . . . 50
9.3.2 Test de validité des documents RDF générés . . . 50
9.3.3 Test de la validité des URIs . . . 51
9.3.4 Test de la classe d'acquisition . . . 51

**10 Limites de la plateforme** **52**
10.1 Flexibilité . . . 52
10.1.1 Résolution des capteurs . . . 52
10.1.2 Calculs statistiques . . . 52
10.2 Utilisation des ressources . . . 52

**11 Conclusion** **53**
11.1 Tâches à effectuer . . . 53

**A Fichiers de configuration de Lighttpd** **56**
A.1 lighttpd.conf . . . 56
A.2 mod_cgi.conf . . . 57
A.3 mod_fastcgi.conf . . . 57

**B Code : application hello** **59**

**C Code : application dbread** **60**

**D Diagramme de flux : application de publication** **62**

**E Code : application de publication** **63**
E.1 Header file . . . 63
E.2 main . . . 63

**F Code : classe d'acquisition** **95**
F.1 Header file . . . 95
F.2 Source file . . . 96

**G Code : adaptation de l'application de publication pour le test mémoire** **99**

**H Code : application de test de performance** **100**
H.1 Header file . . . . . 100
H.2 Source file . . . . . 101
H.3 Main . . . . . 102

**I Code : application de simulation d'acquisition** **103**
I.1 Header file . . . . . 103
I.2 Source file . . . . . 104
I.3 Main . . . . . 105

**J Documents RDF retournés** **106**

# Table des figures


1.1 Réseau de publication de mesures . . . . . 8

2.1 Équipement avec plusieurs capteurs . . . . . 10
2.2 Communication entre deux systèmes . . . . . 11
2.3 Représentation graphique d'un triplet . . . . . 12
2.4 Représentation graphique de l'équipement *deviceX* . . . . . 12
2.5 Représentation graphique des ressources de la plateforme . . . . . 14
2.6 Représentation graphique des ressources retournant des valeurs . . . . . 14
2.7 Chemin depuis la ressource *device* jusqu'à la valeur de la mesure . . . . . 15

3.1 Mécanisme de négociation de contenu avec hachage . . . . . 16
3.2 Vue document de la représentation graphique des ressources . . . . . 17
3.3 Graphique de représentation des statistiques . . . . . 18
3.4 Obtention des valeurs statistiques à l'aide du graphique RDF . . . . . 19
3.5 Histogramme statistique du nombres de mesures dans des intervales . . . . . 19
3.6 Architecture de la plateforme . . . . . 20

4.1 Relation entre le serveur web et l'interface CGI . . . . . 22
4.2 Relation entre le serveur web et l'interface FastCGI . . . . . 22
4.3 Résultat des tests de performances entre hello-cgi et hello-fcgi . . . . . 23
4.4 Résultat des tests de performances entre dbread-cgi et dbread-fcgi . . . . . 24

5.1 Organisation de la base de données . . . . . 25
5.2 Diagramme d'entité-relation de la partie configuration de la plateforme . . . . 26
5.3 Modèle relationnel de la partie configuration de la plateforme . . . . . 26
5.4 Diagramme d'entité-relation de la partie mesures . . . . . 27
5.5 Diagramme relationnel de la partie mesures . . . . . 28
5.6 Diagramme d'entité-relation de la partie de configuration des graphiques . . . 29

6.1 Phases de l'application de publication . . . . . 30
6.2 Diagramme de flux de la négociation de contenu . . . . . 31
6.3 Décomposition de l'URI en niveaux . . . . . 32
6.4 Procédure de génération de fichiers . . . . . 33
6.5 Simulation d'un moteur de modèles basique . . . . . 34

7.1 Processus d'acquisition . . . . . 43
7.2 Diagramme de séquence d'acquisition . . . . . 44
7.3 Héritage de la classe *PollDaemon* . . . . . 45

8.1 Recherche dans une table non indexée et dans une table indexée . . . . . 47

# CHAPITRE 1

# Introduction

Les technologies du Web actuelles offrent de nouvelles possibilités en terme d'échange de données entre applications. Ces nouvelles technologies sont profitables notamment dans le domaine de la publication de mesures pour le suivi environnemental.

Dans la publication de mesures sur le Web, il est nécessaire que le client, une personne physique accédant à l'information à travers un navigateur Web ou une application, puisse obtenir des informations non seulement sur la valeur de la mesure, mais également sur quel capteur a effectué la mesure, quand elle a été faite, quelle est son unité et où elle a été réalisée.

Une plateforme de publication a par définition une présence Web. Elle désigne tout le système permettant d'effectuer cette publication et peut se trouver à n'importe quel endroit dans le monde et publier ses mesures à travers un serveur Web. Elle peut être isolée géographiquement et avoir des capteurs intégrés, ou être composée de plusieurs capteurs déportés dans un réseau propriétaire comme le montre la figure 1.1. La présence actuelle du Web un peu partout permet à tout un chacun de posséder une telle plateforme avec des capteurs intégrés ce qui rend favorable l'utilisation de systèmes à ressources limitées. En effet, les systèmes embarqués sont peu coûteux et consomment peu d'énergie. De plus, ils peuvent être facilement mis en place à proximité des capteurs auxquels ils doivent être connectés grâce à leur petite taille.

Il existe des base de données *registry* répertoriant des plateformes de mesures. Elles offrent la possibilité aux plateformes de s'enregistrer et d'être accessibles plus facilement par un client.

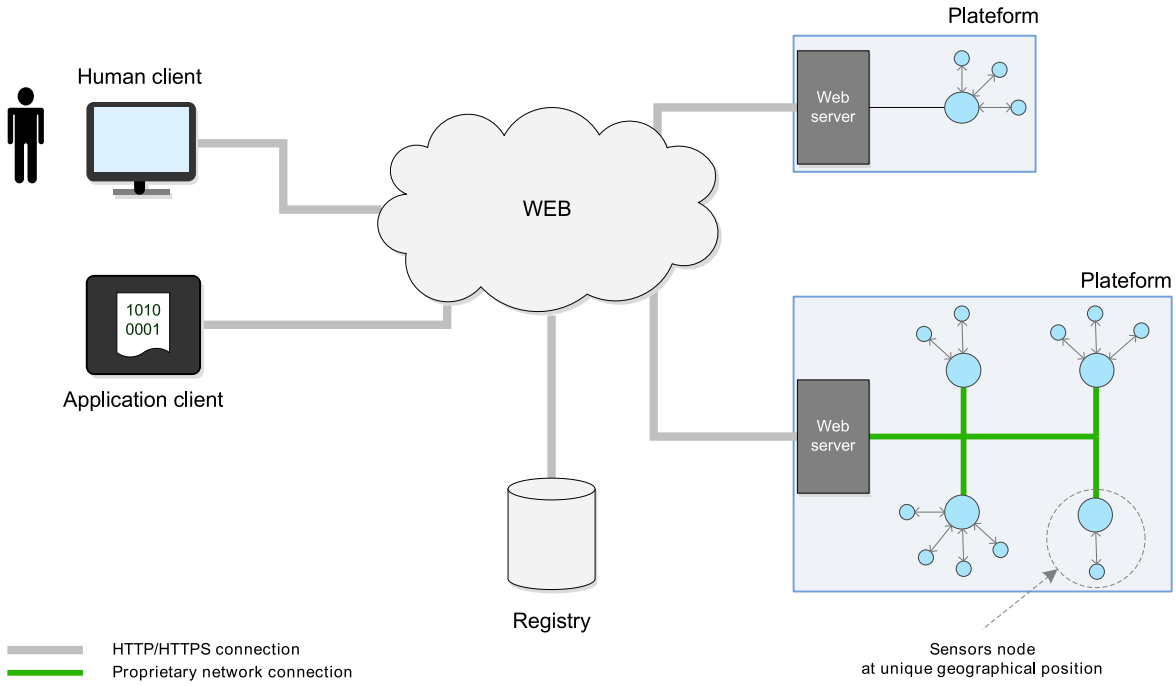



FIGURE 1.1 – Réseau de publication de mesures

Cette publication ne peut pas se faire uniquement en HTML. En effet, une personne physique est capable d'interpréter les données contenues dans un document HTML, tandis qu'une application requiert des données définies explicitement afin de pouvoir les traiter. Le langage HTML est incapable de le faire contrairement au langage XML. Cependant, celui-ci suppose que l'application connaisse le vocabulaire utilisé par le fournisseur de données et ce qu'il représente. Ceci n'est pas concevable dans le cas d'un client machine. Le RDF (*Resource Description Framework*) permet de définir explicitement les données et permet donc à une application client de pouvoir les interpréter et les traiter. Il est décrit dans la section 2.2 (page 10).

## 1.1 Objectifs

L'objectif concret de ce projet est de concevoir la partie de la plateforme publiant les données sur le Web. Pour ce faire, les tâches suivantes doivent être réalisées :

- Conception d'une structure de navigation HTML et RDF
- Design de documents HTML et RDF génériques
- Génération à la volée de documents HTML et RDF
- Réalisation d'un démonstrateur

La publication de mesures doit être évolutive, ce qui signifie qu'il faut que l'on puisse rajouter des capteurs en ayant à faire le moins de modifications possibles dans la structure même de la plateforme.

Le développement d'une classe ordonnant l'acquisition de mesures et leur insertion dans la plateforme fait également partie des objectifs.

CHAPITRE 2

# Web sémantique

## 2.1 Définition

Le Web sémantique est un ensemble de technologies visant à rendre l'information interprétable par des machines en définissant sa signification de manière explicite. Il se base sur l'utilisation de métadonnées - données définissant d'autres données - qui peuvent décrire des ressources (*Things*) telles que des objets réels ou abstraits. Un des principaux modèles de données propres au Web sémantique est le RDF.

## 2.2 RDF

Le RDF est un modèle de description de ressources qui a pour but de lier toute l'information du *World Wide Web*. Dans le domaine du Web sémantique, une ressource peut être toute chose ou entité identifiable par une courte chaîne de caractères unique appelée URI (*Uniform Resource Identifier*). Par exemple, un capteur pourrait être identifié par l'URI suivante où *prefix* représente une chaîne de caractères unique :

```
prefix:humiditySensor123
```

Un document structuré en RDF est un ensemble de triplets (association) :

{ sujet , prédicat , objet }

Le sujet est la ressource à décrire, le prédicat, une propriété applicable à cette ressource et la reliant à l'objet, et l'objet, une donnée ou une autre ressource. Le prédicat est lui-même une ressource. Le RDF est construit sur un principe de classes ressemblant à celui de l'orienté objet (classes, sous-classes, attributs).

La figure 2.1 présente un équipement avec un capteur de température et un capteur d'humidité.

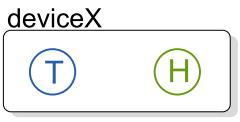



FIGURE 2.1 – Équipement avec plusieurs capteurs

Il est possible de définir cet équipement en utilisant des triplets. Chacune des ressources suivantes est en réalité identifiée par une URI unique. Ces URIs ont été simplifiées pour une question de lisibilité.

| *Sujet* | *Prédicat* | *Objet* |
|---|---|---|
| deviceX | is a | Device |
| | hasDescription | "This is a device with sensors" |
| | hasSensor | T |
| | hasSensor | H |
| T | is a | Sensor |
| | hasDescription | "This is a temperature sensor" |
| H | is a | Sensor |
| | hasDescription | "This is a humidity sensor" |

Dans une vue orientée objet, *deviceX* est une instance de la classe *Device* et a trois attributs dont une chaîne de caractères (*label*) le décrivant, et deux instances *T* et *H* de la classe *Sensor*. Pour que *Device* et *Sensor* soient compris comme des classes et *hasDescription* et *hasSensor* comme des propriétés, le schéma RDF impose leur déclaration explicite à l'aide de sa syntaxe :

| *Sujet* | *Prédicat* | *Objet* |
|---|---|---|
| Device | is a | Class |
| Sensor | is a | Class |
| hasDescription | is a | Property |
| hasSensor | is a | Property |

En effet, un vocabulaire RDF standard existe afin de garantir l'intéropérabilité entre des systèmes n'ayant pas le même vocabulaire. En d'autres termes, tout système partageant des données explicites doit hériter du vocabulaire RDF (figure 2.2). La propriété *is a* utilisée dans l'exemple correspond à *type* dans le schéma RDF qui définit le sujet comme une instance de l'objet.

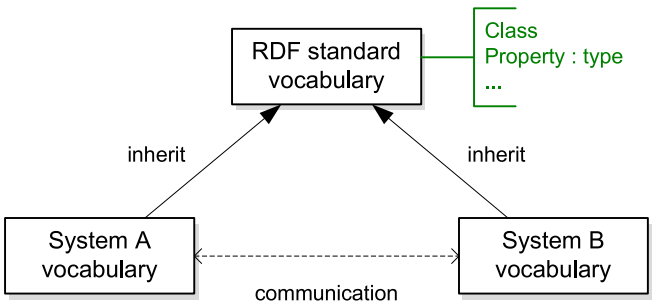



FIGURE 2.2 – Communication entre deux systèmes

Les triplets ci-dessus peuvent être représentés graphiquement avec le format illustré dans la figure 2.3.


FIGURE 2.3 – Représentation graphique d'un triplet

La représentation graphique de l'équipement dans la figure 2.4 montre les relations entre les sujets et les objets (association).

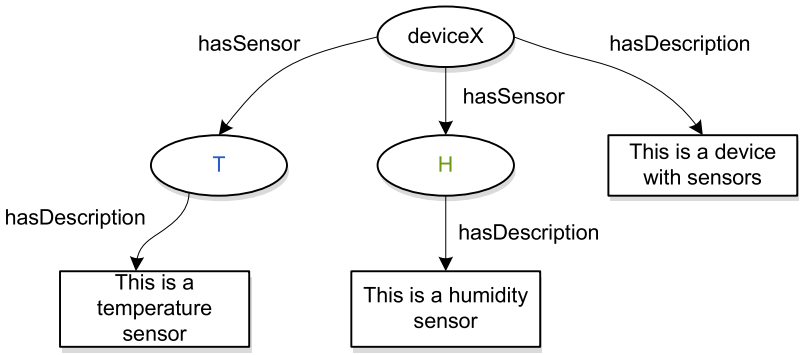



FIGURE 2.4 – Représentation graphique de l'équipement *deviceX*

Le schéma de la figure 2.4 peut être sérialisé à l'aide de la syntaxe XML/RDF. Dans l'exemple suivant, les classes sont définies dans un document à l'adresse :

`http://www.plateform.org/classes.rdf`

et les propriétés dans un document à l'adresse :

`http://www.plateform.org/properties.rdf`

```
1  <?xml version="1.0"?>
2  <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
3         xmlns:classes="http://www.plateform.org/classes.rdf#"
4         xmlns:properties="http://www.plateform.org/properties.rdf#">
5
6    <rdf:Description rdf:ID="Device">
7      <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
8    </rdf:Description>
9
10   <rdf:Description rdf:ID="Sensor">
11     <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
12   </rdf:Description>
13
14   <rdf:Description rdf:about="deviceX">
15     <rdf:type rdf:resource="Device" />
16     <properties:hasDescription>This is a device with sensors</properties:hasDescription>
17     <properties:hasSensor rdf:resource="T" />
18     <properties:hasSensor rdf:resource="H" />
19   </rdf:Description>
20
21   <rdf:Description rdf:about="T">
22     <rdf:type rdf:resource="Sensor" />
23     <properties:hasDescription>This is a temperature sensor</properties:hasDescription>
24   </rdf:Description>
25
26   <rdf:Description rdf:about="H">
27     <rdf:type rdf:resource="Sensor" />
28     <properties:hasDescription>This is a humidity sensor</properties:hasDescription>
29   </rdf:Description>
30
31 </rdf:RDF>
```

L'élément parent contenant la propriété *rdf :Description* représente le sujet du triplet et chaque élément enfant représente l'objet avec le type de propriété qui le lie au sujet, par exemple *properties :hasSensor*. Toutes les propriétés doivent être définies dans un vocabulaire désigné par le terme ontologie.

## 2.3 Ontologies S-TEN et EMEP

*S-TEN* [1] est un projet dont le but est d'exploiter le Web sémantique pour des applications scientifiques et d'ingénierie. Il met à disposition des ontologies définissant des classes et des propriétés applicables notamment au domaine de la publication de mesures scientifiques sur le Web.

L'ontologie *EMEP* [2] définit un vocabulaire visant à rendre explicite les données d'analyse relatives à l'environnement et à la consommation d'énergie. Elle se base sur l'ontologie *S-TEN* [1].

Il a été décidé que la publication RDF des mesures de la plateforme se base sur ces ontologies afin de rendre les données publiées interprétables par toutes les applications scientifiques comprenant ces ontologies.

## 2.4 Structure de la plateforme

La structure de la plateforme regroupe l'ensemble des ressources la composant ainsi que les URIs les identifiant. Elle doit permettre au client d'accéder aux informations concernant l'équipement de mesure, les capteurs ainsi que toutes les mesures. Une bonne idée de format d'URI pour représenter des mesures temporelles est la suivante :

```
http://device12.hevs.ch/measurement_item/2009-06-22T23-00-00
```

où *device* définit l'équipement, *measurement_item* définit le capteur et *2009-06-22T23-00-00* (représentation de la date et du temps ISO 8601) définit la période regroupant une ou plusieurs mesures. Cette structure a l'avantage d'être lisible et explicite.

### 2.4.1 Ressources

Chaque ressource permet d'obtenir des informations. La liste des identificateurs URI de chaque ressource de la plateforme est la suivante :

```
device12.hevs.ch
device12.hevs.ch/measurement_item
device12.hevs.ch/measurement_item/2009
device12.hevs.ch/measurement_item/2009.values
device12.hevs.ch/measurement_item/2009.stat
```

`device12.hevs.ch` est l'identifiant URI de l'équipement. Il permet d'obtenir les informations concernant l'équipement.

`device12.hevs.ch/measurement_item` est l'identifiant URI du capteur. Il permet d'obtenir les informations sur le capteur comme notamment les types d'observations qu'il peut effectuer qui correpondent aux calculs statistiques comme le calcul de la valeur maximale, minimale et moyenne.

`device12.hevs.ch/measurement_item/2009` identifie l'activité du capteur durant une période (ici, l'année 2009). Cette période peut être une année, un mois, un jour ou une heure du format de date ISO.

`device12.hevs.ch/measurement_item/2009.values` est l'identificateur URI des valeurs des mesures d'une période. Il permet d'obtenir toutes les mesures qui ont été effectuées dans cette période. Dans cet exemple, les mesures effectuées entre 2009 et 2010.

`device12.hevs.ch/measurement_item/2009.stat` est l'identificateur URI les valeurs statistiques des mesures d'une période. Il permet d'obtenir les valeurs statistiques, soit la valeur minimale, maximale et moyenne des mesures qui ont été effectuées dans cette période.

Afin de pouvoir rendre accessible toute l'information de la plateforme, il est nécessaire de lier les ressources citées précédemment. Le graphique RDF de la figure 2.5 illustre les différents liens entre les ressources et permet ainsi d'obtenir toute l'information. Les ressources sans extension permettent de lier les ressources entre elles et donc de créer une arborescence. Pour rendre le graphique plus lisible, seulement la dernière partie de l'URI de chaque ressource est précisée et le graphique s'arrête au niveau des mois.

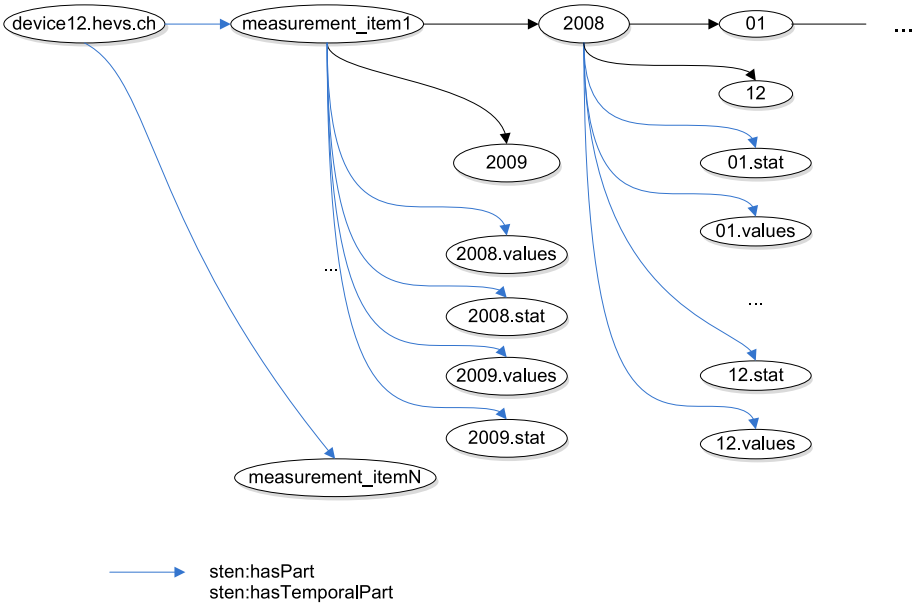



FIGURE 2.5 – Représentation graphique des ressources de la plateforme

Cette arborescence permet d'obtenir les valeurs statistiques et normales de chaque période. Le graphique RDF de la figure 2.6 complète le graphique de la figure 2.5 en illustrant l'affichage des valeurs statistiques et normales à partir des ressources.


FIGURE 2.6 – Représentation graphique des ressources retournant des valeurs

Les propriétés *sten :hasPart*, *sten :hasTemporalPart* et *sten :decimal* sont définies dans l'ontologie *S-TEN* [1].

### 2.4.2 Parcours de la structure

La figure 2.7 montre le chemin parcouru depuis la ressource designant l'équipement, jusqu'à la valeur de la mesure. Dans cet exemple, la valeur de la mesure de température réalisée le 9 avril 2009 à 22h20 est trouvée en parcourant toute la structure des ressources.

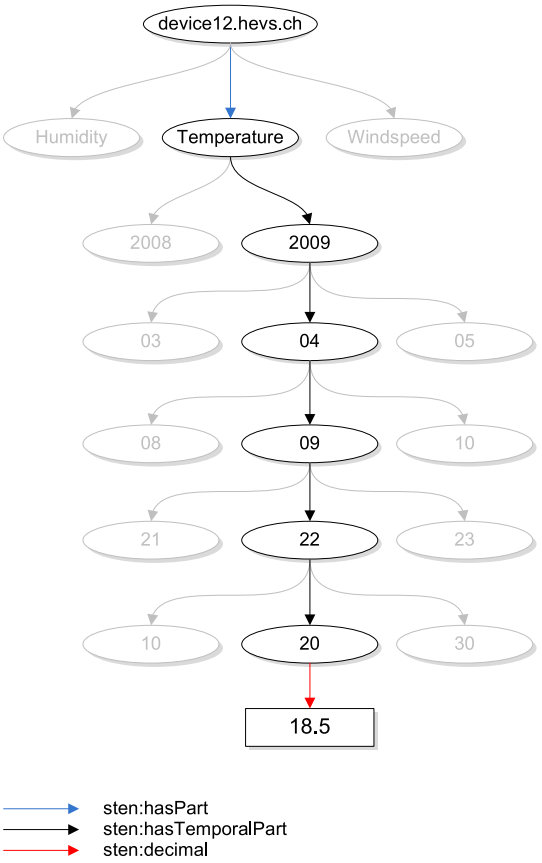



FIGURE 2.7 – Chemin depuis la ressource *device* jusqu'à la valeur de la mesure

Ce graphique montre bien que la valeur de la mesure est obtenue après la ressource représentant la minute. En effet, la plateforme est conçue pour des mesures ayant une résolution maximale d'une minute. Dans la plupart des cas, cette résolution est suffisante.

CHAPITRE 3

# Architecture

## 3.1 Navigation

L'architecture de navigation définit, pour chaque ressource de la structure de la plateformem, le lien sur le document RDF ou HTML la décrivant. Ainsi, à chaque ressource identifée par une URI correspond un document RDF et HTML. Ce document est accessible sur le web par une adresse web appelée URL (*Uniform Resource Locator*) pour la distinguer de l'identifieur d'une ressource URI. L'adresse web du document RDF (URL) est conceptuellement différente de l'identifieur (URI) de la ressource. L'URI et son URL correspondante sont donc différentes. Par contre, la connaissance de l'URI doit permettre de retrouver l'URL correspondante.

Les URIs identifient des ressources pertinentes dans le cadre de la publication de données. Elles doivent donc obligatoirement permettre de retrouver l'URL correspondante au document RDF et HTML décrivant ces ressources. Cependant, il n'est pas possible de donner la même adresse à une ressource réelle et à une ressource Web (les documents RDF et HTML sont des ressources). En effet, il ne serait plus possible de distinguer quelle ressource cette adresse identifie. Pour éviter cette ambiguïté et rendre possible l'adressage de plusieurs documents à partir d'une seule et unique URI, un mécanisme de négociation de contenu avec hachage est proposé par le *W3C* [3] dans le document *Cool URIs for the Semantic Web* [4]. Ce mécanisme est présenté dans la figure 6.2.

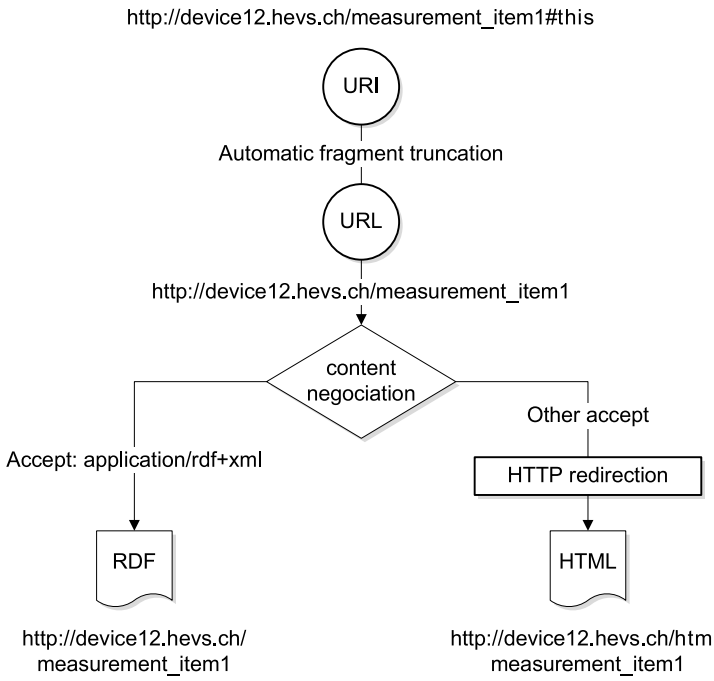



FIGURE 3.1 – Mécanisme de négociation de contenu avec hachage

Les URIs identifiant les ressources réelles contiennent un élément de hachage permettant de

les différencier des URLs. Dans un premier temps, l'élément de hachage de l'adresse URI est automatiquement tronqué pour qu'elle corresponde à l'URL d'un document RDF. Cette URL ne pouvant adresser que le document RDF, il est ensuite nécessaire, dans le cas du HTML, de retourner l'URL adressant le document HTML. Ceci est fait à l'aide de la négociation de contenu qui se base sur le type de contenu (*MIME type*) précisé dans le champ *Accept* de la requête HTTP. Dans le cas du HTML, une redirection HTTP *303 See Other* permet d'indiquer au client que l'URI requise n'identifie pas un document HTML et le force donc à se rediriger sur l'URL HTML retournée. Le préfixe *html* de cette URL la différencie de l'URL du document RDF. Ainsi, il est possible pour un client d'atteindre une des deux représentations à partir de l'URI de l'objet réel sans aucune ambiguïté. Cette solution a malgré tout un désavantage : la redirection HTTP peut causer une latence. Cependant, et dans la plupart des cas, cette dernière n'est pas perceptible par un client humain.

Les documents RDF et HTML contiennent des références croisées (*cross reference*). La référence depuis la représentation RDF sur la représentation HTML se fait à l'aide du triplet où la propriété *isDefinedBy* du schéma RDF met en relation l'URL du document RDF et celle du document HTML :

`RDF_document_URL` isDefinedBy `HTML_document_URL`

En HTML, l'élément *<link>* permet de définir le lien vers le document RDF : l'attribut *type* permet de spécifier le type de contenu du document cible (dans ce cas *application/rdf+xml*) et l'attribut *href*, l'adresse de ce document.

### 3.1.1 Documents RDF

Chaque document RDF contient tous les triplets dont la ressource décrite par le document est le sujet comme le montre la figure 3.2. Ce choix a été fait dans le but d'alléger la taille des documents RDF qui sont déjà verbeux en limitant leur contenu à un niveau de triplets (sujet, prédicat, ressource).

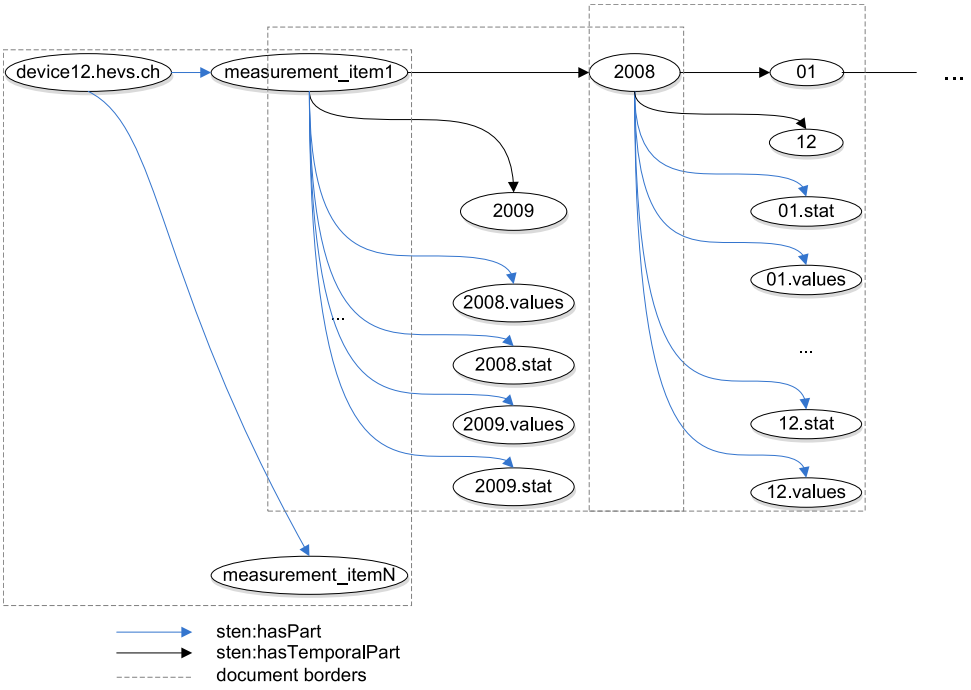



FIGURE 3.2 – Vue document de la représentation graphique des ressources

Les documents RDF sont sérialisés à l'aide du XML. Ils doivent obligatoirement contenir les URIs des ontologies qu'ils utilisent et être construits comme suit :

```
1 <?xml version="1.0" ?>
2
3 <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
4   ...
5 </rdf:RDF>
```

L'élément `<rdf:RDF>` est l'élément principal (*root*). Il définit le document XML comme étant un document RDF. Il contient également une référence sur le vocabulaire RDF standard. Toutes les ontologies utilisées dans le document peuvent être référencées dans l'élément principal. Ces références constituent l'espace de nom du document (*namespace*). Dans le cas de l'utilisation des ontologies *S-TEN* [1] et *EMEP* [2], le document a la forme suivante :

```
1 <?xml version="1.0" ?>
2
3 <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
4     xmlns:sten="http://www.s-ten.eu/sten-core#"
5     xmlns:emep="http://sten-04.hevs.ch/emep.rdf#">
6   ...
7 </rdf:RDF>
```

### 3.1.2 Documents HTML

Les documents HTML contiennent une représentation humaine des informations de la plateforme. Comme vu en début de chapitre, chaque ressource est décrite par un document RDF et HTML. Cette règle a été légèrement contournée afin de rendre l'information plus lisible pour un client humain. En effet, une ressource identifiée par l'URI `device12.hevs.ch/measurement_item/2009.graph` n'est décrite que par un document HTML. Ce document HTML contient une vue graphique des valeurs statistiques des mesures durant une période. Ce graphique est généré à l'aide de l'*API Google Chart* [5] et est présenté dans la figure 3.3.

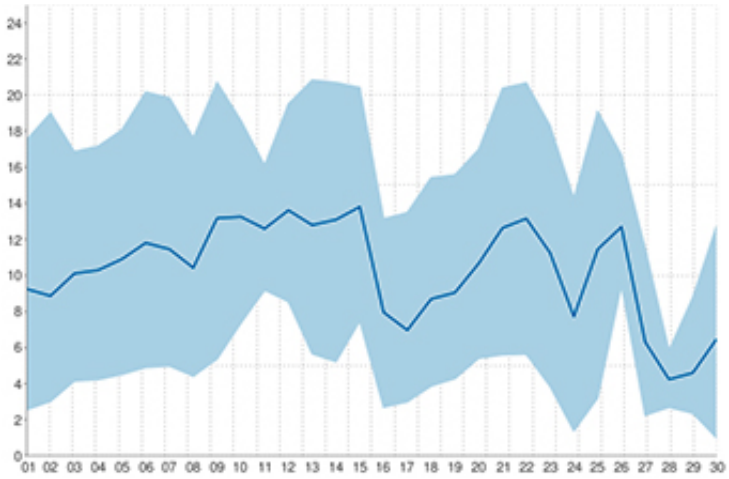



FIGURE 3.3 – Graphique de représentation des statistiques

En admettant que ce graphique expose les valeurs statistiques du mois d'avril 2009, pour obtenir les mêmes résultats, une application client doit accéder aux ressources présentées dans la figure 3.4.

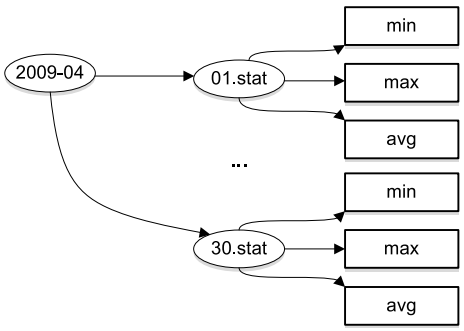



FIGURE 3.4 – Obtention des valeurs statistiques à l'aide du graphique RDF

Dans ce cas, les résultats sont alors obtenus à l'aide de trente et un documents RDF : un contenant la liste des ressources représentant les valeurs statistiques et un document par ressource contenant les valeurs statistiques.

Une deuxième représentation de l'information propre au HTML est le document décrivant la ressource identifiée par l'URI `device12.hevs.ch/measurement_item/2009.stat`. Ce document contient l'histogramme du nombre de mesures dont les valeurs sont comprises dans des intervalles définis ainsi que la valeur minimale, maximale et moyenne de l'ensemble de ces mesures. Le nombre de mesures traîtées dépend de la période précédant l'extension. Un aperçu est présenté dans la figure 3.5. Cet histogramme est une vue purement utilisateur des valeurs statistiques et n'est donc pas disponible en RDF. Il est également généré à l'aide de l'*API Google Chart* [5].

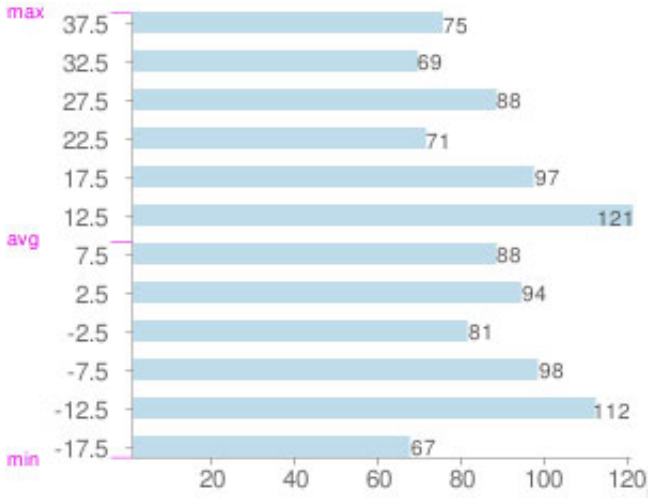



FIGURE 3.5 – Histogramme statistique du nombres de mesures dans des intervales

Les valeurs des mesures définies par la ressource identifiée par l'URI `device12.hevs.ch/measurement_item/2009.values` sont retournées au client dans un document en format texte CSV. Les mesures pouvant être nombreuses, il ne fait pas de sens de les afficher dans un document HTML. En effet, les balises HTML alourdiraient considérablement le document retourné.

En ce qui concerne les URIs sans extension permettant de parcourir l'arborescence des ressources, l'élément *<a href>* est utilisé car il permet d'établir un lien vers un autre document HTML et donc de lier toutes les ressources de l'arborescence.

## 3.2 Plateforme

L'architecture de la plateforme est illustrée dans la figure 3.6. Elle représente tous les éléments nécessaires à l'acquisition des mesures, à leur stockage et à leur publication sur le Web.

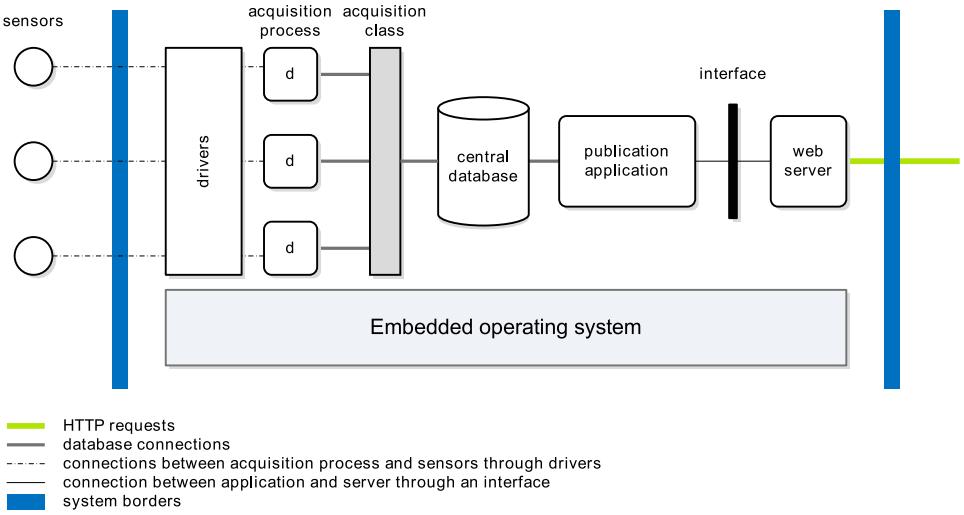



FIGURE 3.6 – Architecture de la plateforme

Le serveur Web est l'élément faisant la liaison entre le Web et la plateforme de mesure. Il a pour rôle de recevoir la requête HTTP client et de la transmettre à l'application de publication. C'est lui qui va exécuter l'application de publication à son démarrage.

L'application de publication lit les données dans la base de données centrale avec lesquelles elle génère un document HTML ou RDF, selon le type de client, et le lui renvoie à travers le serveur Web.

La base de données est l'organe central de la plateforme. C'est elle qui contient toutes les données de la plateforme (mesures, description, ...) ainsi que la configuration des processus d'acquisition.

Le rôle des processus d'acquisition est de lire les valeurs mesurées par chaque capteur à l'aide de drivers et de les écrire dans la base de données centrale à travers une classe d'acquisition. Un processus d'acquisition est dédié à un capteur.

Les capteurs sont indépendants de la plateforme. Ils sont accessibles par la plateforme à l'aide des processus d'acquisition communiquant avec eux à travers des drivers.

Toute la plateforme repose sur un système d'exploitation embarqué.

CHAPITRE 4

# Environnement de la plateforme

L'environnement se compose de tous les outils nécessaires au développement de la plateforme. Les outils suivants ont été choisis pour leurs caractéristiques avantageuses dans le cas d'une application embarquée.

## 4.1 Serveur web

*Lighttpd* [6] est un serveur web performant avec une très faible emprunte mémoire et une bonne gestion de l'utilisation du CPU. Il met à disposition des modules CGI/FastCGI et supporte le SSL.

## 4.2 Système de gestion de base de données

Le moteur de base de données *SQLite* [7] a été imposé dans le cahier des charges. Cette bibliothèque écrite en C utilise peu de ressources par rapport à d'autres moteurs et implémente le standard SQL. Une base de donnée *SQLite* [7] est stockée dans un fichier.

## 4.3 Langage de programmation

Pour que le système soit performant, il a été décidé d'utiliser un langage compilé. Le langage choisi est le C++ de part le grand nombre de frameworks disponibles pour ce langage et les possibilités qu'il offre dont l'orienté objet.

## 4.4 Bibliothèque logicielle Qt

*Qt* [11] est une bibliothèque logicielle orientée objet et multi-plateforme développée en C++. Elle offre des classes intéressantes dans le cadre de ce projet, telles que QSql pour l'utilisation des bases de données SQL (prise en charge de SQLite) et des classes permettant la manipulation de chaînes de caractères (utile pour le traitement de l'URI *URI parsing*). Elle possède également son propre mécanisme de gestion de mémoire (*garbage collector*).

## 4.5 Interfaces CGI et FastCGI

*CGI* [8] et *FastCGI* [9] sont des interfaces permettant d'exécuter un programme et de communiquer avec le serveur web par les flux standards *stdin*, *stdout* et *stderr*. Ces interfaces sont indépendantes du système d'exploitation, du langage de programmation ainsi que du serveur web, c'est pourquoi il est intéressant de les utiliser dans le cas d'une application compilée écrite en C++.

## 4.6 CGI

Dans une interface *CGI* [8], un processus est créé pour chaque requête, ce qui peut faire surcharger le serveur lors de nombreuses requêtes concurrentes.

La relation entre le serveur web et l'interface *CGI* [8] est présentée dans la figure 4.1. Le serveur web communique les informations concernant la requête du client, celles concernant le client et celles concernant le serveur web lui-même dans des variables d'environnement. Les paramètres envoyés à l'aide de la méthode POST passent par le flux de donnée standard *stdin*.

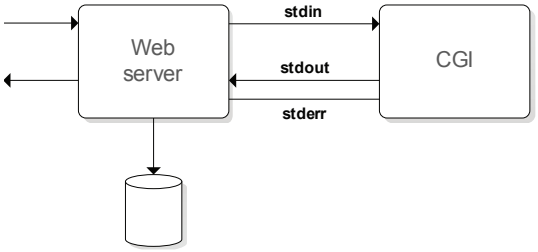



FIGURE 4.1 – Relation entre le serveur web et l'interface CGI

## 4.7 FastCGI

Dans une interface *FastCGI* [9], un ou plusieurs processus est (sont) créé(s) au démarrage du serveur (selon sa configuration). Le ou les processus reste(nt) actif(s) et est (sont) appelé(s) par le serveur lors de requêtes. Ceci peut s'avérer dangereux dans le cas d'allocation dynamique de mémoire où une mauvaise désallocation de mémoire pourrait entraîner la chute du serveur. La relation entre le serveur web et l'interface est présentée à la figure 4.2. Les données échangées passent à travers un protocole dans lequel une en-tête est rajoutée.

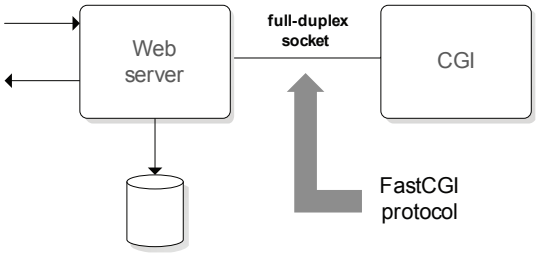



FIGURE 4.2 – Relation entre le serveur web et l'interface FastCGI

Une application traitée par l'interface *FastCGI* [9] doit avoir la structure suivante :

```
1  #include "fcgi_stdio.h"
2
3  int main()
4  {
5     // the main loop
6     while (FCGI_Accept() == 0)
7     {
8        // algorithm
9     }
10
11    return 0;
12 }
```

La fonction *FCGI_Accept* accepte une nouvelle requête du serveur HTTP et créé un environnement d'exécution compatible *CGI* [8] pour la requête (variables d'environnements semblables à celles de *CGI* [8]).

## 4.8 Choix de l'interface

Le choix d'une de ces deux interfaces est fait en fonction du temps de réponse de l'application. Pour les départager, deux tests de performances ont été réalisés à l'aide de l'application *ApacheBench* [10]. Cet outil permet de tester le temps de réponse d'une ou plusieurs requêtes effectuées sur un serveur web.

### 4.8.1 Environnement des tests

La configuration utilisée pour les tests est la suivante :

- Configuration matérielle : Intel Pentium 4 CPU 3.00GHz, 1Go RAM
- Système d'exploitation : Linux v2.6.24-1-686
- Serveur web : Lighttpd - v1.4.19 (ssl)
- Programme de test : ApacheBench - v2.3

Les modules *mod_fastcgi* et *mod_cgi* de *Lighttpd* [6] ont été activés et configurés (les fichiers de configurations sont visibles en annexe A).

### 4.8.2 Tests

Les tests ont été effectués en local sur deux applications écrites en C++. La première, *hello*, retourne un document HTML affichant une *hello world* (voir annexe B). La deuxième , *dbread*, effectue une connexion à une base de données et lit 100'000 enregistrements pour ensuite les afficher dans un document HTML (voir annexe C). Dans la deuxième application, l'interface *FastCGI* [9] permet de n'ouvrir qu'une seule fois la connexion à la base de données tandis qu'avec *CGI* [8], une connexion est ouverte à chaque requête.

Le but de ces tests est d'observer le temps de réponse des deux applications à travers chaque interface et lors de plusieurs requêtes simultanées. Il a été décidé de simuler dix requêtes simultanées répétées cent fois. La commande suivante effectue cette simulation :

```
ab -c10 -n100 application_url.
```

### 4.8.3 Résultats

Les résultats sont présentés dans les figures 4.3 et 4.4. Ils montrent que le *FastCGI* [9] est en moyenne cinquante fois plus rapide que le *CGI* [8] dans le cas de l'application affichant un *hello world*, et environ quatre fois plus rapide pour l'application se connectant à une base de données et effectuant des requêtes. Il est toutefois bien évident que les résultats pourraient être différents selon l'implémentation des applications testées et l'environnement de test.

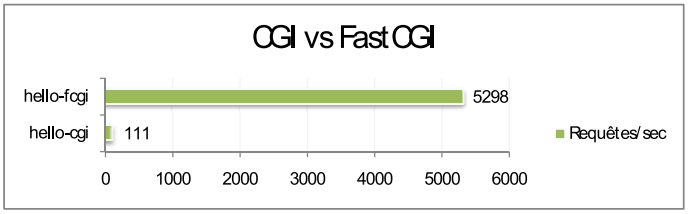



FIGURE 4.3 – Résultat des tests de performances entre hello-cgi et hello-fcgi

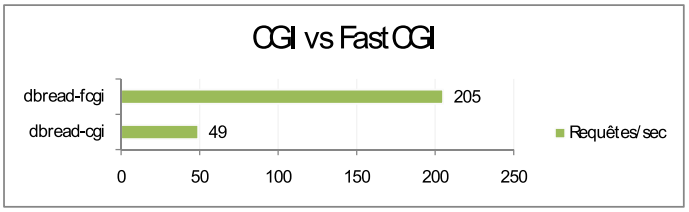



FIGURE 4.4 – Résultat des tests de performances entre dbread-cgi et dbread-fcgi

En réalité, il faut s'attendre à un rapport entre les deux interfaces proche de celui de l'application dbread (rapport d'environ quatre). La ressemblance de la structure des programmes écrits pour chaque interface permettra de réaliser un test réel avec l'application de publication sans devoir apporter de grosses modifications à celle-ci. Le choix a été porté sur le *FastCGI* [9] pour les raisons de performances présentées ci-dessus et de stabilité (nombre fixe de processus actifs).

CHAPITRE 5

# Conception de la base de données

## 5.1 Indroduction

La base de données est la partie centrale de la plateforme : c'est à travers elle que les applications peuvent communiquer. Elle contient non seulement toutes les données de mesures mais également les données de configuration de la plateforme. En effet, pour avoir un système qui puisse évoluer sans grandes modifications, il est nécessaire de définir le maximum d'informations dans la base de données plutôt que dans les applications. La base de données est séparée en quatre parties comme le montre la figure 5.1.

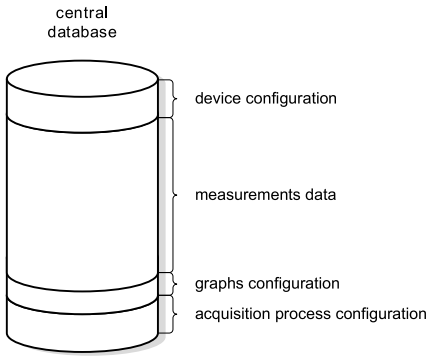



FIGURE 5.1 – Organisation de la base de données

La partie *Configuration de l'équipement* (*Device configuration*) contient toutes les informations de l'équipement de mesure comme son nom, sa position géographique, son type et l'adresse DNS qui le définit.

La partie *Données de mesures* (*Measurements data*) contient toutes les informations relatives aux mesures et aux capteurs les réalisant.

La partie *Configuration des graphiques* (*Graphs configuration*) contient les informations relatives à la mise en forme des graphiques HTML.

La partie *Configuration des processus d'acquisition* (*Acquisition process configuration*) contient toutes les informations requises par les processus d'acquisition pour communiquer avec les capteurs, récupérer les informations et les stocker dans la base de données.

Il est à noter que le moteur de base de données *SQLite* [7], utilisé pour le stockage des données, ne peut manipuler que les types de données suivants :

NULL : valeur NULL
INTEGER : nombre integer signé stocké sur 1 à 8 bytes selon la taille de la valeur.
REAL : nombre à virgule flottante stocké sur 8 bytes
TEXT : chaîne de caractères stockée selon l'encodage de la base de données
BLOB : valeur d'un grand objet binaire

Il est possible de déclarer des colonnes dans d'autres types de données que ceux présentés ci-dessus. Le moteur de base de données les comprend et leur donne une affinité avec ses propres types : par exemple si une colonne est déclarée en CHAR, le moteur de base de données la stockera en tant que type TEXT.

## 5.2 Configuration de la plateforme

Le diagramme d'entité-relation modélisant cette partie est illustré dans la figure 5.2. Il a été décidé de ne définir que deux attributs pour des raisons de flexibilité : un attribut *parameter*, contenant une chaîne de caractères représentant le nom du paramètre et un autre, *value*, contenant la valeur de ce paramètre. En effet, les propriétés géométriques de la plateforme (latitude, longitude, altitude), son nom ainsi que son type sont des paramètres connus mais il se peut que dans le futur, d'autres informations viennent compléter cette liste. C'est pourquoi le modèle choisi est préférable à un modèle où chaque paramètre correspondrait à un attribut. Dans un tel cas, l'ajout de paramètre demanderait une modification de la structure de la base de données, ce qui n'est pas concevable dans un système se devant d'être évolutif.

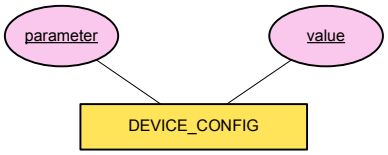



FIGURE 5.2 – Diagramme d'entité-relation de la partie configuration de la plateforme

Le modèle relationnel de cette partie est présenté dans la figure 5.3.

| DEVICE_CONFIG | |
|---|---|
| PK | parameter |
| | value |

FIGURE 5.3 – Modèle relationnel de la partie configuration de la plateforme

Les deux attributs doivent être de type *TEXT* pour une raison d'uniformité : deux paramètres peuvent être de types différents, comme par exemple les positions géométriques, exprimées numériquement et le nom de la plateforme, exprimé littéralement.

## 5.3 Mesures

Le diagramme de la figure 5.4 montre les relations entre les différentes entités nécessaires à la définition de mesures. Les cardinalités entre les entités indiquent qu'un capteur peut mesurer un seul et unique paramètre mais peut, par contre, effectuer un nombre illimité théorique de mesures. Elles montrent également qu'une mesure est unique à un temps donné. Les clés primaires sont nécessaires pour réaliser les relations entre les tables.

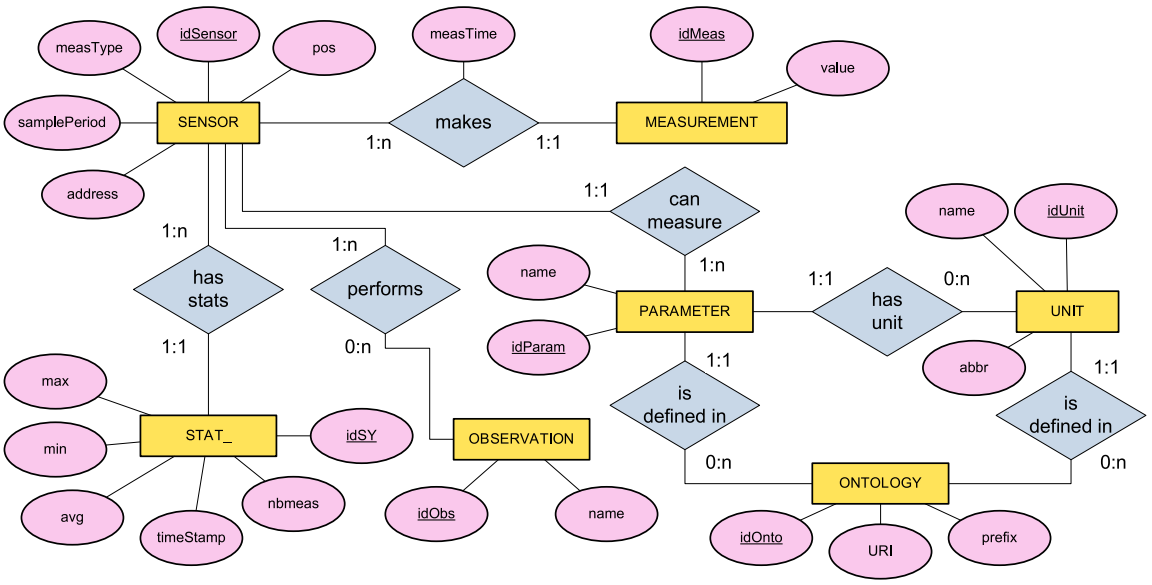



FIGURE 5.4 – Diagramme d'entité-relation de la partie mesures

L'entité *SENSOR* contient la liste de tous les capteurs du système. L'attribut *measType* représente le type de mesure effectué par le capteur. Il est de type *TEXT*. L'attribut *pos* contient une chaîne de caractères (*TEXT*) définissant le lieu du capteur : par exemple si celui-ci se trouve à l'extérieur, cet attribut a la valeur *outside*. Les attributs *samplePeriod* et *addr* sont utilisés par la classe d'acquisition pour remplir la base de données. L'attribut *samplePeriod* indique quel est l'intervalle de temps de mesure du capteur, en minutes. Il est de type *INTEGER*. L'attribut *addr* contient l'adresse du capteur. Il est de type *TEXT* car le format d'adresse peut changer d'un capteur à l'autre (adresse MAC, Bluetooth, ...).

L'entité *PARAMETER* contient la liste de tous les paramètres physiques mesurables par le système. L'attribut *name* représente le nom du paramètre physique, par exemple la température de l'air *AirTemperature*. Il est de type *TEXT*. Chaque paramètre doit être lié à une ontologie comme le représente la relation entre cette entité et l'entité *ONTOLOGIES*. Pour que les paramètres publiés soient compris par les programmes scientifiques se basant sur les ontologies *S-TEN* [1] et *EMEP* [2], ils doivent être définis dans une de ces deux ontologies.

L'entité *UNIT* contient la liste des unités de mesure. L'attribut *name* est le nom de l'unité, par exemple *Celsius* dans le cas d'une température de l'air, et l'attribut *abbr*, l'abréviation de l'unité, $°C$ pour les *Celsius*. Ils sont stockés en type *TEXT*. Comme pour l'entité *MEASUREMENT*, les unités doivent être définies dans une des ontologies respectées. Cette table a été définie car un paramètre peut avoir plusieurs unités. Par exemple, la température peut être exprimée en Celsius et en Kelvin.

L'entité *ONTOLOGY* contient la liste des ontologies utilisées pour définir explicitement les données de mesures de la plateforme. Cette liste est utilisée pour la définition des *namespace* dans les documents générés en RDF. L'attribut *URI* contient les URIs définissant les ontologies et l'attribut *prefix*, le préfixe utilisé dans les documents RDF. Ils sont de type *TEXT*. L'entité *MEASUREMENT* contient les valeurs des mesures effectuées par les capteurs : ces

valeurs étant des nombres à virgule, l'attribut *value* est de type *REAL*. L'attribut *measTime* de la relation entre les deux entités *SENSOR* et *MEASUREMENT* contient le moment auquel a été effectuée la mesure. Il est de type *INTEGER* car le format de date *timestamp UNIX* est utilisé. Cette représentation des dates est moins lisible que le format date ISO

mais à l'avantage d'être performant lors de l'exécution des requêtes. En effet, la comparaison de types *INTEGER* est beaucoup plus rapide que la comparaison de types *TEXT*, dans le cas par exemple de recherches de mesures dans un intervalle de temps.

L'entité *OBSERVATION* contient la liste des observations réalisables par le capteur. Ces observations sont en fait les calculs statistiques qui peuvent être réalisés sur les mesures du capteur. L'attribut *name* de type *TEXT* est le nom de l'observation.

Les entités *STAT_YEARS*, *STAT_MONTHS* et *STAT_DAYS* contiennent les valeurs statistiques des mesures comme la valeur moyenne, la valeur minimale et la valeur maximale contenues dans les attributs *avg*, *min*, *max* de type *REAL* pour chaque période représentée par l'attribut *TIMESTAMP* de type *INTEGER*. Elles sont représentées dans la figure 5.4 par l'entité *STAT_*. Ces entités ont été définies pour améliorer la performance du système (cf. chapitre 8 page 46). L'attribut *nbmeas* contient le nombre total des mesures sur lequel sont basées les valeurs statistiques. En effet, la classe d'acquisition doit mettre à jour les valeurs statistiques des périodes concernées à acquisition de mesure. Elle doit donc connaître le nombre de mesures total pour déterminer la nouvelle valeur moyenne qui est calculée avec l'équation suivante :

$$avg_{new} = \frac{\sum values + value_{new}}{nb_{meas} + 1}$$

où

$$\sum values = avg_{current} \cdot nb_{meas}$$

Le modèle relationnel de la figure 5.5 présente toutes les tables que contient cette partie de la base de données et les relations entre elles. L'attribut *measTime* de la relation entre les entités *SENSOR* et *MEASUREMENT* fait partie de la table *MEASUREMENT* car une mesure est unique à un temps donné.

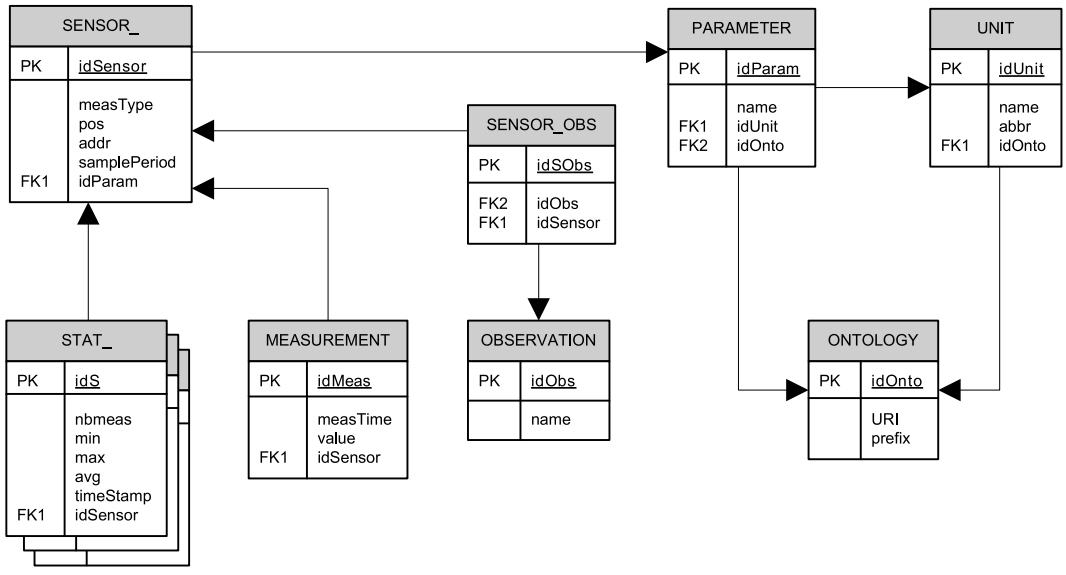



FIGURE 5.5 – Diagramme relationnel de la partie mesures

## 5.4 Configuration des graphiques

Cette partie permet de définir les valeurs minimales et maximales du graphique de statistiques HTML pour chaque capteur. En effet, la plage de valeurs d'un capteur dépend de la propriété physique qu'il mesure, ainsi que de son unité. Un capteur mesurant une température en Celsius a généralement une plage de valeurs allant de -20 à 40, tandis qu'un capteur mesurant la même propriété physique mais en Kelvin a une plage de valeurs allant de 253,15 à 313,15. Dans le modèle relationnel de la figure 5.6, l'attribut *idSensor* de type *INTEGER* contient l'identificateur du capteur défini dans la table *SENSOR* et les attributs *minRange* et *maxRange* de type *INTEGER* contiennent les valeurs minimales et maximales de l'axe des valeurs du graphique.

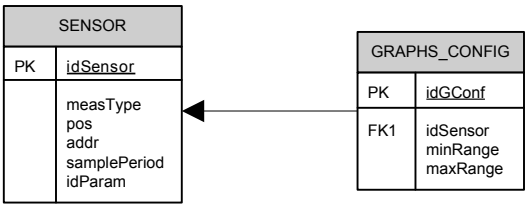



FIGURE 5.6 – Diagramme d'entité-relation de la partie de configuration des graphiques

## 5.5 Configuration des processus d'acquisition

Cette partie n'a pas été conçue car la réalisation des processus d'acquisition ne fait pas partie des objectifs. Cependant, elle doit être prise en considération pour des améliorations futures et devra donc être réalisée. Elle devrait contenir toute les informations nécessaires à la communication entre les processus d'acquisition et les drivers des capteurs.

# CHAPITRE 6

# Application de publication

L'application de publication a pour tâche de générer des documents RDF ou HTML contenant les informations récupérées dans la base de données en fonction de la requête HTTP reçue. Elle communique avec le serveur Web à travers l'interface *FastCGI* [9]. Les phases principales de l'application sont présentées dans la figure 6.1. Le diagramme de flux de l'application de publication est visible en annexe D et le code en annexe E.

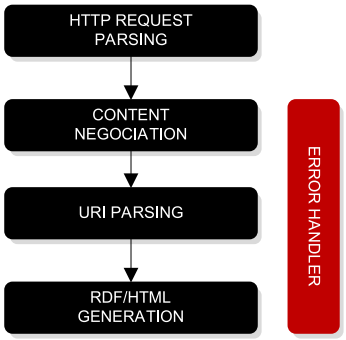



FIGURE 6.1 – Phases de l'application de publication

## 6.1 HTTP request parsing

Dans la première phase de l'application, les informations concernant l'URI et le type de contenu accepté et contenues dans la requête HTTP doivent être récupérées. En effet, ces informations sont nécessaires pour la génération des documents car elles définissent quelle ressource le document décrit (type d'informations) ainsi que dans quel format il doit être retourné.

Les informations de la requête HTTP sont communiquées par le serveur web à l'application dans des variables d'environnement. La fonction *getenv* de la librairie standard C *stdlib* retourne un `char *` pointant sur le contenu de la variable d'environnement passée en paramètre.

La récupération de l'URI doit se faire en plusieurs étapes car elle est séparée dans plusieurs variables. Par exemple, pour récupérer l'URI

`http://device12.hevs.ch/measurement_item/2009-06-10`

les variables d'environnement sont les suivantes :

*SCRIPT_NAME* qui contient `/measurement_item`
*PATH_INFO* qui contient `/2009-06-10`

L'adresse DNS `device12.hevs.ch` de l'URI identifiant l'équipement n'est pas récupérée. En effet, cette adresse identifie la plateforme sur le Web et est donc connue par la plateforme elle-même. Elle ne joue donc aucun rôle dans la détermination de quel document il faut retourner.

Le code suivant montre la récupération de l'URI dans une chaîne de caractères à l'aide de la classe QByteArray de Qt qui offre une fonction *append* permettant d'ajouter une chaîne de caractères à la suite d'une chaîne existante :

```
1 QByteArray reqURI( getenv( "SCRIPT_NAME" ) );
2 reqURI.append( getenv( "PATH_INFO" ) );
```

Il est à noter que *Qt* [11] alloue dans la mémoire une taille supérieure à celle de la chaîne de caractères afin d'optimiser ses modifications. La fonction *capacity* permet de voir la taille réelle de la chaîne de caractères allouée dans la mémoire. Une fonction *squeeze* offre la possibilité de supprimer les éléments vides de cette chaîne et donc de libérer la mémoire.

Le type de contenu accepté est stocké dans la variable d'environnement *HTTP_ACCEPT*.

## 6.2 Content negociation

Cette étape est l'implémentation du mécanisme de négociation de contenu expliqué dans la section 3.1 à la page 31. Dans le champ *Accept* de l'entête HTTP, le type de document RDF est défini par le type de contenu (*MIME type*) *application/rdf+xml* et le type HTML, par *text/html*. Ce champ peut être composé de plusieurs types de contenu avec des notions de précédences et facteurs de qualité (pour plus d'informations, voir la définition du champ Accept de l'en-tête HTTP [12]).

La définition du contenu à retourner peut devenir complexe si le client accepte le RDF et le HTML avec des priorités différentes. Il a donc été décidé de mettre la priorité sur le RDF, peu importe la précédence ou les facteurs de qualité.

L'algorithme doit tester la présence de *rdf* dans la chaîne de caractères contenant le champ *Accept*. Pour ce faire, la classe QByteArray a une fonction *contains* retournant un booléen indiquant si la chaîne de caractères passée en paramètre de la fonction a été trouvée. La définition du contenu se fait à l'aide du code suivant :

```
1  QByteArray httpAccept( getenv( "HTTP_ACCEPT" ) );
2
3  if ( httpAccept.contains( "rdf" ) )
4  {
5     // content-type accepted is RDF
6  }
7  else
8  {
9     // content-type accepted is HTML
10 }
```

Dans le cas du HTML accepté, il est nécessaire de tester si l'adresse requise par le client correspond à l'URL d'un document HTML, c'est-à-dire avec un préfixe *html*. Le diagramme de flux de la figure 6.2 permet de faire ce test.

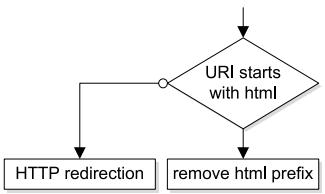



FIGURE 6.2 – Diagramme de flux de la négociation de contenu

L'algorithme ci-dessous correspond au diagramme de flux de la figure 6.2. La fonction *startsWith* de la classe QByteArray permet de tester si l'URL commence par le préfixe *html*.

```
1  if ( reqURI.startsWith( "html" ) )
2  {
3     // remove 'html/'
4     reqURI.remove( 0 , 5 );
5  }
6  else
7  {
8     printf( "Status: 303 See Other\n" );
9     printf( "Location: html/%s\n\n" , reqURI.data() );
10 }
```

où `reqURI.data()` retourne un `char *` pointant sur l'URI requise. Dans le cas où l'URI serait bien formée, le préfixe *html* est supprimé en prévision du traitement de l'URI, qui est le même pour le RDF et le HTML. Un fanion permet de propager le type de document à retourner au reste du programme.

## 6.3 URI parsing

Cette étape consiste à décomposer l'URI en plusieurs niveaux afin de pouvoir définir quelle est la ressource à décrire pour ensuite pouvoir appeler la fonction générant le document correspondant. Les différents niveaux sont illustrés dans la figure 6.3.

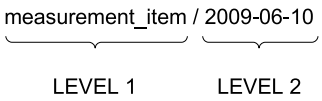



FIGURE 6.3 – Décomposition de l'URI en niveaux

Les niveaux sont séparés par des `/`. Une manière de définir la ressource à décrire est d'utiliser la fonction *split* de la classe QByteArray de *Qt* [11]. Cette fonction permet de séparer une chaîne de caractères en plusieurs chaînes où le caractère de séparation apparaît. Le code suivant illustre la décomposition en niveaux de l'URI présentée dans la figure 6.3 à l'aide de cette fonction :

```
1  QList<QByteArray> splitURI = reqURI.split( '/' );
2
3  splitURI.at(0); // contains measurement_item
4  splitURI.at(1); // contains 2009-06-10
```

Il est encore nécessaire, dans le cas de deux niveaux, de tester si l'URI contient une extension pour déterminer si la ressource est descriptive ou si elle permet d'obtenir des valeurs. Le code suivant permet de définir la ressource identifiée par l'URI :

```
1  if ( !splitURI.at(1).contains( "." ) )
2  {
3     // ressource is /measurement_item/2009-06-10
4  }
5  else if ( splitURI.at(1).contains( ".values" ) )
6  {
7     // ressource is /measurement_item/2009-06-10.values
8  }
9  else if ( splitURI.at(1).contains( ".stat" ) )
10 {
11    // ressource is /measurement_item/2009-06-10.stat
12 }
13 else
14 {
```

```
15     // ressource is not defined in the system
16 }
```

Une fois la ressource déterminée, les fonctions correspondantes, discutées dans la partie *RDF/HTML generation*, sont appelées. Si la ressource identifiée par l'URI n'est pas connue, une erreur est retournée au client à l'aide de la gestion des erreurs *Error handler* (cf. section 6.5 page 41).

## 6.4 RDF/HTML generation

Cette étape consiste à créer le document RDF ou HTML à l'aide des informations stockées dans la base de données et de le renvoyer au client. Une fonction a été implémentée pour chaque ressource. Elle va récupérer les informations concernant la ressource et les retourner au client sous forme de document. Chaque fonction respecte la procédure illustrée dans la figure 6.4.

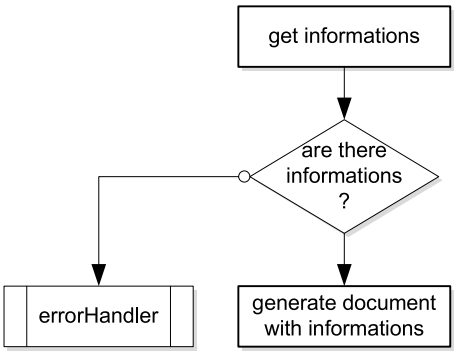



FIGURE 6.4 – Procédure de génération de fichiers

### 6.4.1 Mécanisme de génération de fichiers

Il existe plusieurs solutions pour générer des fichiers depuis une application. Deux de ces solutions sont les suivantes :

- utilisation de modèles (templates)
- codage dur

La première solution consiste à créer manuellement des modèles pour chaque document à générer et d'y insérer des macros qui pourront être remplacées par des données depuis le code à l'aide d'un moteur de modèle (*template engine*). Cette solution permet de modifier facilement les documents sans avoir à modifier le code de l'application. Elle nécessite cependant un moteur de modèle qui doit ouvrir puis parcourir, ligne par ligne, le modèle et remplacer toutes les macros. De telles opérations prennent du temps et risquent de ralentir l'application. De plus, le moteur de modèle ainsi que les modèles utilisent des ressources mémoire réduisant ainsi l'espace mémoire disponible pour les autres composants du système.

La deuxième solution consiste à générer les documents à la volée depuis le code à l'aide de la fonction *printf*. Cette solution est très performante et utilise peu de ressources mémoire. Cependant, la modification des documents peut devenir fastidieuse. En effet, le contenu du document doit être une chaîne de caractères pour qu'il puisse être traité par la fonction *printf*. Ceci implique que toutes les chaînes de caractères du document entourées par des

double guillemets doivent être échappées manuellement à l'aide du caractère *backslash*. Le code doit également être recompilé après chaque modification.

L'idée est de prendre les avantages de chacune de ces solutions pour réaliser un mécanisme de génération de documents performant et assez facile à modifier. Un simple *printf* peut simuler un moteur de modèles basique à l'aide des possibilités offertes par le pré-processeur C. En effet, ce dernier offre une macro permettant d'automatiser l'ajout du caractère d'échappement devant les doubles guillemets. La figure 6.5 présente cette idée.

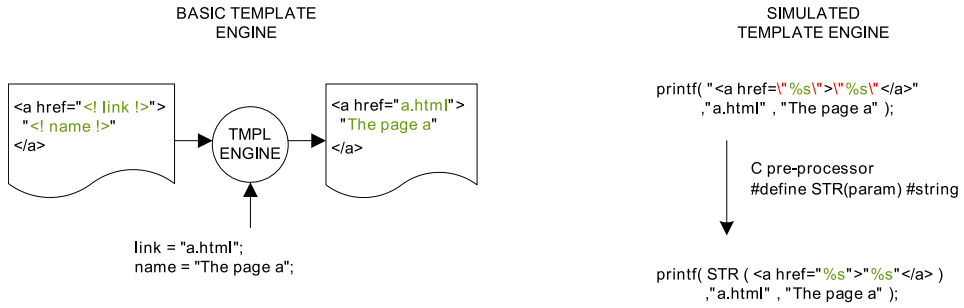



FIGURE 6.5 – Simulation d'un moteur de modèles basique

Ce mécanisme a été utilisé pour la génération de fichiers HTML. Le pseudo-modèle est implémenté dans la fonction *printHTML* du fichier *main.cpp* de l'application *app_publication* visible en annexe E.

### 6.4.2 Liste des fichiers et leurs fonctions

La liste suivante présente les fonctions permettant de générer les fichiers décrivant toutes les ressources de la plateforme.

#### Fichier de description de l'équipement

Ce fichier décrit la ressource `device12.hevs.ch`. Il contient les informations suivantes :

- la description de l'équipement (nom, position géographique, ...)
- la liste des capteurs de l'équipement
- la liste des références sur les ressources du niveau inférieur de l'arborescence
- la référence entre les deux types de documents

Les fonctions suivantes permettent de générer ce fichier dans chacune des représentations :

```
void genRDFSelfDescription()
void genHTMLSelfDescription()
```

Ces fonctions vont lire les informations stockées dans la base de données concernant la description de l'équipement ainsi que la liste des capteurs et de leurs paramètres puis génèrent le document avec ces informations. Elles ne prennent aucun paramètre car l'équipement est unique et donc tous les capteurs en font partie.

#### Fichier de description du capteur

Ce fichier décrit la ressource :

`device12.hevs.ch/measurement_item`

Il contient les informations suivantes :

- la liste des années durant lesquelles des mesures ont été réalisées
- la liste des observations qu'il peut faire
- la liste des références sur les ressources du niveau inférieur de l'arborescence
- la référence entre les deux types de documents

Les observations que peut faire le capteur sont définies dans l'ontologie *EMEP* [2]. Le système ne traite que trois types d'observations :

- *MinimizingActivity* correspondant au calcul de la valeur minimale
- *MaximizingActivity* correspondant au calcul de la valeur maximale
- *AveragingActivity* correspondant au calcul de la valeur moyenne

La fonction générant ce fichier, présentée ci-dessous, n'existe pas pour le HTML. En effet, le HTML ne décrit pas explicitement les observations que fait le capteur. Elles sont décrites dans les graphiques de statistiques discutés dans la section 3.1.2 page 18. Le fichier HTML correspondant est donc généré de la même façon que les documents décrivant la ressource `device12.hevs.ch/measurement_item/2009`.

```
void genRDFMeasItemDescription(const QByteArray * pMeasItem)
```

Cette fonction prend en paramètre un pointeur sur le type de mesure envoyé par la phase de traitement de l'URI. Elle récupère la liste des observations en appelant la fonction `getObservations`.

### Fichier de description de la période

Ce fichier décrit la ressource :

`device12.hevs.ch/measurement_item/2009`

Il contient les informations suivantes :

- la liste des références sur les ressources représentant l'activité du capteur dans la sous-période
- la référence entre les deux types de documents

Les fonctions suivantes génèrent ce fichier en RDF ou en HTML.

```
void genRDFPeriodDescription(const QList<QByteArray> *pParsedURI)
void genHTMLPeriodDescription(const QList<QByteArray> *pParsedURI)
```

Le paramètre `pParsedURI` est un pointeur sur l'URI requise qui permet à la fonction de connaître le type de mesure ainsi que la date requis pour ensuite pouvoir appeler la fonction `getPeriodList`.

#### Fichier contenant les valeurs des mesures

Ce fichier décrit la ressource :

`device12.hevs.ch/measurement_item/2009.values`

Il contient toutes les valeurs des mesures et quand elles ont été réalisées. Dans la représentation HTML, aucune référence sur le fichier RDF n'est faite. En effet, le fichier en format CSV contient exclusivement du texte brut et ne permet donc pas la référenciation. Les fonctions suivantes génèrent ce fichier en RDF ou en HTML.

```
void genRDFValues(const QList<QByteArray> *pParsedURI)
void genHTMLValues(const QList<QByteArray> *pParsedURI)
```

Le paramètre `pParsedURI` est un pointeur sur l'URI requise qui permet à la fonction de connaître le type de mesure ainsi que la date requis pour ensuite pouvoir appeler la fonction `getValues`.

#### Fichier contenant les valeurs statistiques

Ce fichier décrit la ressource :

`device12.hevs.ch/measurement_item/2009.stat`

Il contient les valeurs statistiques observées par le capteur. La liste des observations traitables par la plateforme est visible dans la sous-section 6.4.2. Les fonctions suivantes génèrent ce fichier en RDF ou en HTML.

```
void genRDFStats(const QList<QByteArray> *pParsedURI)
```

Cette fonction appelle la fonction `getStats` lui retournant la liste des valeurs statistiques dans la période et pour le capteur pointés par le paramètre `pParsedURI`.

```
void genHTMLStats(const QList<QByteArray> *pParsedURI)
```

Cette fonction prépare les paramètres à passer dans l'URL requise par l'*API Google Chart* [5] pour créer l'histogramme de statistiques à l'aide des informations retournées par la fonction `getStats`.

#### Fichier contenant l'histogramme des valeurs statistiques

Ce fichier décrit la ressource propre au HTML :

`device12.hevs.ch/measurement_item/2009.graph`

Il contient le graphique de statistiques représentant la valeur maximale, minimale et moyenne des mesures de chaque sous-période. La fonction suivante permet de générer ce fichier :

```
void genHTMLGraph(const QList<QByteArray> *pParsedURI)
```

Cette fonction prépare les paramètres nécessaires à la génération du graphique de statistiques à l'aide de l'*API Google Chart* [5].

### 6.4.3 Liste des fonctions de traîtement de l'information

La liste suivante décrit les fonctions utilisées pour récupérer les informations depuis la base de données. Elles sont appelées par les fonctions générant les fichiers décrivant les ressources.

### Récupération de la description de l'équipement

La fonction suivante permet de récupérer les informations concernant l'équipement stockées dans la table *Device_config* :

```
void getDeviceConfig(QMap<QByteArray, QByteArray> * pDeviceConfig)
```

Paramètre :

pointeur sur la liste des informations à remplir

### Récupération de la liste des capteurs et de leur description

La fonction suivante permet de récupérer la liste des capteurs ainsi que toutes les informations les décrivant dont leur position et les paramètres qu'ils mesurent :

```
void getSensorsInfos(QList<QByteArray> *pSensorsIDs, QList<QByteArray>
*pMeasTypes, QList<QByteArray> *pSensorsPos, QList<QByteArray>
*pParamNames, QList<QByteArray> *pSystemNames)
```

Paramètres :

1. pointeur sur la liste à remplir avec les identifiants des capteurs
2. pointeur sur la liste à remplir avec les types de mesure les définissant
3. pointeur sur la liste à remplir avec leurs positions (inside, outside)
4. pointeur sur la liste à remplir avec les paramètres qu'ils mesurent
5. pointeur sur la liste à remplir avec les systèmes dont ils font partie

Toutes ces informations sont retrouvées à l'aide d'une seule requête SQL joignant plusieurs tables.

### Récupération de la liste des années de mesures

La fonction suivante permet de récupérer la liste des années durant lesquelles le capteur a réalisé des mesures en interrogeant la table *Stat_years* :

```
void getAvailableYearsList(QList<int> *pYearsList, const QByteArray
*pMeasItem)
```

Paramètres :

1. pointeur sur la liste à remplir avec les années récupérées
2. pointeur sur le type de mesure

### Récupération de la liste des observations

Les fonctions suivantes permettent de récupérer la liste des observations :

```
void getObservations(QMap<int,int> *pSensors_obs, QMap<int,QByteArray>
*pObsList)
```

Paramètres :

1. pointeur sur la liste à remplir avec les identifiants des capteurs et des observations
2. pointeur sur la liste à remplir avec l'identifiant de l'observation et son nom

Cette fonction permet de récupérer la liste des observations pour chaque capteur. Pour ce faire, le contenu des tables *Sensor_obs* et *Observation* est retourné dans deux listes puis traité dans la fonction appelante afin de limiter les accès à la base de donnée. En effet, le capteur peut faire plusieurs observations d'où la nécessité d'avoir deux listes : une pour lier chaque identifiant du capteur avec l'identifant des observations qu'il réalise et l'autre, pour retrouver le nom de l'observation à partir de son identifiant.

```
void getObservations(QList<QByteArray> *pObslist, const QByteArray
*pMeasureItem)
```

Paramètres :

1. pointeur sur la liste à remplir avec les observations
2. pointeur sur le type de mesure

Cette fonction permet de récupérer la liste des observations pour un capteur.

### Récupération de la liste des ontologies

La fonction suivante permet de récupérer la liste de toutes ontologies stockées dans la table *ONTOLOGY* et utilisées pour former l'espace de nom (*namespace*) des documents RDF :

```
void getOntologies(QList<QByteArray> *pPrefixes, QList<QByteArray>
*pRefURIs)
```

Paramètres :

1. pointeur sur la liste à remplir avec les préfixes de chaque ontologie
2. pointeur sur la liste à remplir avec les URIs de chaque ontologie

### Définition des bornes de la période

La définition des bornes de la période a pour but de calculer jusqu'à quelle date s'étend la période demandée. Si la période demandée est `2009-04`, la date de fin de période est `2009-05-01T00-00-00` non comprise. Ces informations sont nécessaires pour construire les requêtes SQL qui doivent permettre de récupérer toutes les mesures ou encore toutes les périodes dans un intervalle de temps. La fonction suivante réalise ce calcul :

```
int getCoveredPeriod(const QByteArray *pISODate)
```

Paramètre :

pointeur sur la date spécifiée dans l'URI

Retour :

limite non comprise de la période en format *Unix timestamp*

La valeur de retour est en format *Unix timestamp* afin de pouvoir l'utiliser dans une requête SQL, la base de données stockant les dates dans ce format.

### Récupération de la liste des périodes

La liste des périodes peut être récupérée à partir des tables de statistiques ou de la table des mesures selon la période spécifiée dans l'URI. La fonction suivante implémente la récupération de la liste des périodes :

```
void getPeriodList(QList<int> *pPeriodList, const QByteArray *pMeasItem,
const QByteArray *from, const int to)
```

Paramètres :

1. pointeur sur la liste des périodes à remplir
2. pointeur sur le type de mesure
3. pointeur sur la date requise
4. pointeur sur la date limite (format *Unix timestamp*)

Le paramètre `from` permet de définir dans quelle table doit être récupérée la liste des périodes. Si par exemple la date contient une année et un mois, l'algorithme de la fonction va déduire que les périodes recherchées sont des jours et va donc lire les jours correspondant au type de mesure dans la table *Stat_days* de la base de données. Il représente également la borne inférieure de l'intervalle dans lequel doivent être retournées les valeurs, le paramètre `to` représentant la borne supérieure.

### Récupération de la liste des statistiques

La liste des statistiques peut être récupérée par la fonction suivante directement à partir des tables de statistiques ou de la table des mesures selon la période spécifiée dans l'URI :

```
void getStats(QList<double> *pStats, const QByteArray *pMeasItem, const
QByteArray *pPeriod)
```

Paramètres :

1. pointeur sur la liste des statistiques à remplir
2. pointeur sur le type de mesure
3. pointeur sur la date requise

Cette fonction retourne les valeurs statistiques des mesures du capteur, défini par le paramètre `pMeasItem`, dans l'intervalle de temps calculé à partir du paramètre `pPeriod` et de la fonction `getCoveredPeriod`.

### Récupération de la liste des mesures

La liste des mesures peut être récupérée à partir de la table *Measurements*. Elle contient la période dans laquelle chaque mesure a été effectuée et sa valeur. La fonction suivante permet de générer cette liste :

```
void getValues(QList<int,double> *pValues, QByteArray *pMeasItem, const
int from, cons int to)
```

Paramètres :

1. pointeur sur la liste des mesures à remplir
2. pointeur sur le type de mesure
3. date en format *Unix timestamp* représentant le début de l'intervalle
4. date en format *Unix timestamp* représentant la fin de l'intervalle

Cette fonction retourne les valeurs mesurées par le capteur représenté par `pMeasItem` dans l'intervalle de temps allant de la date `from` à la date `to` non comprise.

### Récupération de l'unité de la mesure

Les unités sont utilisées dans les documents RDF pour définir explicitement les valeurs des mesures. La fonction suivante permet de récupérer l'unité mesurée par le capteur :

```
void getMeasureUnit(QByteArray *pUnit, const QByteArray *pMeasItem)
```

Paramètres :

1. pointeur sur la variable à remplir avec l'unité
2. pointeur sur le type de mesure

### Récupération de la configuration des graphiques

La fonction suivante permet de récupérer la plage de valeurs permettant de construire les axes des graphiques utilisés dans la représentation HTML :

```
void getRange(QList<int> *pRange, const QByteArray *pMeasureItem)
```

Paramètres :

1. pointeur sur la liste à remplir avec les valeurs définissant la plage de mesures
2. pointeur sur le type de mesure

### Conversion d'un date au format ISO en un *UNIX timestamp*

La conversion d'une date au format ISO en un format *Unix timestamp* est nécessaire pour pouvoir l'utiliser dans les requêtes exécutées sur la base de données. La fonction suivante réalise cette conversion :

```
int getUnixTimestamp(const QByteArray *pDate)
```

Paramètre :

pointeur sur la date à convertir

Retour :

date convertie au format *Unix timestamp*

### Conversion d'un *UNIX timestamp* en date au format ISO

La conversion d'une date au format *Unix timestamp* en format ISO est nécessaire lors de la récupération de toute date depuis la base de données qui doit être utilisée dans des références, celles-ci étant des URIs contenant des dates au format ISO. La fonction suivante réalise cette conversion :

```
QByteArray getISOTimeStamp(const int UNIXtimestamp, int level)
```

Paramètre :

1. date à convertir
2. niveau de la date à retourner

Retour :

1. date convertie au format *Unix timestamp*

Le paramètre `level` autorise des valeurs de 1 à 6. Voici les différentes dates retournées en fonction du niveau :

- niveau 1 : "yyyy"
- niveau 2 : "yyyy-MM"
- niveau 3 : "yyyy-MM-dd"
- niveau 4 : "yyyy-MM-ddThh"
- niveau 5 : "yyyy-MM-ddThh-mm"
- niveau 6 : "yyyy-MM-ddThh-mm-ss"

#### Comptage des mesures

Le comptage des mesures est utilisé pour générer l'histogramme de statistiques. La fonction suivante permet de compter les mesures dans un intervalle de temps et de valeurs.

```
int getCount(const QByteArray *pMeasItem, const int from, const int to,
const int fromTS, const int toTS)
```

Paramètres :

1. pointeur sur le type de mesure
2. nombre représentant le début de l'intervalle des valeurs
3. nombre représentant la fin de l'intervalle des valeurs
4. date au format *Unix timestamp* représentant le début de l'intervalle du temps
5. date au format *Unix timestamp* représentant la fin de l'intervalle du temps

Retour :

1. nombre de mesures comptées

Cette fonction retourne le nombre de mesures dont la valeur est comprise entre `from` et `to` non compris et dans l'intervalle de temps allant de `fromTS` à `toTS` non compris.

## 6.5 Error handler

La gestion des erreurs retourne au client le status HTTP représentant l'erreur qui s'est produite et un document HTML la détaillant.

```
void errorHandler(int errorCode, int errorType)
```

Paramètres :

1. code de l'erreur correspondant au status HTTP
2. type de l'erreur

Le code de l'erreur définit le type de status à renvoyer, par exemple *404 Not Found* dans le cas d'un document non trouvé. Le type d'erreur permet de décrire l'erreur qui s'est produite. L'énumération suivante est déclarée dans le fichier d'en-tête de l'application de publication (annexe E) et répertorie toutes les erreurs qui peuvent se produire :

`enum` **`eError`**

1. ERR_URI : URI mal formée
2. ERR_EMPTYDEVICE : l'équipement n'a aucun capteur
3. ERR_BADMEASITEM : le capteur demandé n'existe pas
4. ERR_NODATA : aucune donnée n'a été trouvée

CHAPITRE 7

# Classe d'acquisition

La classe d'acquisition *PollDaemon* permet d'ordonner une acquisition à un moment donné et de communiquer avec la base de données. Le code source l'implémentant est visible en annexe F. Elle rend abstrait tout le processus de lecture et d'écriture des mesures dans la base de données au processus d'acquisition appelé également démon d'acquisition comme le montre la figure 7.1. Un démon d'acquisition est un processus exécuté en tâche de fond.

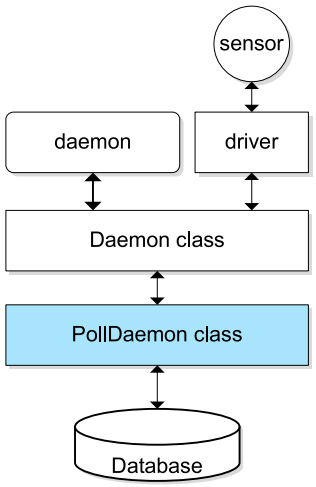



FIGURE 7.1 – Processus d'acquisition

La classe du démon d'acquisition permet de communiquer avec le capteur et avec la classe d'acquisition. Cette classe étant dépendante du type de capteur et de ses drivers, elle n'est pas implémentée dans la plateforme. Le diagramme de séquence de la figure 7.2 décrit le déroulement d'acquisition.

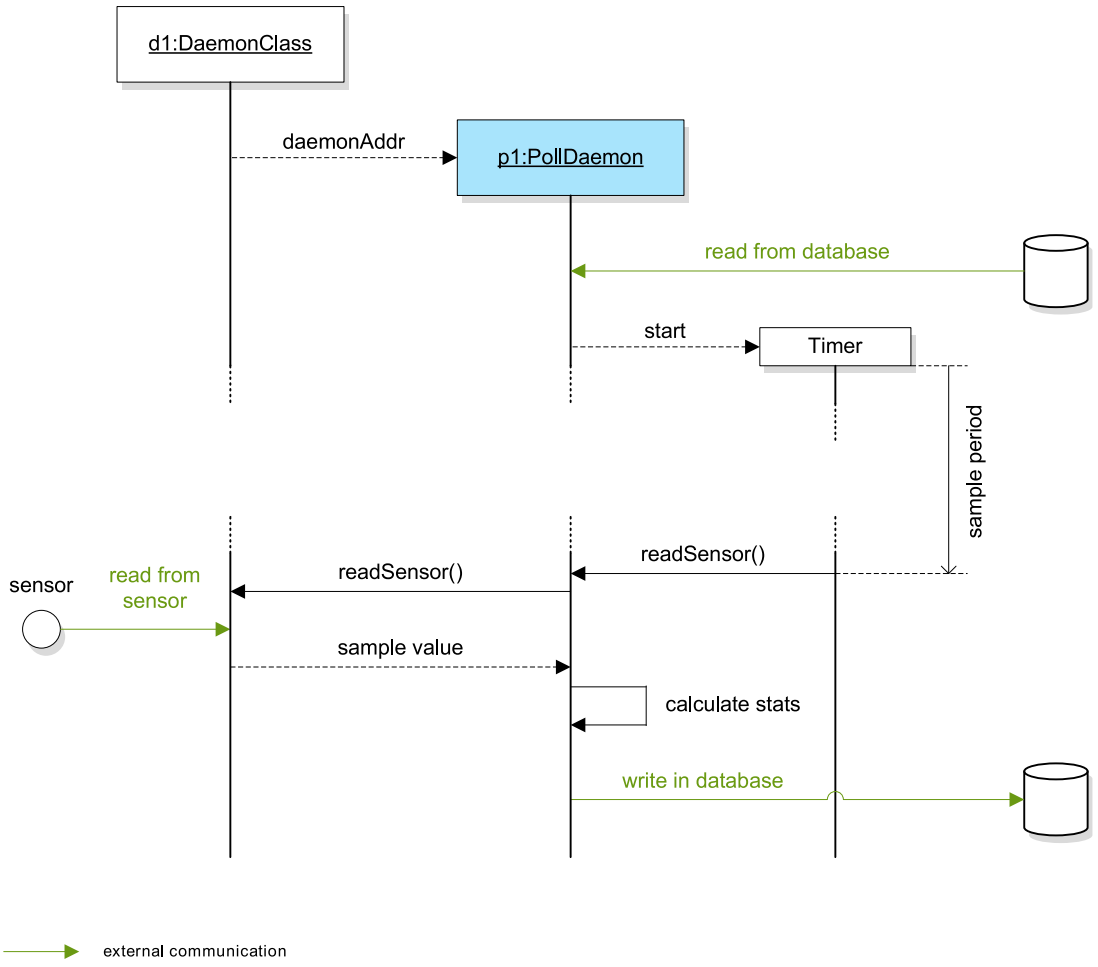



FIGURE 7.2 – Diagramme de séquence d'acquisition

La classe récupère la période d'acquisition *sample period* pour le capteur concerné et démarre un timer qui envoie un signal appelant la fonction `readSensor` à chaque période d'acquisition. Cette fonction lit la valeur du capteur et la retourne. Elle est ensuite stockée dans la base de données et les statistiques sont calculées, puis mises à jour. Ce cycle recommence à chaque fois que le timer atteint la période d'acquisition. La fonction `readSensor` est virtuelle et doit être implémentée dans la classe du démon car c'est elle qui permet de communiquer avec le capteur à travers le driver.

```
virtual double readSensor()
```

Retour :

valeur de la mesure

## 7.1 Utilisation

Pour que le démon puisse écrire les mesures dans la base de données, sa classe doit hériter de la classe *PollDaemon* et implémenter la fonction `readSensor` comme le montre la figure 7.3.

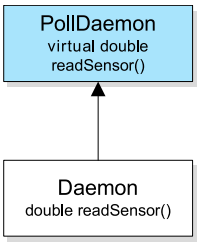



FIGURE 7.3 – Héritage de la classe *PollDaemon*

La classe *PollDaemon* requiert l'adresse du capteur afin de pouvoir l'identifier dans la liste de capteurs contenue dans la table *SENSOR* de la base de données. Pour ce faire, la classe du démon doit appeler le constructeur de la classe *PollDaemon* en passant en paramètre l'adresse du capteur. Ceci doit être fait comme suit :

```
1 #include "polldaemon.h"
2
3 class DaemonClass : public PollDaemon
4 {
5   DaemonClass(...) : PollDaemon( daemonAddr )
6   {
7     ...
8 };
```

où `daemonAddr` est un `const char *` pointant sur l'adresse du démon. À la création d'un objet de la classe *DaemonClass*, une instance de la classe *PollDaemon* va être créée et va lancer le processus d'acquisition.

CHAPITRE 8

# Optimisation

## 8.1 Utilisation des ressources

### 8.1.1 Passage de paramètres par référence

Afin de limiter au maximum l'utilisation mémoire de l'application de publication, tous les passages de paramètres concernant des chaînes de caractères ont été faits par référence. Ceci évite une copie locale du paramètre qui n'est pas concevable dans une application manipulant des chaînes de caractères.

### 8.1.2 Optimisation des documents RDF

Comme le montre le chapitre 9, les documents RDF peuvent être volumineux et donc utiliser beaucoup de mémoire. La taille de ces documents a été réduite en utilisant les entités XML (*ENTITY*). Elles permettent de donner un diminutif aux chaînes de caractères redondantes dans le document. Un exemple d'utilisation pourrait être le remplacement de l'adresse de l'équipement, `http://device12.hevs.ch`, qui, dans certains cas, peut être longue, par *dp* (pour *data provider*, fournisseur de données). Cette modification parait anodine en terme de gain d'espace mémoire mais dans un document où l'adresse de l'équipement reviendrait plusieurs centaines de fois, le gain passerait de quelques bytes à plusieurs KBytes pour une seule entité.

## 8.2 Performances

### 8.2.1 Base de données

La base de données est un point critique de la performance du système. Si celle-ci n'est pas bien conçue, un grand nombre d'enregistrements entraîne une augmentation considérable du temps de réponse des requêtes. La table *Measurement* peut contenir un nombre élevé d'enregistrements, qui peut être représenté par l'équation suivante dans le cas d'un système où les valeurs physiques sont mesurées régulièrement dans un intervalle de temps de quelques minutes :

$$nb_{meas} = nb_{sensors} \cdot nb_{hoursperday} \cdot nb_{measperhour} \cdot [(nb_{bisyears} \cdot 366) + (nb_{years} \cdot 365)]$$

En partant du principe que le système est composé de dix capteurs mesurant six valeurs par heure et est actif depuis cinq ans (une année bissextile), le nombre d'enregistrement s'éléve à *2'629'440*. Dans une base de données simple, un requête calculant la valeur moyenne des mesures dans un intervalle de temps doit parcourir tous les enregistrements les uns après les autres, y compris ceux qui ne sont pas pris en compte par la condition de la requête. Ceci ralentit considérablement le temps d'exécution de l'application. L'ajout de tables clés ainsi que de mécanismes peuvent augmenter les performances.

### Tables de statistiques

Les tables de statistiques ont été conçues pour stocker les valeurs statistiques dans la base de donnée et donc éviter de parcourir la table des mesures pour effectuer les calculs statistiques. Cette solution est plus performante mais utilise plus de ressources mémoire étant donné que les valeurs statistiques pour chaque période doivent être stockées.

Un compromis a été trouvé pour améliorer la performance du système tout en évitant de trop surcharger la base de données : il a été choisi de ne stocker que les valeurs statistiques des années, mois et jours. En effet, en se basant sur le système ci-dessus, la table de statistiques journalières *Stat_days* contiendrait 83'955 enregistrements, soit environ trente et une fois moins d'enregistrements à parcourir que dans la table des mesures, ce qui reste raisonnable en terme de performance et d'utilisation mémoire.

### Mécanisme d'indexation

La plupart des moteurs de base de données offrent la possibilité de créer des indexes. Ces indexes permettent de trier des tables selon un ou plusieurs champs. Lors de la création d'un index, les champs indexés sont copiés. Ce mécanisme offre un gain de performance dans le cas de tables contenant beaucoup d'enregistrements. Il ralentit cependant l'insertion d'enregistrements dans la table étant donné qu'il doit à la fois la remplir et maintenir à jour la table d'indexation.

Une indexation de la table de mesures sur les champs contenant l'identifiant du capteur et contenant la date de mesure a été créée. Grâce à cette indexation, le moteur de base de données peut optimiser la recherche d'enregistrements. La comparaison entre la recherche dans une table non indexée et dans une table indexée est illustrée dans la figure 8.1.

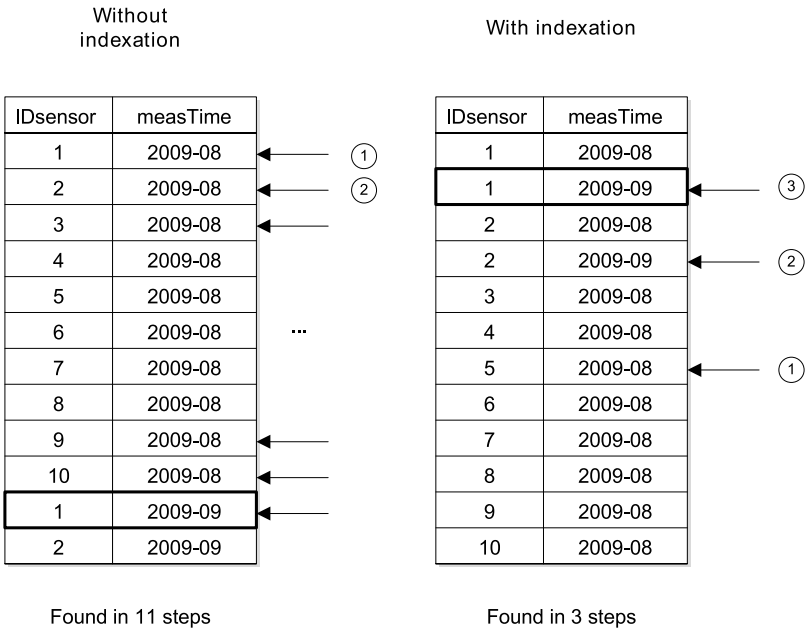



FIGURE 8.1 – Recherche dans une table non indexée et dans une table indexée

Dans la table non indexée, tous les enregistrements doivent être parcourus par le moteur de base de données tandis que dans la table indexée, le moteur de recherche peut, à l'aide d'un algorithme, estimer l'emplacement de l'enregistrement par rapport à l'endroit de la table sur lequel il pointe. Le nombre d'étapes dépend de l'algorithme implémenté. Le gain en termes d'étapes de recherche est significatif.

# CHAPITRE 9

# Tests et résultats

Tous les tests effectués sur la plateforme ont été faits sur une machine locale. Il est nécessaire de prendre en considération que les tests de performance peuvent différer par rapport à un test dans des conditions réelles (système embarqué avec accès client depuis le Web).

## 9.1 Tests de limitations

Les tests de limitations sont réalisés afin de définir l'emprunte mémoire du système et d'être sûr que la mémoire utilisée par les applications est entièrement restituée. Ceci est fait dans le but de pouvoir fixer les limitations en termes de ressources du système, selon la cible sur laquelle il est implémenté.

### 9.1.1 Emprunte mémoire

Le test de l'emprunte mémoire du système permet de déterminer l'espace mémoire maximum utilisé par les éléments le composant. L'emprunte mémoire des applications a été mesurée à l'aide de l'outil *top*. Ce dernier permet de voir la mémoire résidente utilisée par l'application. La mémoire résidente comprend la mémoire allouée pour le code, la mémoire allouée pour les données utilisées par l'application ainsi que la taille de la pile (*Stack size*).

#### Application de publication

L'emprunte mémoire de l'application de publication en état d'attente est de 4,6 Mbytes. Cette application traitant des données, son emprunte mémoire maximale dépend donc de la taille des données qu'elle génère.

Le fichier le plus volumineux retournable par l'application est celui contenant la liste des valeurs d'une année au format RDF. Sa taille est de 15 Mbytes. Un requête sur l'URI identifiant la ressource décrite par ce document a été effectuée. Le résultat montre que l'application utilise 10 Mbytes de mémoire résidente.

La taille des données générées par l'application (environ 5,4 Mbytes) est donc inférieure à la taille du document retourné. Ceci est dû au fait que seules les données récupérées depuis la base de données sont stockées en mémoire. Tout le balisage du document RDF est généré à la volée à l'aide de la fonction *printf*.

Il est possible d'en déduire que les données générées à l'aide de la fonction *printf* sont stockées hors de la mémoire attribuée à l'application de publication.

#### Classe d'acquisition

Il n'est pas possible de tester l'emprunte mémoire utilisée uniquement par la classe d'acquisition. En effet, pour être exécutée, cette dernière doit être héritée par une autre classe. Pour avoir un orde de grandeur, l'emprunte mémoire de l'application simulant l'acquisition a été testée (voir annexe I). Elle utilise 4,6 Mbytes de mémoire.

#### Serveur Web

Le serveur Web en état d'attente consomme 1 Mbytes de mémoire. Son emprunte mémoire maximale dépend du nombre de requêtes en attente ainsi que du nombre de données qu'il doit retourner. Il est donc difficile de la déterminer.

L'emprunte mémoire du serveur Web, dans le cas d'une requête demandant les valeurs d'une année au format RDF (fichier le plus volumineux), est de 15 Mbytes. Il semble que le serveur Web ne restitue pas la mémoire qu'il a allouée. En effet, un document plus léger a été requis et il a été constaté que la mémoire consommée par le serveur restait à 15 Mbytes. Il est nécessaire pour la suite du projet d'analyser la provenance de cette consommation de mémoire.

#### Base de données

L'emprunte mémoire de la base de données dépend du nombre d'enregistrements qu'elle contient. Pour donner un ordre d'idée, une base de données contenant dix années de mesures pour six capteurs, soit environ trois millions de mesures, a une emprunte mémoire de 150 MBytes.

### 9.1.2 Restitution de la mémoire

La restitution de la mémoire peut être testée à l'aide de l'outil *Valgrind* [13]. Le test se fait à l'aide de la commande suivante où l'option `--leak-check` permet de tester les fuites de mémoire :

```
valgrind --leak-check=summary application_bin application_parameters
```

#### Application de publication

L'application de publication étant exécutée sur un serveur et réagissant à des requêtes HTTP, il n'est pas possible de la tester directement à l'aide de *Valgrind* [13]. Il est nécessaire de modifier l'application pour en faire un simple exécutable prenant en paramètre les données passées par le flux standard *stdin*. La partie modifiée de l'application de publication pour autoriser ce test est visible en annexe G.

Toute l'architeture de la plateforme a été testée afin de contrôler qu'aucune des fonctions de l'application n'engendre des fuites de mémoire. Pour chaque test, le résultat suivant montre que toute la mémoire utilisée a bien été restituée à la fin de l'exécution de l'application :

```
definitely lost: 0 bytes in 0 blocks.
  possibly lost: 0 bytes in 0 blocks.
```

#### Classe d'acquisition

La classe d'acquisition a été testée à l'aide d'une simple fonction l'instanciant. Aucune fuite de mémoire n'a été détectée comme le montre le résultat suivant :

```
definitely lost: 0 bytes in 0 blocks.
  possibly lost: 0 bytes in 0 blocks.
```

## 9.2 Tests de performance

La performance du système dépend du temps d'accès à la base de données ainsi que du temps de traitement des données. Il est difficile de définir précisément le délai de réponse de la plateforme lors d'une requête client car il dépend de la taille des documents retournés.

Il a été décidé de faire un comparatif de performance entre une base de données légère et une base de données chargée afin de voir l'influence du nombre de données sur le temps de réponse du système. La base de données légère *Light DB* contient une année de mesures pour un capteur, soit 52'560 enregistrements, et la base de données chargée *Heavy DB* contient dix années de mesures pour six capteurs, soit 3'155'328 enregistrements.

Le temps de réponse de la plateforme a été testé à l'aide d'une application effectuant une requête HTTP sur celle-ci. Le temps de réponse correspond au temps écoulé depuis la requête jusqu'à la réception du dernier byte de la réponse. Le code source de cette application est visible en annexe H. Le tableau suivant montre le temps de réponse du système en fonction du nombre de mesures retournées et de la taille de la base de données. 52'650 représente le nombre de mesures pour une année, 4'644, le nombre de mesures pour un mois et 144, le nombre de mesures pour un jour.

| Number of measurements | Request response time [ms] | |
|---|---|---|
| | Light DB | Heavy DB |
| 52'650 | 2'530 | 2'652 |
| 4'464 | 350 | 358 |
| 144 | 105 | 118 |

La faible différence de temps de réponse entre ces deux bases de données montre que le nombre d'enregistrements influence peu le temps de réponse du système. Il est donc possible de constater que le temps de traitement des données définit principalement le temps de réponse du système.

## 9.3 Tests fonctionnels

Les tests fonctionnels ont pour but de voir si les documents RDF et HTML générés permettent de parcourir l'arborescence des ressources jusqu'à l'obtention de la valeur de la mesure et de tester la réponse du système dans le cas d'URIs erronées ou identifiant une ressource inexistante.

### 9.3.1 Test de la publication des données

Le test a été effectué pour les deux représentations en partant du document représentant l'équipement. À partir de ce document, toute l'arborescence a été parcourue en utilisant les références contenues dans les documents jusqu'à la valeur de la mesure. Les documents RDF retournés décrivant chaque ressource sont visibles en annexe J. La structure est fonctionnelle.

### 9.3.2 Test de validité des documents RDF générés

Toutes les documents RDF décrivant les ressources ont été validés à l'aide du validateur RDF du *W3C* [3].

### 9.3.3 Test de la validité des URIs

Le tableau suivant présente les résultats des tests effectués sur toutes les URIs de la structure de la plateforme avec les deux types de contenu acceptés. Les tests étant effectués en local, l'équipement est identifié par `http://localhost/` et abrégé en `/` pour une question de lisibilité. Le capteur *IndoorTemperature* est considéré comme existant dans le système.

| URI | Accept | Status code | Test result |
|---|---|---|---|
| `/` | RDF | 200 OK | OK |
| `/IndoorTemperature` | RDF | 200 OK | OK |
| `/IndoorTemperature/2009` | RDF | 200 OK | OK |
| `/IndoorTemperature/2009.values` | RDF | 200 OK | OK |
| `/IndoorTemperature/2009.stat` | RDF | 200 OK | OK |
| `/html/IndoorTemperature` | RDF | 404 Not Found | OK |
| `/` | HTML | 200 OK | OK |
| `/html/IndoorTemperature` | HTML | 200 OK | OK |
| `/html/IndoorTemperature/2009` | HTML | 200 OK | OK |
| `/html/IndoorTemperature/2009.values` | HTML | 200 OK | OK |
| `/html/IndoorTemperature/2009.stat` | HTML | 200 OK | OK |
| `/html/IndoorTemperature/2009.graph` | HTML | 200 OK | OK |
| `/IndoorTemperature` | HTML | 303 See Other | OK |
| `/wrong/uri/format` | both | 404 Not Found | OK |
| `/IndoorTemperature/3000` | RDF | 404 Not Found | OK |
| `/html/IndoorTemperature/3000` | HTML | 404 Not Found | OK |

Les résultats montrent qu'une réponse est à chaque fois renvoyée et que le status est correct pour chaque cas de figure. Le status *303 See Other* est bien renvoyé dans le cas de la négociation de contenu.

### 9.3.4 Test de la classe d'acquisition

La classe d'acquisition a été testée à l'aide d'une application simulant un processus d'aquisition visible en annexe I. Cette application simule plusieurs démons d'acquisition qui implémentent la fonction virtuelle `readSensor` de la classe d'acquisition et retournent une valeur numérique aléatoire entre -20 et 40 représentant la mesure. Il a été constaté que la classe d'acquisition fait correctement son travail d'ordonnancement d'acquisition de mesure. Elle exécute également correctement tout le processus d'insertion de la mesure dans la base de données ainsi que le calcul et la mise à jour des valeurs statistiques à chaque acquisition.

# CHAPITRE 10

# Limites de la plateforme

## 10.1 Flexibilité

### 10.1.1 Résolution des capteurs

Le système actuel a une résolution maximale d'une minute. La structure de navigation ne permet pas d'accéder à des mesures effectuées plus d'une fois par minute. En règle générale, dans le domaine des mesures physiques pour l'analyse environnementale, une résolution d'une minute est suffisante. Mais dans l'hypothèse où des capteurs effectuant des mesures à plus haute résolution devraient être utilisés, le système doit être adapté et le code source modifié.

### 10.1.2 Calculs statistiques

Le système réalisé permet l'ajout de capteurs sans avoir à modifier le code source. Cependant, les observations que peuvent faire ces capteurs sont prédéfinies. En effet, le système actuel n'est pas capable de retourner des valeurs statistiques autres que la valeur maximale, la valeur minimale et la valeur moyenne d'un groupe de mesures. L'ajout d'un calcul de statistique nécessite une modification du code source.

## 10.2 Utilisation des ressources

L'emprunte mémoire de la plateforme dépend du nombre de mesures stockées et donc du nombre de capteurs ainsi que de la taille du document retourné. L'emprunte mémoire de l'application de publication, de la classe d'acquisition ainsi que du document retournable le plus volumineux est présentée dans le chapitre 9. Les limitations des ressources mémoires dépendent de l'espace mémoire du système embarqué ainsi que du système d'exploitation et des outils présents sur celui-ci. Les limites ne pouvant pas être clairement définies, il incombe donc au lecteur de prendre en considération les valeurs exposées lors des tests et de définir les limitations en fonction du système utilisé.

# CHAPITRE 11

# Conclusion

Ce projet a abouti à une plateforme évolutive capable de publier toutes les informations relatives aux mesures ainsi qu'aux capteurs les ayant réalisées et à la plateforme elle-même. La publication est faite en RDF et en HTML pour rendre l'information interprétable autant par un client humain que par une application client.

La plateforme a été développée pour être performante afin de réduire au maximum son temps de réponse. Les résultats sont convaincants car il ne lui faut que trois secondes pour retourner toutes les mesures d'une année, soit 52'650 valeurs. Elle a également été conçue pour être la plus évolutive possible. En effet, l'ajout de capteurs peut se faire facilement à travers la base de données sans avoir à modifier le code source des applications.

Une classe d'acquisition communiquant avec la base de données et ordonnant l'acquisition des données a également été développée. Cette classe sera utilisable lors de l'implémentation de la partie d'acquisition de la plateforme.

Une application simulant un processus d'acquisition ajoutant une mesure aléatoire dans la base de données à travers la classe d'acquisition et à chaque intervalle de temps a été réalisée. Cette simulation permet d'avoir une vision du système proche de la réalité et de pouvoir tester toute la plateforme de publication. En effet, les mesures ajoutées dans la base de données sont fictives mais cohérentes.

La plateforme a été développée et testée sur une machine locale mais n'a pas été implantée sur un système embarqué par manque de temps. Cette implantation fait partie des tâches à effectuer.

Le démonstrateur n'as pas été implémenté par manque de temps. Ce démonstrateur peut être réalisé à l'aide d'une interface HTML derrière laquelle tourne une application communiquant avec l'application de publication pour obtenir les mesures réalisées par chaque capteur. À partir des données récupérées, cette application pourrait générer des graphiques de comparaison entre, par exemple, la température intérieure et extérieure mesurée par deux capteurs durant une période.

## 11.1 Tâches à effectuer

Le stade actuel de la plateforme consiste à la publication des données sur le Web ainsi qu'à l'ordonnancement de l'acquisition des données. Il reste donc à développer toute la partie effectuant l'acquisition réelle des données. Cette partie concerne principalement l'implémentation des processus d'acquisition et des drivers permettant de communiquer avec les capteurs.

Il est également nécessaire d'implanter la plateforme sur un système à ressources limitées pour pouvoir faire des tests dans un environnement réel. L'objectif de cette implantation est d'installer un système d'exploitation embarqué optimisé pour les tâches que la plateforme doit effectuer.

L'objectif final est d'avoir un système embarqué performant, autonome et dédié exclusivement à l'acquisition et à la publication de données de mesures sur le Web. Une idée pour l'amélioration des performances est de compiler le serveur Web ainsi que le moteur de base de données avec des options d'optimisation. *Lighttpd* [6] et *SQLite* [7] offrent de telles possibilités.

Vionnaz, le 5 juillet 2009
Johann Seydoux

# Bibliographie


[1] S-TEN Project. [En ligne] 1er juillet 2009.
<http://www.s-ten.eu>

[2] Environmental Monitoring and Engery Performance (EMEP). Purpose. [En ligne] 1er juillet 2009.
<http://emep.hevs.ch/emep-purpose.html>

[3] World Wide Web Consortium (W3C). World Wide Web Consortium - Web Standards. [En ligne] 5 juillet 2009.
<http://www.w3.org>

[4] Cool URIs for the Semantic Web. [En ligne] 1er juillet 2009.
<http://www.w3.org/TR/2007/WD-cooluris-20071217>

[5] Google Chart API. Developer's Guide - Google Chart API - Google Code. [En ligne] 1er juillet 2009.
<http://code.google.com/intl/fr/apis/chart/>

[6] LIGHTTPD (*lighty*). lighttpd fly light. [En ligne] 2 mai 2009.
<http://www.lighttpd.net>

[7] SQLite. SQLite Home Page. [En ligne] 2 mai 2009.
<http://www.sqlite.org/>

[8] The Common Gateway Interface (CGI). [En ligne] 3 mai 2009.
<http://hoohoo.ncsa.illinois.edu/cgi/>

[9] Fast Common Gateway Interface (FastCGI). [En ligne] 3 mai 2009.
<http://www.fastcgi.com>

[10] Apache Bench (ab). ab - Apache HTTP server benchmarking tool. [En ligne] 11 mai 2009.
<http://httpd.apache.org/docs/2.0/programs/ab.html>

[11] Qt. Qt - A cross-platform application and UI framework. [En ligne] 18 mai 2009.
<http://www.qtsoftware.com/>

[12] HTTP/1.1 Header Fields Definition. [En ligne] 6 juin 2009.
<http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html>

[13] Valgrind. Valgrind Home. [En ligne] 5 juillet 2009.
<http://valgrind.org>

# ANNEXE A

# Fichiers de configuration de Lighttpd

## A.1 lighttpd.conf

```
# Debian lighttpd configuration file
#

############ Options you really have to take care of ####################

## modules to load
# mod_access, mod_accesslog and mod_alias are loaded by default
# all other module should only be loaded if neccesary
# - saves some time
# - saves memory

server.modules       = (
   "mod_access",
   "mod_alias",
   "mod_accesslog",
      "mod_compress",
    "mod_fastcgi"
)

## a static document-root, for virtual-hosting take look at the
## server.virtual-* options
server.document-root = "/var/www/"

## where to upload files to, purged daily.
server.upload-dirs = ( "/var/cache/lighttpd/uploads" )

## where to send error-messages to
server.errorlog     = "/var/log/lighttpd/error.log"

## files to check for if .../ is requested
index-file.names    = ( "index.php", "index.html",
                        "index.htm", "default.htm",
                        "index.lighttpd.html" )

#### accesslog module
accesslog.filename  = "/var/log/lighttpd/access.log"

## deny access the file-extensions
#
# ~  is for backupfiles from vi, emacs, joe, ...
# .inc is often used for code includes which should in general not be part
#    of the document-root
url.access-deny     = ( "~", ".inc" )

##
# which extensions should not be handle via static-file transfer
#
# .php, .pl, .fcgi are most often handled by mod_fastcgi or mod_cgi
static-file.exclude-extensions = ( ".php", ".pl", ".fcgi" )


######### Options that are good to be but not neccesary to be changed #######

## Use ipv6 only if available.
include_shell "/usr/share/lighttpd/use-ipv6.pl"

```

```
57 ## to help the rc.scripts
58 server.pid-file     = "/var/run/lighttpd.pid"
59
60 ## virtual directory listings
61 dir-listing.encoding = "utf-8"
62 server.dir-listing   = "enable"
63
64 ## change uid to <uid> (default: don't care)
65 server.username     = "www-data"
66
67 ## change uid to <uid> (default: don't care)
68 server.groupname    = "www-data"
69
70 #### compress module
71 compress.cache-dir   = "/var/cache/lighttpd/compress/"
72 compress.filetype    = ("text/plain", "text/html", "application/x-javascript", "text/css")
73
74 #### external configuration files
75 ## mimetype mapping
76 include_shell "/usr/share/lighttpd/create-mime.assign.pl"
77
78 ## load enabled configuration files,
79 ## read /etc/lighttpd/conf-available/README first
80 include_shell "/usr/share/lighttpd/include-conf-enabled.pl"
81
82 #### handle Debian Policy Manual, Section 11.5. urls
83 ## by default allow them only from localhost
84 ## (This must come last due to #445459)
85 ## Note: =~ "127.0.0.1" works with ipv6 enabled, whereas == "127.0.0.1" doesn't
86 $HTTP["remoteip"] =~ "127.0.0.1" {
87   alias.url += (
88     "/doc/" => "/usr/share/doc/",
89     "/images/" => "/usr/share/images/"
90   )
91   $HTTP["url"] =~ "^/doc/|^/images/" {
92     dir-listing.activate = "enable"
93   }
94 }
```

## A.2 mod_cgi.conf

```
1  ## CGI programs allow you to enhance the functionality of the server in a very
2  ## straight and simple way..
3  ##
4  ## Documentation: /usr/share/doc/lighttpd-doc/cgi.txt
5  ##            http://www.lighttpd.net/documentation/cgi.html
6
7  server.modules += ( "mod_cgi" )
8
9  $HTTP["remoteip"] =~ "127.0.0.1" {
10   alias.url += ( "/cgi-bin/" => "/usr/lib/cgi-bin/" )
11   $HTTP["url"] =~ "^/cgi-bin/" {
12     cgi.assign = ( "" => "" )
13   }
14 }
15
16 $HTTP["url"] =~ "^/cgi-bin/" {
17   cgi.assign = ( "" => "" )
18 }
```

## A.3 mod_fastcgi.conf

```
1  ## FastCGI programs have the same functionality as CGI programs,
2  ## but are considerably faster through lower interpreter startup
3  ## time and socketed communication
4  ##
5  ## Documentation: /usr/share/doc/lighttpd-doc/fastcgi.txt.gz
```

```
6  ##            http://www.lighttpd.net/documentation/fastcgi.html
7
8  server.modules += ( "mod_fastcgi" )
9
10 fastcgi.server = (
11   #"hello.fcgi" =>
12   #((
13   # "host" => "127.0.0.1",
14   # "port" => 1026,
15   # "bin-path" => "/usr/lib/fcgi-bin/hello.fcgi",
16   # "check-local" => "disable",
17   # "max-procs" => 1
18   #)),
19   #"dbtest.fcgi" =>
20   #((
21   # "host" => "127.0.0.1",
22   # "port" => 1027,
23   # "bin-path" => "/usr/lib/fcgi-bin/dbtest.fcgi",
24   # "check-local" => "disable",
25   # "min-procs" => 1,
26   # "max-procs" => 1
27   #)),
28   #"loop.fcgi" =>
29   #((
30   # "host" => "127.0.0.1",
31   # "port" => 1028,
32   # "bin-path" => "/usr/lib/fcgi-bin/loop.fcgi",
33   # "check-local" => "disable",
34   # "max-procs" => 1
35   #)),
36   "/" =>
37   ((
38     "host" => "127.0.0.1",
39     "port" => 1029,
40     "bin-path" => "/usr/lib/fcgi-bin/publication.fcgi",
41     "check-local" => "disable",
42     "max-procs" => 1
43   ))
44 )
```

# ANNEXE B

## Code : application hello

```
1  /*
2   - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
3    Filename : hello.cpp
4    Project : Embedded Measurement Plateform
5    Part : CGI vs FCGI benchmark
6
7    Author : Johann Seydoux <johann dot seydoux at students dot hevs dot ch>
8    Version : 1.0
9    Created : 2009-05-11
10   Modified : 2009-06-29
11
12   Summary :
13   This program print an simple "Hello world"
14  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
15 */
16
17 #include "fcgi_stdio.h"
18 #include <stdlib.h>
19 #include <stdio.h>
20
21 int main(void)
22 {
23   while(FCGI_Accept() >= 0)
24   {
25     printf("Content-type: text/html\r\n"
26       "\r\n"
27                   "<title>Hello world</title>"
28                   "<h1>Hello world</h1>" );
29   }
30
31   return 0;
32 }
```

# ANNEXE C

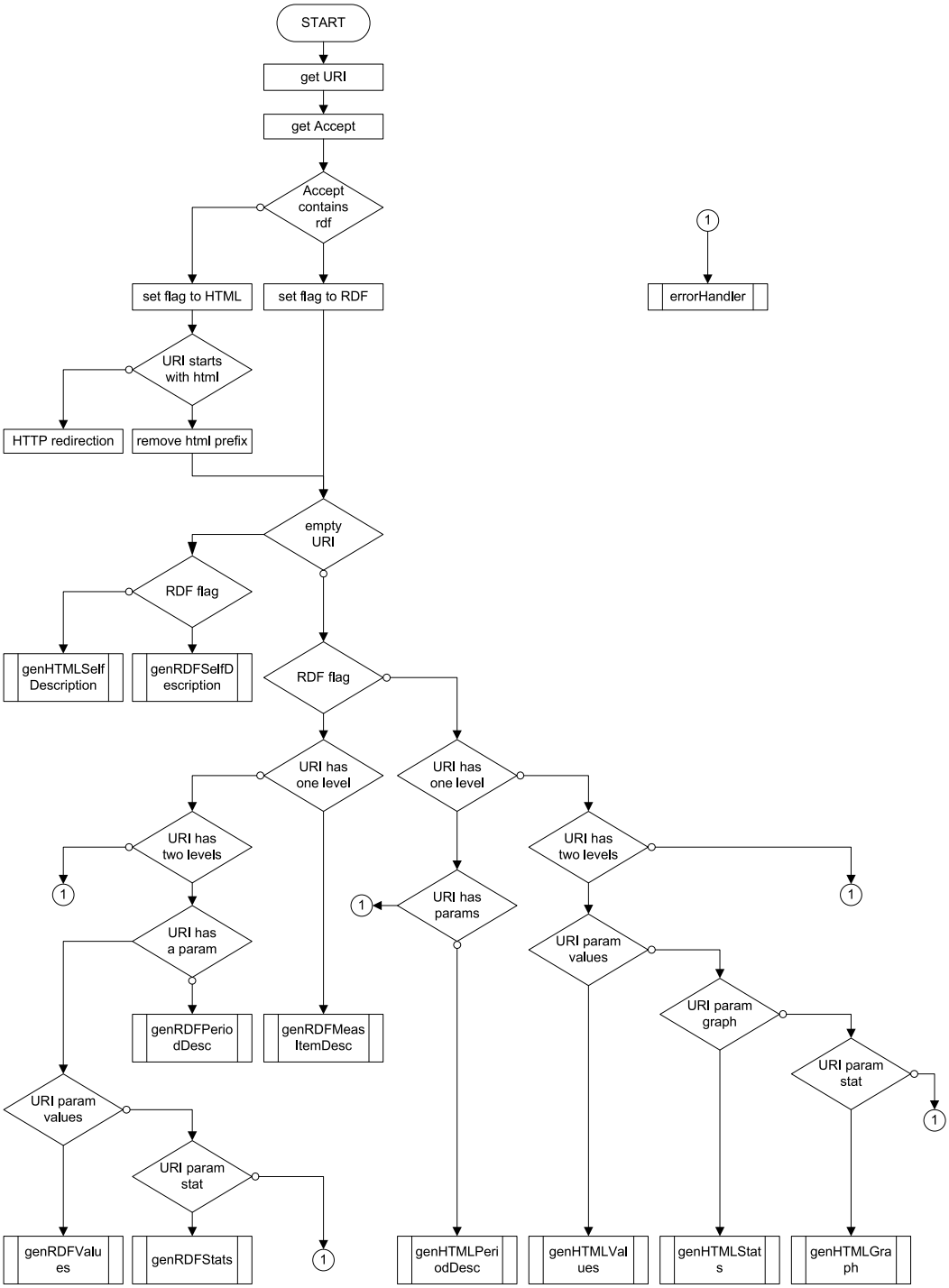
## Code : application dbread

```
1  /*
2   - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
3    Filename : dbread.cpp
4    Project : Embedded Measurement Plateform
5    Part : CGI vs FCGI benchmark
6
7    Author : Johann Seydoux <johann dot seydoux at students dot hevs dot ch>
8    Version : 1.0
9    Created : 2009-05-11
10   Modified : 2009-06-29
11
12   Summary :
13   This program read 10000 values from a database and print them.
14   - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
15  */
16
17 #include <QtSql>
18 #include <QSqlDatabase>
19 #include <QString>
20 #include <QProcess>
21 #include <stdlib.h>
22 #include <iostream>
23 #include "fcgi_stdio.h"
24
25 #define PATH_TO_DB "path_to_database/database.sqlite3"
26
27 int main()
28 {
29    while(FCGI_Accept() >= 0)
30    {
31       QSqlDatabase db = QSqlDatabase::addDatabase( "QSQLITE" , "cgiConnect" );
32
33       db.setDatabaseName( PATH_TO_DB );
34       db.open();
35
36       QString queryStr, num, errorStr, timeStr, retValue;
37       QSqlError error;
38       QSqlQuery result;
39       int i;
40
41       queryStr = "SELECT * FROM numbers";
42       result = db.exec( queryStr );
43
44       i = 1;
45
46       while ( result.next() )
47       {
48             retValue += result.value(0).toString();
49             i++;
50       }
51
52       num.setNum( i );
53
54       printf("Content-type: text/html\r\n"
55                   "\r\n"
56                   "<html>"
57                   "<title>BDACCESS Bench</title>"
58                   "</html><body>"
59                   "<span style=\"color: red\">%s</span><br/>"
60                   "%s results are:<br/>%s<br/>"
```

```
61                      "</body></html>",
62                      errorStr.toAscii().data(), num.toAscii().data(), retValue.toAscii().data());
63          db.close();
64      }
65
66      QSqlDatabase::removeDatabase( "cgiConnect" );
67
68      return 0;
69  }
```

# ANNEXE D

## Diagramme de flux : application de publication



# ANNEXE E

# Code : application de publication

## E.1 Header file

```
1  /*
2    PUBLICATION APPLICATION DEFINES
3  */
4
5  #define ACCEPT_RDF 0
6  #define ACCEPT_HTML 1
7  #define ACCEPT_OTHER 2
8
9  enum eErreur
10 {
11   ERR_URI,
12   ERR_EMPTYDEVICE,
13   ERR_BADMEASITEM,
14   ERR_NODATA
15 };
16
17 // PREPROCESSING DEFINES (FOR RDF/HTML GENERATION)
18 #define toStr(string) #string
19
20 #define PRINT printf(
21 #define E_PRINT );
22
23 // PATH TO THE CENTRAL DATABASE
24 #define PATH_TO_DB "path_to_database/database.sqlite3"
```

## E.2 main

```
1  /*
2   - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
3    Filename : main.cpp
4    Project : Embedded Measurement Plateform
5    Part : Data provider application
6
7    Author : Johann Seydoux <johann dot seydoux at students dot hevs dot ch>
8    Version : 1.0
9    Created : 2009-05-11
10   Modified : 2009-06-29
11
12   Summary :
13   This is the Data Provider application which provide a RDF or HTML document
14   containing informations depending on the client request.
15  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
16 */
17
18 // C/C++ Standard libraries
19 #include <stdio.h>
20 #include <stdlib.h>
21 #include <cmath>
22
23 // Qt libraries
24 #include <QtSql>
25
26 // FastCGI library
```

```
27 #include "fcgi_stdio.h"
28
29 // Application definitions
30 #include "pub_app_config.h"
31
32 // Globals
33 QSqlDatabase db;
34
35 // RDF generation prototypes
36 void genRDFSelfDescription();
37 void genRDFMeasItemDescription( const QByteArray * );
38 void genRDFPeriodDescription( const QList<QByteArray> * );
39 void genRDFValues( const QList<QByteArray> * );
40 void genRDFStats( const QList<QByteArray> * );
41
42 // HTML generation prototypes
43 void genHTMLSelfDescription();
44 void genHTMLPeriodDescription( const QList<QByteArray> * );
45 void genHTMLValues( const QList<QByteArray> * );
46 void genHTMLGraph( const QList<QByteArray> * );
47 void printHTML( QByteArray * , QByteArray * , QByteArray * );
48 void genHTMLStats( const QList<QByteArray> * );
49
50 void getSensorsInfos( QList<QByteArray> * , QList<QByteArray> * , QList<QByteArray> * , QList<
       QByteArray> * , QList<QByteArray> * );
51 void getDeviceConfig( QMap<QByteArray, QByteArray> * );
52 void getObservations( QMap<int,int> * , QMap<int,QByteArray> * );
53 void getObservations( QList<QByteArray> * , const QByteArray * );
54 void getOntologies( QList<QByteArray> * , QList<QByteArray> * );
55 void getAvailableYearsList( QList<int> * , const QByteArray * );
56 void getPeriodList( QList<int> * , const QByteArray * , const QByteArray * , const int );
57 void getMeasureUnit( QByteArray * , const QByteArray * );
58 int getCoveredPeriod( const QByteArray * );
59 void getValues( QMap<int,double> * , const QByteArray * , const int , const int );
60 void getStats( QList<double> * , const QByteArray * , const QByteArray * );
61 void getRange( QList<int> * , const QByteArray * );
62 int getCount( QByteArray * , const int , const int , const int , const int );
63
64 // TIMESTAMP CONVERSION
65 int getUnixTimestamp( const QByteArray * );
66 QByteArray getISOTimeStamp( const int , int );
67
68 void errorHandler( int , int );
69
70 int main()
71 {
72    // Setting database connection
73    db = QSqlDatabase::addDatabase( "QSQLITE" , "fromDataProviderApp" );
74    db.setDatabaseName( PATH_TO_DB );
75
76    if ( db.open() )
77    {
78       while( FCGI_Accept() >= 0 )
79       {
80          QByteArray reqURI( getenv( "SCRIPT_NAME" ) );
81          QByteArray HTTPAccept( getenv( "HTTP_ACCEPT" ) );
82
83          // remove '/' at start of URI
84          reqURI.remove(0,1);
85          reqURI.append( getenv( "PATH_INFO" ) );
86
87          // get the accept type
88          int acceptType;
89
90          if ( HTTPAccept.contains( "rdf" ) )
91          {
92             acceptType = ACCEPT_RDF;
93          }
94          else
95          {
96             acceptType = ACCEPT_HTML;
97          }
98
```

```
99          bool stopApp = 0;
100
101         if ( acceptType == ACCEPT_HTML )
102         {
103            // content negociation
104            if ( !reqURI.startsWith( "html" ) )
105            {
106               // HTML REDIRECTION
107               printf( "Status: 303 See Other\n" );
108               printf( "Location: html/%s\n\n" , reqURI.data() );
109               stopApp = 1;
110            }
111            else
112            {
113               // remove "html/" for the URI levels analysis
114               reqURI.remove(0,5);
115            }
116         }
117
118         // release not required memory
119         reqURI.squeeze();
120
121         if ( !stopApp )
122         {
123            // check if there's an requested URI
124            if ( !reqURI.isEmpty() )
125            {
126               // URI parsing -> levels
127               // <measurement_item>/<year>-<month>-<day>T<hour>-<min>
128
129               QList<QByteArray> parsedURI = reqURI.split( '/' );
130
131               if ( acceptType == ACCEPT_RDF )
132               {
133                  // 0 : <measurement_item>
134                  // 1 : <year>-<month>-<day>T<hour>-<min>-<sec>
135                  if ( parsedURI.size() == 1 )
136                  {
137                     if ( !parsedURI.at(0).contains( "." ) )
138                     {
139                        // the requested URI contains the measurement item only
140                        genRDFMeasItemDescription( &parsedURI.at(0) );
141                     }
142                     else
143                     {
144                         errorHandler( 404 , ERR_URI );
145                     }
146                  }
147                  else if ( parsedURI.size() == 2 )
148                  {
149                     // check URI parameters
150                     if ( !parsedURI.at(1).contains( "." ) )
151                     {
152                        // URI doesn't contain parameter -> description
153                        genRDFPeriodDescription( &parsedURI );
154                     }
155                     else if ( parsedURI.at(1).contains( ".values" ) )
156                     {
157                        genRDFValues( &parsedURI );
158                     }
159                     else if ( parsedURI.at(1).contains( ".stat" ) )
160                     {
161                        genRDFStats( &parsedURI );
162                     }
163                     else
164                     {
165                        errorHandler( 404 , ERR_URI );
166                     }
167                  }
168                  else if ( parsedURI.size() > 2 )
169                  {
170                     errorHandler( 404 , ERR_URI );
171                  }
```

```
172                }
173                else
174                {
175                   // 0 : <measurement_item>
176                   // 1 : <year>-<month>-<day>T<hour>-<min>-<sec>
177
178                   // maximum levels
179                   if ( parsedURI.size() < 3 )
180                   {
181                      if ( parsedURI.size() > 1 )
182                      {
183                         // the URI contains the measurement item only
184                         if ( !parsedURI.at(1).contains( "." ) )
185                         {
186                            genHTMLPeriodDescription( &parsedURI );
187                         }
188                         else if ( parsedURI.at(1).contains( ".values" ) )
189                         {
190                            genHTMLValues( &parsedURI );
191                         }
192                         else if ( parsedURI.at(1).contains( ".stat" ) )
193                         {
194                            genHTMLStats( &parsedURI );
195                         }
196                         else if ( parsedURI.at(1).contains( ".graph" ) )
197                         {
198                            genHTMLGraph( &parsedURI );
199                         }
200                         else
201                         {
202                            errorHandler( 404 , ERR_URI );
203                         }
204                      }
205                      else
206                      {
207                         if ( parsedURI.at(0).contains( "." ) )
208                         {
209                            errorHandler( 404 , ERR_URI );
210                         }
211                         else
212                         {
213                            genHTMLPeriodDescription( &parsedURI );
214                         }
215                      }
216                   }
217                   else
218                   {
219                      errorHandler( 404 , ERR_URI );
220                   }
221                }
222             }
223             else
224             {
225                // the requested URI is empty (device description)
226                ( acceptType == ACCEPT_RDF ) ? genRDFSelfDescription() : genHTMLSelfDescription();
227             }
228          }
229       }
230    }
231    else
232    {
233       // database connection failed
234       printf( "Content-type: text/html\n\n"
235             "<html>"
236             "<head>"
237             "<title>ERROR</title>"
238             "</head>"
239             "<body>Could not connect to database</body>"
240             "</html>" );
241    }
242
243    return 0;
244 }
```

```
245
246 /**
247  * Function : getDeviceConfig
248  * Parameter : pDeviceConfig pointer on where to return informations
249  * Summary : return all informations about the device
250  **/
251 void getDeviceConfig( QMap<QByteArray, QByteArray> * pDeviceConfig )
252 {
253    QSqlQuery query = db.exec( "SELECT property, value FROM Device_config" );
254
255    if ( query.isActive() )
256    {
257       while ( query.next() )
258       {
259          pDeviceConfig->insert( query.value(0).toByteArray() ,
260                                 query.value(1).toByteArray() );
261       }
262    }
263 }
264
265 /**
266  * Function : getOntologies
267  * Summary : return the prefixes and URIs from all ontologies
268  **/
269 void getOntologies( QList<QByteArray> * pPrefixes , // where to return ontologies prefixes
270                     QList<QByteArray> * pRefURIs     // where to return ontologies URI
271                   )
272 {
273    QSqlQuery query = db.exec( "SELECT prefix, uri FROM Ontology" );
274
275    if ( query.isActive() )
276    {
277       while ( query.next() )
278       {
279          pPrefixes->append( query.value(0).toByteArray() );
280          pRefURIs->append( query.value(1).toByteArray() );
281       }
282    }
283 }
284
285 /**
286  * Function : getSensorsInfos
287  * Summary : Get all sensors types with what they measure
288  **/
289 void getSensorsInfos( QList<QByteArray> * pSensorsIDs , // where to return IDs
290                       QList<QByteArray> * pMeasTypes , // where to return measTypes
291                       QList<QByteArray> * pSensorsPos , // where to return position
292                       QList<QByteArray> * pParamNames , // where to return parameters
293                       QList<QByteArray> * pSystemNames // where to return system names
294                     )
295 {
296    QSqlQuery query = db.exec(
297                            "SELECT Sensor.idSensor, Sensor.measType, Sensor.pos, Parameter.name,
                                System.name "
298                            "FROM Parameter, Sensor, System "
299                            "WHERE Parameter.idParam = Sensor.idParam "
300                            "AND Sensor.idSys = System.idSys"
301                          );
302
303    if ( query.isActive() )
304    {
305       while ( query.next() )
306       {
307          pSensorsIDs->append( query.value(0).toByteArray() );
308          pMeasTypes->append( query.value(1).toByteArray() );
309          pSensorsPos->append( query.value(2).toByteArray() );
310          pParamNames->append( query.value(3).toByteArray() );
311          pSystemNames->append( query.value(4).toByteArray() );
312       }
313    }
314 }
315
316 /**
```

```
317  * Function : getAvailableYearsList
318  * Summary : return all available years for the requested measurement item
319  **/
320 void getAvailableYearsList( QList<int> * pYearsList , // where to return available years
321                     const QByteArray * pMeasItem // measurement type
322                   )
323 {
324    QSqlQuery query = db.exec ( "SELECT timestamp "
325                        "FROM Stat_years "
326                        "WHERE idSensor = ( SELECT idSensor FROM Sensor WHERE measType = '" + *
                               pMeasItem + "' )"
327                      );
328
329    if ( query.isActive() )
330    {
331       while ( query.next() )
332       {
333          QByteArray ISODate = getISOTimeStamp( query.value(0).toInt() , 1 );
334          pYearsList->append( ISODate.toInt() );
335       }
336    }
337 }
338
339 /**
340  * Function : getMeasureUnit
341  * Summary : return the measurement unit for the requested measurement type
342  **/
343 void getMeasureUnit( QByteArray * pUnit ,       // where to return unit
344                const QByteArray * pMeasItem    // measurement item
345              )
346 {
347    QSqlQuery query = db.exec(  "SELECT Unit.name "
348                           "FROM Unit, Parameter, Sensor "
349                           "WHERE Sensor.measType = '" + *pMeasItem + "' "
350                           "AND Parameter.idParam = Sensor.idParam "
351                           "AND Parameter.idUnit = Unit.idUnit"
352                        );
353
354    if ( query.isActive() )
355    {
356       while ( query.next() )
357       {
358          *pUnit = query.value(0).toByteArray();
359       }
360    }
361 }
362
363 /**
364  * Function : errorHandler
365  * Summary : return the HTTP Status corresponding to the error code and an HTML document
366  *   that describes the error occurred
367  **/
368 void errorHandler( int errorCode ,              // HTTP status code
369                int errorType                    // error type defined in
370                                                 // publication_application.h
371              )
372 {
373    QByteArray status, errorString;
374
375    // here must be defined the status code
376    // returned by the error codes
377    if ( errorCode == 404 )
378    {
379       status.append( "Status: 404 Not Found" );
380    }
381
382    if ( errorType == ERR_URI )
383    {
384       errorString.append( "The URI is not well formed" );
385    }
386    else if ( errorType == ERR_EMPTYDEVICE )
387    {
388       errorString.append( "The emmbedded station does not have any sensors" );
```

```
389     }
390     else if ( errorType == ERR_BADMEASITEM )
391     {
392        errorString.append( "The required measurement does not exist" );
393     }
394     else if ( errorType == ERR_NODATA )
395     {
396        errorString.append( "No data found" );
397     }
398
399     printf( "Content-type: text/html\n%s\n\n"
400           "<html>"
401           "<head>"
402           "<title>%s</title>"
403           "<meta name=\"robots\" content=\"NONE,NOARCHIVE\">"
404           "</head>"
405           "<style type=\"text/css\">"
406           "html, body { font: small sans-serif; margin: 0; padding: 0; }"
407           "div#error { width: 100%%; background-color: #fee8aa; border-bottom: 1px solid #e5e5e5;
                  padding-left: 10px; padding-top: 5px; }"
408           "h1 { font-weight: normal; color: #515151 }"
409           "h1 span { font-size:60%%; color:#666; font-weight:normal; }"
410           "table { border:none; border-collapse: collapse; width:100%%; }"
411           "td, th { vertical-align:top; padding:2px 3px; }"
412           "th { width:12em; text-align:right; color:#666; padding-right:.5em; }"
413           "</style>"
414           "<body>"
415           "<div id=\"error\"><h1>%s<span> (%s)</span></h1></div>"
416           "</body>"
417           "</html>" , status.data() , status.data() , errorString.data() , status.data() );
418 }
419
420 /**
421  * Function : getObservations
422  * Summary : return the observations made by each sensor
423  **/
424 void getObservations( QMap<int,int> * pSensors_obs , // where to return relations between sensors
       and observations
425                       QMap<int,QByteArray> * pObsList    // where to return observations and their IDs
426                     )
427 {
428     QSqlQuery query = db.exec ( "SELECT idSensor, idObs "
429                                 "FROM Sensor_obs ORDER BY idSensor"
430                               );
431
432     if ( query.isActive() )
433     {
434        while ( query.next() )
435        {
436           pSensors_obs->insertMulti( query.value(0).toInt() , query.value(1).toInt() );
437        }
438     }
439
440     query = db.exec ( "SELECT idObs, name FROM Observation" );
441
442     if ( query.isActive() )
443     {
444        while ( query.next() )
445        {
446           pObsList->insertMulti( query.value(0).toInt() , query.value(1).toByteArray() );
447        }
448     }
449 }
450
451 /**
452  * Function : getObservations
453  * Summary : return the observations made by the specified sensor
454  **/
455 void getObservations( QList<QByteArray> * pObslist , const QByteArray * pMeasureItem )
456 {
457     QSqlQuery query = db.exec ( "SELECT Observation.name "
458                                 "FROM Observation, Sensor_obs "
459                                 "WHERE Sensor_obs.idSensor = ( SELECT idSensor FROM Sensor WHERE measType
```

```
                                = '" + *pMeasureItem + "' ) "
460                           "AND Sensor_obs.idObs = Observation.idObs"
461                         );
462
463     if ( query.isActive() )
464     {
465        while ( query.next() )
466        {
467           pObslist->append( query.value(0).toByteArray() );
468        }
469     }
470  }
471
472  /**
473   * Function : genRDFSelfDescription
474   * Summary : return the RDF document that contains device description
475   **/
476  void genRDFSelfDescription()
477  {
478     int nbSensors;
479
480     // get the device informations
481     QMap<QByteArray, QByteArray> deviceConfig;
482     getDeviceConfig( &deviceConfig );
483
484     // get the sensors informations
485     QList<QByteArray> sensorsIDs, measTypes, sensorsPos, paramNames, systemNames;
486     getSensorsInfos( &sensorsIDs , &measTypes , &sensorsPos , &paramNames , &systemNames );
487     nbSensors = sensorsIDs.size();
488
489     // get the ontologies
490     QList<QByteArray> prefixes, refURIs;
491     getOntologies( &prefixes , &refURIs );
492
493     PRINT "Content-type: application/rdf+xml\n\n" E_PRINT
494     PRINT toStr( <?xml version="1.0" encoding="UTF-8"?> ) E_PRINT
495     PRINT
496     "<!DOCTYPE rdf:RDF [<!ENTITY dp \"%s\"><!ENTITY dploc \"%sLocation\"><!ENTITY host \"%s\">"
497     , deviceConfig.value( "name" ).data() , deviceConfig.value( "name" ).data() , deviceConfig.
            value( "baseUrl" ).data()
498     E_PRINT
499
500     // entities list
501     for ( int i = 0 ; i < nbSensors; ++i ) {
502        PRINT "<!ENTITY m%d \"%s\">" , i , measTypes.at(i).data() E_PRINT
503     }
504
505     PRINT "]>" E_PRINT
506
507     // NAMESPACES DEFINITION
508     PRINT "<rdf:RDF " E_PRINT
509
510     for ( int i = 0 ; i < refURIs.size() ; ++i )
511     {
512        // NAMESPACES DECLARATION
513        if ( prefixes.at(i).contains( "emep" ) )
514        {
515              PRINT "xmlns=\"%s\" " , refURIs.at(i).data() E_PRINT
516        }
517        else
518        {
519              PRINT "xmlns:%s=\"%s\" " , prefixes.at(i).data() , refURIs.at(i).data() E_PRINT
520        }
521     }
522
523     PRINT ">" E_PRINT
524
525     // Sensors list
526     PRINT "<rdf:Description rdf:about=\"#&dp;\"><rdfs:label>&dp;</rdfs:label>" E_PRINT
527     PRINT "<rdfs:isDefinedBy rdf:resource=\"&host;html/\"/>" E_PRINT
528
529     for ( int i = 0 ; i < nbSensors; ++i )
530     {
```

```
531         if ( !systemNames.at(i).contains( "None" ) )
532         {
533            PRINT "<sten:hasPart rdf:resource=\"#%s\"/>" , systemNames.at(i).data() E_PRINT
534         }
535
536         PRINT "<sten:hasPart rdf:resource=\"#&m%d;Sensor\"/>" , i E_PRINT
537      }
538
539      PRINT
540      toStr (
541         <sten:occupies rdf:resource="#&dploc;"/>
542         </rdf:Description>
543      )
544      E_PRINT
545
546      // Device location definition
547      PRINT "<rdf:Description rdf:about=\"#&dploc;\">" E_PRINT
548      PRINT "<geo:altitude rdf:datatype=\"http://www.w3.org/2001/XMLSchema#double\">%s</geo:altitude
             >" , deviceConfig.value( "altitude" ).data() E_PRINT
549      PRINT "<geo:latitude rdf:datatype=\"http://www.w3.org/2001/XMLSchema#double\">%s</geo:latitude
             >" , deviceConfig.value( "latitude" ).data() E_PRINT
550      PRINT "<geo:longitude rdf:datatype=\"http://www.w3.org/2001/XMLSchema#double\">%s</geo:
             longitude>" , deviceConfig.value( "longitude" ).data() E_PRINT
551      PRINT "</rdf:Description>" E_PRINT
552
553      // Measurements definition
554      for ( int i = 0 ; i < nbSensors ; ++i )
555      {
556         if ( !systemNames.at(i).contains( "None" ) )
557         {
558            PRINT "<%sSystem rdf:about=\"#%s\">" , systemNames.at(i).data() , systemNames.at(i).data
                  () E_PRINT
559            PRINT "<consume rdf:resource=\"#&m%d;\"/>" , i E_PRINT
560            PRINT "</%sSystem>" , systemNames.at(i).data() E_PRINT
561         }
562
563         PRINT "<%s rdf:about=\"#&m%d;\">" , paramNames.at(i).data() , i E_PRINT
564         PRINT "<sten:at rdf:resource=\"#dploc;\"/>" E_PRINT
565
566         if ( !sensorsPos.at(i).isEmpty() )
567         {
568            PRINT "<%s rdf:resource=\"#&dploc;\"/>" , sensorsPos.at(i).data() E_PRINT
569         }
570
571         PRINT "</%s>" , paramNames.at(i).data() E_PRINT
572      }
573
574      /*
575        GET THE SENSOR_OBS TABLE AND OBSERVATION TABLE IN TWO MAPS
576        Reason : SELECT query in loop = bad performance w/ SQLite
577      */
578      QMap<int,int> sensor_obs;
579      QMap<int,QByteArray> obsList;
580      getObservations( &sensor_obs , &obsList );
581
582      for ( int i = 0 ; i < nbSensors ; ++i )
583      {
584         PRINT "<sten:ObservingDevice rdf:about=\"#&m%d;Sensor\">" , i E_PRINT
585         PRINT "<sten:performsObservation rdf:resource=\"&host;&m%d;#&m%d;MeasureActivity\"/>" , i ,
                i E_PRINT
586
587         QList<int> obsForSensor = sensor_obs.values( sensorsIDs.at(i).toInt() );
588
589         for ( int j = 0 ; j < obsForSensor.size() ; ++j )
590         {
591            QByteArray obs( obsList.value( obsForSensor.value(j) ) );
592            PRINT "<sten:performs rdf:resource=\"&host;&m%d;#&m%d;%sActivity\"/>" , i , i , obs.data
                  () E_PRINT
593         }
594
595         PRINT "<sten:observes rdf:resource=\"#&m%d;\"/>" , i E_PRINT
596         PRINT "</sten:ObservingDevice>" E_PRINT
597      }
```

```
598
599     PRINT "</rdf:RDF>" E_PRINT
600 }
601
602 /**
603  * Function : genRDFMeasItemDescription
604  * Parameter : pMeasureItem pointer on the measurement item name
605  * Summary : return the RDF document that contains the sensor description
606  **/
607 void genRDFMeasItemDescription( const QByteArray * pMeasureItem )
608 {
609     QList<int> yearsList;
610     getAvailableYearsList( &yearsList , pMeasureItem );
611
612     if ( yearsList.size() > 0 )
613     {
614         QMap<QByteArray, QByteArray> deviceConfig;
615         getDeviceConfig( &deviceConfig );
616
617         // get the ontologies
618         QList<QByteArray> prefixes, refURIs;
619         getOntologies( &prefixes , &refURIs );
620
621         PRINT "Content-type: application/rdf+xml\n\n" E_PRINT
622         PRINT toStr( <?xml version="1.0" encoding="UTF-8"?> ) E_PRINT
623
624         // DOCTYPE WITH ENTITIES
625         PRINT "<!DOCTYPE rdf:RDF [<!ENTITY host \"%s\"><!ENTITY m \"%s\"><!ENTITY emep \"http://
                sten-04.hevs.ch/emep.rdf\"><!ENTITY time \"http://registry.emep.hevs.ch/time/\">]>" ,
                deviceConfig.value( "baseUrl" ).data() , pMeasureItem->data() E_PRINT
626
627         // NAMESPACES DEFINITION
628         PRINT "<rdf:RDF " E_PRINT
629
630         for ( int i = 0 ; i < refURIs.size() ; ++i )
631         {
632             // NAMESPACES DECLARATION
633             if ( prefixes.at(i).contains( "emep" ) )
634             {
635                   PRINT "xmlns=\"%s\" " , refURIs.at(i).data() E_PRINT
636             }
637             else
638             {
639                   PRINT "xmlns:%s=\"%s\" " , prefixes.at(i).data() , refURIs.at(i).data() E_PRINT
640             }
641         }
642
643         PRINT ">" E_PRINT
644
645         PRINT toStr ( <rdf:Description rdf:about="&m;MeasureActivity"> ) E_PRINT
646         PRINT toStr ( <rdf:type rdf:resource="http://www.s-ten.eu/sten-core#ObservingActivity"/> )
                E_PRINT
647         PRINT toStr ( <rdf:type rdf:resource="http://www.s-ten.eu/sten-core#TransmittingActivity"/>
                ) E_PRINT
648         PRINT "<sten:giveResultRecord>" E_PRINT
649         PRINT "<sten:ResultRecord>" E_PRINT
650
651         for ( int i = 0 ; i < yearsList.size() ; ++i )
652         {
653             PRINT "<sten:hasTemporalPart rdf:resource=\"&host;&m;/%d#this\"/>" , yearsList.at(i)
                    E_PRINT
654             PRINT "<sten:hasPart rdf:resource=\"&host;&m;/%d.values\"/>" , yearsList.at(i) E_PRINT
655         }
656
657         PRINT "</sten:ResultRecord>" E_PRINT
658         PRINT "</sten:giveResultRecord>" E_PRINT
659         PRINT toStr ( <sten:observationPerformedByDevice rdf:resource="#&m;Sensor"/> ) E_PRINT
660         PRINT toStr ( <sten:observes rdf:resource="#&m;"/> ) E_PRINT
661         PRINT "<rdfs:isDefinedBy rdf:resource=\"&host;html/&m;\"/>" E_PRINT
662         PRINT "</rdf:Description>" E_PRINT
663
664         QList<QByteArray> observationsList;
665         getObservations( &observationsList , pMeasureItem );
```

```
666
667         for ( int i = 0 ; i < observationsList.size() ; ++i )
668         {
669            PRINT "<rdf:Description rdf:about=\"#&m;%sActivity\">" , observationsList.at(i).data()
                   E_PRINT
670            PRINT "<rdf:type rdf:resource=\"&emep;%s#Activity\"/>" , observationsList.at(i).data()
                   E_PRINT
671            PRINT "<sten:giveResultRecord>" E_PRINT
672            PRINT "<sten:ResultRecord>" E_PRINT
673
674            for ( int i = 0 ; i < yearsList.size() ; ++i )
675            {
676                  PRINT "<sten:hasTemporalPart rdf:resource=\"&host;&m;/%d#stat\"/>" , yearsList.at(
                       i) E_PRINT
677                  PRINT "<sten:hasTemporalPart rdf:resource=\"&host;&m;/%d.stat\"/>" , yearsList.at(
                       i) E_PRINT
678            }
679
680            PRINT "</sten:ResultRecord>" E_PRINT
681            PRINT "</sten:giveResultRecord>" E_PRINT
682            PRINT toStr ( <sten:observationPerformedByDevice rdf:resource="#&m;Sensor"/> ) E_PRINT
683            PRINT toStr ( <sten:observes rdf:resource="#&m;"/> ) E_PRINT
684            PRINT "</rdf:Description>" E_PRINT
685         }
686
687         for ( int i = 0 ; i < yearsList.size() ; ++i )
688         {
689            PRINT "<rdf:Description rdf:about=\"&host;&m;/%d#this\">" , yearsList.at(i) E_PRINT
690            PRINT "<rdf:type rdf:resource=\"http://www.s-ten.eu/sten-core#ResultRecord\"/>" E_PRINT
691            PRINT "<rdfs:label>&m; For Year %d</rdfs:label>" , yearsList.at(i) E_PRINT
692            PRINT "<sten:during rdf:resource=\"&time;%d#Year-%d\"/>" , yearsList.at(i) , yearsList.
                   at(i) E_PRINT
693            PRINT "</rdf:Description>" E_PRINT
694            PRINT "<rdf:Description rdf:about=\"&host;&m;/%d#stat\">" , yearsList.at(i) E_PRINT
695            PRINT "<rdf:type rdf:resource=\"http://www.s-ten.eu/sten-core#ResultRecord\"/>" E_PRINT
696            PRINT "<rdfs:label>&m; For Year %d</rdfs:label>" , yearsList.at(i) E_PRINT
697            PRINT "<sten:during rdf:resource=\"&time;%d#Year-%d\"/>" , yearsList.at(i) , yearsList.
                   at(i) E_PRINT
698            PRINT "</rdf:Description>" E_PRINT
699         }
700
701         PRINT "</rdf:RDF>" E_PRINT
702      }
703      else
704      {
705         errorHandler( 404 , ERR_NODATA );
706      }
707   }
708
709   /**
710    * Function : genRDFPeriodDescription
711    * Parameter : pParsedURI pointer on the parsed URI
712    *                      pParsedURI->at(0) : URI level 1
713    *                      pParsedURI->at(1) : URI level 2
714    * Summary : return the RDF document that contains the period description
715    **/
716   void genRDFPeriodDescription( const QList<QByteArray> * pParsedURI )
717   {
718      // parsedDateTime at 0 -> date
719      // parsedDateTime at 1 -> time
720      QList<QByteArray> parsedDateTime = pParsedURI->at(1).split( 'T' );
721
722      QList<QByteArray> parsedDate = parsedDateTime.at(0).split( '-' );
723      QList<int> periodList;
724      QByteArray reqPeriod, subPeriod;
725
726      int toUTS = getCoveredPeriod( &pParsedURI->at(1) );
727      int level = 0;
728      getPeriodList( &periodList , &pParsedURI->at(0) , &pParsedURI->at(1) , toUTS );
729
730      // check if there's data for this URI
731      if ( periodList.size() > 0 )
732      {
```

```
733         if ( parsedDateTime.size() == 1 )
734         {
735            // the request contains a date only
736            if ( parsedDate.size() == 1 )
737            {
738               reqPeriod = "Year";
739               subPeriod = "Month";
740               level = 2;
741            }
742            else if ( parsedDate.size() == 2 )
743            {
744               reqPeriod = "Month";
745               subPeriod = "Day";
746               level = 3;
747            }
748            else if ( parsedDate.size() == 3 )
749            {
750               reqPeriod = "Day";
751               subPeriod = "Hour";
752               level = 4;
753            }
754         }
755         else
756         {
757            // the request contains a date and time
758            QList<QByteArray> parsedTime = parsedDateTime.at(1).split( '-' );
759
760            if ( parsedTime.size() == 1 )
761            {
762               reqPeriod = "Hour";
763               subPeriod = "Minute";
764               level = 5;
765            }
766         }
767
768         QMap<QByteArray, QByteArray> deviceConfig;
769         getDeviceConfig( &deviceConfig );
770
771         // get the ontologies
772         QList<QByteArray> prefixes, refURIs;
773         getOntologies( &prefixes , &refURIs );
774
775         PRINT "Content-type: application/rdf+xml\n\n" E_PRINT
776
777         // XML DEFINITION
778         PRINT toStr( <?xml version="1.0" encoding="UTF-8"?> ) E_PRINT
779
780         // DOCTYPE WITH ENTITIES
781         PRINT
782         "<!DOCTYPE rdf:RDF [<!ENTITY host \"%s\"><!ENTITY rUrl \"%s%s/%s\"><!ENTITY m \"%s\"><!
                ENTITY time \"http://registry.emep.hevs.ch/time/\">]>"
783         , deviceConfig.value( "baseUrl" ).data() , deviceConfig.value( "baseUrl" ).data() ,
                pParsedURI->at(0).data() , pParsedURI->at(1).data() , pParsedURI->at(0).data()
784         E_PRINT
785
786         // NAMESPACES DEFINITION
787         PRINT "<rdf:RDF " E_PRINT
788
789         for ( int i = 0 ; i < refURIs.size() ; ++i )
790         {
791            // NAMESPACES DECLARATION
792            if ( prefixes.at(i).contains( "emep" ) )
793            {
794               PRINT "xmlns=\"%s\" " , refURIs.at(i).data() E_PRINT
795            }
796            else
797            {
798               PRINT "xmlns:%s=\"%s\" " , prefixes.at(i).data() , refURIs.at(i).data() E_PRINT
799            }
800         }
801
802         PRINT ">" E_PRINT
803
```

```
804         PRINT "<sten:ResultRecord rdf:about=\"&rUrl;#this\">" E_PRINT
805         PRINT "<rdfs:label>&m; For %s %s</rdfs:label>" , reqPeriod.data() , pParsedURI->at(1).data
                () E_PRINT
806         PRINT "<sten:during rdf:resource=\"&time;%s#%s-%s\"/>" , pParsedURI->at(1).data() ,
                reqPeriod.data() , pParsedURI->at(1).data() E_PRINT
807
808         for ( int i = 0 ; i < periodList.size() ; ++i )
809         {
810            if ( level != 5 )
811               PRINT "<sten:hasTemporalPart rdf:resource=\"&host;&m;/%s#this\"/>" , getISOTimeStamp(
                      periodList.at(i) , level ).data() E_PRINT
812
813            PRINT "<sten:hasPart rdf:resource=\"&host;&m;/%s.values\"/>" , getISOTimeStamp(
                   periodList.at(i) , level ).data() E_PRINT
814         }
815
816         PRINT "<rdfs:isDefinedBy rdf:resource=\"&host;html/%s\"/>" , pParsedURI->at(1).data()
                E_PRINT
817         PRINT "</sten:ResultRecord>" E_PRINT
818
819         // check if the period is hours
820         // it that case, the sub periods are minutes and
821         // that means they are only value and have no
822         // statistics values and description
823         if ( level != 5 )
824         {
825            // STATS
826            PRINT "<sten:ResultRecord rdf:about=\"&rUrl;#stat\">" E_PRINT
827            PRINT "<rdfs:label>&m; For %s %s</rdfs:label>" , reqPeriod.data() , pParsedURI->at(1).
                   data() E_PRINT
828            PRINT "<sten:during rdf:resource=\"&time;%s#%s-%s\"/>" , pParsedURI->at(1).data() ,
                   reqPeriod.data() , pParsedURI->at(1).data() E_PRINT
829            PRINT "<sten:hasPart rdf:resource=\"&rUrl;.stat\"/>" E_PRINT
830
831            for ( int i = 0 ; i < periodList.size() ; ++i )
832            {
833               PRINT "<sten:hasTemporalPart rdf:resource=\"&host;&m;/%s#stat\"/>" , getISOTimeStamp(
                      periodList.at(i) , level ).data() E_PRINT
834               PRINT "<sten:hasPart rdf:resource=\"&host;&m;/%s.stat\"/>" , getISOTimeStamp(
                      periodList.at(i) , level ).data() E_PRINT
835            }
836
837            PRINT "<rdfs:isDefinedBy rdf:resource=\"&host;html/%s\"/>" , pParsedURI->at(1).data()
                   E_PRINT
838            PRINT "</sten:ResultRecord>" E_PRINT
839
840            for ( int i = 0 ; i < periodList.size() ; ++i )
841            {
842               PRINT "<rdf:Description rdf:about=\"&host;&m;/%s#this\">" , getISOTimeStamp(
                      periodList.at(i) , level ).data() E_PRINT
843               PRINT "<rdfs:label>&m; For %s %s</rdfs:label>" , subPeriod.data() , getISOTimeStamp(
                      periodList.at(i) , level ).data() E_PRINT
844               PRINT "<sten:during rdf:resource=\"&time;%s#%s-%s\"/>" , getISOTimeStamp( periodList.
                      at(i) , level ).data() , subPeriod.data() , getISOTimeStamp( periodList.at(i) ,
                      level ).data() E_PRINT
845               PRINT " </rdf:Description>" E_PRINT
846
847               // STATS
848               PRINT "<rdf:Description rdf:about=\"&host;&m;/%s#stat\">" , getISOTimeStamp(
                      periodList.at(i) , level ).data() E_PRINT
849               PRINT "<rdfs:label>&m; For %s %s</rdfs:label>" , subPeriod.data() , getISOTimeStamp(
                      periodList.at(i) , level ).data() E_PRINT
850               PRINT "<sten:during rdf:resource=\"&time;%s#%s-%s\"/>" , getISOTimeStamp( periodList.
                      at(i) , level ).data() , subPeriod.data() , getISOTimeStamp( periodList.at(i) ,
                      level ).data() E_PRINT
851               PRINT " </rdf:Description>" E_PRINT
852            }
853         }
854
855         PRINT "</rdf:RDF>" E_PRINT
856      }
857      else
858      {
```

```
859         errorHandler( 404 , ERR_NODATA );
860      }
861   }
862
863   /**
864    * Function : genRDFValues
865    * Parameter : pParsedURI pointer on the parsed URI
866    *                       pParsedURI->at(0) : URI level 1
867    *                       pParsedURI->at(1) : URI level 2
868    * Summary : return the RDF document that contains all values
869                in a period
870    **/
871   void genRDFValues( const QList<QByteArray> * pParsedURI )
872   {
873      // pDateParam contains something like 2009-06-10.values
874      QList<QByteArray> dateParam = pParsedURI->at(1).split( '.' );
875      QList<QByteArray> periodValues, measTime;
876      QByteArray unit;
877      QMap<int, double> values;
878      int from, to;
879
880      // get end period interval
881      to = getCoveredPeriod( &dateParam.at(0) );
882      // convert req. period into UNIX timestamp format
883      from = getUnixTimestamp( &dateParam.at(0) );
884
885      getValues( &values , &pParsedURI->at(0) , from , to );
886
887      if ( values.size() > 0 )
888      {
889         getMeasureUnit( &unit , &pParsedURI->at(0) );
890
891         // get the ontologies
892         QList<QByteArray> prefixes, refURIs;
893         getOntologies( &prefixes , &refURIs );
894
895         QMap<QByteArray, QByteArray> deviceConfig;
896         getDeviceConfig( &deviceConfig );
897
898         PRINT "Content-type: application/rdf+xml\n\n" E_PRINT
899
900         // XML DEFINITION
901         PRINT toStr( <?xml version="1.0" encoding="UTF-8"?> ) E_PRINT
902
903         // DOCTYPE WITH ENTITIES
904         PRINT "<!DOCTYPE rdf:RDF [<!ENTITY host \"%s\"><!ENTITY m \"%s\"><!ENTITY mrec \"&m;
                RecordFor-\"><!ENTITY time \"http://registry.emep.hevs.ch/time/\">]>" , deviceConfig.
                value( "baseUrl" ).data() , pParsedURI->at(0).data() E_PRINT
905
906         // NAMESPACES DEFINITION
907         PRINT "<rdf:RDF " E_PRINT
908
909         for ( int i = 0 ; i < refURIs.size() ; ++i )
910         {
911            // NAMESPACES DECLARATION
912            if ( prefixes.at(i).contains( "emep" ) )
913            {
914               PRINT "xmlns=\"%s\" " , refURIs.at(i).data() E_PRINT
915            }
916            else
917            {
918               PRINT "xmlns:%s=\"%s\" " , prefixes.at(i).data() , refURIs.at(i).data() E_PRINT
919            }
920         }
921
922         PRINT ">" E_PRINT
923
924         QMapIterator<int, double> i(values);
925
926         while (i.hasNext()) {
927            i.next();
928            QByteArray ISODate = getISOTimeStamp( i.key() , 5 );
929            PRINT "<sten:ResultGraph rdf:about=\"#&mrec;%s\">" , ISODate.data() E_PRINT
```

```
930          PRINT "<sten:%s rdf:parseType=\"Resource\">" , unit.data() E_PRINT
931          PRINT "<sten:decimal>%.2f</sten:decimal>" , i.value() E_PRINT
932          PRINT "</sten:%s>" , unit.data() E_PRINT
933          PRINT "<sten:atTime rdf:resource=\"&time;%s#%s\"/>", ISODate.data() , ISODate.data()
                 E_PRINT
934          PRINT "<sten:resultGraphGivenBy rdf:resource=\"&host;&m;#&m;MeasureActivity\"/>" E_PRINT
935          PRINT "</sten:ResultGraph>" E_PRINT
936       }
937
938       PRINT "<rdf:Description rdf:about=\"\"><rdfs:isDefinedBy rdf:resource=\"&host;html/&m;/%s
              \"/></rdf:Description>" , pParsedURI->at(1).data() E_PRINT
939
940       PRINT "</rdf:RDF>" E_PRINT
941    }
942    else
943    {
944       errorHandler( 404 , ERR_NODATA );
945    }
946 }
947
948 /**
949  * Function : genRDFStats
950  * Parameter : pParsedURI pointer on the parsed URI
951  *                      pParsedURI->at(0) : URI level 1
952                         pParsedURI->at(1) : URI level 2
953  * Summary : return the RDF document that contains statistics for a period
954  **/
955 void genRDFStats( const QList<QByteArray> * pParsedURI )
956 {
957    QList<QByteArray> dateParam = pParsedURI->at(1).split( '.' );
958
959    QList<double> stats;
960
961    getStats( &stats , &pParsedURI->at(0) , &dateParam.at(0) );
962
963    if ( !stats.isEmpty() )
964    {
965       QByteArray unit, period;
966
967       if ( dateParam.at(0).length() == 4 )
968       {
969          period = "Year";
970       }
971       else if ( dateParam.at(0).length() == 7 )
972       {
973          period = "Month";
974       }
975       else if ( dateParam.at(0).length() == 10 )
976       {
977          period = "Day";
978       }
979       else if ( dateParam.at(0).length() == 13 )
980       {
981          period = "Hour";
982       }
983       else
984       {
985          period = "Minute";
986       }
987
988       getMeasureUnit( &unit , &pParsedURI->at(0) );
989
990       // get the ontologies
991       QList<QByteArray> prefixes, refURIs;
992       getOntologies( &prefixes , &refURIs );
993
994       QMap<QByteArray, QByteArray> deviceConfig;
995       getDeviceConfig( &deviceConfig );
996
997       PRINT "Content-type: application/rdf+xml\n\n" E_PRINT
998
999       // XML DEFINITION
1000      PRINT toStr( <?xml version="1.0" encoding="UTF-8"?> ) E_PRINT
```

```
1001
1002        // DOCTYPE WITH ENTITIES
1003        PRINT "<!DOCTYPE rdf:RDF [<!ENTITY host \"%s\"><!ENTITY m \"%s\"><!ENTITY mrec \"&m;
                RecordFor-\"><!ENTITY time \"http://registry.emep.hevs.ch/time/\">]>" , deviceConfig.
                value( "baseUrl" ).data() , pParsedURI->at(0).data() E_PRINT
1004
1005        // NAMESPACES DEFINITION
1006        PRINT "<rdf:RDF " E_PRINT
1007
1008        for ( int i = 0 ; i < refURIs.size() ; ++i )
1009        {
1010           // NAMESPACES DECLARATION
1011           if ( prefixes.at(i).contains( "emep" ) )
1012           {
1013              PRINT "xmlns=\"%s\" " , refURIs.at(i).data() E_PRINT
1014           }
1015           else
1016           {
1017              PRINT "xmlns:%s=\"%s\" " , prefixes.at(i).data() , refURIs.at(i).data() E_PRINT
1018           }
1019        }
1020
1021        PRINT ">" E_PRINT
1022
1023        QList<QByteArray> observationsList;
1024        getObservations( &observationsList , &pParsedURI->at(0) );
1025
1026        if ( observationsList.contains( "Maximizing" ) )
1027        {
1028           PRINT "<sten:ResultGraph rdf:about=\"&host;&m;/%s#max\"><sten:during rdf:resource=\"&
                   time;%s#%s-%s\"/><sten:%s rdf:parseType=\"Resource\"><sten:decimal>%f</sten:decimal
                   ></sten:%s><sten:resultGraphGivenBy rdf:resource=\"&host;&m;#&m;MaximizingActivity
                   \"/></sten:ResultGraph>"
                   , dateParam.at(0).data() , dateParam.at(0).data() , period.data() , dateParam.at(0).
                       data() , unit.data() , stats.at(1) , unit.data() E_PRINT
1029
1030        }
1031
1032        if ( observationsList.contains( "Minimizing" ) )
1033        {
1034           PRINT "<sten:ResultGraph rdf:about=\"&host;&m;/%s#min\"><sten:during rdf:resource=\"&
                   time;%s#%s-%s\"/><sten:%s rdf:parseType=\"Resource\"><sten:decimal>%f</sten:decimal
                   ></sten:%s><sten:resultGraphGivenBy rdf:resource=\"&host;&m;#&m;MinimizingActivity
                   \"/></sten:ResultGraph>"
                   , dateParam.at(0).data() , dateParam.at(0).data() , period.data() , dateParam.at(0).
                       data() , unit.data() , stats.at(0) , unit.data() E_PRINT
1035
1036        }
1037
1038        if ( observationsList.contains( "Averaging" ) )
1039        {
1040           PRINT "<sten:ResultGraph rdf:about=\"&host;&m;/%s#avg\"><sten:during rdf:resource=\"&
                   time;%s#%s-%s\"/><sten:%s rdf:parseType=\"Resource\"><sten:decimal>%f</sten:decimal
                   ></sten:%s><sten:resultGraphGivenBy rdf:resource=\"&host;&m;#&m;AveragingActivity
                   \"/></sten:ResultGraph>"
                   , dateParam.at(0).data() , dateParam.at(0).data() , period.data() , dateParam.at(0).
                       data() , unit.data() , stats.at(2) , unit.data() E_PRINT
1041
1042        }
1043
1044        PRINT "<rdf:Description rdf:about=\"\"><rdfs:isDefinedBy rdf:resource=\"&host;html/&m;/%s
                \"/></rdf:Description>" , pParsedURI->at(1).data() E_PRINT
1045
1046        PRINT "</rdf:RDF>" E_PRINT
1047     }
1048     else
1049     {
1050        errorHandler( 404 , ERR_NODATA );
1051     }
1052  }
1053
1054  /**
1055   * Function : printHTML
1056   * Summary : return the HTML template document with the specified content
1057   **/
1058  void printHTML( QByteArray * pContent ,            // pointer on the HTML content
```

```
1059             QByteArray * pRDFLink ,                // pointer on the URI of RDF representation
1060             QByteArray * pNavigation               // pointer on the navigation content
1061           )
1062 {
1063    QMap<QByteArray, QByteArray> deviceConfig;
1064    getDeviceConfig( &deviceConfig );
1065
1066    printf( "Content-type: text/html\n\n" );
1067
1068    /* HTML TEMPLATE */
1069    /* TO MODIFY THE TEMPLATE, THESE RULES MUST BE RESPECTED :
1070       - THE DOCUMENT MUST BE WRITTEN IN toStr( ... )
1071       - EACH printf PARAMETER MUST CORRESPOND TO A %s IN THE HTML DOCUMENT (READ printf DOC FOR
             MORE INFORMATIONS)
1072    */
1073
1074    printf(
1075    toStr(
1076
1077    <?xml version="1.0" encoding="UTF-8"?>
1078    <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/
          xhtml1-transitional.dtd">
1079    <html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
1080    <head>
1081       <title>%s</title>
1082       <link rel="alternate" type="application/rdf+xml" title="Semantic version" href="%s"/>
1083       <link rel="stylesheet" type="text/css" media="screen" href="http://emep.hevs.ch/style/emep-
             screen.css"/>
1084       <script type="text/javascript" src="http://emep.hevs.ch/script/ie.js"></script>
1085       <script type="text/javascript" src="http://emep.hevs.ch/script/ibox.js"></script>
1086       <script type="text/javascript">iBox.setPath('');iBox.default_width = 800;</script>
1087       <script src="http://maps.google.com/maps?file=api&amp;v=2&amp;key=
             ABQIAAAAS6oY6Bb53ZhZl_cAujQnaxQy_m-1gWUUYfSZshtNgRBgulJgtxSc_6uO8ubz_PuCuaQsNobHJVniAA"
              type="text/javascript"></script>
1088       <script type="text/javascript">
1089          //<![CDATA[
1090          function load()
1091          {
1092             if (GBrowserIsCompatible())
1093             {
1094                var deviceIcon = new GIcon();
1095                deviceIcon.image = "http://sten.hevs.ch/images/Building.png";
1096                deviceIcon.iconSize = new GSize(18, 18);
1097                deviceIcon.iconAnchor = new GPoint(9, 9);
1098                var position = new GLatLng( %s , %s );
1099                var map = new GMap2(document.getElementById("map"));
1100                map.setCenter(position, 15);
1101                map.setMapType( G_HYBRID_MAP );
1102                map.addOverlay( new GMarker( position , {icon: deviceIcon}) );
1103             }
1104          }
1105          //]]>
1106       </script>
1107    </head>
1108    <body onload="load()" onunload="GUnload()">
1109    <div class="frame">
1110       <div class="header">
1111          <h1>%s</h1>
1112          <div class="navigation">
1113             <ul>%s</ul>
1114          </div>
1115       </div>
1116       <div class="content">
1117          <a name="this"/>
1118          %s
1119       </div>
1120       <div class="footer"></div>
1121    </div>
1122    </body>
1123    </html>
1124
1125    )
1126    , deviceConfig.value( "name" ).data() , pRDFLink->data() , deviceConfig.value( "latitude" ).
```

```
        data() , deviceConfig.value( "longitude" ).data() , deviceConfig.value( "name" ).data() ,
        pNavigation->data() , pContent->data() );
}

/**
 * Function : genHTMLSelfDescription
 * Summary : prepare the content and the navigation content to send
 *           to printHTML with the informations about device
 **/
void genHTMLSelfDescription()
{
   QByteArray content, rdflink, navigation;

   QMap<QByteArray, QByteArray> deviceConfig;
   getDeviceConfig( &deviceConfig );

   // get the sensors informations
   QList<QByteArray> sensorsIDs, measTypes, sensorsPos, paramNames, systemNames;
   getSensorsInfos( &sensorsIDs , &measTypes , &sensorsPos , &paramNames , &systemNames );

   navigation.append( toStr ( <li><a href="http://sten.hevs.ch">S-TEN</a></li><li><a href="http
       ://EMEP.hevs.ch">EMEP a S-TEN Application</a></li><li><a href="http://registry.emep.hevs.
       ch">EMEP Registry</a></li><li>Measurements<ul> ) );

   content.append( "<table style=\"width:500px;\"><tr><th>Name:</th><td>" + deviceConfig.value( "
       baseUrl" ) +
           "</td></tr><tr><th>Altitude:</th><td>" + deviceConfig.value( "altitude" ) +
           " m</td></tr><tr><th>Latitude:</th><td>" + deviceConfig.value( "latitude" ) +
           " &deg;</td></tr><tr><th>Longitude:</th><td>" + deviceConfig.value( "longitude" )
               +
           "&deg;</td></tr><th>Measurements:</th><td>" );

   for ( int i = 0 ; i < sensorsIDs.size() ; ++i )
   {
      navigation.append( "<li><a href=\"" + deviceConfig.value( "baseUrl" ) + "html/" + measTypes
          .at(i) + "\">" + measTypes.at(i) + "</a></li>" );

      content.append ( "<table class=\"measure\"><tr><th colspan=\"5\" class=\"centered\">" +
          measTypes.at(i) +
              "</th></tr><tr><td><a title=\"Step into TemperatureMeasure\" href=\"" +
                  deviceConfig.value( "baseUrl" ) +
              "html/" + measTypes.at(i) + "\"><img src=\"http://sten.hevs.ch/images/
                  detail_icon.png\" alt=\"detail\"/></a>"
              "</td></tr></table>" );
   }

   navigation.append( "</ul></li>" );
   navigation.squeeze();

   content.append ( "</td><tr><th>Map:</th><td><div class=\"googleSmallMap\" id=\"map\"></div></
       td></tr></table>" );
   content.squeeze();

   rdflink.append( deviceConfig.value( "baseUrl" ) + "#this" );
   rdflink.squeeze();

   printHTML( &content , &rdflink , &navigation );
}

/**
 * Function : genHTMLPeriodDescription
 * Summary : prepare the content and the navigation content to send
 *           to printHTML with the informations about the period description
 **/
void genHTMLPeriodDescription( const QList<QByteArray> * pParsedURI )
{
   // pParsedURI->at(0) -> measurement
   // pParsedURI->at(1) -> period
   QList<int> periodList;

   int nbLevels = pParsedURI->size(); // number of URI levels
   int dateLevel = 0;         // number of datetime levels
                              // for example :
```

```
1189                            // dataLevel = 2 -> 2009-04
1190
1191    if ( nbLevels == 1 )
1192    {
1193       // URI contains the measurement item only
1194       getAvailableYearsList( &periodList , &pParsedURI->at(0) );
1195    }
1196    else if ( nbLevels > 1 )
1197    {
1198       int to = getCoveredPeriod( &pParsedURI->at(1) );
1199       getPeriodList( &periodList , &pParsedURI->at(0) , &pParsedURI->at(1) , to );
1200    }
1201
1202    if ( periodList.size() > 0 )
1203    {
1204       QByteArray periodLabel;
1205
1206       if ( nbLevels == 1 )
1207       {
1208          periodLabel.append( "for Ever" );
1209       }
1210       else if ( nbLevels > 1 )
1211       {
1212          // separate date and time
1213          QList<QByteArray> parsedDateTime = pParsedURI->at(1).split( 'T' );
1214
1215          // separate year, month, day
1216          QList<QByteArray> parsedDate = parsedDateTime.at(0).split( '-' );
1217
1218          if ( parsedDateTime.size() == 1 )
1219          {
1220             dateLevel = parsedDate.size();
1221
1222             // the request contains a date only
1223             if ( dateLevel == 1 )
1224             {
1225                periodLabel.append( "for Year" );
1226             }
1227             else if ( dateLevel == 2 )
1228             {
1229                periodLabel.append( "for Month" );
1230             }
1231             else if ( dateLevel == 3 )
1232             {
1233                periodLabel.append( "for Day" );
1234             }
1235
1236             // used for parameter level in getISOTimeStamp
1237             dateLevel++;
1238          }
1239          else
1240          {
1241             periodLabel.append( "for Hour" );
1242             dateLevel = 5;
1243          }
1244       }
1245
1246       QByteArray content, rdflink, navigation;
1247       QMap<QByteArray, QByteArray> deviceConfig;
1248
1249       getDeviceConfig( &deviceConfig );
1250
1251       if ( nbLevels > 1 )
1252       {
1253          content.append ( "<h2>" + pParsedURI->at(0) + " " + periodLabel + " " + pParsedURI->at
                (1) + "</h2></h2><div class=\"googleSmallMap\" id=\"map\"></div><table class=\"graph
                \">" );
1254          rdflink.append( deviceConfig.value( "baseUrl" ) + pParsedURI->at(0) + "/" + pParsedURI->
                at(1) + "#this" );
1255       }
1256       else
1257       {
1258          content.append ( "<h2>" + pParsedURI->at(0) + " " + periodLabel + "</h2></h2><div class
```

```
            =\"googleSmallMap\" id=\"map\"></div><table class=\"graph\">" );
1259        rdflink.append( deviceConfig.value( "baseUrl" ) + pParsedURI->at(0) + "#this" );
1260     }
1261
1262     for ( int i = 0 ; i < periodList.size() ; ++i )
1263     {
1264        QByteArray ISODate;
1265
1266        if ( nbLevels > 1 )
1267        {
1268           ISODate = getISOTimeStamp( periodList.at(i) , dateLevel );
1269        }
1270        else
1271        {
1272           ISODate = QByteArray::number( periodList.at(i) );
1273        }
1274
1275        // the period is hour
1276        if ( dateLevel == 5 )
1277        {
1278           content.append ( "<tr><td><a title=\"View data\" rel=\"ibox\" href=\"" + deviceConfig
                .value( "baseUrl" ) + "html/" + pParsedURI->at(0) + "/" + ISODate + ".values\">"
                + ISODate + "</a>" );
1279        }
1280        else
1281        {
1282           content.append ( "<tr><th>" + ISODate + "</th><td>"
1283                       "<a title=\"Step into " + ISODate + "\" href=\"" + deviceConfig.value( "
                            baseUrl" ) + "html/" + pParsedURI->at(0) + "/" + ISODate + "\">"
1284                       "<img src=\"http://sten.hevs.ch/images/detail_icon.png\" alt=\"detail
                            \"/></a></td>"
1285                       "<td><a title=\"Graphic representation\" rel=\"ibox\" href=\"" +
                            deviceConfig.value( "baseUrl" ) + "html/" + pParsedURI->at(0) + "/"
                            + ISODate + ".graph\">"
1286                       "<img src=\"http://sten.hevs.ch/images/graph_icon.png\" alt=\"graph\"/></
                            a></td><td>"
1287                       "<a title=\"Download data\" href=\"" + deviceConfig.value( "baseUrl" ) +
                            "html/" + pParsedURI->at(0) + "/" + ISODate + ".values\">"
1288                       "<img src=\"http://sten.hevs.ch/images/data_icon.png\" alt=\"download
                            data file\"/></a></td>"
1289                       "<td><a title=\"Average value\" rel=\"ibox\" href=\"" + deviceConfig.
                            value( "baseUrl" ) + "html/" + pParsedURI->at(0) + "/" + ISODate + "
                            .stat\">"
1290                       "<img src=\"http://sten.hevs.ch/images/avg_icon.png\" alt=\"average\"/>"
                            );
1291        }
1292     }
1293
1294     content.append( "</table>" );
1295
1296     navigation.append( "<li><a href=\"javascript:back()\">Back</a></li><li><a href=\"" +
          deviceConfig.value( "baseUrl" ) + "html/\">About this Data Provider</a></li>" );
1297
1298     printHTML( &content , &rdflink , &navigation );
1299  }
1300  else
1301  {
1302     errorHandler( 404 , ERR_NODATA );
1303  }
1304 }
1305
1306 /**
1307  * Function : genHTMLValues
1308  * Parameter : pParsedURI pointer on the parsed URI
1309  *                       pParsedURI->at(0) : URI level 1
1310  *                       pParsedURI->at(1) : URI level 2
1311  * Summary : prepare the content and the navigation content to send to printHTML
1312  *           with the values for the specified period and measurement item
1313  **/
1314 void genHTMLValues( const QList<QByteArray> * pParsedURI )
1315 {
1316    QList<QByteArray> date;
1317    QMap<int,double> values;
```

```
1318
1319    if ( pParsedURI->size() > 1 )
1320    {
1321       date = pParsedURI->at(1).split( '.' );
1322    }
1323    else
1324    {
1325       date = pParsedURI->at(0).split( '.' );
1326    }
1327
1328    // check if the period is a minute
1329    if ( date.at(0).count('-') == 3 )
1330    {
1331       // in this case print this minute value in a popup frame
1332       getValues( &values , &pParsedURI->at(0) , getUnixTimestamp( &date.at(0) ) ,
               getCoveredPeriod( &date.at(0) ) );
1333    }
1334    else
1335    {
1336       if ( pParsedURI->size() > 1 )
1337       {
1338          // there is a specified period
1339          getValues( &values , &pParsedURI->at(0) , getUnixTimestamp( &date.at(0) ) ,
                   getCoveredPeriod( &date.at(0) ) );
1340       }
1341    }
1342
1343    if ( !values.isEmpty() )
1344    {
1345       QMap<QByteArray, QByteArray> deviceConfig;
1346       getDeviceConfig( &deviceConfig );
1347
1348       // check if the period is a minute
1349       if ( date.at(0).count('-') == 3 )
1350       {
1351          printf (
1352          toStr (
1353                Content-type: text/html\n\n
1354                <?xml version="1.0" encoding="UTF-8"?>
1355                <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/
                       xhtml1/DTD/xhtml1-transitional.dtd"><html xmlns="http://www.w3.org/1999/xhtml"
                        xml:lang="en" lang="en">
1356                <head>
1357                <title>%s of %s for minute %s</title>
1358                <link rel="stylesheet" type="text/css" media="screen" href="http://sten.hevs.ch/
                       style/emep-screen.css"/>
1359                <script type="text/javascript" src="http://sten.hevs.ch/script/ie.js"></script>
1360                </head>
1361                <body>
1362                <div class="popupframe">
1363                   <div class="header"><h1>%s</h1></div>
1364                   <div class="content"><h3>%s for minute %s</h3>
1365             )
1366          , pParsedURI->at(0).data() , deviceConfig.value( "name" ).data() , date.at(0).data() ,
               deviceConfig.value( "name" ).data() , pParsedURI->at(0).data() , date.at(0).data() )
               ;
1367
1368          printf( "<table><tr><th class=\"big\">Value</th>" );
1369
1370          QMapIterator<int, double> i(values);
1371          while (i.hasNext()) {
1372             i.next();
1373             printf( "<td class=\"big\">%.2f</td>" , i.value() );
1374          }
1375
1376          printf( "</tr></table></div><div class=\"footer\"></div></div></body></html>" );
1377       }
1378       else
1379       {
1380          printf( "Content-type: text/csv\nContent-disposition: attachment; filename=\"data.csv\"\
                   n\n" );
1381          printf( "Station: %s at %s m Latitude %s Longitude %s" , deviceConfig.value( "name" ).
                   data() , deviceConfig.value( "altitude" ).data() , deviceConfig.value( "latitude" ).
```

```
                data() , deviceConfig.value( "longitude" ).data() );
1382
1383            // print measurement item without parameter
1384            if ( pParsedURI->size() > 1 )
1385            {
1386               printf( "\n%s;value\n" , pParsedURI->at(0).data() );
1387            }
1388            else
1389            {
1390               printf( "\n%s;value\n" , date.at(0).data() );
1391            }
1392
1393            QMapIterator<int, double> i(values);
1394            while (i.hasNext()) {
1395               i.next();
1396               printf( "%s;%f\n" , getISOTimeStamp( i.key() , 6 ).data() , i.value() );
1397            }
1398         }
1399      }
1400      else
1401      {
1402         errorHandler( 404 , ERR_NODATA );
1403      }
1404   }
1405
1406   /**
1407    * Function : genHTMLGraph
1408    * Parameter : pParsedURI pointer on the parsed URI
1409    *                      pParsedURI->at(0) : URI level 1
1410    *                      pParsedURI->at(1) : URI level 2
1411    * Summary : prepare the content and the navigation content to send to printHTML
1412    *           with a graph that plots the staticstics values for each sub period
1413    *           of the corresponding measurement item
1414    **/
1415   void genHTMLGraph( const QList<QByteArray> * pParsedURI )
1416   {
1417      float xGrid, yGrid;
1418      int dateLevel;
1419
1420      QList<int> periodList;
1421      QByteArray periodLabel;
1422
1423      // separate period from parameter
1424      QList<QByteArray> date = pParsedURI->at(1).split( '.' );
1425
1426      // separate date and time
1427      QList<QByteArray> parsedDateTime = date.at(0).split( 'T' );
1428
1429      // separate year, month, day
1430      QList<QByteArray> parsedDate = parsedDateTime.at(0).split( '-' );
1431
1432      if ( parsedDateTime.size() == 1 )
1433      {
1434         dateLevel = parsedDate.size();
1435
1436         // the request contains a date only
1437         if ( dateLevel == 1 )
1438         {
1439            periodLabel.append( "for Year" );
1440         }
1441         else if ( dateLevel == 2 )
1442         {
1443            periodLabel.append( "for Month" );
1444         }
1445         else if ( dateLevel == 3 )
1446         {
1447            periodLabel.append( "for Day" );
1448         }
1449
1450         // used for parameter level in getISOTimeStamp()
1451         dateLevel++;
1452      }
1453      else
```

```
1454    {
1455       periodLabel.append( "for Hour" );
1456       dateLevel = 5;
1457    }
1458
1459    int to = getCoveredPeriod( &date.at(0) );
1460    getPeriodList( &periodList , &pParsedURI->at(0) , &date.at(0) , to );
1461
1462    QByteArray maxValues, avgValues, minValues, period;
1463
1464    QMap<QByteArray, QByteArray> deviceConfig;
1465    getDeviceConfig( &deviceConfig );
1466
1467    // print the HTML document
1468    printf (
1469    toStr (
1470    Content-type: text/html\n\n
1471    <?xml version="1.0" encoding="UTF-8"?>
1472    <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/
         xhtml1-transitional.dtd">
1473    <html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
1474    <head>
1475       <title>%s of %s %s %s</title>
1476       <link rel="stylesheet" type="text/css" media="screen" href="http://sten.hevs.ch/style/emep-
             screen.css"/>
1477       <script type="text/javascript" src="http://sten.hevs.ch/script/ie.js"></script>
1478    </head>
1479    <body>
1480       <div class="popupframe">
1481          <div class="header"><h1>%s</h1></div>
1482          <div class="content"><h3>%s %s %s</h3>
1483    )
1484    , pParsedURI->at(0).data() , deviceConfig.value( "name" ).data() , periodLabel.data() , date.
         at(0).data() , deviceConfig.value( "name" ).data() , pParsedURI->at(0).data() ,
         periodLabel.data() , date.at(0).data() );
1485
1486    // prepare the graph with statistics values for each sub period
1487    printf ( "<img width=\"600\" height=\"400\" src=\"http://chart.apis.google.com/chart?chs=600
         x400&amp;cht=lc&amp;chd=t:" );
1488
1489    for ( int i = 0 ; i < periodList.size() ; i++ )
1490    {
1491       QByteArray xPeriod = getISOTimeStamp( periodList.at(i) , dateLevel );
1492
1493       QList<double> stats;
1494       getStats( &stats , &pParsedURI->at(0) , &xPeriod );
1495
1496       if ( !stats.isEmpty() )
1497       {
1498          if ( i == 0 )
1499          {
1500             minValues.append( QByteArray::number(stats.at(0),'f',4) );
1501             maxValues.append( QByteArray::number(stats.at(1),'f',4) );
1502             avgValues.append( QByteArray::number(stats.at(2),'f',4) );
1503          }
1504          else
1505          {
1506             minValues.append( "," + QByteArray::number(stats.at(0),'f',4) );
1507             maxValues.append( "," + QByteArray::number(stats.at(1),'f',4) );
1508             avgValues.append( "," + QByteArray::number(stats.at(2),'f',4) );
1509          }
1510       }
1511
1512       xPeriod.remove( 0 , xPeriod.size() - 2 );
1513       period.append( "|" + xPeriod );
1514    }
1515
1516    if ( periodList.size() == 1 )
1517    {
1518       xGrid = 100;
1519    }
1520    else
1521    {
```

```
1522        xGrid = (float) 100 / ( periodList.size() - 1 );
1523     }
1524
1525     // get the range of the parameter to fix y-axis min and max values
1526     QList<int> range;
1527     getRange( &range , &pParsedURI->at(0) );
1528
1529     // calculate the number of grids on the y-axis
1530     yGrid = (float) 100 / ( ( range.at(1) - range.at(0) ) / 5 ) ;
1531
1532     printf( "%s|%s|%s&amp;chds=%d,%d,%d,%d&amp;chxt=x,y&amp;chxr=1,%d,%d&amp;chxl=0:%s&amp;chco=
             BFDCEB,0076B3,BFDCEB&amp;chm=b,BFDCEB,0,2,0&amp;chf=bg,s,FFFFFF&amp;chg=\%f,%f\" alt=\"
             Google charts\"/></div><div class=\"footer\"></div></div></body></html>" , maxValues.data
             () , avgValues.data() , minValues.data() , range.at(0) , range.at(1) , range.at(0) , range
             .at(1) , range.at(0) , range.at(1) , period.data() , xGrid , yGrid );
1533 }
1534
1535 /**
1536  * Function : genHTMLStats
1537  * Parameter : pParsedURI pointer on the parsed URI
1538  *                      pParsedURI->at(0) : URI level 1
1539  *                      pParsedURI->at(1) : URI level 2
1540  * Summary : prepare the content and the navigation content to send to printHTML
1541  *           with an histogramm that plots the count of measurements in
1542  *           different ranges of values and with the statistics values for the
1543  *           corresponding measurement item
1544  **/
1545 void genHTMLStats( const QList<QByteArray> * pParsedURI )
1546 {
1547    QList<double> stats;
1548    QList<int> range;
1549
1550    int dateLevel, fromTS, toTS;
1551    QByteArray periodLabel;
1552    QByteArray measureItem;
1553    QList<QByteArray> date;
1554
1555    QMap<QByteArray, QByteArray> deviceConfig;
1556    getDeviceConfig( &deviceConfig );
1557
1558    // separate period from parameter
1559    date = pParsedURI->at(1).split( '.' );
1560
1561    // separate date and time
1562    QList<QByteArray> parsedDateTime = date.at(0).split( 'T' );
1563
1564    // separate year, month, day
1565    QList<QByteArray> parsedDate = parsedDateTime.at(0).split( '-' );
1566
1567    if ( parsedDateTime.size() == 1 )
1568    {
1569       dateLevel = parsedDate.size();
1570
1571       // the request contains a date only
1572       if ( dateLevel == 1 )
1573       {
1574          periodLabel.append( "for Year" );
1575       }
1576       else if ( dateLevel == 2 )
1577       {
1578          periodLabel.append( "for Month" );
1579       }
1580       else if ( dateLevel == 3 )
1581       {
1582          periodLabel.append( "for Day" );
1583       }
1584    }
1585
1586    measureItem = pParsedURI->at(0);
1587
1588    getStats( &stats , &pParsedURI->at(0) , &date.at(0) );
1589
1590    // get min and max range for graph axis
```

```
1591    getRange( &range , &pParsedURI->at(0) );
1592
1593    printf (
1594    toStr (
1595          Content-type: text/html\n\n
1596          <?xml version="1.0" encoding="UTF-8"?><!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
                  Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd"><html xmlns="
                  http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
1597          <head>
1598          <title>%s of %s %s %s</title>
1599          <link rel="stylesheet" type="text/css" media="screen" href="http://sten.hevs.ch/style/
                  emep-screen.css"/>
1600          <script type="text/javascript" src="http://sten.hevs.ch/script/ie.js"></script>
1601          </head>
1602          <body>
1603             <div class="popupframe">
1604                <div class="header"><h1>%s</h1></div>
1605                <div class="content">
1606                   <h3>%s %s %s</h3>
1607                   <table class="graph">
1608    )
1609    , pParsedURI->at(0).data() , deviceConfig.value( "name" ).data() , periodLabel.data() , date.
          at(0).data() , deviceConfig.value( "name" ).data() , pParsedURI->at(0).data() ,
          periodLabel.data() , date.at(0).data() );
1610
1611    // prepare time interval for getCount function
1612    fromTS = getUnixTimestamp( &date.at(0) );
1613    toTS = getCoveredPeriod( &date.at(0) );
1614
1615    // histogram graph generation
1616    if ( !stats.isEmpty() )
1617    {
1618       int countMax = 0;
1619       int count;
1620
1621       printf( "<tr><td rowspan=\"6\" style=\"text-align: left\" width=\"500\"><img width=\"300\"
              height=\"400\" src=\"http://chart.apis.google.com/chart?chs=300x400&chco=BFDCEB&cht=bhs
              &chxt=x,y,y&chd=t:") ;
1622
1623       // get count for each step in the range
1624       for ( int i = range.at(1) ; i > range.at(0) ; i -= 5 )
1625       {
1626          // get the number of measurements in this value range and time interval
1627          count = getCount( &measureItem , i , i - 5 , fromTS , toTS );
1628
1629          if ( count > countMax )
1630             countMax = count;
1631
1632          if ( i == range.at(1) )
1633          {
1634             printf( "%d" , count );
1635          }
1636          else
1637          {
1638             printf( ",%d" , count );
1639          }
1640       }
1641
1642       // calculate min, max and avg position on the axis (rule of 3)
1643       double minPos = ( ( stats.at(0) + abs( range.at(0) ) ) * 100 ) / ( range.at(1) - range.at
              (0) );
1644       double maxPos = ( ( stats.at(1) + abs( range.at(0) ) ) * 100 ) / ( range.at(1) - range.at
              (0) );
1645       double avgPos = ( ( stats.at(2) + abs( range.at(0) ) ) * 100 ) / ( range.at(1) - range.at
              (0) );
1646
1647       printf( "&chds=0,%d&chxr=0,1,%d|1,%.1f,%.1f&chm=N*f0*,555555,0,-1,10,0&chbh=10,8,0&chxl=2:|
              min|max|avg&chxp=2,%.1f,%.1f,%.1f&chxs=2,DD00DD,9,-1,lt,DD00DD&chxtc=2,10\"/>" ,
              countMax , countMax , range.at(0) + 2.5 , range.at(1) - 2.5 , minPos , maxPos , avgPos
              );
1648       printf( "</td><th style=\"text-align: center\">Max</th></tr><tr><td style=\"text-align:
              center\">%.2f</td></tr><tr><th style=\"text-align: center\">Average</th></tr><tr><td
              style=\"text-align: center\">%.2f</td></tr><tr><th style=\"text-align: center\">Min</th
```

```
            ></tr><tr><td style=\"text-align: center\">%.2f</td></tr>" , stats.at(1) , stats.at(2)
            , stats.at(0) );
1649     }
1650
1651     printf( "</table></div><div class=\"footer\"></div></div></body></html>" );
1652  }
1653
1654  /**
1655   * Function : getUnixTimestamp
1656   * Parameter : pDate pointer on the ISO date to convert
1657   * Summary : return the Unix timestamp format of an ISO date
1658   **/
1659  int getUnixTimestamp( const QByteArray * pDate )
1660  {
1661     QDateTime timeStamp;
1662
1663     // check the date level
1664     if ( pDate->count( '-' ) == 4 )
1665     {
1666        timeStamp = QDateTime::fromString( *pDate , "yyyy-MM-ddThh-mm-ss" );
1667     }
1668     else if ( pDate->count( '-' ) == 3 )
1669     {
1670        timeStamp = QDateTime::fromString( *pDate , "yyyy-MM-ddThh-mm" );
1671     }
1672     else if ( pDate->count( '-' ) == 2 )
1673     {
1674        if ( pDate->count( 'T' ) == 1 )
1675        {
1676           timeStamp = QDateTime::fromString( *pDate , "yyyy-MM-ddThh" );
1677        }
1678        else
1679        {
1680           timeStamp = QDateTime::fromString( *pDate , "yyyy-MM-dd" );
1681        }
1682     }
1683     else if ( pDate->count( '-' ) == 1 )
1684     {
1685        timeStamp = QDateTime::fromString( *pDate , "yyyy-MM" );
1686     }
1687     else
1688     {
1689        timeStamp = QDateTime::fromString( *pDate , "yyyy" );
1690     }
1691
1692     return timeStamp.toTime_t();
1693  }
1694
1695  /**
1696   * Function : getISOTimeStamp
1697   * Summary : return a period in ISO format from a Unix timestamp
1698   **/
1699  QByteArray getISOTimeStamp( const int UNIXtimeStamp , // Unix timestamp
1700                              int level                 // parameter to choose the ISO format level
1701                            )
1702  {
1703     QDateTime ISOtimeStamp = QDateTime::fromTime_t( UNIXtimeStamp );
1704
1705     if ( level == 1 )
1706     {
1707        return ISOtimeStamp.toString( "yyyy" ).toAscii();
1708     }
1709     else if ( level == 2 )
1710     {
1711        return ISOtimeStamp.toString( "yyyy-MM" ).toAscii();
1712     }
1713     else if ( level == 3 )
1714     {
1715        return ISOtimeStamp.toString( "yyyy-MM-dd" ).toAscii();
1716     }
1717     else if ( level == 4 )
1718     {
1719        return ISOtimeStamp.toString( "yyyy-MM-ddThh" ).toAscii();
```

```
1720     }
1721     else if ( level == 5 )
1722     {
1723        return ISOtimeStamp.toString( "yyyy-MM-ddThh-mm" ).toAscii();
1724     }
1725     else
1726     {
1727        return ISOtimeStamp.toString( "yyyy-MM-ddThh-mm-ss" ).toAscii();
1728     }
1729  }
1730
1731  /**
1732   * Function : getCoveredPeriod
1733   * Parameter : pISODate pointer on the ISO date (begin of time interval)
1734   * Summary : return the end of interval depending on the start interval date
1735   **/
1736  int getCoveredPeriod( const QByteArray * pISODate )
1737  {
1738     QByteArray to;
1739     int toUTS = -1;
1740     char twoDigitsNumber [2];
1741
1742     if ( pISODate->count( '-' ) == 0 )
1743     {
1744        // date contains a year only
1745        to.setNum( pISODate->toInt() + 1 );
1746        toUTS = getUnixTimestamp( &to );
1747     }
1748     else if ( pISODate->count( '-' ) == 1 )
1749     {
1750        // date contains a year and a month
1751        // date.at(0) contains the year
1752        // date.at(1) contains the month
1753        QList<QByteArray> date = pISODate->split( '-' );
1754
1755        // check if the month is december
1756        if ( date.at(1).toInt() == 12 )
1757        {
1758           // end time interval is the first january
1759           // of the next year
1760           to.setNum( date.at(0).toInt() + 1 );
1761           toUTS = getUnixTimestamp( &to );
1762        }
1763        else
1764        {
1765           // end time interval is the first day of the next month
1766           sprintf( twoDigitsNumber, "%02d" , date.at(1).toInt() + 1 );
1767           to.append( date.at(0) + "-" );
1768           to.append( twoDigitsNumber );
1769           toUTS = getUnixTimestamp( &to );
1770        }
1771     }
1772     else if ( pISODate->count( '-' ) == 2 )
1773     {
1774         // check if the date contains an hour
1775        if ( pISODate->count( 'T' ) == 0 )
1776        {
1777           to.append( *pISODate + "T23-59-59" );
1778        }
1779        else
1780        {
1781           to.append( *pISODate + "-59-59" );
1782        }
1783
1784        toUTS = getUnixTimestamp( &to );
1785        // add second
1786        // that's the time interval end
1787        toUTS++;
1788     }
1789     else
1790     {
1791        to.append( *pISODate + "-59" );
1792        // convert to UNIX timestamp
```

```
1793        toUTS = getUnixTimestamp( &to );
1794        // add second
1795        // that's the time interval end
1796        toUTS++;
1797     }
1798
1799     return toUTS;
1800  }
1801
1802  /**
1803   * Function : getPeriodList
1804   * Summary : return the periods in a time interval
1805   **/
1806  void getPeriodList( QList<int> * pPeriodList ,  // where to return the periods
1807                      const QByteArray * pMeasureItem , // pointer on the measurement item name
1808                      const QByteArray * from ,        // pointer on the requested period (begin of
                            time interval)
1809                      const int to                     // end of time interval in Unix timestamp
                            format
1810                )
1811  {
1812     QSqlQuery query;
1813
1814     // check in which table the periods must be selected
1815     if ( from->count( '-' ) >= 2 )
1816     {
1817        // the requested period contains at least a year, a month and a day
1818        query = db.exec( "SELECT measTime "
1819                         "FROM Meas_test2 "
1820                         "WHERE idSensor = ( SELECT idSensor FROM Sensor WHERE measType = '" + *
                                pMeasureItem + "' ) "
1821                         "AND measTime >= " + QByteArray::number( getUnixTimestamp( from ) ) + " AND
                                measTime < " + QByteArray::number( to )
1822                       );
1823
1824        if ( ( from->count( '-' ) == 2 ) && ( from->count( 'T' ) == 0 ) )
1825        {
1826           // subperiod of day
1827           if ( query.isActive() )
1828           {
1829              while ( query.next() )
1830              {
1831                 // avoid repetitions
1832                 if ( pPeriodList->count() == 0 || pPeriodList->last() <= query.value(0).toInt() -
                         3600 )
1833                    pPeriodList->append( query.value(0).toInt() );
1834              }
1835           }
1836        }
1837        else
1838        {
1839           // subperiod of hour
1840           if ( query.isActive() )
1841           {
1842              while ( query.next() )
1843              {
1844                 // avoid repetitions
1845                 if ( pPeriodList->count() == 0 || pPeriodList->last() <= query.value(0).toInt() -
                         60 )
1846                    pPeriodList->append( query.value(0).toInt() );
1847              }
1848           }
1849        }
1850     }
1851     else
1852     {
1853        // the requested period contains a year and/or a month
1854        if ( from->count( '-' ) == 1 )
1855        {
1856           query = db.exec( "SELECT timeStamp "
1857                            "FROM Stat_days "
1858                            "WHERE idSensor = ( SELECT idSensor FROM Sensor WHERE measType = '" + *
                                  pMeasureItem + "' ) "
```

```
1859                        "AND timeStamp >= " + QByteArray::number( getUnixTimestamp( from ) ) + " AND
                                timeStamp < " + QByteArray::number( to )
1860                       );
1861       }
1862       else
1863       {
1864          query = db.exec( "SELECT timeStamp "
1865                        "FROM Stat_months "
1866                        "WHERE idSensor = ( SELECT idSensor FROM Sensor WHERE measType = '" + *
                              pMeasureItem + "' ) "
1867                        "AND timeStamp >= " + QByteArray::number( getUnixTimestamp( from ) ) + " AND
                                timeStamp < " + QByteArray::number( to )
1868                       );
1869       }
1870
1871       if ( query.isActive() )
1872       {
1873          while ( query.next() )
1874          {
1875             pPeriodList->append( query.value(0).toInt() );
1876          }
1877       }
1878    }
1879 }
1880
1881 /**
1882  * Function : getValues
1883  * Summary : return the measurement values for the corresponding measurement item
1884  *           and in a time interval
1885  **/
1886 void getValues( QMap<int,double> * pValues ,       // where to return values
1887             const QByteArray * pMeasureItem ,      // pointer on the measurement item name
1888             const int from ,                       // begin of time interval in Unix timestamp
                        format
1889             const int to                           // end of time interval in Unix timestamp
                        format
1890           )
1891 {
1892    QSqlQuery query;
1893
1894    query = db.exec( "SELECT measTime, value "
1895                  "FROM Meas_test2 "
1896                  "WHERE idSensor = ( SELECT idSensor FROM Sensor WHERE measType = '" + *
                          pMeasureItem + "' ) "
1897                  "AND measTime >= " + QByteArray::number(from) + " AND measTime < " + QByteArray::
                          number(to) );
1898
1899    if ( query.isActive() )
1900    {
1901       while ( query.next() )
1902       {
1903          pValues->insertMulti( query.value(0).toInt() , query.value(1).toDouble() );
1904       }
1905    }
1906 }
1907
1908 /**
1909  * Function : getStats
1910  * Summary : return the statistics values for the corresponding measurement item
1911  *           and in a time interval
1912  **/
1913 void getStats( QList<double> * pStats ,              // where to return statistic values
1914             const QByteArray * pMeasureItem ,        // pointer on the measurement item name
1915             const QByteArray * pPeriod               // pointer on the requested period (ISO
                        format)
1916           )
1917 {
1918    QSqlQuery query;
1919
1920    int from, to;
1921
1922    // get time interval in Unix timestamp format
1923    from = getUnixTimestamp( pPeriod );
```

```
1924     to = getCoveredPeriod( pPeriod );
1925
1926     // check in which table the select must be done (depending on the period level)
1927     if ( pPeriod->count('-') == 0 )
1928     {
1929        // period is year
1930        query = db.exec( "SELECT Stat_years.min, Stat_years.max, Stat_years.avg "
1931                    "FROM Stat_years "
1932                    "WHERE idSensor = ( "
1933                       "SELECT idSensor "
1934                       "FROM Sensor "
1935                       "WHERE measType = '" + *pMeasureItem + "' "
1936                    ") "
1937                    "AND timeStamp >= " + QByteArray::number( from ) + " "
1938                    "AND timeStamp < " + QByteArray::number( to ) );
1939
1940     }
1941     else if ( pPeriod->count('-') == 1 )
1942     {
1943        // period is month
1944        query = db.exec( "SELECT Stat_months.min, Stat_months.max, Stat_months.avg "
1945                    "FROM Stat_months "
1946                    "WHERE idSensor = ( "
1947                       "SELECT idSensor "
1948                       "FROM Sensor "
1949                       "WHERE measType = '" + *pMeasureItem + "' "
1950                    ") "
1951                    "AND timeStamp >= " + QByteArray::number( from ) + " "
1952                    "AND timeStamp < " + QByteArray::number( to ) );
1953     }
1954     else if ( pPeriod->count('-') > 1 )
1955     {
1956        if ( pPeriod->count('T') == 0 )
1957        {
1958           // period is day
1959           query = db.exec( "SELECT Stat_days.min, Stat_days.max, Stat_days.avg "
1960                       "FROM Stat_days "
1961                       "WHERE idSensor = ( "
1962                          "SELECT idSensor "
1963                          "FROM Sensor "
1964                          "WHERE measType = '" + *pMeasureItem + "' "
1965                       ") "
1966                       "AND timeStamp >= " + QByteArray::number( from ) + " "
1967                       "AND timeStamp < " + QByteArray::number( to ) );
1968        }
1969        else
1970        {
1971           // period is hour or minute
1972           query = db.exec( "SELECT MIN(value), MAX(value), AVG(value) "
1973                       "FROM Meas_test2 "
1974                       "WHERE idSensor = ( "
1975                          "SELECT idSensor "
1976                          "FROM Sensor "
1977                          "WHERE measType = '" + *pMeasureItem + "' "
1978                       ") "
1979                       "AND measTime >= " + QByteArray::number( from ) + " "
1980                       "AND measTime < " + QByteArray::number( to ) );
1981        }
1982     }
1983
1984     if ( query.isActive() )
1985     {
1986        while ( query.next() )
1987        {
1988           pStats->append( query.value(0).toDouble() );
1989           pStats->append( query.value(1).toDouble() );
1990           pStats->append( query.value(2).toDouble() );
1991        }
1992     }
1993  }
1994
1995  /**
1996   * Function : getRange
```

```
1997  * Summary : return the range corresponding to the parameter measured
1998  *           by the measurement item which fix the min and max value
1999  *           of the plots y-axis
2000  **/
2001 void getRange( QList<int> * pRange ,               // where to return range
2002             const QByteArray * pMeasureItem         // pointer on the measurement item name
2003           )
2004 {
2005    QSqlQuery query = db.exec( "SELECT minRange, maxRange "
2006                         "FROM Graphs_config "
2007                         "WHERE idSensor = ( "
2008                             "SELECT idSensor "
2009                             "FROM Sensor "
2010                             "WHERE measType = '" + *pMeasureItem + "' "
2011                         ")" );
2012
2013    if ( query.isActive() )
2014    {
2015       while ( query.next() )
2016       {
2017          pRange->append( query.value(0).toInt() );
2018          pRange->append( query.value(1).toInt() );
2019       }
2020    }
2021 }
2022
2023 /**
2024  * Function : getCount
2025  * Summary : return number of measurements which have a value
2026  *           in the value interval and which have been measured
2027  *           during the time interval
2028  **/
2029 int getCount( QByteArray * pMeasureItem ,         // pointer on the measurement item name
2030             const int from ,                        // begin of the value interval
2031             const int to ,                          // end of the value interval
2032             const int fromTS ,                      // begin of the time interval in Unix
                    timestamp format
2033             const int toTS                          // end of the time interval in Unix timestamp
                    forma
2034           )
2035 {
2036    int count = 0;
2037    QSqlQuery query;
2038
2039    // check if the count must be done for ever
2040    if ( fromTS == 0 )
2041    {
2042       query = db.exec( "SELECT count(*) "
2043                   "FROM Meas_test2 "
2044                   "WHERE idSensor = ( "
2045                     "SELECT idSensor "
2046                     "FROM Sensor "
2047                     "WHERE measType = '" + *pMeasureItem + "' "
2048                   ") "
2049                   "AND value <= " + QByteArray::number(from) + " "
2050                   "AND value > " + QByteArray::number(to) );
2051    }
2052    else
2053    {
2054       query = db.exec( "SELECT count(*) "
2055                   "FROM Meas_test2 "
2056                   "WHERE idSensor = ( "
2057                     "SELECT idSensor "
2058                     "FROM Sensor "
2059                     "WHERE measType = '" + *pMeasureItem + "' "
2060                     ") "
2061                   "AND measTime >= " + QByteArray::number(fromTS) + " "
2062                   "AND measTime < " + QByteArray::number(toTS) + " "
2063                   "AND value <= " + QByteArray::number(from) + " "
2064                   "AND value > " + QByteArray::number(to) );
2065    }
2066
2067    if ( query.isActive() )
```

```
2068     {
2069        while ( query.next() )
2070        {
2071           count = query.value(0).toInt();
2072        }
2073
2074        return count;
2075     }
2076     else
2077     {
2078        return 0;
2079     }
2080  }
```

# ANNEXE F

# Code : classe d'acquisition

## F.1 Header file

```
1  /*
2   - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
3    Filename : polldaemon.h
4    Project : Embedded Measurement Plateform
5    Part : Acquisition
6
7    Author : Johann Seydoux <johann dot seydoux at students dot hevs dot ch>
8    Version : 1.0
9    Created : 2009-06-25
10   Modified : 2009-06-26
11
12   Summary :
13   This class holds a timer which send an acquisition request to the sensor
14   through a daemon after each sensor sample period. It write measurements
15   in database and calculate and update statistics in database.
16  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
17 */
18
19 #ifndef POLLDAEMON_H
20   #define POLLDAEMON_H
21
22 // THE DATABASE PATH
23 #define PATH_TO_DB "path_to_database/database.sqlite3"
24
25 #include <QtSql>
26 #include <QTimer>
27
28 enum statsTarget { YEARS , MONTHS , DAYS };
29
30 class PollDaemon : public QObject
31 {
32   Q_OBJECT;
33
34 private:
35   QSqlDatabase db;
36        QTimer *AQtimer;
37        int idSensor;
38        int samplePeriod;
39
40        void insertSample( double , QDateTime * );
41        void updateStats( double , QDateTime * , int );
42 private slots :
43        void onSamplePeriod();
44 public:
45   PollDaemon(void);
46        PollDaemon( const char * );
47   ~PollDaemon(void);
48
49   virtual double readSensor() = 0;
50 };
51
52 #endif
```

## F.2 Source file

```
/*
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
  Filename : polldaemon.cpp
  Project : Embedded Measurement Plateform
  Part : Acquisition

  Author : Johann Seydoux <johann dot seydoux at students dot hevs dot ch>
  Version : 1.0
  Created : 2009-06-25
  Modified : 2009-06-26

  Summary :
  This is the implementation of the PollDaemon class. This class holds a timer
  which send an acquisition request to the sensor through a daemon after each
  sensor sample period. It write measurements in database and calculate and
  update statistics in database.
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
*/

#include "polldaemon.h"

/**
 * Function :  constructor
 * Parameters : pDaemonAddr pointer on daemon address
 * Summary :
 * Create the database connection, read sensor configuration in database
 * and start the acquisition timer
 **/

PollDaemon::PollDaemon( const char * pDaemonAddr )
{
   // connect to database
   db = QSqlDatabase::addDatabase( "QSQLITE" , QByteArray::QByteArray(pDaemonAddr) );
   db.setDatabaseName( PATH_TO_DB );

   if ( db.open() )
   {
      // get sensor configuration
      QSqlQuery query = db.exec( "SELECT idSensor, samplePeriod "
                           "FROM Sensor "
                           "WHERE daemon_name = '"
                           + QByteArray::QByteArray(pDaemonAddr) +
                           "'" );

      if ( query.isActive() )
      {
            while ( query.next() )
            {
                  idSensor = query.value(0).toInt();
                  samplePeriod = query.value(1).toInt();
            }
      }

      // start acquisition timer
      QTimer *AQtimer = new QTimer(this);
      connect(AQtimer, SIGNAL(timeout()), this, SLOT(onSamplePeriod()));
      AQtimer->start( samplePeriod * 60000 );
   }
   else
   {
      exit(1);
   }
}

/**
 * Function :  insertSample
 * Parameters : value     value of measurement
 *              measTime  pointer on sample time
 * Summary :
```

```
70  * Insert the sample into database
71  **/
72 void PollDaemon::insertSample( double value , QDateTime *measTime )
73 {
74    QSqlQuery query( "INSERT INTO Meas_test2( measTime , value , idSensor ) "
75                  "VALUES ( "
76                  + QByteArray::number( measTime->toTime_t() ) + " , "
77                  + QByteArray::number( value ) + " , "
78                  + QByteArray::number( idSensor ) + " ) " , db );
79 }
80
81 /**
82  * Function :  onSamplePeriod
83  * Parameters : none
84  * Summary :
85  * Read the sensor value and call functions which
86  * insert it in database and calculate statistical
87  * values at each sample period
88  **/
89 void PollDaemon::onSamplePeriod(void)
90 {
91    // read the sensor sample
92    double value = readSensor();
93
94    QDateTime cDateTime( QDateTime::currentDateTime() );
95
96    // forward sample and datetime
97    insertSample( value , &cDateTime );
98    updateStats( value , &cDateTime , YEARS );
99    updateStats( value , &cDateTime , MONTHS );
100   updateStats( value , &cDateTime , DAYS );
101 }
102
103 /**
104  * Function :  updateStats
105  * Parameters : value      value of measurement
106  *              measTime   pointer on sample time
107  *              target     db stat table type
108  * Summary :
109  * Calculate the statistics and update the target table value with them
110  **/
111 void PollDaemon::updateStats( double value , QDateTime *measTime , int target )
112 {
113    QByteArray tableName;
114    int fromTS = 0, toTS = 0, nbmeas = 0;
115    double avg = 0, min = 0, max = 0;
116
117    if ( target == YEARS )
118    {
119       tableName.append( "Stat_years" );
120
121       QDateTime year( QDate::fromString( measTime->toString( "yyyy" ) , "yyyy" ) );
122       fromTS = year.toTime_t();
123       toTS = year.addYears(1).toTime_t();
124    }
125    else if ( target == MONTHS )
126    {
127       tableName.append( "Stat_months" );
128
129       QDateTime month( QDate::fromString( measTime->toString( "yyyy-MM" ) , "yyyy-MM" ) );
130       fromTS = month.toTime_t();
131       toTS = month.addMonths(1).toTime_t();
132    }
133    else if ( target == DAYS )
134    {
135       tableName.append( "Stat_days" );
136
137       QDateTime day( QDate::fromString( measTime->toString( "yyyy-MM-dd" ) , "yyyy-MM-dd" ) );
138       fromTS = day.toTime_t();
139       toTS = day.addDays(1).toTime_t();
140    }
141
142    QSqlQuery query( "SELECT avg, min, max, nbmeas "
```

```
143                 "FROM " + tableName + " "
144                 "WHERE idSensor = " + QByteArray::number( idSensor ) + " "
145                 "AND timeStamp >= " + QByteArray::number( fromTS ) + " "
146                 "AND timeStamp < " + QByteArray::number( toTS ) , db );
147
148    if ( query.isActive() )
149    {
150       while ( query.next() )
151       {
152             avg = query.value(0).toDouble();
153             min = query.value(1).toDouble();
154             max = query.value(2).toDouble();
155             nbmeas = query.value(3).toInt();
156       }
157    }
158
159    if ( nbmeas > 0 )
160    {
161       // calculate the new average
162       avg = ( ( avg * nbmeas ) + value ) / ( nbmeas + 1);
163
164       if ( value > max )
165       {
166          max = value;
167       }
168       else if ( value < min )
169       {
170          min = value;
171       }
172
173       nbmeas++;
174
175       query.exec( "UPDATE " + tableName + " "
176                "SET "
177                "avg = " + QByteArray::number( avg ) + " , "
178                "min = " + QByteArray::number( min ) + " , "
179                "max = " + QByteArray::number( max ) + " , "
180                "nbmeas = " + QByteArray::number( nbmeas ) + " "
181                "WHERE timeStamp = " + QByteArray::number( fromTS ) + " "
182                "AND idSensor = " + QByteArray::number( idSensor ) );
183    }
184    else
185    {
186       query.exec( "INSERT INTO " + tableName + "( avg, min , max , nbmeas , timeStamp , idSensor
           ) "
187                "VALUES ( "
188                + QByteArray::number( value ) + " , "
189                + QByteArray::number( value ) + " , "
190                + QByteArray::number( value ) + " , "
191                + QByteArray::number( ++nbmeas ) + " , "
192                + QByteArray::number( fromTS ) + " , "
193                + QByteArray::number( idSensor ) + ")" );
194    }
195 }
196
197 /**
198  * Function :  destructor
199  * Summary :
200  * Close the database connection
201  **/
202 PollDaemon::~PollDaemon(void)
203 {
204    db.close();
205    delete AQtimer;
206 }
```

# ANNEXE G

## Code : adaptation de l'application de publication pour le test mémoire

```
1  int main(int argc, char **argv)
2  {
3     QByteArray SCRIPT_NAME("");
4     QByteArray PATH_INFO("");
5     QByteArray HTTP_ACCEPT("");
6
7     if ( argc == 4 )
8     {
9        SCRIPT_NAME.append( argv[1] );
10       PATH_INFO.append( argv[2] );
11       HTTP_ACCEPT.append( argv[3] );
12    }
13
14    // Setting database connection
15    db = QSqlDatabase::addDatabase( "QSQLITE" , "fromDataProviderApp" );
16    db.setDatabaseName( PATH_TO_DB );
17
18    if ( db.open() )
19    {
20       QByteArray reqURI( SCRIPT_NAME ), HTTPAccept( HTTP_ACCEPT );
21
22       // remove '/' at start of URI
23       reqURI.remove(0,1);
24       reqURI.append( PATH_INFO );
25
26       ...
```

# ANNEXE H

# Code : application de test de performance

## H.1 Header file

```
1  /*
2   - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
3    Filename : perftest.h
4    Project : Embedded Measurement Plateform
5    Part : Plateform performance test
6
7    Author : Johann Seydoux <johann dot seydoux at students dot hevs dot ch>
8    Version : 1.0
9    Created : 2009-06-30
10   Modified : 2009-06-30
11
12   Summary :
13   This class shows the time elapsed between the sending of a request and the
14   reading of the last response byte.
15  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
16 */
17
18 #ifndef PERFTEST_H
19 #define PERFTEST_H
20
21 #include <QObject>
22 #include <QTime>
23 #include <QtDebug>
24 #include <QHttp>
25
26 class PerfTest : public QObject
27 {
28    Q_OBJECT
29
30 public:
31    PerfTest( const QByteArray );
32
33 public slots:
34    void onRequestStarted();
35    void onRequestFinished();
36    void onRespopnseHeaderReceived(QHttpResponseHeader);
37
38 private:
39    QHttp *pReq;
40    QTime t;
41 };
42
43 #endif // PERFTEST_H
```

## H.2 Source file

```
1  /*
2   - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
3    Filename : perftest.cpp
4    Project : Embedded Measurement Plateform
5    Part : Plateform performance test
6
7    Author : Johann Seydoux <johann dot seydoux at students dot hevs dot ch>
8    Version : 1.0
9    Created : 2009-06-30
10   Modified : 2009-06-30
11
12   Summary :
13   This class shows the time elapsed between the sending of a request and the
14   reading of the last response byte.
15  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
16 */
17
18 #include "perftest.h"
19
20 PerfTest::PerfTest( const QByteArray path )
21 {
22    pReq = new QHttp;
23
24    qDebug() << path;
25
26    // create the HTTP request
27    QHttpRequestHeader *header = new QHttpRequestHeader( "GET" , path );
28    header->setValue( "Host" , "localhost" );
29    header->setValue( "Accept" , "application/rdf+xml" );
30
31    qDebug() << "Requested URI : " << header->path();
32
33    // set the host and send the request
34    pReq->setHost( "localhost" );
35    pReq->request( *header );
36
37    connect( pReq , SIGNAL( requestStarted( int ) ) , this , SLOT( onRequestStarted( void ) ) );
38    connect( pReq , SIGNAL( responseHeaderReceived(QHttpResponseHeader) ) , this , SLOT(
          onRespopnseHeaderReceived(QHttpResponseHeader) ) );
39    connect( pReq , SIGNAL( requestFinished( int , bool ) ) , this , SLOT( onRequestFinished( void
          ) ) );
40 }
41
42 void PerfTest::onRequestStarted()
43 {
44    // start time count
45    t.start();
46 }
47
48 void PerfTest::onRequestFinished()
49 {
50    // read all the received bytes and show the time elapsed
51    // since the request has been sent
52    QByteArray response = pReq->readAll();
53    qDebug() << "Time elapsed : " << t.elapsed() << " ms";
54 }
55
56 void PerfTest::onRespopnseHeaderReceived( QHttpResponseHeader resHeader )
57 {
58    qDebug() << "Status code : " << resHeader.statusCode();
59 }
```

## H.3 Main

```
1  /*
2   - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
3    Filename : main.cpp
4    Project : Embedded Measurement Plateform
5    Part : Plateform performance test
6
7    Author : Johann Seydoux <johann dot seydoux at students dot hevs dot ch>
8    Version : 1.0
9    Created : 2009-06-30
10   Modified : 2009-06-30
11
12   Summary :
13   The main function which use the performance test class.
14  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
15 */
16
17 #include <QtCore/QCoreApplication>
18
19 #include "perftest.h"
20
21 int main(int argc, char *argv[])
22 {
23    QCoreApplication a(argc, argv);
24    PerfTest perftest;
25
26    if ( argc == 2 )
27    {
28       perftest( QByteArray::QByteArray( argv[1] ) );
29    }
30
31    return a.exec();
32 }
```

ANNEXE I

# Code : application de simulation d'acquisition

## I.1 Header file

```
1  /*
2   - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
3    Filename : Randaemon.cpp
4    Project : Embedded Measurement Plateform
5    Part : Acquisition
6
7    Author : Johann Seydoux <johann dot seydoux at students dot hevs dot ch>
8    Version : 1.0
9    Created : 2009-06-25
10   Modified : 2009-06-26
11
12   Summary :
13   This class was made for simulation. It returns a random number which plays the
14   part of a sensor to the acquisition class.
15  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
16 */
17
18 #ifndef RANDAEMON_H
19   #define RANDAEMON_H
20
21 #include <time.h>
22 #include "polldaemon.h"
23
24 class RanDaemon : public PollDaemon
25 {
26 public:
27    RanDaemon( const char * );
28    double readSensor();
29 };
30
31 #endif
```

## I.2 Source file

```
1  /*
2   - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
3    Filename : Randaemon.cpp
4    Project : Embedded Measurement Plateform
5    Part : Acquisition
6
7    Author : Johann Seydoux <johann dot seydoux at students dot hevs dot ch>
8    Version : 1.0
9    Created : 2009-06-25
10   Modified : 2009-06-26
11
12   Summary :
13   This is the implementation of the Randaemon class. This class was made for
14   simulation. It returns a random number which plays the part of a sensor to
15   the acquisition class.
16  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
17  */
18
19  #include "Randaemon.h"
20
21  /**
22   * Function :  constructor
23   * Parameters : pDaemonAddr pointer on daemon address
24   * Summary :
25   * Initialize daemon all parameters that allows communication
26   * between daemon and sensor
27   **/
28  RanDaemon::RanDaemon( const char * pDaemonAddr ) : PollDaemon( pDaemonAddr )
29  {
30     // daemon initialization (drivers, ...)
31  }
32
33  /**
34   * Function :  readSensor
35   * Parameters : none
36   * Summary :
37   * Read the sensor value and return it
38   **/
39  double RanDaemon::readSensor()
40  {
41     return (double)rand() / (double)RAND_MAX * 60 - 20;
42  }
```

## I.3 Main

```
/*
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
  Filename : main.cpp
  Project : Embedded Measurement Plateform
  Part : Acquisition

  Author : Johann Seydoux <johann dot seydoux at students dot hevs dot ch>
  Version : 1.0
  Created : 2009-06-25
  Modified : 2009-06-26

  Summary :
  This function simulates the creation of acquisition daemons.
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
*/

#include "Randaemon.h"

#include <QCoreApplication>
#include <QtDebug>

int main( int argc , char ** argv )
{
   QCoreApplication app( argc , argv );

   // acquisition processus creation
   char daemonName[] = "daemon1";
   RanDaemon d1( daemonName );
   strcpy( daemonName , "daemon2" );
   RanDaemon d2( daemonName );
   strcpy( daemonName , "daemon3" );
   RanDaemon d3( daemonName );
   strcpy( daemonName , "daemon4" );
   RanDaemon d4( daemonName );
   strcpy( daemonName , "daemon5" );
   RanDaemon d5( daemonName );
   strcpy( daemonName , "daemon6" );
   RanDaemon d6( daemonName );

   return app.exec();
}
```

# ANNEXE J

## Documents RDF retournés

Document retourné par l'URI `http://localhost/`

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <!DOCTYPE rdf:RDF [<!ENTITY dp "embedded-station">
3  <!ENTITY dploc "embedded-stationLocation"><!ENTITY host "http://localhost/"><!ENTITY m0 "
       IndoorTemperature"><!ENTITY m1 "IndoorHumidity"><!ENTITY m2 "Rainfall"><!ENTITY m3 "
       OutdoorTemperature"><!ENTITY m4 "OutdoorHumidity">]>
4  <rdf:RDF xmlns="http://sten-04.hevs.ch/emep.rdf#" xmlns:sten="http://www.s-ten.eu/sten-core#"
       xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" xmlns:geo="http://www.w3.org/2003/01/
       geo/wgs84_pos#" xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#" >
5        <rdf:Description rdf:about="#&dp;">
6            <rdfs:label>&dp;</rdfs:label>
7            <rdfs:isDefinedBy rdf:resource="&host;html/"/>
8            <sten:hasPart rdf:resource="#&m0;Sensor"/>
9            <sten:hasPart rdf:resource="#&m1;Sensor"/>
10           <sten:hasPart rdf:resource="#&m2;Sensor"/>
11           <sten:hasPart rdf:resource="#&m3;Sensor"/>
12           <sten:hasPart rdf:resource="#&m4;Sensor"/>
13           <sten:occupies rdf:resource="#&dploc;"/>
14       </rdf:Description>
15       <rdf:Description rdf:about="#&dploc;">
16           <geo:altitude rdf:datatype="http://www.w3.org/2001/XMLSchema#double">500.100</
                 geo:altitude>
17           <geo:latitude rdf:datatype="http://www.w3.org/2001/XMLSchema#double">46.240536</
                 geo:latitude>
18           <geo:longitude rdf:datatype="http://www.w3.org/2001/XMLSchema#double">7.359077</
                 geo:longitude>
19       </rdf:Description>
20       <AirTemperature rdf:about="#&m0;">
21           <sten:at rdf:resource="#dploc;"/>
22           <inside rdf:resource="#&dploc;"/>
23       </AirTemperature>
24       <Humidity rdf:about="#&m1;">
25           <sten:at rdf:resource="#dploc;"/>
26           <inside rdf:resource="#&dploc;"/>
27       </Humidity>
28       <Rainfall rdf:about="#&m2;">
29           <sten:at rdf:resource="#dploc;"/>
30           <outside rdf:resource="#&dploc;"/>
31       </Rainfall>
32       <AirTemperature rdf:about="#&m3;">
33           <sten:at rdf:resource="#dploc;"/>
34           <outside rdf:resource="#&dploc;"/>
35       </AirTemperature>
36       <Humidity rdf:about="#&m4;">
37           <sten:at rdf:resource="#dploc;"/>
38           <outside rdf:resource="#&dploc;"/>
39       </Humidity>
40       <sten:ObservingDevice rdf:about="#&m0;Sensor">
41           <sten:performsObservation rdf:resource="&host;&m0;#&m0;MeasureActivity"/>
42           <sten:performs rdf:resource="&host;&m0;#&m0;MinimizingActivity"/>
43           <sten:performs rdf:resource="&host;&m0;#&m0;MaximizingActivity"/>
44           <sten:performs rdf:resource="&host;&m0;#&m0;AveragingActivity"/>
45           <sten:observes rdf:resource="#&m0;"/>
46       </sten:ObservingDevice>
47       <sten:ObservingDevice rdf:about="#&m1;Sensor">
48           <sten:performsObservation rdf:resource="&host;&m1;#&m1;MeasureActivity"/>
49           <sten:observes rdf:resource="#&m1;"/>
50       </sten:ObservingDevice>
51       <sten:ObservingDevice rdf:about="#&m2;Sensor">
```

```
52          <sten:performsObservation rdf:resource="&host;&m2;#&m2;MeasureActivity"/>
53          <sten:observes rdf:resource="#&m2;"/>
54      </sten:ObservingDevice>
55      <sten:ObservingDevice rdf:about="#&m3;Sensor">
56          <sten:performsObservation rdf:resource="&host;&m3;#&m3;MeasureActivity"/>
57          <sten:performs rdf:resource="&host;&m3;#&m3;MinimizingActivity"/>
58          <sten:performs rdf:resource="&host;&m3;#&m3;MaximizingActivity"/>
59          <sten:performs rdf:resource="&host;&m3;#&m3;AveragingActivity"/>
60          <sten:observes rdf:resource="#&m3;"/>
61      </sten:ObservingDevice>
62      <sten:ObservingDevice rdf:about="#&m4;Sensor">
63          <sten:performsObservation rdf:resource="&host;&m4;#&m4;MeasureActivity"/>
64          <sten:observes rdf:resource="#&m4;"/>
65      </sten:ObservingDevice>
66  </rdf:RDF>
```

Document retourné par l'URI `http://localhost/IndoorTemperature`

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE rdf:RDF [<!ENTITY host "http://localhost/">
<!ENTITY m "OutdoorTemperature"><!ENTITY emep "http://sten-04.hevs.ch/emep.rdf"><!ENTITY time "
    http://registry.emep.hevs.ch/time/">]>
<rdf:RDF xmlns="http://sten-04.hevs.ch/emep.rdf#" xmlns:sten="http://www.s-ten.eu/sten-core#"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" xmlns:geo="http://www.w3.org/2003/01/
    geo/wgs84_pos#" xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#" >
    <rdf:Description rdf:about="&m;MeasureActivity">
        <rdf:type rdf:resource="http://www.s-ten.eu/sten-core#ObservingActivity"/>
        <rdf:type rdf:resource="http://www.s-ten.eu/sten-core#TransmittingActivity"/>
        <sten:giveResultRecord>
            <sten:ResultRecord>
                <sten:hasTemporalPart rdf:resource="&host;&m;/2009#this"/>
                <sten:hasPart rdf:resource="&host;&m;/2009.values"/>
            </sten:ResultRecord>
        </sten:giveResultRecord>
        <sten:observationPerformedByDevice rdf:resource="#&m;Sensor"/>
        <sten:observes rdf:resource="#&m;"/>
        <rdfs:isDefinedBy rdf:resource="&host;html/&m;"/>
    </rdf:Description>
    <rdf:Description rdf:about="#&m;AveragingActivity">
        <rdf:type rdf:resource="&emep;Averaging#Activity"/>
        <sten:giveResultRecord>
            <sten:ResultRecord>
                <sten:hasTemporalPart rdf:resource="&host;&m;/2009#stat"/>
                <sten:hasTemporalPart rdf:resource="&host;&m;/2009.stat"/>
            </sten:ResultRecord>
        </sten:giveResultRecord>
        <sten:observationPerformedByDevice rdf:resource="#&m;Sensor"/>
        <sten:observes rdf:resource="#&m;"/>
    </rdf:Description>
    <rdf:Description rdf:about="#&m;MaximizingActivity">
        <rdf:type rdf:resource="&emep;Maximizing#Activity"/>
        <sten:giveResultRecord>
            <sten:ResultRecord>
                <sten:hasTemporalPart rdf:resource="&host;&m;/2009#stat"/>
                <sten:hasTemporalPart rdf:resource="&host;&m;/2009.stat"/>
            </sten:ResultRecord>
        </sten:giveResultRecord>
        <sten:observationPerformedByDevice rdf:resource="#&m;Sensor"/>
        <sten:observes rdf:resource="#&m;"/>
    </rdf:Description>
    <rdf:Description rdf:about="#&m;MinimizingActivity">
        <rdf:type rdf:resource="&emep;Minimizing#Activity"/>
        <sten:giveResultRecord>
            <sten:ResultRecord>
                <sten:hasTemporalPart rdf:resource="&host;&m;/2009#stat"/>
                <sten:hasTemporalPart rdf:resource="&host;&m;/2009.stat"/>
            </sten:ResultRecord>
        </sten:giveResultRecord>
        <sten:observationPerformedByDevice rdf:resource="#&m;Sensor"/>
        <sten:observes rdf:resource="#&m;"/>
    </rdf:Description>
    <rdf:Description rdf:about="&host;&m;/2009#this">
        <rdf:type rdf:resource="http://www.s-ten.eu/sten-core#ResultRecord"/>
        <rdfs:label>&m; For Year 2009</rdfs:label>
        <sten:during rdf:resource="&time;2009#Year-2009"/>
    </rdf:Description>
    <rdf:Description rdf:about="&host;&m;/2009#stat">
        <rdf:type rdf:resource="http://www.s-ten.eu/sten-core#ResultRecord"/>
        <rdfs:label>&m; For Year 2009</rdfs:label>
        <sten:during rdf:resource="&time;2009#Year-2009"/>
    </rdf:Description>
</rdf:RDF>
```

Document retourné par l'URI `http://localhost/IndoorTemperature/2009`

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE rdf:RDF [<!ENTITY host "http://localhost/">
<!ENTITY rUrl "http://localhost/OutdoorTemperature/2009"><!ENTITY m "OutdoorTemperature"><!ENTITY
     time "http://registry.emep.hevs.ch/time/">]>
<rdf:RDF xmlns="http://sten-04.hevs.ch/emep.rdf#" xmlns:sten="http://www.s-ten.eu/sten-core#"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" xmlns:geo="http://www.w3.org/2003/01/
    geo/wgs84_pos#" xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#" >
    <sten:ResultRecord rdf:about="&rUrl;#this">
        <rdfs:label>&m; For Year 2009</rdfs:label>
        <sten:during rdf:resource="&time;2009#Year-2009"/>
        <sten:hasTemporalPart rdf:resource="&host;&m;/2009-06#this"/>
        <sten:hasPart rdf:resource="&host;&m;/2009-06.values"/>
        <sten:hasTemporalPart rdf:resource="&host;&m;/2009-07#this"/>
        <sten:hasPart rdf:resource="&host;&m;/2009-07.values"/>
        <rdfs:isDefinedBy rdf:resource="&host;html/2009"/>
    </sten:ResultRecord>
    <sten:ResultRecord rdf:about="&rUrl;#stat">
        <rdfs:label>&m; For Year 2009</rdfs:label>
        <sten:during rdf:resource="&time;2009#Year-2009"/>
        <sten:hasPart rdf:resource="&rUrl;.stat"/>
        <sten:hasTemporalPart rdf:resource="&host;&m;/2009-06#stat"/>
        <sten:hasPart rdf:resource="&host;&m;/2009-06.stat"/>
        <sten:hasTemporalPart rdf:resource="&host;&m;/2009-07#stat"/>
        <sten:hasPart rdf:resource="&host;&m;/2009-07.stat"/>
        <rdfs:isDefinedBy rdf:resource="&host;html/2009"/>
    </sten:ResultRecord>
    <rdf:Description rdf:about="&host;&m;/2009-06#this">
        <rdfs:label>&m; For Month 2009-06</rdfs:label>
        <sten:during rdf:resource="&time;2009-06#Month-2009-06"/>
    </rdf:Description>
    <rdf:Description rdf:about="&host;&m;/2009-06#stat">
        <rdfs:label>&m; For Month 2009-06</rdfs:label>
        <sten:during rdf:resource="&time;2009-06#Month-2009-06"/>
    </rdf:Description>
    <rdf:Description rdf:about="&host;&m;/2009-07#this">
        <rdfs:label>&m; For Month 2009-07</rdfs:label>
        <sten:during rdf:resource="&time;2009-07#Month-2009-07"/>
    </rdf:Description>
    <rdf:Description rdf:about="&host;&m;/2009-07#stat">
        <rdfs:label>&m; For Month 2009-07</rdfs:label>
        <sten:during rdf:resource="&time;2009-07#Month-2009-07"/>
    </rdf:Description>
</rdf:RDF>
```

Document retourné par l'URI `http://localhost/IndoorTemperature/2009-07-01T00.values`

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE rdf:RDF [<!ENTITY host "http://localhost/">
<!ENTITY m "OutdoorTemperature"><!ENTITY mrec "&m;RecordFor-"><!ENTITY time "http://registry.emep
    .hevs.ch/time/">]>
<rdf:RDF xmlns="http://sten-04.hevs.ch/emep.rdf#" xmlns:sten="http://www.s-ten.eu/sten-core#"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" xmlns:geo="http://www.w3.org/2003/01/
    geo/wgs84_pos#" xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#" >
    <sten:ResultGraph rdf:about="#&mrec;2009-07-01T00-02">
        <sten:Celsius rdf:parseType="Resource">
            <sten:decimal>5.99</sten:decimal>
        </sten:Celsius>
        <sten:atTime rdf:resource="&time;2009-07-01T00-02#2009-07-01T00-02"/>
        <sten:resultGraphGivenBy rdf:resource="&host;&m;#&m;MeasureActivity"/>
    </sten:ResultGraph>
    <sten:ResultGraph rdf:about="#&mrec;2009-07-01T00-12">
        <sten:Celsius rdf:parseType="Resource">
            <sten:decimal>20.43</sten:decimal>
        </sten:Celsius>
        <sten:atTime rdf:resource="&time;2009-07-01T00-12#2009-07-01T00-12"/>
        <sten:resultGraphGivenBy rdf:resource="&host;&m;#&m;MeasureActivity"/>
    </sten:ResultGraph>
    <sten:ResultGraph rdf:about="#&mrec;2009-07-01T00-22">
        <sten:Celsius rdf:parseType="Resource">
            <sten:decimal>24.53</sten:decimal>
        </sten:Celsius>
        <sten:atTime rdf:resource="&time;2009-07-01T00-22#2009-07-01T00-22"/>
        <sten:resultGraphGivenBy rdf:resource="&host;&m;#&m;MeasureActivity"/>
    </sten:ResultGraph>
    <sten:ResultGraph rdf:about="#&mrec;2009-07-01T00-32">
        <sten:Celsius rdf:parseType="Resource">
            <sten:decimal>30.49</sten:decimal>
        </sten:Celsius>
        <sten:atTime rdf:resource="&time;2009-07-01T00-32#2009-07-01T00-32"/>
        <sten:resultGraphGivenBy rdf:resource="&host;&m;#&m;MeasureActivity"/>
    </sten:ResultGraph>
    <sten:ResultGraph rdf:about="#&mrec;2009-07-01T00-42">
        <sten:Celsius rdf:parseType="Resource">
            <sten:decimal>-14.96</sten:decimal>
        </sten:Celsius>
        <sten:atTime rdf:resource="&time;2009-07-01T00-42#2009-07-01T00-42"/>
        <sten:resultGraphGivenBy rdf:resource="&host;&m;#&m;MeasureActivity"/>
    </sten:ResultGraph>
    <sten:ResultGraph rdf:about="#&mrec;2009-07-01T00-52">
        <sten:Celsius rdf:parseType="Resource">
            <sten:decimal>10.84</sten:decimal>
        </sten:Celsius>
        <sten:atTime rdf:resource="&time;2009-07-01T00-52#2009-07-01T00-52"/>
        <sten:resultGraphGivenBy rdf:resource="&host;&m;#&m;MeasureActivity"/>
    </sten:ResultGraph>
    <rdf:Description rdf:about="">
        <rdfs:isDefinedBy rdf:resource="&host;html/&m;/2009-07-01T00.values"/>
    </rdf:Description>
</rdf:RDF>
```

Document retourné par l'URI `http://localhost/IndoorTemperature/2009-07-01T00.stat`

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <!DOCTYPE rdf:RDF [<!ENTITY host "http://localhost/">
3  <!ENTITY m "OutdoorTemperature"><!ENTITY mrec "&m;RecordFor-"><!ENTITY time "http://registry.emep
       .hevs.ch/time/">]>
4  <rdf:RDF xmlns="http://sten-04.hevs.ch/emep.rdf#" xmlns:sten="http://www.s-ten.eu/sten-core#"
       xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" xmlns:geo="http://www.w3.org/2003/01/
       geo/wgs84_pos#" xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#" >
5       <sten:ResultGraph rdf:about="&host;&m;/2009-07-01T00#max">
6            <sten:during rdf:resource="&time;2009-07-01T00#Hour-2009-07-01T00"/>
7            <sten:Celsius rdf:parseType="Resource">
8                 <sten:decimal>30.492100</sten:decimal>
9            </sten:Celsius>
10           <sten:resultGraphGivenBy rdf:resource="&host;&m;#&m;MaximizingActivity"/>
11      </sten:ResultGraph>
12      <sten:ResultGraph rdf:about="&host;&m;/2009-07-01T00#min">
13           <sten:during rdf:resource="&time;2009-07-01T00#Hour-2009-07-01T00"/>
14           <sten:Celsius rdf:parseType="Resource">
15                <sten:decimal>-14.958200</sten:decimal>
16           </sten:Celsius>
17           <sten:resultGraphGivenBy rdf:resource="&host;&m;#&m;MinimizingActivity"/>
18      </sten:ResultGraph>
19      <sten:ResultGraph rdf:about="&host;&m;/2009-07-01T00#avg">
20           <sten:during rdf:resource="&time;2009-07-01T00#Hour-2009-07-01T00"/>
21           <sten:Celsius rdf:parseType="Resource">
22                <sten:decimal>12.887803</sten:decimal>
23           </sten:Celsius>
24           <sten:resultGraphGivenBy rdf:resource="&host;&m;#&m;AveragingActivity"/>
25      </sten:ResultGraph>
26      <rdf:Description rdf:about="">
27           <rdfs:isDefinedBy rdf:resource="&host;html/&m;/2009-07-01T00.stat"/>
28      </rdf:Description>
29 </rdf:RDF>
```