Table des matières

Introduction

Ce rapport de stage décrit un travail dont le but était de proposer le design d'une station météo autonome. Dans l'optique d'une utilisation large, le design proposé dans ce travail est suffisamment ouvert pour être librement adapté à d'autres utilisations.
Le rapport est composé de 3 sections, la première reprend les exigences et besoins auxquels doit répondre la station météo. La seconde détaille la solution proposée dans le cadre de ce travail. Enfin, la dernière section propose des pistes afin d'étendre les fonctionnalités du design proposé.

1 Cahier des charges

Ce travail portait sur l'élaboration d'un design ouvert d'une station météo autonome. Seuls les aspects théoriques sont donc considérés dans ce travail.
Cette section présente le cahier des charges à travers trois dimensions que sont les fonctionnalités, les performances et les contraintes.

1.1 Fonctionnalités

— La station doit pouvoir prélever la température et l'humidité à intervalle régulier.
— Elle doit pouvoir sauvegarder ces mesures et les recharger plus tard.
— Il doit être possible de générer des graphes retraçant l'évolution des mesures.
— Le design doit inclure une interface utilisateur basique.
— Des indicateurs statistiques sur les données collectées doivent être affichés par la station.
— La station doit être capable d'interagir avec un smartphone.
— Des estimations sur les conditions météo à court terme doivent être disponibles.

1.2 Performances

— La précision des estimations et des indicateurs statistiques doit être aussi grande que possible.

1.3 Contraintes

— La station doit pouvoir être implémentée sur un Raspberry Pi [1].
— Elle doit pouvoir fonctionner en permanence 24h/24, 7j/7.

1. https://www.raspberrypi.org/help/what-is-a-raspberry-pi/

2 Spécifications de la SMART

Cette section décrit les spécifications du design « SMART », pour Station Météorologique Autonome Reliée et Temps réel, qui est celui proposé dans le cadre de ce travail. La section présente les points qui permettent de remplir les exigences du cahier des charges énoncé plus haut.

2.1 Collecte des données

La SMART collecte et enregistre des données à intervalle régulier. Après avoir fait une mesure de température et d'humidité, elle stocke ces informations dans un fichier texte, ce qui permet une certaine historicité et un traitement ultérieur des données.
Le fichier généré suit un format strict. Chaque ligne de celui-ci contient une donnée, dont voici un exemple.

```
2015-08-06T14:59:31+02:00  0  0  24.2  3
```

Une donnée est composée de 4 à 5 valeurs. La première représente le temps à la seconde près lors de la mesure, ce qui permet une grande quantité de détails dans les relevés. La seconde est un paramètre qui détermine quelle est la nature de la donnée, à savoir un relevé de température ou d'humidité. Le tableau 1 spécifie les valeurs de ce paramètre.

Paramètre	Nature de la donnée
0	Température
1	Humidité

Table 1 – Nature des données

La troisième valeur est présente si la donnée représente un relevé de température. C'est un paramètre qui détermine la nature du relevé de température, à savoir un relevé exprimé en degrés Celsius ou en degrés Fahrenheit. Le tableau 2 spécifie les valeurs de ce paramètre.

Paramètre	Échelle de température
0	Celsius
1	Fahrenheit

Table 2 – Échelle de la température

La quatrième valeur est un nombre à virgule flottante qui représente, en fonction des paramètres précédents, soit une température en degrés Celsius ou Fahrenheit, soit un pourcentage d'humidité. Enfin, la dernière valeur assure l'intégrité de la donnée. Cette valeur est égale au reste de la division euclidienne de la somme des chiffres composant la ligne par 13.
Dans notre exemple, $2+0+1+5+0+8+0+6+1+4+5+9+3+1+0+2+0+0+0+0+2+4+2 = 55$, et $55 \bmod 13 = 3$. La dernière valeur sera ici égale a 3.
L'exemple plus haut représente donc un relevé de température de 24,2 °C le jeudi 06 août 2015 à 14h59m31s. La SMART prélève simultanément la température et l'humidité.

Le fichier peut commencer par entête constituée d'une série de lignes préfixées d'un hashtag (#). Ces lignes sont des lignes de configuration. Elles n'ont aucune utilité immédiate mais pourront être utilisé dans le futur afin de spécifier par exemple une autre valeur pour la division, ou contenir des informations sur la compression éventuelle du fichier. Toute information utile en rapport avec les données pourra donc être inscrite dans les en-têtes.
La SMART prévoit aussi un accès direct aux senseurs, afin de rester ouverte à des utilisations diverses.

2.2 Interface utilisateur

Une interface utilisateur est un autre moyen de visualiser rapidement des informations cruciales. La SMART agence l'interface en une série de « Frame ». Afin de rester compatible avec une interface en ligne de commande réduite, chaque Frame est composée exclusivement de texte. L'interface affiche une Frame à la fois. Il est possible de se déplacer de Frame en Frame grâce aux flèches du clavier comme indiqué dans la figure 1.

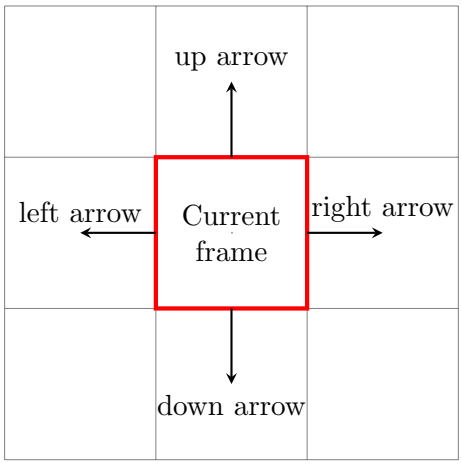

FIGURE 1 – Système de Frames et déplacement

Il n'y a pas de restriction quant au nombre de Frames ni même quant à leur disposition pour que l'interface puisse les afficher de façon cohérente. Les Frames seront utilisées pour afficher les divers indicateurs statistiques tels que la moyenne, la variance ou encore les minima et maxima calculés par la station.

2.3 Interaction avec les smartphones

Dans un monde de plus en plus mobile où la plupart des gens sont en possession d'un smartphone, l'accès à Internet est quasi permanent, permettant à tout instant et en tout endroit de recueillir et d'interagir avec de l'information en tout genre.
La SMART va dans ce sens en proposant une interaction avec les smartphones simple mais adaptable. Elle intègre Pushbullet [2] pour fournir des notifications en temps-réel sur tout type d'appareil. Il est possible d'envoyer du texte, mais aussi des fichiers images grâce à ce service. Les possibilités créées par cet ajout sont nombreuses.

2.3.1 Alertes temps-réel

Avec l'intégration de Pushbullet, il est simple d'ajouter des alertes prévenant qu'un indicateur passe en dessous ou dépasse un seuil fixé. On peut imaginer une alerte de température maximale dans une salle de serveur, ou un degré d'humidité inadéquat à la culture de végétaux.

2.3.2 Reporting

Toujours grâce à Pushbullet, il est possible d'imaginer un système de reporting à intervalle régulier, émettant un rapport de la journée, ou de la semaine précédente. Le contenu du rapport peut être librement adapté à l'utilisation finale de la station.

2.3.3 Langage de requête

Enfin, Pushbullet pourrait servir de canal de communication entre l'utilisateur et la station. Un langage de requête pourrait alors être implémenté pour récupérer des informations ou encore paramétrer les alertes et le reporting de façon interactive.

2.4 Graphique

La SMART permet de réaliser des séries temporelles, afin de pouvoir visualiser simplement l'évolution de la température ou de l'humidité. Une série temporelle est une suite de points qui associent chacun une valeur à un moment dans le temps. Le temps se trouve sur l'axe des x et les valeurs sur l'axe des y. Il est possible de superposer plusieurs types de données dans une même série temporelle, et ainsi visualiser par exemple l'évolution de la température et de sa moyenne glissante [3] sur 1 heure au sein d'un même graphe.
Les graphiques ne seront pas affichés via les Frames, toujours dans l'optique d'une compatibilité interface en ligne de commande, mais sous forme de notifications grâce à Pushbullet.

2. https://www.pushbullet.com/
3. https://fr.wikipedia.org/wiki/Moyenne_glissante

2.5 Régression linéaire et polynomiale

La méthode de prédiction statistique qui a été retenue dans le design de la SMART est la régression. Elle permet d'approximer la tendance d'une série de points et dégager une courbe pouvant alors être évaluée hors des limites de ces points. Grâce à cette méthode, il est donc possible d'estimer la température ou l'humidité sur des petites périodes de temps. La figure 2 montre un exemple d'une application d'une régression linéaire et d'une régression polynomiale [4] sur une portion du graphe de la température.

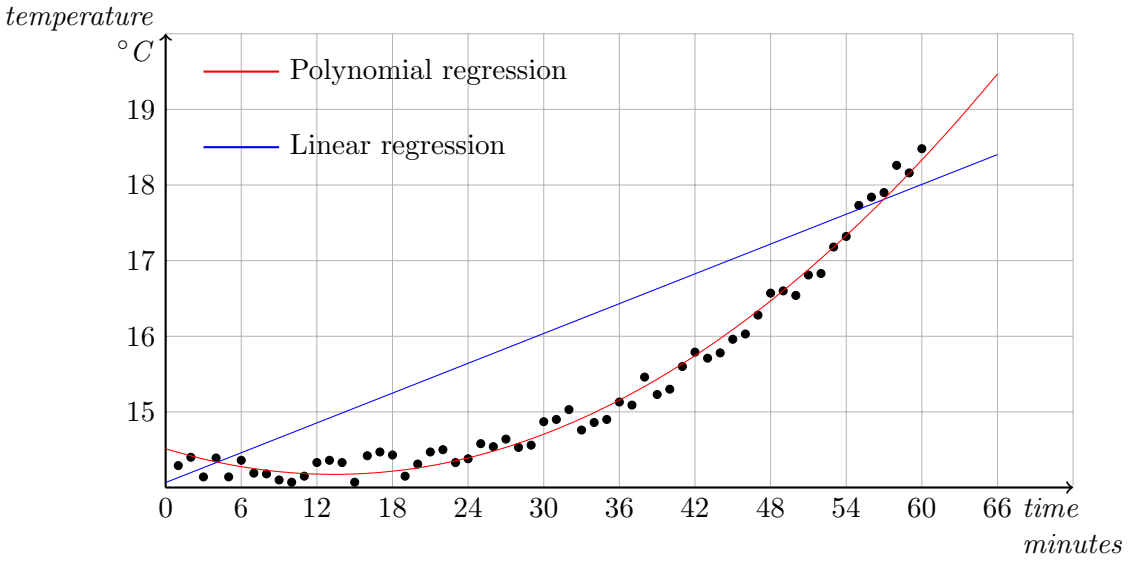

FIGURE 2 – Exemple de régression

La régression linéaire est une méthode moins complexe, qui peut donner une prédiction plus éloignée de la réalité. Dans l'exemple ci-dessus, la régression polynomiale semble effectivement mieux approximer le nuage de points.
Les prédictions pourront être communiquées sous une forme textuelle ou via un graphe montrant la série de points et son approximation.

4. https://en.wikipedia.org/wiki/Polynomial_regression/

3 Extensions possibles

Cette dernière section répertorie quelques extensions ajoutables au design décrit plus haut. Elles présentent un réel intérêt mais n'ont pas été intégrées au design de la SMART afin de garder la simplicité d'une spécification basique.

3.1 Nettoyage de données

La qualité des données de la SMART et des indicateurs qu'elle calcule est affectée par la qualité de son senseur. Un algorithme de détection d'erreur des relevés pourrait gérer les cas où le senseur renvoie une valeur tout à fait erronée. Le mécanisme de détection d'erreurs dans l'enregistrement des données (Section 2.1) pourrait être aussi exploité pour remplacer les données erronées en approximant la donnée perdue sur base des données proches.

3.2 Compression

La liberté donnée par les entêtes des fichiers de données permet de compresser le fichier avec une méthode tel que le codage de Huffmann [5] et d'y insérer dans les entêtes les informations non-compressées permettant de décompresser le reste du fichier.

Conclusion

En conclusion de ce rapport, la solution proposée répond bien au cahier des charges, et reste ouverte et souple à toute utilisation. Néanmoins, ce design reste à être validé par une implémentation. Une librairie aidant le développement a été créer à cet effet et des interfaces sont détaillées dans les annexes.

5. https://fr.wikipedia.org/wiki/Codage_de_Huffman

A Interfaces pour l'implémentation

A.1 Sample

```java
/**
 * Interface représentant une prise de mesure.
 */
public interface Sample {

    /**
     * @pre -
     * @post Retourne le temps.
     */
    public Instant getTime();

    /**
     * @pre -
     * @post Retourne la température.
     * @see #isCelsius()
     */
    public float getTemperature();

    /**
     * @pre -
     * @post Retourne le pourcentage d'humidité.
     */
    public float getHumidity();

    /**
     * @pre -
     * @post Retourne true si la température est exprimée en degrés Celsius,
     * false si elle est exprimée en Fahrenheit.
     */
    public boolean isCelsius();
}
```

A.2 SampleIO

```java
/**
 * Interface représentant un moyen de sauver et de récupérer des mesures.
 */
public interface SampleIO {

    /**
     * @pre sample != null
     * @post Le sample est sauvé et peut être récupérer de manière identique
     * suivant un appel adéquat à
     * {@link #readSample(java.time.Instant, java.time.Instant)}.
     */
    public void writeSample(Sample sample);

    /**
     * @pre from != null, from est l'instant à partir duquel il faut extraire
     * les données, to != null, to est l'instant à partir duquel il faut arrêter
     * d'extraire les données, from antécédent à to.
     * @post Retourne la liste de mesures prise entre les instants from et now,
     * null sinon
     */
    public List<Sample> readSample(Instant from, Instant to);

    /**
     * @pre -
     * @post Retourne la liste de toutes les mesures.
     */
    public List<Sample> readSample();
}
```

A.3 SampleFileIO

```java
/**
 * Interface représentant un moyen de sauver et de récupérer des mesures dans un
 * fichier.
 */
public interface SampleFileIO extends SampleIO {

    /**
     * @pre path != null, path est un chemin valide vers un fichier (qui peut ne
     * pas exister).
     * @post Les appels subséquents à {@link #writeSample(interfaces.Sample)} et
     * {@link #readSample(java.time.Instant, java.time.Instant)} s'effectuent
     * vers et à partir du fichier au chemin path.
     */
    public void open(String path);

    /**
     * @pre -
     * @post Toutes les ressources inhérentes à l'instance sont prêtes à être
     * libérées.
     */
    public void close();
}
```

A.4 Stats

```java
/**
 * Interface représentant un moyen de calculer des valeurs statistiques.
 */
public interface Stats {

    /**
     * @pre samples.size() &gt; 0, samples un liste de mesures. Toutes les mesures de températures sont exprimées dans la même unité.
     * @post Retourne la moyenne mathématique de la liste de mesures.
     */
    public Sample getAverage(List<Sample> samples);

    /**
     * @pre samples.size() &gt; 0, samples un liste de mesures. Toutes les mesures de températures sont exprimées dans la même unité.
     * @post Retourne le minimum de la liste de mesures.
     */
    public Sample getMin(List<Sample> samples);

    /**
     * @pre samples.size() &gt; 0, samples un liste de mesures. Toutes les mesures de températures sont exprimées dans la même unité.
     * @post Retourne le max de la liste de mesures.
     */
    public Sample getMax(List<Sample> samples);

    /**
     * @pre samples.size() &gt; 0, samples un liste de mesures. Toutes les mesures de températures sont exprimées dans la même unité.
     * @post Retourne la variance de la liste de mesures.
     */
    public Sample getVariance(List<Sample> samples);

    /**
     * @pre samples.size() &gt; 0, samples un liste de mesures. Toutes les mesures de températures sont exprimées dans la même unité.
     * @post Retourne l'écart-type de la liste de mesures.
     */
    public Sample getStandardDeviation(List<Sample> samples);
}
```

A.5 SampleListener

```java
/**
 *  Interface permettant de recevoir directement tous les samples pris par {@link SensorConnector}.
 */
public interface SampleListener {
    /**
     * Méthode appelée chaque fois que le {@link SensorConnector} prend une mesure.
     * @pre sample != null, sample est la mesure prise.
     */
    public void sampleTaken(Sample sample);
}
```

A.6 Frame

```java
/**
 *  Interface permettant d'afficher une Frame dans le terminal grâce à {@link
 FrameManager}.
 */
public interface Frame {

    /**
     *  Méthode appelée par {@link Display} pour afficher la Frame dans le terminal.
     *  @post Retourne un String représentant la chaine de caractère qui sera
     affichée dans le terminal.
     */
    public String getText();
}
```

A.7 FrameManager

```java
/**
 *  Interface permettant d'afficher un système de Frame dans le terminal grâce à {
 @link Display}.
 */
public interface FrameManager {

    /**
     * Méthode appelée par {@link Display} lorsque l'utilisateur utilise les flèches
      du clavier pour changer de Frame dans le terminal.
     * @pre movement est un entier issu des constantes définies dans Display
     représentant le movement ordonné par l'utilisateur
     * @post Retourne la Frame à afficher à l'issue du mouvement
     */
    public Frame getFrame(int movement);
}
```

A.8 PushListener

```java
/**
 *  Interface permettant de recevoir des notifications Pushbullet grâce à {@link
 PushbulletClient}.
 */
public interface PushListener {

    /**
     *  Méthode appelée par {@link PushbulletClient} lorsqu'une notification est
     reçue.
     *  @pre title est le titre de la notification reçue, body est le corps de la
     notification reçue.
     *  @post -
     */
    public void pushReceived(String title, String body);
}
```