

U.E. 4.4 : Rapport d'avancement, Jumeau Numérique de Chantier

FISE - 2024



ENSTA Bretagne
2 rue François Verny
29806 Brest Cedex 9, France

ALAMRI Mohammed, mohammed.alamri@ensta-bretagne.org
ALHOMIDY Abdulmalik, abdulmalik.alhomidy@ensta-bretagne.org
KENGNE TAGNE Ivan, ivan.kengne@ensta-bretagne.org
KOUYOUMDJIAN Elie, elie.kouyoumdjian@ensta-bretagne.org

Table des matières

Introduction	5
Résumé du semestre 3 – Présentation générale du projet	6
Cahier des Charges	7
Présentation des sprints du semestre 4	10
Figure 2 : Extrait du diagramme de Gantt du projet pour septembre 2022	12
Figure 3 : Extrait du diagramme de Gantt du projet pour janvier 2023	13
Partie JN	14
I. Identification des acteurs	14
II. Diagramme de contexte	15
Figure 4 : Diagramme de contexte, Système de gestion et de planification des projets	16
III. Diagramme de cas d'utilisation	16
Figure 5 : Diagramme de cas d'utilisation, Système de gestion et de planification des projets	17
Figure 6 : Diagramme de cas d'utilisation, système de gestion et de planification des projets	18
IV. Diagramme de classe	18
A. Partie gestion de chantier	18
Figure 7 : Diagramme de classe, Système de gestion et de planification des projets	19
B. Partie génération de diagramme BPMN	20
C. Interfaces mises en œuvre	22
V. Conclusion	24
Partie Capteurs & Transmission	25
I. Récapitulatif des choix du projet au niveau capteur	25
A) Choix effectués	25
B) Organisation finale	26
II. Acquisition et transmission des données terrain	27
A) Choix du capteur de son	27
B) Transmission à la passerelle	28
III. Mise en place de la passerelle	31
A) Fonction de la passerelle	31
B) Réception des données capteurs	32
C) Transmission au JN par client-serveur	34
IV. Utilisation du code (destiné aux prochaines années)	36
A) Explication	36
B) Démonstration	36

Bilan des réalisations du projet.....	38
Bilan Travail en Groupe.....	39
Références	40
Annexe : Récapitulatif des concepts d'UML 2.....	41

Introduction

Avec l'essor d'Internet et du Cloud ces dernières années, de plus en plus d'entreprises développent des jumeaux numériques en même temps que leurs produits ou services, dans le but d'optimiser les phases de vie de ces derniers. Ainsi, le projet ***Jumeau Numérique de Chantier*** a pour objectifs de concevoir le jumeau numérique d'un chantier fictif sur le campus de l'ENSTA Bretagne. Le système recevra des données du chantier via des capteurs, interprétera les données obtenues et en déduira le niveau d'avancement du chantier, pour aider le contremaître dans sa tâche de gestion.

Le rapport présente le travail réalisé sur le projet système de 2^{ème} année dans le cadre de l'UE 4. Il se concentre sur le résultat final du projet, et sur les objectifs atteints de janvier à avril.

Il est à mettre en rapprochement avec le rapport de mi-projet rendu en janvier 2023.

Résumé du semestre 3 – Présentation générale du projet

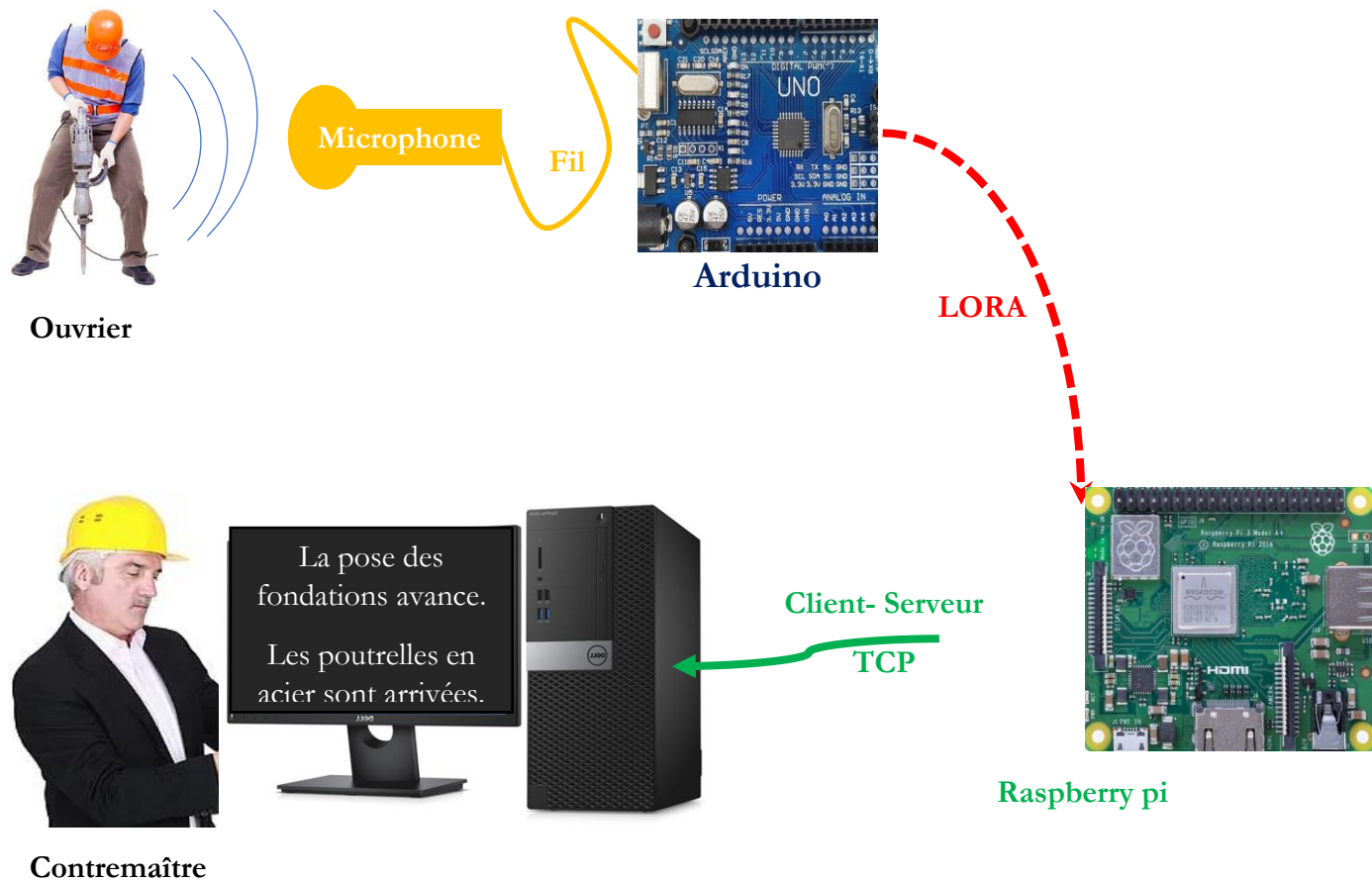


Figure 1 : Schéma de principe du système

Le premier semestre avait été l'occasion de poser les bases du projet (architecture fonctionnelle, tests), ainsi que de réaliser un travail de bibliographie sur le sujet du jumeau numérique.

L'objectif de ce semestre était de finaliser complètement le projet, de l'acquisition des données terrains à la construction du jumeau.

Dès le départ, le projet s'articule en deux grandes parties : Tout d'abord, l'acquisition, le traitement et le transfert des données terrains, les données du chantier jusqu'au JN.

Dans une seconde partie, la prise en compte de ces données terrain pour la construction du jumeau numérique, et la génération de diagramme BPMN pour montrer l'avancement du chantier.

Cahier des Charges

Pour donner suite au travail présenté à la fin du semestre 3 et la réunion de mi-parcours réalisée, la liste des exigences (à jour de la rédaction du présent rapport) est donnée dans le **Tableau 1**, et un diagramme pieuvre est donné en **Figure 1** (on y voit les fonctions techniques en bleu et la fonction principale en rouge, avec la nature du flux pour cette dernière).

On rappelle ici la liste des exigences du projet de Jumeau Numérique de chantier.

Exigence	Raison
Le système doit recueillir des données du chantier grâce à un capteur de son.	Le jumeau numérique reçoit des données du jumeau physique.
Le système doit pouvoir caractériser certains sons d'engins de chantier.	Permet de savoir quelle ressource est utilisée.
Les capteurs doivent faire parvenir l'identification qu'ils ont faite des sons perçus à une unité centrale via Internet.	Cette transmission sans fil permet de centraliser les données reçues vers un ordinateur qui mettra l'information en parallèle avec l'agenda.
Le système doit prendre en compte les ressources matérielles et humaines disponibles sur le chantier pour réaliser les tâches.	Ces ressources sont essentielles pour repérer les incohérences dans l'agenda.
Le système doit déduire, des données qu'il reçoit des capteurs, quelles tâches du chantier sont en cours de réalisation.	Permet de faire évoluer l'état d'avancement.
Le système doit détecter les incohérences entre les données reçues des capteurs et l'agenda du chantier.	Permet de savoir si une tâche est exécutée à un moment inopportun.
Le système doit repérer les retards pris dans l'avancement du chantier.	Aide le contremaître à replanifier son agenda en cas de retard.
Le système doit anticiper les incohérences entre les ressources disponibles sur le chantier et l'agenda prévu.	Aide le contremaître à modifier son agenda pour contourner le problème si possible.
Le système doit pouvoir afficher des diagrammes d'état du chantier suivant la norme BPMN.	La norme que suivent les diagrammes d'état du processus.
Le système doit pouvoir afficher le diagramme d'état du chantier lorsque le contremaître le lui demande.	Permet au contremaître de connaître l'état d'avancement de son chantier.
Le système doit sauvegarder l'état d'avancement du chantier au début de chaque jour.	Permet de constituer un historique des actions menées.
Le système doit savoir quand passer commande des matériaux requis à la réalisation d'une tâche.	Aide le contremaître à anticiper les besoins.

Tableau 1 : Liste des exigences du projet JN-Chantier.

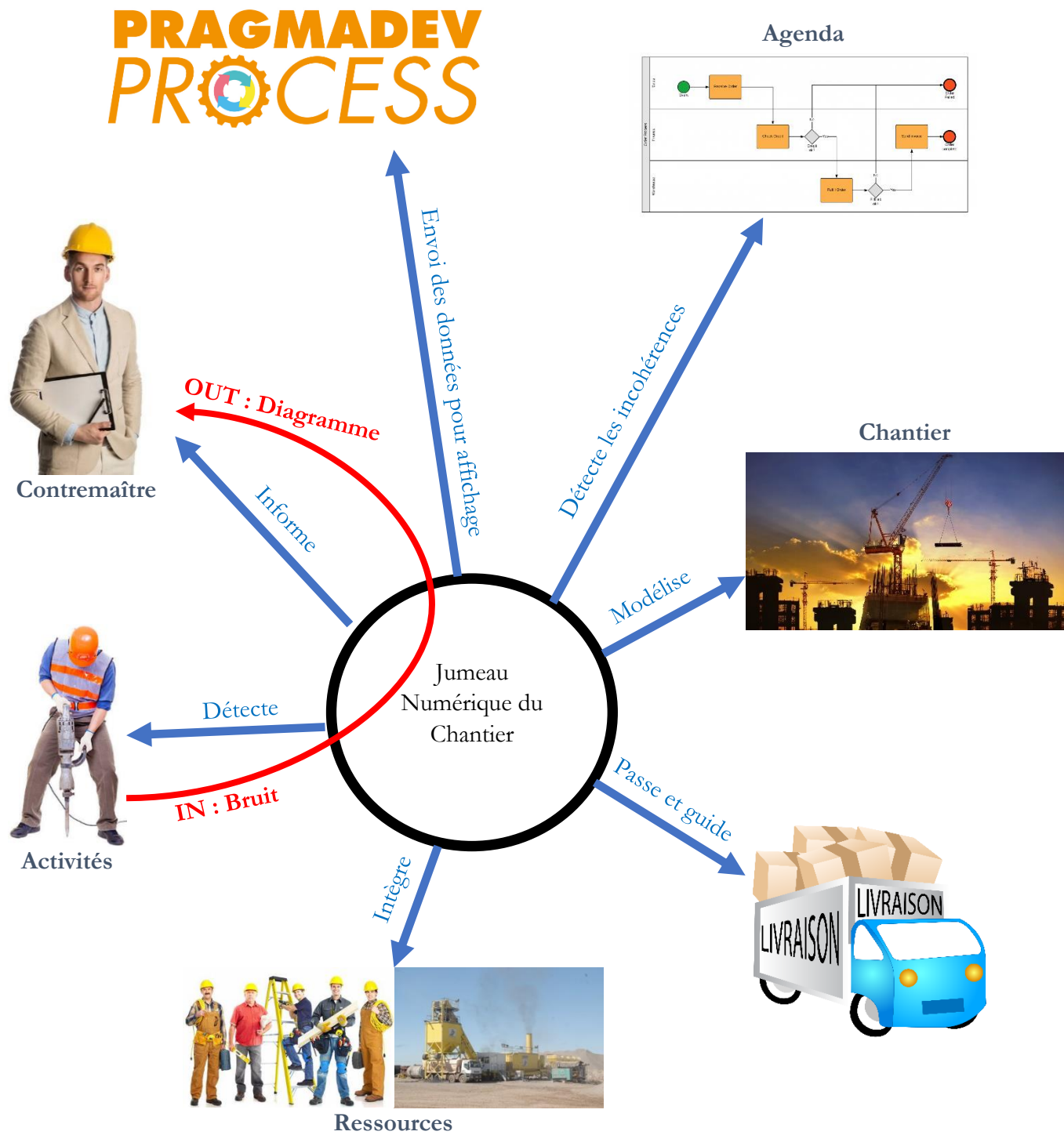


Figure 1: Diagramme pieuvre du système

Présentation des sprints du semestre 4

Numéro de sprint	Taches	Statut en fin de sprint
(9/01-16/01)	Etablir un plan de projet détaillé incluant les objectifs, les livrables	Fait(Tous)
(16/01-23/01)	Rédiger les spécifications fonctionnelles détaillées du système. Implémenter les fonctionnalités de base du système : la réception et le traitement des données envoyées par les capteurs.	Fait (Tous)
(23/01-30/01)	Avancer sur l'analyse du système et mettre à jour le diagramme de classe. Préparer les livrables pour la révision de sprint et les présenter à l'équipe. Valider l'envoi de données réelles des capteurs.	Fait (Ivan,Abdelmalik) Fait (Elie,Mohammed)
(30/01-6/02)	Rédiger une documentation détaillée des fonctionnalités développées et de leur utilisation. Implémenter le stockage de données capteurs sur la passerelle.	Fait (Ivan, Abdelmallik) Fait (Elie, Mohammed)
(6/02-13/02)	Développer l'interface utilisateur permettant d'afficher le diagramme d'état du chantier, selon les spécifications définies Créer une base de données pour stocker les données du chantier Implémenter l'envoi de la passerelle au serveur.	Fait (Ivan, Abdelmallik) Fait (Elie, Mohammed)
(20/02-27/02)	Implémenter la fonctionnalité permettant d'afficher les retards pris dans l'avancement du chantier. Mettre en place les tests pour vérifier le stockage de données sur la passerelle.	Fait (Ivan, Abdelmallik) Fait (Elie, Mohammed)
(27/02-6/03)	Développer l'interface utilisateur permettant d'afficher les incohérences entre les ressources disponibles sur le chantier et l'agenda prévu. Implémentation du pattern client-serveur.	Fait (Ivan, Abdelmallik) Fait (Elie, Mohammed)
(6/03-13/03)	Établir un plan de projet détaillé incluant les objectifs, les livrables attendus et les échéances pour le sprint 3. Implémentation du pattern client-serveur(suite)	Fait (Tous).
(13/03-20/03)	Mettre en place les premiers tests pour vérifier l'envoi capteur-passerelle-serveur. Rédaction du rapport de fin de projet.	Fait (Elie) Fait (Tous)

(20/03-27/03)	Mettre en place les premiers tests pour vérifier l'envoi capteur-passerelle-serveur. (Suite) Commencement du poster.	Fait (Elie, Mohammed). Fait (Tous)
(27/03 – 6/04)	Finalisation rapport projet pour le projet. Mise en place de la démonstration.	Fait(Tous)

Comme au semestre précédent, nous avons réalisé un diagramme de Gantt afin de planifier nos sprints dans le temps.

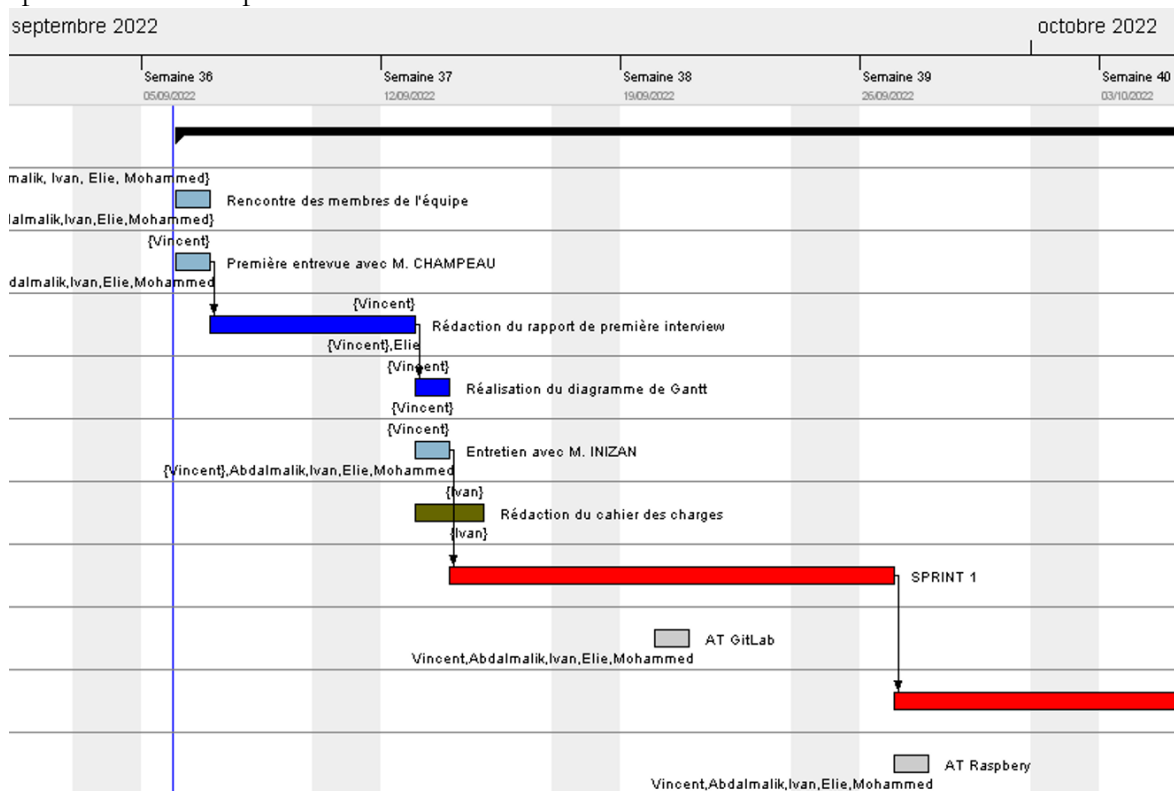


Figure 2 : Extrait du diagramme de Gantt du projet pour septembre 2022



Figure 3 : Extrait du diagramme de Gantt du projet pour janvier 2023

Partie JN

La partie nommée « Jumeau Numérique » est la partie finale de l'application, au sens où c'est elle qui centralise les données reçues des capteurs pour en déduire l'avancement du chantier, et qui fait le contact avec l'utilisateur (i.e. le gestionnaire du chantier). C'est elle qui, *in fine* permettra d'optimiser la résolution des failles d'une gestion manuelle, l'idée étant de traiter les opérations de la gestion d'un chantier de façon à minimiser les pertes de temps et à rendre les informations accessibles.

I. Identification des acteurs

Un acteur d'un système est une entité externe à ce système qui interagit avec lui. Les acteurs permettent de cerner l'interface que le système va offrir à son environnement. Chaque acteur du système est un profil pouvant factoriser plusieurs personnes ayant les mêmes droits. Nous avons capturé les acteurs qui vont interagir avec notre système, comme le montre ce tableau pour l'identification des acteurs. (Cf. Tableau 1).

Tableau 1 : Identification des acteurs

Nom de l'acteur	Rôle
Responsable de projet	<p>Le responsable de projet est une personne qui peut :</p> <ul style="list-style-type: none">• Gérer ses informations personnelles.• Ajouter, modifier ou supprimer une tâche.• Affecter des ressources humaines (intervenant) et matérielles à une tâche.• Planifier et visualiser dans un diagramme de Gantt les tâches du projet.• Visualiser l'état d'avancement du chantier dans un diagramme BPMN• Importer et exporter toutes les données de l'application

Intervenant	<p>L'intervenant est une personne qui peut :</p> <ul style="list-style-type: none"> • Gérer ses informations personnelles. • Afficher ses ordres de travail. • Modifier l'état d'une tâche. • Recevoir une notification (Email et SMS).
-------------	---

II. Diagramme de contexte

Le diagramme de contexte est une représentation UML, il modélise le système sous forme d'une boîte noire en interaction avec les acteurs du système. Ce diagramme vient avant les diagrammes de cas d'utilisation. L'étude conceptuelle nous a permis d'élaborer les diagrammes de contexte ci-dessous :

Ce diagramme de contexte est pour système de gestion et de planification des projets. Il contient deux acteurs, le responsable de projet et l'intervenant (cf. Figure 1).

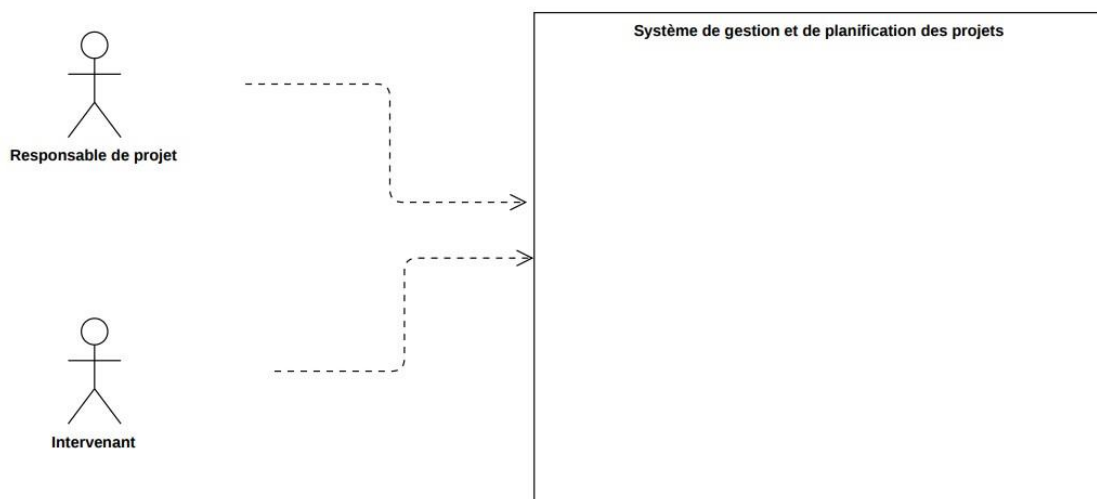


Figure 4 : Diagramme de contexte, Système de gestion et de planification des projets

III. Diagramme de cas d'utilisation

Les diagrammes de cas d'utilisation (DCU) sont des diagrammes UML utilisés pour une représentation du comportement fonctionnel d'un système logiciel. Ils sont utiles pour des présentations auprès de la direction ou des acteurs d'un projet, mais pour le développement, les cas d'utilisation sont plus appropriés.

L'étude de différentes fonctionnalités et l'analyse des actions que les acteurs peuvent exécuter nous a permis d'élaborer les diagrammes ci-dessous :

Ce Diagramme de cas d'utilisation pour système de gestion et de planification des projets, permet de voir les différentes fonctionnalités disponibles pour les différents acteurs, les cas d'utilisation montrés sur ces figures au-dessous (cf. Figures 1 et 2)

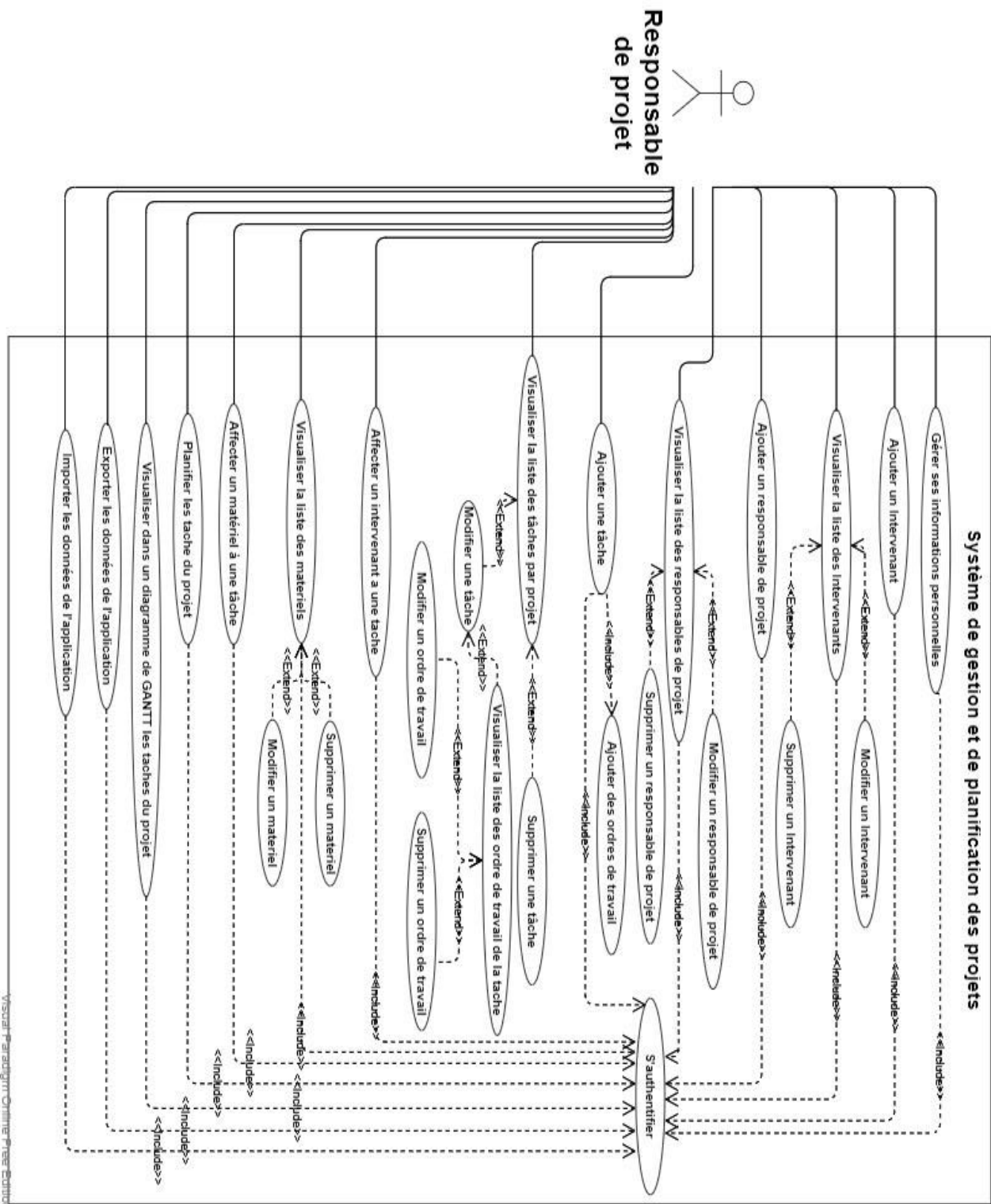


Figure 5 : Diagramme de cas d'utilisation, Système de gestion et de planification des projets

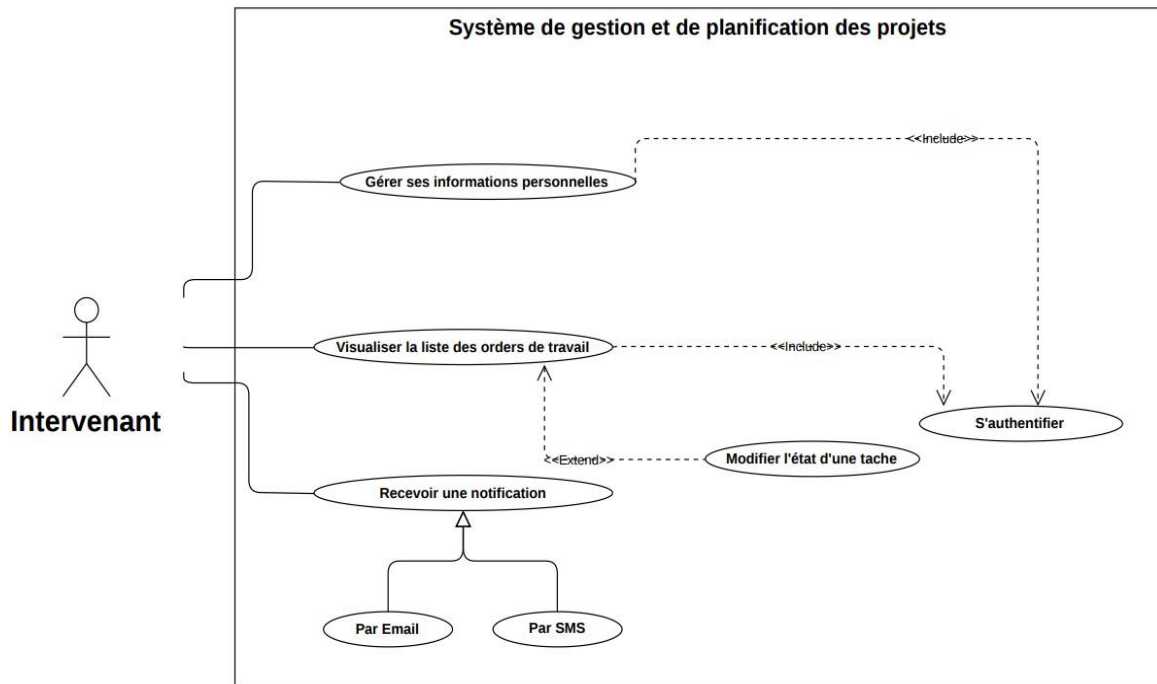


Figure 6 : Diagramme de cas d'utilisation, système de gestion et de planification des projets

IV. Diagramme de classe

A. Partie gestion de chantier

Le diagramme de classe constitue un élément important de la modélisation, puisqu'il permet de définir les composants du système final. Et avec le dressage des besoins du système, et les différents cas d'utilisation, la vision sur le système devient plus claire et une bonne conception du diagramme est possible. Le diagramme ci-dessous illustre la relation entre les différentes classes de l'application :

Ce diagramme de classes pour le système de gestion et de planification des projets permet de visualiser les différentes classes existantes et les relations entre elles (cf. Figure 4).

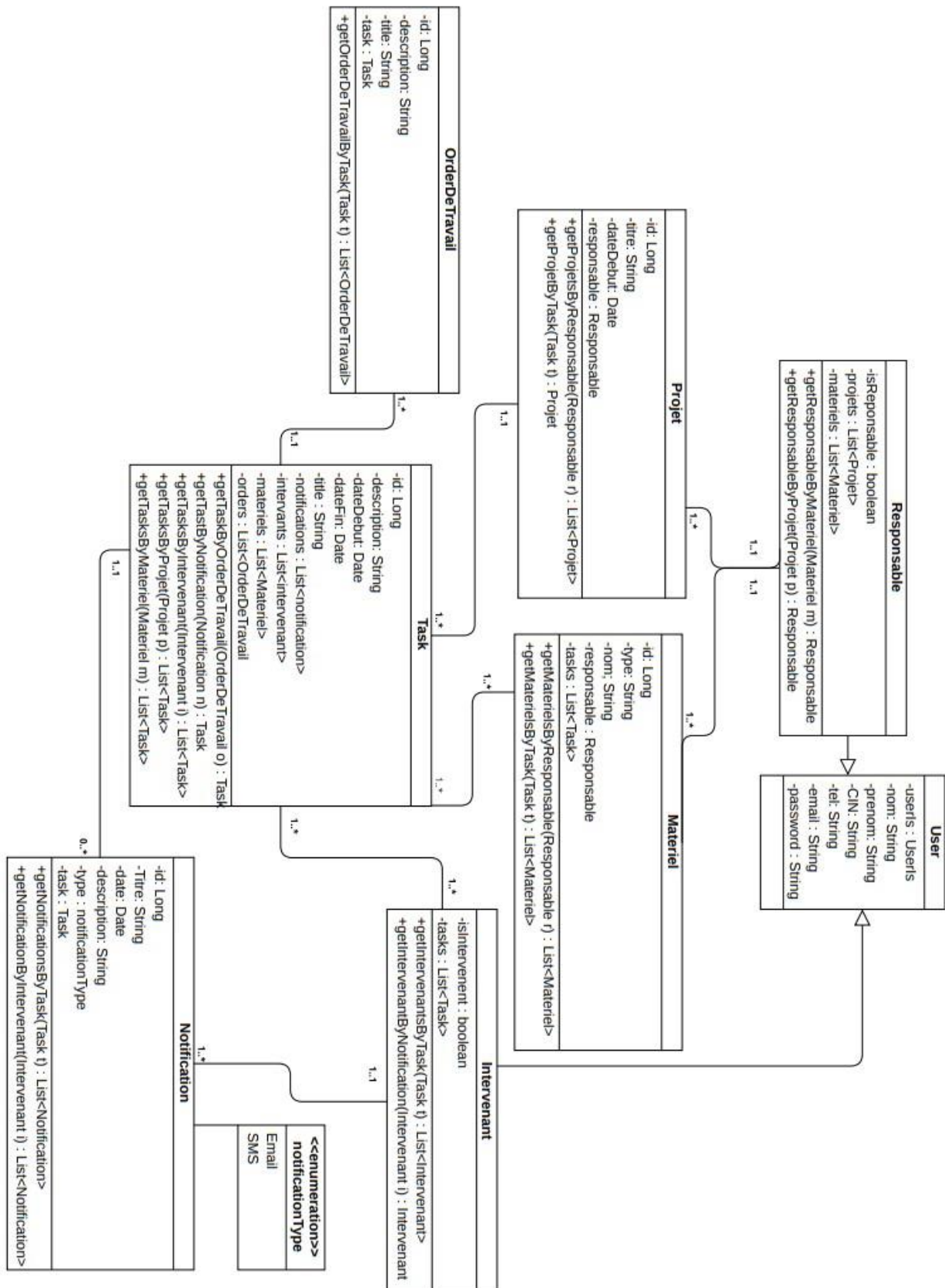


Figure 7 : Diagramme de classe, Système de gestion et de planification des projets

B. Partie génération de diagramme BPMN

1. **BpmnModelInstance**

- Méthodes:
 - **newInstance()**: Crée une nouvelle instance de modèle BPMN.
 - **readFromInputStream()**: Lit un fichier BPMN à partir d'un `InputStream` et crée une instance de modèle BPMN.
 - **writeToOutputStream()**: Écrit un modèle BPMN dans un `OutputStream`.

2. **Process** (étend **FlowElementsContainer**)

- Attributs:
 - **List<FlowElement> flowElements**: Une liste des éléments de flux contenus dans le processus.
- Méthodes:
 - **getFlowElements()**: Retourne la liste des éléments de flux.
 - **addFlowElement()**: Ajoute un élément de flux au processus.

3. **FlowElement**

- Méthodes:
 - **getId()**: Retourne l'ID de l'élément de flux.
 - **setId()**: Définit l'ID de l'élément de flux.

4. **FlowNode** (étend **FlowElement**)

- Méthodes:
 - **getIncoming()**: Retourne la liste des séquences de flux entrantes.
 - **getOutgoing()**: Retourne la liste des séquences de flux sortantes.

5. **Event** (étend **FlowNode**)

- Pas de nouveaux attributs ou méthodes, utilise ceux de **FlowNode**.

6. **Activity** (étend **FlowNode**)

- Pas de nouveaux attributs ou méthodes, utilise ceux de **FlowNode**.

7. **Gateway** (étend **FlowNode**)

- Pas de nouveaux attributs ou méthodes, utilise ceux de **FlowNode**.

8. **SequenceFlow** (étend **FlowElement**)

- Attributs:
 - **FlowNode sourceRef**: Référence à l'élément de flux source.
 - **FlowNode targetRef**: Référence à l'élément de flux cible.
- Méthodes:
 - **getSourceRef()**: Retourne la référence de l'élément de flux source.
 - **setSourceRef()**: Définit la référence de l'élément de flux source.
 - **getTargetRef()**: Retourne la référence de l'élément de flux cible.
 - **setTargetRef()**: Définit la référence de l'élément de flux cible.


9. **DataObject** (étend **FlowElement**)

- Pas de nouveaux attributs ou méthodes, utilise ceux de **FlowElement**.


10. **DataStore** (étend **FlowElement**)

- Pas de nouveaux attributs ou méthodes, utilise ceux de **FlowElement**.


C. Interfaces mises en œuvre




JN chantier




User Login





Login

Forgot your password ?



Java Project

Home


My Tasks

Charts


Profile

LogOut


Dashboard




projects




Intervenants




Tasks




profile



affecter une tache



Import/export

**Java Project**

Home

My Tasks

Charts

Profile

LogOut

Ajouter un projet

Titre :

Responsable :

priorité :

date début :

date Livraison :

Ajouter

**Java Project**

Home

My Tasks

Charts

Profile

LogOut

Ajouter une tâche

Projet :

tâche :

date debut :

date Livraison :

description :

Ajouter

V. Conclusion

Dans ce chapitre, nous avons traité l'étude conceptuelle du projet. En premier lieu, nous avons précisé la méthodologie utilisée. Ensuite, nous avons identifié les différents acteurs et les diagrammes de contexte utilisés. Cela étant, nous avons présenté les différents diagrammes de contexte, de cas d'utilisation et de classe pour le système.

Premièrement, nous décrivons les classes et les associations pour la gestion du chantier de construction. La classe **Capteur** recueille des données sur le bruit et est liée au **Projet**. La classe **Projet** contient une liste de **Task**, d'**Intervenant**, d'**OrderDeTravail**, un **Responsable**, et une liste de **Materiel**. Chaque **Task** est associée à un ou plusieurs **Intervenant** et chaque **Intervenant** peut être impliqué dans plusieurs **Task**. Les **OrderDeTravail** sont liés au **Materiel** et au **Responsable**.

Deuxièmement, nous décrivons les classes et les associations pour la génération de diagrammes BPMN. La classe **BpmnModelInstance** est liée à la classe **FlowElementsContainer** via la classe **Process**. Les classes **FlowElement**, **FlowNode**, **Event**, **Activity**, **Gateway**, **SequenceFlow**, **DataObject** et **DataStore** sont toutes liées les unes aux autres pour représenter les différents éléments d'un diagramme BPMN.

Troisièmement, nous décrivons les associations entre les classes des deux modèles. Pour lier le modèle de gestion de chantier au modèle de génération de diagramme BPMN, nous associons la classe **Projet** à la classe **BpmnModelInstance**. De cette manière, chaque projet a une instance de modèle BPMN associée qui décrit les différentes tâches et les relations entre elles. Les données recueillies par le **Capteur** peuvent être utilisées pour mettre à jour l'état des tâches et des intervenants dans le modèle BPMN.

Enfin nous avons les images de l'interface graphique de l'application finale développée.

Partie Capteurs & Transmission

I. Récapitulatif des choix du projet au niveau capteur

Au premier semestre, nous avons donc choisi d'utiliser une carte raspberry comme passerelle dans notre projet.

L'information allait du capteur de son, branché à une arduino, à la carte raspberry, qui était censée la transmettre au Jumeau Numérique. Néanmoins aucun choix n'avait été arrêté sur la transmission des données de la passerelle raspberry au Jumeau.

Le travail à réaliser portait donc sur la transmission des données au jumeau, mais aussi sur la réception/acquisition des données. En effet au semestre précédent seul des tests d'envoi de caractères avaient été validés. Un envoi groupé de données utiles était absent.

A) Choix effectués

Le principal choix portait donc sur l'envoi au Jumeau Numérique.

Dans un premier temps nous voulions sauvegarder les fichiers de données sur la raspberry, puis les envoyer au JN. Cela posait néanmoins plusieurs problèmes. D'abord au niveau de l'automatisation du projet, cela nécessitait de lancer manuellement plusieurs codes, ce qui contrecarrait complètement la volonté d'avoir un jumeau fonctionnant de manière autonome, ce qui était comme même l'objectif du projet.

De plus nous avons été confrontés à des problèmes de sauvegardes et de format non décidable par l'ordinateur lors de la lecture du fichier.

Après notre rendez-vous de mi-projet nous avons décidé d'abandonner cette idée, notamment sous l'impulsion de Monsieur Champeau qui nous a expliqué qu'utiliser un fonctionnement de client-serveur serait plus simple et plus adapté.

Pour l'envoi de la raspberry nous avons donc choisi un modèle client-serveur, avec les compétences que nous avons acquis en cours de conception logicielle au semestre 4.

B) Organisation finale

En résumé, l'organisation finale du projet se résume par la figure suivante :

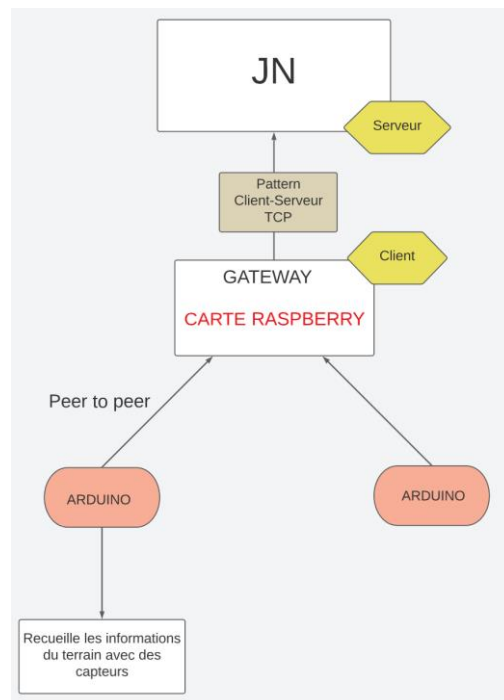


Figure 9 : Schéma représentant l'organisation du projet

On a l'installation suivante :

On voit sur cette image :

- la carte Arduino branchée au capteur de son
- la carte Raspberry (passerelle)
- les deux dragino shields et antenne permettant la communication en protocole LoRa

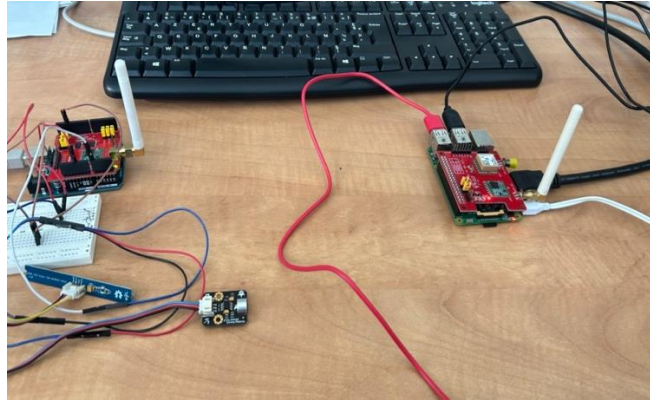


Figure 10 : Composants utilisé lors du projet

II. Acquisition et transmission des données terrain

A) Choix du capteur de son

La mesure du bruit se fait par l'intermédiaire d'un capteur de son branché sur la carte Arduino.

Le capteur choisi est le Analog Sound V2 de Sensor Robot. D'autres capteurs ont été testés avec peu de succès.

Il a une plage de tension de 3.3V à 5V, une interface analogique et peut ainsi détecter le niveau sonore ambiant. Le capteur fonctionne en transformant les variations de pression acoustique en signaux électriques, qui peuvent ensuite être lus.

A partir de la tension mesurée par le capteur, on calcule la valeur du son en db par conversion. C'est ensuite cette valeur qui est manipulée au long du projet.



Figure 11 : Capteur de son utilisé

B) Transmission à la passerelle

Dans la continuité du travail réalisé au semestre 3, on envoie les informations à la Raspberry par peer-to-peer, en utilisant le protocole LoRa.

Tout au long de ce rapport, le protocole de communication LoraWan qui découle de la technologie de modulation des ondes radios LoRa seront confondus.

On parlera de protocole LoRa.

1) Technologie Lora

Il s'agit d'une technologie de communication sans fil basse consommation qui permet l'échange de petits paquets de données sur des fréquences radio (868 MHz en France). Lora est notamment

utilisée dans les technologies IoT (Internet of Things). Elle a donné naissance au protocole LoRaWAN.

Le réseau LoRa (LoRaWAN) est un LPWAN : Low Power Wide area network. C'est un réseau bas-débit et de longue portée.

Il utilise une modulation de spectre étalé qui permet d'étaler le signal sur une large bande de fréquences, ce qui rend la transmission plus résistante aux interférences et au bruit.

Cela permet de transmettre des données sur de longues distances : allant de plusieurs kilomètres en zone urbaine (grandes villes) à plusieurs dizaines de kilomètres en zone rurale (campagne). Le réseau est également capable de fonctionner dans des environnements difficiles comme des zones industrielles et les espaces souterrains.

Étant une technologie consommant peu d'énergie et à faible coût, la technologie LoRa peut être préférée dans certains cas à des technologies plus difficiles à mettre en place comme la 4G, 5G.

Bien qu'elle ait un débit faible allant de 0,3 à 0,5 kbit/s, LoRa est adaptée à la transmission de données provenant de capteurs tels que des thermomètres, des capteurs d'humidité ou des détecteurs de mouvement. Dans notre cas, nous allons l'utiliser pour envoyer des données sur le son, de la carte Arduino à notre passerelle, la carte Raspberry.

La structure d'un réseau LoRa est souvent la suivante : des capteurs qui prennent l'information, l'envoient à une passerelle qui reçoit puis transmet les données à des serveurs réseau. Cela ressemble beaucoup à la structure utilisée dans notre projet.

2) Configuration et Shields utilisés

On rappelle ici la configuration et les shields utilisés. On utilise les configurations suivantes :

Pour faire communiquer l'Arduino et la Raspberry en *peer-to-peer* on utilise des Lora-shield avec des antennes de 864 Hertz placées sur ces derniers.

Le Dragino Lora-shield est un émetteur-récepteur longue portée, permettant l'envoi à très longue distance et avec un faible débit de données. Il offre une grande immunité aux interférences tout en permettant une faible consommation de données. De plus, le shield Dragino permet l'envoi d'information à grande portée, et on en dispose d'un sur l'Arduino et d'un sur la Raspberry. Sur le shield Dragino, on trouve un module Lora SX127x, qui permet la communication avec un autre module de même type.



Figure 12 : Shield Dragino utilisé lors du projet

On configure ensuite la Raspberry en autorisant le protocole SPI, qui permet la communication par Lora, et pour l'Arduino, on utilise une carte de type UNO, notamment la library RadioHead. Sur l'Arduino, on a donc aussi relié un shield Dragino, sur lequel on a aussi un module Lora SX127x.

3) Format des données envoyées par l'Arduino

On souhaite donc envoyer des informations au JN en passant par la passerelle. Comme la passerelle raspberry se contente d'être un intermédiaire, c'est au niveau de l'Arduino qu'on doit choisir le format des données.

On veut envoyer les données suivantes :

- Nom du capteur qui envoie la donnée
- Valeur du son en db
- Date de l'envoi (format time stamp)

Le format est le suivant : Nom Capteur, Valeur du son en db, Date de l'envoi.

Ce format est important car c'est le même qui sera envoyé au JN, le client, une fois que l'information est passé par la passerelle, et qui sera directement stocké dans la base de données.

4) Mise en œuvre de la solution

Le code Arduino est le code receiver.ino. Ce code vise donc à mesurer la valeur du son en db, et à l'envoyer à la passerelle.

Toutes les mesures et envoi sont réalisées dans la loop.

Le son est mesuré en prenant la valeur de tension sur le port A, et en la convertissant pour avoir la valeur en db.

```
//Lecture du son avec le capteur branché
adc = analogRead(MIC);
pdB = (adc-89.32827) / 10.000140 + 10;
```

Figure 13 : Lecture du son et conversion en db

Le deuxième rôle du code, envoyer les 3 valeurs à la passerelle raspberry, est rempli avec la fonction send de rf95.

```
delay(2000);
rf95.send((uint8_t*)Total.c_str(),Total.length()+1);
//Cette ligne envoie le nom du capteur sur la raspberry et la valeur du son avec le format demandé
//delay(2000);
```

Figure 14 : Envoi des données avec le module rf95

On envoie les informations sous forme de strings. Un des problèmes rencontrés dans l'envoi était qu'en fonction des cas, on envoyait qu'une des deux informations, soit le nom soit la valeur du son en db, et non pas les deux.

Ce problème a été réglé par l'utilisation d'un double delay qui entourait le code. Néanmoins comme dans un second temps on envoie les deux informations en même temps (son et nom), ce problème ne se pose plus.

III. Mise en place de la passerelle

A) Fonction de la passerelle

La passerelle, carte Raspberry, permet donc de transférer les informations de l'Arduino au serveur. Elle remplit donc à la fois les fonctions de réception et d'émission.

La passerelle permettrait théoriquement lors d'une amélioration du projet d'utiliser d'autres capteurs et de retransmettre toutes les informations au jumeau Numérique. Comme indiqué précédemment, ce système de passerelle pour transmettre les informations capteurs à des serveurs est souvent utilisé pour les réseaux LoRaWAN.



Figure 15 : Composants d'une Carte Raspberry

B) Réception des données capteurs

Le code python sur la raspberry sert à la fois à réceptionner les données envoyées par protocole Lora par l'arduino et à envoyer ces dernières par protocole client-serveur.

Dans cette partie, on se concentre sur la partie réception du code.

Par ailleurs, cette partie et aussi le code Arduino a été réalisé à l'aide d'un tutoriel fourni par notre encadrant Monsieur Inizan, présent sur le site circuit-digest.com.

Cela nous a permis d'avoir une idée approfondie de la mise en place de classe pour gérer la communication peer-to-peer en Lora pour une communication Arduino-Raspberry.

EXPLICATION DU CODE :

Le programme commence par la définition de la classe LoRa et les trois méthodes init, start et on_rxdone.

La méthode init permet d'avoir un module Lora initialisé à 433 MHz, et une bande passante de 125 KHz. On met ensuite le mode veille pour des raisons d'économie d'énergie.

Dans la fonction start, on configure le module en tant que récepteur, et on obtient les valeurs d'état de fonctionnement.

Dans la fonction on_rx_done, toute l'information est transférée dans le payload. C'est cette variable qui sera manipulée au cours de l'envoi et qui sera print dans le shell, une fois décodée. *La variable payload est centrale, c'est elle qui contient toutes les informations.*

La suite du programme permet d'afficher les valeurs reçues dans la console et de les envoyer au serveur (ce processus est expliqué dans la partie suivante). Pour print le payload de manière à contrôler les informations qui circulent on va d'abord utiliser la fonction `decode()` pour éviter un affichage des caractères en hexa décimal.


```
def on_rx_done(self):  
    print("\nRecu : ")  
    print("On a reçu un paquet")  
    i = 0  
    print(i)  
    i=i+1  
    #print(i)  
    self.clear_irq_flags(RxDone=1)  
    payload = self.read_payload(nocheck=True)  
    print(bytes(payload).decode("utf-8", 'ignore'))
```

Figure 16 : Définition de la fonction rx_done

C'est dans cette fonction qu'on obtient le paquet contenant toutes les informations, le payload.

On observe ainsi sur la console les informations suivantes :

```

35 print("Connection on {}".format(port))
36
37 class LoRaRcvCont(LoRa):
38
39     i = 0
40
41     def __init__(self, verbose=False):
42
43
44
Shell x
socket1.connect((hote,port))
OSError: [Errno 113] No route to host

Recu :
On a reçu un paquet
0
=====
raw payload [255, 255, 0, 0, 53, 48, 44, 67, 97, 112, 49, 44, 0]
0050,Cap1,0
=====
type <class 'str'>

```

Figure 17: Affichage des informations sur le terminal de la raspberry

On note qu'on retrouve bien sur la passerelle le même format de donnée que celui envoyé sur l'Arduino :

Valeur du son en db, Identifiant du capteur. Donc ici : 50, Cap1.

La date qui ne provient pas de l'arduino mais qui est calculé avec la fonction `datetime.date.now()` peut aussi être affichée.

C) Transmission au JN par client-serveur

1) Pattern client-serveur TCP

Pour faire communiquer la passerelle raspberry et le JN, on passe donc par un pattern client-serveur TCP.

La communication utilise le protocole TCP, (Transmission Control Protocol) couche de transport qui assure la fiabilité entre les deux machines.

La passerelle est le client, et le Jumeau numérique est le serveur. Le pattern utilisé, provient du cours de Mr Champeau en conception logicielle, pour la construction du serveur.

Le protocole suivi est le suivant :

Le serveur et un client sont lancés. Chaque serveur TCP est associé à un processus, et à un port et adresse IP. Le client se connecte ensuite au serveur identifié par le port et l'adresse IP.

Une fois connecté, la transaction peut avoir lieu. A chaque envoi on crée une socket, que l'on ferme une fois l'envoi de la donnée effectué.

2) Mise en œuvre

C'est dans le code python de la raspberry qu'on se connecte au serveur.

On commence par créer la socket et on vise à se connecter au serveur voulu en renseignant son adresse IP et le nom de port. Cela est fait par les lignes suivantes :

```
socket1 = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

socket1.connect((hote,port))
print("Connection on {}".format(port))

socket1.send(bytes(payload))

socket1.send(bytes(str(date),encoding='utf-8'))

socket1.close()
```

Figure 18 : Code pour créer la socket

On crée la socket avec la ligne `socket1 = socket.socket`

Puis on connecte la socket au serveur en renseignant l'adresse IP et le port, préalablement écrit dans les variables `hôte` et `port` au début du code.

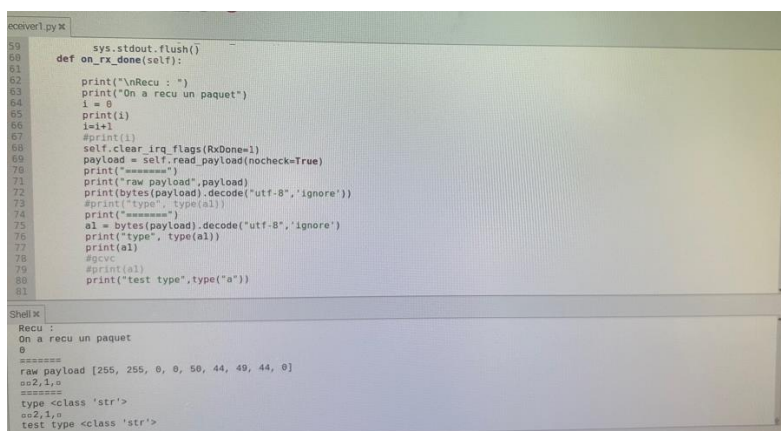
C'est dans la fonction `rx_done`, celle qui s'exécute à chaque passage que l'on crée une socket, au port et à l'adresse correspondante.

Puis on envoie le payload, puis la date, avec la fonction `send`. Donc toutes les données reçues sont directement envoyées au serveur.

A noter que l'on doit entourer le payload de la fonction `bytes` pour que cela fonctionne.

Une fois que cela est réalisé, on peut fermer la socket. On a donc fait le choix de créer une socket à chaque envoi, et de ne pas réutiliser la même à chaque fois.

Finalement au niveau du terminal de la Raspberry, on peut observer les messages suivants :



```
ecouter1.py
59 sys.stdout.flush()
60 def on_rx_done(self):
61     print("\nReçu : ")
62     print("On a reçu un paquet")
63     i = 0
64     print(i)
65     i+=1
66     @print(1)
67     self.clear_irq_flags(RxDone=1)
68     payload = self.read_payload(nocheck=True)
69     print("=====")
70     print("raw payload",payload)
71     print(bytes(payload).decode("utf-8","ignore"))
72     @print("type", type(a1))
73     print("=====")
74     a1 = bytes(payload).decode("utf-8","ignore")
75     print("type", type(a1))
76     print(a1)
77     @gcvc
78     @print(a1)
79     print("test type",type("a"))
80
81
Shell X
Reçu :
On a reçu un paquet
0
====
raw payload [255, 255, 0, 0, 50, 44, 49, 44, 0]
2,1,0
type <class 'str'>
2,1,0
test type <class 'str'>
```

Figure 19 : Affichage des données sur la raspberry

On note qu'on obtient bien les données reçues par la passerelle. Ici on a une valeur de 2 db et un capteur d'identifiant 1.

La mention raw payload qui apparait dans le terminal correspond aux valeurs envoyées en hexa.

IV. Utilisation du code (destiné aux prochaines années)

A) Explication

Sous l'impulsion de Monsieur Inizan, nous avons choisi de rajouter cette partie afin de permettre une éventuelle relecture plus facile et une meilleure compréhension des codes capteurs pour les années suivantes. En effet le projet étant éventuellement destiné à être amélioré/étendu, cette partie paraît utile.

Les paragraphes suivants auront pour objectif de permettre à un utilisateur ayant lu l'explication des codes capteurs de les utiliser pour une première démonstration.

B) Démonstration

Tout d'abord l'utilisateur doit brancher la carte Arduino à un ordinateur et lancer le code ARDUINO_SENDER_2.ino dessus. En ouvrant le moniteur série, il peut voir les valeurs d'envoi qui défile.

Ensuite l'utilisateur lance le code python Receiver.py sur la carte Raspberry.

A ce moment là l'utilisateur doit normalement voir les informations envoyées par l'Arduino défiler dans le terminal de la Raspberry. Comme le serveur n'est pas encore lancé, des messages d'erreurs de non-connexion de la socket vont apparaître.

Sur le terminal doit apparaître la valeur du son mesuré, le nom du capteur et la date.

C'est les trois informations qui seront envoyées.

Il faut maintenant saisir l'adresse IP de l'ordinateur qui joue le rôle de serveur dans le code Receiver.py.

Il faut aussi saisir le port utilisé.

L'utilisateur peut alors maintenant lancer le code serveur, qui doit correspondre au bon port et à l'adresse IP, pour nous il s'agit du JN codé en Java. On a maintenant à la fois le client et le serveur qui sont lancés. Parfois il faut lancer le serveur avant pour être sûr d'éviter des bugs de connexion.

ATTENTION : Au niveau de la connexion client-serveur TCP, le pattern ne permet que la connexion entre la raspberry et le serveur si ces derniers sont connectés au même réseau wifi.

Sur le terminal de la Raspberry doit apparaître les informations de connexion, et un message de connexion réussie doit apparaître.

On a bien les informations recueillies par l'Arduino qui sont envoyées à la Raspberry en peer-to-peer Lora, puis envoyées au serveur par transmission client-serveur.

Bilan des réalisations du projet

Pour conclure, nous sommes en mesure de présenter une version d'un jumeau numérique de chantier en fin de projet. Cette version remplit en majorité le cahier des charges réalisé en début de projet, au semestre 3.

Certaines exigences n'ont toutefois, pour des raisons de temps, pas été remplies : la prise en compte de données capteurs de type différents, ou la gestion complète des décisions du contremaître.

Au niveau de la partie capteurs et transmission, tous les objectifs de début de projet ont été atteints. L'information sonore est captée par le micro branché à l'Arduino, transmise à la passerelle en peer-to-peer via Lora, puis au jumeau numérique via une communication client-serveur, accompagné de d'autres données. Le suivi de M. Inizan a permis de surmonter les problèmes de transmission rencontrés pour le dialogue peer-to-peer entre l'Arduino et la Raspberry, ainsi que le traitement de l'information sur l'Arduino.

Une des perspectives d'améliorations serait de rajouter plusieurs capteurs à la base de l'application, pour capter d'autres données (température par exemple). Le système de passerelle permet justement de faire cela.

Au niveau de la partie JN a été détaillée, une première version de la modélisation du chantier a été implémentée, et de la prise en compte de données (capteurs) en provenance du chantier pour suivre l'état d'avancement, a été validée (interface graphique et génération de diagramme BPMN). C'était l'objectif principal du projet et il a été réalisé.

Bilan Travail en Groupe

L'ambiance du groupe est globalement bonne, et a permis le bon déroulement du projet.

L'auto-évaluation de groupe a néanmoins souligné quelques dysfonctionnements au niveau de la répartition de la charge de travail, aussi accentuée par le départ de notre collègue Vincent en substitution.

Continuer le projet à 4 était plus challengeant au niveau du temps, mais a été possible en se réorganisant le travail.

En définitive, ce projet système a été l'occasion de mettre en pratique de nombreuses compétences acquises lors du cycle de formation à l'ENSTA Bretagne et sera bénéfique pour la suite de notre scolarité.

Références

Image de titre : <https://www.esrifrance.fr/jumeau-numerique.aspx>

Annexe : Récapitulatif des concepts d'UML 2

A titre de synthèse, il est proposé dans cette annexe une liste récapitulative des concepts d'UML 2 présentés dans ce rapport. Ce récapitulatif est donné par diagramme après un rappel des concepts communs.

Concepts communs à tous les diagrammes :

- stéréotype,
- mot-clé,
- note,
- contrainte.

Concepts du diagramme de classe (DCL) :

- Attribut,
- Opération,
- Visibilité,
- Association,
- Navigabilité,
- Classe et classe abstraite,
- Rôle,
- Multiplicité,
- Dépendance,
- Agrégation et composition
- Qualification,
- Généralisation et spécialisation
- Héritage.

Diagramme des cas d'utilisation (DCU) :

- Acteur,
- Interaction,
- Scénario nominal,
- Scénario alternatif,
- Pré et post condition,
- Inclusion, extension, généralisation de cas d'utilisation.

Concepts du diagramme d'activité (DAC) :

- Action,
- Activité,
- Flot de contrôle,
- Flot de données,
- Noeud de jonction,
- Noeud de test-décision,
- Noeud de fusion-test,
- Noeud de données,
- Pin d'entrée et de sortie,
- Partition (couloir d'activités).

Concepts du diagramme de séquence (DSE) :

- Ligne de vie,
- Message synchrone et asynchrone,
- Fragment d'interaction,
- Opérateur,
- Interaction,
- Opérateur alt,
- Opérateur opt,
- Opérateur loop,
- Opérateur par,
- Opérateurs strict/weak,
- Opérateur break,
- Opérateurs ignore/consider,
- Opérateur critical,
- Opérateur negative,
- Opérateur assertion,
- Opérateur ref.