Rapport de stage

Intitulé de la mission confiée

Maxime CLUCHAGUE

Année 2016-2017

Stage de 2e année réalisé *à l'INRIA (Sophia-Antipolis)*

Maître de stage : *Yves PAPEGAY*
Encadrant universitaire : *Prénom Nom*

Déclaration sur l’honneur de non-plagiat

Je soussigné(e),

Nom, prénom : _____

Elève-ingénieur(e) régulièrement inscrit(e) en 2e année à TELECOM Nancy

N° de carte d’étudiant(e) : _____

Année universitaire : 20__ - 20__

Auteur(e) du document, mémoire, rapport ou code informatique intitulé :

_____
_____

Par la présente, je déclare m’être informé(e) sur les différentes formes de plagiat existantes et sur les techniques et normes de citation et référence.

Je déclare en outre que le travail rendu est un travail original, issu de ma réflexion personnelle, et qu’il a été rédigé entièrement par mes soins. J’affirme n’avoir ni contrefait, ni falsifié, ni copié tout ou partie de l’œuvre d’autrui, en particulier texte ou code informatique, dans le but de me l’accaparer.

Je certifie donc que toutes formulations, idées, recherches, raisonnements, analyses, programmes, schémas ou autre créations, figurant dans le document et empruntés à un tiers, sont clairement signalés comme tels, selon les usages en vigueur.

Je suis conscient(e) que le fait de ne pas citer une source ou de ne pas la citer clairement et complètement est constitutif de plagiat, que le plagiat est considéré comme une faute grave au sein de l’Université, et qu’en cas de manquement aux règles en la matière, j’encourrais des poursuites non seulement devant la commission de discipline de l’établissement mais également devant les tribunaux de la République Française.

Fait à _____, le ___/ ___/ 20___

Signature :

Résumé *(250 mots max)*
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam posuere, odio sit amet tincidunt cursus, mauris justo feugiat nisl, at condimentum tellus neque et odio. Etiam in porttitor nisl, eget feugiat nulla. Etiam sollicitudin dui lectus, et ultricies libero tempus sed. Pellentesque lobortis pretium nisi dapibus tincidunt. Nam massa tortor, elementum ac fermentum a, sollicitudin nec felis. Quisque magna lectus, mattis quis ipsum id, blandit porta diam. Suspendisse potenti. Donec posuere ipsum sed aliquet ullamcorper.

Integer hendrerit nulla vestibulum rhoncus fringilla. Cras tincidunt turpis sit amet ultricies placerat. Mauris pellentesque pharetra lectus, at consequat nunc. Donec nec erat eros. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Cras in lacus non nulla auctor tincidunt. Ut tristique diam aliquam ante iaculis, eu euismod risus egestas. Donec hendrerit ante ipsum, eu semper arcu hendrerit eget. Praesent lobortis, orci non vulputate faucibus, felis leo molestie nisl, in venenatis nisl elit ac tortor. Etiam fermentum nunc tellus, ut elementum lacus sodales sed.

Maecenas at consectetur justo, in adipiscing elit. Nunc lacinia elit et pellentesque dignissim. Maecenas tristique eu sapien sit amet aliquet. Aliquam erat volutpat. Donec a neque ac diam luctus viverra. Proin volutpat ultricies lacus, et condimentum enim congue vitae. Quisque pellentesque odio vitae mi pellentesque, ut iaculis lorem suscipit. Pellentesque nec ultrices quam. Phasellus quis eros tincidunt, pulvinar enim eu, viverra mauris. Curabitur suscipit facilisis rutrum. Nam adipiscing, felis vel pretium tristique, dolor ligula lacinia lacus, et facilisis massa lacus a massa. Suspendisse potenti. Cras iaculis placerat.

Mots-clés : *(3 mots-clés)*

Summary *(250 mots max)*
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam posuere, odio sit amet tincidunt cursus, mauris justo feugiat nisl, at condimentum tellus neque et odio. Etiam in porttitor nisl, eget feugiat nulla. Etiam sollicitudin dui lectus, et ultricies libero tempus sed. Pellentesque lobortis pretium nisi dapibus tincidunt. Nam massa tortor, elementum ac fermentum a, sollicitudin nec felis. Quisque magna lectus, mattis quis ipsum id, blandit porta diam. Suspendisse potenti. Donec posuere ipsum sed aliquet ullamcorper.

Integer hendrerit nulla vestibulum rhoncus fringilla. Cras tincidunt turpis sit amet ultricies placerat. Mauris pellentesque pharetra lectus, at consequat nunc. Donec nec erat eros. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Cras in lacus non nulla auctor tincidunt. Ut tristique diam aliquam ante iaculis, eu euismod risus egestas. Donec hendrerit

Mots-clés : (3 mots-clés)

Avant-propos

Remerciements

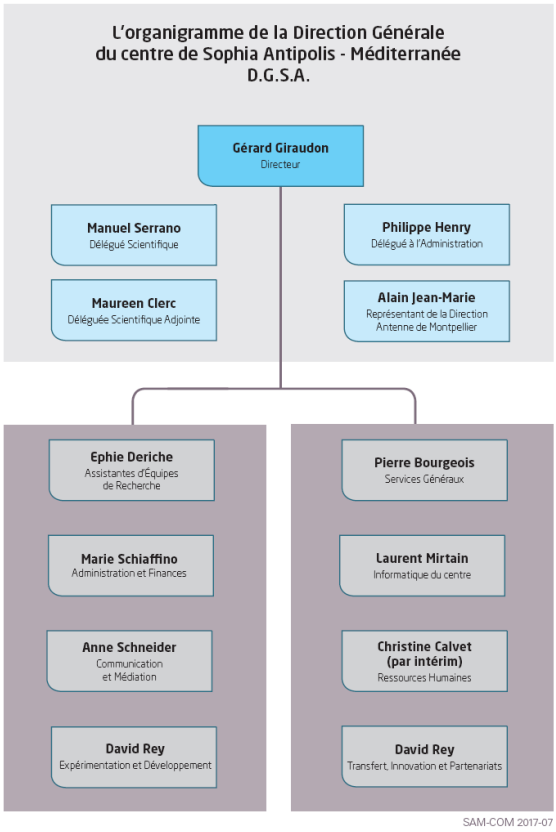
Sommaire

1. Présentation de l’Inria

1.1. Présentation de L’Inria

(A REFORMULER)

L'Institut national de recherche en informatique et en automatique (Inria) est un institut de recherche français en mathématiques et informatique. Créé le 3 janvier 1967 dans le cadre du plan « Calcul », il a le statut d'établissement public à caractère scientifique et technologique. Son objectif est de mettre en réseau les compétences et talents de l'ensemble du dispositif de recherche français et international, dans ses domaines de compétence. Ce stage s’est déroulé à l’Inria de Sophia-Antipolis située dans le département des Alpes-Maritimes, qui est la plus importante technopole de France et d'Europe, fondée en 1969 par l'Association Sophia Antipolis et le groupement d'intérêt économique SAVALOR.



1.2. L’Équipe-Projet Hephaistos

(A REFORMULER)

Le projet HÉPHAÏSTOS a été initialisé le 1er Janvier 2014. Il regroupe des chercheurs INRIA et des collaborateurs extérieurs, décidés à conjuguer leurs compétences pour développer des dispositifs d’assistance robotisés (au sens large), pour les personnes fragiles (âgées et handicapées), ainsi que leurs aidants et pouvant fournir des informations à la communauté médicale.

La définition de ce projet a été proposée après deux ans de consultation, qui ont conduit à identifier des priorités (par exemple l’aide à la mobilité) et des lignes de développement (faible coût et à caractère non intrusif, travail en flotte d’objets connectés…). Ces développements qui vont du concept au prototype, en abordant l’intégralité des aspects mécaniques, électroniques, logiciels et automatiques, s’appuient sur des méthodes appropriées de modélisation, de conception et d’évaluation, avec en particulier la prise en compte des incertitudes. Un axe majeur du projet est d’ailleurs l’étude des robots parallèles, qui constituent une réponse intéressante au problème de la mobilité.

D’un point de vue mathématique, les travaux du projet s’appuient sur la théorie des jeux et l’analyse par intervalles.

2. Étude des besoins

2.1. Présentation du contexte

L'équipe Hephaistos de l'INRIA de Sophia-Antipolis travaille actuellement sur un projet qui a pour but d'étudier le déplacement de personnes âgées vus comme des individus non indentifiables au sein de l'infrastructure de soin, l'Institut Claude Pompidou. Ce projet est développé en collaboration avec les équipes de médecines de cet institut. Il doit permettre aux équipes médicales sur place de mieux comprendre le comportement de ces personnes dans une telle infrastructure en établissant des modèles statistiques afin d'adapter l'environnement aux observations. Pour établir de tels modèles, l'équipe récupère les données relatives à ces flux de déplacements via des capteurs Infrarouges de proximité installés au sein de cette infrastructure.

2.2. Description et analyse du problème

Actuellement le traitement de ces données se fait par des programmes C embarqués sur les cartes Phidgets connectées aux capteurs. Ces programmes analysent, interprètent et enregistrent ces données capteurs sur des fichiers log journaliers. Cependant cette architecture pose un certain nombre de problèmes comme la limitation en terme de puissance de calcul et la limitation de mémoire pour le stockage des données. De plus l'équipe ne dispose d'aucune interface permettant de visualiser l'état de ces dispositifs en temps réel. Mon projet a pour but de développer une architecture qui permettrait, d'une part, de pallier à ces différents problèmes et d'autre part d'implémenter un ou plusieurs prototypes d'interfaces (si la durée du stage le permet) permettant la visualisation en temps réel de l'état de ces différents dispositifs robotiques. Parmi les prototypes demandés : une visualisation en temps réel du nombre de personnes présentes dans chacune des pièces au sein de l'Institut Claude Pompidou.

2.3. Cahier des charges

Le but de mon projet est de développer un ou plusieurs prototypes d'application interfaçant différents capteurs et actionneurs existants. Ces applications doivent être accessibles depuis un écran tactile de 65'' muni d'un cœur Android dont dispose l'équipe Hephaistos et, si possible, depuis n'importe quel machine (Linux, Mac OS, Windows, Android, IOS, Windows Phone...). Un de ces prototypes d'interface demandé est une visualisation en temps réel de dispositifs robotiques d'assistance aux personnes fragiles. Ce prototype doit de plus afficher le nombre de personnes présentes au sein d'un institut de soin (dans le cadre du projet Hephaistos il s'agit de l'Institut Claude Pompidou) où un réseau de capteurs de proximité est déployé sur place. Dans un premier temps il est donc nécessaire de développer une architecture qui permettra par la suite de développer ces différents prototypes d'interfaces. Celle-ci doit permettre la transmission des données de ces différents dispositifs robotiques vers ces applications. Ces échanges de messages doivent s'effectuer à distance et en temps réel. D'où la nécessité d'utiliser une architecture de type Client-Serveur permettant la transmission de données à distance sur le réseau internet. Par ailleurs, il m'est demandé d'utiliser la technologie WebSocket pour que les échanges s'effectuent en temps réel et de façon bilatérale entre les clients (les dispositifs robotiques ou l'utilisateur de l'application) et le serveur. Cette architecture doit donc être compatible avec n'importe quel type de dispositif

robotique et n'importe quel prototype d'application de visualisation de ces dispositifs pour que, par la suite, de nouveaux prototypes d'interface puissent être développés.

3. Réalisation et implémentation du produit

3.1. L'architecture développée

3.1.1. Présentation de l'architecture

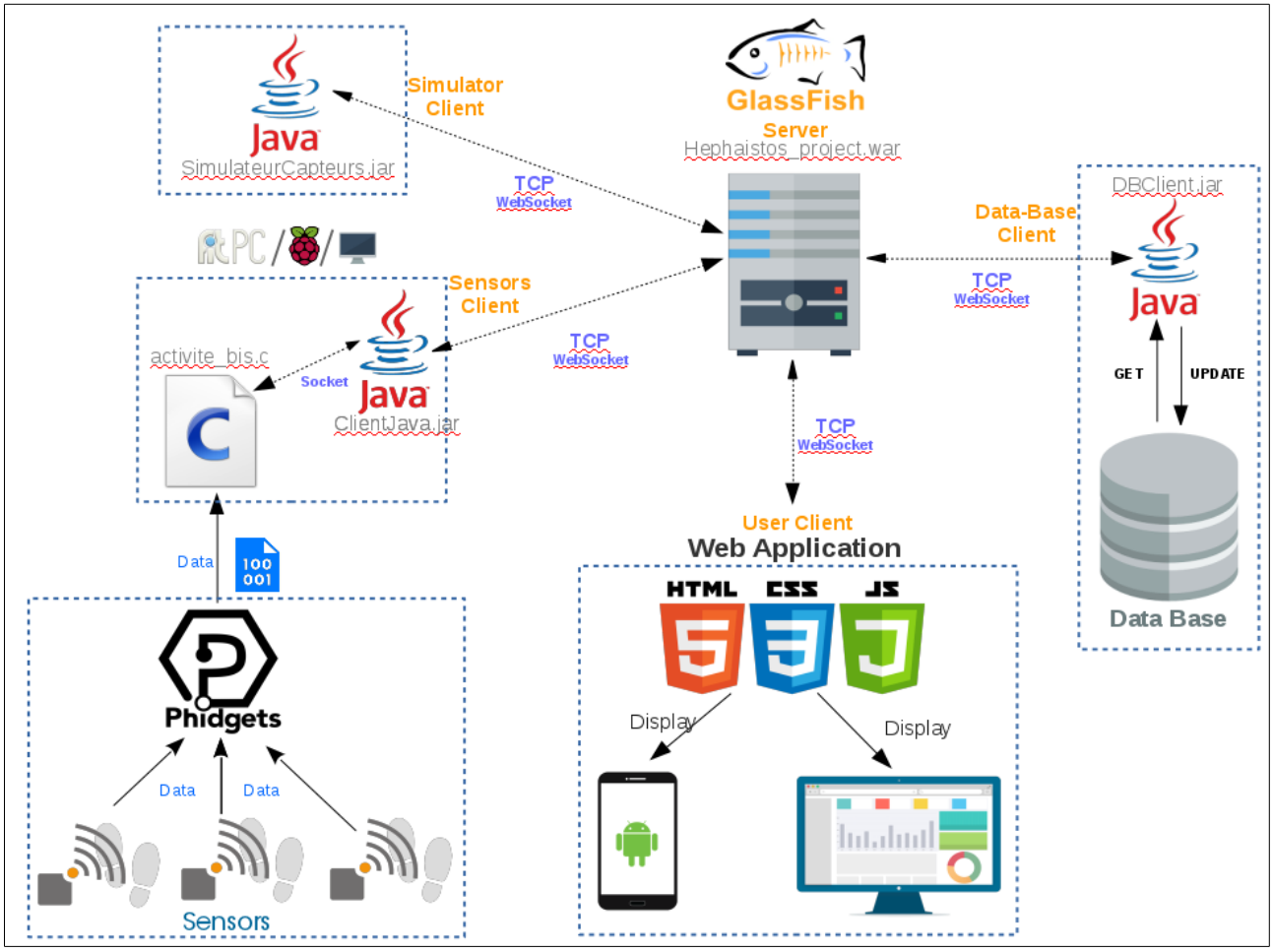
Le projet s'appuie sur une architecture de type client-serveur. Cela désigne un mode de communication à travers un réseau (wifi local, local, internet, localhost ...etc) entre plusieurs programmes. Un programme, appelé programme client, envoie une requête à un autre programme appelé serveur qui répond à ces requêtes. Cela permet à différent programmes de communiquer entre eux à distance. Cependant, contrairement à une architecture client-serveur classique, celle développée dans le cadre de ce projet utilise un mode de communication par WebSocket ce qui permet des échanges bilatéraux entre clients et serveur (voir partie 3.3.3. pour plus de précisions sur le protocole de communication). Ici, l'architecture de ce projet est composée d'un serveur central et de quatre Clients indépendants :

- **Un serveur GlassFish** 4.0 développé en Java qui va permettre de communiquer avec l'ensemble des clients, ou servir d'intermédiaire à plusieurs d'entre-eux afin qu'ils échangent des informations et/ou des données. Lorsque le serveur reçoit des données capteurs il les transfère à tous les Clients qui sont connectés avec celui-ci, y compris au client émetteur. Ce serveur peut-être hébergé sur n'importe quel machine disposant d'une connexion internet et dont le Pare-feu du routeur associé à cette connexion autorise le port de communication du serveur (par défaut 8080 pour les WebSockets). Le serveur ainsi développé pourrait également être déployé sur un service d'hébergement en ligne.
- **Une machine cliente** disposant d'une connexion internet (Rasberry Pi, Fit Pc ou ordinateur) et d'une **carte Phidget** (branchée sur un port USB) sur laquelle sont connectés les capteurs. Cette machine héberge deux programmes qui communiquent par Socket :
  - Un premier programme, écrit en C tourne en boucle infinie. Il va permettre d'effectuer un premier traitement sur les données acquises, gérer les événements capteurs, interagir avec eux, etc. Ce programme utilise la librairie Phidget pour être sensible aux changements d'état des capteurs, et construit le JSON du message qui va être transmis à un second programme par liaison Socket.
  - Ce second programme est un Client écrit en Java et tourne en boucle infinie. Il possède une liaison WebSocket ouverte en permanence avec le serveur. Il a pour rôle de transmettre les données capteurs au Serveur GlassFish à chaque fois qu'il reçoit un message JSON en provenance du programme C (premier programme).

A noter que cette machine doit être reliée aux cartes Phidgets présentes au sein de l'Infrastructure désirée (en l'occurrence l'ICP dans le cadre de ce projet). Elle doit donc être physiquement située aux abords immédiats de l'institut.

- **L'application Web**, un Client qui est à l'écoute du serveur lorsque l'utilisateur décide d'ouvrir le lien WebSocket. Cette application permet à l'utilisateur de visualiser, analyser et interpréter en temps réel les données capteurs (La description détaillée et complète des fonctionnalité de l'application est donnée dans la partie 3.3.). Ce client peut-être lancé depuis n'importe quelle machine à partir de l'adresse internet de l'application, dès lors que le serveur est déployé. Ainsi, plusieurs utilisateurs venant de différentes machines peuvent lancer ce client simultanément, et un utilisateur peut lancer plusieurs fois ce même client sur une même machine.
- Un programme client en boucle infinie qui **interagit avec une base de donnée**. Ce programme est à l'écoute permanente du serveur grâce à une liaison WebSocket avec celui-ci. Il récupère les données émises depuis le serveur, et les ajoute à une base de données (fichier log, csv, MySQL, json, xml... etc). A noter que ce client peut-être lancé de n'importe qu'elle machine (distante ou pas) du serveur.
- Un Client qui joue le rôle de **Simulateur de capteurs** afin de pouvoir travailler en amont sur le développement de l'application et le développement des algorithmes d'analyse des flux générés par les capteurs activés... Ce Client est un programme Java qui transmet aléatoirement les données de 12 capteurs fictifs pendant un temps donné.

La figure suivante résume l'architecture présentée ci-dessus :



3.1.2. Récupération du projet

Le projet est disponible sur la plateforme GitHub d'hébergement et de gestion de projets à l'adresse : 'https://github.com/maximeCluchague/hephaistos_application.git'. Pour le récupérer, télécharger le zip à cette même adresse, puis sélectionner : 'Clone or download' > 'Download ZIP'. Enfin dézipper l'archive sur votre machine. Il est possible de cloner le projet en local sur n'importe quelle machine. Cela permet à l'utilisateur de disposer de la dernière version du projet en cas de mise à jour de celui-ci sur Github, et ce, à l'aide de commandes simples. De plus, si l'utilisateur est un contributeur du projet, celui-ci pourra effectuer des modifications et les enregistrer sur GitHub. Pour cloner le projet en local sur votre machine déplacez-vous à travers un terminal dans le futur répertoire racine de votre projet (à l'aide de la commande cd) puis exécuter la commande :

```
$ : git clone https://github.com/maximeCluchague/hephaistos_application.git
```

Remarque : La documentation du projet est disponible dans le fichier README.md dans le dossier du projet « hephaistos_application » et est directement visible à l'adresse du Git du projet.

3.1.3. Démarrage du serveur

a) Lancement automatique du serveur

Afin de simplifier au maximum le déploiement de l'architecture à travers le lancement du serveur, une procédure automatique de lancement du serveur a été développée. Elle utilise notamment la fonctionnalité « autodeploy » du serveur GlassFish. Cette procédure automatique utilise le domaine « hephaistosDomain »créé par défaut, dont le port administrateur est 5000 et l'adresse est « localhost ». Celui-ci se situe dans le dossier 'glassfish4/glassfish/domains'. Un .war du projet est situé dans 'glassfish4/glassfish/domains/hephaistosDomain/autodeploy' . Tout d'abord, il faut commencer par compiler les fichiers C qui vont permettre la création du domaine, son démarrage et le déploiement de ce fichier .war. Un makefile est disponible pour effectuer ces taches de compilation. Pour l'exécuter, ouvrir un terminal, se déplacer dans le dossier du projet « hephaistos_application » (à l'aide de la commande cd) puis exécuter la commande suivante :

```
$ : make
```

Enfin, pour démarrer le Serveur, exécuter dans ce même terminal

```
$ : ./StartServer
```

Et, pour arrêter le Serveur :

```
$ : ./StopServer
```

b) Lancement manuel du serveur

- **Création du domaine**

  Commencer par ouvrir un terminal dans le dossier « hephaistos_application » du projet et exécuter les commandes suivantes :

```
$ : ./glassfish4/glassfish/bin/asadmin
```

le "asadmin tool" va alors être lancé dans le terminal. vous pouvez quitter "asadmin tool" à tout moment en tapant "exit".

```
$ : create-domain --adminport <admin_port> --profile developer --user admin <nom_domaine>
```

- <admin_port> est le port d'accès de l'administrateur du serveur (ex : 4848) ce port permettra à l'administrateur de modifier les paramètres du serveur
- <nom_domaine> est le nom de domaine que vous allez définir pour votre futur serveur (ex : domainTest)

➢ **Démarrage et arrêt du domaine**

Utiliser la ligne de commande suivante pour démarrer le domaine <nom_domaine> créé précédemment :

```
$ : start-domain <nom_domaine>
```

Pour arrêter un domaine il vous suffit d'exécuter la commande :

```
$ : stop-domain <nom_domaine>
```

➢ **Déploiement du serveur**

Une fois le domaine créé et lancé il faut déployer le serveur. Le serveur est un fichier .war (un exécutable java pour les serveurs). Le .war du projet est disponible dans le dossier du projet. Avant de déployer le .war s'assurer qu'un domaine est démarré. Enfin pour déployer le serveur, exécuter la commande :

```
$ : deploy --port <admin_port> --host <adresse> <PATHwar>
```

- <admin_port> est le port d'accès à l'administrateur du serveur (ex : 4848) ce port permettra de modifier les paramètres du serveur. Il s'agit du même port administrateur que celui utilisé pour le domaine.
- <adresse> est l'adresse IP de la machine (ex: localhost, 138.96.192.120 ...).
- <PATHwar> est le chemin absolu du fichier .war

ATTENTION : Il est impossible de déployer le serveur si celui-ci est déjà déployé sur un autre domaine. En cas d'erreur il faut arrêter le serveur à l'aide de la commande suivante :

```
$ : undeploy --port 4848 --host localhost <nom_du_war>
```

- <nom_du_war> est le nom du war déployé (sans le .war).

3.1.4. Lancement du client gestionnaire des capteurs

Avant toute chose :

- Vérifier que la librairie Phidget en C est bien installée sur la machine cliente qui servira à transmettre les données capteurs vers le serveur.
- Connecter le Phidget possédant les capteurs sur cette machine par port USB.

Commencer tout d'abord par générer l'ensemble des exécutables sur votre machine, pour ce faire ouvrir un terminal dans le dossier du projet « hephaistos_application » et éxecuter la commande :

```
$ : make
```

Enfin dans ce même terminal, lancer le script qui va lancer deux terminaux externes correspondant respectivement au programme C qui récupère les événements du Phidget et au client Java qui transmet les données vers le serveur. Pour ce faire, exécuter la commande :

./execSensorClient localhost 8080

```
$ : ./execSensorClient <adresse> <port>
```

- <adresse> est l'adresse IP de la machine hébergeant le serveur (ou bien si un service d'hébergement en ligne est utilisé, le nom de domaine donné par celui-ci).
- <port> est le port de communication utilisé par le serveur (par défaut 8080).

3.1.5. Modification et maintenance du projet

Interface administrateur et modification des paramètres du serveur

Pour accéder à l'interface administrateur, il vous suffit d'ouvrir un navigateur web et d'entrer l'url : http:///<admin_port> (ex: Le serveur tourne en localhost et le port de l'administrateur est 4848 il suffit d'entrer l'adresse : http://localhost/4848). Une fois l'interface chargée il est possible, par exemple, de modifier le port 8080 de communication du serveur, qui est défini par défaut :

```
Configurations > Server-config > Network Config > http-listener-1
```

Modification du code source du projet

Pour modifier le Serveur, ouvrir le projet web dynamique 'hephaistos_project' dans Eclipse java EE. Le code source du serveur se situe dans la classe « HephaistosWebServer.java » du package « java Ressource/src/com.za.websocket/ ». Les classes « SensorMessage.java », « SensorMessageDecoder.java » et « SensorMessageEncoder.java » sont les classes qui permettent d'encoder et de décoder les messages reçus et émis par le serveur. Pour modifier l'application Web, ouvrir le projet « hephaistos_project » dans Eclipse java EE et se diriger dans le dossier « WebContent ». Celui-ci contient l'ensemble des pages html de l'application. De plus, les fichiers JavaScript associés se situent dans le dossier « WebContent/js » et les fichier CSS dans le dossier « WebContent/css » pour la mise en forme de l'application. Une fois toutes les modifications

effectuées sur le projet il faut générer le nouveaux .war du serveur pourra alors être déployé sur un domaine existant. Pour ce faire, ouvrir le projet « hephaistos_project » dans eclipse puis aller dans :

```
fichier > export > web > WAR file.
```

Enregistrer le .war dans le dossier souhaité.

Pour déployer le .war généré de façon manuelle suivre la méthode décrite dans la partie 3.1.3, b). Pour le déployer de façon automatique, copier ce nouveau .war du projet généré dans le dossier « hephaistos_application/glassfish4/glassfish/domains/hephaistosDomain/autodeploy »

Enfin, suivre la méthode décrite dans la partie 3.1.3, a) pour déployer le serveur automatiquement.

3.2. Technologies utilisées

3.2.1. Environnement de développement et matériel utilisé

Le projet a été développé sur une machine utilisant la distribution Fedora du système d'exploitation Linux. Des capteurs de proximité binaires, une carte Phidget, une Fit-Pc ainsi qu'un Rasberry Pi ont été utilisés pour effectuer des tests et vérifier le bon fonctionnement de l'application. Le produit a été développé sous l'IDE Eclipse java EE 7, qui est utilisée notamment pour des application de type « Dynamic Web Application » (application web dynamique). Firefox a été le navigateur Web qui a permis de tester et valider l'application web.

3.2.3. Langages de programmation

- ✔ **Java**
  - Le serveur a été développé en java. Il s'agit d'un serveur Glassfish 4.0 qui permet notamment l'utilisation de WebSocket(s) pour les communications. Le langage orienté objet a permis de structurer et d'organiser l'implémentation du serveur, notamment à travers les encodeurs et décodeurs de messages.
  - Le simulateur de capteur a également été développé en Java.
  - Le client local qui écoute les messages entrants pour stocker les messages émis par le serveur sur une base de données a été écrit en java.
- ✔ **C**
  - Le programme qui récupère les données brutes des capteurs connectés au Phidget et les transmet au Client java par socket.

- ✔ **Html-CSS-JavaScript**
  - Html a été utilisé pour créer les éléments présents sur la page web : images, texte, boutons...
  - CSS a été utilisé pour le design, le positionnement des éléments, les animations et l'esthétique de l'application Web.
  - JavaScript a été utilisé pour gérer la partie applicative et fonctionnelle de l'application Web

- ✔ **Shell**
  - Pour la gestion du projet avec Git (commit, pull, push, status...)
  - Pour compiler des programmes C et exécuter les scripts
  - Pour lancer le serveur manuellement dans le terminal
  - Pour construire le Makefile du projet

3.3.3. Protocole de communication

Le serveur échange des données à l'aide de WebSocket(s), qui utilisent un canal de communication full-duplex sur un socket TCP (Transmission Control Protocol). Cette communication bilatérale, contrairement au protocole HTTP, permet la notification au client d'un changement d'état du serveur ainsi que l'envoi de données (Push) du serveur vers le client (sans que ce dernier ait à effectuer une requête). De plus les WebSocket(s) sont compatibles avec presque tout les langages (C,C++,C#, Python, Java, JavaScript) et s'appuient sur des méthodes suivant une norme commune :

- OnOpen : Détecte une connexion
- OnClose : Détecte une déconnexion
- OnMessage : Détecte un message entrant
- OnError : Détecte les erreurs

3.3.4. Structure des messages transmis

Les messages envoyés par le serveur dans le cadre de cette application suivent le format JSON qui permet de structurer les informations. Les données sont organisées de la manière suivante :

{"idCapteur","acquisition","date","commande"}

- ✔ **idCapteur :** le champ idCapteur contient l'identifiant du capteur sous forme de chaîne de caractères (ex : capteur_IR)
- ✔ **acquisition :** le champ acquisition contient les données d'acquisition du capteur, que ce soit une valeur de type INT ou FLOAT simple ou un JSON si jamais il est nécessaire de transmettre plusieurs données (ex : vitesse, accélération, distance ...)
- ✔ **date :** la date système de l'acquisition sur forme de chaîne de caractère
- ✔ **commande :** une commande (qui est associée ou pas à un capteur) qui peut être envoyée puis exécutée par le serveur (ex : suprimmerCapteur, consulterCapteur, afficherCapteur , etc)

3.3. Présentation de l’application web

3.3.1. Portabilité de l’application

la visualisation en temps réel de données capteurs induit une contrainte majeure : une connexion internet permanente pour réceptionner les messages transmis par le serveur. Ainsi, il a donc été naturel de s’orienter vers un format d’application de type web. En effet, ce format rend l’application accessible sur tous les navigateurs compatibles avec HTML5 (pour l’utilisation de

WebSocket) comme Chrome, Firefox, Opera, Safari, internet Explorer ou encore Android Browser. Ainsi, l'application est accessible via tous les systèmes d'explorations sur ordinateur (Windows, Linux ou encore Mac OS) mais aussi sur smartphone ou tablette (Android, IOS, Windows Phone...). Toutes ces compatibilités lui procure une très bonne portabilité. De plus l'application ne nécessite aucun téléchargement ou installation car le script qui sera exécuté sur la machine de l'utilisateur est délivré par le serveur lors du téléchargement complet de la page web. Il est important de signaler que la conception et le design de l'application en CSS la rend compatible avec toutes les tailles d'écran. Ainsi l'application est parfaitement adaptée avec l'écran Android que possède l'équipe Hephaistos faisant 65'' muni d'un cœur android.

3.3.2. Lancement de l'application

Prérequis

Avant toute chose le serveur doit être lancé. Il peut-être hébergé sur n'importe quel machine disposant d'une connexion internet et dont le Pare-feu du routeur associé à cette connexion autorise le port de communication du serveur (par défaut 8080 pour les WebSockets). Le serveur pourrait également être déployé sur un service d'hébergement en ligne. Pour le lancement du serveur se référer à la section 3.1.3.

Pour faire fonctionner l'application correctement il faut recevoir des données capteurs. Ainsi, l'utilisateur doit au préalable installer les capteurs au sein de l'infrastructure désirée et les connectés au serveur. Ceux-ci doivent être connectés à des cartes Phidget elles-même connectées à une machine disposant d'une connexion Wifi (Rasberry Pi, Fit-Pc ou Ordinateur). Cette machine doit exécuter le programme Client (boucle infinie) afin de communiquer les données acquises par les capteurs avec le serveur en transmettant les messages sur une liaison WebSocket. Chaque capteur se voit attribuer un identifiant en fonction du port de connexion sur sa carte Phidget associée. L'utilisateur de l'application doit avoir connaissance de l'installation du réseau de capteurs afin de déterminer les emplacements de ceux-ci au sein de l'infrastructure. De plus, afin de pouvoir identifier les capteurs sur l'application, il doit avoir connaissance des numéros de port sur les Phidgets sur lequels sont connectés les capteurs. A noter que l'identifiant de chaque capteur est unique.

Remarque : Pour tester l'application web il est aussi possible de lancer le programme Client java qui simule un ensemble de capteurs. Pour se faire, ouvrir un terminal dans le dossier « hephaistos_application » du projet et entrer la commande suivante :

```
$ : java -jar SimulateurCapteurs.jar
```

Pour accéder à l'application, l'utilisateur peut la lancer en entrant son l'URL sur un navigateur internet (comme Chrome, Firefox, ou Internet Explorer). Celle-ci est de la forme http::://<adresseServer>/<port>/hephaistos_project/home.html. Avec :

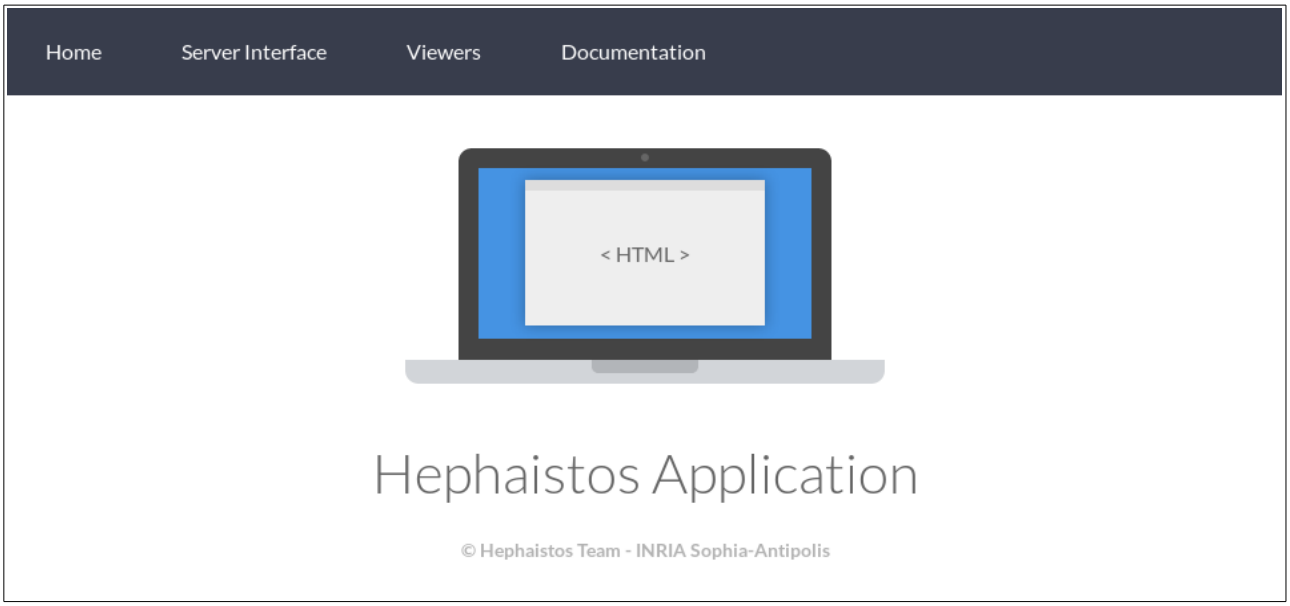
- <adresseServer> l'adresse du serveur qui peut-être donnée soit par l'adresse IP de la machine qui sert d'hébergeur au serveur, soit par le nom du domaine délivré par un hébergeur qui met a disposition ses services (comme OpenShift ou Heroku).
- <port> le port de communication du serveur (par défaut 8080 mais peut-être modifié dans l'interface administrateur)

3.3.3. Page d'accueil

L'adresse de l'application donne accès à sa page 'home'. Cette page possède une barre de menu située en haut de la page et composée de 4 sections : Home, Server Interface, Viewers, Documentation. Chacune d'entre elles possèdent leur propre rôle spécifique :

- **Home** donne accès à la page d'accueil de l'application.
- **Server Interface** est une interface qui permet de visualiser les messages transmis entre les différents clients et le serveur à travers une « Web Console ». De plus, elle permet de transmettre manuellement des messages vers le serveur pour simuler des capteurs.
- **Viewers** possède deux sous-sections **Map Sensor Editor** et **Real Time Chart** qui sont des visualisations en temps réel des données acquises par les capteurs.
- **Documentation** décrit l'application, son fonctionnement pour permettre de répondre aux questions de l'utilisateur sur la prise en main de l'application en cas de problème.

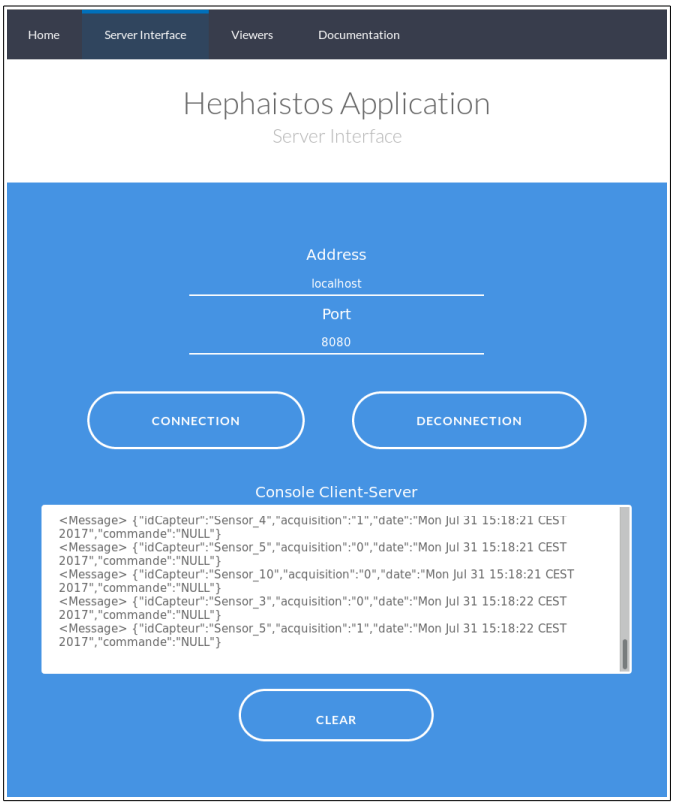
A terme la page d'accueil pourra, par exemple, être composée d'informations concernant le projet de l'équipe Hephaistos, les nouveautés ou encore d'afficher les résultats obtenus par le projet ...



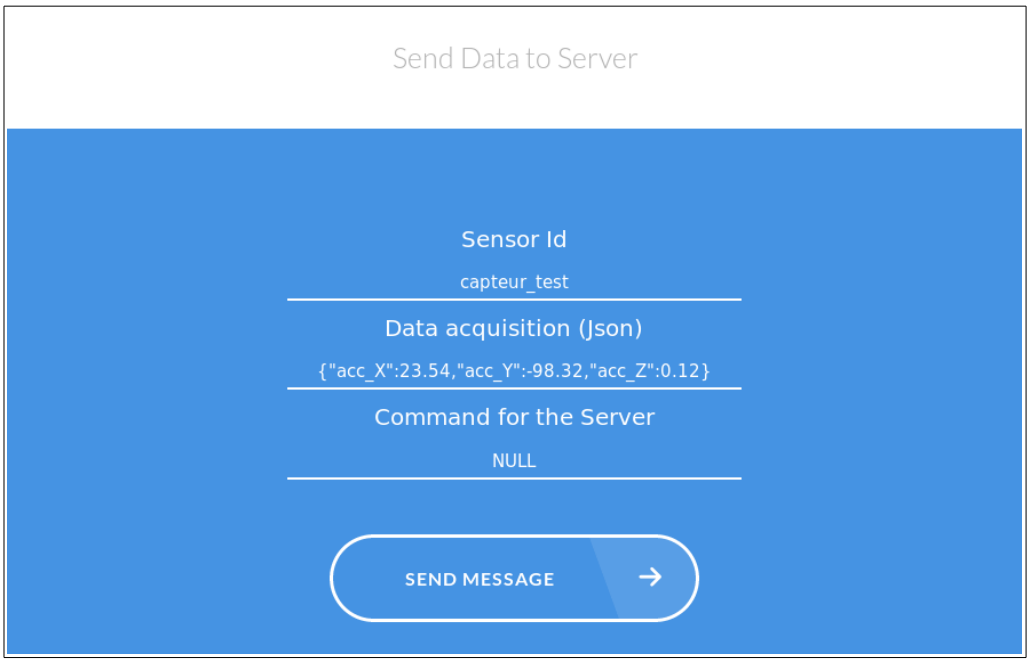
3.3.4. La web console du serveur

La section **Server Interface** est une vue qui permet de visualiser en temps réel les messages reçus par le serveur. Par défaut l'utilisateur n'est pas connecté au serveur. L'utilisateur peut, s'il le souhaite, établir une connexion WebSocket avec le serveur en entrant dans les champs correspondants l'adresse ainsi que le port de communication du serveur. A tout moment il est possible de se déconnecter du serveur à l'aide du boutton 'Deconnection' ainsi que de supprimer le contenu de la console à l'aide du boutton 'Clear'. Lorsque l'utilisateur envoie une requête de demande de connexion, le message : ---> Connection request s'affiche sur la console. En cas de connexion établie avec le serveur, le message suivant apparait : <--- Open connection with WebSocket 'ws://adresse:port/hephaistos_project/hephaistoswebserver' De même en cas de connexion impossible ou bien de connexion interrompue avec le serveur, le message x--- Disconnected of 'ws://localhost:8080/hephaistos_project/hephaistoswebserver' apparaît. A noter qu'un clic gauche de la souris sur la console augmente les dimensions du champ correspondant à la

console pour plus de lisibilité. Les messages sont affichés sur la console tels qu'ils sont reçus par les serveurs : sous le format JSON. De plus, les messages affichés sur la console sont l'ensemble des messages transmis par tous les clients connectés au serveur.

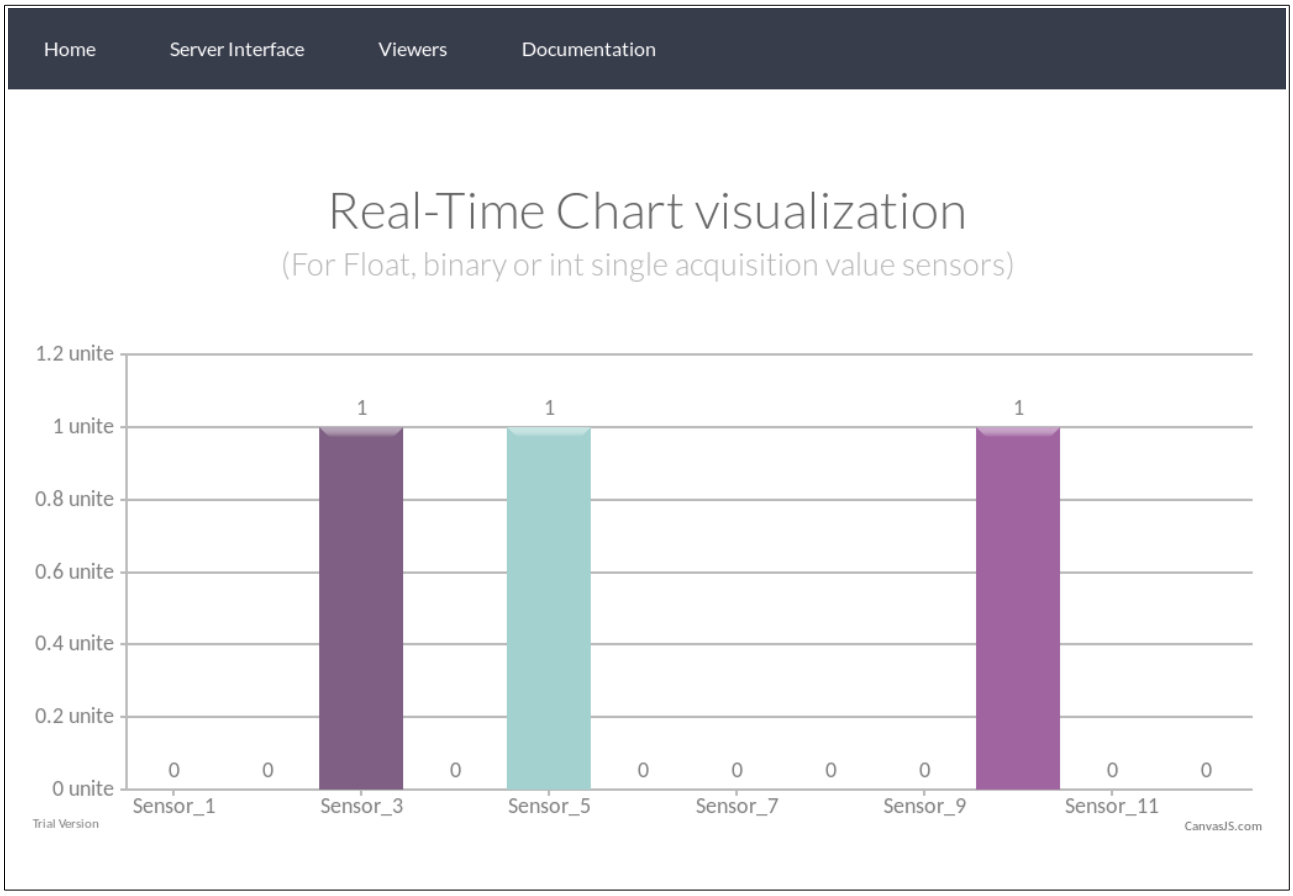


Comme vu précédemment l'utilisateur à la possibilité de transmettre un message vers le serveur en y entrant l'identifiant du capteur sous forme de chaîne de caractère, ainsi que l'acquisition sous forme de float, int ou encore de message structuré JSON en cas d'acquisition multiple (exemple d'un accéléromètre qui transmet l'accélération selon les 3 vecteurs spatiaux). Il a de plus la possibilité d'envoyer des instructions vers le serveur dans le champ **Command for the server** afin de décrocher un capteur du serveur (commande **deccrocheCapteur**) ou encore d'afficher les capteurs (commande **afficherCapteurs**) connectés au serveur.



3.3.5. Les visualisations en temps réel

Il est possible de visualiser en temps réel les données des capteurs qui sont connectés au serveur. Pour ce faire, l'utilisateur peut se rendre dans une des deux sous-sections de la section **Viewers,** à savoir **Real Time Chart** ou **Map Sensor Editor**. Real time chart permet de visualiser en temps réel des capteurs boolean, ou qui possèdent une acquisition unique (entière ou décimale). L'affichage est sous forme de diagramme en bâtons utilisant la librairie Canvas.js de JavaScript. Celui-ci se met à jour lorsqu'un capteur notifie le serveur d'un changement d'état. Cette vue permet à l'utilisateur de s'assurer du bon fonctionnement de l'application web ainsi que de la bonne transmission des données capteurs vers le serveur.



La seconde vue **Map Sensor Editor**, est le cœur du projet. En effet celle-ci permet à l'utilisateur, dans le cadre de ce projet, de visualiser en temps réel le nombre de personnes présentes dans chaque pièce de l'Institut Claude Pompidou (ICP). A noter que cette application peut-être étendue à n'importe quelle autre infrastructure. Cette vue est composée d'un panel (à gauche) qui permet la connexion du client de l'application au serveur, d'un environnement de travail central qui permet de visualiser la carte des capteurs en temps réel, un panel (en bas) pour afficher les coordonnées de la souris sur l'environnement de travail ainsi que d'un panel (à droite) qui permet d'éditer l'environnement.

1) Chargement de la carte

L'utilisateur charge la représentation d'un plan de bâtiment qu'il veut visualiser en chargeant l'image associée (format jpg, jpeg, jpeg2000, png, svg ...).

2) Connexion au serveur

L'utilisateur établit un lien WebSocket avec le serveur à l'aide du panel de gauche en entrant l'adresse du serveur ainsi que le numéro de port utilisé par celui-ci. La connexion n'a pas été automatisée car l'adresse du serveur ainsi que le port utilisé peuvent varier en fonction du lieu de l'infrastructure où sont déployés les capteurs. On peut voir les capteurs connectés au serveur dans le panel de gauche, dans la liste déroulante 'Sensor Selected' de la section 'Add sensor'.

3) Ajout de capteurs

En connaissant l'emplacement des capteurs au sein de l'ICP ainsi que les numéros de ports sur les Phidget où sont connectés les capteurs, l'utilisateur peut positionner les capteurs dynamiquement sur la carte de l'application. Pour ce faire il faut se diriger dans la section 'Add sensor' du panel de droite, sélectionner un capteur dans la liste déroulante 'Sensor Selected', appuyer sur le bouton 'Add' et sélectionner deux points présents sur la carte. Avant l'ajout du premier point, l'application dessine dynamiquement un point aux cordonnées courantes de la souris survolant la carte. Enfin, l'application va tracer une ligne dynamiquement entre le premier point et la position courante de la souris sur la carte afin de matérialiser l'emplacement du capteur. A tout moment, l'utilisateur peut annuler la procédure d'ajout à l'aide du bouton 'Cancel' de cette même section. Une fois ces deux points sélectionnés, l'application va tracer une ligne entre ceux-ci puis va afficher (au niveau du barycentre des deux points) l'identifiant du capteur. A chaque fois qu'un capteur est ajouté sur la carte, les informations le concernant (identifiant, position des deux points sur la carte, valeur d'acquisition courante) sont ajoutés dans la liste des capteurs présents sur la carte SensorOnMap[ ] au format JSON. A noter que l'ajout de capteurs n'est possible que si des capteurs sont connecté au serveur. De plus, l'ajout est sécurisé par des messages d'alerte (ex : image

non chargée, aucun capteur connecté au serveur) afin de protéger l'application d'éventuelles erreurs.

4) Ajout de Zones

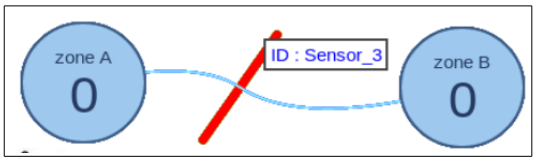
Une zone est définie par une ou plusieurs pièces voisines situées au sein de l'infrastructure. Les zones sont séparées entre elles par des murs et des capteurs. Ainsi, en fonction de l'emplacement des capteurs sur la carte, des zones sont induites. Une zone est caractérisée par un nom (unique), un rayon représentatif de sa taille sur la carte, ainsi que du nombre courant de personnes présentes. L'utilisateur a la possibilité d'ajouter des zones manuellement sur la carte. Pour ce faire, il se rend dans la section 'Add area' du panel de droite, définit un nom pour la zone, un rayon ainsi qu'un nombre de personnes initialement présentes (par défaut 0) , appuie sur le bouton 'Add' et sélectionne un point sur la carte. Dès lors que l'utilisateur appuie sur le bouton 'Add' , un simple survol de la carte avec la souris permet de visualiser dynamiquement par un cercle les dimensions et la position de la zone qui va être ajoutée. A tout moment l'utilisateur peut annuler la procédure d'ajout à l'aide du bouton 'Cancel' de cette même section. A chaque fois qu'une zone est ajoutée sur la carte, les informations la concernant (nom, position, nombre de personnes courantes, rayon) sont ajoutées dans la liste des zones présentes sur la carte zone[ ] au format JSON. De plus, lors de l'ajout de la zone, un cercle de rayon égal à la valeur entrée dans le champs 'Ray' est dessiné avec son nom et son nombre de personnes présentes actuellement. A noter que l'ajout de zones est sécurisé par des messages d'alerte (ex : nom déjà utilisé, image non chargée) afin de protéger l'application d'éventuelles erreurs.

5) Création des liaisons et génération du graphe

Comme vu précédemment, une zone peut être entourée de plusieurs capteurs, chaque capteur sépare deux zones voisines. On peut donc modéliser ces relations par un graphe où les sommets sont les zones et les arrêtes sont caractérisées par un lien entre deux zones et qui traversent un capteur. Ainsi on modélise ce graphe en machine par 2 listes :

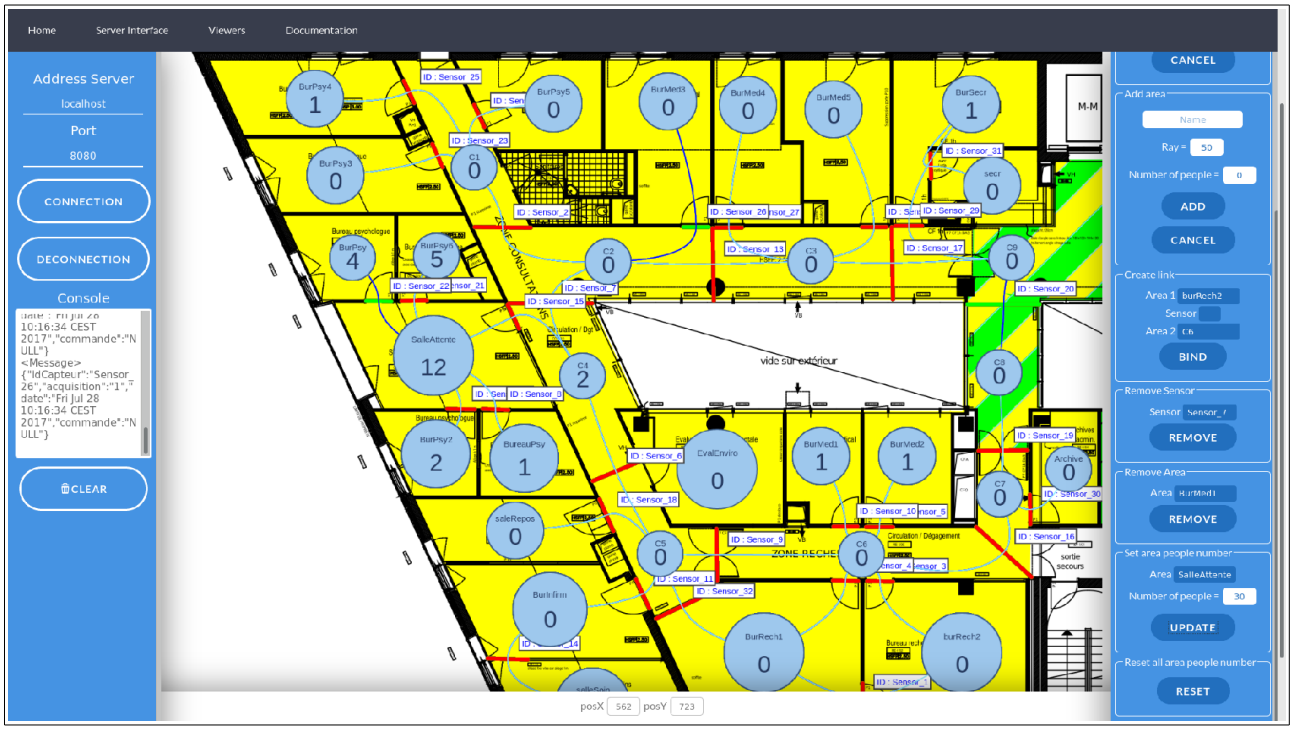
- zone[ ] représente les zones présentes sur la carte (Sommet du graphe). Il s'agit de la même liste que celle vue précédemment.
- listeArete[ ] représente la liste des arêtes du graphe. Une arête est composée des deux zones voisines ainsi que du capteur les séparant.

Pour créer la liaison, l'utilisateur doit se rendre dans la section 'Create link' du panel de droite. Il sélectionne deux zones distinctes présentes sur la carte dans les liste déroulante 'Area 1' et 'Area 2' ainsi que le capteur qui sert de séparateur. En cliquant sur le bouton 'Bind' la liaison se créée et apparaît sur la carte sous forme de courbe de Bezier allant des deux zones choisies et passant pas le capteur. A noter que la création des liaisons est sécurisée par des messages d'alerte (ex : nom des deux zones non distinctes, image non chargée, élément manquant à la création de l'arrête) afin de protéger l'application d'éventuelles erreurs.



6) Visualisation en temps réel

Les capteurs connectés au serveur vont transmettre les valeurs d'acquisition au serveur grâce aux WebSocket, ce qui va permettre une transmission quasi instantanée. Ainsi, si les capteurs sont ajoutés sur la carte, ils apparaissent en rouge si la valeur d'acquisition est nulle (personne ne traverse la ligne correspondant au capteur) et en vert lorsqu'un passage est détecté. Ainsi on peut apercevoir, à chaque changement d'état des capteurs un changement de couleur. De plus, pour chaque capteur connecté à des zones voisines, un algorithme de calcul va pouvoir estimer avec plus ou moins de précision le sens de traversée et donc de mettre à jour le nombre de personnes présentes dans ces zones voisines après activation de celui-ci. Actuellement l'algorithme utilisé est un algorithme heuristique qui permet uniquement de faire une démonstration de l'application. L'algorithme qui sera utilisé à l'ICP est en cours de développement par les membres de l'équipe Hephaistos et pourra par la suite se substituer à l'algorithme heuristique utilisé temporairement dans le cadre de ce projet.



7) Éditer le graphe courant

A tout moment l'utilisateur peut éditer dynamiquement le graphe sans interrompre la visualisation en temps réel. A chaque modification l'environnement de travail est rafraîchi en étant redessiner. L'utilisateur dispose de quatre fonctionnalités pour éditer le graphe :

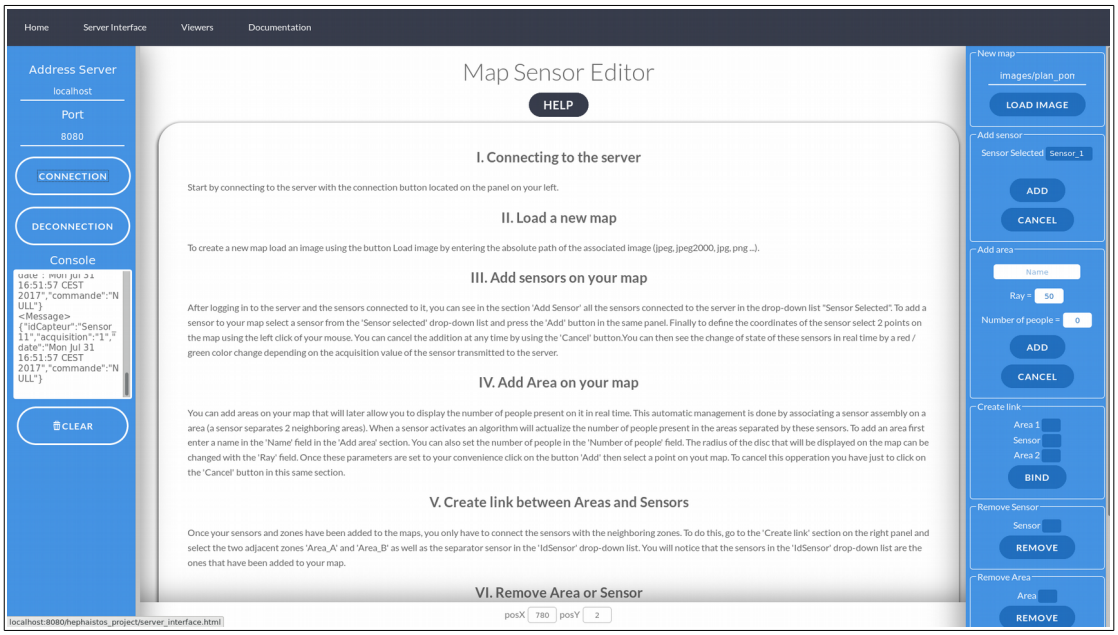
- **Supprimer des capteurs :** l'utilisateur peut supprimer des capteurs en se rendant dans la section 'Remove Sensor'. Pour ce faire, celui-ci sélectionne un capteur présent sur la carte dans la liste déroulante 'Sensor' puis en cliquant sur le bouton 'Remove' le capteur sera supprimé ainsi que, si celle-ci existe, l'arrête le traversant.
- **Supprimer des Zones :** l'utilisateur peut supprimer des zones en se rendant dans la section 'Remove Area'. Pour ce faire, celui-ci sélectionne une zone présente sur la carte dans la liste

déroulante ‘Area’ puis en cliquant sur le bouton ‘Remove’ la zone sera supprimée ainsi que toute les liaisons qu’elle possède.

- **Modifier le nombre de personnes d‘une zone :** l’utilisateur peut modifier le nombre de personne présent actuellement dans une zone en se rendant dans la section ‘Set area people number’ et en sélectionnant la zone souhaitée ainsi que le nombre de personne à définir.
- **Réinitialiser le graphe :** l’utilisateur peut remettre à 0 le nombre de personnes présentes dans l’ensemble des zones constituants les sommets du graphe en allant dans la section ‘Reset all area people number’ avec le bouton ‘Reset’. Ceci pourrait-être utilisé, par exemple, à l’heure où ferme l’ICP lorsqu’il n’y a plus personne au sein de l’Institut.

8) Consulter l’aide

l’utilisateur à la possibilité de consulter une aide en cas d’incompréhension des méthodes d’édition de l’environnement de travail ou bien la logique des étapes à suivre ainsi que l’utilité et le but de l’application. Pour ce faire il peut cliquer sur le bouton ‘Help’ situé en haut de l’environnement de travail. Lorsque l’aide apparaît, l’environnement de travail disparaît pour laisser place à l’aide. Un second clic sur ‘Help’ permet à l’utilisateur de quitter l’aide.



3.3.6. Documentation de l’application

La documentation de l’application est disponible sur la vue **Documentation** de l’application. Celle-ci décrit l’ensemble de l’architecture de son installation à son fonctionnement. Il s’agit de la même documentation présente dans le README.md du projet.

3.4. Démarche, méthodes de travail et suivi du projet

3.4.1. Suivi du projet

Le développement et le suivi du projet a été assuré par Github, un web hébergeur qui gère la gestion et le suivi de développement de logiciels. Celui-ci m'a permis de mettre à jour le projet au fur et à mesure de son avancement. Cela permet également d'avoir plus de sécurité au niveau du stockage du projet en l'ayant à disposition sur une plateforme extérieure. De plus, ce service permet de pouvoir télécharger le projet à partir de n'importe quelle machine disposant d'un accès internet. Un journal de bord journalier a été régulièrement mis à jour pour archiver les tâches qui ont été effectuées afin de suivre le bon avancement du projet et son évolution tout au long du stage. Régulièrement une démonstration de l'avancement du projet était reporté à mon responsable de stage.

3.4.2. Démarche et méthodes de travail

Semaine 1	✔ Prise en main du matériel : installations nécessaires : (eclipse,java EE, librairie ..) ✔ Documentation sur l'architecture client/serveur et sockets. ✔ implémentation complète de l'architecture client/serveur en localhost, ✔ Test de l'architecture sur deux terminaux différents pour vérifier la bonne communication entre ceux-ci.
Semaine 2	✔ Problème : cette architecture ne permet pas les échanges entre appareils externes (réseau web) mais uniquement sur le réseau local. De plus, les échanges sont unilatéraux : lorsque le client fait une requête vers le serveur, celui-ci lui répond. ✔ Nouvelle piste de recherche : les WebSockets et le protocole WAMP qui permettent des échanges bilatéraux (le serveur peut notifier le client de lui-même) ✔ Documentation et recherche sur cette nouvelle piste de travail.
Semaine 3	✔ Implémentation d'une architecture WebServeur/WebSockets en java EE , serveur de type GlassFish : communication entre client (sur page html) avec le serveur en localhost. ✔ Recherche d'hébergeur gratuit pour héberger le serveur (Heroku, Openshift). ✔ Piste de recherche pour l'affichage des données capteur sur l'application : D3.js et Canvas.js.
Semaine 4	✔ Documentation sur le langage html et JavaScript. ✔ Implémentation du code html et JavaScript l'application Web pour visualiser les données émises par le serveur à travers la vue « Server Interface ». ✔ Implémentation de la vue « real time chart » de l'application web.
Semaine 5	✔ Installation de l'architecture sur un Rasberry Pi du serveur. ✔ Création du Client Java qui transmet les données vers le serveur. ✔ Création du programme C qui s'appuie sur la librairie Phidget pour récupérer les données capteurs, structure les données et les transmet par Socket au Client Java précédent.

	✔ Test de l’architecture avec un capteur sur une carte Phidget connectée au Rasberry Pi : Fonctionnel, cependant temps d’envoi de l’acquisition du capteur vers le serveur trop long (12 secondes sur le Rasberry Pi).
Semaine 6	✔ Optimisation de la procédure d’envoi du message du capteur vers le serveur : le temps d’exécution passe de 12 à moins de 1/10 seconde sur le Rasberry Pi. ✔ Documentation sur CSS pour la mise en forme de l’application web ✔ Amélioration de l’interface de l’application web avec CSS.
Semaine 7	✔ Ajout d’une page d’accueil, d’un menu et d’une documentation pour l’application Web. ✔ Ajout de la fonctionnalité de suppression dynamique de capteurs et zones dans la vue « Sensor Map Editor » de l’application web. ✔ Ajout des fonctionnalités de mise à jour et de remise à zéro des zones de façon dynamique dans la vue « Sensor Map Editor » de l’application web.
Semaine 8	✔ Rédaction de la documentation du projet pour l’équipe Hephaistos : installation, description et maintenance de l’architecture et utilisation de l’application Web. ✔ Rédaction du rapport de stage.

4. Bilan du stage

4.1. Résultats obtenus

Suite à des restrictions sur les ports de communication (Pare-Feu) au sein du réseau de l’Inria, le serveur déployé sur une des machine de l’Inria n’est pas accessible depuis l’extérieur de l’Inria. Cependant l’architecture fut testé sur le réseau local et est parfaitement fonctionnel en localhost. De plus, elle a été testé sur un réseau wifi et machine personnel avec un Pare-Feu autorisant les communications sur le port 8080 : tout est fonctionnel. Ainsi, pour étendre l’application sur le réseau web il faudrait que le service informatique de l’Inria autorise les communications entrantes et sortantes de ce port 8080. Cela permettrait un transfert de messages par liaisons WebSocket entre une machine distante de l’Inria (ex : la machine de l’ICP qui transmet les données capteurs) et le serveur qui sera hébergé par une des machines de l’équipe Hephaistos. De plus, le prototype d’application de visualisation en temps réel est parfaitement fonctionnel, aucun beug apparent n’a été signalé. De plus après plusieurs tests et prise en main par différents membres de l’équipe Hephaistos il a été rapporter que sa prise en main est très intuitive et bien documenté. Il ne reste donc plus qu’à attendre le développement complet de l’algorithme de mise à jour du nombre de personnes présentes dans chacune des pièces à chaque acquisition capteur envoyé au serveur, par les membres de l’équipe Hephaistos. Cet algorithme pourra alors être intégré à l’application Web et remplacer l’ heuristique utilisé jusque-là dans le cadre de ce projet.

4.2. Difficultés rencontrées et solutions apportées

Une des difficultés principales fut la restriction dûe aux normes de sécurité sur le réseau de l'Inria qui posa des notamment des problèmes sur les tests de vérification du bon fonctionnement de l'architecture Client/Serveur développée. Ainsi pour pallier à ce problème, le serveur GlassFish implémenté au centre de l'architecture fut hébergé sur Un Rasberry Pi puis un Fit Pc connecté à un Phidget lui-même lié à des capteurs. Ceci me permit de tester le bon fonctionnement de l'application dans sa globalité : de l'acquisition des données capteurs à la visualisation sur l'application Web. La seconde difficulté majeure fut le temps de transfert des données capteurs connectés au Rasberry Pi vers le serveur qui était de 12 secondes environ. En effet à chaque notification des capteurs vers le Phidget, le programme C qui gérait ces événements exécutait un jar exécutable à travers une commande système. Celui-ci transmettait les données par liaison WebSocket vers le serveur. Pour résoudre ce problème le programme C notifie un Client java en boucle infinie par Socket du changement d'état d'un des capteurs. Ainsi, le lancement du jar ne se faisait qu'au lancement de la boucle infinie, ce qui remmena à moins d'une 1/10 seconde le temps de transfert vers le serveur.

4.3. Amélioration envisageable du produit

La vue de l'application de visualisation en temps réel du nombre de personnes « Sensor Map Editor » pourrait-être amélioré avec notamment une fonctionnalité d'enregistrement de l'environnement de travail : enregistrer la carte, les capteurs et les zones ajoutés ainsi que les liaisons entre ceux-ci. Pour ce faire, il suffirait d'enregistrer dans un fichier le chemin de l'image associée à la carte et enregistrer les listes des capteurs, zones et liaisons sous forme de chaîne de caractère. Enfin on pourrait ajouter une fonctionnalité qui permettrait de charger d'anciennes cartes existantes avec ses capteurs et zones associés.

4.4. Bilan personnel

Ce projet fut très intéressant du point de vue professionnels car il m'a permis de développer des compétences dans les technologies du web à travers la découverte de langages tel que javaScript, html et css. Mais également la découverte du fonctionnement de l'architecture client-serveur avec ses protocoles (TCP/IP et WebSocket dans le cadre de ce projet) et ses méthodes propres. De plus il m'a permis d'avoir une première expérience sur le développement d'un projet de 2 mois. Il m'aura permis de mieux comprendre le fonctionnement d'un institut de recherche. Du point de vue de la communication, il m'aura permis d'avoir les premières relations professionnelles qui sont essentiel pour un ingénieur.

5. Bibliographie

Les bases sur les Sockets & communiquer sous stockets en Java:

- https://openclassrooms.com/courses/introduction-aux-sockets-1
- http://www.chicoree.fr/w/Communication_par_socket_sous_Java

Créer un serveur en java :

- http://gfx.developpez.com/tutoriel/java/network/

Créer un serveur web sur le terminal :

- https://openclassrooms.com/courses/un-serveur-d-hebergement-multiutilisateur-sous-linux

Envoie de fichier vers un serveur :

- https://openclassrooms.com/courses/creez-votre-application-web-avec-java-ee/formulaires-l-envoi-de-fichiers

Téléchargement d'un fichier depuis un serveur :

-https://openclassrooms.com/courses/creez-votre-application-web-avec-java-ee/le-telechargement-de-fichiers

websocket API pour les applications Web :

- https://www.youtube.com/watch?v=yRvweIRws7o

Websocket chat java :

- https://angelozerr.wordpress.com/2011/07/26/websockets_jetty_step3/

Websocket java + page internet :

- https://www.jmdoudoux.fr/java/dej/chap-api_websocket.htm

Wabsocket + application web pour placer des points sur un tableau blanc

- https://www.youtube.com/watch?v=yRvweIRws7o

Websockets envoie de donnée en direct sur une page web

- https://www.youtube.com/watch?v=D_0yUwr5pcU

Dynamic charts :

- http://canvasjs.com/html5-javascript-dynamic-chart/

Tuto websocket :

- https://www.xul.fr/html5/websocket.php
- http://sii-rennes.developpez.com/articles/un-chat-en-html5-avec-les-websockets/

Tuto websocket javascript :

- https://developer.mozilla.org/fr/docs/WebSockets/Writing_WebSocket_client_applications

Chat websocket :

- https://www.youtube.com/watch?v=4nWZ2Oog_w8

tuto webSocket toutes les étapes :

- http://www.zaneacademy.com/

Déployer GlassFish sur OpenShift :

- https://blog.openshift.com/running-java-apps-in-the-cloud-with-glassfish-and-a-paas/
- https://github.com/shekhargulati/glassfish4-openshift-quickstart
- http://svanimpe.be/blog/glassfish-openshift.html
- http://www.vancura.cz/?p=537
- https://github.com/shekhargulati/glassfish4-openshift-quickstart

Migration de GlassFish vers WildFly (JBoss)

- http://wildfly.org/news/2014/02/06/GlassFish-to-WildFly-migration/

GlassFish hebergé sur RedHat :

- http://multikoop.blogspot.fr/2012/09/adf-essentials-in-redhat-cloud.html

Node.js chat app Openshift :

- https://www.youtube.com/watch?v=RAP50adrFN8

Android app chat openshift websocket :

- https://www.youtube.com/watch?v=a3aChxGpbDg
- https://www.youtube.com/watch?v=w5E_ZbJEuhw

6. Annexe