TP Chaîne d'acquisition – 6 – IR
Programmation Orientée Objet NodeJS

Objectif :
Dans le TP précédent, un nano-ordinateur sous Raspberry OS (Linux) doit recevoir des informations issues d'un système externe (Arduino) via sa liaison série, traiter ses informations pour les afficher en HTML ou les enregistrer dans une base de données.
Le code créé ne respect pas complètement la philosophie OBJET. Ce TP va nous aider à nous améliorer en Programmation Orientée Objet (POO)

On vous fournit :

- Une platine Arduino avec un logiciel embarqué qui génère des trames GPS, ou qui est capable de recevoir des commandes/
- Un nano-ordinateur Raspberry et sa carte SD préchargée.
- Un câble de liaison USB

Pré-requis : Le TP Chaine d'acquisition 1 et 3-IR

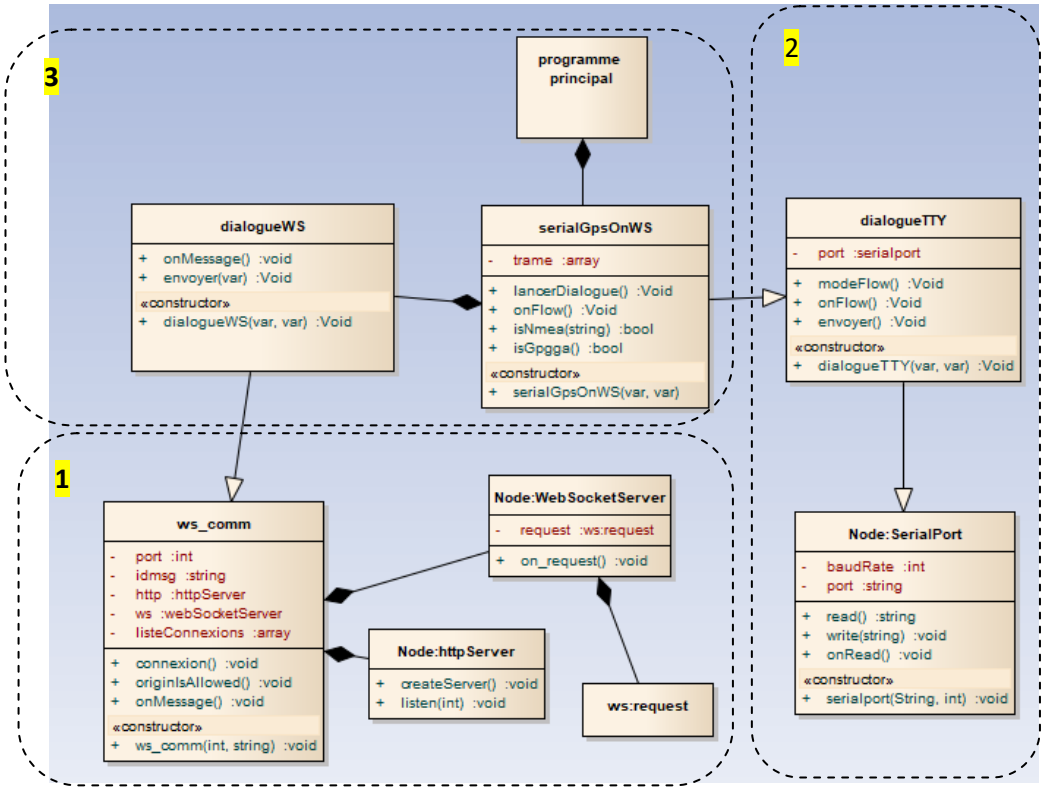
Sommaire

1 ANALYSE (A ETUDIER)

Dans le TP3, nous avons utilisé tel quel un exemple fourni avec la documentation WebSocket / NodeJS. Côté analyse, la répartition du code « tord » un peu le modèle objet.

Dans ce TP, nous allons essayer de structurer tout ça, toujours dans le but de réutiliser le code dans d'autres projets.

Voici une proposition d'analyse :



Bloc 1 : Le socle commun à tous les projets WebSocket

La classe ws_comm utilise les bibliothèques Node JS pour fournir une base exploitable. Il suffit de redéfinir la méthode « onMessage ».

Bloc 2 : Le socle commun à tous les projets Liaison Série

La classe « dialogueTTY » ajoute au port série la gestion du « flot de ligne », c'est-à-dire la génération d'un événement à chaque réception se terminant par le caractère « fin de ligne ». C'est une amélioration de la méthode « onRead ».
Si vous lisez le code fourni, ce fonctionnement est obtenu par l'utilisation des objets « parser-readline » et « pipe » de la bibliothèque « serialport ».

Bloc 3 : L'application proprement dite - à développer

La classe « dialogueWS » est prête à l'emploi si on doit simplement envoyer les informations aux clients WebSocket.
Il faut modifier sa méthode « onMessage » si l'application doit recevoir des informations des clients WebSocket.
Note : Cette classe traite TOUS les clients de la même façon. Elle n'est pas prévue pour faire un dialogue individuel différent avec chaque client connecté.

2 DE L'OBJET AVEC NODEJS : EXPLICATIONS (A ETUDIER)

2.1 Constructeur

La structure de classe est la même qu'avec le Javascript client :
Exemple :

```
class ws_comm {
	constructor (port, idmsg) {
		// Des attributs :
		this.port = port;
		this.idmsg = idmsg;
		this.listeConnexions = []; // Liste des clients

	}

	connexion() { // Une méthode

			// code...

	}
};
```

L'objet s'initialise comme suit : `var toto = new ws_com(8080, "idmsg:");`
L'objet s'utilise comme suit : `toto.connexion();`

2.2 Créer une classe fille (héritage)

Par exemple, la classe « dialogueTTY » hérite de la classe SerialPort :

```
class dialogueTTY extends SerialPort { … } ;
```

2.3 Constructeur de classe fille : super ?

Si on doit appeler le constructeur de la classe mère, on utilise le mot clé « **super** » dans le constructeur de la classe fille.

Cas simple d'un constructeur avec arguments :

```
constructor (port, baud){
		super(port, baud);
}
```

Cas d'un constructeur qui retourne une valeur (Ex : l'objet « serialport » de NodeJS) :

```
constructor (port, baud){
		var p = super(port, {baudRate: baud});
		this.port = p;
	}
```

2.4 Rendre une classe importable : module.export

On crée généralement les classes dans des fichiers séparés. Il faut ensuite les inclure là où ils sont utilisés. Pour cela on utilise la technique du « module export », ajouté à la fin de la définition de la classe.

```
class ws_comm {
	constructor (port, idmsg) { }

	connexion() {}
};

module.exports = ws_comm;
```

2.5 Importer des classes externes : require

Ensuite, pour utiliser une classe contenue dans un fichier externe, on utilise le mot clé « **require** » :

Exemples :

```
const gps = require('./serialGpsOnWS.js');// Bibliothèque personnelle

const SerialPort = require('serialport'); // Bibliothèques NodeJs
const Readline = require('@serialport/parser-readline');
```

Pour créer un objet de cette classe importée, on écrit :

```
var tty = new gps('/dev/ttyUSB0', 4800);
```

ou dans le cas d'une classe héritée :

```
class dialogueTTY extends SerialPort {} ;
```

3 AU TRAVAIL !

Réalisez la classe « **serialGpsOnWS** » qui fonctionnera avec le programme « serveur.js » fourni.

Cette classe hérite de « dialogueTTY »
Elle redéfinit la méthode « onFlow » de « dialogueTTY ». Il suffit de réécrire une méthode avec le même nom, elle sera prioritaire sur la méthode de la classe mère.

3.1 Le constructeur

Il appelle le constructeur de la classe mère « dialogueTTY » et lui transmet le nome du port série et la vitesse de transmission.
Il initialise un objet de type « dialogueWS »
Il crée un attribut « trame », tableau vide : `this.trame = []`

3.2 Ce que fait la méthode « onFlow(ligne) »

La méthode « onFlow » est un « callback », c'est-à-dire une fonction exécutée à chaque fois qu'un évènement particulier se produit.
Ici l'événement est : « le port série a reçu une ligne de caractère ».
La variable « ligne » contient les caractères reçus.

1. Il faut tester si c'est une trame nmea (doit commencer par « $ »). La méthode « isNmea » doit faire ce test.
2. Si oui, il faut « éclater » la ligne avec la virgule comme séparateur
   (utiliser la méthode « split » des chaines de caractères JavaScript
   https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Global_Objects/String/split)
3. Si la trame est bien une $GPGGA (c'est la méthode « isNmea » qui fait ce test), on la convertit en trame JSON (avec `JSON.stringify`)
4. Et on l'envoie aux clients connectés par WebSocket (méthode « envoyer » de l'objet « dialogueWS » créé dans le constructeur).

3.3 La méthode « lancerDialogue »

Cette méthode « démarre » le système de réception du flot de données par le port série :
Très simplement, elle appelle la méthode « modeFlow » de la classe mère.

3.4 Pour tester

Démarrez le serveur créé : `node serveur.js`
Ouvrez un navigateur qui accède à la page « `index.html` » fournie.
La latitude GPS devrait apparaitre et se mettre à jour.
Ajoutez la longitude et l'altitude.