Architecture et fonctionnalités du client NMEA

Ref : TV240114_TU_SM

Rédacteur : Serge Morvan

Les connexions au serveur

Les différents périphériques et capteurs fournissent les données de navigation au serveur, suivant différents protocoles. Ces données sont multiplexées par le serveur. Sur requête des clients elles sont analysées, transformées dans un format XML standard et envoyées aux différents clients. Ces clients peuvent être sur la même machine dans le cas le plus simple ou sur d'autres ordinateurs, tablettes ou smartphones. NaVisu est le client le plus riche.

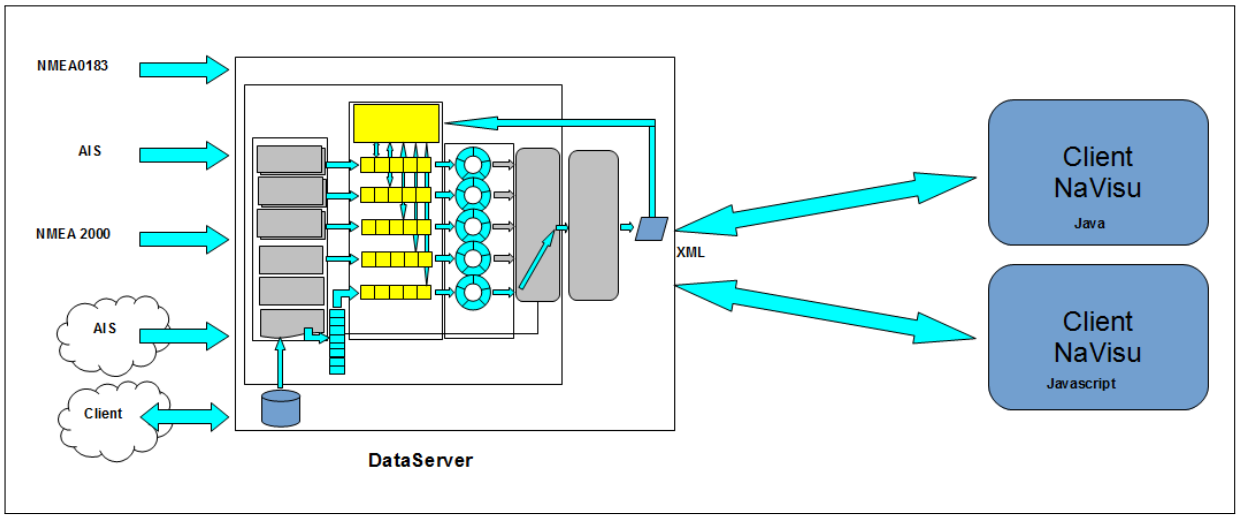

Figure 1 – *Schéma général des entrées/sorties*

Les données en entrée

Il n'existe pas, à notre connaissance, de schéma XML bien défini pour l'ensemble des données NMEA. A l'exception de quelques cas particuliers comme : GPX [1] ou AIS [2] par exemple. Nous

1. http://www.topografix.com/gpx.asp
2. http://www.thsoa.org/hy07/04P_10.pdf

avons donc défini notre propre modèle NMEA [3] associé à un schéma nmea.xsd structurant les données sans ambiguïtés et permettant leur validation.

Extrait du schéma `nmea.xsd`

```
<xs:complexType name="gga">
  <xs:sequence>
    <xs:element name="device" type="xs:string" minOccurs="0"/>
    <xs:element name="sentence" type="xs:string" minOccurs="0"/>
    <xs:element name="utc" type="xs:dateTime" minOccurs="0"/>
    <xs:element name="latitude" type="xs:float"/>
    <xs:element name="longitude" type="xs:float"/>
    <xs:element name="gpsQualityIndicator" type="xs:int"/>
    <xs:element name="numberOfSatellitesInView" type="xs:int"/>
    <xs:element name="horizontalDilutionOfPrecision" type="xs:float"/>
    <xs:element name="antennaAltitude" type="xs:float"/>
    <xs:element name="unitsOfAntennaAltitude" type="xs:string" minOccurs="0"/>
    <xs:element name="geoidAltitude" type="xs:float"/>
    <xs:element name="unitsOfGeoidAltitude" type="xs:string" minOccurs="0"/>
  </xs:sequence>
</xs:complexType>
```

Extrait d'un fichier de données conforme au schéma `nmea.xsd`

```
<?xml version="1.0" encoding="UTF-8" standalone="true"?>
-<sentences>
  -<gga>
    <device>GP</device>
    <sentence>
      $GPGGA,191138.000,4826.2632,N,00429.8614,W,2,05,1.2,72.4,M,52.1,M,0.8,0000*50
    </sentence>
    <utc>2014-01-05T19:11:38.973+01:00</utc>
    <latitude>48.43772</latitude>
    <longitude>-4.4976897</longitude>
    <gpsQualityIndicator>2</gpsQualityIndicator>
    <numberOfSatellitesInView>5</numberOfSatellitesInView>
    <horizontalDilutionOfPrecision>1.2</horizontalDilutionOfPrecision>
    <antennaAltitude>72.4</antennaAltitude>
    <unitsOfAntennaAltitude>M</unitsOfAntennaAltitude>
    <geoidAltitude>52.1</geoidAltitude>
    <unitsOfGeoidAltitude>M</unitsOfGeoidAltitude>
 </gga>
</sentences>
```

3. Voir article Modèle NMEA

Le projet `NaVisuSimpleClient`

Le projet NaVisuest divisé en sous-projets, chaque sous-projet possède une architecture type composants. La plupart des sous-projets sont développés sous forme d'API, ayant le minimun de dépendance avec les autres sous-projets. C'est le cas de l'API `navisu-client`, qui ne dépend que du module `navisu-domain` : description des modèles d'objets utilisés. Il est donc simple de présenter le serveur sous forme d'un projet indépendant : `NaVisuSimpleClient`

Téléchargement

https://github.com/terre-virtuelle/NaVisu-client.git

Paramétrisation

l'API NaVisu-client ne fourni pas d'IHM, pour la paramétrisation, celle ci se fait à l'aide du fichier de propriétés : `properties/client.properties`

```
# Web client parameters
hostName = localhost
port = 8080
period = 1000
```

La variable `port` correspondant au numéro du port de communication avec les clients, attention de n'avoir pas déjà des applications utilisant ce port.

Lancement

A partir du fichier jar : `java -jar NaVisuSimpleClient.jar`

Développement

Le serveur fourni les données à l'aide du protocole WebSocket, le client doit donc se conformer à ce protocole. La propriété period permet de modifier la fréquence d'acquisition des données sur le client. Le protocole WebSocket permet des actions en pull ou en push. Nous avons fait le choix de faire l'acquisition uniquement en mode pull. Le client décide d'envoyer une requête : `nmea` au serveur qui lui renvoie un paquet de données. Entre deux requêtes les données arrivées sur le serveur peuvent être perdues. Pour le client NaVisu, présenté ici nous avons choisi d'utiliser le framework `Vert.x` , comme pour le serveur, mais tout autre solution est possible. [4]

4. http://vertx.io/

La classe de test est : `bzh.terrevirtuelle.navisu.client.app.ClientMain`

```
ComponentManager componentManager = ComponentManager.componentManager;
// deploy components
LOGGER.log(Level.INFO, "\n Start", componentManager.startApplication(
        NmeaClientImpl.class
));
NmeaClientServices nmeaClientServices =
       componentManager.getComponentService(NmeaClientServices.class);

nmeaClientServices.open("localhost", 8080);
nmeaClientServices.request(100);
```

La première partie est relative aux composants, ensuite le client se connecte sur le serveur, sur le port 8080 puis fait des requêtes toutes les 100 millisecondes.

Diffusion de données

Dans l'exemple présenté, les données reçues sont parsées à l'aide de l'API JAXB [5]. Nous utilisons le support de la programmation événementielle de C^3. A chaque objet NMEA instancié, l'événement correspondant est émis et diffusé auprès des abonnés.

```
public interface NMEAEvent
        extends ComponentEvent {
    public <T extends NMEA> void notifyNmeaMessageChanged(T data);
}

public interface GGAEvent
        extends NMEAEvent {
}
```

Ecriture d'un souscripteur à un événement

L'inscription à la réception d'événements d'un type particulier suit la démarche de C^3 :

1. Recherche du `componentManager`
2. Recherche de l'objet de souscription à un événement particulier `ggaES`
3. Souscription et sur-définition de la méthode réponse `notifyNmeaMessageChanged(T data)`
4. La méthode étant générique, l'héritage des événements NMEA, n'étant pas un véritable héritage, il convient d'appliquer une coercition de type afin d'analyser les données reçues.
5. Traitement des données.

5. https://jaxb.java.net/

Exemple de souscription et de traitement après réception d'un événement type GGA

```
public class Locator {

    ComponentManager cm = ComponentManager.componentManager;
    ComponentEventSubscribe<GGAEvent> ggaES =
                cm.getComponentEventSubscribe(GGAEvent.class);

    public Locator() {
        ggaES.subscribe(new GGAEvent() {

            @Override
            public <T extends NMEA> void notifyNmeaMessageChanged(T data) {
                GGA gga = (GGA) data;
                System.out.println("Latitude : " + gga.getLatitude()
                            + "   Longitude : " + gga.getLongitude());
            }
        });
    }
}
```

Autre développement possible

L'écriture d'un client n'impose que le respect du protocole WebSocket, il est donc tout à fait possible d'écrire des clients dans un autre langage que java et avec une philosophie de diffusion que la programmation événementielle. Un exemple de client élémentaire écrit en Javascript :

```
<html>
<head><title>Web Socket Test</title></head>
<body>
<script>
    var socket;
    if (window.WebSocket) {
        socket = new WebSocket("ws://localhost:8080/nmea");
        socket.onmessage = function(event) {
            alert("Received data from websocket: " + event.data);
        }
        socket.onopen = function(event) {
            alert("Web Socket opened!");
        };
        socket.onclose = function(event) {
            alert("Web Socket closed.");
        };
    } else {
        alert("Your browser does not support Websockets. (Use Chrome)");
    }
```

```
    function send(message) {
        if (!window.WebSocket) {
            return;
        }
        if (socket.readyState == WebSocket.OPEN) {
             socket.send(message);
        } else {
            alert("The socket is not open.");
        }
    }
</script>
<form onsubmit="return false;">
    <input type="text" name="message" value="Nmea"/>
    <input type="button" value="Send Web Socket Data"
                onclick="send(this.form.message.value)"/>
</form>
</body>
</html>
```

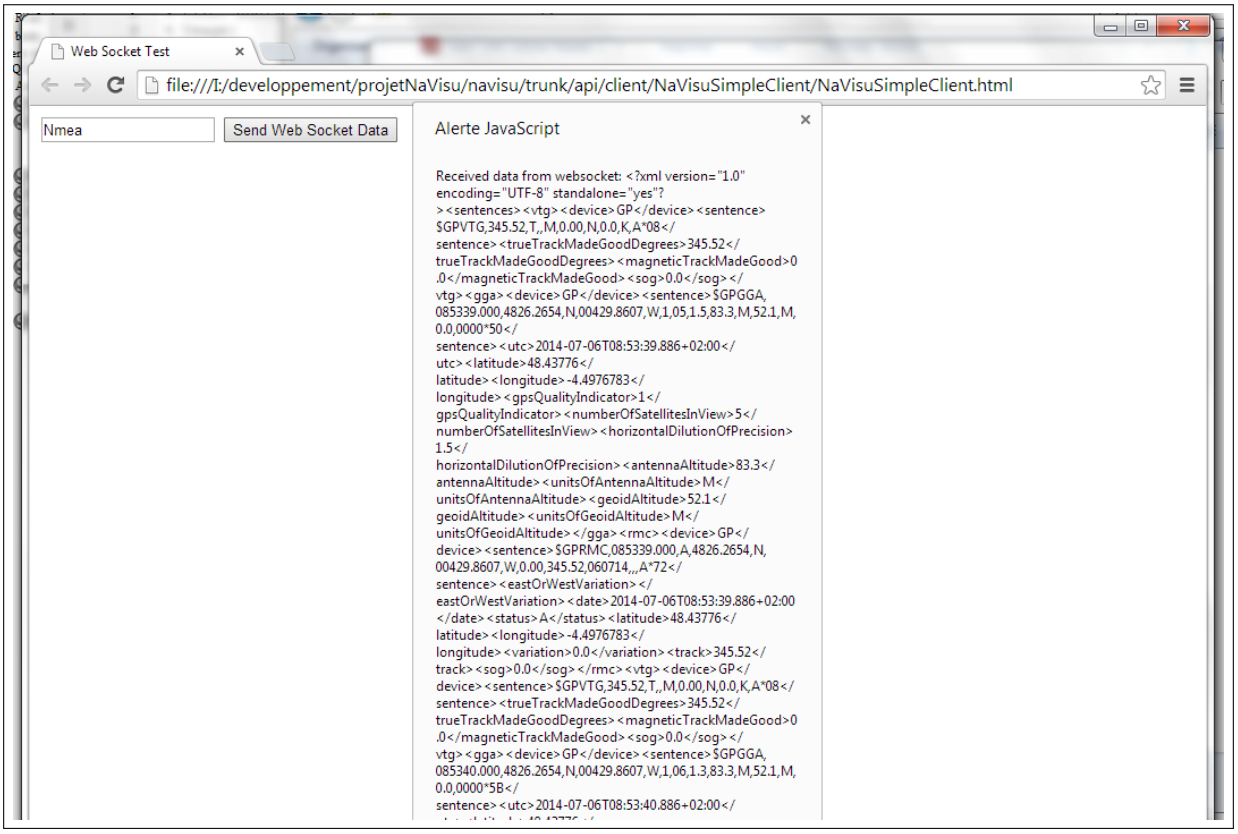

FIGURE 2 – *Un client élémentaire en javascript*