

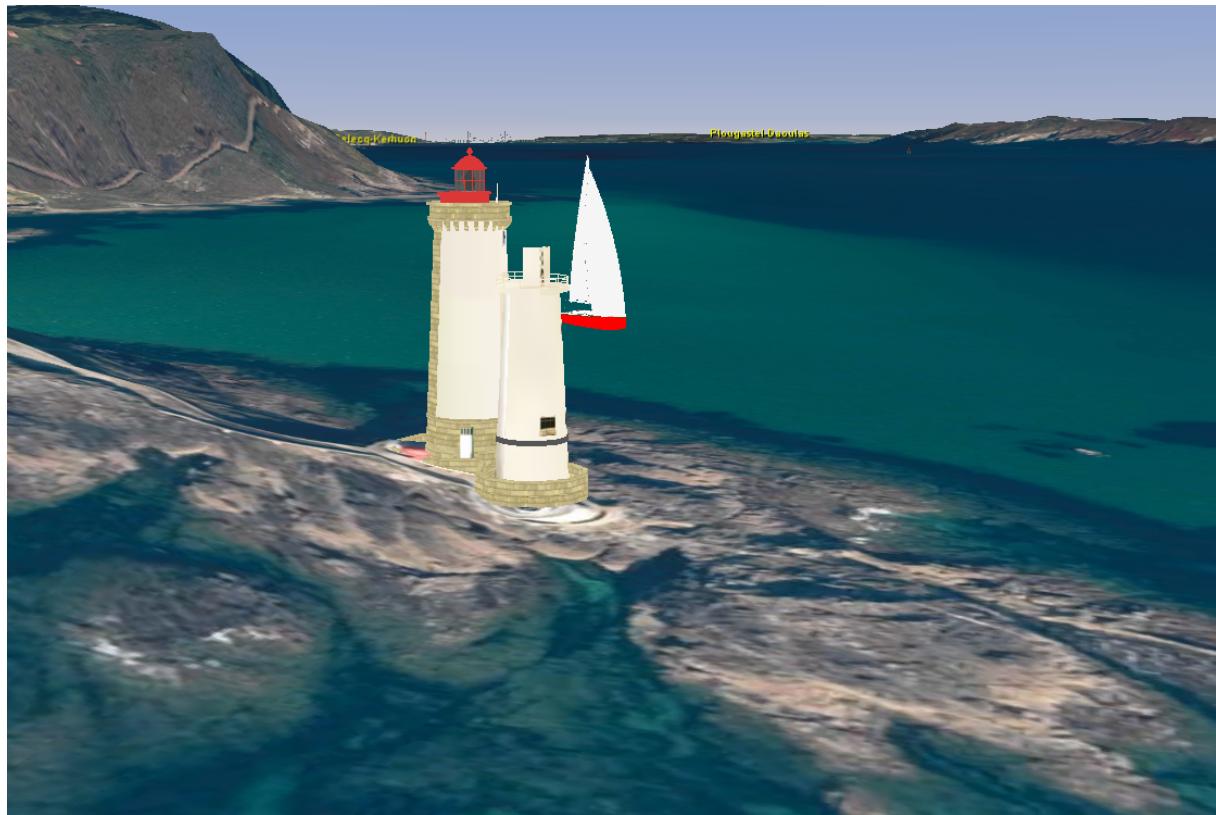
ASSOCIATION TERRE VIRTUELLE

PROJET NAVISU

<https://github.com/terre-virtuelle/navisu.git>

DOSSIER DE DÉVELOPPEMENT

LOGICIEL D'AIDE À LA NAVIGATION
MARITIME EN 3D



2015





Traçabilité du document

Rédaction

Personnes ayant participées au projet

Noms
3dphi
jojal29
lithops
mhyrdin
tibus29

Historique des évolutions

Version	Date
1.0	mai 2015

Image de couverture : Entrée du goulet de Brest, le phare du petit Minou (France)



Table des matières

1 Installation et personnalisation du serveur de données NAVISU	17
1.1 Présentation	17
1.2 Le projet	18
1.3 Téléchargement	18
1.4 Paramétrisation	18
1.5 Lancement	18
1.6 Développement	19
2 Les données NMEA 0183	21
2.1 Le standard NMEA	21
2.2 Modèle NMEA0183	22
2.2.1 Ensemble des phrases analysées	22
2.2.2 Minimun d'information pour la navigation	24
2.2.3 Date et heure	24
2.2.4 Position : latitude, longitude	25
2.2.5 Informations de vitesse et de distance	26
2.2.6 Informations sur les waypoints	27
2.2.7 Informations sur les satellites	28
2.2.8 Informations sur le vent	28
2.2.9 Bathymétrie	29
2.3 Présentation de l'API	29
2.3.1 L'analyseur lexical et syntaxique	29
2.4 Glossaire	31
3 Architecture et fonctionnalités du client NMEA	33
3.1 Les connexions au serveur	33
3.2 Les données en entrée	33
3.2.1 Extrait du schéma <code>nmea.xsd</code>	34
3.2.2 Extrait d'un fichier de données conforme au schéma <code>nmea.xsd</code>	34
3.3 Le projet <code>NaVisuSimpleClient</code>	35
3.4 Téléchargement	35
3.5 Paramétrisation	35
3.6 Lancement	35
3.7 Développement	35
3.8 Diffusion de données	36
3.9 Ecriture d'un souscripteur à un événement	36
3.9.1 Exemple de souscription et de traitement après réception d'un événement type <code>GGA</code>	37



3.10	Autre développement possible	37
4	Création d'un nouveau Display	39
4.1	Architecture	39
4.1.1	Principes	39
4.1.2	Diagramme UML	40
4.2	Créer un nouveau Display : Compass	40
5	Création des menus	41
5.1	Objet	41
5.2	Le graphisme	42
5.2.1	Module navisu-app	42
5.2.2	Module navisu-widgets	42
5.3	Le code	42
5.3.1	Module navisu-app	42
5.3.2	Choix des callbacks	44
5.3.3	Module navisu-widgets	45
5.4	Principe des DriverManager	45
6	Cartographie	49
6.1	Présentation	49
6.1.1	La projection Equirectangulaire	49
6.1.2	Tuilage	50
6.2	Cartographie raster	51
6.2.1	Les cartes BSB/KAP	51
6.2.2	Acquisition	53

Table des figures

1	<i>Déploiement centralisé</i>	12
2	<i>Déploiement distribué</i>	12
3	<i>Aucune requête client, les données dynamiques venant des capteurs sont perdues</i>	14
4	<i>Exemple de requête client, suivie d'une réponse asynchrone à partir du premier port série enregistré.</i>	15
5	<i>Exemple de requête client, suivie d'une réponse synchrone à partir de données provenant d'un fichier</i>	15
1.1	<i>Déploiement distribué</i>	17
2.1	<i>Ensemble des classes de l'API NMEA</i>	22
2.2	<i>Catégorisation des informations NMEA</i>	23
2.3	<i>Minimun d'information pour la navigation</i>	24
2.4	<i>Date</i>	24
2.5	<i>Informations de position</i>	25
2.6	<i>Informations de vitesse et de distance</i>	26
2.7	<i>Informations sur les waypoints</i>	27
2.8	<i>Informations sur les satellites</i>	28
2.9	<i>Informations sur le vent</i>	28
2.10	<i>Bathymétrie</i>	29
2.11	<i>Analyse de la phrase GSV</i>	30
2.12	<i>Analyse de la phrase RMB</i>	30
3.1	<i>Schéma général des entrées/sorties</i>	33
3.2	<i>Un client élémentaire en javascript</i>	38
4.1	<i>Diagramme UML simplifié d'un Display</i>	40
5.1	<i>L'item Instruments</i>	42
5.2	<i>L'icône centrale de Instruments et les sous-items</i>	42
5.3	<i>Le menu Instruments</i>	44
5.4	<i>Une modification de l'IHM va impacter l'application</i>	46
5.5	<i>Si les contrats de services sont respectés, un changement du code de l'IHM ne concerne pas l'application</i>	46
5.6	<i>Gradle détectera un tel problème</i>	46
6.1	<i>Projection équirectangulaire</i>	49
6.2	<i>Conventions de la numérotation des tuiles WorldWind</i>	50
6.3	<i>Frontières d'une carte</i>	51
6.4	<i>Les Etats Unis</i>	52
6.5	<i>Affichage de la carte</i>	52



6.6 <i>Niveaux de détail et la 3D</i>	53
---	----





Architecture et fonctionnalités du serveur de données

Ref : TV080114_TU_SM

Rédacteur : Serge Morvan

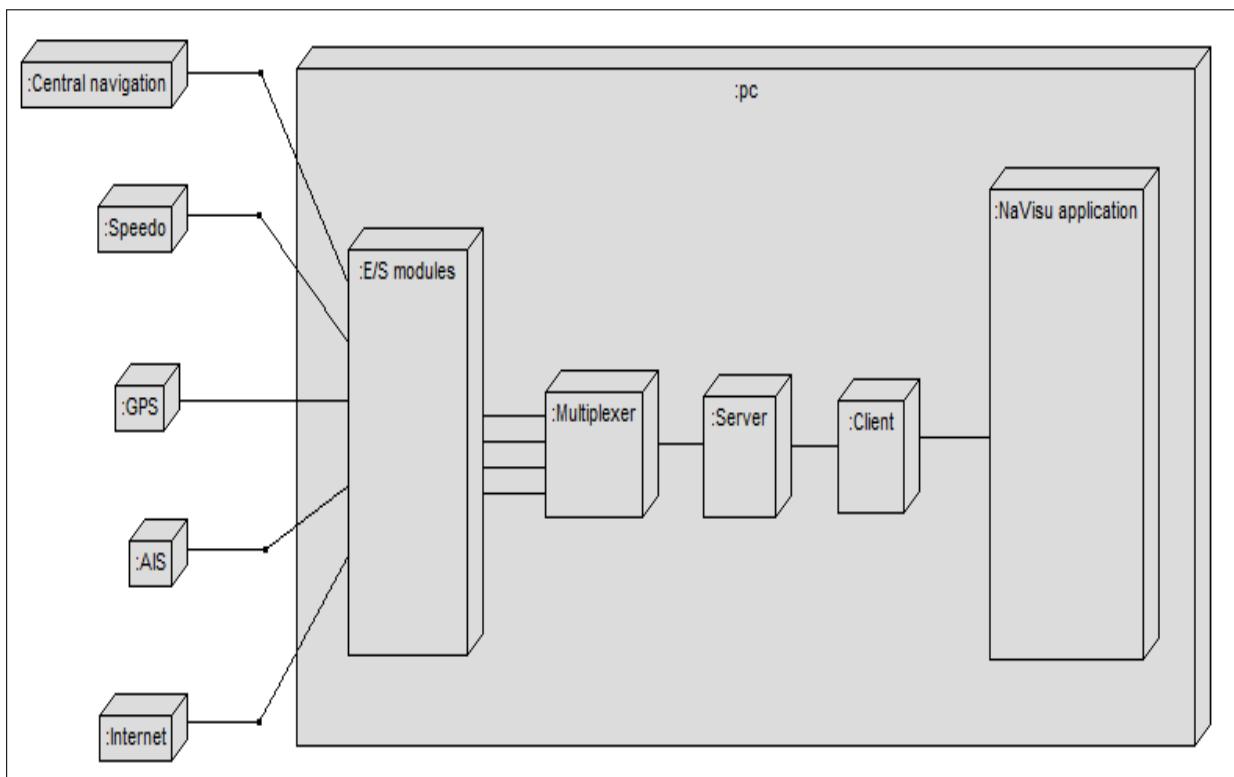
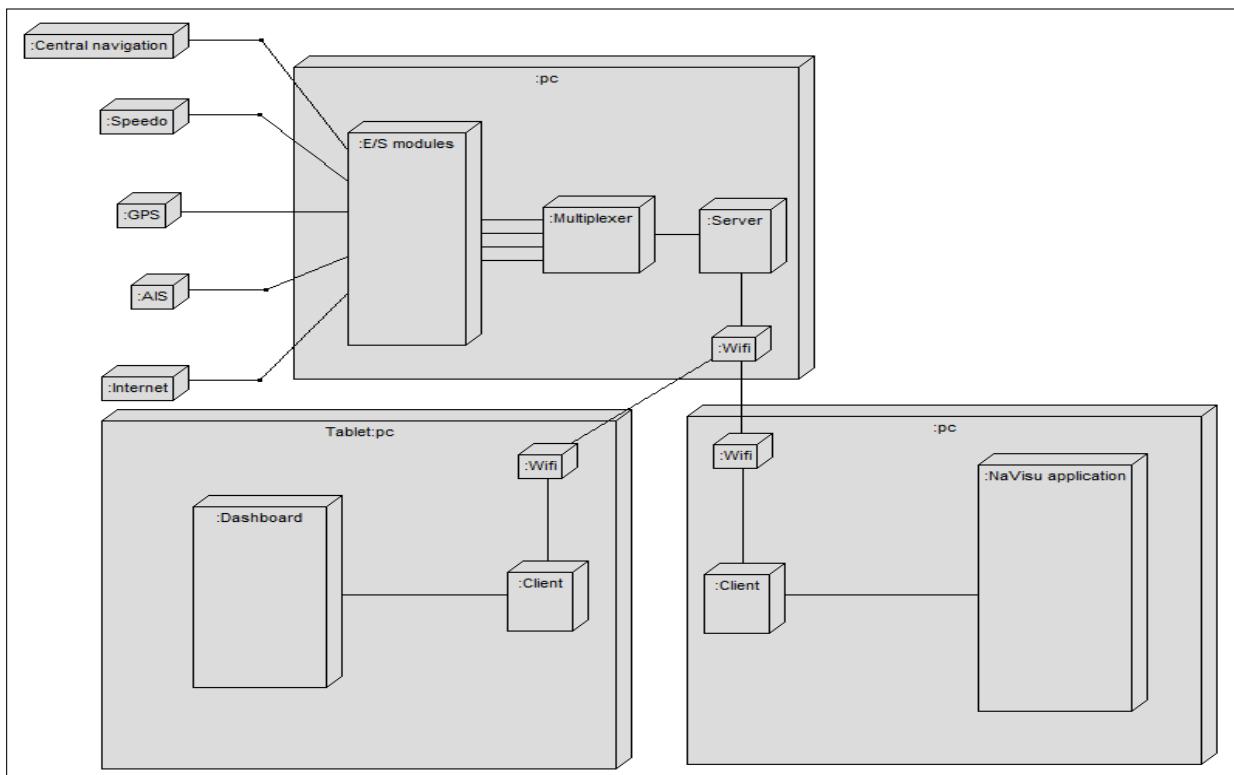
Présentation

Le module d'acquisition de données de NAVISU est distribué. Un serveur reçoit les données des différents capteurs les analyse, les traite puis les distribue au format XML [ou JSON (option)] aux différents clients. Dans la version la plus compacte serveur et client se trouvent sur une même machine. Le protocole de communication utilisé sont les WebSockets.¹ ce protocole permet les actions bidirectionnelles, *push* et *pull*, avec les clients. Actuellement seules les requêtes *pull* venant des clients sont prises en compte, mais il est simple d'introduire des actions *push* de la part du serveur pour des remontées d'alarmes par exemple. Le protocole WebSocket est implémenté dans la plupart des langages actuellement. L'implémentation du serveur repose sur le *framework* Vert.x^{2,3}, ce *framework* assure la communication asynchrone entre les entrées et les sorties du serveur, à l'aide de bus événementiels : **EventBus**. Il est simple à utiliser, ne nécessite pas d'installation particulière. L'intérêt de la distribution est la possibilité des développer des clients ayant différentes technologies d'interfaces : JavaFX, HTML5 et différents OS : Windows7/Windows8, Mac OS, iOS, Linux ou Android. les clients peuvent être une application NAVISU complète ou de simple affichages de données capteurs : des instruments par exemple, ou des clients de communication type chat.

1. <http://fr.wikipedia.org/wiki/WebSocket>

2. <http://vertx.io/>

3. <http://fr.wikipedia.org/wiki/Vert.x>


 FIGURE 1 – *Déploiement centralisé*

 FIGURE 2 – *Déploiement distribué*

Les données en entrée

Les données sont envoyées au serveur via les ports série, USB ou Internet. Il est aussi possible d'avoir des données à partir de fichiers, pour un rejeu de parcours par exemple. L'acquisition de données est réalisée par des objets Reader,

- **SerialReader** pour la communication série
- **HTTPReader** pour la saisie sur internet
- **FileReader** pour la lecture de données sur disque

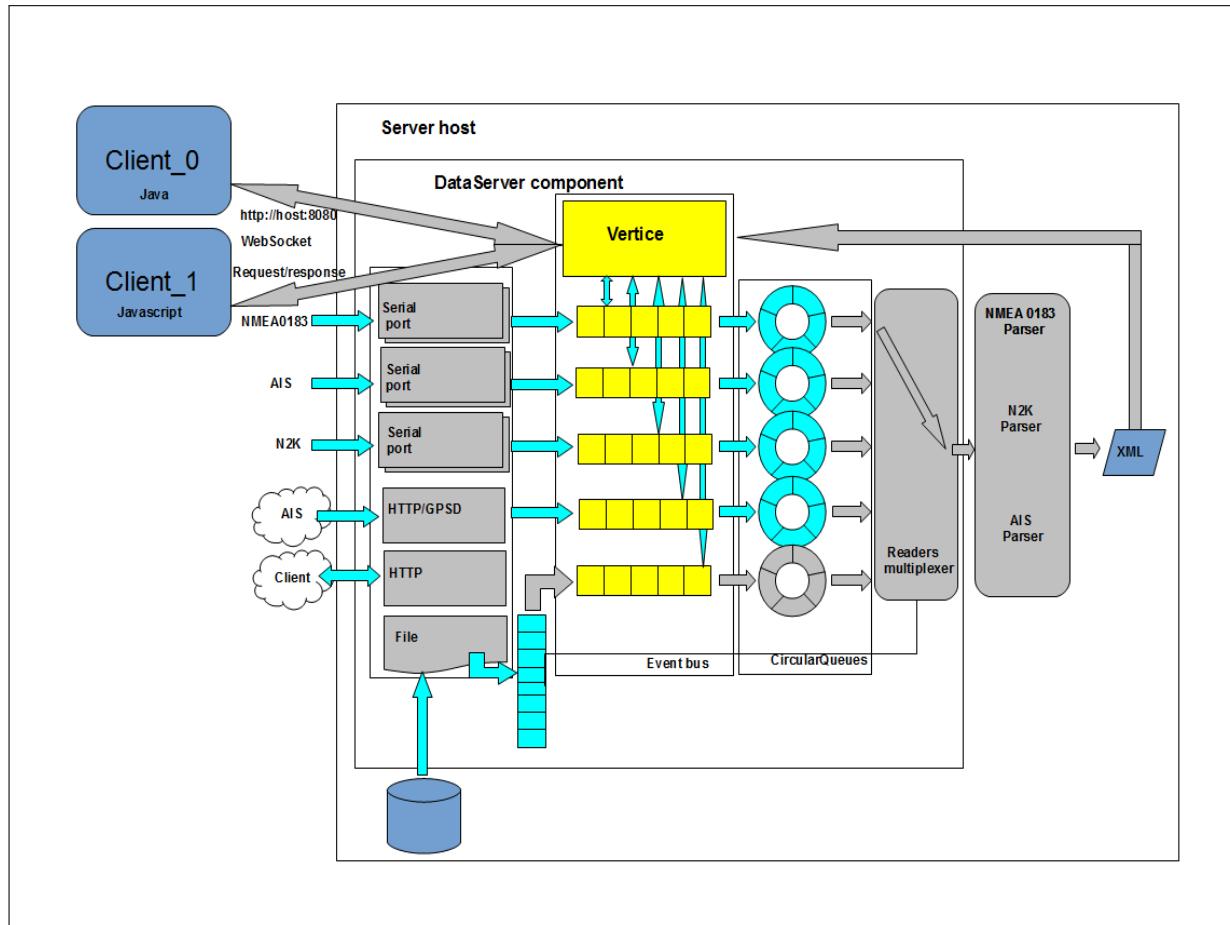
Le composant **DataServer** peut créer en dynamique les différents Reader et les paramétrer. Exemples de débits sur les ports série :

- 4800 bauds (GPS, centrale de navigation, ..., NMEA0183 ASCII)
- 38400 bauds (AIS, NMEA0183 binaire)
- 250 Kbits/sec (GPS, centrale de navigation, moteurs, ..., N2K binaire)

Exemples de frames pour les entrées capteurs :

```
NMEA0183 : $GPRMC,093518.470,A,4826.2552,N,00429.8708,W,000.0,203.5,051113,,,A*7A
AIS      : !AIVDM,1,1,,B,13IMVT00000cEt8KcIfW75F@0<0>,0*52
NMEA 2000 : <0x18eeff01> [8] 05 a0 be 1c 00 a0 a0 c0
```

L'architecture⁴



4. Dans les figures qui suivent les flux de données dynamiques sont coloriés en turquoise, les canaux de communication en gris clair.

FIGURE 3 – Aucune requête client, les données dynamiques venant des capteurs sont perdues

Un Reader spécialisé est associé à chaque entrée, un bus d'événements Vert.x lui est attribué, ainsi qu'une file circulaire. En entrée continu les données sont envoyées, via leur bus, à leur file. En sortie le débit d'acquisition est piloté par chaque client. Si il n'y a pas de requêtes client, les nouvelles données écrasent les anciennes, sauf pour celles venant de fichiers qui sont transférées intégralement. Un sélecteur de Reader assure le multiplexage des données, actuellement la stratégie du choix est simple parcours circulaire, mais une stratégie avec file à priorité peut être facilement mise en œuvre. Sur requête d'un client la file circulaire active délivre ses données au sélecteur d'analyseur syntaxique : NMEA0183, AIS, N2K. Le parseur choisi traite les données, génère des objets NMEA, les transforme au format XML et les envoie à l'objet Vertice qui les envoie au client. Une nouvelle file est sélectionnée dans l'attente d'une autre requête client. Les données sont donc analysées (*parsées*) deux fois, une première fois par le parseur NMEA spécialisé, une deuxième fois par le client à partir de d'un format XML. Ceci se justifie par le fait que l'écriture du premier parseur est complexe du fait de la grande variabilité des formats NMEA, les connexions électriques avec les capteurs peuvent aussi engendrer un bruitage des données, ces données sont ensuite mises en forme suivant un schéma XML unique, les clients n'ont aucune difficultés à les transformer en objets à partir de l'API JAXB. Dès que le format JSON sera bien intégré à JEE, ce format sera préféré à XML. D'autres protocoles, tel que CoAP sont envisageables dans l'avenir.

La dynamique

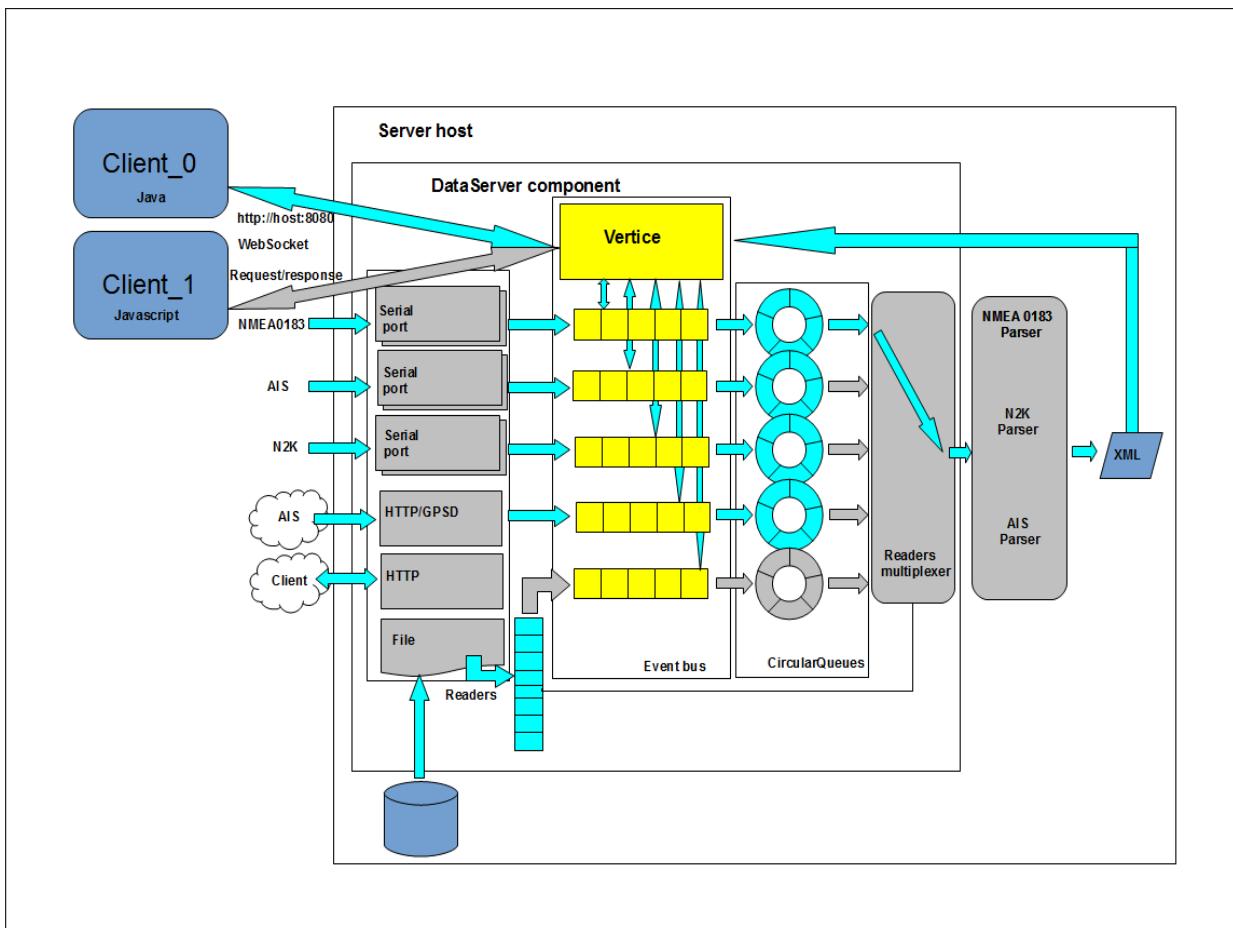


FIGURE 4 – Exemple de requête client, suivie d'une réponse asynchrone à partir du premier port série enregistré.

Performances

En test, le serveur supporte une requête client toutes les milisecondes, dans la pratique, un navire naviguant à 15 noeuds parcourt 7.5m en une seconde, un barreur va régler le débit d'affichage des données à 2 ou 3 secondes. Une période de 1 seconde pour les requêtes semble raisonnable.

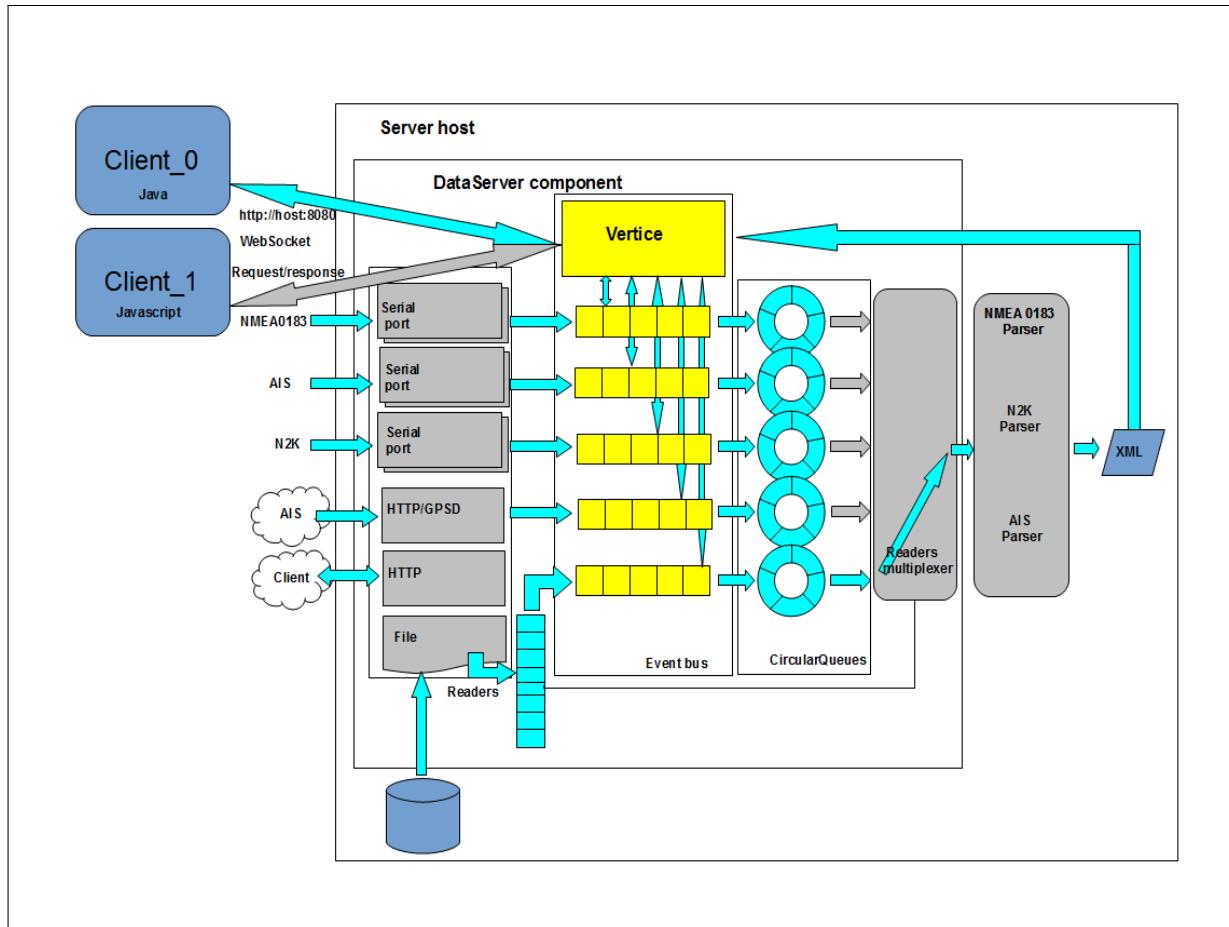


FIGURE 5 – Exemple de requête client, suivie d'une réponse synchrone à partir de données provenant d'un fichier.

Les données en sortie (extrait)

Extrait

```
<?xml version="1.0" encoding="UTF-8" standalone="true"?>
-<sentences>
-<rmc>
<device>GP</device>
<sentence>
$GPRMC,191137.304,A,4826.2620,N,00429.8665,W,000.0,225.3,050114,,,A*7C
```



```
</sentence>
<eastOrWestVariation/>
<date>2014-01-05T19:11:37.972+01:00</date>
<status>A</status>
<latitude>48.437702</latitude>
<longitude>-4.4977746</longitude>
<variation>0.0</variation>
<track>225.3</track>
<sog>0.0</sog>
</rmc>
-<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>
```

Chapitre 1

Installation et personnalisation du serveur de données NAVISU

1.1 Présentation

Ce tutoriel fait suite à celui ci : Architecture et fonctionnalités du serveur de données . Il a pour objet l'explication de la démarche complète d'installation, de paramétrisation, voire de la modification, du serveur de données capteurs. Ce serveur fait l'acquisition des données, assure leur multiplexage ainsi que la diffusion de ces données au format `xml`. Il est nécessaire d'installer ce serveur, dans le cas d'une application distribuée où l'acquisition et la visualisation ne se fait pas complètement ou uniquement au sein de NaVisu.

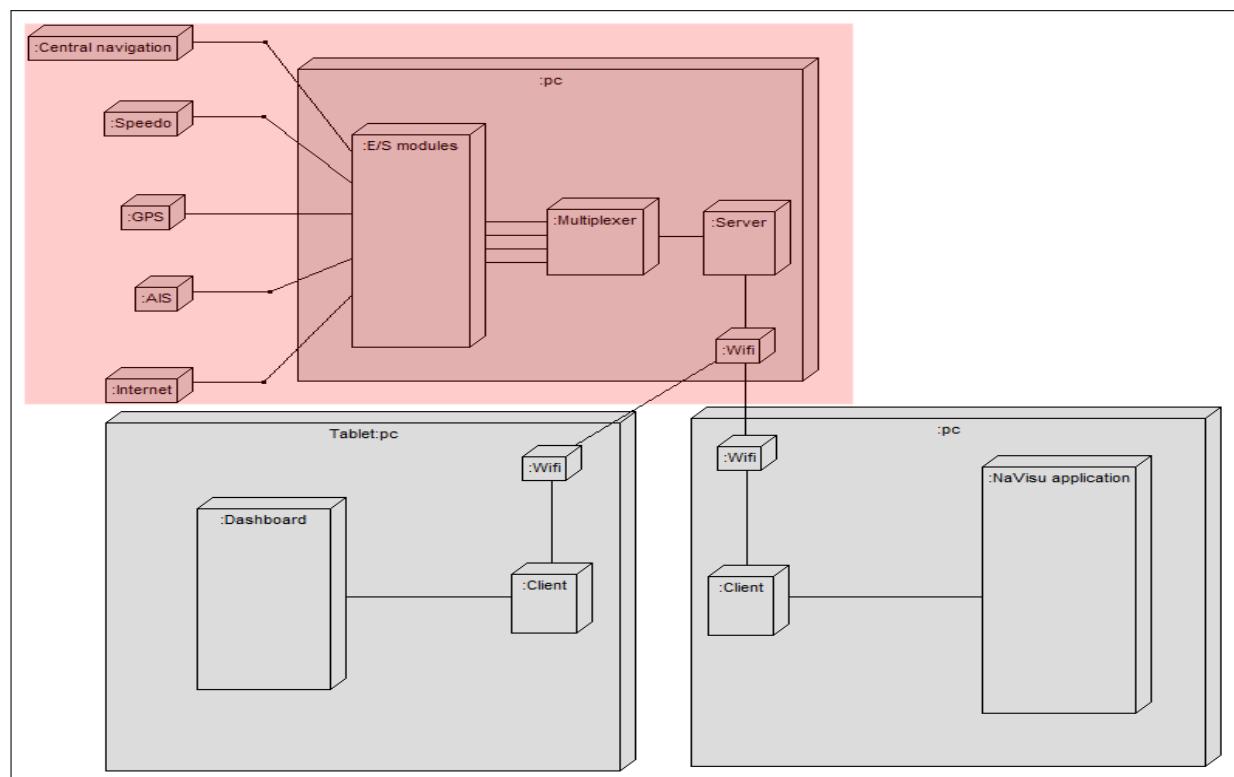


FIGURE 1.1 – Déploiement distribué



1.2 Le projet

Le projet NaVisu est 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 minimum de dépendance avec les autres sous-projets. C'est le cas de l'API `navisu-server`, 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.

1.3 Téléchargement

<https://github.com/terre-virtuelle/Navisu-server.git>

1.4 Paramétrisation

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

```
# NmeaServer properties file

#File name for tests without serial comm
fileName = data/nmea/gps.txt

# Web server parameters
hostName = localhost
port = 8080
queueSize = 5

# Serial parameters
# [portNumber - 1] for Linux
portNumber = 5
# COM for Windows OS, /dev/ttyS for Linux
portName = COM5
baudRate = 4800
dataBits = 8
stopBits = 1
parity = 0
```

Modifier les paramètres en fonction de votre configuration. Attention à la dénomination différente des ports sous Windows et sous Linux.

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

1.5 Lancement

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



1.6 Développement

La classe de test est : `bzh.terrevirtuelle.navisu.server.app.ServerMain`

```
ComponentManager componentManager = ComponentManager.componentManager;
// deploy components
LOGGER.log(Level.INFO, "\n Start", componentManager.startApplication(
    DataServerImpl.class
));
DataServerServices nmeaServerServices =
    componentManager.getComponentService(DataServerServices.class);

// Test avec choix des parametres de comm
// nmeaServerServices.init("localhost", 8080);
// nmeaServerServices.openSerialPort("COM4", 4800, 8, 1, 0);
// nmeaServerServices.openSerialPort("COM5", 4800, 8, 1, 0);

// Test avec les parametres de comm dans properties/nmea.properties
nmeaServerServices.init();
nmeaServerServices.openSerialPort();
// nmeaServerServices.openFile();
```

La première partie du code est relative à la gestion des composants. Ensuite deux séries d'exemples la première série fixe dans le code les paramètres d'acquisition et de diffusion. La deuxième série utilise les valeurs par défaut indiquées dans le fichier de propriétés. L'initialisation doit être unique, par contre on peut ouvrir autant d'entrées que l'on veut.



Chapitre 2

Les données NMEA 0183

2.1 Le standard NMEA

<http://standards.nmea.org/> Le site officiel :

<http://www.nmea.org/>

Des informations sur le contenu des phrases :

<http://gpsinformation.net/>

<http://gpsd.berlios.de/NMEA.html>

<http://www.kh-gps.de/nmea.faq>

Un projet de service démon :

<http://catb.org/gpsd/>

2.2 Modèle NMEA0183

2.2.1 Ensemble des phrases analysées

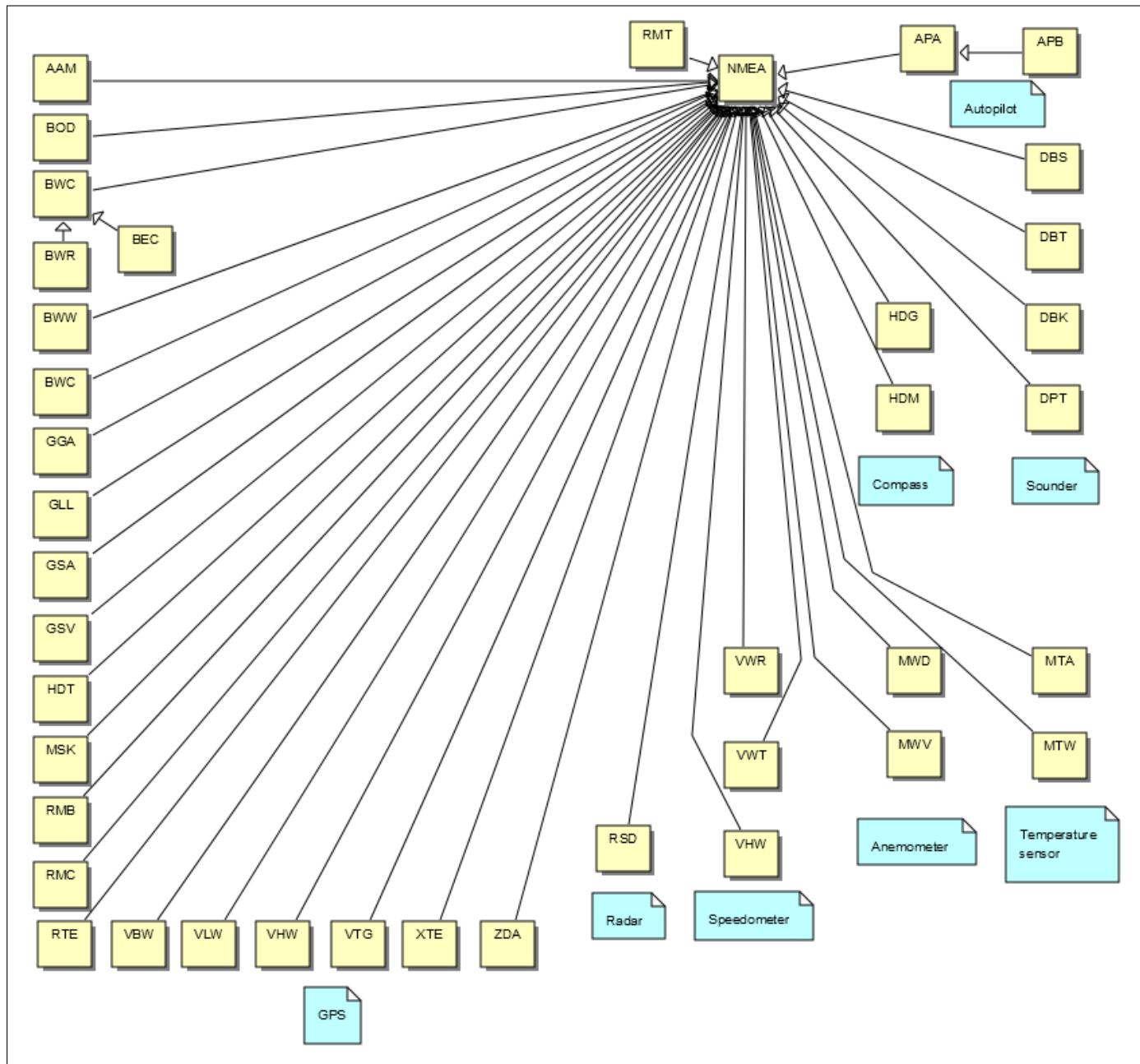


FIGURE 2.1 – Ensemble des classes de l'API NMEA

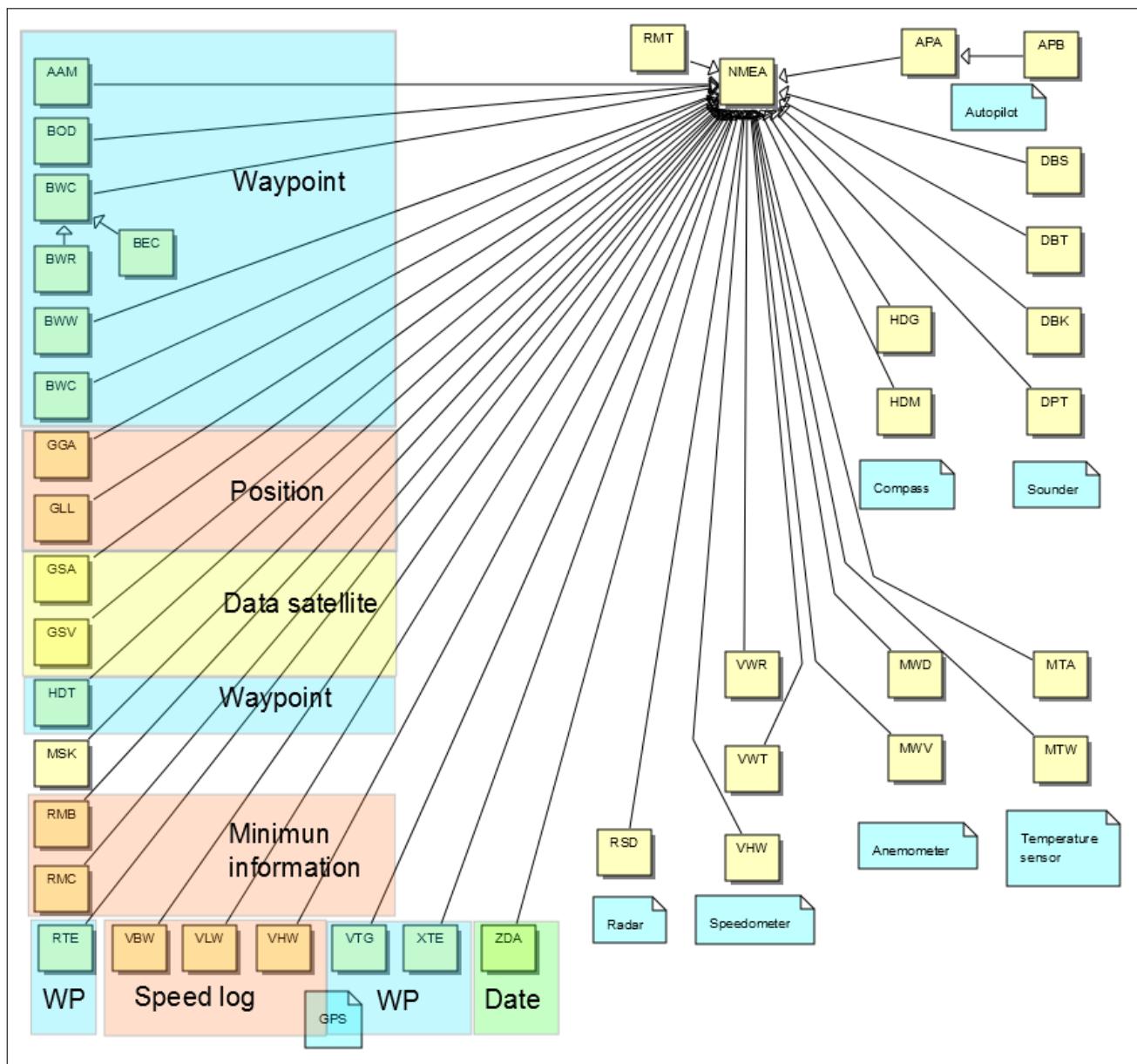


FIGURE 2.2 – Catégorisation des informations NMEA



2.2.2 Minimun d'information pour la navigation

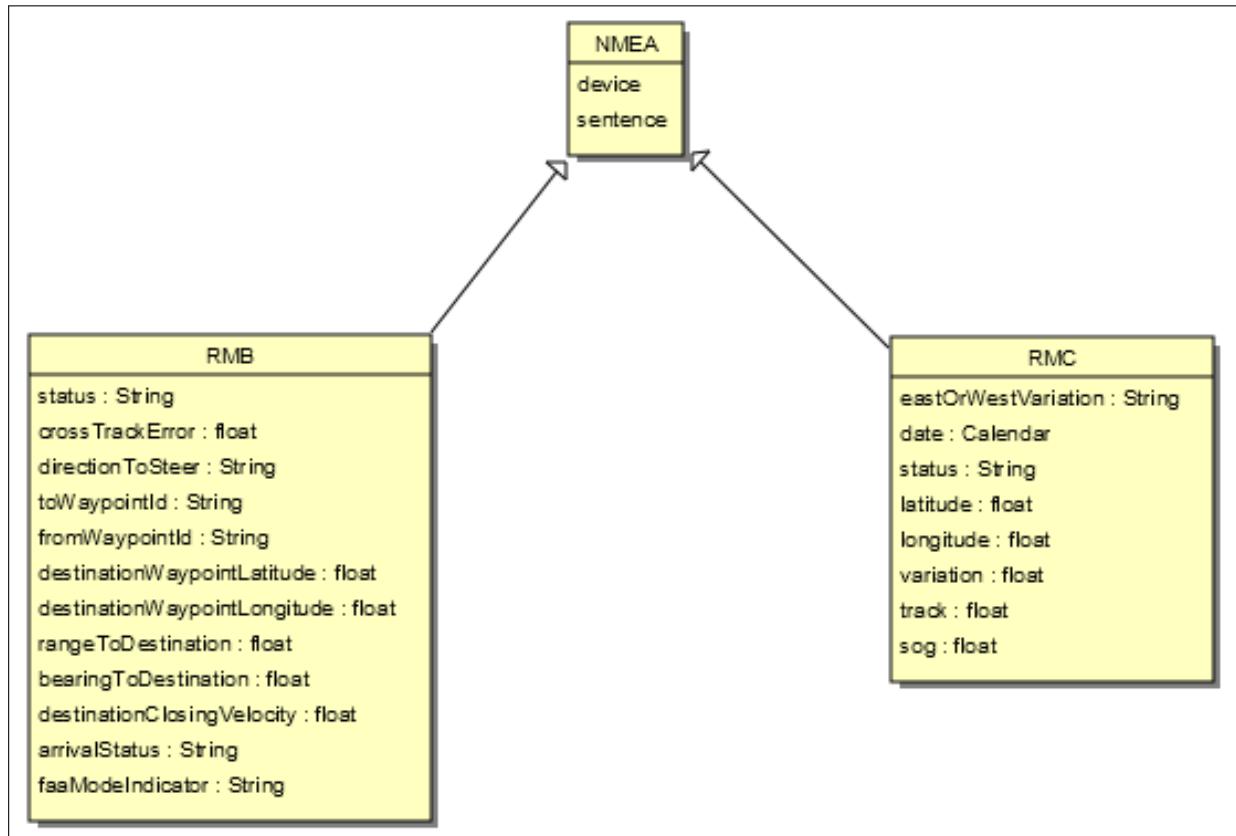


FIGURE 2.3 – *Minimun d'information pour la navigation*

2.2.3 Date et heure

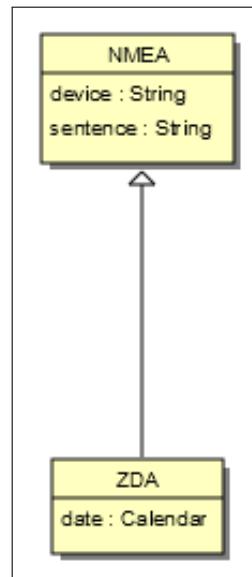


FIGURE 2.4 – *Date*

2.2.4 Position : latitude, longitude

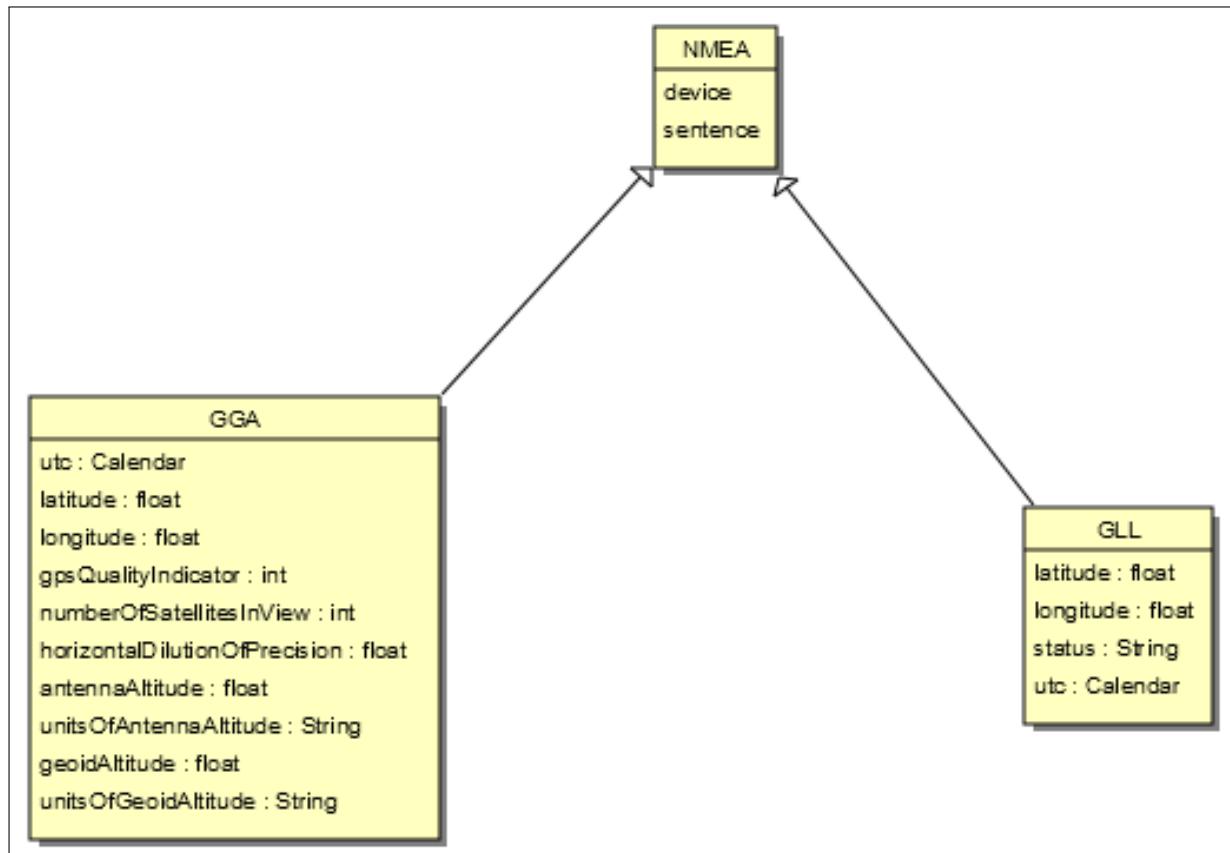


FIGURE 2.5 – *Informations de position*



2.2.5 Informations de vitesse et de distance

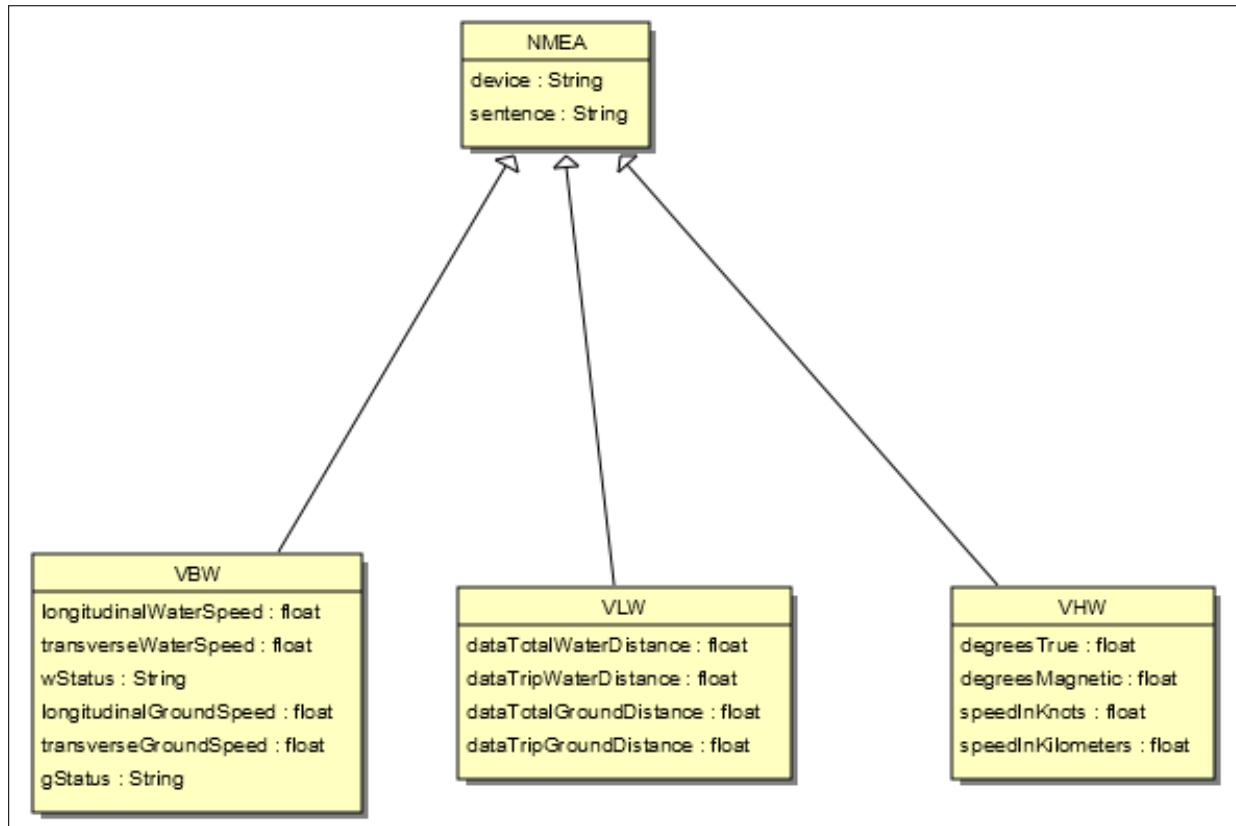


FIGURE 2.6 – *Informations de vitesse et de distance*

2.2.6 Informations sur les waypoints

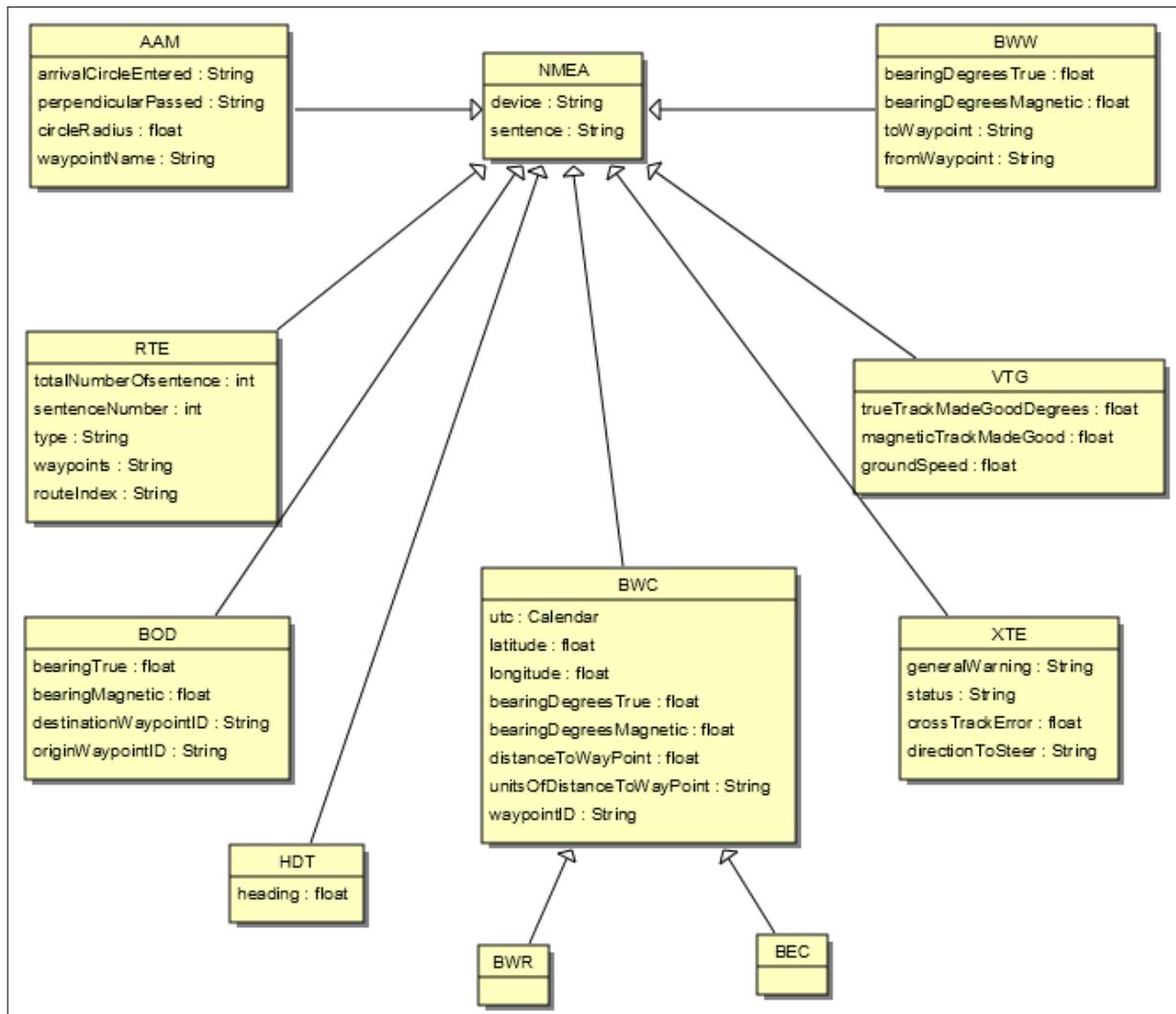


FIGURE 2.7 – *Informations sur les waypoints*

2.2.7 Informations sur les satellites

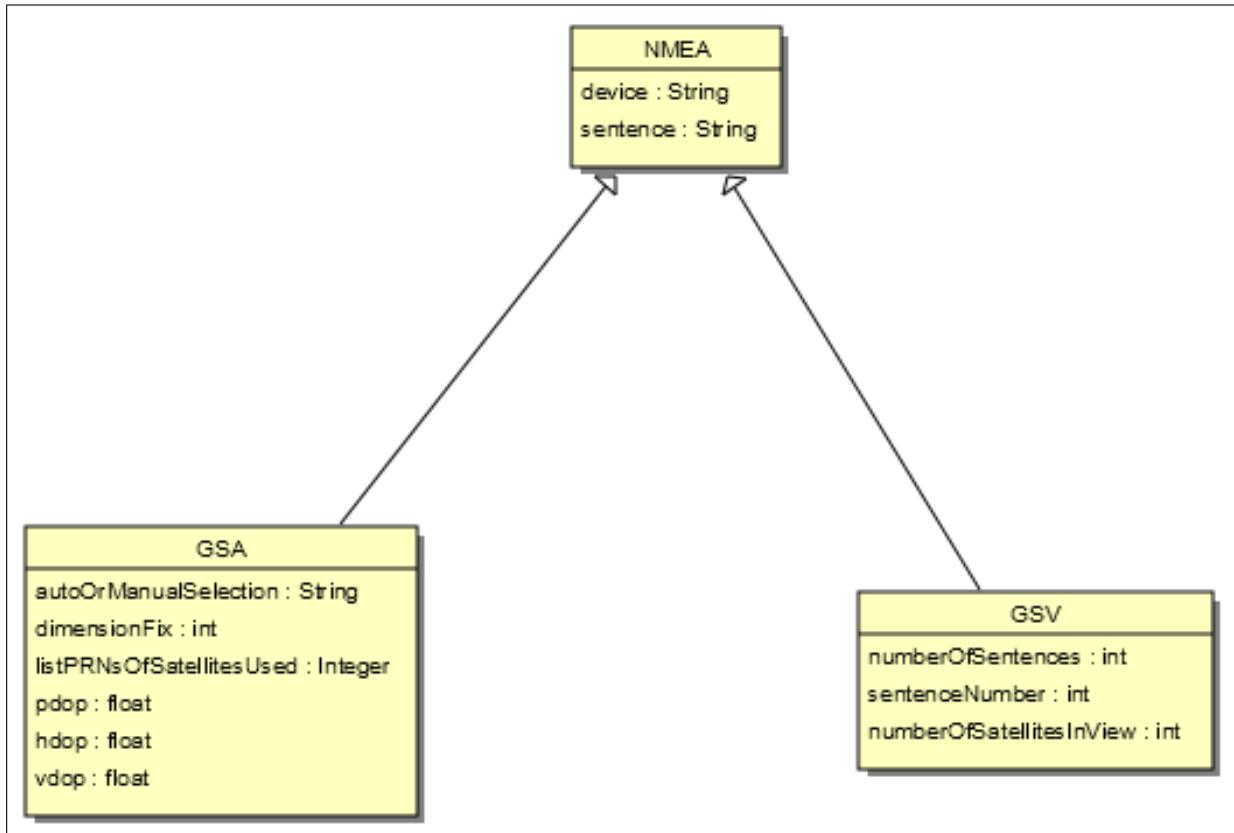


FIGURE 2.8 – *Informations sur les satellites*

2.2.8 Informations sur le vent

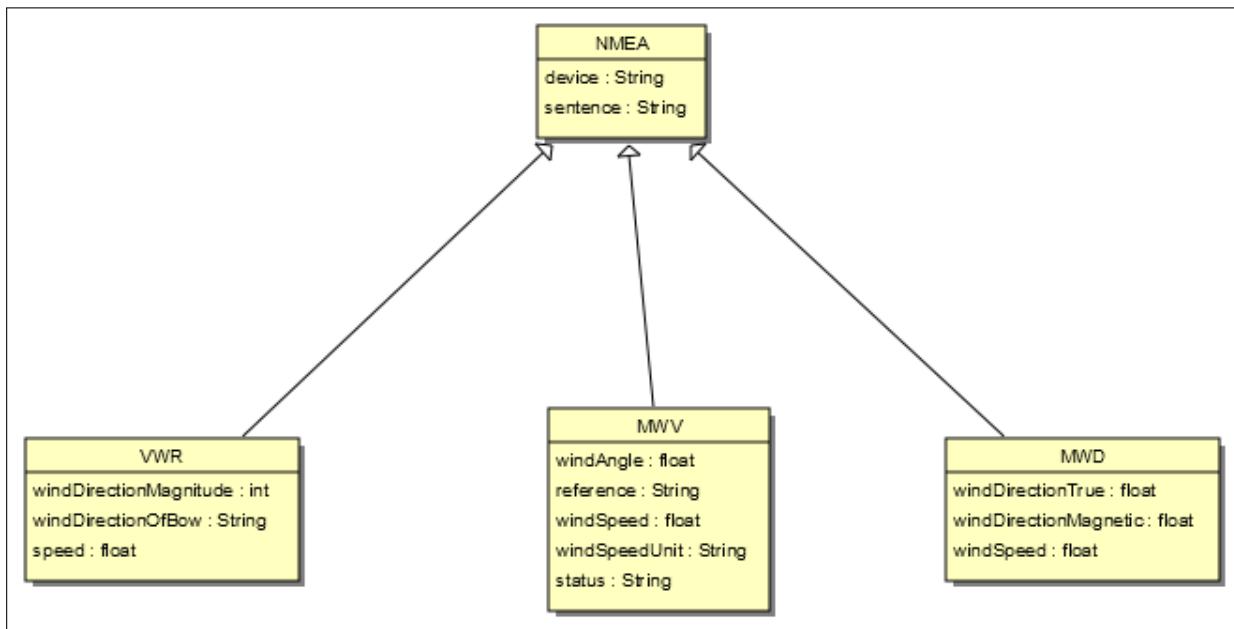


FIGURE 2.9 – *Informations sur le vent*



2.2.9 Bathymétrie

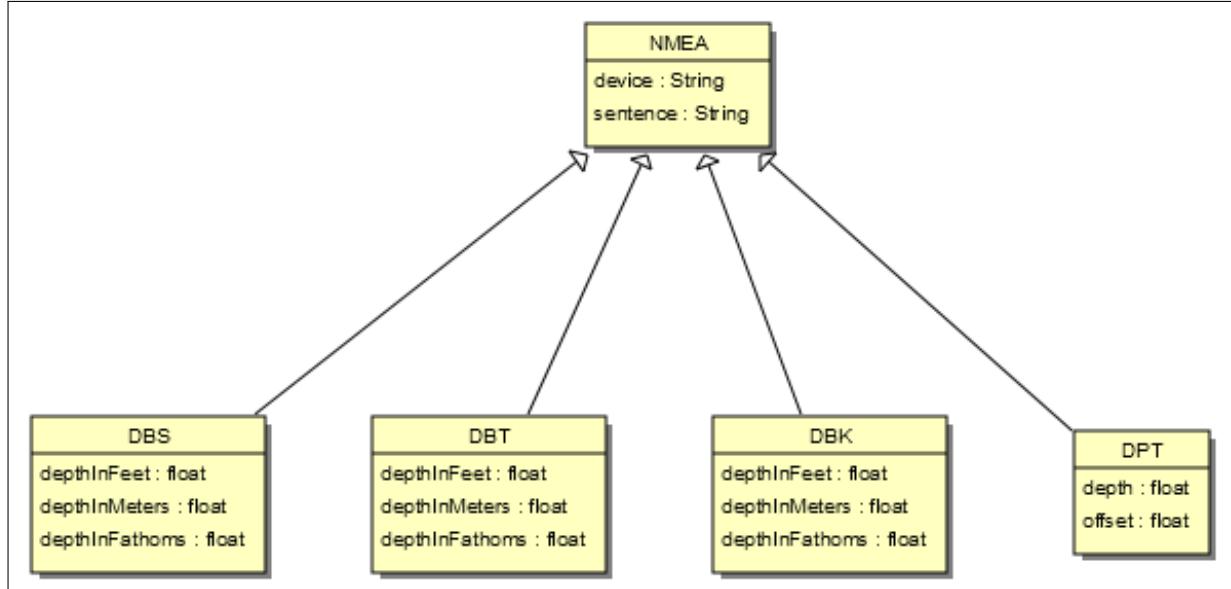


FIGURE 2.10 – *Bathymétrie*

2.3 Présentation de l'API

L'API permet l'accès au modèle NMEA que nous avons choisi, celui-ci est proche de l'ensemble des phrases mais s'en éloigne parfois lorsqu'il y a redondance d'information dans une phrase, plusieurs valeurs de vitesse dans différentes unités par exemple, ou lorsqu'une structure plus élaborée permet de stocker des données : liste, dictionnaire, ...

2.3.1 L'analyseur lexical et syntaxique

Le standard NMEA a beaucoup évolué, plusieurs constructeurs ont imposé leur modifications, ce qui fait que de nombreuses variantes apparaissent dans les sorties NMEA des différents appareils. Nous avons choisi d'utiliser le générateur d'analyseurs lexicaux et syntaxiques ANTLR pour plus de clarté du code.

<http://www.antlr.org/>



En plus de cet outil nous utilisons l'atelier ANTLRWorks :

<http://www.antlr.org/works/>

Cet atelier permet de visualiser les grammaires créées.

Un exemple simple

```
GSV      : '$' device=DEVICE 'GSV'  
(NUMBER | SEP)+  
checksum=CHECKSUM  
{ //action }
```

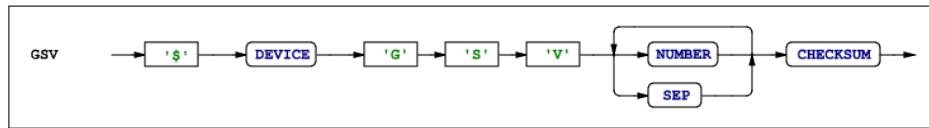


FIGURE 2.11 – Analyse de la phrase GSV

Un exemple plus complexe

```
RMB      : '$' device = DEVICE 'RMB' SEP  
status = LETTERS SEP  
(crossTrackError = NUMBER)* SEP  
(directionToSteer = LETTERS)* SEP  
(fromWaypointId = LETTERS | fromWaypointId = NUMBER)* SEP  
(toWaypointId = LETTERS | toWaypointId = NUMBER)* SEP  
(destinationWaypointLatitude = NUMBER)* SEP (ns = LETTERS)* SEP  
(destinationWaypointLongitude = NUMBER)* SEP (ew = LETTERS)* SEP  
(rangeToDestination = NUMBER)* SEP  
(bearingToDestination = NUMBER)* SEP  
(destinationClosingVelocity = NUMBER)* SEP  
(LETTERS SEP)*  
(arrivalStatus = LETTERS | '\u0000')*  
checksum=CHECKSUM  
{ // action}
```

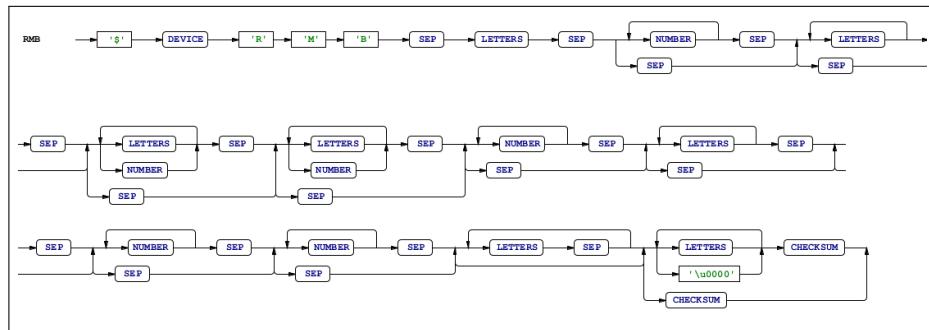


FIGURE 2.12 – Analyse de la phrase RMB



2.4 Glossaire

Acronyme	Expansion
AAM	Waypoint Arrival Alarm
ALM	GPS Almanac Data
APA	Autopilot Sentence “A”
APB	Autopilot Sentence “b”
BEC	Bearing and Distance to Waypoint, Dead Reckoning
BOD	Bearing, Waypoint to Waypoint
BWC	Bearing and Distance to Waypoint
BWR	Bearing and Distance to Waypoint
BWW	Bearing and Waypoint to Waypoint
DBK	Depth Below Keel
DBS	Depth Below Surface
DBT	Depth Below Transducer
DPT	Depth and offset
GGA	Global Positioning System Fix Data. Time
GLL	Geographic Position and Latitude/Longitude
GSA	GPS DOP and active satellites
GSV	Satellites in view
GTD	Geographic Location in Time Differences
HDG	Heading, Deviation and Variation
HDM	Heading Magnetic
HDT	Heading True
MTW	Water Temperature
MWV	Wind Speed and Angle
RMA	Recommended Minimum Navigation Information
RMB	Recommended Minimum Navigation Information
RMC	Recommended Minimum Navigation Information
RTE	Routes
VBW	Dual Ground/Water Speed
VHW	Water Speed and Heading
VLW	Distance Traveled through Water
VPW	Speed, Measured Parallel to Wind
VTG	Track Made Good and Ground Speed
VWR	Relative Wind Speed and Angle
XTE	Cross-Track Error Measured
ZDA	Time, Date and UTC, Day, Month, Year and Local Time Zone



Chapitre 3

Architecture et fonctionnalités du client NMEA

3.1 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.

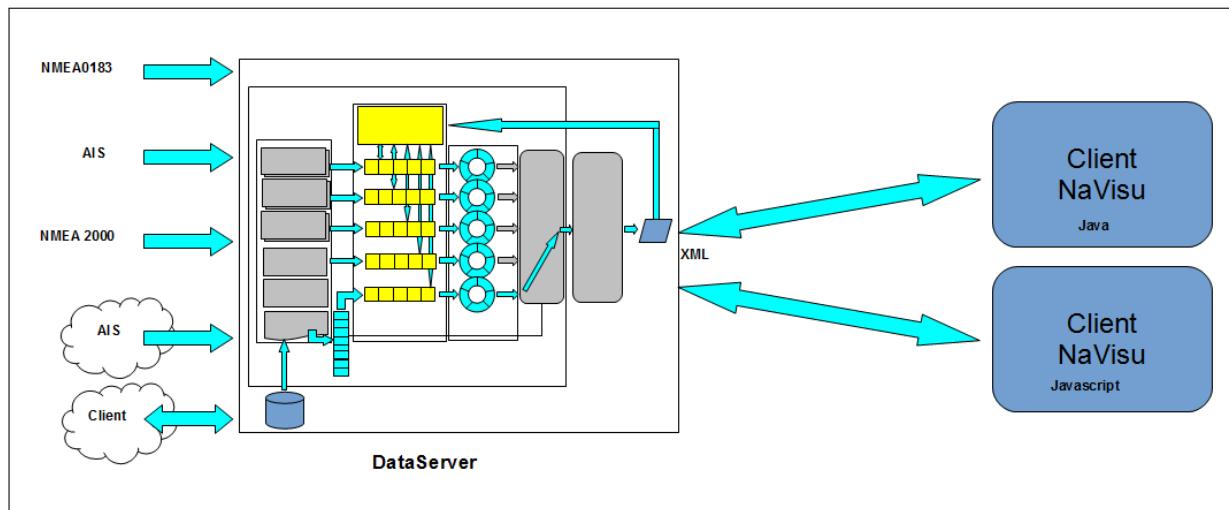


FIGURE 3.1 – Schéma général des entrées/sorties

3.2 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 : GPX¹ ou AIS² 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 NMEA³ associé à un schéma nmea.xsd structurant les données sans ambiguïtés et permettant leur validation.

3.2.1 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="number0fSatellitesInView" 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>
```

3.2.2 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>
    <number0fSatellitesInView>5</number0fSatellitesInView>
    <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



3.3 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 minimum 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`

3.4 Téléchargement

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

3.5 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.

3.6 Lancement

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

3.7 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.

3.8 Diffusion de données

Dans l'exemple présenté, les données reçues sont parsées à l'aide de l'API JAXB⁵. Nous utilisons le support de la programmation événementielle de C³. 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 {
}
```

3.9 Ecriture d'un souscripteur à un événement

L'inscription à la réception d'événements d'un type particulier suit la démarche de C³ :

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/>



3.9.1 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 notifyNMEAChanged(T data) {  
                GGA gga = (GGA) data;  
                System.out.println("Latitude : " + gga.getLatitude()  
                    + " Longitude : " + gga.getLongitude());  
            }  
        });  
    }  
}
```

3.10 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)");  
    }</script>
```



```
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>
```

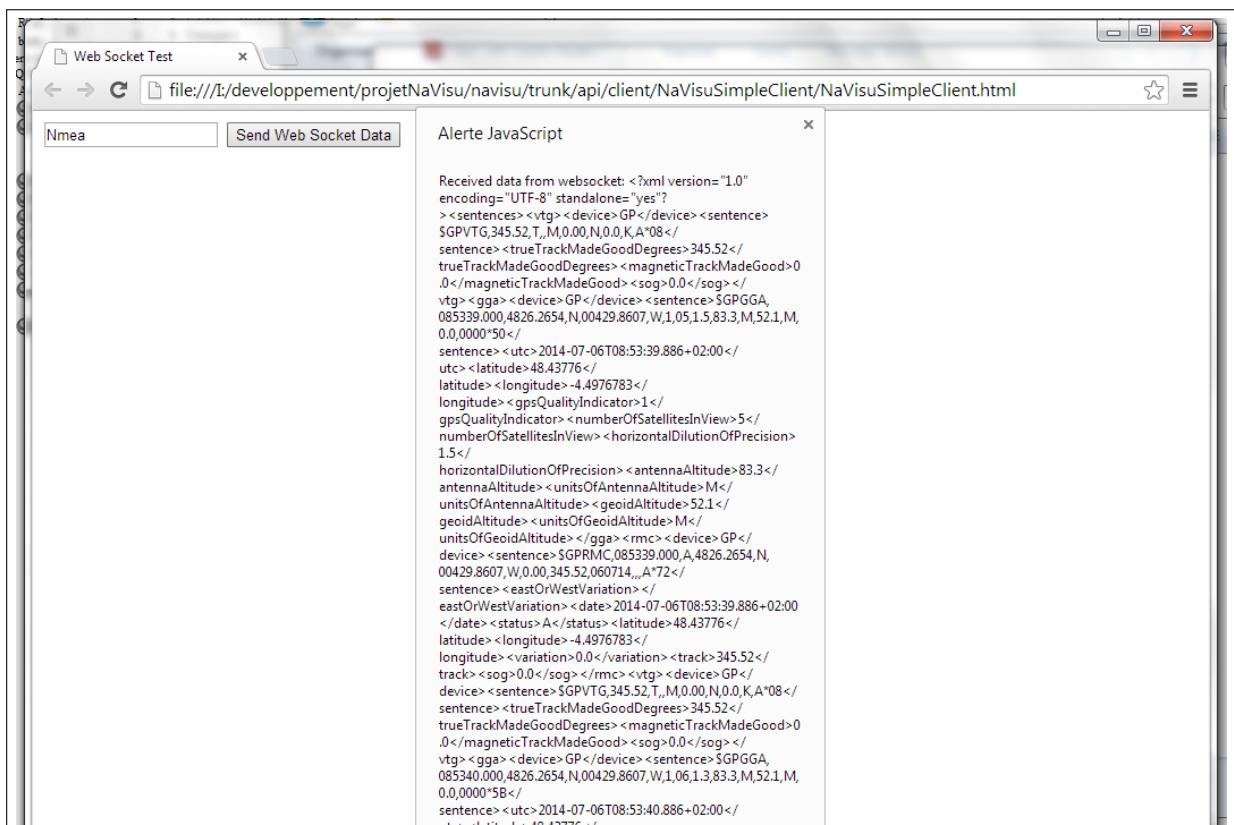


FIGURE 3.2 – Un client élémentaire en javascript

Chapitre 4

Création d'un nouveau Display

Ref : TV310315_TU_SM

Rédacteur : Serge Morvan

4.1 Architecture

4.1.1 Principes

Un **Display** est un composant, au sens de C³, il peut ainsi bénéficier des services du composant **GuiAgent** par exemple. L'application n'a aucun lien codé avec lui. Le design du nouveau composant est complètement séparé du code. Chaque élément variable possède un id. Le contrôleur, instancié par le **Display** lors de l'initialisation charge le fichier **.fxml**. On retrouve dans le code du contrôleur les id de la partie graphique. Préconisations : le composant graphique principal est un **Group**, son id est **:view**, chaque **Display** possède un bouton de fermeture, dont l'id est **quit**. Le contrôleur hérite de la classe **Widget2D** qui offre les services de l'interactivité. L'attribut **KEY_NAME**, ici : **"InstrumentTemplate"** permet au Dock de le rechercher et de l'afficher, lorsque l'item associé est sélectionné. Comme ci dessous dans la classe **DockManagerImpl** du module **navisu-app**, lors de la création du Dock :

```
instrumentsRadialMenu = RadialMenuBuilder.create()
    .centralImage("instrumentsradialmenu150.png")
    .createNode(0, "navigation.png", 1, "ais.png", 1, "template.png", (e) -> open("InstrumentTemplate"))
    .build();
```

Le nouveau **Display** devra s'enregistrer auprès du **InstrumentDriverManagerServices**, dans la classe **AppMain** du module **navisu-launcher**.

```
// deploy components
LOGGER.info("\n"
+ componentManager.startApplication(DpAgentImpl.class,
.....
InstrumentTemplateImpl.class
)
);
.....
InstrumentTemplateServices instrumentTemplateServices
    = componentManager.getComponentService(InstrumentTemplateServices.class);
.....
instrumentDriverManagerServices.registerNewDriver(instrumentTemplateServices.getDriver());
```



4.1.2 Diagramme UML

La figure ci dessous présente un diagramme simplifié pour un Display appelé **InstrumentTemplate**,

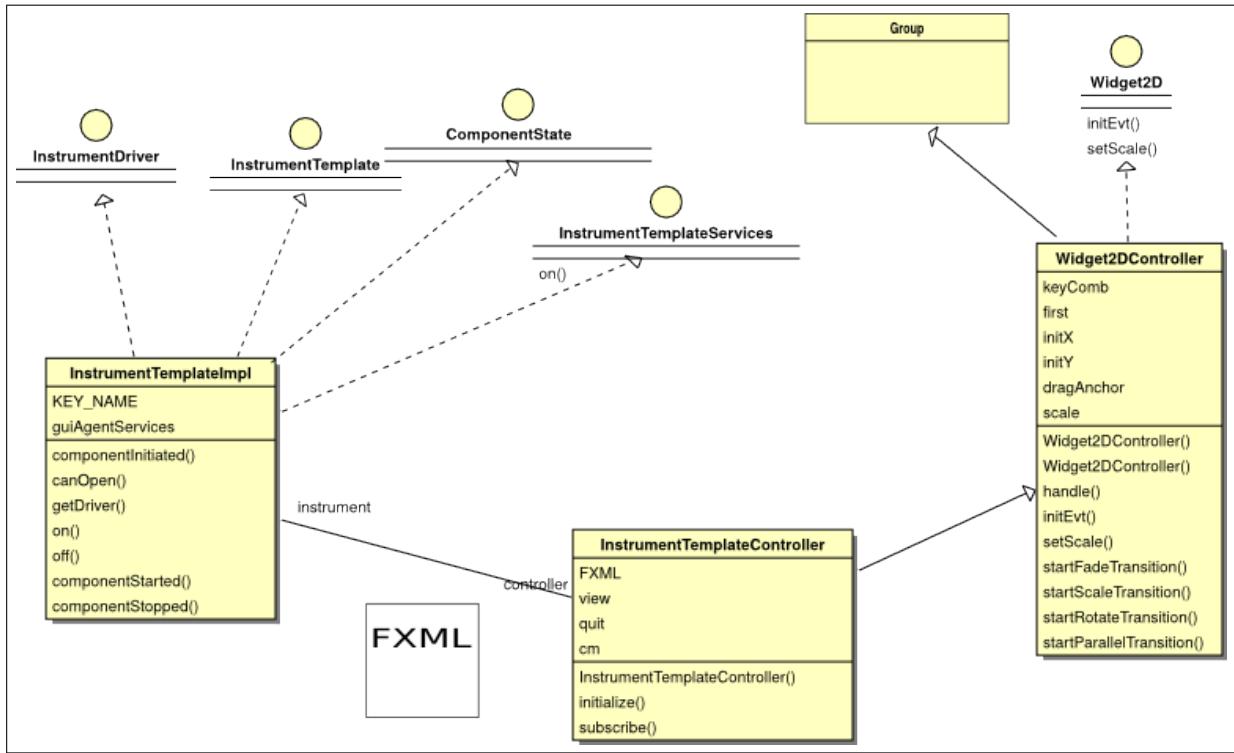


FIGURE 4.1 – Diagramme UML simplifié d'un Display

4.2 Créez un nouveau Display : Compass

Reprendre le code du **InstrumentTemplate**, écrire les interfaces **Compass** et **CompassServices**, implémenter les classes **CompassImpl** et **CompassController**.Créer le fichier graphique **compass.fxml**. Identifier par un id chaque variable. Reprendre ces variables en mode public dans le contrôleur. Eventuellement ajouter des services, par défaut un **Display** offre le service **on()**. Implémenter les contrôles.

Chapitre 5

Création des menus

Ref : TV110415_TU_SM

Rédacteur : Serge Morvan

5.1 Objet

Le but de cette section est d'expliquer la façon de rajouter un menu et ses sous-menus au Dock. Un certain nombre de services doivent être activés, ou désactivés par l'utilisateur, pour cela, le menu du Dock est privilégié. Le service à activer devra faire partie du groupe des **Driver** :

- **Driver** : pour l'ouverture de fichiers
- **DDriver** : pour le parcours de répertoires
- **InstrumentDriver** : pour le choix d'un instrument
- **DatabaseDriver** : pour la connexion à une base de données
- **WebDriver** : pour la connexion à un site web via un URL
- ...autres à venir

Ce service doit être enregistré dans le module **navisu-launcher** :

```
InstrumentDriverManagerServices instrumentDriverManagerServices =
    componentManager.getComponentService(InstrumentDriverManagerServices.class);
instrumentDriverManagerServices.init();
instrumentDriverManagerServices.registerNewDriver(sonarServices.getDriver());
instrumentDriverManagerServices.registerNewDriver(radarServices.getDriver());
```

Ensuite **deux modules** sont concernés, le module **navisu-app** pour spécifier le nouvel item dans le Dock (icônes et code) et le module **navisu-widgets** (icônes) car les **RadialMenu** sont des widgets. Dans les deux cas les images sont placées dans les ressources, correspondant à l'arborescence des classes **DockManagerImpl** et **RadialMenu** respectivement. Dans la suite nous prendrons l'exemple de l'instanciation, du démarrage ou de l'arrêt d'un **Instrument**. Bien entendu cet **Instrument** doit être implémenté au préalable.

```
public class SonarImpl
    implements Sonar, SonarServices, InstrumentDriver, ComponentState {
```



5.2 Le graphisme

5.2.1 Module navisu-app

Dessiner l'icône de l'item : à placer dans :
bzh/terrevirtuelle/navisu/app/guiagent/dock/impl/dock_icons des ressources



FIGURE 5.1 – L'item Instruments

5.2.2 Module navisu-widgets

Dessiner les icônes de l'item central du RadialMenu et de ses sous-items : à placer dans : bzh.terrevirtuelle.navisu.widgets.radialmenu.menu des ressources.

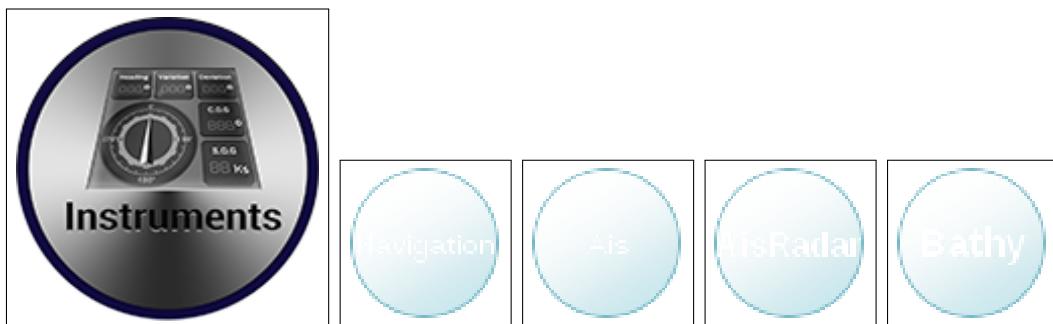


FIGURE 5.2 – L'icône centrale de Instruments et les sous-items

5.3 Le code

5.3.1 Module navisu-app

Dans la classe DockManagerImpl du module navisu-app, ajouter un item au Dock :

```
public final DockItem[] ICONS = new DockItem[] {
    DockItemFactory.newImageItem("instruments",
        ICON_PATH + "dock_icons/instruments.png",
        (e) -> {
            instrumentsRadialMenu.setVisible(!instrumentsRadialMenu.isVisible());
        }),
    ...
}
```



Ici "instruments" correspond à l'infobulle associée, ICON_PATH+"dock_icons/instruments.png" est le chemin de l'image dans le répertoire resources. Le dernier argument du constructeur est le callback associé, généralement celui là. Appeler la méthode de création du RadialMenu associé :

```
@Override  
public void makeDock() {  
    createDockWidget(scene);  
    createBooksRadialWidget();  
    createChartsRadialWidget();  
  
    createInstrumentsRadialWidget();  
  
    createMeteoRadialWidget();  
    createTidesRadialWidget();  
    createToolsRadialWidget();  
    createNavigationRadialWidget();  
}
```

Coder cette méthode : le menu radial est créé à l'aide d'un RadialMenuBuilder

```
//-----INSTRUMENTS-----  
private void createInstrumentsRadialWidget() {  
    instrumentsRadialMenu = RadialMenuBuilder.create()  
        .centralImage("instrumentsradialmenu.png")  
        .createNode(0, "navigation.png", 0, "ais.png", 0, "aisradar.png",  
                   (e) -> open("AisRadar"))  
        .createNode(0, "navigation.png", 1, "bathy.png", 0, "sonarOn.png",  
                   (e) -> open("Sonar"))  
        .build();  
  
    instrumentsRadialMenu.setLayoutX((width / 2) - 40);  
    instrumentsRadialMenu.setLayoutY(height / 2);  
    root.getChildren().add(instrumentsRadialMenu);  
  
    radialMenus.add(instrumentsRadialMenu);  
}
```

- La méthode `createNode` va préciser pour chaque item, son placement, ses images associées et son callback.
- Un choix d'ergonomie a été fait, les menus radiaux ont deux couches de sous-menus puis des feuilles, les feuilles correspondent aux actions.
- Dans la première couche les segments sont numérotés : 0, 1, 2, ... Dans l'exemple un seul segment, son icône est `navigation.png`
- Dans la deuxième couche, idem, les segments sont numérotés : 0, 1, 2, ... Dans l'exemple



deux segments, les icônes `ais.png` et `bathy.png`

- Chaque segment reçoit les feuilles correspondant aux items, ici un item par segment.
les images `aisradar.png` et `sonarOn.png` respectivement.
- Enfin, le dernier argument correspond au callback associé à cet item.

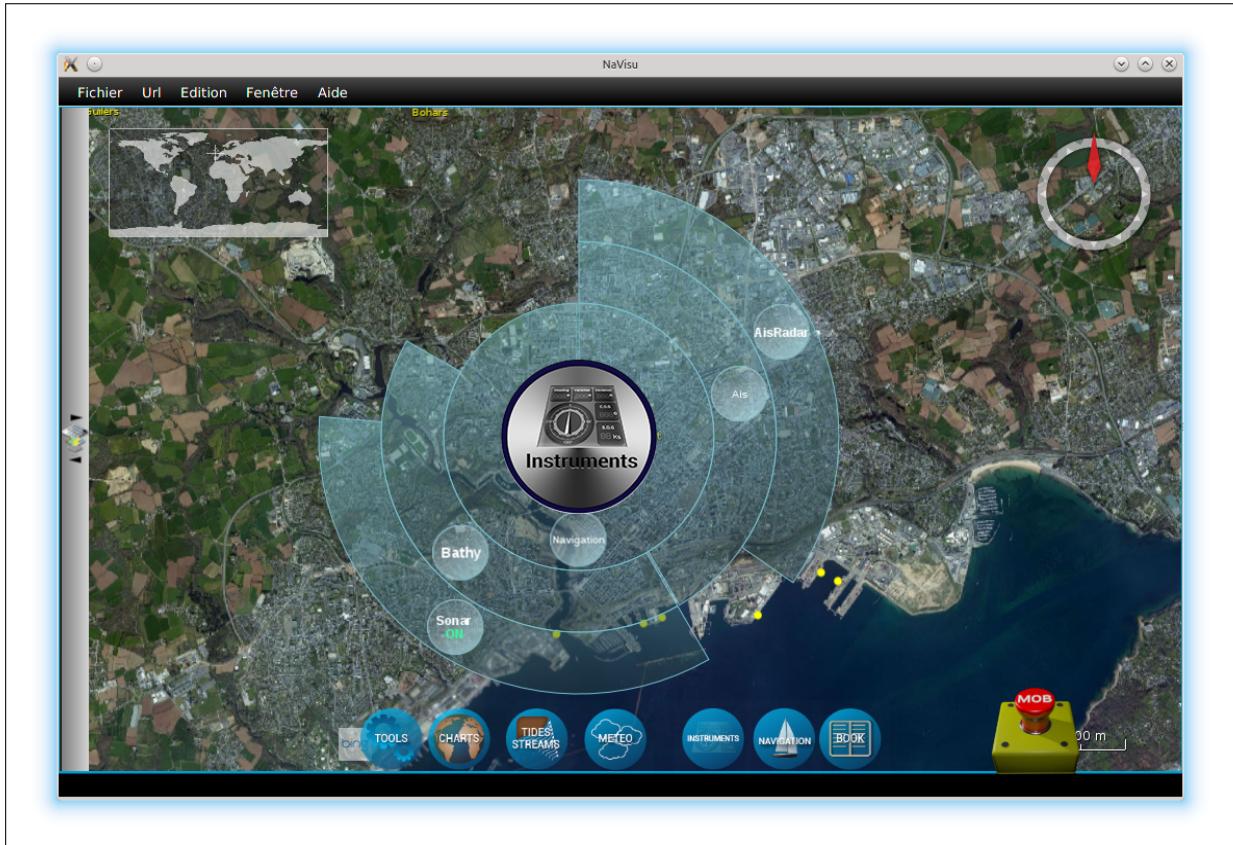


FIGURE 5.3 – *Le menu Instruments*

5.3.2 Choix des callbacks

La plupart du temps les callbacks associés aux items n'ont pas besoin d'être codés, il sont fournis par NaVisu. Ci après la liste non exhaustive des callbacks et des cas d'utilisation.

- Cas d'un instrument : l'argument correspond au nom de l'instrument à activer.

```
private void open(String keyName) {  
    instrumentDriverManagerServices.open(keyName);  
    clear();  
}
```



- Cas d'ouverture d'un fichier : le premier argument est le KEY_NAME du Driver sachant interpréter ces fichiers : Sedimentology, Currents, ... Le deuxième argument le ou les extensions des fichiers : .shp, .SHP ou .000 par exemple.

```
private void open(String description, String... des) {  
    String[] tab = new String[des.length];  
    int i = 0;  
    for (String s : des) {  
        tab[i] = "*" + s;  
        i++;  
    }  
    driverManagerServices.open(description, tab);  
    clear();  
}
```

- Connexion à une base de données :

```
private void openDB(String dbName, String hostName, String protocol, String port,  
String driverName, String userName, String passwd) {  
    databaseDriverManagerServices.connect(dbName, hostName, protocol,  
                                         port, driverName, userName, passwd);  
    clear();  
}
```

NAVISU gère autant de bases de données qu'il est nécessaire, elles doivent être installées au préalable, sauf pour une base embarquée. C'est l'API JDBC qui est exhibée comme un ensemble de services.

5.3.3 Module navisu-widgets

Pas de code à écrire, il faut simplement placer les images correspondant au menu et à ses items, dans le fichier **resources**.

5.4 Principe des DriverManager

Comme il a été dit dans la préface, les services accessibles à partir du menu doivent s'enregistrer, voici l'explication. Le premier principe respecté est **l'inversion de dépendance** : la logique de l'application, contenue dans le module **navisu-app** ne doit pas dépendre des modules quelle active :

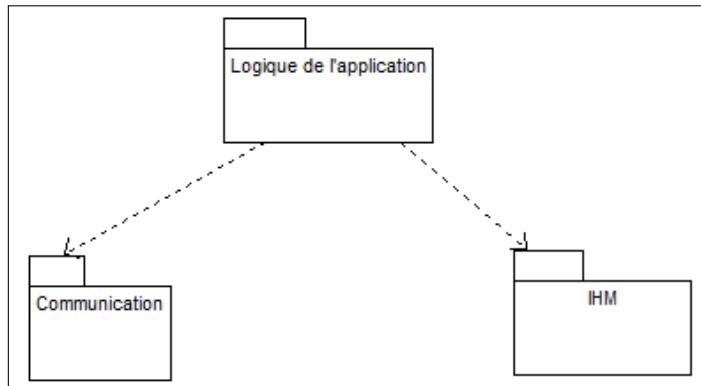


FIGURE 5.4 – *Une modification de l'IHM va impacter l'application*

La logique de l'application doit s'appuyer sur des services :

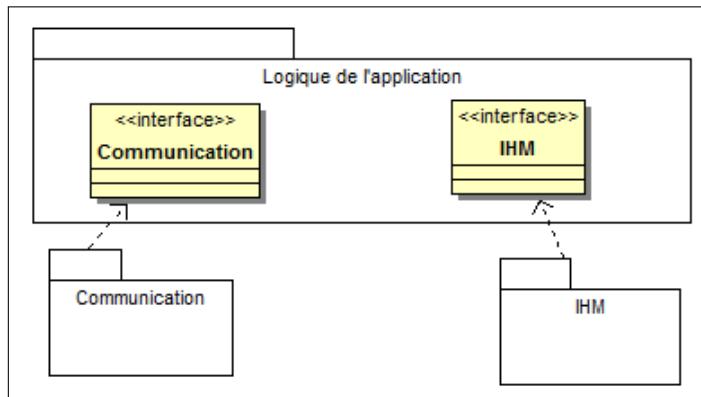


FIGURE 5.5 – *Si les contrats de services sont respectés, un changement du code de l'IHM ne concerne pas l'application*

Le deuxième principe est celui de **pas de dépendances cycliques** :

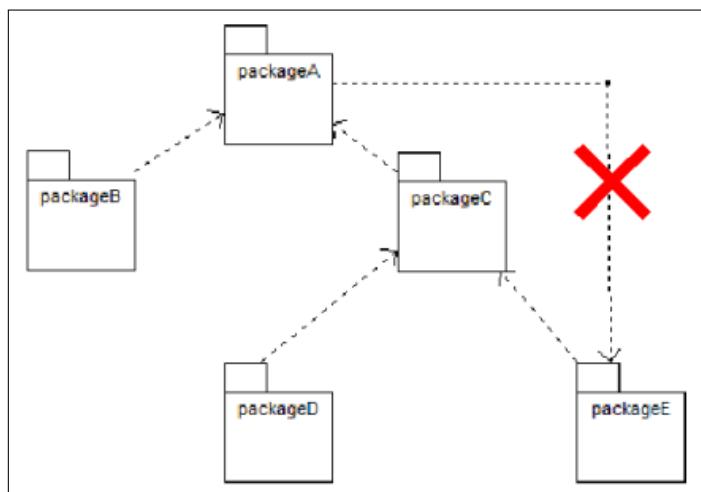


FIGURE 5.6 – *Gradle détectera un tel problème*



Dans chaque fichier `build.gradle` on trouve les dépendances du module, par exemple dans le sous-projet `navisu-instruments`

```
dependencies {
    compile project(':navisu-core')
    compile project(':navisu-client')
    compile project(':navisu-domain')
    compile project(':navisu-app')
    compile project(':navisu-bathymetry')
    compile fileTree(dir: 'lib', include: '*.jar')
}
```

Si le module `navisu-app` devait instancier un `Instrument` directement, il devrait connaître, donc dépendre du module `navisu-instruments` ce qui constitue une rupture du principe numéro deux. Au lieu de cela les différents `Driver` s'inscrivent auprès de leur `DriverManager` spécifique, ensuite lors d'une requête lancée par les callbacks du dock, le `Driver` répondant aux critères de sélection est activé.

```
// class InstrumentDriverManagerImpl

protected List<InstrumentDriver> availableDriverList = new ArrayList<>();

@Override
public void registerNewDriver(InstrumentDriver driver) {
    Checker.notNull(driver, "Driver must not be null.");
    this.availableDriverList.add(driver);
}

@Override
public void open(String category) {
    InstrumentDriver driver = findDriver(category);
    if (driver != null) {
        driver.on();
    } else{
        System.out.println("Unrecognized instrument");
    }
}

protected InstrumentDriver findDriver(String category) {
    InstrumentDriver compatibleDriver = null;
    for (InstrumentDriver driver : this.availableDriverList) {
        if (driver.canOpen(category)) {
            compatibleDriver = driver;
            break;
        }
    }
    return compatibleDriver;
}
```



Exemple pour la classe Sonar les méthodes répondant au DriverManager

```
// public class SonarImpl
    implements Sonar, SonarServices, InstrumentDriver, ComponentState {

    private final String NAME = "Sonar";

    @Override
    public boolean canOpen(String category) {
        return category.equals(NAME);
    }
    @Override
    public void on() {
        ...
    }
}
```

Chapitre 6

Cartographie

6.1 Présentation

NAVISU utilise les cartes raster BSB/KAP. Avant d'utiliser vos cartes il est nécessaire d'opérer un pré-traitement, c'est l'opération de tuilage : la carte au format KAP, de 1 à plusieurs Mo, sera transformée en plusieurs sous images ou tuiles. Les répertoires où sont placés les cartes et les tuiles doivent être renseignés dans l'application.

6.1.1 La projection Equirectangulaire

NAVISU, s'appuyant sur WorldWind, utilise une projection cylindrique simple (appelée projection Equirectangulaire ou Plate Carrée) avec comme référentiel géodésique le WGS84. Dans la projection Equirectangulaire l'intersection avec les méridiens se fait à angle droit, mais contrairement à la projection Mercator, les parallèles sont des lignes droites équidistantes.

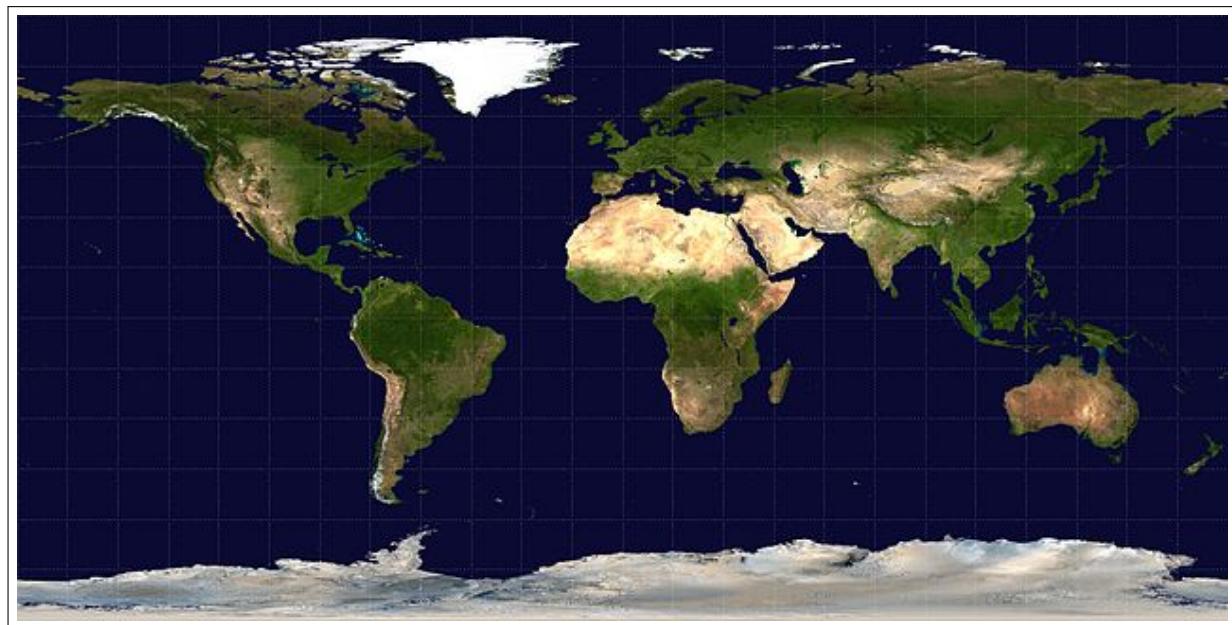


FIGURE 6.1 – *Projection équirectangulaire*

Pour plus d'informations sur le sujet consulter, par exemple, le site du “Intergovernmental Committee on Surveying and Mapping” (ICSM) :



http://www.icsm.gov.au/mapping/about_projections.html

Et pour un exposé plus large des problèmes de projection :

<http://kartoweb.itc.nl/geometrics/>

6.1.2 Tuilage

Le principe

Comme la plupart des applications géoréférencées (GoogleEarth, Yahoo, Bing, ...) pour fournir un niveau de détail important et une bonne fluidité, NAVISU utilise la méthode du tuilage des cartes. Les images affichées sont de 256x256 pixels. Elles sont choisies suivant le niveau de zoom et l'emplacement géographique, l'affichage à l'écran est recréé dynamiquement. Chaque carte est constituée d'un répertoire, puis de nombreux sous-répertoires numérotés comme suit :

Level	Resolution	Number of Tiles
Level 0	36 degrees	50 tiles
Level 1	18 degrees	200 tiles
Level 2	9 degrees	800 tiles
Level 3	4.5 degrees	3200 tiles
Level n		

FIGURE 6.2 – Conventions de la numérotation des tuiles WorldWind

Lors d'un premier accès ces images peuvent provenir de différents sites internet, elles sont mises en cache sur le disque utilisateur. Lors des accès suivants l'application commence par chercher dans le cache. Elles peuvent être dès le départ être mise dans le cache.

Sous Windows 7 par défaut WorldWind utilise le répertoire :



C:\ProgramData\WorldWindData

Pour plus de détails sur le tuilage, voir :

http://www.worldwindcentral.com/wiki/Tiling_System

6.2 Cartographie raster

6.2.1 Les cartes BSB/KAP

L'utilisateur peut définir son propre répertoire pour passer facilement d'un ordinateur à un autre, sans avoir à recopier les données.

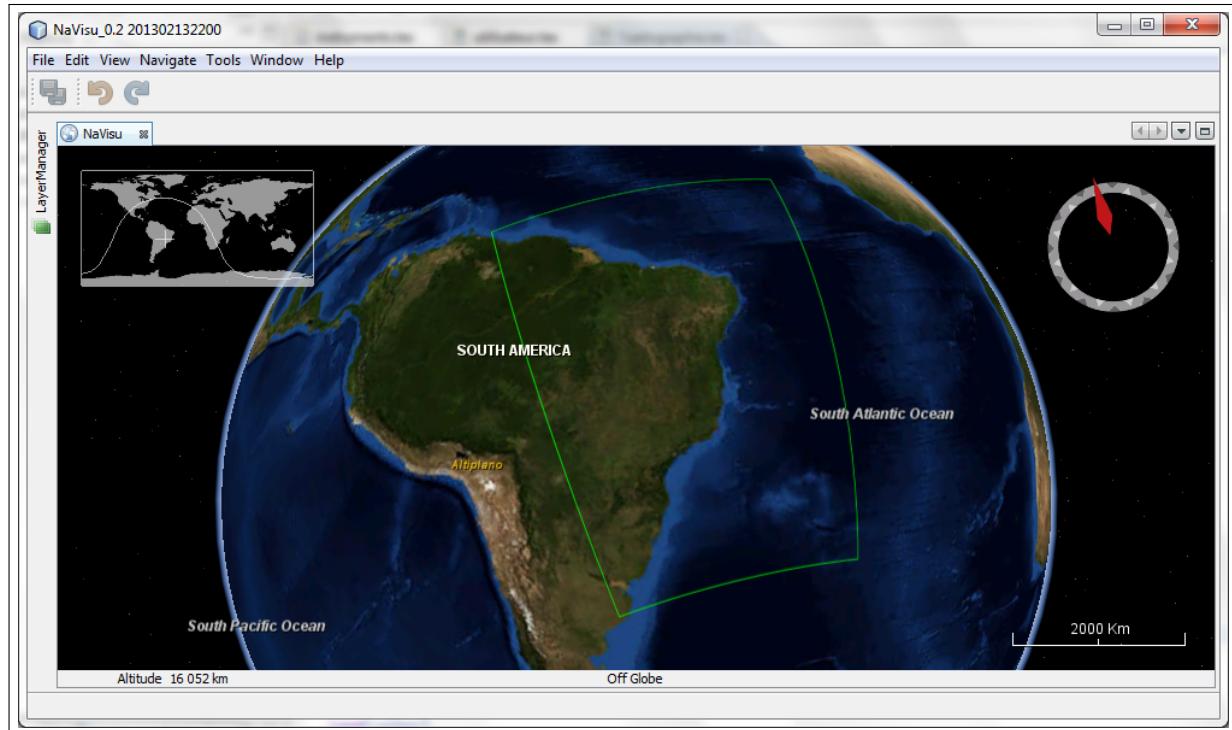
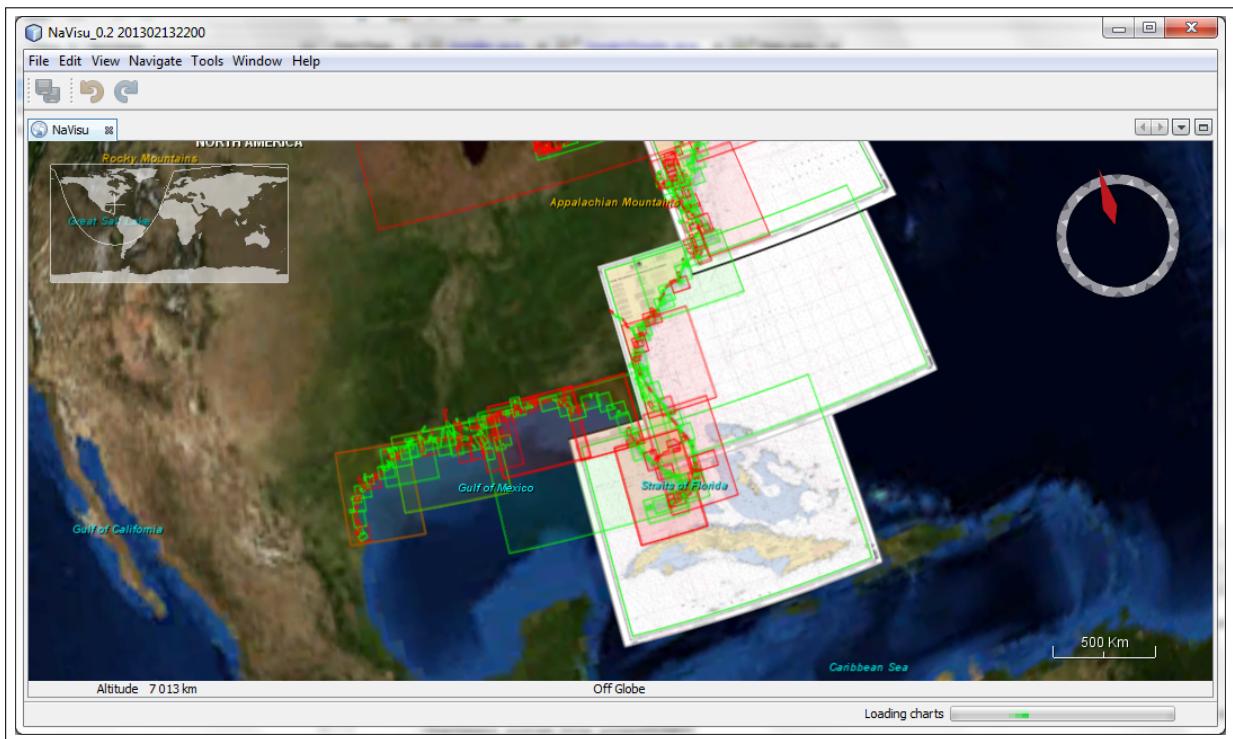
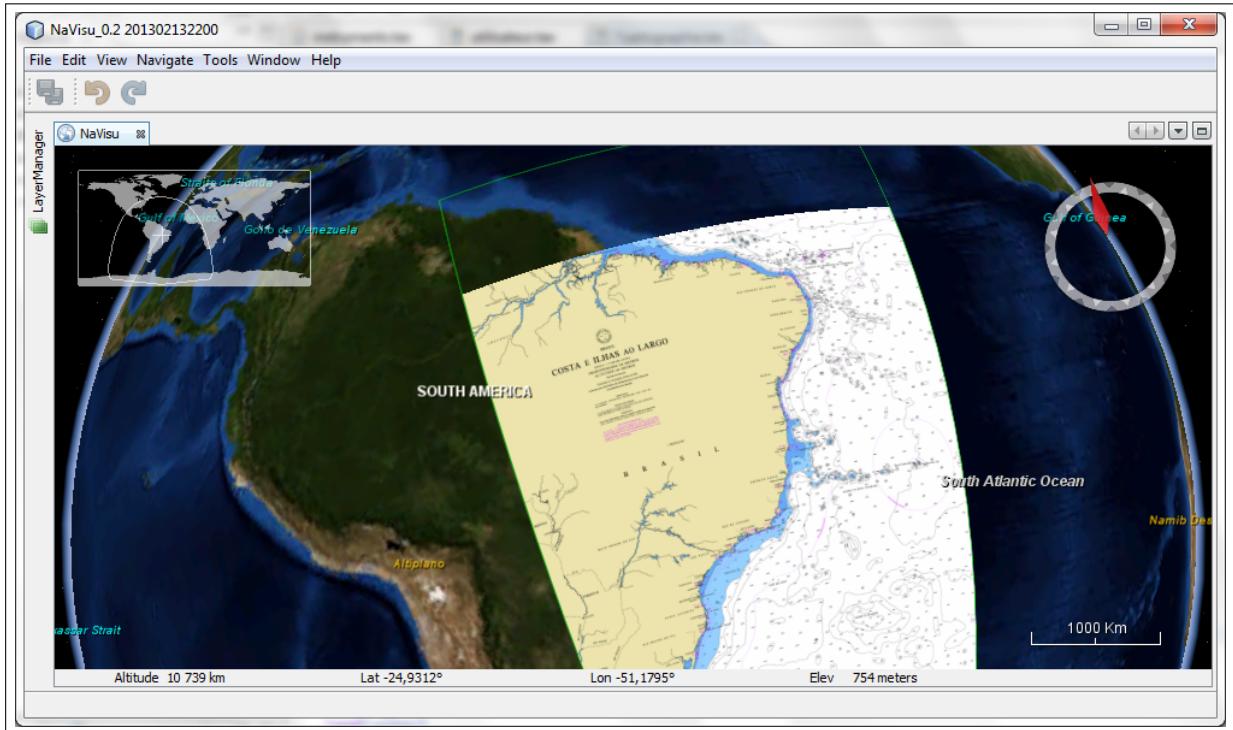


FIGURE 6.3 – Frontières d'une carte

Les frontières sont en vert : la carte 101.KAP est dans le répertoire désigné et la carte a été tuilée, les tuiles sont dans le cache désigné. Si les tuiles ne sont pas dans le cache les polygones frontière sont rouges.

FIGURE 6.4 – *Les Etats Unis*FIGURE 6.5 – *Affichage de la carte*

Double clic à l'intérieur du polygone : affichage de la carte, dès que l'altitude le permet. Ici la distance fait que les tuiles les plus hautes en latitude ne s'affichent pas encore.

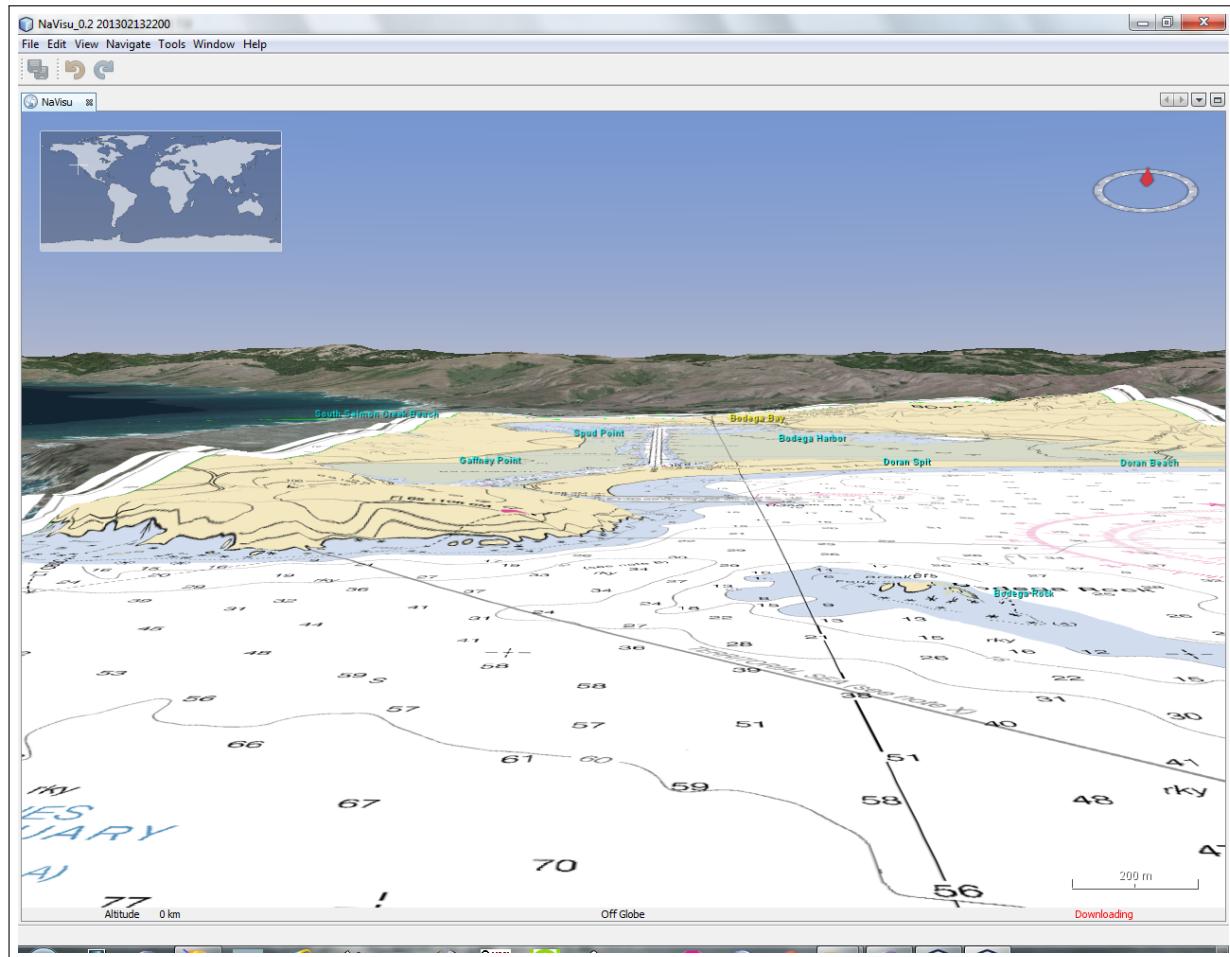


FIGURE 6.6 – Niveaux de détail et la 3D

6.2.2 Acquisition

Quelques sites pour obtenir des cartes raster au format BSB/KAP :

<http://marine.geogarage.com/>

<http://www.justmagic.com/RasterChart2BSB.html>

https://www.mar.mil.br/dhn/chm/cartas/download/cartasbsb/cartas_eletronicas_Internet.htm

<http://www.nauticalcharts.noaa.gov/mcd/Raster/index.htm>

<http://www.1yachtua.com/>