

MASTER 2 – ISIDIS

Projet de synthèse

V.A.L.

**Spécifications Fonctionnelles**

Sommaire

[I. Glossaire 3](#_Toc343154453)

[II. Use Case Model 3](#_Toc343154454)

[1) Les acteurs 3](#_Toc343154455)

[2) Le diagramme 4](#_Toc343154456)

[3) Description des Use Cases 5](#_Toc343154457)

[III. Décomposition des Use Cases 6](#_Toc343154458)

[1) Use Case 1 : Collecter les informations terrains 6](#_Toc343154459)

[A. Use Case Detail : 6](#_Toc343154460)

[B. Diagramme de séquence système : 6](#_Toc343154461)

[C. Diagramme de classes participantes : 7](#_Toc343154462)

[D. Conception détaillée : 7](#_Toc343154463)

[2) Use Case 2 : Prioriser messages 8](#_Toc343154464)

[A. Use Case Detail : 8](#_Toc343154465)

[B. Diagramme de séquences système : 9](#_Toc343154466)

[C. Diagramme de classes participantes : 9](#_Toc343154467)

[D. Conception détaillée : 10](#_Toc343154468)

[3) Use Case 3 : Orchestrer les messages 10](#_Toc343154469)

[4) Use Case 4 : Configurer le démarrage des services 10](#_Toc343154470)

[5) Use Case 5 : Transmettre les messages au terrain 10](#_Toc343154471)

[6) Use Case 6 : Effectuer les interventions automatiques 10](#_Toc343154472)

[7) Use Case 7 : Expand messages 10](#_Toc343154473)

1. Glossaire

|  |  |
| --- | --- |
| RTDG | C’est ce qui représente notre [S.U.D](#SUD), c’est le système qui fait la transmission des messages entre le [RTDRS](#RTDRS) et le [composant embarqué](#composant_embarque), il envoie aussi des contre-mesures au [composant embarqué](#composant_embarque) |
| RTDRS | C’est le système qui représente le centre de contrôle développé par l’équipe ISIAD, ce composant fournit la représentation du réseau ferré, le planning, les messages d’information... etc |
| S.U.D | System Under Design, représente notre périmètre applicatif, c’est le composant qui va permettre le pilotage du réseau en temps réel, la remontée d’informations des équipements terrain |
| Référentiel | Emplacement où sont stockées les informations dont a besoin notre système pour fonctionner, décrivant entre autre la structure des messages reçus du terrain pour pouvoir les interpréter, connaitre la criticité des messages, ... etc. |
| Equipements terrain | Ce sont toutes les sources d’informations du terrain, comprenant les composants terrain et les composants embarqués |
| Composant terrain | ce sont toutes les puces électroniques implantées sur les tunnels, sur les rails et sur les stations |
| composants embarqués | Les puces électroniques qui sont implantées sur les rames |
| MOM | Message Oriented Middleware, c’est un framework qui permet la communication entre des applications via un réseau informatique, il permet un couplage faible entre les applications. |
| ActiveMQ | Une implémentation du JMS (Java Message Service) |
| framework de simulation | Système qui simulera le travail que font les [composants embarqués](#composant_embarque) en générant des messages aléatoires (critiques ou non) qui seront passés au composant [RTDG](#RTDG) |
| EFIC | [Composant embarqué](#composant_embarque) qui collecte les messages des [composants embarqués](#composant_embarque) et l’envoie au [RTDG](#RTDG) |
| EHQMR | [Composant embarqué](#composant_embarque) qui reçoit les messages envoyés par le [RTDG](#RTDG) |
| Buffers | Structures où seront stockés les messages reçus des [composants embarqués](#composant_embarque) selon la criticité |
| Criticité | critère qui nous dit quel message devons-nous traiter en premier lieu |

1. Use Case Model
2. Les acteurs

En lisant le sujet du projet de synthèse et en faisant la conception préliminaire, on a pu trouver 3 acteurs qui interviennent dans notre [S.U.D](#SUD):

* Admin\_RTDG : Cet acteur est celui qui administre notre système [RTDG](#RTDG), son rôle est de démarrer et arrêter le système et de configurer les règles de gestion de ce système, c’est donc à lui que revient la charge de mettre en place les [référentiels](#Réferentiel) qui seront dans la base de données, il peut à tout moment modifier, ajouter ou supprimer des informations sur ces [référentiels](#Réferentiel) tel les adresses MAC de chaque composant ou la contre-mesure à effectuer pour un message donné.
* [RTDRS](#RTDRS) : Cet acteur représente pour nous le centre de contrôle, c’est lui qui nous fournit les informations concernant l’état du trafic, on lui transmet quelques messages qu’on reçoit depuis les [composants terrain](#Composant_terrain), et il nous fournit les contre-mesures à renvoyer au terrain, on lui envoie aussi des notifications des contre-mesures qu’on fait sans le solliciter, la communication avec cet acteur se fait via un [MOM](#mom) appelé [ActiveMQ](#activemq).
* Composant\_Embarqué : Cet acteur représente pour nous les [composants embarqués](#composant_embarque) contenus dans les rames des métros, ils nous enverront les messages que nous devons traiter, dans notre projet, à défaut d’avoir de vrais [composants embarqués](#composant_embarque), nous réaliserons un [framework de simulation](#framework) qui réalisera ce travail, nous auront alors un composant appelé [EFIC](#efic) qui collecte les informations depuis les capteurs embarqués et les envoie à notre système [RTDG](#RTDG) et un autre composant appelé [EHQMR](#ehqmr) qui réceptionnera les messages qu’on lui enverra sous forme d’ordre pour qu’il les effectue.

1. Le diagramme



1. Description des Use Cases

|  |  |
| --- | --- |
| **Nom** | **Description** |
| Collecter les informations terrains | Cet UC a comme rôle la réception de tout type d'informations terrain à partir des différents [capteurs embarqués](#composant_embarque) et qui nous seront envoyés par le composant [EFIC](#efic) |
| Prioriser messages | Dans cet UC on gère les messages selon un [référentiel](#Réferentiel), on met chaque message dans le buffer adéquat, sachant qu'on a plusieurs [buffers](#buffer) selon la [criticité](#criticité) du message. Après ceci, on priorisera nos messages pour les traiter selon une stratégie qu'on aura préalablement défini. |
| Orchestrer les messages | Cet UC permet la gestion de la communication des messages provenant des centres de contrôles vers notre système d'information. |
| Configurer le démarrage des services | Cet UC permet de faire un chargement total des informations de la base de données en mémoire pour démarrer notre système d'information, ces informations  seront mises dans des [référentiels](#Réferentiel), tel celui des adresses MAC de chaque composant ou celui des contre-mesures relatives à chaque incident. |
| Transmettre les messages au terrain | Cet UC a comme rôle la transmission des messages vers un [composant embarqué](#composant_embarque) identifié par son adresse MAC suivant un [référentiel](#Réferentiel). |
| Effectuer les interventions automatiques | Permet de traiter les messages et de définir les contre mesure et sous actions sur les évènements critiques en utilisant un [référentiel](#Réferentiel) qu'on va définir. |
| Expand messages | C’est au niveau de cet UC qu'on transforme un message fonctionnel en un message technique, car le message qu'on obtiendra du côté du [RTDRS](#RTDRS) reste fonctionnel et il nous faudra alors  l'interpréter pour pouvoir savoir quelle contre-mesure y affecter. |

1. Décomposition des Use Cases
2. Use Case 1 : Collecter les informations terrains
3. Use Case Detail :

Acteur principal : Composant\_Embarqué

Acteur secondaire : -

Objectifs :

Cet UC a pour objectif, la récupération des informations depuis les [composants embarqués](#composant_embarque) et leur enregistrement dans un [buffer](#buffer) pour  pouvoir les traiter par notre système [RTDG](#RTDG).

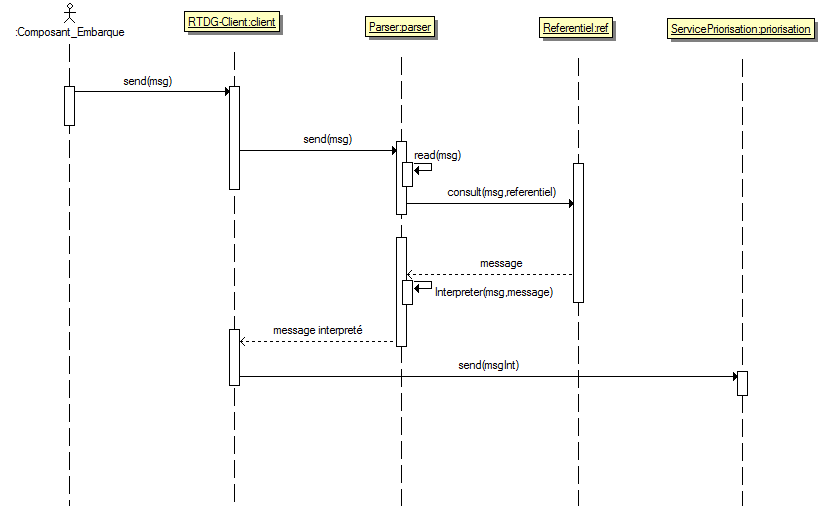
Pré-conditions :

- [Framework de simulation](#framework) doit être démarré

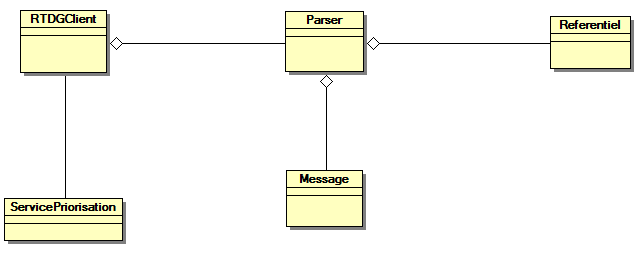
-  [RTDG](#RTDG) doit être en écoute pour pouvoir recevoir les messages depuis les [composants embarqués](#composant_embarque)

Scénario nominal: S-01017 : Interpréter un [message](#message) reçu depuis un [composant embarqué](#composant_embarque) (parser)

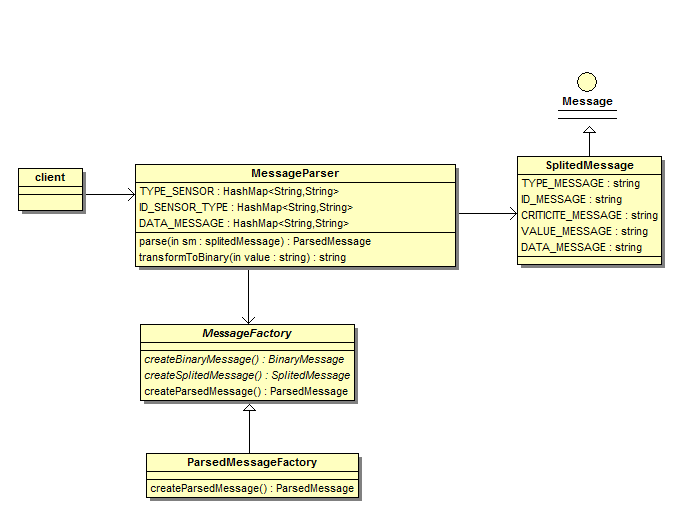
1. le [composant embarqué](#composant_embarque) envoie un message à RTDG-Client
2. le RTDG-Client lit le message et consulte le référentiel
3. le RTDG-Client interprète le message
4. le RTDG-Client envoie le message interprété au service de priorisation
5. Diagramme de séquence système :



1. Diagramme de classes métier :



1. Conception détaillée :
   * Diagramme de classes détaillé



1. Use Case 2 : Prioriser messages
2. Use Case Detail :

Acteur principal : -

Acteur secondaire : Composant\_Embarqué

Objectifs :

L’objectif de ce Use Case et de prioriser les messages reçus pour pouvoir déterminer lequel devra être traité en premier.

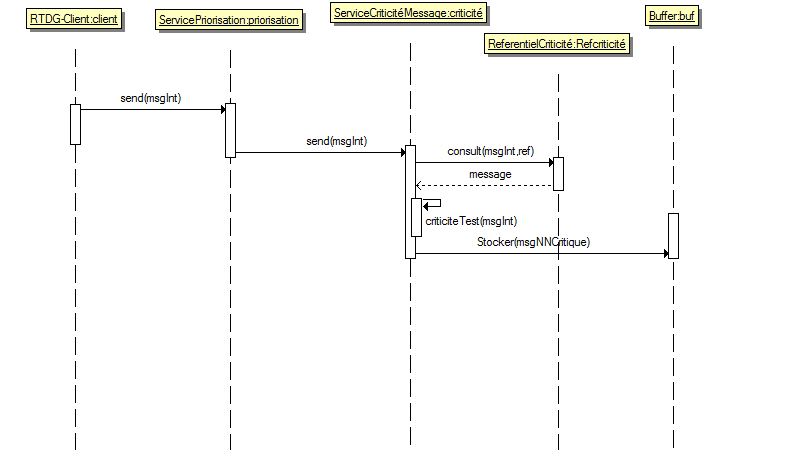
Pré-conditions :

- [Framework de simulation](#framework) doit être démarré

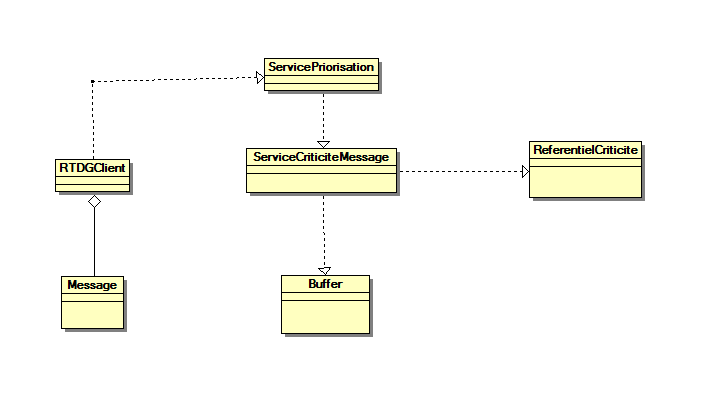
- Les [référentiels](#Réferentiel) doivent avoir été chargés en mémoire

Scénario nominal: S-01020 : affecter un message non critique dans un buffer (nom de scénario à discuter)

1. le service de priorisation reçoit le message interprété depuis RTDG-Client
2. le service de priorisation envoie le message interprété au service de gestion des messages « ServiceCriticiteMessage »
3. Stocker les messages non critique dans un buffer spécifique pour cela
4. Diagramme de séquences système :

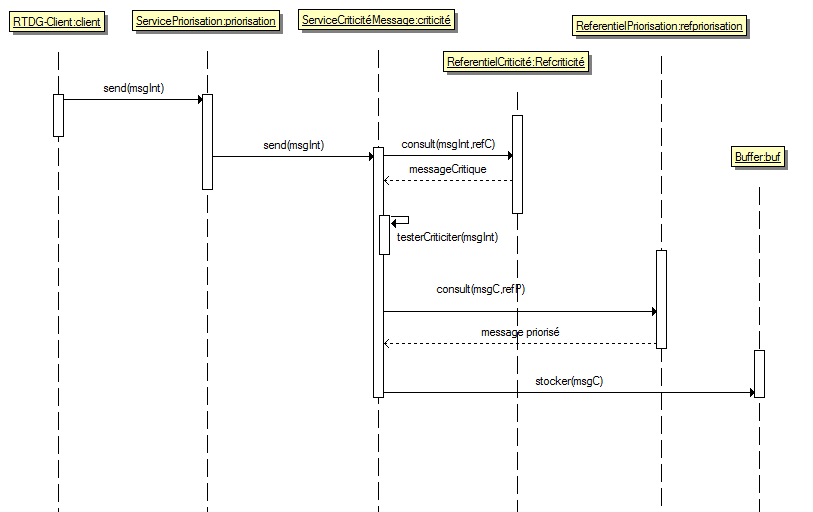


1. Diagramme de classes métiers :

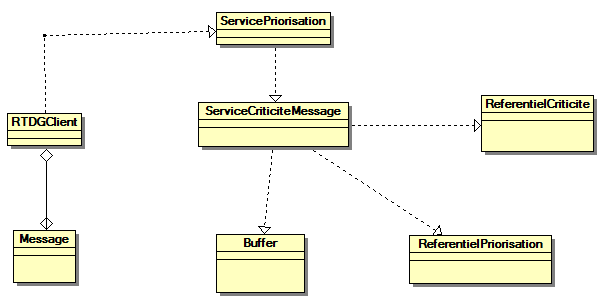


Scénario Alternatif 1: S-01029: Affecter les messages critiques par ordre

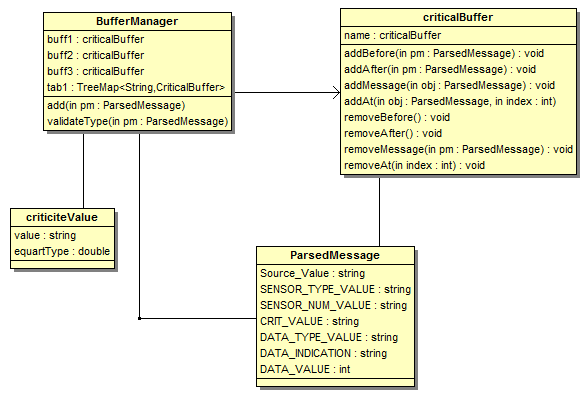
1. le service de priorisation reçoit le message interprété depuis RTDG-Client
2. le service de priorisation envoie le message interprété au service de gestion des messages « ServiceCriticiteMessage »
3. Prioriser le message selon les référentiels
4. Stocker les messages critique dans un buffer spécifique
5. Diagramme de séquences système :



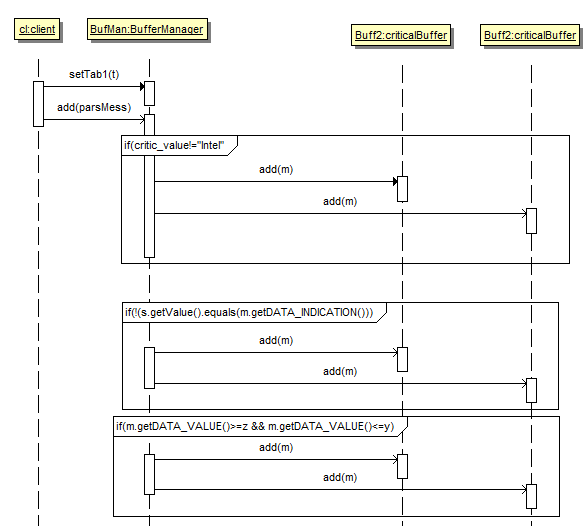
1. Diagramme de classes métiers :



1. Conception détaillée :
   * Diagramme de classes détaillé :



* + Diagramme de séquences détaillé :



1. Hors périmètre
2. S01021 :Création du générateur de données

Dans le cadre de notre projet, il va nous falloir réaliser un Framework de simulation pour reproduire le réseau terrain, ce qui nous permettra de tester l’envoi et la réception des messages entre le RTDG et le [composant embarqué](#composant_embarque) et le traitement de ces messages.

Ce simulateur a deux sources de données différentes :

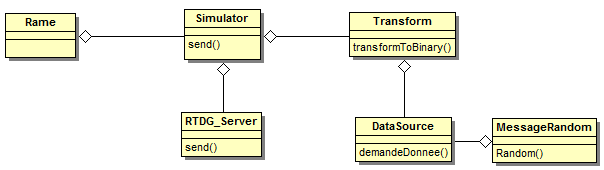
La première est celle où le simulateur génère les données à partir d’un train qui nous envoie les données, que ce soit celles de sa position ou n’importe quelle donnée. Le simulateur crée alors une trame depuis le message reçu et l’envoie au RTDG sous format de tableau de bytes à l’aide du protocole de communication.

La deuxième source de données que peut utiliser notre framework est la génération aléatoire de données depuis un HashMap qui fait office de référentiel et qui sont chargés au lancement du RTDG.

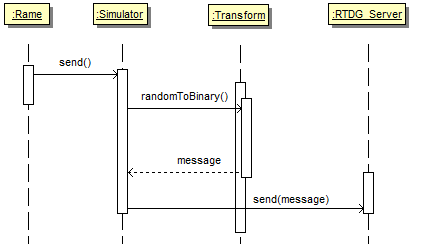
Le simulateur crée alors une trame depuis le message reçu et l’envoie au RTDG sous format de tableau de bytes à l’aide du protocole de communication.

1. Diagrammes de la conception préliminaire

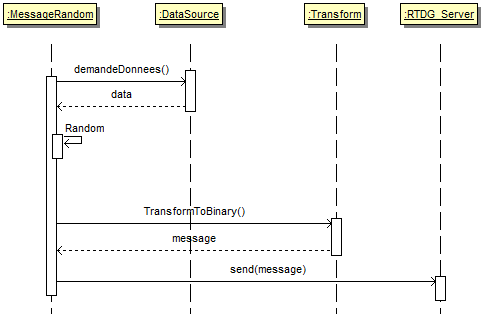
* Diagramme de classes métier



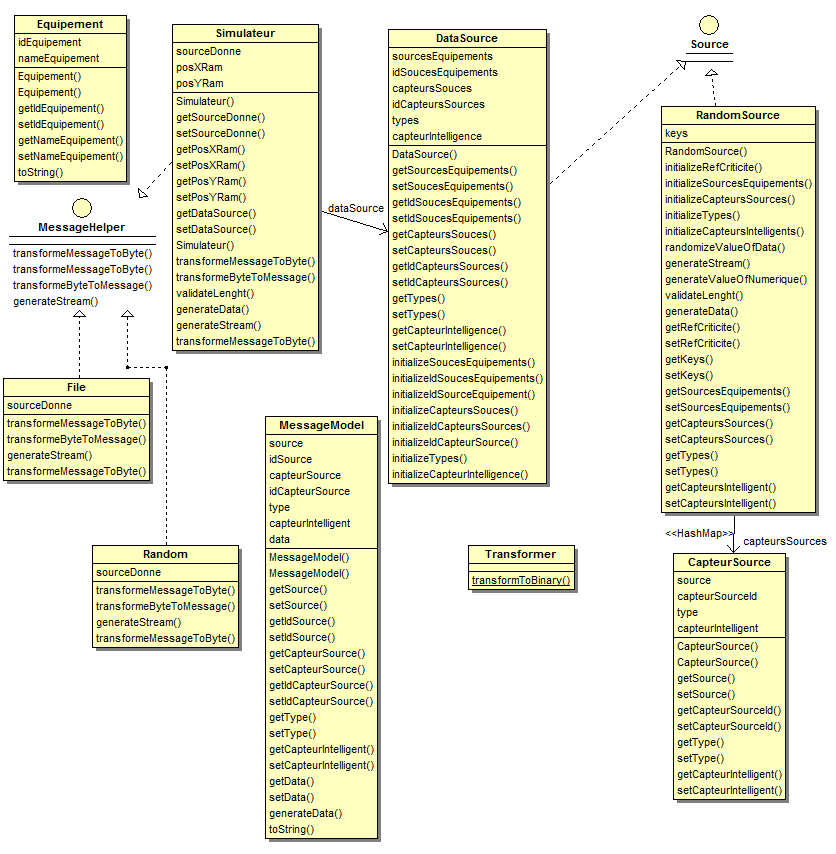
* Diagramme de séquences système (rame)



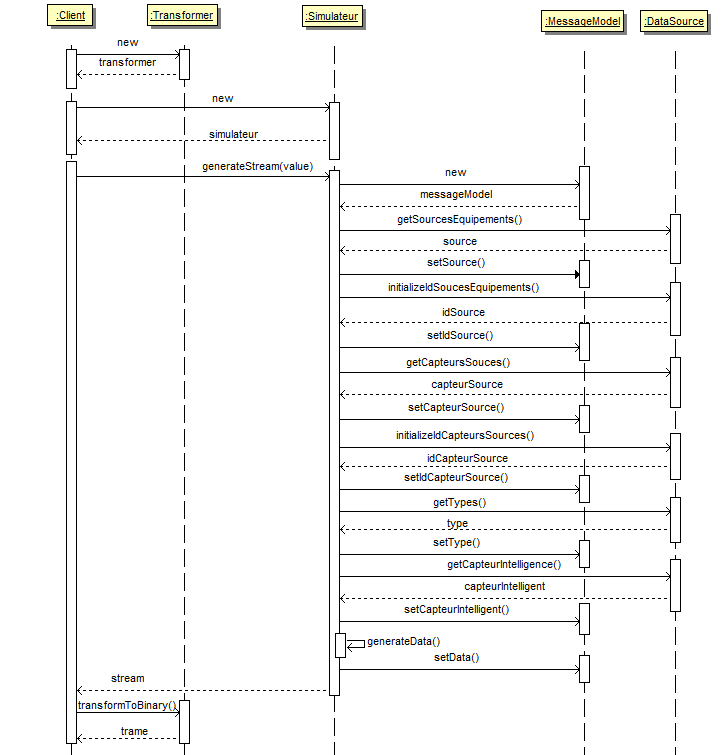
* Diagramme de séquences système (Source de données)



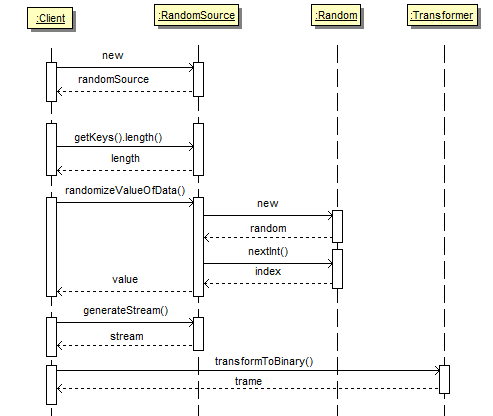
* Diagramme de classe détaillé



* Diagramme de séquence détaillé (rame)

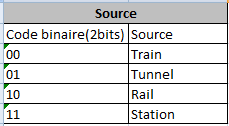


* Diagramme de séquence détaillé (Random)



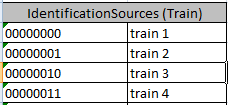
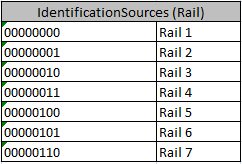
V- Référentiels :

Comme c’est expliqué dans le glossaire, les référentiels sont les emplacements ou sont stockées les informations dont a besoin notre application, dans notre contexte on a un composant embarqué qui envoie un message binaire, pour traduire ce message notre application consulte un référentiel qu’on a définit sur différents types de référentiel, un pour le parser, un pour la criticité et un pour les contres mesures ; le message c’est un message binaire qui est sur 32 bits.



1)

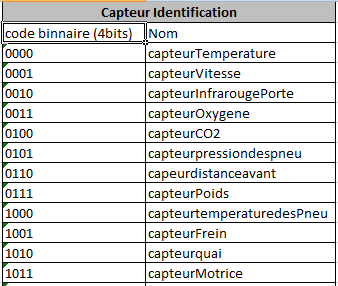
On a 2 bits qui définissent la source des messages comme il est expliqué dans tableau source.



2)

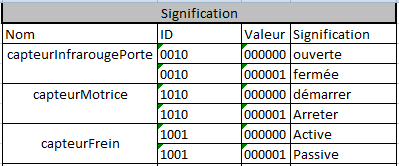
8bits qui définissent du quelle source s’agit il parce qu’il peut exister pour chaque source un certains nombres, la même chose pour les rail, les stations et les tunnels. (Voir les tableaux)

3)

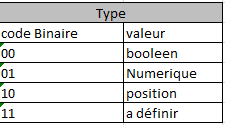


4bits pour l’identification des capteurs qui sera comme suit :

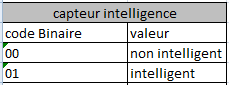
Et les 6bits seront signifié comme suit dans le tableau signification :



4) un autre tableau qui identifie le type des messages sur 2 bits comme suit :



5) les 2 suivants bits seront compté pour identifié les capteurs intelligents et les non intelligents :



Et les 10 derniers bits on les a réservés pour les données que ces capteurs peuvent envoyer.

On a définit un autre référentiel pour la criticité qui peut traduire soit une criticité simple, soit une criticité complexe comme suite :

Dans notre contexte on a des capteurs intelligents qui peuvent envoyés des messages qui se traduits automatiquement en un message critique s’ils valident certains conditions selon le référentiel suivant :

|  |  |  |  |
| --- | --- | --- | --- |
| CriticiteSimple | | | |
| Nom | code binaire (4 bits) | criticité | Ecart-type |
| capteurTemperature | 0000 | 80 | 10 |
| capteurVitesse | 0001 | 60 | 30 |
| capteurInfrarougePorte | 0010 |  |  |
| capteurOxygene | 0011 | 10% | 5% |
| capteurCO2 | 0100 | 10% | 5% |
| capteurpressiondespneu | 0101 | 50 | 10 |
| capeurdistanceavant | 0110 | 50 | 10 |
| capteurPoids | 0111 | 1000 | 100 |
| capteurtemperaturedesPneu | 1000 | 80 | 20 |
| capteurquai | 1001 |  |  |
| capteurMotrice | 1010 |  |  |
|  | 1011 |  |  |
|  | 1100 |  |  |
|  | 1101 |  |  |
|  | 1110 |  |  |
|  | 1111 |  |  |

On a aussi une criticité qu’on a appelée événement complexe, cette criticité se base aussi sur un référentiel qui se traduira en un événement critique, chaque événement se traduit par un certains nombre de donnée envoyés par différents capteurs spécifique pour chaque événement.

