

MASTER 2 – ISIDIS

Projet de synthèse

V.A.L.

**Spécifications Fonctionnelles**

Sommaire

[I. Glossaire 3](#_Toc347243628)

[II. Use Case Model 5](#_Toc347243629)

[1) Les acteurs 5](#_Toc347243630)

[2) Le diagramme 5](#_Toc347243631)

[3) Description des Use Cases 6](#_Toc347243632)

[III. Décomposition des Use Cases 7](#_Toc347243633)

[1) Use Case 1 : Collecter les informations terrains 7](#_Toc347243634)

[A. Use Case Detail : 7](#_Toc347243635)

[B. Diagramme de séquence système : 8](#_Toc347243636)

[C. Diagramme de classes métier : 8](#_Toc347243637)

[D. Conception détaillée : 9](#_Toc347243638)

[2) Use Case 2 : Prioriser messages 10](#_Toc347243639)

[A. Use Case Detail : 10](#_Toc347243640)

[B. Scénario nominal: S-01020 : affecter les messages non critiques à un buffer 10](#_Toc347243641)

[a) Diagramme de séquences système : 10](#_Toc347243642)

[b) Diagramme de classes métiers : 11](#_Toc347243643)

[C. Scénario Alternatif 1: S-01029: Affecter les messages critiques aux buffers par ordre 11](#_Toc347243644)

[a) Diagramme de séquences système : 12](#_Toc347243645)

[b) Diagramme de classes métiers : 12](#_Toc347243646)

[c) Conception détaillée : 13](#_Toc347243647)

[IV. Hors périmètre 14](#_Toc347243648)

[1) S01021 :Création du générateur de données 14](#_Toc347243649)

[A. Diagrammes de la conception préliminaire 15](#_Toc347243650)

[B. Diagrammes de la conception détaillée 16](#_Toc347243651)

[V. Référentiels : 19](#_Toc347243652)

1. Glossaire

|  |  |
| --- | --- |
| RTDG | C’est ce qui représente notre [S.U.D](#SUD), c’est le système qui fait la transmission des [messages](#message) 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](#Equipement_terrain) |
| Référentiel | Emplacement où sont stockées les informations dont a besoin notre système pour fonctionner, décrivant entre autre la structure des [messages](#message) reçus du terrain pour pouvoir les interpréter, connaitre la [criticité](#criticité) des [messages](#message), ... etc. |
| Equipements terrain | Ce sont toutes les sources d’informations du terrain, comprenant les [composants terrain](#Composant_terrain) et les [composants embarqués](#composant_embarque) |
| 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](#framework_de_simulation) 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](#message) aléatoires (critiques ou non) qui seront passés au composant [RTDG](#RTDG) |
| EFIC | [Composant embarqué](#composant_embarque) qui collecte les [messages](#message) des [composants embarqués](#composant_embarque) et l’envoie au [RTDG](#RTDG) |
| EHQMR | [Composant embarqué](#composant_embarque) qui reçoit les [messages](#message) envoyés par le [RTDG](#RTDG) |
| Buffers | Structures où seront stockés les [messages](#message) reçus des [composants embarqués](#composant_embarque) selon la [criticité](#criticité) |
| Criticité | critère qui nous dit quel [message](#message) devons-nous traiter en premier lieu |
| Message | Trame de bits, contenant les informations transmises entre le [composant embarqué](#composant_embarque) et le composant [RTDG](#RTDG) |
| Protocole de communication | Protocole permettant la communication entre le [RTDG](#RTDG) et le [composant embarqué](#composant_embarque) |
| Parser | Composant qui reçoit le message découpé, et associe chaque partie de ce [message](#message) à ce dont il correspond selon les [référentiels](#Référentiel) |
| Capteurs intelligents | Ce sont des [capteurs](#composant_embarque) qui ne nous envoient que des [messages](#message) critiques |

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](#Référentiel) qui seront dans la base de données, il peut à tout moment modifier, ajouter ou supprimer des informations sur ces [référentiels](#Référentiel) tel les adresses MAC de chaque composant ou la contre-mesure à effectuer pour un [message](#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](#message) 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](#composant_embarque): Cet acteur représente pour nous les [composants embarqués](#composant_embarque) contenus dans les rames des métros, ils nous enverront les [messages](#message) que nous devons traiter, dans notre projet, à défaut d’avoir de vrais [composants embarqués](#composant_embarque), nous réaliserons un [framework de simulation](#framework_de_simulation) 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](#message) 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](#message) | Dans cet UC on gère les [messages](#message) selon un [référentiel](#Référentiel), on met chaque [message](#message) dans le [buffer](#buffer) adéquat, sachant qu'on a plusieurs [buffers](#buffer) selon la [criticité](#criticité) du [message](#message). Après ceci, on priorisera nos [messages](#message) 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](#message) vers un [composant embarqué](#composant_embarque) identifié par son adresse MAC suivant un [référentiel](#Référentiel). |
| 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](#Référentiel) 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).

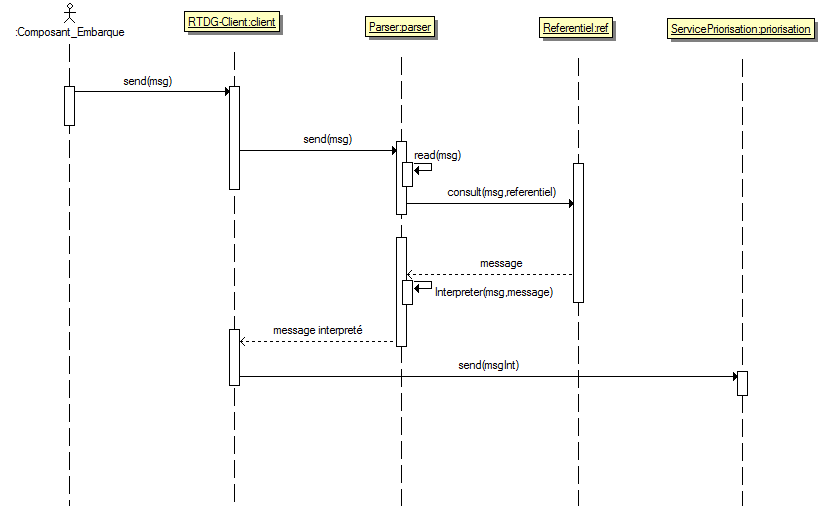
Pré-conditions :

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

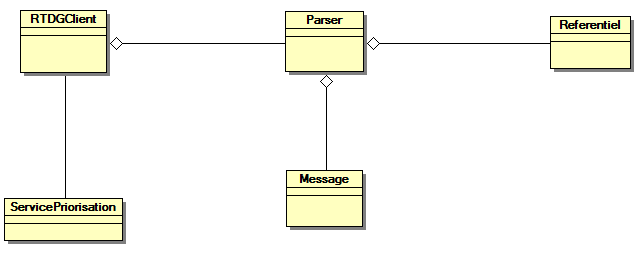
-  [RTDG](#RTDG) doit être en écoute pour pouvoir recevoir les messages depuis les [composants embarqués](#composant_embarque)

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

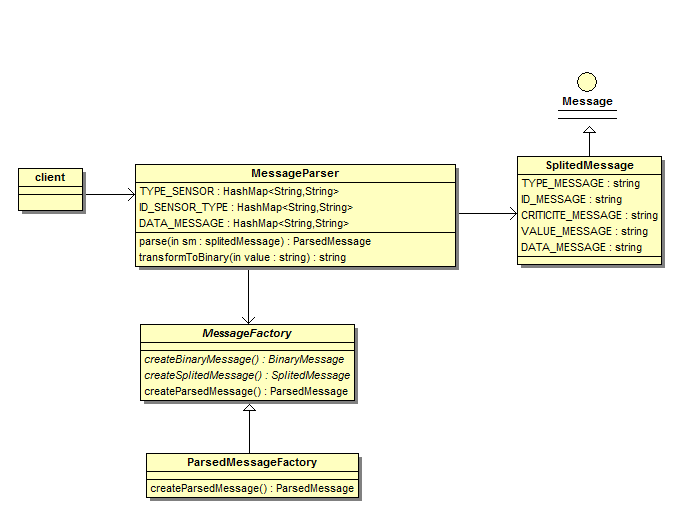
1. le [composant embarqué](#composant_embarque) envoie un [message](#message) à RTDG-Client
2. le RTDG-Client lit le message et consulte le [référentiel](#Référentiel)
3. le RTDG-Client interprète le [message](#message)
4. le RTDG-Client envoie le [message](#message) interprété au service de priorisation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

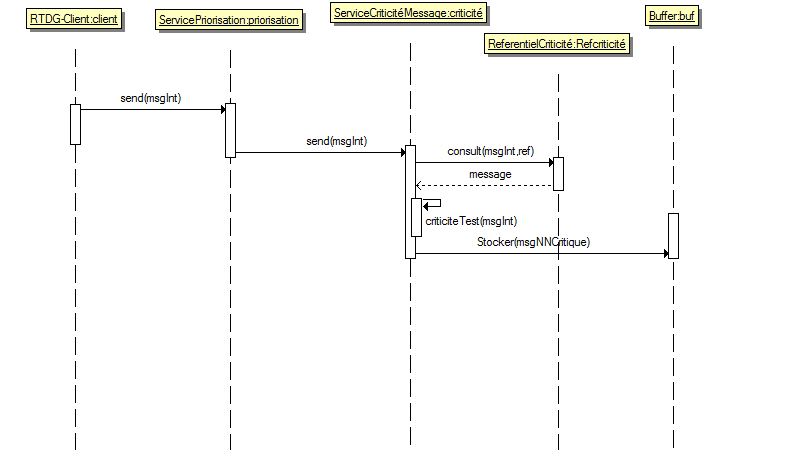
L’objectif de ce Use Case est de prioriser les [messages](#message) reçus pour pouvoir déterminer lequel devra être traité en premier.

Pré-conditions :

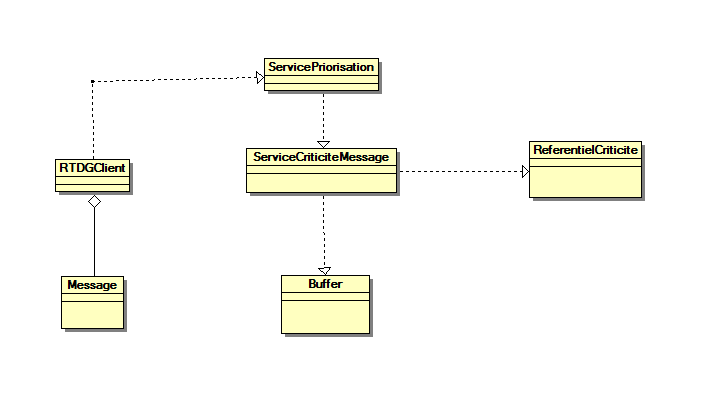
- [Framework de simulation](#framework_de_simulation) doit être démarré

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

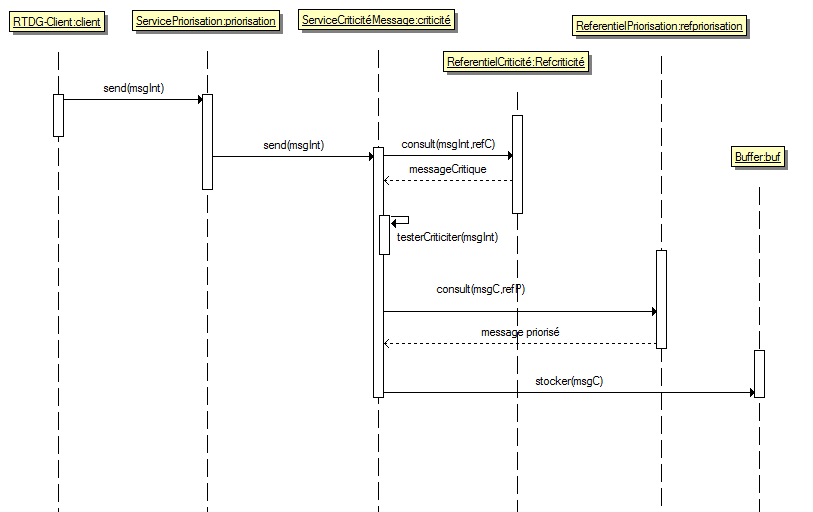
1. Scénario nominal: S-01020 : affecter les [message](#message)s non critiques à un buffer
2. le service de priorisation reçoit le [message](#message) interprété depuis RTDG-Client
3. le service de priorisation envoie le [message](#message) interprété au service de gestion des messages « ServiceCriticiteMessage »
4. Stocker les [messages](#message) non critique dans un [buffer](#buffer) dédié aux [messages](#message) non critiques
5. Diagramme de séquences système :



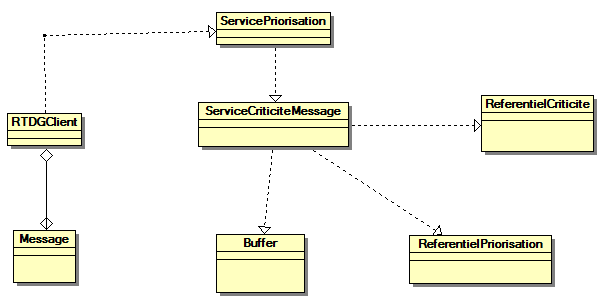
1. Diagramme de classes métiers :



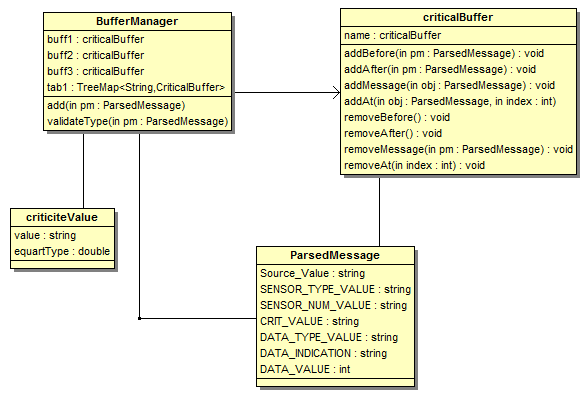
1. Scénario Alternatif 1: S-01029: Affecter les messages critiques aux buffers par ordre
2. le service de priorisation reçoit le [message](#message) interprété depuis RTDG-Client
3. le service de priorisation envoie le [message](#message) interprété au service de gestion des [messages](#message) « ServiceCriticiteMessage »
4. Prioriser le [message](#message) selon les [référentiels](#Référentiel)
5. Stocker les [messages](#message) critique dans un [buffer](#buffer) spécifique
6. Diagramme de séquences système :



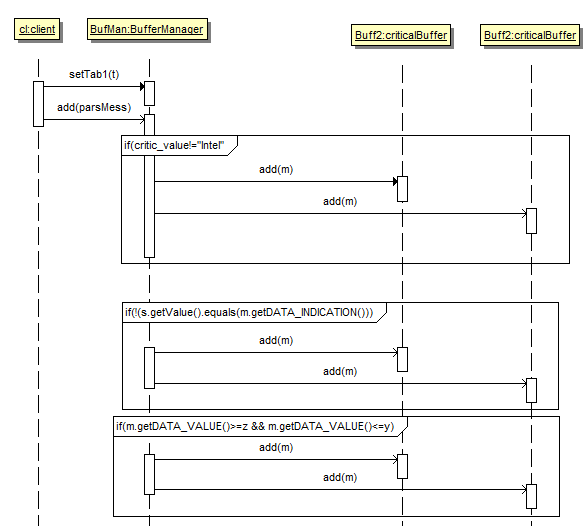
1. Diagramme de classes métiers :



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](#framework_de_simulation) pour reproduire le réseau terrain, ce qui nous permettra de tester l’envoi et la réception des [messages](#message) entre le [RTDG](#RTDG) et le [composant embarqué](#composant_embarque) et le traitement de ces [messages](#message).

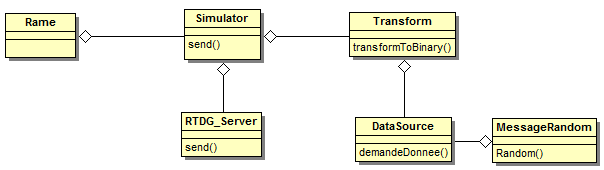
Ce [framework](#framework_de_simulation) a deux sources de données différentes :

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](#RTDG) sous format de tableau de bytes à l’aide du [protocole de communication](#protocole_de_communication).

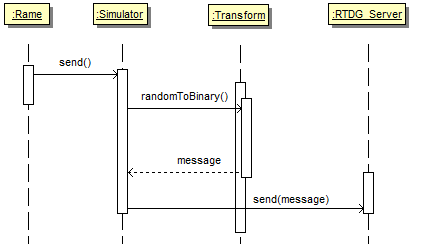
La deuxième source de données que peut utiliser notre [framework](#framework_de_simulation) est la génération aléatoire de données depuis un HashMap qui fait office de [référentiel](#Référentiel) et qui sont chargés au lancement du [RTDG](#RTDG).

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

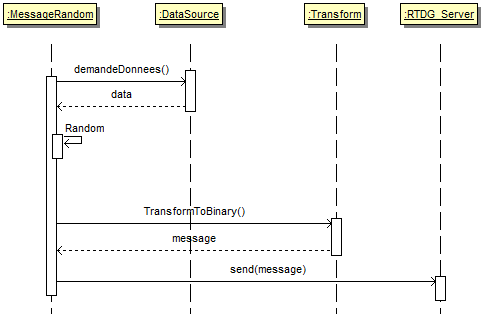
1. Diagrammes de la conception préliminaire
   * Diagramme de classes métier



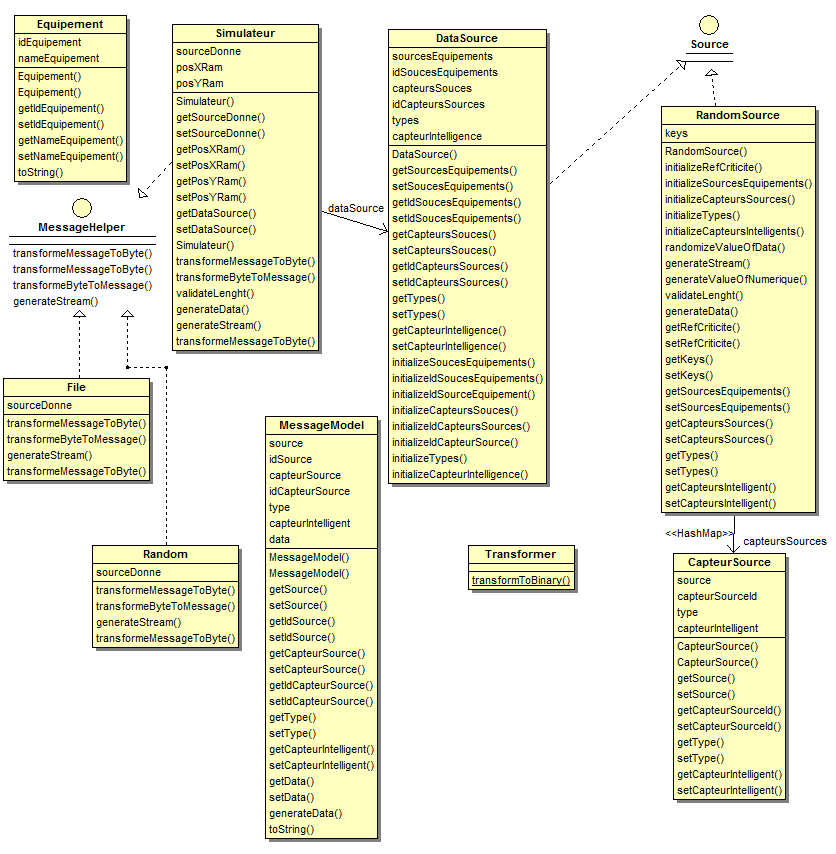
* + Diagramme de séquences système (rame)



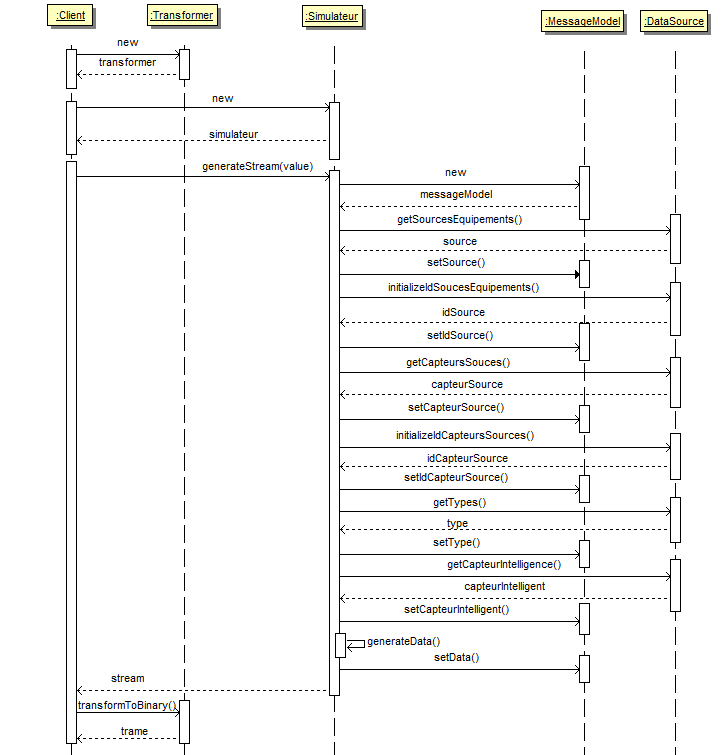
* + Diagramme de séquences système (Source de données)



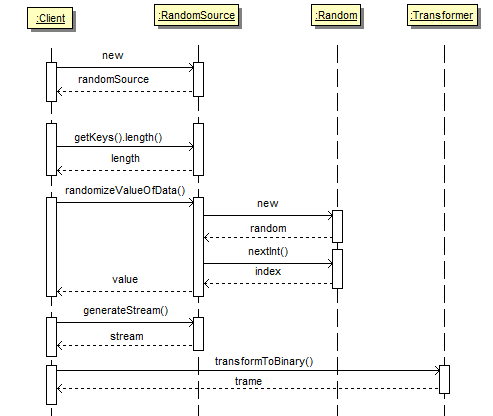
1. Diagrammes de la conception détaillée
   * Diagramme de classes détaillé



* + Diagramme de séquence détaillé (rame)

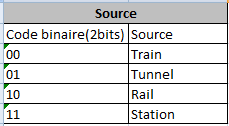


* + Diagramme de séquence détaillé (Random)

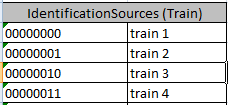
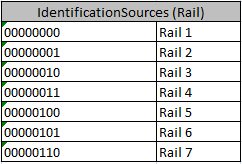


1. Référentiels :

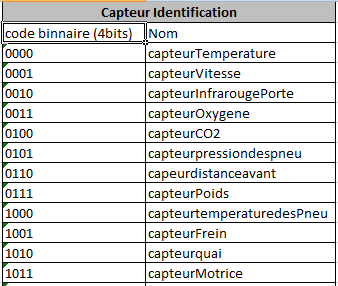
Comme expliqué dans le glossaire, les [référentiels](#Référentiel) sont les emplacements où sont stockées les informations dont a besoin notre application, dans notre contexte on a un [composant embarqué](#composant_embarque) qui envoie un message binaire. Pour traduire ce message, notre application consulte un [référentiel](#Référentiel) qu’on a défini sur plusieurs tableaux, un pour le [parser](#parser), un pour la [criticité](#criticité) et un pour les contres mesures. Le [message](#message) est alors une suite binaire de 32 bits. On a 2 bits qui définissent la source des messages comme il est expliqué dans ce tableau source :



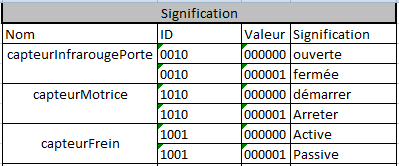
2) 8bits qui définissent du quelle source est-ce qu’il s’agit, parce qu’il peut exister un certain nombre pour chaque source, la même chose pour les rails, les stations et les tunnels. (Voir les tableaux)

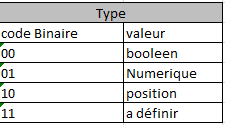
3) 4bits pour l’identification des [capteurs](#composant_embarque) qui sera comme suit :



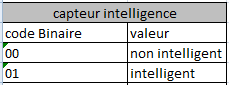
Et les 6bits seront représentés comme suit dans le tableau de signification :



4) un autre tableau qui identifie le type des [messages](#message) sur 2 bits comme suit :



5) les 2 suivants bits servent à identifier les [capteurs intelligents](#capteur_intelligent) et les non intelligents :



On a réservé les 10 derniers bits, pour les données que ces [capteurs](#composant_embarque) peuvent envoyer.

On a définit un autre [référentiel](#Référentiel) pour la [criticité](#criticité) qui peut traduire soit une criticité simple, soit une criticité complexe.

Dans notre contexte on a des [capteurs intelligents](#capteur_intelligent) qui peuvent envoyer des [messages](#message) qui se traduisent automatiquement en un message critique s’ils valident certaines conditions selon le [référentiel](#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](#criticité) qu’on a appelé événement complexe, cette [criticité](#criticité) se base aussi sur un [référentiel](#Référentiel) qui se traduira en un événement critique, chaque événement se traduit par un certains nombre de donnée envoyés par différents [capteurs](#composant_embarque) spécifique pour chaque événement.

