

MASTER 2 – ISIDIS

Projet de synthèse

V.A.L.

**Spécifications Fonctionnelles**

Sommaire

[I. Glossaire 3](#_Toc352163440)

[II. Use Case Model 5](#_Toc352163441)

[1) Les acteurs 5](#_Toc352163442)

[2) Le diagramme de cas d’utilisation 6](#_Toc352163443)

[3) Description des Use Cases 7](#_Toc352163444)

[III. Décomposition des Use Cases 8](#_Toc352163445)

[1) Use Case 1 : Collecter les informations terrains 8](#_Toc352163446)

[A. Diagramme de classes métier : 8](#_Toc352163447)

[B. Diagramme de classes participantes : 8](#_Toc352163448)

[C. Use Case Detail : 9](#_Toc352163449)

[D. Diagramme de séquence système : 10](#_Toc352163450)

[E. Conception détaillée : 10](#_Toc352163451)

[2) Use Case 2 : Prioriser messages 12](#_Toc352163452)

[A. Diagramme de classes métier : 12](#_Toc352163453)

[B. Diagramme de classes participantes : 13](#_Toc352163454)

[C. Use Case Detail : 13](#_Toc352163455)

[D. Conception préliminaire : 14](#_Toc352163456)

[E. Conception détaillée 15](#_Toc352163457)

[IV. Exigences du projet 19](#_Toc352163458)

[1) Réplication 19](#_Toc352163459)

[V. Hors périmètre 23](#_Toc352163460)

[1) S01021 :Création du générateur de données 23](#_Toc352163461)

[A. Diagrammes de la conception préliminaire 24](#_Toc352163462)

[B. Diagrammes de la conception détaillée 25](#_Toc352163463)

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 |
| Référentiel | Format de message (parser) | Emplacement où sont stockées les informations relatives au message, où est décrite la structure des [messages](#message) reçus du terrain pour pouvoir les interpréter. Cette trame englobe un certain nombre d’informations détaillés comme suit :   * Source : 2 bits qui nous disent si le message vient d’une station, d’une rame, de rails ou d’un tunnel. * Id\_source : 8 bits représentant l’id du composant qui nous envoie l’information. Chaque rail, rame, station ou tunnel possède un id qui lui correspond. * Sensor Identification : 10 bits pour représenter le capteur qui envoie l’information. Les 4 premiers bits représentent le type du capteur, par exemple si c’est un capteur de température, d’oxygène ou d’ouverture d’une porte. Les 6 bits qui suivent représentent l’id de ce capteur qui envoie l’information. * Type : 2 bits pour représenter le type de l’information qu’on va recevoir, par exemple pour une porte, la donnée peut être booléenne pour dire que la porte est soit fermée soit ouverte, ou bien un capteur qui envoie des informations numériques tel le capteur de température. * Intelligent sensor : 2 bits servant à dire si le capteur est intelligent ou pas, les capteurs intelligents sont ceux qui ne nous envoient que les messages critiques, les autres sont neutres et c’est à nous de définir si le message est critique ou pas. * Data : 10 bits de données qui représentent la donnée que nous envoie le capteur. |
| Criticité | Emplacement où est stockée la [criticité](#criticité) des [messages](#message). Il y a la décomposition des informations dont la criticité est simple, dont voici le tableau :   |  |  |  |  | | --- | --- | --- | --- | | CriticiteSimple | | | | | Nom | criticité | Ecart-type | Unité | | capteurTemperature | 80 | 10 | Celsius | | capteurVitesse | 60 | 30 | Km/H | | capteurInfrarougePorte | - | - | - | | capteurOxygene | 10 | 5 | Kpascal | | capteurCO2 | 10 | 5 | Kpascal | | capteurpressiondespneu | 50 | 10 | KPascal | | capeurdistanceavant | 50 | 10 | mètre | | capteurPoids | 1000 | 100 | Kg | | capteurtemperaturePneu | 80 | 20 | Celsius | | capteurquai | - | - | - | | capteurMotrice | - | - | - | | CapteurAlarme | - | - | - | | CapteurNeige | - | - | - |     Et d’autres dont la criticité est plus complexe, c’est-à-dire, qu’un incident peut être constitué à partir de plusieurs messages :    L’écart-type est un pourcentage qui définit l’intervalle où la valeur est normale, par exemple si on met pour la température une valeur de 100 degrés Celsius avec un écart-type de 10% ceci voudrait dire qu’entre les valeurs 90 et 110, la température est normale, sinon, le composant embarqué envoie un message au RTDG. |
| 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 |
| 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) |
| Message | | Trame de bits, contenant les informations transmises entre le [composant embarqué](#composant_embarque) et le composant [RTDG](#RTDG) |
| Message traité | | Un message, qui est reçu par le service de gestion des messages et qui est décomposé en plusieurs informations |
| Capteurs intelligents | | Ce sont des [capteurs](#composant_embarque) qui ne nous envoient que des [messages](#message) critiques |
| Réplication | | La réplication est un processus de partage d'informations pour assurer la cohérence de données entre plusieurs sources de données redondantes, pour améliorer la fiabilité, la [tolérance aux pannes](http://fr.wikipedia.org/wiki/Tol%C3%A9rance_aux_pannes) ou l'accessibilité |
| Contexte du RTDG | | Ce contexte est composé des différents messages reçus à partir du composant terrain |

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.

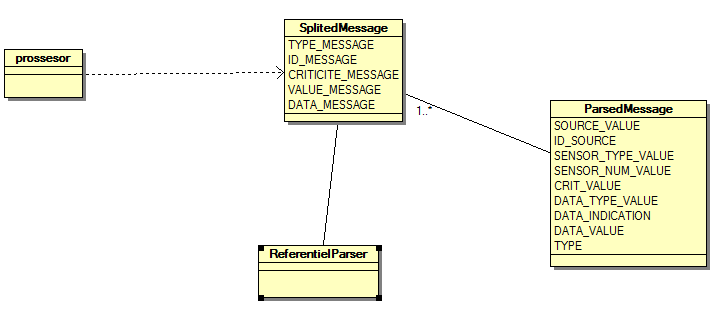
1. Le diagramme de cas d’utilisation



1. Description des Use Cases

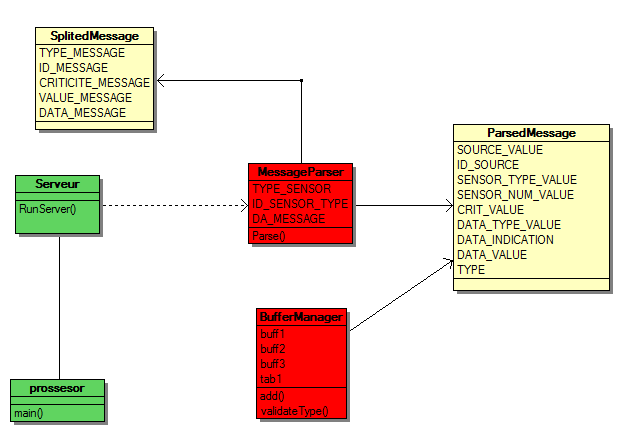
|  |  |
| --- | --- |
| Nom | Description |
| Collecter les informations terrains | Cet UC a comme rôle la réception de tout type d'informations terrain à partir des différents [capteurs embarqués](#composant_embarque) et qui nous seront envoyés par le composant [EFIC](#efic) |
| Prioriser [messages](#message) | 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. Diagramme de classes métier :



1. Diagramme de classes participantes :

Rouge : contrôle  
vert : dialogue  
Jaune : entité



1. 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).

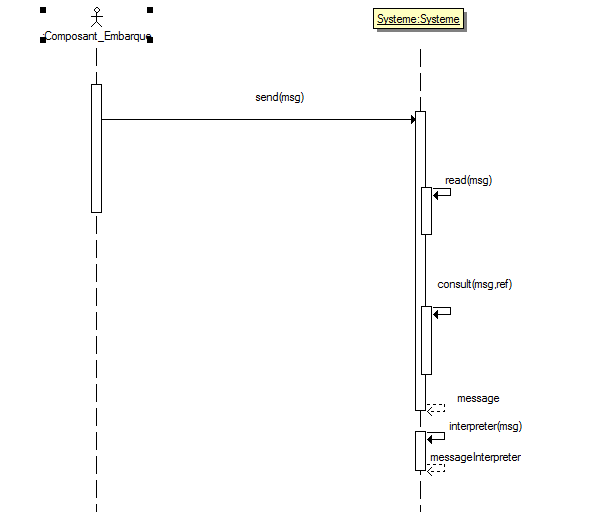
Pré-conditions :

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

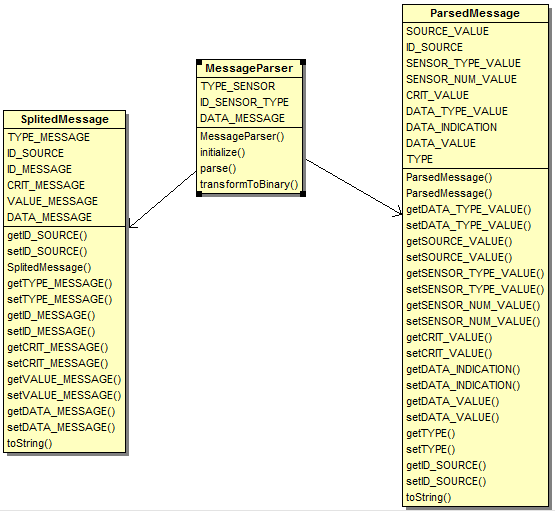
-  Le [RTDG](#RTDG) doit être en écoute pour pouvoir recevoir les messages depuis les [composants embarqués](#composant_embarque)

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

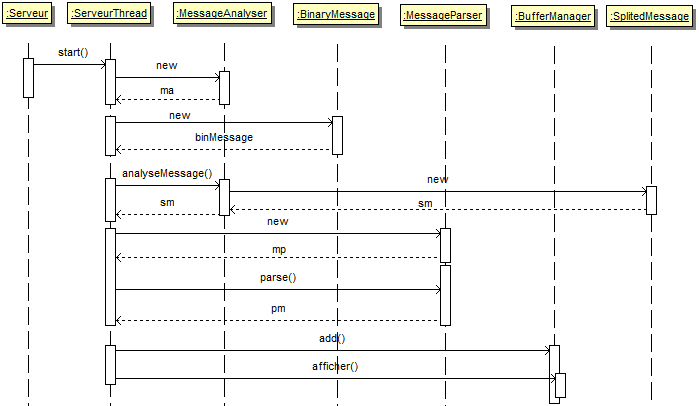
1. Le [composant embarqué](#composant_embarque) génère une trame composée de 0 et de 1
2. Le [composant embarqué](#composant_embarque) envoie cette trame au RTDG-Client via un protocole de communication
3. Le RTDG-Client réceptionne ce message et consulte le [référentiel](#Référentiel) où il y a la signification de cette trame
4. Le RTDG-Client interprète le [message](#message) selon les référentiels
5. Le RTDG-Client envoie le [message](#message) interprété au service de priorisation
6. Diagramme de séquence système :



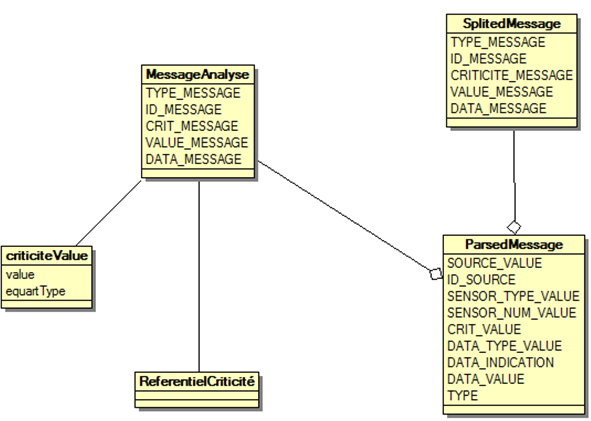
1. Conception détaillée :
   * Diagramme de classes détaillé



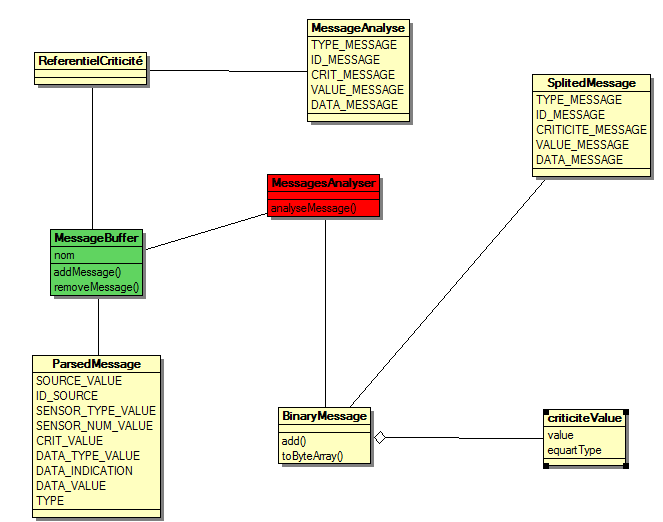
* + Diagramme de séquences détaillé



1. Use Case 2 : Prioriser messages
2. Diagramme de classes métier :



1. Diagramme de classes participantes :



1. Use Case Detail :

Acteur principal : -

Acteur secondaire : Composant\_Embarqué

Objectifs :

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

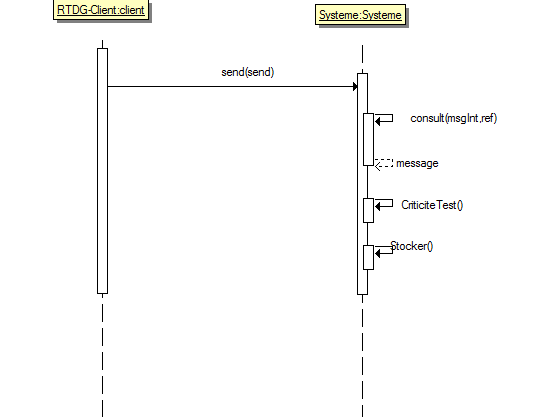
Pré-conditions :

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

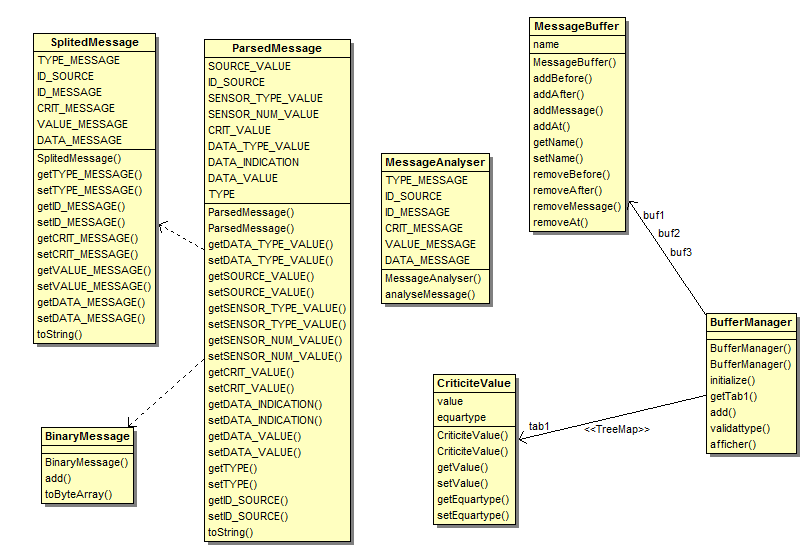
- Les [référentiels](#Référentiel) doivent avoir été chargés en mémoire

**Scénario nominal: S-01020 : affecter les** [**message**](#message)**s non critiques à un buffer**

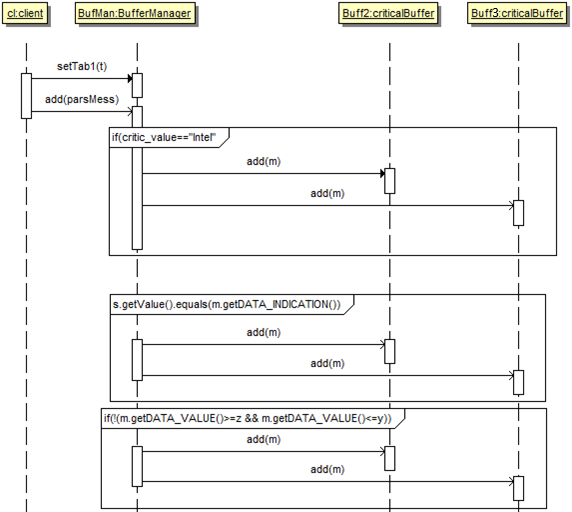
1. le service de priorisation reçoit le [message interprété](#message_traité) depuis le RTDG-Client sous forme de parsedMessage
2. le service de priorisation envoie le [message interprété](#message_traité) au service de gestion des messages « ServiceCriticiteMessage »
3. Le service de gestion de messages, vérifie que le [message](#message) ne vient pas d’un capteur intelligent (à partir des octets 21 et 22)
4. Le service consulte le [référentiel](#Référentiel) pour savoir si le message reçu est critique ou pas.
5. Le message est alors stocké dans un buffer réservé aux [messages](#message) non critiques mais aussi dans un buffer cache.
6. Conception préliminaire :
7. Diagramme de séquences système :



1. Diagramme de classes détaillé :

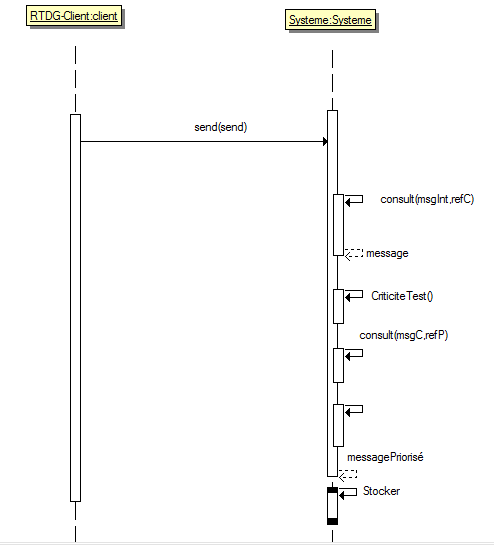


1. Conception détaillée
2. Diagramme de séquences détaillé

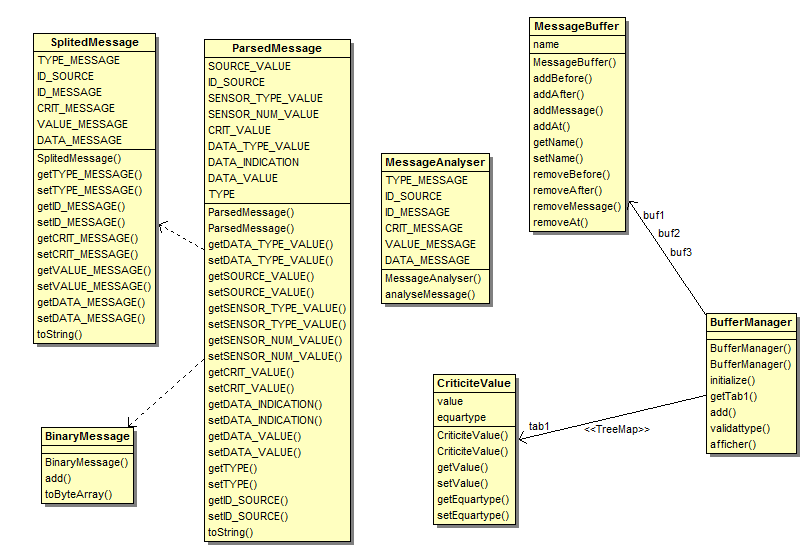


**Scénario Alternatif 1: S-01029: Affecter les messages critiques aux buffers par ordre**

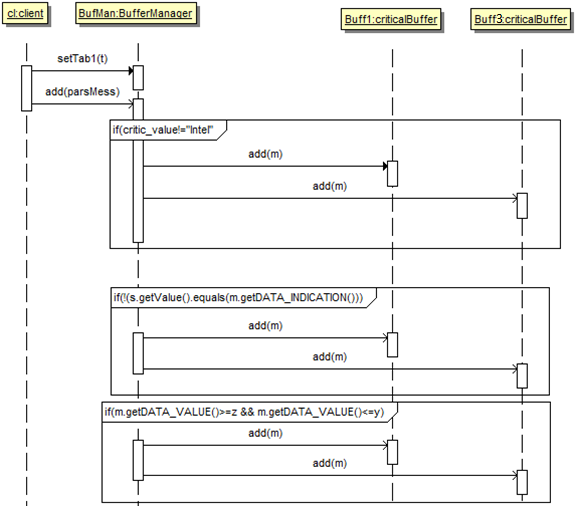
1. le service de priorisation reçoit le [message](#message) interprété depuis le RTDG-Client sous forme de parsedMessage
2. le service de priorisation envoie le [message](#message) interprété au service de gestion des messages « ServiceCriticiteMessage »
3. Le service de gestion des messages, vérifie si le message vient d’un capteur intelligent (à partir des octets 21 et 22)
4. Si c’est le cas, on passe à l’étape 6
5. Sinon, le service consulte le référentiel pour savoir si le message reçu est critique ou pas.
6. Le message est alors stocké dans un buffer réservé aux messages critiques mais aussi dans un buffer cache.
7. Diagramme de séquences système :



1. Conception détaillée :
   * Diagramme de classes détaillé :



* + Diagramme de séquences détaillé :



1. Use Case 3 : orchestrer messages
2. Diagramme de classes métier
3. Diagramme de classes participantes
4. Use Case Detail

Acteur principal : -

Acteur secondaire : Composant\_Embarqué

Objectifs :

Cet UC a pour objectif, la transmission et la réception des informations collectées depuis et vers le composant embarqué en passant un message qui contiendra le nom de la source ,codetypeEvenement, la position et le degré de la criticité (qui pourra être soit 1,soit 5 ) , et cette transmission se fait via l’implémentation du JMS (ActiveMQ)

Pré-conditions :

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

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

- RTDRS doit être démarré

- serveur ArtiveMQ (jms) doit être démarré (la queue qui partage les messages doit être démarré)

**Scénario nominal: S-01051 : l’envoie du message au RTDRS**

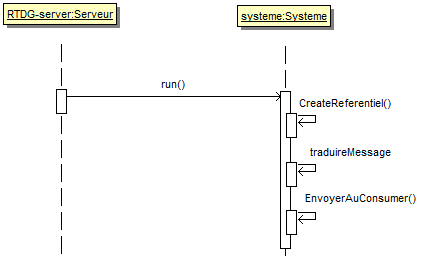
1. L’orchestrateur reçoit le message depuis le service de priorisation
2. Le message est filtré avec un autre service dans le même UC (orchester message) et reçus sous format d’objet
3. Le message est marshalé pour être envoyé en format xml
4. Le message est envoyé à la queue
5. Conception préliminaire
6. Conception détaillée

**Scénario alternatif : S-01034 : réception du message depuis RTDRS**

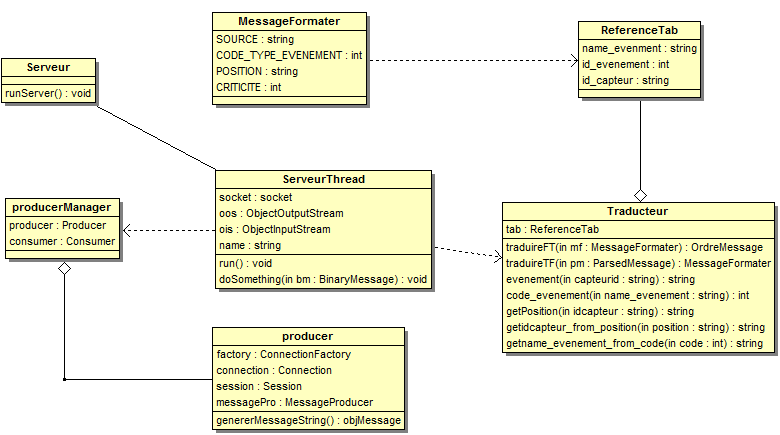
1. Activer le serveur
2. Mettre le consumer en mode écoute pour recevoir le message – réception de l'objet depuis RTDRS qui contient les informations
3. Unmarshaler le message
4. récupérer l’information et la sauvegarder dans un buffer
5. Conception préliminaire
6. Conception détaillée

**User Story : Envoi du message au TraducteurFT**

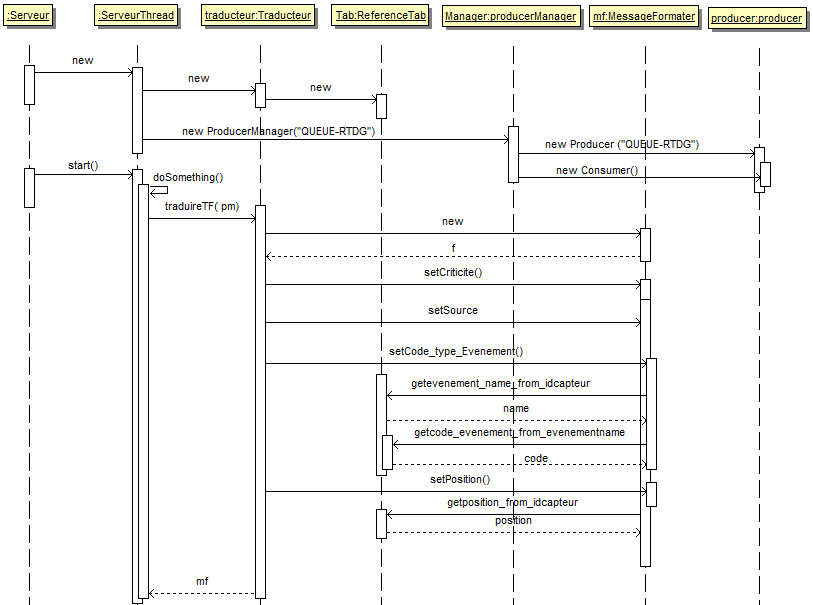
1. Activer L’activeMQ
2. Activer le Serveur
3. Lancer un client
4. Réception d’un Message du composant de priorisation
5. Traduire le message du fonctionnel au technique
6. Faire passer le message traduit au Producer de l’activeMQ pour qu’il soit envoyé au RTDRS
7. Conception préliminaire :
   * Diagramme de séquence système :



1. Conception détaillée :
   * Diagramme de classes détaillé



* + Diagramme de séquences détaillé

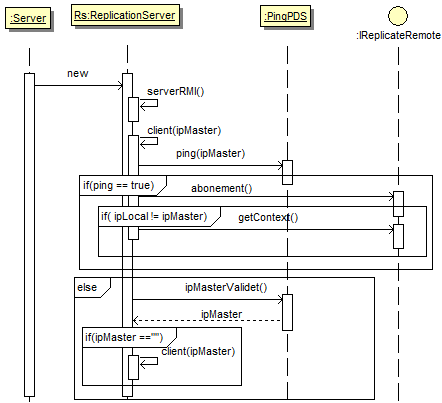


1. Exigences du projet
2. Réplication

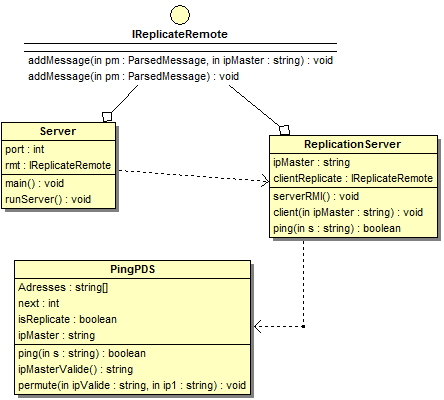
Pour pouvoir assurer la tolérance aux pannes, il va nous falloir faire une [réplication](#replication) du [contexte](#contexte) de notre [RTDG](#RTDG), de façon à ce que le serveur [RTDG](#RTDG) puisse à tout moment être disponible même si l’un des serveurs tombe en panne, car il y aura un autre qui prendra le relais.

Description du scénario:

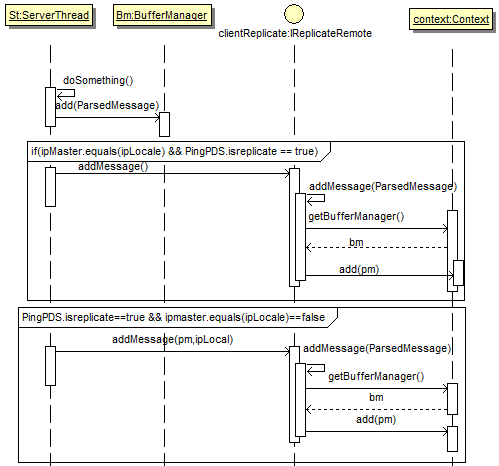
1. Démarrage du serveur
2. Si c’est le premier à démarrer, il est élu comme master et passe à l’étape 4
3. Si ce n’est pas le premier à démarrer, il demande le contexte au master
4. Le serveur peut notifier le master
5. Le master notifie alors, tous les autres slaves excepté celui qui l’a notifié.
   * Diagramme de séquence détaillé du démarrage d’un serveur



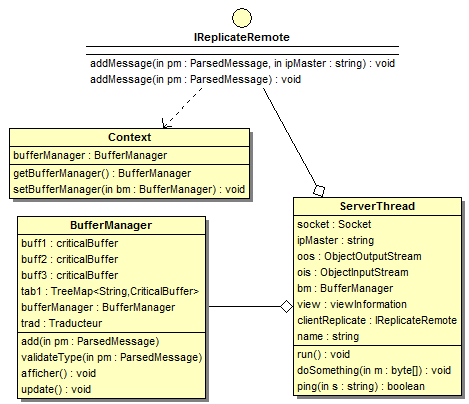
* + Diagramme de classes détaillé du démarrage d’un serveur



* + Diagramme de séquence détaillé de la notification



* + Diagramme de classes détaillé de la notification



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).

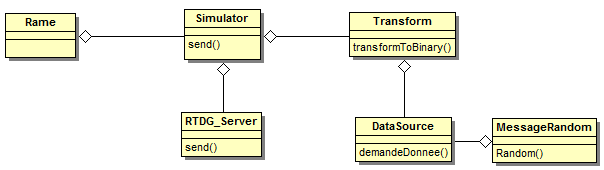
Ce [framework](#framework_de_simulation) a deux sources de données différentes :

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).

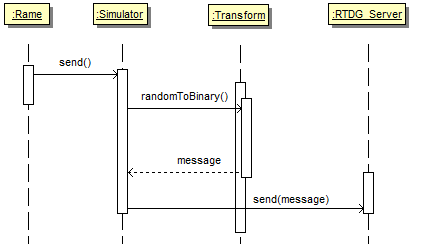
La deuxième source de données que peut utiliser notre [framework](#framework_de_simulation) est la génération aléatoire de données depuis un HashMap qui fait office de [référentiel](#Référentiel) et qui sont chargés au lancement du [RTDG](#RTDG).

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).

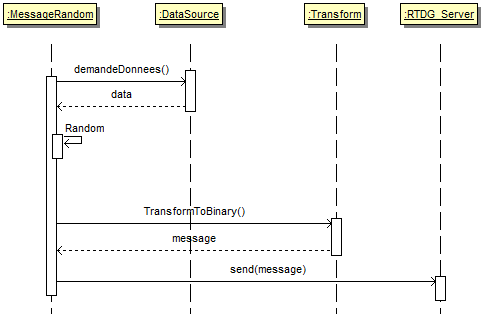
1. Diagrammes de la conception préliminaire
   * Diagramme de classes métier



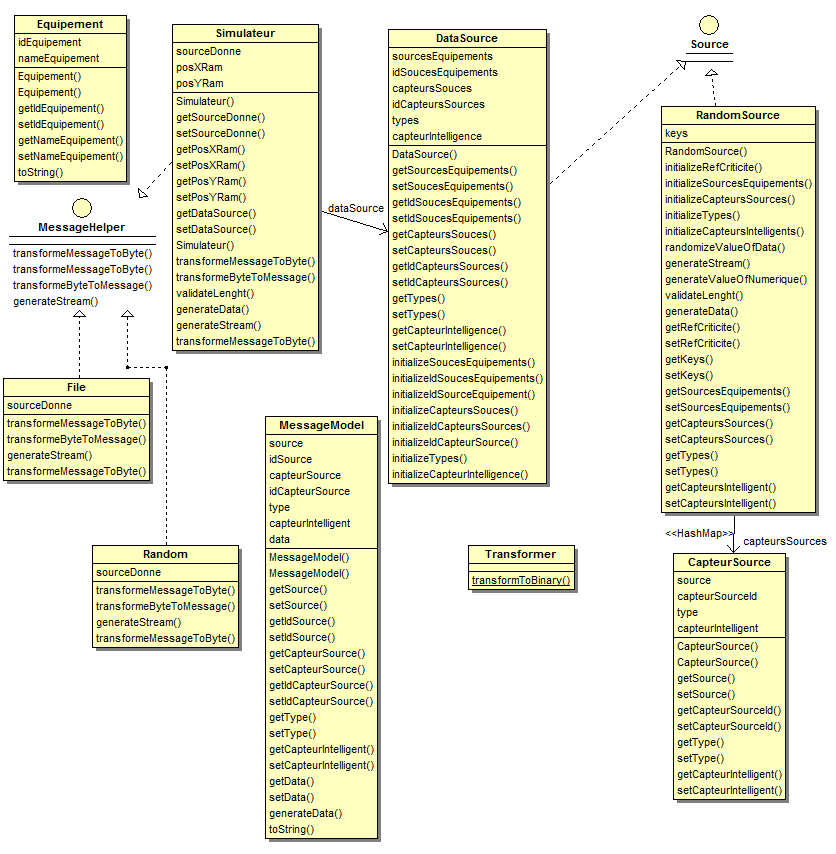
* + Diagramme de séquences système (rame)



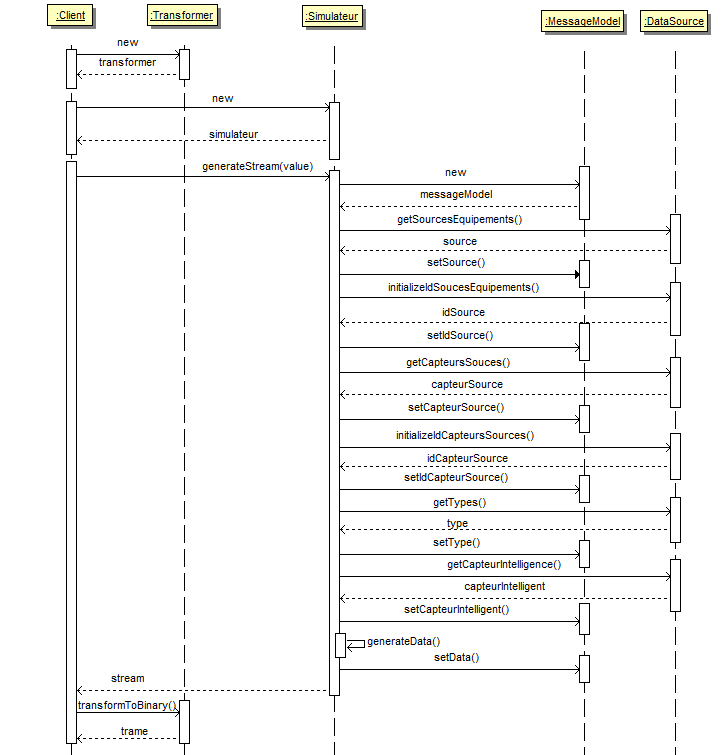
* + Diagramme de séquences système (Source de données)



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



* + Diagramme de séquence détaillé (rame)



* + Diagramme de séquence détaillé (Random)

