
# London Ambulance System

## Architecture logicielle

Groupe 2
Simon Busard,
Antoine Cailliau,
Laurent Champon,
Erick Lavoie,
Quentin Pirmez,
Frederic Van der Essen,
Géraud Talla Fotsing

# Table des matières

# 1. Architecture logique

Cette section présente les alternatives d'architecture que nous avons exploré. Ensuite, nous détaillons l'architecture que nous avons finalement choisie.

## 1.1. Architectures évaluées

Cette section présente les deux autres alternatives que nous avons exploré, à savoir l'architecture *pipe* et l'architecture *bus*.

### 1.1.1. Architecture *Pipe and Filter*

Ce type d'architecture consiste en un flux de données, et une série de filtres qui les transforment séquentiellement, pour arriver au résultat désiré. Cette architecture est attractive car très simple, modulaire et facilement parallélisable, et nous avons donc cherché à savoir si elle pouvait s'adapter à notre problème.

Un des principaux problèmes de ce type d'architecture est de s'accorder sur le type de données du flux. Ici un type s'imposait, à savoir l'incident. En effet tout dans notre systeme, tourne autour de celui ci : il démarre quand un nouvel incident est créé, et s'arrête lorsque l'incident est résolu. Chaque filtre modifierait donc la représentation interne de l'incident, mais surtout, s'assurerait que cette représentation s'accorde avec le monde externe.

Cependant, au fur et à mesure que l'on considère cette architecture nous sommes forcés de nous en écarter. Premièrement, il n'est pas possible de modifier l'état incrémentalement dans le module lui même, les modifications de l'incident doivent être atomiques.

Ensuite, s'il est possible de réaliser une séquence de filtres indépendants, la réussite de chacun est dépendante du monde réel, et n'est donc pas garantie. Un échec d'un module requiert un retour au module précédent, voir tout au début.

Cela consiste donc finalement à implémenter la machine à état de l'incident qui est notre architecture finale.

Malgré le fait que cette architecture semblait assez éloignée de notre problème et qu'elle n'a finalement pas été retenue, elle aura largement influencé notre architecture finale, nous montrant ainsi qu'aucun shéma ne doit ètre écarté à priori.

## 1.2. Architecture en bus

L'architecture en bus est une variante de l'architecture event-based dans laquelle chaque module génère et écoute des événements. Contrairement au schéma général utilisant un "broker" qui centralise les "intêrets" des acteurs pour certains événements et notifie les acteurs interessés quand un événement survient, dans l'architecture en bus, tout les modules écoutent tous les événements et filtrent uniquement ceux qui les intéressent. Il est donc facile, dans cette dernière de modifier un module en y ajoutant ses intêrets pour certains événement sans devoir toucher aux autres modules.

D'intuition cette architecture semble simple cependant elle n'est pas très adaptée pour notre système dans le sens où chaque module doit tenir un historique des états du système afin de raisonner correctement lorsqu'il reçoit un événement. Un exemple simple est le cas où le système veut mobiliser une ambulance qui refuse la mobilisation, le système tente alors de mobiliser une deuxième ambulance qui refuse aussi.

Lorsque le système mobilisera une troisième ambulance, il devra faire attention à ne pas mobiliser la première qui à déjà refuser, ni la deuxième.

## 1.3. Architecture finale du système

Cette section présente l'architecture du système de gestion d'ambulance. La section 1.3.1 présente les définitions des différents modules, la section 1.3.2 présente les liens entre ces différents modules, la section 1.3.3 présente la gestion des divers obstacles au sein de notre architecture, la section 1.3.4 présente les évolutions du système, la section 1.3.5 justifie l'utilisation des liens USE au sein de l'architecture.

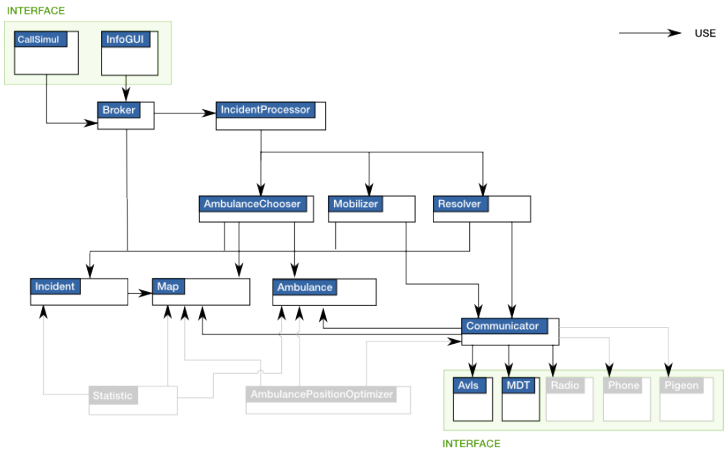


Figure 1.1. Architecture logique pour la partie du système.

### 1.3.1. Définition des modules

Les définitions sont présentées dans le tableau 1.3.

### 1.3.2. Lien entre les modules et agencement

Afin de présenter l'architecture, nous allons effectuer un parcours de celle ci pour le cas idéal de résolution d'incident.

Nous commençons par l'InfoGUI qui est l'interface graphique que l'opérateur téléphonique utilise pour entrer les données caractérisant l'incident.

Ces données sont transmises au Broker. Celui possède deux fonctions. La première, architecturale, est de présenter une interface unique aux différents système d'introduction de données : Interface Utilisateur, simulateur, ou encore d'autres logiciels de

| Module | Définition |
|---|---|
| CallSimul | Ce module est reponsable de la notification des incidents en provenance du simulateur. |
| InfoGUI | Ce module est responsable de l'interface graphique permettant l'ajout d'incident dans le système. |
| Broker | Ce module est responsable de fournir une API fixe pour l'ajout d'incident au sein du système. |
| IncidentProcessor | Ce module est reponsable du suivi des incidents, il s'occupera de faire en sorte que l'incident soit traité corectement et dans les délais. |
| AmbulanceChoser | Ce module est reponsable du choix de l'ambulance. |
| Mobilizer | Ce module est responsable de la mobilisation virtuelle et concrète de l'ambulance |
| Resolver | Ce module est responsable de la correcte terminaison d'un incident. |
| Incident | Ce module est responsable de la gestion des incidents au sein du système. |
| Map | Ce module est responsable de la gestion de la carte et donner les informations appropriés aux autres modules. |
| Ambulance | Ce module est reponsable de la gestion des ambulances et de l'information qui y est lié au sein du système. |
| Communicator | Ce module est reponsable de la gestion correcte des communications à partir du système. |
| Avls | Ce module est responsable de l'envoi correct de la position de l'ambulance. |
| MDT | Ce module est reponsable de la communication entre les équipages de l'ambulance et le système. |

Tableau 1.1. Définition des différents modules utilisé au sein du système.

gestion d'hopitaux ou de pompiers par exemple.

L'autre rôle, plus fonctionnel, du broker est d'instancier un Incident, et de transmettre une référence vers celui ci à l'InfoProcessor.

L'InfoProcesseur est le chef d'orchestre de notre système. Il se charge d'abord de compléter la représentation de l'Incident en évaluant le type d'ambulance requise et les coordonnées GPS de la destination. pour cela il s'aidera du module Map et Incident.

Ensuite, L'InfoProcessor va appeler différent modules afin de faire avancer état par état la résolution de l'Incident.

En premier il devra choisir l'ambulance grâce à AmbulanceChooser celui ci fera pour cela appel au module Map pour évaluer les distances, et au module Ambulance pour savoir lesquelles sont libres.

Une fois l'ambulance choisie, le Mobilizer se chargera d'envoyer l'ordre de mobilisation. Il s'aidera du Communicator pour envoyer l'ordre et récuperer la confirmation.

Le module Resolver permettra de gérer les évènements intervenant entre le moment ou l'ambulance est mobilisée et le moment ou elle à réussi ou échoué à résoudre l'incident.

Le module Mobilizer est ensuite utilisé à nouveau pour démobiliser l'ambulance ayant terminé sa mission.

Tout au long de cette opération, le module Communicator se charge de recevoir les events provenant du monde extérieur et de mettre à jour la base de données des ambulances et de la carte. Il s'assure aussi de l'intégrité des données recues, et d'envoyer au modules de résolutions des events qui peuvent interrompre ceux ci.

### 1.3.3. Gestion des obstacles dans l'architecture

L'architecture modélisant la machine à état de l'Incident, la résolution d'obstacles est aisée et ne nécessite généralement que la modification que de quelques modules.

- Communicator afin de prendre en compte un nouveau type d'event correspondant à l'obstacle si nécessaire.
- le module ambulance, map ou Incident afin d'y ajouter les propriétés supplémentaires nécessaires.
- le module chargé du changement d'état de l'Incident concerné par l'obstacle.
- Si les changements outrepassent le secret du module précédent, il faudra modifier IncidentProcessor.

Par exemple la résolution de l'obstacle consistant en une duplication des incidents consiste à ajouter à Incident une opération permettant de détecter un duplicata, et de modifier IncidentProcessor pour l'utiliser et arrèter la résolution dans de tels cas.

### 1.3.4. Evolutions du système et de l'architecture

Nous avons énuméré une série d'évolutions du système et avons identifié les modules devant etres modifiés ou ajoutés.

#### Nouveaux types de véhicules

- Modification d'Ambulance : nouvelle catégorie,
- Modification d'AmbulanceChooser : le calcul de la plus courte distance peut etre différent avec le nouveau type, et il y a une nouvelle catégorie de véhicule à choisir,
- Eventuelle modification de Map pour ajouter une distance spécifique au véhicule ( par exemple à vol d'oiseau pour un hélicoptère ) ;

#### Incident demandant l'intervention de plusieurs ambulances

- Modification d'InfoProcessor : choisit plusieurs ambulances ;

#### Intégration aux systèmes de Police, d'Hopital ou Pompiers

- Modification de Broker : Nouveaux canaux de création d'incidents,
- Modification de Communicator : Nouveaux canaux d'écoute pour obstacles ;

#### Annulation d'un incident

- Modification de Broker et Communicator : nouveaux events de demande d'annulation,
- Modification d'InfoProcessor : écoute de ces events, et gestion de ceux-ci ;

#### Ajout d'un canal de communication

- Modification de Communicator : écoute et envoi dans ce nouveau canal ;

Utilisation d'une base de données afin de sauvegarder l'état du système

- Ajout d'un module Database,
- Modification d'Incident,Ambulance,Map pour faire usage de celle-ci ;

Utilisation de données géographiques plus complexes.

- Modification du module Map ;

Réalisation de statistiques sur le comportement du système

- Ajout d'un module Statistique qui utilise Incident,Ambulance et Map ;

Suivi de la progression de l'incident par les témoins.

- Ajout d'un nouveau module FollowUpGUI utilisant Map,Incident et Ambulance pour permettre à l'opérateur téléphonique d'obtenir des informations de suivi sur l'incident ;

### 1.3.5. Justification des liens USE

Chaque lien USE dans le schéma peut être justifié. Le tableau 1.2 synthétise la justification des différents liens pour le système.

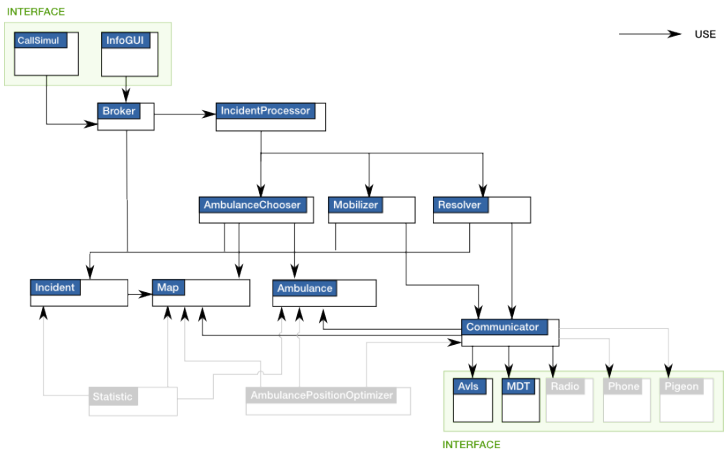


Figure 1.2. Architecture logique pour la partie du système.

## 1.4. Architecture finale du simulateur

La section 1.4.1 présente la définition des modules du simulateur. La section 1.4.2 présente la justification des liens USE au sein du simulateur.

### 1.4.1. Définition des modules du simulateur

Le tableau 1.3 synthétise tous les différents modules repris dans le simulateur.

### 1.4.2. Présentation de la structure

L'architecture proposée pour le simulateur implémente une architecture basée sur MVC. En l'occurence, le modèle correspond aux objets de simulation, le contrôleur au simulateur et la vue correspond au GUI. Additionnellement, la définition du scénario à exécuter est définie dans le Scénario et la communication est regroupée sous communication. L'utilisation du patron de conception *Observer* est utilsée sur les objets de simulation et les canaux de communication pour permettre un maximum de flexibilité dans la connection des différents objets et faciliter la propagation des évènements aux travers du simulateur.

La simulation est synchrone au sens où il y a un temps global défini pour l'éxécution de chacun des objets. Tous se synchronisent sur le step initié par le simulateur. Le choix d'une simulation synchrone a été préféré à une simulation asynchrone car il est plus facile de contrôler l'évolution de la simulation en présence de multi-processus. Il est alors possible de paralléliser le traitement des évènements de chacun des objets tout en maintenant une synchronisation globale. La récupération des évènements en provenance du LAS est synchrone (effectuée à chaque *step*). Cependant, la propagation des évènements des objets de la simulation vers les canaux de communication est asynchrone, par l'utilisation d'Observer, pour éviter la gestion du dispatching des évènements au simulateur.

Les machines à états finis sont des secrets des différents objets de simulation et n'ont pas été explicités dans l'architecture.

Le tableau 1.4 présente la justification des différents liens USE utilisé dans le simulateur.

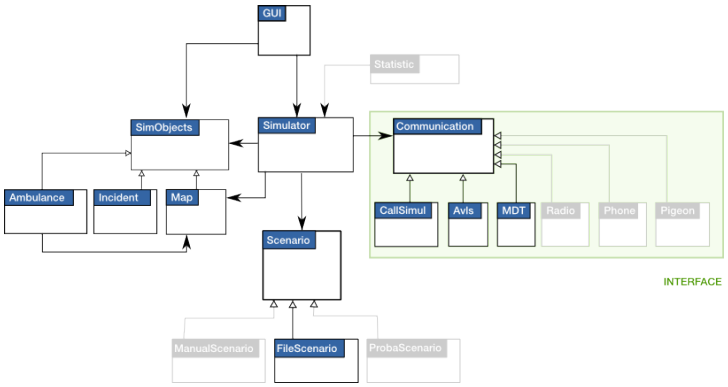


Figure 1.3. Architecture logique pour la partie du système.

| Source | Destination | Justification |
|---|---|---|
| CallSimul et InfoGUI | Broker | L'implémentation de l'interface d'ajout d'un incident ou l'interface d'ajout automatique d'un incident dépends de la spécification externe du Broker. Ce dernier étant responsable de donner une API fixe pour l'ajout d'incident de la part du monde extérieur. |
| Broker | IncidentProcessor | L'implémentation du broker dépends de la spécification externe de l'incidentProcessor. Le broker est responsable de l'appel et du lancement de la procédure de gestion des incidents. |
| IncidentProcessor | AmbulanceChooser | L'implémentation correcte de l'IncidentProcessor dépends de la spécification externe de l'AmbulanceChooser. Si le module ne retourne pas l'ambulance choisie alors le reste de l'implémentation du traitement de l'incident peut ne plus être correcte. |
| IncidentProcessor | Mobilizer | L'implémentation correcte de l'IncidentProcessor dépends de la spécification externe du Mobilizer. Si l'ambulance n'est pas correctement mobilisée alors le reste de la procédure de traitement de l'incident ne sera peut-être pas correcte. |
| IncidentProcessor | Resolver | L'implémentation correcte de l'IncidentProcessor dépends de la spécification externe du Resolver. Si l'incident n'est pas correctement clot alors le reste de la procédure de traitement de l'incident ne sera peut-être pas correcte. |
| * | Ambulance | L'implémentation des différents modules dépendra directement des spécifications externe du module Ambulance, ce dernier étant reponsable de la gestion de l'information utilisée à travers le système. |
| * | Map | L'implémentation des différents modules dépendra directement des spécifications externe du module Map, ce dernier étant reponsable de la gestion de l'information utilisée à travers le système. |
| * | Incident | L'implémentation des différents modules dépendra directement des spécifications externe du module Incident, ce dernier étant reponsable de la gestion de l'information utilisée à travers le système. |
| Mobilizer | Communicator | L'implémentation correcte du Mobilizer dépends directement de la spécification externe du Communicator. En effet, si le Communicator n'envoi pas correctement les informations, le système peut ne plus fonctionner correctement. |
| Communicator | AVLS et MDT | L'implémentation du Communicator dépends directement de la spécification externe de l'AVLS et du MDT. En effet, si le comportement de l'AVLS ou du MDT change alors l'implémentation du Communicator peut ne plus être correcte. |

Tableau 1.2. Justification des liens USE du système.

| Module | Définition |
|---|---|
| GUI | Ce module est responsable de l'interface graphique permettant de modifier les paramètres du simulateur |
| Simulator | Ce module est responsable de la gestion de la simulation d'un incident à partir des informations recues d'un scénario |
| SimObjects | Ce module est responsable la gestion des objets de simulation au sein de l'environnement simulé |
| Ambulance | Ce module est responsable de la gestion des ambulances et de l'information qui y est lié au sein de l'environnement simulé |
| Incident | Ce module est reponsable de la gestion des incidents et de l'information qui y est lié au sein de l'environnement simulé |
| Map | Ce module est responsable de la gestion de la carte de l'environnement simulé |
| Scenario | Ce module est responsable de transformer les différents format de simulations en format unique compréhensible du simulateur |
| FileScenario | Ce module est responsable de la creation de simulations prédéfinies contenues dans un fichier. |

Tableau 1.3. Définition des différents modules utilisé au sein du système.

| Source | Destination | Justification |
|---|---|---|
| Gui | Simulator | L'implémentation de l'interface pour lancer et stopper la simulation dépend de la spécfication externe de Simulator. |
| Gui | SimObjects | L'implémentation de la visualisation de la simulation dans l'interface dépend de la spécification externe de SimObjects. En effet, l'interface écoute les évènements générés par SimObjects définissant les objets à afficher. |
| Simulator | SimObjects | L'implémentation de la simulation dépend de l'interface externe des SimObjects. Le simulateur ne pourra pas connecter les SimObjects entre eux si leur interface change. |
| Simulator | Map | L'implémentation de la simulation dépend de l'interface externe de Map car le Simulator ajoute et enlève les obstacles de celle ci pour correspondre au scénario de simulation. |
| Simulator | Communication | L'implémentation de la simulation dépend de l'interface externe du module de Communication. Si l'opération recieve change sa spécification, le simulateur ne pourra pas fonctionner. |
| Ambulance | Map | L'implémentation de Ambulance dépend de l'interface externe de Map, En effet si l'opération nextPos de Map change de spécification, l'ambulance ne se déplacera plus correctement. |
| Simulator | Scenario | L'implémentation du Simulator dépend des spécifications de Scenario car si l'opération nextStep de Scénario change de spécification, le simulateur ne simulera pas les bonnes étapes. |

Tableau 1.4. Justification des liens USE du système.

# 2. Architecture physique

L'architecture physique des deux parties du logiciel développé sont présenté ci-dessous.

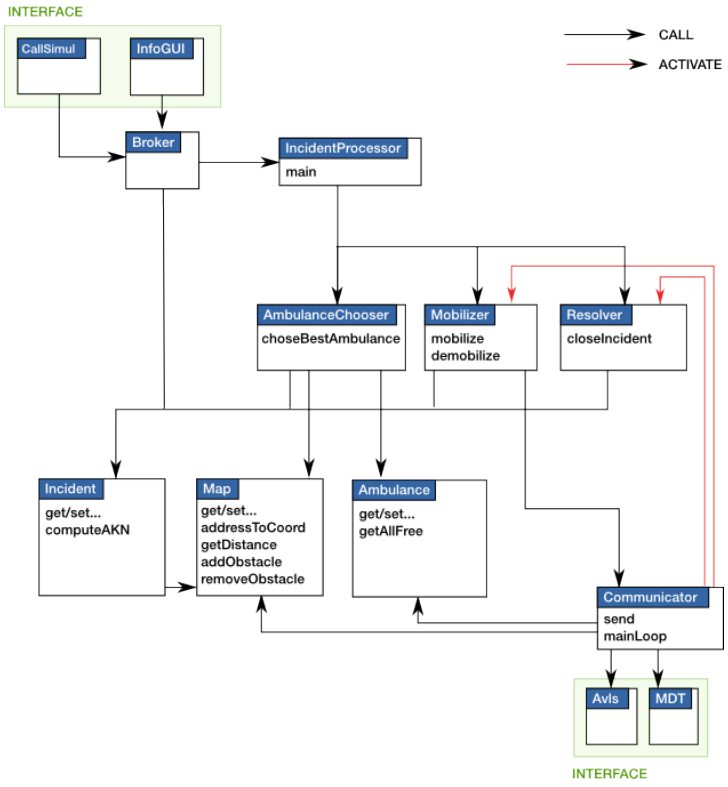


Figure 2.1. Architecture physique pour la partie du système.

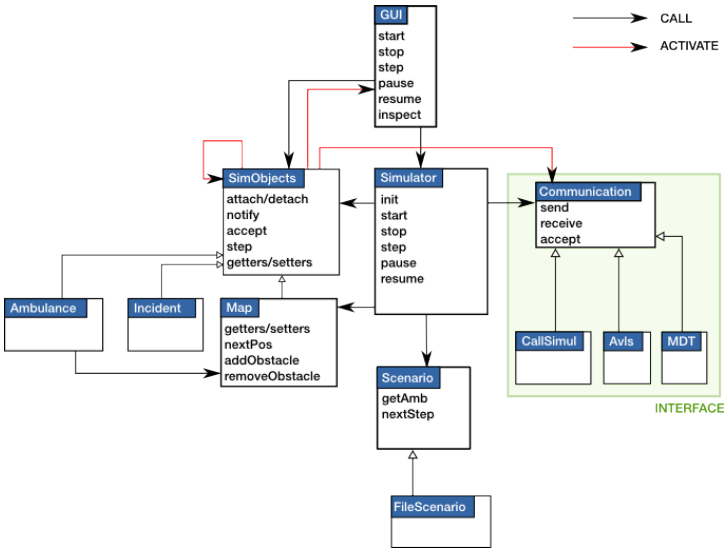


Figure 2.2. Architecture physique pour la partie du simulateur.

# 3. Évènements à l'interface

Cette section décrit les différents messages qui sont échangés entre le système logiciel et l'environement, simulé ou non. Ces évènement sont échangé aux travers des différentes interfaces de communications possible entre les deux lieux : l'AVLS, le MDT et l'interface d'entrée des informations pour les incidents.

**Incident :** L'événement incident est envoyé à partir du simulateur vers le système et comporte les informations suivantes : l'âge de la victime, enceinte ou non, la localisation de l'incident (typiquement l'adresse), et une description de l'incident.

**AVLSMessage :** L'événement AVLSMessage est envoyé à partir du simulateur vers le système et comporte la posisition (coordonnées géographique) et l'identifiant d'un ambulance.

**MDTMessage :** L'événement MDTMessage peut provenir soit du système soit du simulateur. Il y a plusieurs type de MDTMessage

| **nom du message** | **arguments** | **description** |
|---|---|---|
| mobilisationOrder | incidentID, incidentPosition, ambulanceID | Le message de mobilisation est envoyé par le système à une ambulance (AmbulcanceID) afin de la mobiliser pour un incident (incidentID) qui à lieu à la position (incidentPosition) |
| demobilisationOrder | incidentID, incidentPosition, ambulanceID | Le message demobilisationOrder est envoyé par le système à une ambulance (AmbulanceID) afin de la démobiliser pour l'incident (incidentId) se trouvant à la position (incidentPosition) |

Tableau 3.1. MDTMessage du système vers le simulateur

| **nom du message** | **arguments du message** | **description du message** |
|---|---|---|
| mobilisationConfirmation | incidentID, ambulanceID, un booleen yes/no | Un message de confirmation est envoyé par l'ambulance (ambulanceID) pour accepter (yes) ou refuser (no) l'ordre de mobilisation concernant l'incident (incidentID) |
| ambulanceBroken | ambulanceID | Message envoyé par l'ambulance (ambulanceID) pour signaler qu'elle est en panne |
| ambulanceRepaired | ambulanceID | Message envoyé par l'ambulance (ambulanceID) pour signaler qu'elle est réparée |
| obstacle | position | Un message Obstacle avec une position en argument est envoyé par une ambulance au système pour signaler qu'il existe une obstacle à cette position |
| incidentResolved | incidentID, ambulanceID | Un message incidentResolved est envoyé par l' ambulance (ambulanceID) au système pour signaler que l'intervention pour l'incident (incidentID) est résolue |
| destinationUnreachable | ambulanceID, incidentID | Une message destinationUnreachable est envoyé par l'ambulance (ambulanceID) au système pour signaler que la position de l'incident (incidentID) est inaccessible |

Tableau 3.2. MDTMessage du simulateur vers le système

# 4. Plan de développement

Cette partie présente le plan de développement de notre logiciel ainsi que la répartition du travail au sein du groupe.

Le travail est réparti en équipe de développeurs, ces développeurs sont des équipes de deux ou trois personnes au maximum, le but étant de minimiser les interactions tout en conservant une relecture et une liberté d'implémentation aux équipes.

Dans chacune des phases, l'équipe se voit assigner un ensemble de module à écrire et à tester, sur base des tests black-box précédemment conçu. Ces tests seront rédigé à l'aide JUnit et serviront également de tests de régressions. Les tests seront écrits à l'aide de JUnit 4 et le code java sera écrit en code compatible Java 6.

## 4.1. Phase 1 : La gestion de l'information

Cette phase permet de mettre en place tous les objets qui seront utilisé par le reste de l'application. Ces objets seront stocké, dans un premier temps, pour la durée de l'exécution du logiciel.

Les modules concernés par cette phase sont, pour la partie système : Incident, Map, Ambulance et pour la partie simulateur : SimObjects, Map. La répartition du travail pour les différents module et pour les équipes est décrite ci-dessous :

| Module | Équipe de développement | Équipe de test |
|---|---|---|
| Incident | Team A | Team B |
| Map | Team A | Team B |
| Ambulance | Team A | Team B |
| SimObjects | Team B | Team A |
| Map | Team B | Team A |

Fin de la première phase : 27 novembre.

## 4.2. Phase 2 : Communication

Cette phase met en place la communication entre les deux mondes.

Les modules concernés par cette seconde phase sont les suivants, pour le système : Communicator, Broker et pour le simulateur : Communication, CallSimul, AVLS et MDT. À nouveau, la répartition est présentée dans le tableau suivant :

| Module | Équipe de développement | Équipe de test |
|---|---|---|
| Communicator | Team A | Team B |
| Broker | Team A | Team B |
| Communication | Team B | Team A |
| CallSimul | Team B | Team A |
| AVLS | Team B | Team A |
| MDT | Team B | Team A |

Fin de la seconde phase : 2 décembre.

## 4.3. Phase 3 : Le coeur

Cette phase va s'appuyer sur les phases précédentes afin de les exploiter et de faire en sorte que le logiciel fasse ce pourquoi ce dernier a été conçu.

Les modules concernés sont les suivants : pour le système : IncidentProcessor, AmbulanceChooser, Mobilizer et Resolver, pour le simulateur : Simulator, Scenario et FileScenario. À nouveau, la répartition est présentée dans le tableau suivant :

| Module | Équipe de développement | Équipe de test |
|---|---|---|
| IncidentProcessor | Team A | Team B |
| AmbulanceChooser | Team A | Team B |
| Mobilizer | Team A | Team B |
| Resolver | Team A | Team B |
| Simulator | Team B | Team A |
| Scenario | Team B | Team A |
| FileScenario | Team B | Team A |

Fin de la seconde phase : 11 décembre.

## 4.4. Phase 4 : L'enrichissement

Cette phase est l'occasion d'ajouter des modules à notre architecture afin de proposer une plus grand nombre de fonctionnalité.

En fonction du temps et des affinités des équipes, il sera possible d'implémenter certaines de ces fonctionnalités.

Parmis ces fonctionnalités, nous avons :
- Utilisation d'une base de donnée relationnelle pour sauvegarder les données de manière pérenne.
- L'ajout d'un module de statistique du coté du simulateur
- L'ajout d'un module de statistique du coté du système
- L'ajout d'un module s'occupant du placement des ambulances afin de maximiser la couverture géographique
- L'utilisation d'une carte plus complexe, pour louvain-la-neuve par exemple.
- Le développement de canaux de communication supplémentaire (Radio, téléphonne, etc.)
- L'ajout de scénario probabiliste
- L'ajout de scénario généré manuellement