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

1.1. Architectures évaluées

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

2. Architecture physique

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
incidentCancelled	incidentID, ambulanceID	Un message incidentCancelled est envoyé par l'ambulance (ambulanceID) au système pour signaler que l'intervention pour l'incident (incidentID) est annulée
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 B	Team A
Ambulance	Team A	Team B
SimObjects	Team A	Team B
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 B	Team A
Broker	Team B	Team A
Communication	Team B	Team A
CallSimul	Team A	Team B
AVLS	Team A	Team B
MDT	Team A	Team B

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 B	Team A
Mobilizer	Team A	Team B
Resolver	Team A	Team B
Simulator	Team B	Team A
Scenario	Team A	Team B
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

– ...

5. Spécifications externes des modules

6. 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 6.1. 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
incidentCancelled	incidentID, ambulanceID	Un message incidentCancelled est envoyé par l'ambulance (ambulanceID) au système pour signaler que l'intervention pour l'incident (incidentID) est annulée
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 6.2. MDTMessage du simulateur vers le système

7. 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.

7.1. 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 B	Team A
Ambulance	Team A	Team B
SimObjects	Team A	Team B
Map	Team B	Team A

Fin de la première phase : 27 novembre.

7.2. 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 B	Team A
Broker	Team B	Team A
Communication	Team B	Team A
CallSimul	Team A	Team B
AVLS	Team A	Team B
MDT	Team A	Team B

Fin de la seconde phase : 2 décembre.

7.3. 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 B	Team A
Mobilizer	Team A	Team B
Resolver	Team A	Team B
Simulator	Team B	Team A
Scenario	Team A	Team B
FileScenario	Team B	Team A

Fin de la seconde phase : 11 décembre.

7.4. 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

– ...

8. Spécifications externes des modules