LAS : London Ambulance System

Analyse des besoins

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

Table des matières

Introduction

La méthode du *Requirement Engineering* permet, entre autre, d'aborder des problématiques complexes afin d'en tirer l'essentiel et l'utile pour la conception de logiciels critiques. Dans le cadre du cours de génie logiciel, il nous est demandé de concevoir un logiciel de gestion d'ambulances similaire au logiciel utilisé à Londres il y a quelques années.

Ce rapport présente la première partie du projet : l'analyse des besoins. Nous avons utilisé une modélisation par buts pour guider l'ensemble de l'analyse. Le rapport est découpé en deux parties : la première explique la manière dont nous avons construit notre modèle et illustre les liens inter-modèles existants. La seconde partie liste les buts et les objets modélisés à l'aide du logiciel Objectiver.

1. Analyse

Ce chapitre présente l'analyse qui a été faite du domaine. Le tout est volotairement découpé en multiple section afin de présenter le plus clairement possible les liens existants entre les modèles.

La section 1.1 présente le modèle de buts, base du reste de la construction. La section 1.2 présente les objets manipulés et utilisés dans les modèles. La section 1.4 présente les scénarios ainsi que les machines à états définies dans le cadre de cette analyse. La section 1.5 présente les différentes opérations afférants aux buts et exécutées par les agents. La section 1.6 présente enfin les obstacles, point de départ pour une seconde itération.

1.1. Modèle de buts

Dans cette partie, nous vous présentons le modèle de buts. En particulier, nous vous présentons la formalisation des états impliqués par les buts que nous avons utilisés pour découler de manière rapide et cohérente les opérations, les machines à états et augmenter la cohérence de notre modèle de buts.

1.1.1. Première couche

Cette section présente les divers buts de plus haut niveau. Ces buts sont les buts principaux de notre système.

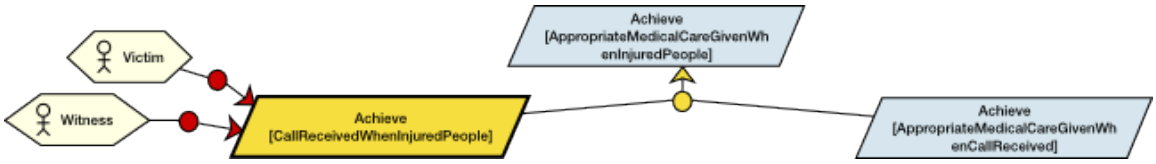

FIG. 1.1. Les buts de plus haut niveau

$$
\begin{aligned}
AppropriateMedicalCareGiven(i : Incident) &= \exists AmbulanceMobilized(a, i), \\
&\wedge AmbulanceOnScene(a, i) \\
&\wedge \#a.medicalCare = 1 \\
&\wedge \#i.victim.medicalCare = 1 \\
&\wedge a.medicalCare \\
&\rightarrow i.victim.medicalCare \\
CallReceived(i : Incident) &= \exists c : Call, \\
&c.about \rightarrow i \\
&\wedge \#c.about = 1 \\
&\wedge \#i.about = 1
\end{aligned}
$$

1.1.2. Seconde couche

La « seconde couche »est constituée des enfants des buts feuilles de la première couche. Nous les avons présentés ci-dessous.

Notons que le *Achieve[AmbulanceChosen]* est en fait *Achieve[-AmbulanceChosenWhenAvailabilityKnownAndAmbulanceKind-KnownAndAccurateAmbulancePositionKnown]* pour des raisons évidentes de concision et de lisibilité, nous avons changé l'intitulé.

La formalisation des états atteints par les buts feuilles du premier fils sont les suivants.

$$
\begin{aligned}
IncidentInfoKnown(i : IncidentInfo) &= IncidentInfoProcessed(i) \\
AmbulanceMobilized(a : Ambulance) &= \exists i : Incident, \\
&MobilizationOrderConfirmed(a, i)
\end{aligned}
$$

La formalisation des états atteints par les buts feuilles du

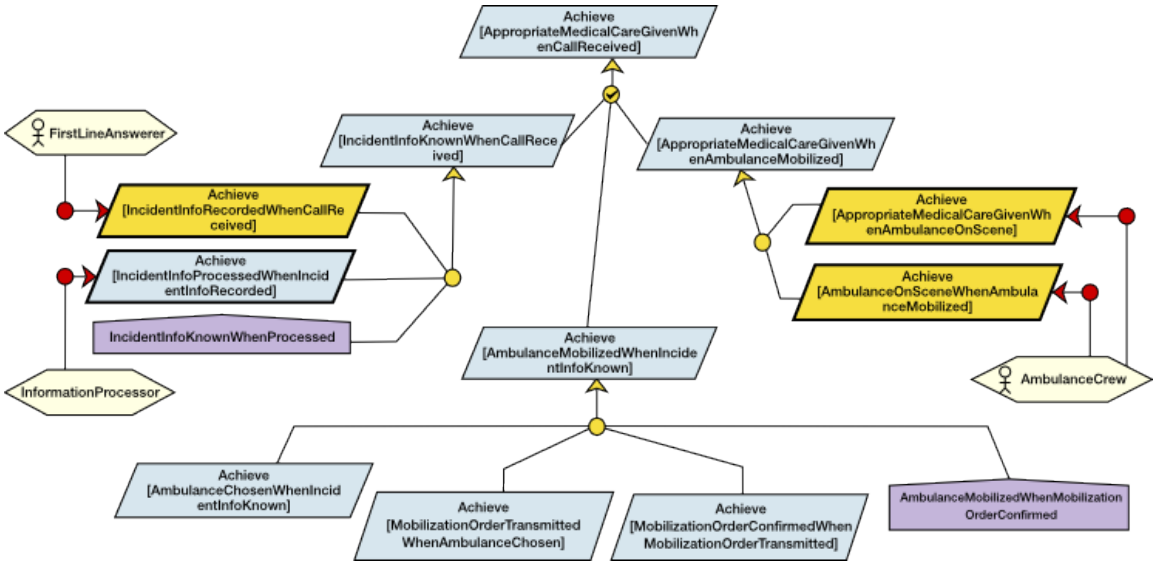

FIG. 1.2. Le détail des sous-buts de haut niveau.

second fils sont les suivants.

$$
\begin{aligned}
IncidentInfoRecorded(c : Call) \quad = \quad & \exists j : IncidentInfo, \exists i : Incident, \\
& c.about \rightarrow i \\
& \wedge j.reporting \rightarrow c \\
& \wedge j.localisation \neq nul \\
& \wedge j.description \neq null \\
& \wedge j.victimAge \neq null \\
& \wedge j.victimPregrant \neq null \\
& \wedge j.id \neq null \\
IncidentInfoProcessed(j : IncidentInfo) \quad = \quad & j.pos \neq null \\
& \wedge j.ambulanceKindNeeded \neq null \\
& \exists c : Call, \exists i : Incident, \\
& (c.about \rightarrow i \wedge c.reporting \rightarrow j)
\end{aligned}
$$

La formalisation des états atteints par les buts feuilles du troisième fils sont les suivants.

$$
\begin{aligned}
AmbulanceChosen(i : IncidentInfo) &= \exists a : AmbulanceInfo \\
&\quad \wedge a.choice = 1 \\
&\quad \wedge \#i.choice = 1 \\
&\quad \wedge a.choice \rightarrow i \\
MobilizationOrderTransmitted(a : Ambulance, i : Incident) & \\
&= \exists m : MobilizationOrder, \\
&\quad m.sender \rightarrow a \\
&\quad \wedge m.incident = 1 \\
MobilizationOrderConfirmed(a : Ambulance, i : Incident) & \\
&= \exists m : MoblisationOrder : \\
&\quad m.sender \rightarrow a \wedge m.incidentId = i.id \\
&\quad \wedge m.acknowledgement = True
\end{aligned}
$$

La formalisation des états atteints par les buts feuilles du dernier fils sont les suivants.

$$
AmbulanceOnScene(a : Ambulance, I : Incident) \quad = \quad a.pos \approxeq i.pos
$$

1.1.3. Troisième couche

La troisième couche est constituée des buts enfants des buts de la seconde couche. Les figures 1.3, 1.4, et 1.5 reprennent les sous-arbres des buts. Aussi, nous présentons la formalisation de certains des états atteints par les buts.

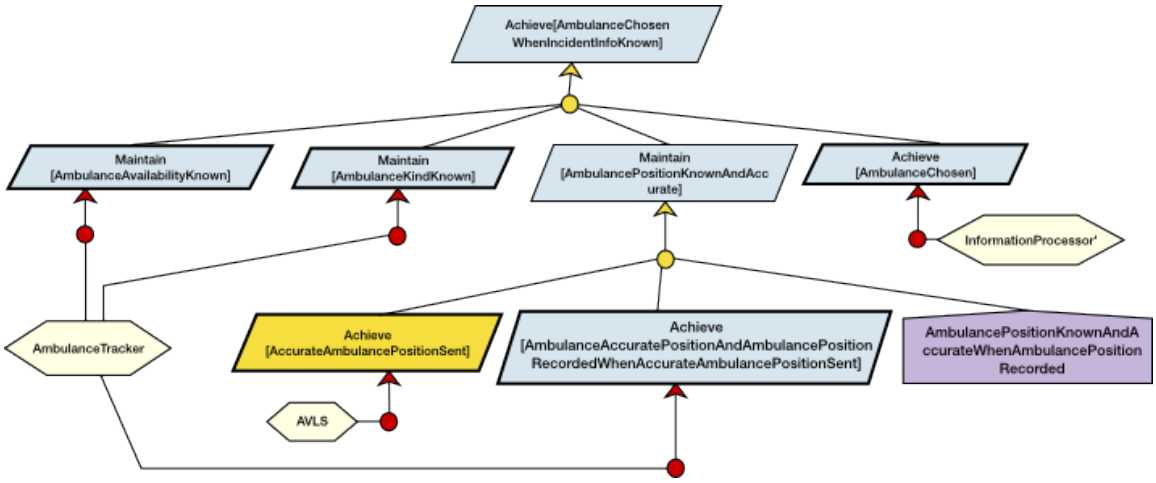

FIG. 1.3. Troisième niveau, raffinement du but *AmbulanceChosenWhenIncidentInfoKnown*

$$
\begin{aligned}
AmbulanceAvailabilityKnown(a : Ambulance) &= \exists b : AmbulanceInfo, \\
&\quad b.id = a.id \\
&\quad \wedge \#b.mobilized = \#a.mobilized \\
AmbulanceKindKnown(a : Ambulance) &= \exists b : AmbulanceInfo, \\
&\quad b.id = a.id \\
&\quad \wedge typeof(a) = b.kind \\
AmbulancePositionKnown(a : Ambulance) &= \exists b : AmbulanceInfo, \\
&\quad b.id = a.id \\
&\quad \wedge a.pos \approxeq b.pos \\
PositionAccurate(a : Ambulance) &= \exists b : AmbulanceInfo, \\
&\quad a.id = b.id \\
&\quad \wedge b.pos = a.pos
\end{aligned}
$$

1.2. Modèle des objets

Dans cette partie, nous présentons la conception de notre modèle objet. Ce modèle est un modèle du domaine c'est-à-dire que les classes, associations et attributs correspondent à des éléments du domaine (éventuellement partagés avec le logiciel), apparus lors de l'analyse des objectifs.

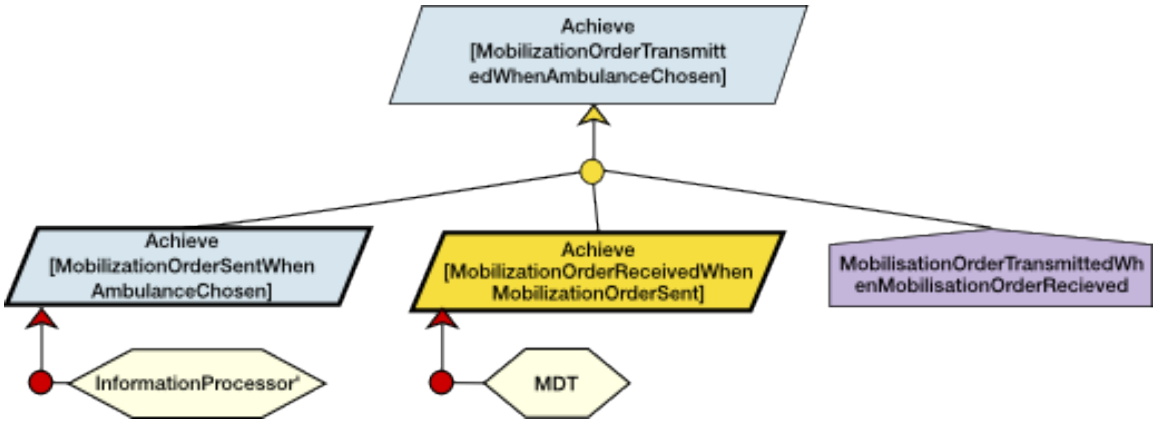

FIG. 1.4. La suite du troisième niveau, raffinement du but reponsable de l'envoi de l'ordre de mobilisation.

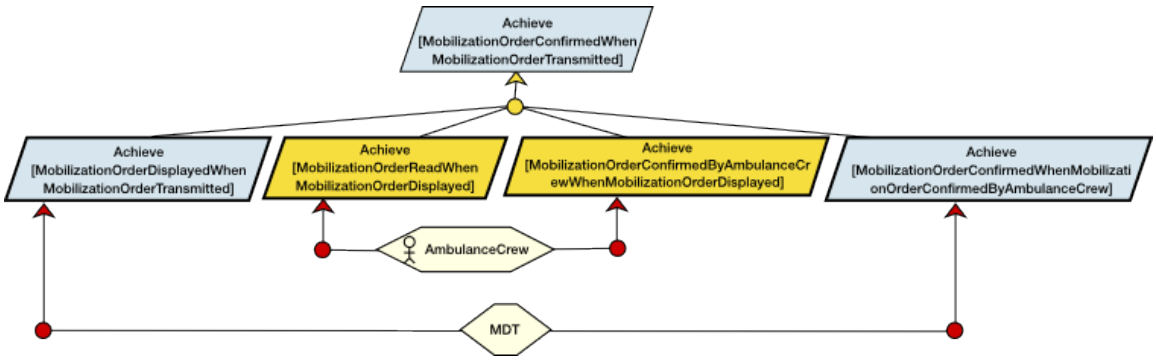

FIG. 1.5. La fin du troisième niveau, raffinement du but responsable de la confirmation de la mobilisation.

Au cours de l'élaboration du diagramme d'objets, nous avons dû déterminer quelles classes, quelles associations et quels attributs du domaine devaient avoir une image manipulée par le logiciel comme données persistantes dans une base de données conceptuelle, et comme classe abstraite intervenant dans des composants de l'architecture à développer.

À partir de notre modèle de buts nous avons donc abouti à un modèle d'objets pour le moins solide et respectant ainsi la cohérence intermodèle.

Les classes représentant les éléments du domaine sont illustrées dans la figure 1.6. Certaines d'entre elles ont une "image" dans le système, c'est le cas de la classe Incident et Ambulance dont les classes "miroirs" respectives sont IncidentInfo et AmbulanceInfo (figure 1.8). Il existe donc une association "tracking" entre ces objets réels et leur représentation dans le système.

Cette association existe grâce aux objets que l'on nomme "interface" (figure 1.7) et qui assurent la communication entre les objets du monde réel et leur représentation dans le système.

Cette démarche nous pousse à introduire de nouveaux buts non fonctionnels notamment de précision, exprimant que l'état de la base de données du logiciel doit refléter fidèlement l'état des objets/associations de l'environnement que les objets/associations logiciels représentent.

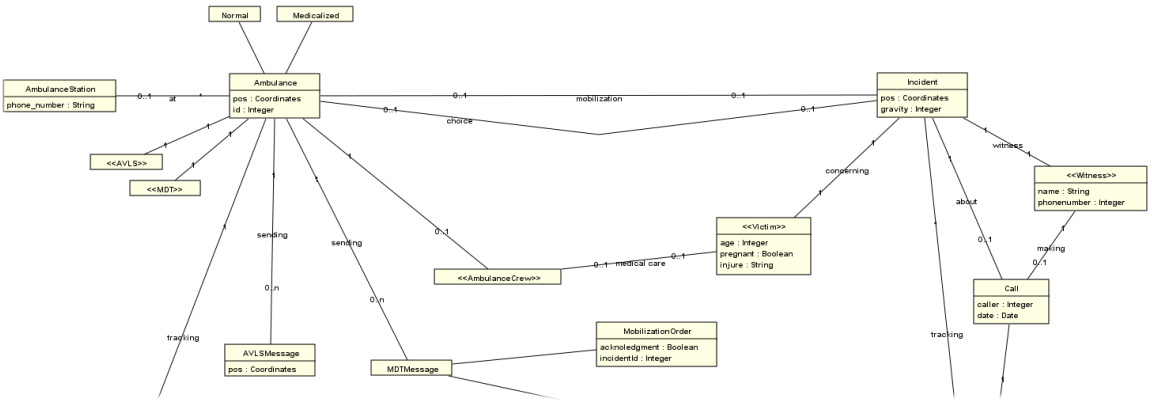

FIG. 1.6. Objets appartenant à l'environnement

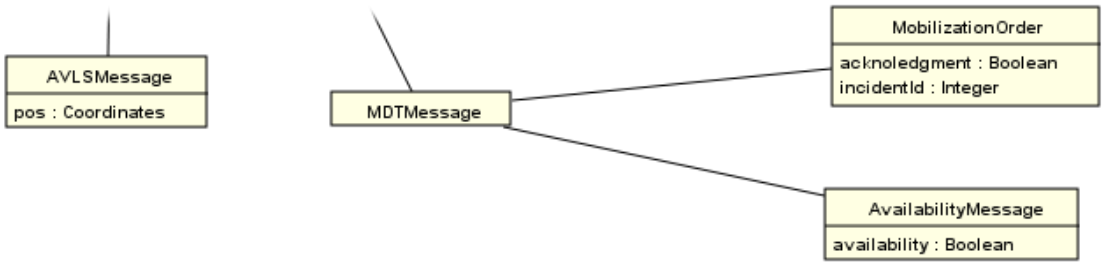

FIG. 1.7. Objets à l'interface de l'environnement et du logiciel

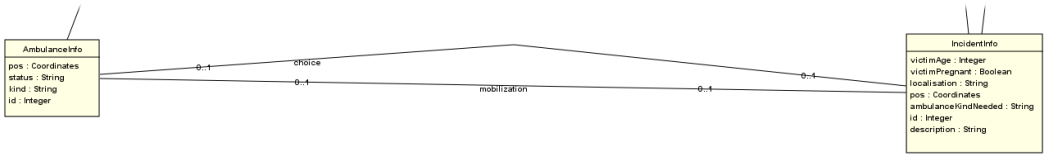

FIG. 1.8. Objets appartenant au logiciel

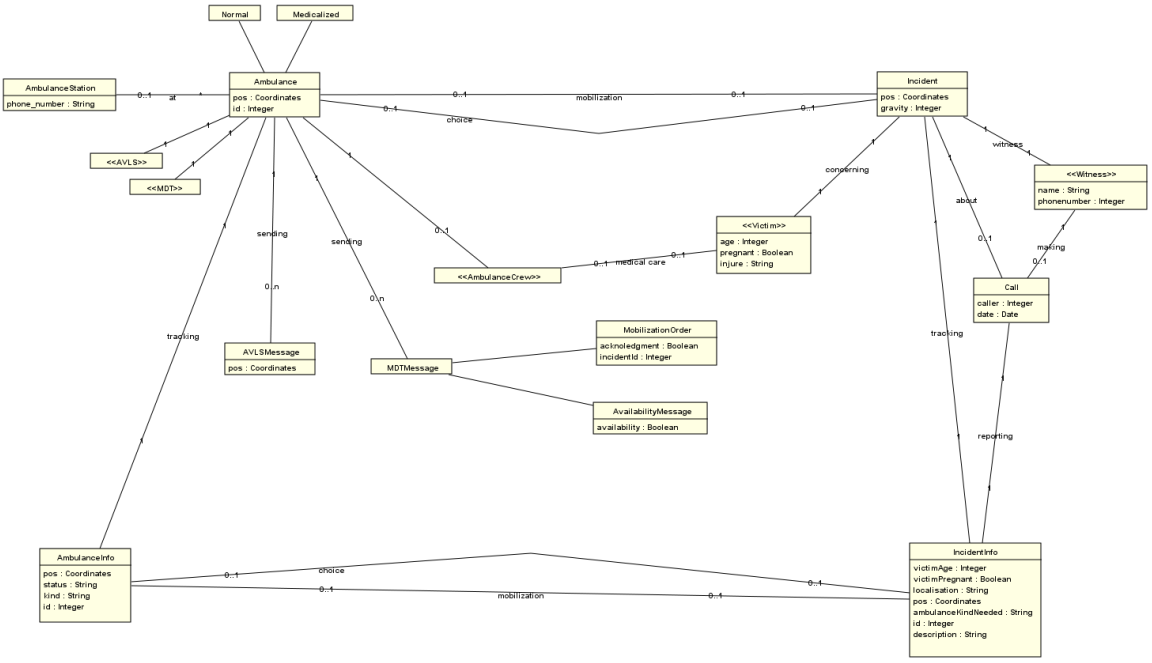

FIG. 1.9. Schéma de l'ensemble des objets

1.3. Modèle des agents

Le diagramme de contexte met en lumière les différentes interactions entre agents du système au travers des variables qu'ils contrôlent et monitorent. Il est construit à partir du modèle de buts (où chaque but feuille est assigné à un agent) et du modèle objet.

L'intérêt principal du diagramme de contexte est de représenter les interactions à l'intérieur du logiciel (interactions agent logiciel/agent logiciel) et les interactions à l'interface (interactions agent logiciel/agent d'environnement).

Dans le cas où des noms de classes sont utilisés au lieu d'attributs ou d'associations, il est sous-entendu que l'agent qui contrôle les attributs de la classs mentionnée les contrôle tous et l'agent qui monitore les monitore tous. Cette notation a été utilisée pour rendre le diagramme plus concis.

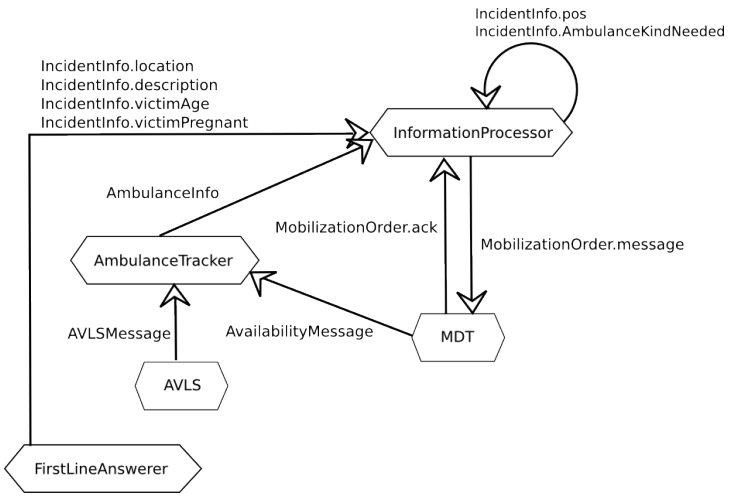

FIG. 1.10. Diagramme de contexte des interactions logicielles et d'interface

1.4. Modèle de comportement

Cette section présente les scénarios (1.4.1) ainsi que les machines à états (1.4.2) des ambulances et des incidents.

1.4.1. Scénario

Les scénarios les plus intéressants pour la compréhension du modèle sont fournis dans cette section. Dans les cas où les buts feuilles semblaient trop simples pour obtenir un scénario digne d'intérêt, les buts parents ont été utilisés. Notons ici que nous avons omis le scénario correspondant au but Maintain[AmbulanceKindKnown] car le type d'ambulance est assumé constant. Cette situation ne génère alors pas de communication entre les différents agents.

Pour chacun des scénarios présentés, le but correspondant est mentionné. De plus, les états sous-entendus pour chacun des scénarios sont présentés en marge des différents scénarios, en turquoise pour les états de l'incident et en gris/violet pour les états de l'ambulance. Ces états correspondent aux états présentés dans la section 1.4.2.

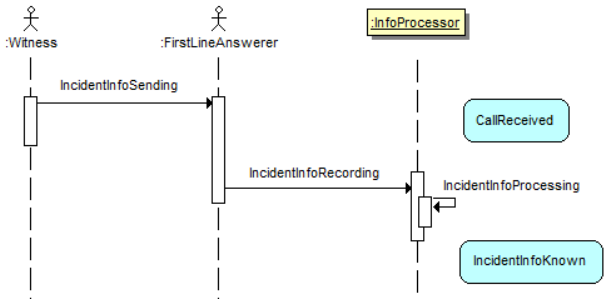

FIG. 1.11. Ce scénario illustre le but *Achieve[IncidentInfoKnownWhenCallReceived]*. On peut y voir le début de la chaîne.

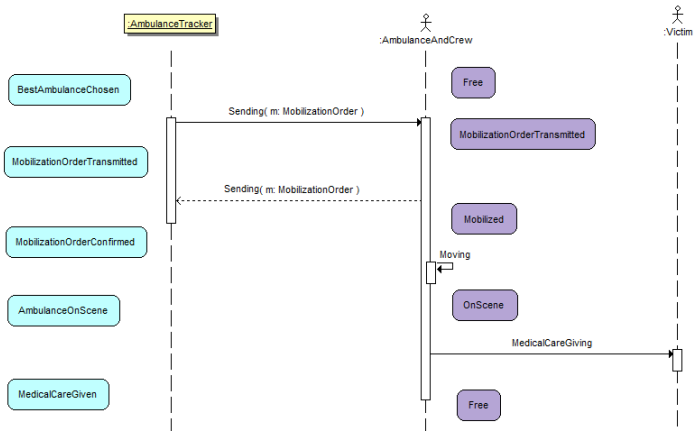

FIG. 1.12. Ce scénario illustre la mobilisation d'une ambulance jusqu'au soins prodigués à la victime. Les bulles bleues indiquent les états de l'*ambulance* et de l'*ambulanceInfo*, les bulles mauves quant à elles illustrent les états de l'*incident* et de l'*incidentInfo*

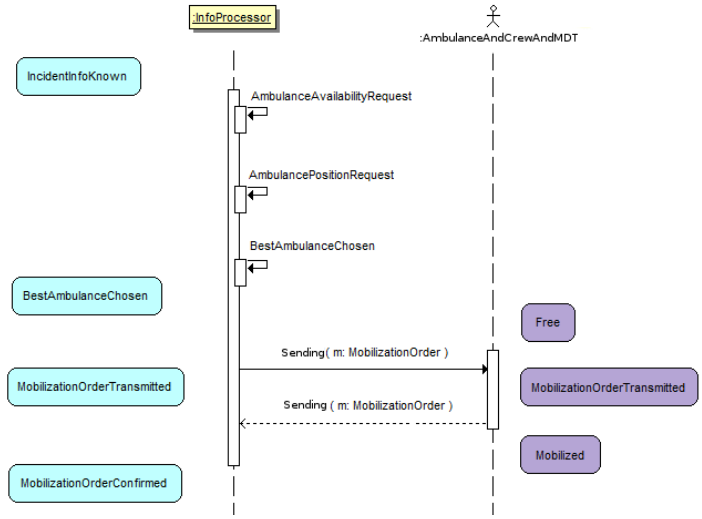

FIG. 1.13. Ce scénario illustre la mobilisation d'une ambulance.

1.4.2. Machines à état

La section présente une généralisation des scénarios présentés dans la section 1.4.1. Tel que suggéré par le tuteur, les machines à états dans le cas de l'ambulance et de l'incident correspondent à l'aggrégation des états des objets réels et des *tracking objects*. De plus, il n'y a pas de distinction réalisée entre *AmbulanceCrew* et *Ambulance*. Ces choix ont été réalisés pour mettre l'emphase sur le comportement du système et éviter la

complexité liée à la présence des objets nécessaires à la transmission de l'information entre l'environnement et le logiciel.

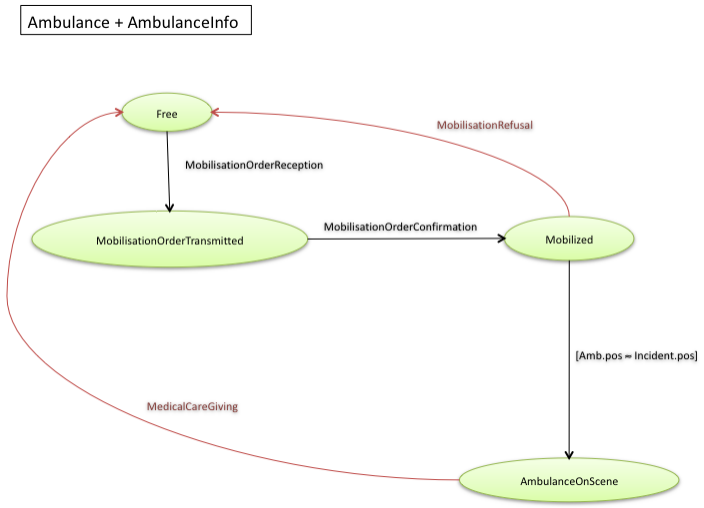

FIG. 1.14. États de l'Ambulance et de l'AmbulanceInfo

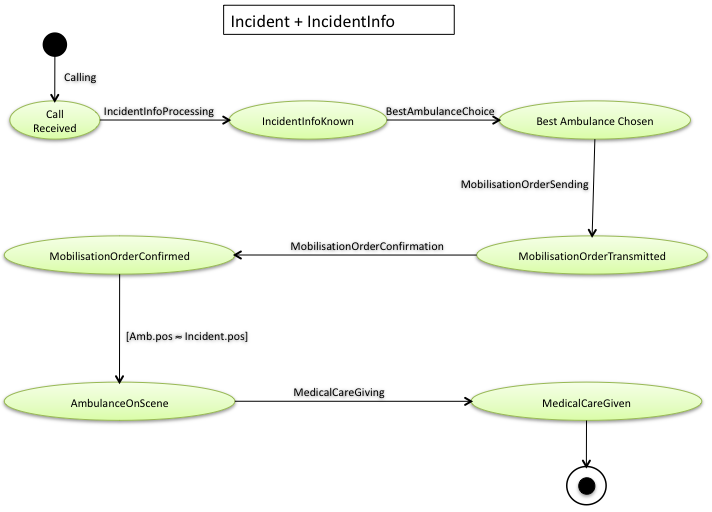

FIG. 1.15. États de l'Incident et l'IncidentInfo

1.5. Modèle des opérations

Cette section présente un ensemble d'opérations. Nous avons choisi les quatres opérations correspondant à des buts feuilles et assignées à des agents logiciels à développer.

Afin d'illustrer les liens entre les modèles, nous reprenons le nom du but qui est opérationnalisé par l'opération, le nom de l'agent qui effectuera l'opération, l'état à atteindre après cette opération et enfin, l'évènement attaché à cette opération.

1.5.1. Traitement des données

L'opération *processIncidentInfo* est une opération qui est effectuée par l'*InformationProcessor*, partie logicielle gérant tout le traitement des données au sein de l'application. Cette opération consiste à transformer les informations reçues par l'appellant en informations directement exploitables par le système informatique. Typiquement, cette opération déterminera le type d'ambulance qui sera nécessaire et les positions « informatisées »de la localisation. Par exemple, si le témoin a donné comme information que la victime était une femme enceinte inconsciente sur la place de l'accueil à louvain-la-neuve, le système calculerait qu'il faut une ambulance médicalisée et que l'incident à eu lieu à la position $(50.670932, 4.616318)$.

La figure 1.16 illustre le but opérationnalisé par cette opération, en montrant où cette dernière est placée dans le diagramme de but. La figure 1.15 quant à elle montre les transitions d'état de l'incident et donc, la transition correspondant à l'opération. L'opération est synthétisée au tableau 1.1.

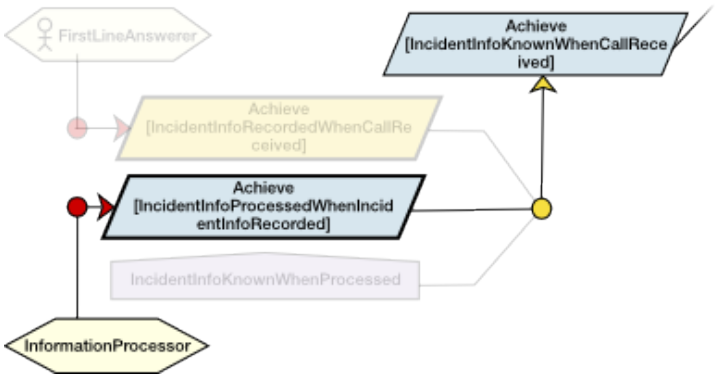

FIG. 1.16. L'opération *Achieve[processIncidentInfo]* opérationnalise *Achive[IncidentInfoProcessedWhenIncidentInfoRecorded]*.

Goal	Achieve[IncidentInfoProcessed-WhenIncidentInfoRecorded]
Agent	InformationProcessor
État atteint	IncidentInfoKnown
Évènement lié	IncidentInfoProcessing
In	$i : IncidentInfo$
Out	$i : IncidentInfo$
Pre	$\exists c : Call, \exists j : Incident(c.about \rightarrow j \wedge c.reporting \rightarrow i)$
Post	$i.pos \neq null \wedge i.ambulanceKindNeeded \neq null$ La position (pos) contenue dans i correspond à la position (localisation) de i sous forme exploitable par le système. Le type d'ambulance nécessaire de i est calculé selon les règles données par le gouvernement et sur base des informations présentes dans i.

TAB. 1.1. L'opération *Achieve[processIncidentInfo]*

1.5.2. Enregistrement de la position

L'ambulance, élément du monde réel, se déplace avec le temps. Dans le système informatique, il nous faut naturellement refléter cette réalité au sein du logiciel. L'opération *Achieve[recordAccurateAmbulancePosition]* se charge d'enregistrer la position précise envoyée par l'ambulance dans le logiciel. La figure 1.17 illustre le but opérationnalisé et le tableau 1.2 synthétise l'opération.

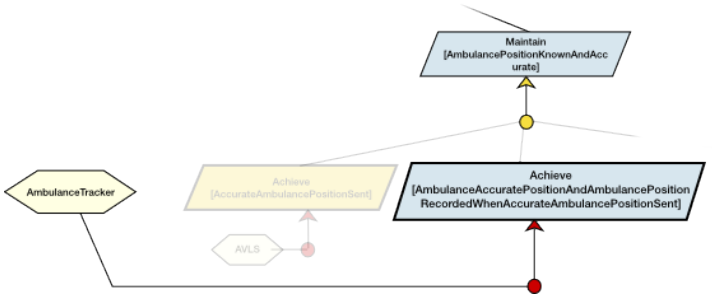

FIG. 1.17. L'opération *Achieve[recordAccurateAmbulancePosition]* enregistre l'opération. On peut voir le but correspondant sur la figure.

Goal	Achieve[AccurateAmbulancePositionRecorded WhenAccurateAmbulancePositionSent]
Agent	AmbulanceTracker
Évènement lié	AccurateAmbulancePositionRecording
In	$a : Ambulance$
Out	$b : AmbulanceInfo$
Pre	$\exists b : ambulanceInfo : a.id = b.id$
Post	$b.pos = a.pos$

TAB. 1.2. L'opération *Achieve[recordAccurateAmbulancePosition]*

1.5.3. Choix de l'ambulance

Lors d'un incident, le système envoie une ambulance. Pour pouvoir envoyer la bonne ambulance, le logiciel doit nécessairement choisir quel ambulance envoyer. L'opération *chooseAmbulance* s'occupe de choisir l'ambulance à envoyer. La figure 1.18 contextualise le but opérationnalisé et le tableau 1.3 synthétise l'opération.

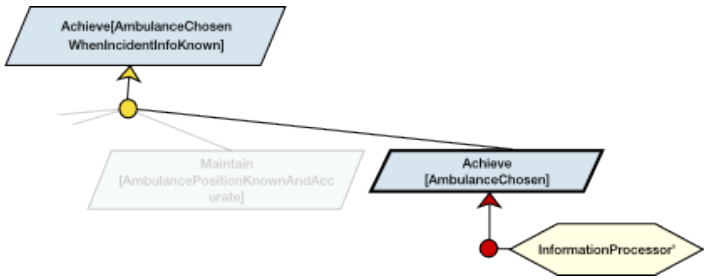

FIG. 1.18. L'opération *chooseAmbulance* opérationalise le but *Achieve[AmbulanceChosen]*, le but est indiqué sur la figure, les autres étant plus claires.

Goal	Achieve[AmbulanceChosen]
Agent	InfoProcessor
État atteint	BestAmbulanceChosen
Évènement lié	BestAmbulanceChoice
In	$i : IncidentInfo$
Out	$a : AmbulanceInfo$
Pre	$\exists a : AmbulanceInfo : \#a.mobilisation = 0 \wedge \#a.choice = 0$
Post	$\#a.choice = 1 \wedge \#i.choice = 1 \wedge i.choice \rightarrow a$

TAB. 1.3. L'opération *Achieve[chooseAmbulance]*

1.5.4. Envoi de l'ordre de mobilisation

Une fois l'ambulance choisie, il est nécessaire de la mobiliser afin qu'elle se rende sur place. L'opération *Achieve[sendMobilizationOrder]* s'occupe d'envoyer l'ordre de mobilisation à l'ambulance.

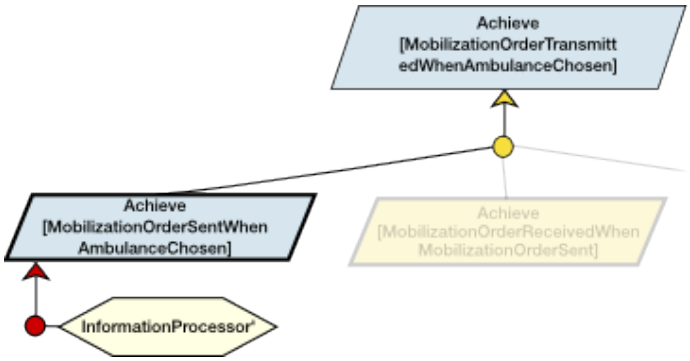

FIG. 1.19. L'opération *sendMobilizationOrder* opérationnalise le but *Achieve[MobilizationOrderSentWhenAmbulanceChosen]*

Goal	Achieve[MobilizationOrderSentWhenAmbulanceChosen]
Agent	InfoProcessor
État atteint	MobilizationOrderTransmitted
Évènement lié	MobilizationOrderSending
In	$a : AmbulanceInfo, i : IncidentInfo$
Out	$m : MobilizationOrder$
Pre	$\#a.choice = 1 \wedge a.choice \rightarrow i$
Post	$\exists m : MobilizationOrder : m.ambulanceId = a \wedge m.incidentId = i$

TAB. 1.4. L'opération *sendMobilizationOrder*

1.6. Résolution d'obstacles

Le but principal de la découverte d'obstacles est de rendre l'analyse du problème plus complète en y incluant les cas limites et les exceptions. La démarche est la suivante : prendre un but feuille (exigence ou hypothèse), le nier, raffiner autant que possible cette négation pour enfin résoudre chacun de ces raffinements. Cela permet alors de définir de nouvelles exigences et hypothèses qui résolvent les cas limites et d'exception et ainsi obtenir un modèle plus complet et plus sûr.

Pour être complet, il faudrait obstruer chaque but feuille du modèle de buts idéal. Dans le présent rapport, seuls quelques buts feuilles intéressants ont été obstrués.

1.6.1. Achieve[AccurateAmbulancePositionRecorded When AccurateAmbulancePositionSent]

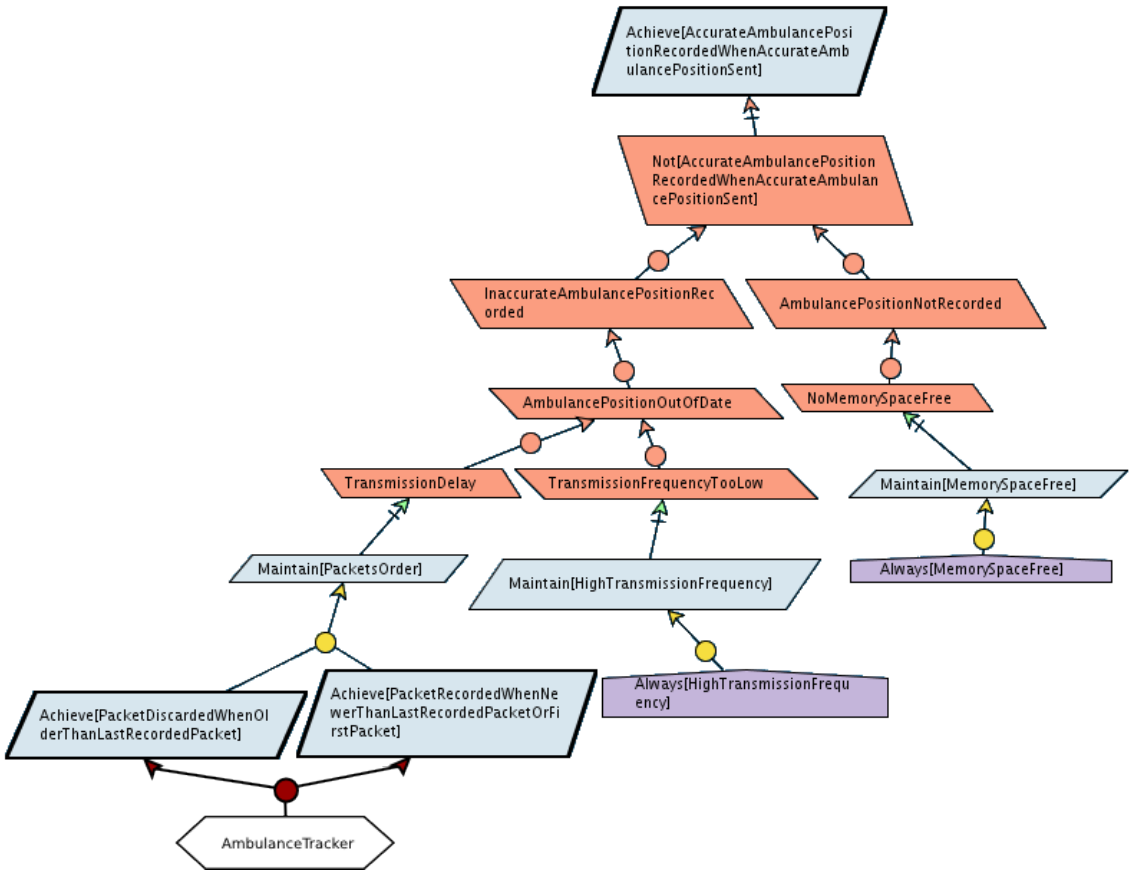

FIG. 1.20. Diagramme de résolution d'obstacle

Le but qui nous concerne ici est d'assurer que la précision des données envoyées est garantie à la réception et que ces données puissent être enregistrées. Les obstacles et leur résolution coulent de source.

La propriété du domaine *Always[HighTransmissionFrequency]* est assurée par une configuration correcte de l'AVLS. *Always[-MemorySpaceFree]* est, quant à elle, assurée par l'utilisation d'un matériel informatique adéquat pour faire tourner le système logiciel.

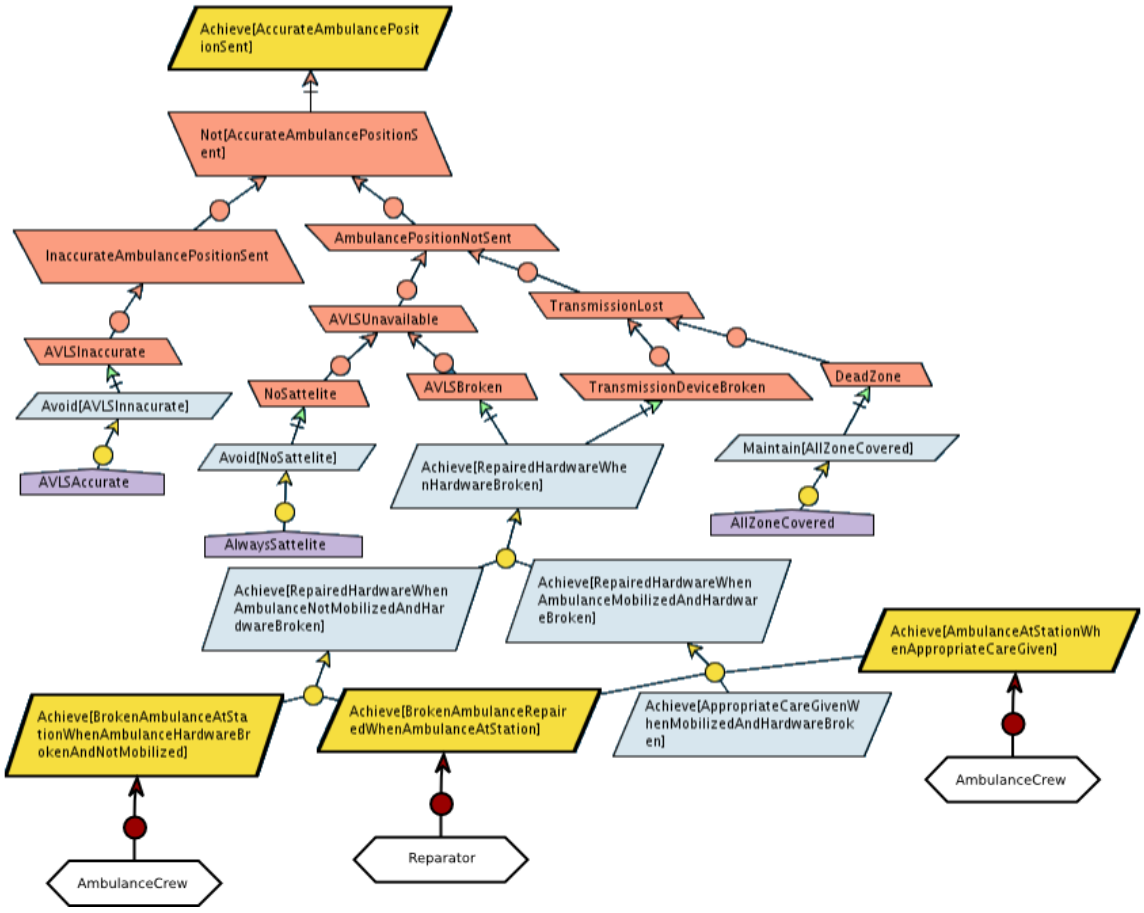

FIG. 1.21. Diagramme de résolution d'obstacle

1.6.2. Achieve[AccurateAmbulancePositionSent]

Ce but correspond à l'envoi des données de l'AVLS et de s'assurer que celles-ci sont correctes.

L'obstacle dont la résolution est la plus intéressante est le non fonctionnement du matériel responsable de l'exécution de ce but.

La résolution distingue deux cas, celui où l'ambulance est libre, et celui où elle est mobilisée. Dans le premier cas, on envoie directement l'ambulance à une station où elle sera réparée. Dans le second cas, l'ambulance effectue son travail et retourne ensuite à la station pour réparation.

Cette résolution est incomplète car notre modèle des buts et objet ne permet pas de gérer les objectifs suivants :

– Empêcher de choisir ou de mobiliser l'ambulance lors-

qu'elle doit être réparée ou est en réparation,
– Ne pas mobiliser une ambulance choisie si elle a eu un problème entre temps.

De plus, d'autres soucis apparaissent ; L'état "AmbulanceOnScene" ne saura être détecté par le système informatique dans le cas d'une ambulance dysfonctionelle.

Pour résoudre ces problèmes il faut ajouter un attribut "broken" à l'ambulance et modifier les définitions des buts de choix et de mobilisation de l'ambulance.

De plus l'AVLS et le système de transmission ne sont sans doute pas les seuls appareils de l'ambulance pouvant tomber en panne, le but de réparation correspond très certainement à un but de plus haut niveau consistant à maintenir l'ambulance en état de marche.

1.6.3. Achieve[AmbulanceOnSceneWhenAmbulanceMobilized]

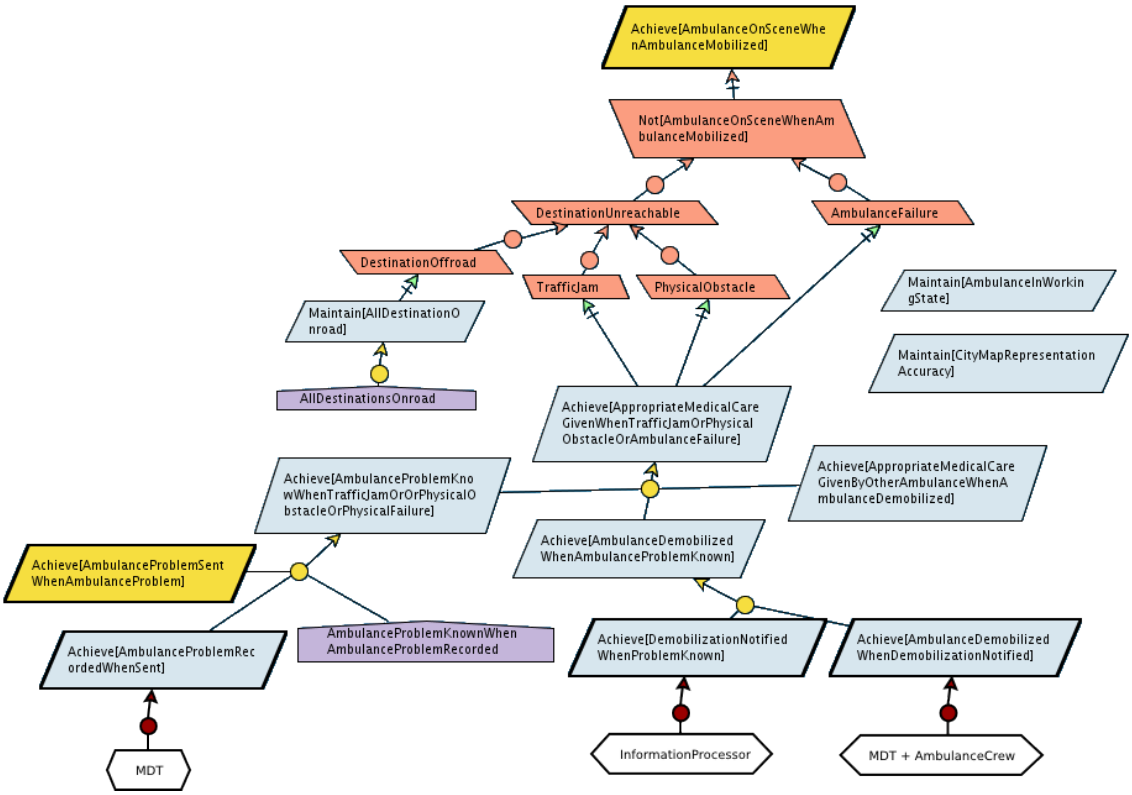

FIG. 1.22. Diagramme de résolution d'obstacle

Ce but correspond à l'arrivée de l'ambulance sur les lieux indiqués de l'incident après sa mobilisation.

Le raffinement consistant à démobiliser une ambulance accidentée ou coincée peut sembler superflu, mais est justifié par le respect des cardinalités de notre modèle objet.

Le but *Achieve[MedicalCareGivenByOtherAmbulanceWhen-AmbulanceDemobilized]* n'est manifestement pas complet ni suffisamment raffiné.

En fait, deux alternatives s'offrent à nous. La première et la moins bonne consiste à répliquer l'entièreté de l'arbre des buts *Achieve[MedicalCareGiven When AmbulanceMobilized]* comme raffinement du précédent. L'autre alternative consiste à modifier les buts parents afin qu'ils se chargent de mobiliser l'ambulance de secours. Pour cela, il faut transformer *Achieve[AmbulanceMobilized When IncidentInformationKnown]* par un *Maintain[WorkingAnbulanceMobilizedWhenIncident InformationKnownAndNotResolved]* Ce but s'assurera donc qu'il y a toujours une ambulance en état de marche mobilisée pour chaque incident.

Cependant cela nécessite de définir deux états supplémentaires : *Working(a :Ambulance)* et *Resolved(i :IncidentInformation).* L'ajout de ces deux états nécessite de modifier notre modèle objet et d'ajouter de nouveaux buts et raffinements, à identifier lors d'une deuxième passe d'analyse et de réflexion sur notre modèle.

Enfin, Cette résolution n'est pas non plus complète. Il faut en effet modifier la représentation de la carte du système de routage et de choix d'ambulance pour que celui-ci n'envoie pas une deuxième ambulance dans un embouteillage.

Pour finir, il faut aller rechercher l'ambulance accidentée et la réparer.

Ces deux buts sont à rattacher comme raffinements de buts plus généraux qui ne se trouvent pas dans notre modèle actuel. Cela révèle une fois de plus la nécessité d'une seconde passe d'analyse.

2. Cahier des charges

Ce chapitre définit de manière complète les buts et les objets utilisé dans la modélisation.

2.1. Définition des buts

2.2. Définition des objets

Conclusion

Au travers de cette première partie de projet, nous avons pu nous rendre compte de l'importance d'une méthode d'analyse rigoureuse et systématique. Les systèmes critiques, tel qu'un logiciel de dispatching d'ambulance, demandent une grande analyse et une bonne connaissance du domaine.

Cette première passe dans les modèles nous a permis d'acquérir une certaine connaissance du domaine. Toutefois, nous nous sommes rendu compte de l'importance de faire un grand nombre d'itérations afin d'avoir un modèle complet et correspondant à la réalité. Actuellement, notre modèle est loin d'être complet mais nous pensons avoir un bon embryon de départ.

L'analyse iter-modèle permet de rapidement se rendre compte des buts, des états, des opérations, des raffinements manquants ou incomplets. Sans cette pluralité des modèles, il est probablement difficile d'aborder aussi facilement et rapidement des problématiques aussi complexes.

Par ailleurs, nous nous sommes également rendu compte des difficultés que les équipes d'analyse peuvent rencontrer telles que : les problèmes de communication sur les définitions, la synchronisation entre les équipes, etc.

Nous aurions aimé avoir le temps de faire une autre itération sur notre modèle ainsi que de formaliser l'ensemble du modèle. Le peu de formalisation que nous avons fait nous a permis de nous rendre compte de ses avantages (et de ses inconvénients). Nottement, la facilité de dérivation et de maintient de la cohérence entre les modèles nous a probablement fort aidé dans la fin de la première partie du projet.

Évaluation d'Objectiver

Objectiver nous a été fort utile durant cette première partie du projet, cependant quelques petites améliorations pourraient rendre son utilisation nettement plus efficace.

- Le format de sauvegarde en .xml met tout le fichier sur une seule ligne, S'il était correctement indenté, des outils tels que svn / git pourraient gérer automatiquement les conflits de versions qui arrivent régulièrement lorsqu'on travaille à 7 sur un même fichier.
- Un "Search and Replace" dans le noms des buts et leurs définitions nous aurait fait gagner un temps précieux.
- La manipulation des graphes pourrait être nettement améliorée, par exemple s'il était possible de déplacer un noeud parent et tous ses enfants en même temps, ou de "minimiser" un noeud et ses enfants.
- Il serait pratique de pouvoir disposer de plusieurs instances d'un même but/objet/obstacle dans les diagrammes, cela permettrait de les arranger plus clairement.
- Une vraie version linux ne serait pas de refus.
- L'export SVG pourrait être grandement amélioré. L'export actuel étant presque inexploitable.
- Il devrait être possible d'exporter des sous-graphes des diagrammes.