THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par :

l'Université Paul Sabatier

Présentée et soutenue par :

Gaëtan Séverac

le 15 novembre 2013

Titre :

Étude et simulation des mécanismes d'interaction pour l'exécution de missions d'exploration planétaire réalisées par un ensemble d'engins hétérogènes

---

ÉCOLE DOCTORALE SYSTÈMES

Unité de recherche :
ONERA/DSCD, Centre de Toulouse
Directeurs de Thèse :
Abdel-Illah Mouaddib et Éric Bensana,

Jury :

Rapporteurs :

Philippe Martinet	-	École Centrale de Nantes
Michel Occello	-	IUT de Valence

Examinateurs :

Sabine Moreno	-	CNES
Marie-Pierre Gleizes	-	IRIT
René Mandiau	-	Université de Valenciennes

Table des matières

Glossaire

CCSDS	Consultative Committee for Space Data Systems
FDIR	Fault Detection Identification and Recovery
FIPA	Foundation for Intelligent Physical Agents
HTN	Hierarchical Task Network
IA	Intelligence Artificielle
IAD	Intelligence Artificielle Distribuée
IHM	Interface Homme Machine
ISIS	In-Situ Interraction Service
LAAS	Laboratoire d'Analyse et d'Architecture des Systèmes du CNRS
MAS	Multi Agent System
OROCOS	Open Robot Control Software
OWL	Web Ontology Language
SMA	Système Multi Agents
SOA	Service-Oriented Architecture
UAV	Unmanned Aerial Vehicle
XML	Extensible Markup Language

Résumé

L'objet de ces travaux est de proposer une manière dont les missions d'exploration planétaire multi-engins pourraient être gérées. Actuellement, beaucoup de travaux portent sur l'accroissement de l'autonomie par rapport au sol des engins envoyés en exploration, mais on reste dans un schéma globalement mono-engin où chaque engin est géré individuellement.

Dans le futur, l'exploration planétaire va évoluer vers des missions automatisées, plus complexes, mettant en jeu des engins hétérogènes comme par exemple des rovers, orbiteurs, balises ou stations fixes, voire même d'autres types d'engins mobiles comme des drones ou des dirigeables. De plus, les missions seront exécutées selon des phases liées à l'arrivée de nouveaux engins ou à la fin de l'activité d'autres. Compte tenu des contraintes de délai de communication, il est difficile d'imaginer que tout le processus de supervision de telles missions puisse être géré depuis la Terre. Le travail effectué concerne les mécanismes d'interactions que l'on pourrait embarquer à bord des engins afin d'améliorer l'autonomie du système global.

Il s'agit en particulier d'étudier comment il est possible de caractériser les compétences d'un engin et comment les engins peuvent s'échanger ces informations, pour mettre en place des stratégies de collaboration afin d'atteindre les objectifs missions.

La thématique globale est celle de la gestion dynamique de systèmes multi-engins.

Chapitre 1

Introduction

L'exploration planétaire robotisée a débuté dans les années 70 avec le Programme Lunokhod, où des rovers soviétiques télé-opérés ont été envoyés sur la Lune, voir illustration de la figure 1.1. Depuis, l'exploration robotique a montré de nombreux atouts. Globalement, ces systèmes robotiques coûtent moins cher, sont moins risqués que des vols habités, et permettent néanmoins de recueillir de précieuses informations scientifiques. Les agences spatiales internationales considèrent donc l'exploration robotique comme primordiale car :

- C'est une première étape, obligatoire et complémentaire, des futures missions habitées.
- Elle permet l'exploration de zones inaccessibles à l'homme car trop éloignées de la terre ou présentant un environnement trop hostile (e.g. Saturne, Titan, Vénus, ...), comme cela est rappelé dans les recommandations du "Space Advisory Group" de la Commission Européenne [40] et dans des rapports concernant le futur des missions de la NASA [55] ou de la Jaxa [61].

Les engins robotisés d'exploration évoluent rapidement. Ils embarquent une puissance de calcul grandissante et le nombre de leurs capteurs, ainsi que leurs précisions augmentent avec l'objectif de les rendre de plus en plus autonomes vis à vis du contrôle sol. Dans un avenir relativement proche, le nombre d'engins autonomes d'exploration planétaire est appelé à croître. Ils pourront ainsi être plusieurs à la surface, ou en orbite, d'un même objet, collaborant pour mener à bien une mission commune. Cette perspective permet d'imaginer des applications scientifiques plus variées, mais elle pose également de nouveaux problèmes d'organisation et de gestion des missions.

Le travail de thèse se place dans un contexte d'étude prospectif concernant les futures missions d'explorations planétaires, telles qu'elles pourront se

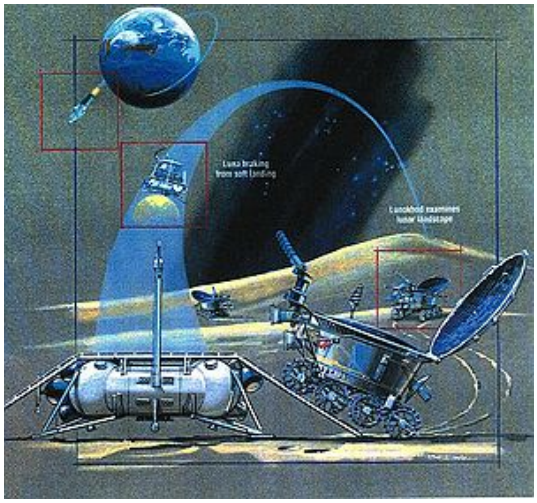

FIGURE 1.1 – Illustration de la mission Lunokhod-1, 1970

dérouler dans les prochaines décennies. L'objet de cette recherche est de faire évoluer la manière dont est géré ce type de missions. Actuellement, beaucoup de travaux portent sur l'accroissement de l'autonomie par rapport au sol, des engins envoyés. Mais ils restent dans un schéma globalement mono-engin où chaque engin est géré individuellement. Il s'agit ici de proposer, d'étudier et de simuler les mécanismes d'interactions possibles, entre plusieurs engins robotisés hétérogènes, dans un contexte de mission réaliste.

Le premier chapitre introduit le contexte global de l'exploration planétaire robotique, ses enjeux et ses intérêts. Puis une étude et une spécification du problème de l'exploration multi-engins sont présentées, où les thématiques des systèmes multi-robots, de la représentation et de l'échange des connaissances est plus particulièrement mises en avant. Dans le deuxième chapitre, est ensuite présenté un état de l'art sur les domaines connexes, à savoir les systèmes multi-agents (SMA), les architectures logicielles basées sur la notion de service, la modélisation , le partage de la connaissance, les ontologies, ainsi que les technologies existantes dédiées à l'interopérabilité dans le domaine spatial. Le chapitre trois décrit une modélisation du système d'exploration multi-engins ainsi que sa représentation dans une ontologie. Le chapitre quatre traite des mécanismes de partage de la connaissance et des interactions associées, ainsi que de l'intégration de ces mécanismes à l'architecture logicielle des engins, ceci via la définition d'un module logiciel dédié. Le chapitre cinq décrit l'environnement d'expérimentation et de simulation mis en place pour pouvoir implémenter et tester les solutions proposées. Puis

le chapitre six présente les tests et les évaluations effectués sur des scénarios d'applications représentatifs, ainsi qu'une discussion de leurs résultats. Le dernier chapitre conclut ce document en faisant un bilan des résultats et des limites de ces travaux et en proposant une ouverture possible sur de nouveaux axes de recherche.

1.1 Contexte

1.1.1 L'exploration planétaire robotisée

Actuellement, l'exploration robotique se développe et les succès passés et actuels des rovers martiens "Spirit" et "Opportunity", développés par la NASA, ont été diffusés [5] depuis 2004.

Moins médiatisée, la mission Cassini-Huygens [36], composée d'une sonde d'exploration (Cassini) et d'un atterrisseur (Huygens), est un autre exemple significatif de mission robotique multi-engins, qui a accru considérablement notre connaissance de Saturne et de Titan.

Au cours de cette mission complexe, la phase de largage de l'atterrisseur et sa descente, ont été réalisé de manière totalement automatique, sans intervention du contrôle sol. Lors de la descente de l'atterrisseur Huygens, la sonde Cassini s'est réorientée vers celui-ci pour établir une communication et stocker les informations qu'il lui envoyait. La Terre n'était alors plus en visibilité. Une fois cette phase terminée, la sonde s'est à nouveau orientée vers la Terre pour transmettre les informations précédemment enregistrées.

Plus récemment la mission de la NASA, Mars Science Laboratory (MSL), avec son rover "Curiosity", de la taille d'une voiture, a réalisé un atterrissage spectaculaire sur Mars (en utilisant des rétrofusées et un treuil) et a fait la une de nombreux journaux. Les équipements scientifiques que transporte le rover fournissent déjà des retours d'informations nombreux et de qualité.

Généralement, les activités des véhicules sont prévues sur Terre, avant d'être compilées en un plan qui est envoyé à bord des véhicules pour être exécuté. L'horizon de planification dépend des capacités embarquées et correspond à quelques heures d'activité ce qui, compte tenu de la faible vitesse des véhicules, conduit à une exploration plutôt limitée [1], voir figure 1.2. Les situations critiques et les premières procédures correspondantes sont répertoriées à bord, pour assurer la sauvegarde de l'engin, le diagnostic de

1. Les robots martiens "Spirit" et "Opportunity" ont parcouru quelques kilomètres seulement au cours des sept années de leur mission.

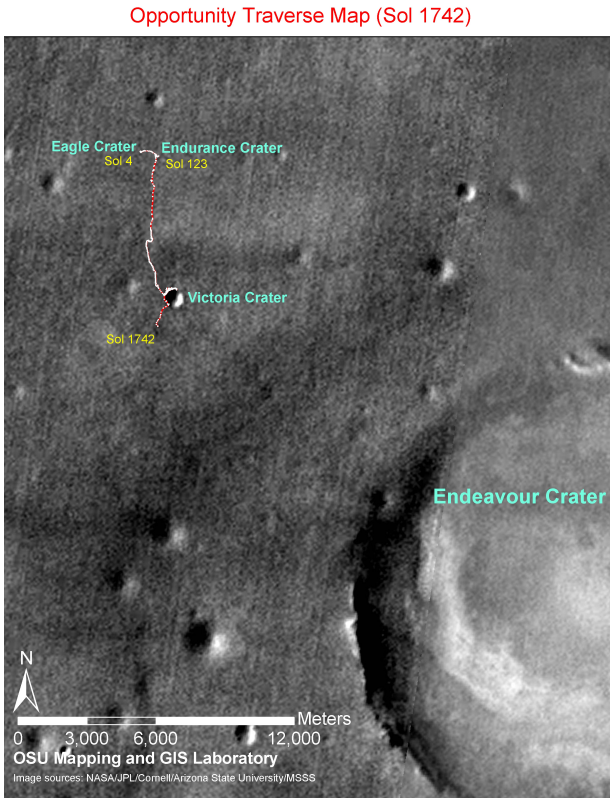

FIGURE 1.2 – En rouge, trajet du Rovor de la NASA "Opportunity" depuis son atterrissage le 4 janvier 2004, jusqu'au 17 décembre 2008. Durant cette période le rover aura parcouru 13,62 Km.

la situation et les solutions de restauration sont eux, élaborés sur la Terre. En cas de mission multi-véhicules la stratégie actuelle est de conserver une supervision directe de chaque engin, le contrôle au sol assurant la coordination.

Exceptés les exemples de la mission Cassini-Huygens et de la coopération entre un orbiteur et les rovers sur Mars, il y a peu de systèmes robotiques d'exploration multi-engins. Et dans le cas ou plusieurs engins sont impliqués dans une même mission, il n'y a pas d'interaction directe et automatique (non planifiée au sol) entre ces engins. Les interactions locales sont essentiellement limitées à du stockage de données et au relais de communications et restent supervisées et contrôlées par le contrôle au sol, voir figure 1.3.

Notre objectif est d'améliorer l'autonomie globale d'un système d'engins d'exploration en limitant les interactions entre le sol et les engins et en autorisant et favorisant les interactions directes entre engins, non supervisées de manière détaillée depuis le sol.

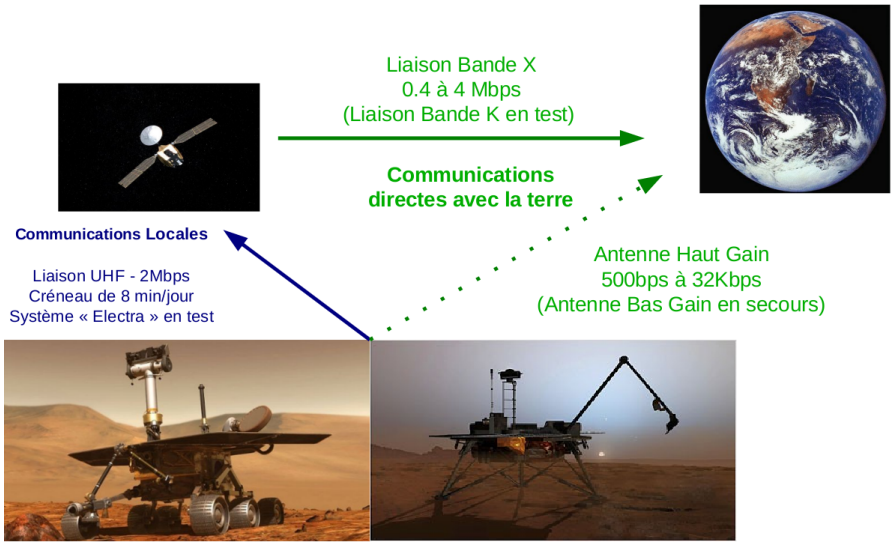

FIGURE 1.3 – Organisation actuelle des communications avec les engins d'exploration de Mars.

1.1.2 Les perspectives technologiques

Plusieurs pistes d'améliorations technologiques existent, en voici trois qui semblent parmi les plus intéressantes pour des applications d'exploration spatiale robotisées :

Les systèmes multi-robots : les recherches sur les systèmes multi-robots dans les applications terrestres, se sont développées très rapidement ces 10 dernières années [66]. Ici l'étude se focalisera sur les systèmes multi-véhicules et leurs applications liées à l'exploration spatiale, mais elles peuvent également être militaires (e.g. Coopération entre robots terrestres, aériens et maritimes[89], équipes de drones et vols en formation [37]), civiles (e.g. mission de secours [94]), ou ludiques (comme dans le cas d'équipes de football [13]). D'une manière générale, l'augmentation du nombre d'engins permet de couvrir plus rapidement des surfaces plus importantes, que ce soit pour réaliser des observations ou des actions. Ces progrès et solutions nouvelles de collaboration, peuvent s'étendre au domaine de l'exploration planétaire.

Indépendamment de l'aspect collaboratif, les engins d'exploration planétaire eux-mêmes, ainsi que les missions, tendent à se diversifier et à se complexifier. Les projets de missions actuelles envisagent des engins de toutes sortes, adaptés à l'environnement qu'ils devront explorer, voir section 1.1.3.

Les robots polymorphiques : ces engins sont capables de modifier automatiquement leur forme, à partir des briques de base qui les composent, pour s'adapter aux différentes situations qu'ils rencontrent. L'article de M. Yim et al. [100] présente plusieurs de ces projets. Les auteurs insistent sur le fait que, même si la technologie actuelle n'est pas encore prête pour des applications fonctionnelles, la capacité d'auto-reconfiguration de ces systèmes les rendra rapidement plus robustes et plus versatiles que des robots conventionnels, comme l'illustre la figure 1.4. Ce type d'engins représenterait donc le futur de la robotique et serait adapté à des missions complexes, en environnement inconnu, comme le sont les missions d'exploration spatiale. Le groupe de recherche sur les concepts avancés de la NASA (NIAC) expliquait dans un rapport de l'an 2000 [23] que les robots polymorphiques seraient utilisés dans l'exploration spatiale en 2010. Même si cette vision était plutôt optimiste, elle montre l'intérêt des agences spatiales pour cette technologie.

La bio-mimétique : elle est source d'inspiration pour les concepteurs de missions d'exploration spatiale. Les êtres vivants actuels sont le fruit de milliards d'années d'évolution et sont donc particulièrement bien adaptés à leur environnement. Le rapport de G. Scott [78] pour l'ESA, présente un engin marcheur destiné à l'exploration de Mars. Il explique que la bio-mimétique est

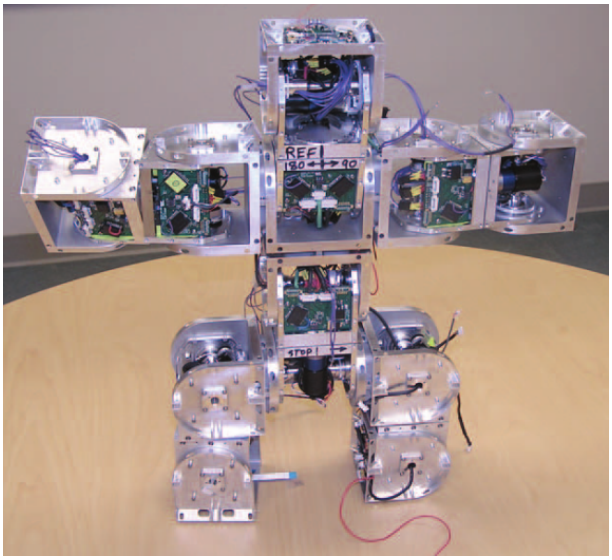

FIGURE 1.4 – Robot polymorphe SuperBot, composé de blocs indépendants, réalisé par l'University of Southern California.

appropriée pour trouver des méthodes de locomotion efficaces en environnement difficile. Dans des articles plus généralistes [82, 75], plusieurs systèmes mimant le monde vivant sont présentés, voir figure 1.5. Les auteurs y expliquent l'intérêt d'architectures physiques bio-inspirées pour l'exploration planétaire, car elles permettent des déplacements sur des terrains accidentés. Ils abordent également les possibilités des architectures distribuées, ou d'intelligence en essaim [2], inspirées de l'organisation des insectes sociaux (e.g. fourmis, termites, guêpes...). Là encore, l'adaptabilité et la robustesse de tels systèmes sont mises en avant.

Globalement, les engins d'exploration planétaire ont donc tendance à se diversifier. L'augmententation du nombre d'engins en service devrait permettre d'enrichir les possibilités d'exploration en autorisant l'accès à des environnements plus difficiles que ceux traditionnellement accessibles aux rovers.

1.1.3 Les missions et leurs enjeux

Les missions robotisées sont une partie essentielle de l'exploration spatiale. Elles peuvent être conçues pour la préparation de futures missions

2. SWARM en anglais

FIGURE 1.5 – Robot marcheur "Scorpion", testé par la NASA sur des terrains très accidentés.

habitées comme celles envisagée sur la Lune ou sur Mars, l'exploration des planètes inaccessibles aux humains en raison de leurs éloignements ou d'environnements trop hostiles (comme Vénus par exemple) ou même l'exploitation des ressources locales (par exemple pour la production de propergol ou de l'eau pour les installations habitées [58]).

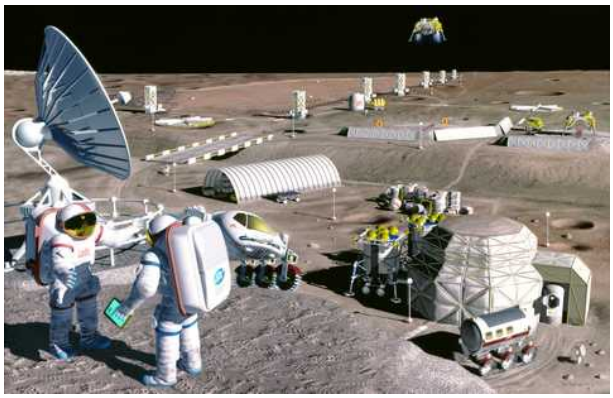

FIGURE 1.6 – Vue d'artiste d'une base lunaire.

Malgré leur complexité évidente, les missions d'exploration multi-véhicules permettraient d'améliorer le retour scientifique en augmentant le nombre et le type de données collectées, en permettant le traitement distribué d'informations, en réalisant des expériences et en combinant les capacités d'équipements adaptés à différents terrains et différentes situations. Par exemple, l'exploration des cratères ou des failles présente un intérêt scientifique majeur pour

la compréhension de la géologie planétaire, mais sont des zones très difficiles d'accès pour des engins roulants.

Il est aussi envisagé des missions impliquant une collaboration homme / robot pour la construction ou la maintenance de bases lunaires par exemple, voir illustration 1.6, où des robots et des "robonautes" [3] assisteront les humains sur des tâches techniques.

Les premières missions seront donc vraisemblablement consacrées à l'exploration, avec l'utilisation d'engins spécialisés, comme par exemple des robots flottants, des petits avions, des hélicoptères, des ballons captifs ou dirigeables, des robots grimpeurs[9].

1.1.4 La collaboration internationale

Améliorer les performances de l'exploration robotique ouvre de nombreuses possibilités et fait l'objet de nombreuses études préliminaires de la part des acteurs majeurs de l'exploration spatiale dans le monde entier : en Europe [40, 90], aux États-Unis [55, 67], au Japon [61], en Russie, en Inde et en Chine.

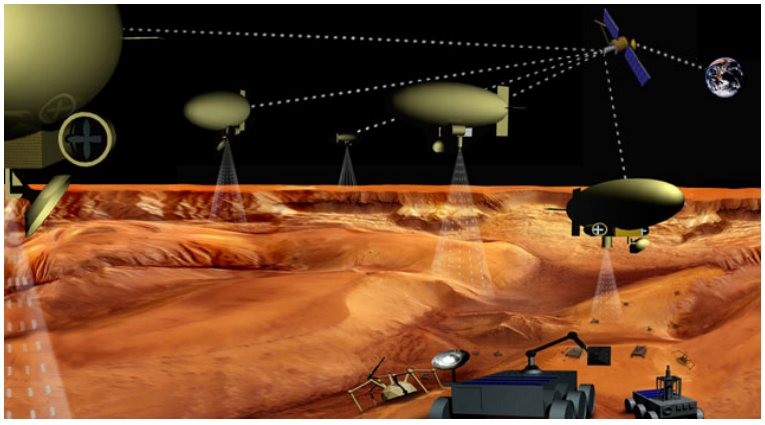

FIGURE 1.7 – Vue d'artiste d'une mission d'exploration multi-engins.

Mais ces nouvelles technologies coûtent cher à mettre en œuvre et le coût global des missions augmente. Pour y faire face, l'établissement de collaborations internationales est de plus en plus fréquent. La notion de concurrence est en train de disparaître pour laisser la place à la coopération. Cela permet de partager les technologies et les moyens pour diminuer les coûts

3. Robonaute - Terme désignant un robot humanoïde, fonctionnant dans l'espace pour assister les astronautes.

et maximiser les retombées scientifiques. Jusqu'à présent, ces collaborations consistent principalement à embarquer des instruments d'origines diverses sur un même véhicule [4] mais les missions du futur intégreront des véhicules venant de différentes origines.

Comme exemple il est possible de citer ExoMars [27], où une collaboration entre l'ESA et la NASA prévoyait l'envoi simultané de deux rovers sur Mars en 2018, mais suite à des restrictions budgétaire, la NASA s'est désengagée du projet. Elle a maintenant été remplacée par l'agence spatiale russe, qui fournira le lanceur et l'atterrisseur pour le rover Européen.

Un autre exemple concerne la Chine et la Russie qui ont récemment lancé ensemble la mission Phobos-Grunt, à destination de satellites de Mars, qui a malheureusement échoué avant de quitter l'orbite terrestre. A l'avenir, une multiplication de ce genre de collaborations et donc du nombre d'engins d'exploration d'origines diverses présents en même temps à la surface d'un objet, est fortement probable [95].

Cela implique également des efforts dans le domaine de la standardisation des interfaces entre les équipements et les systèmes logiciels. Les industriels majeurs du secteur (EADS), parlent déjà depuis quelques années de développer des standards commerciaux dans le domaine spatial [69]. Le Consultative Committee for Space Data Systems (CCSDS) [5], en charge de développer des standards internationaux depuis 1982, est toujours très actif et implique de plus en plus d'acteurs.

Les efforts internationaux de standardisation et d'interopérabilité des systèmes sont présentés plus en détail dans la partie 2.4 de ce document.

1.1.5 Les contraintes spécifiques

Par rapport aux applications multi-robots envisagées sur Terre, les missions spatiales sont soumises à des contraintes spécifiques :

- La conception des engins doit prendre en compte les contraintes liées au lancement et au transfert (masse maximale, volume, vibrations etc.).
- La puissance de calcul embarqué reste très en deçà de ce qui existe sur Terre car il faut recourir à des équipements informatiques spécialisés et certifiés, résistants aux conditions extrêmes des environnements explorés. On assiste néanmoins à une montée en puissance très rapide des performances, surtout de la part de la NASA.

4. Actuellement, chaque machine reste le plus souvent exploitée principalement par l'organisme qui l'a développé.

5. Consultative Committee for Space Data Systems (CCSDS) - www.ccsds.org

– Les communications bord/sol sont limitées, en termes de créneaux d'opportunités, de volume et de délai d'acheminement. Selon les technologies employées et les distances, les vitesses de communication vont de quelques centaines de bits par secondes à quelques mégabits par seconde au mieux. A titre d'exemple une communication Terre / Lune prend quelques secondes, une communication Terre / Mars varie de 6 à 20 min en fonction de la position de Mars par rapport à la Terre et pour atteindre Pluton, le temps de parcours des ondes radio est déjà de plus de 10 heures.
– Inaccessibilité physique à l'engin une fois celui-ci en vol ou sur site, le seul lien étant la liaison radio bord-sol.
– La production d'énergie, qui dépend des conditions opérationnelles dans le cas de capteurs solaires, a une puissance très limitée. Y compris dans le cas de générateur thermoélectrique à radio-isotope, où la production d'électricité est continue, mais ne permet pas d'alimenter tous les équipements en permanence.
– L'environnement dans lequel opèrent les engins est mal connu. Ceci impacte directement les modèles de performance des actionneurs et donc les possibilités de déplacement des engins. L'absence de repère (pas de GPS, peu de cartes) rend les fonctions de localisation et de navigation plus complexes.

1.2 Étude et spécification du problème

Sont présentées ici les hypothèses sur le système d'exploration planétaire multi-engins tel qu'il est considéré dans cette thèse, afin de permettre sa modélisation.

1.2.1 Caractérisation d'un système multi-engins

1.2.1.1 Un réseau dynamique d'engins

Chaque engin est considéré comme une entité au sein d'un système d'exploration distribué. Chaque entité possède un certain degré d'autonomie et est capable d'interagir avec son environnement et les autres entités dans son voisinage (dicté par les possibilités de communication principalement) .

Les engins sont dans un environnement réel et donc dynamique. Des modifications imprévisibles peuvent provenir de différentes origines, externes aux véhicules (conditions météorologiques, obstacle, arrivée / départ d'un véhicule dans le réseau, modification d'une fonctionnalité d'un véhicule voisin...) ou interne à celui-ci (défaillance matérielle, niveau d'alimentation électrique,

utilisation d'une ressource unique...). Ces aspects sont donc aussi liés à la thématique de la détection de défauts, leur identification et la re-configuration (en anglais "Fault Detection Isolation and Recovery" - FDIR) ainsi qu'à sa mise en œuvre dans un environnement distribué.

L'aspect dynamique de ce réseau, dit ouvert au sens de la terminologie des systèmes multi-agents, est très important, il est dû à l'évolution des missions dans le temps et dans l'espace. La structure de ce réseau doit évoluer lors de l'arrivée ou de la mise à disposition d'un nouvel engin, ou lors de son départ de zone, de la fin de sa mise à disposition, ou de la défaillance d'un ou de l'ensemble de ses équipements.

Les zones à explorer peuvent être modifiées au cours de la mission selon les résultats et les décisions des scientifiques, et l'environnement extérieur évolue lui aussi. Des engins (nouvellement arrivés sur place ou bien ayant terminés leur propre mission) peuvent être amenés à collaborer sur des missions en cours, sans que cela n'ait été prévu initialement. Ces situations sont considérées comme des participations opportunistes à des missions pour des raisons de proximité ou pour résoudre un problème imprévu.

1.2.1.2 Les interactions

Dans [29], une interaction apparaît quand un ou plusieurs engins établissent une relation basée sur des actions.

Le rapport de l'OTAN [85] constitue un état de l'art très complet sur les systèmes multi-robots, leurs applications et leur développement. Il permet de définir correctement et efficacement la notion d'équipe d'agents, qui est intéressante à préciser. Celle-ci comporte plusieurs degrés d'interactions possibles entre les agents :

La coordination : qui représente le degré le plus fort, tous les agents sont alors considérés comme des ressources qui œuvrent pour le bien de l'équipe, à la satisfaction d'objectifs communs.

La coopération : où chaque agent a un objectif local (limiter sa consommation d'énergie, survivre, ...) en plus de l'objectif global de l'équipe ; il devra donc toujours faire un compromis entre ces deux objectifs en fonction de priorités préalablement définies.

La collaboration : mode dans lequel la priorité est donnée aux objectifs personnels de l'agent, qui peut, dans certaines conditions, collaborer avec d'autres agents.

Au niveau du contrôle et de la planification, il existe différents modèles d'organisation possibles, entre les engins formant des équipes, plus ou moins hiérarchiques et avec des mécanismes de prise de décision plus ou moins distribués.

L'organisation au sein des équipes d'engins est donc importante, avec la possibilité d'un contrôle centralisé (notion de leader et de hiérarchie) ou bien décentralisé impliquant une prise de décision distribuée. Il est également possible de mettre en place des mécanismes permettant de passer dynamiquement d'une organisation à une autre en fonction de la situation extérieure, des capacités des engins, des objectifs de la mission robotique, ou imposé par le contrôle Sol.

Considérant un système composé d'engins hétérogènes, par leurs conceptions, leurs origines, leurs types ... il faut pouvoir garantir leur interopérabilité pour qu'ils puissent interagir entre eux, coopérer et former un réseau apte à remplir une mission. Les bases pour pouvoir définir l'interopérabilité dans un système multi-robots sont de disposer [3] :

- d'une modélisation universelle de l'environnement.
- d'un langage de description correspondant.

1.2.1.3 Les communications

Comme l'a décrit F. Legras dans sa thèse [48] qui traite de l'organisation dynamique d'équipes d'engins autonomes : la base de tout système multi-robots est la communication entre engins. C'est elle qui permet, en supportant les interactions, de transmettre l'information et, par conséquent, qui autorise les engins à prendre des décisions en fonction de leur niveaux d'autonomie. Il faut donc trouver le bon moyen de communiquer et définir une méthode de diffusion de l'information (utilisation de routeurs simples, protocoles épidémiques...). Les engins de communications doivent également essayer de contrôler la véracité de l'information, de détecter les fautes et de les corriger.

Cette étude, prospective, se placera dans un contexte matériel plausible à 10 ou 20 ans (mutualisation des ressources, capacité d'emport en énergie, poids et volumes plus importants...). Il faudra tenir compte des limitations physiques liées à l'environnement spatial et aux longues distances qui créent des délais et des perturbations sur les communications. Même si les problèmes matériels et ceux liés aux protocoles de bas niveaux, liés aux communications, ne sont pas directement traités. Les protocoles de communication qui

permettent de compenser ces perturbations et de pallier aux problèmes de visibilité pourront être considérés. Comme le Disrupion Tolerant Nestwork (DTN) [43] qui permet aux engins de servir de relais de communications et de stocker les messages en attendant qu'il puissent être émis. Ces thématiques seront détaillées dans le second chapitre.

1.2.1.4 Architecture logicielle embarquée et autonomie

Les engins pourront être très différents entre eux au niveau de leurs algorithmes de contrôle, de leurs capacités de calcul, de leurs fonctions.

Leur autonomie pourra être faible, dans le cas d'une coopération étroite, par exemple un petit robot marcheur, qui peut se déplacer sur un terrain accidenté, aura besoin d'interagir avec un véhicule plus "intelligent" qui pourra calculer pour lui des trajectoires à suivre, afin d'effectuer une tâche d'exploration. Ou bien elle pourra être plus importante pour des engins exécutant des commandes orientées buts, dans le cas de travail en équipe avec plusieurs objectifs et la répartition de tâches diverses.

Chacun d'eux possédera un logiciel embarqué, modulaire et hiérarchique, connecté à des équipements (sources d'énergie, capteurs et actionneurs, antennes...). L'intérêt est ici de définir les blocs fonctionnels qui seront intégrés aux engins (navigation, communication...) et de réfléchir aux types d'interfaces qu'il faudra implémenter pour permettre la collaboration entre engins, que ce soit au niveau de la représentation et du stockage de l'information (en utilisant les ressources informatiques de la charge utile), ou bien au niveau des communications (en utilisant les ressources matérielles de la charge utile).

Les interactions directes entre les véhicules sont supposées être faites par des liens bidirectionnels et des équipements dédiés (récepteur et émetteur). Les communications indirectes pourront être mises en place en passant par un véhicule tiers qui servira de relais de communication. Les interactions avec l'environnement physique incluront toutes les autres formes d'interactions et se feront grâce à des capteurs et des actionneurs intégrés dans la charge utile ou dans la plate-forme du véhicule.

On conservera toujours une interaction possible directe entre chaque engin et le sol.

1.2.2 Modélisation et partage des connaissances embarquées

Il s'agit ici de modéliser l'environnement de la mission ; les connaissances et les possibilités des engins. Le modèle de connaissance devra bien entendu être dynamique et adapté à la structure de décision choisie pour constituer une architecture globale permettant aux robots d'exploration planétaire d'interagir efficacement et de manière autonome. Trois points sont donc à approfondir :

Identification de la connaissance embarquée

Qu'est ce que la connaissance et comment la modéliser ? Il faut identifier et structurer l'information à manipuler. Définir les briques de base de la connaissance (position, image, ...), jusqu'aux concepts de plus hauts niveaux comme les plans de missions des engins voisins, les équipes d'engins ou bien les services proposés par un engin. Il faudra choisir une représentation formelle de la connaissance d'un domaine, sous forme de réseau conceptuel.

Partage de la connaissance

Le but est ici d'étudier la propagation de la connaissance et donc de proposer un protocole d'échange de haut niveau permettant de la maintenir à jour et de la disperser entre tous les engins. Il pourra s'appuyer sur des protocoles de plus bas niveaux actuellement utilisés dans le spatial (TM/TC, PUS) ou envisagés pour le futur (réseau spatial TCP/IP). Mais c'est bien la partie haut niveau qui sera discutée ici. Concernant la confiance entre engins, dans le cadre d'une exploration planétaire, il est possible de considérer qu'il n'y aura pas d'engins malveillants, mais il faudra cependant prendre en compte la véracité des informations échangées. Comment accepter comme "vraie" une information extérieure ? Il peut apparaître des erreurs durant les mesures, le stockage ou la propagation d'une information qu'il faudrait éliminer.

Utilisation de la connaissance dans l'exécution des missions

Cette partie concernera l'utilisation des informations. Des moyens seront mis en œuvre pour modéliser l'information d'une mission, l'implémenter au sein d'un engin, effectuer des actions de commande et contrôle comme par exemple situer un engin, utiliser un service d'un autre engin... L'ensemble du système d'exploration devra être dynamique pour permettre des re-configurations et des évolutions du système.

1.2.3 Cadre applicatif des missions d'exploration types

Pour les missions d'explorations lointaines, le système sera décrit par une composante planétaire et une composante de soutien sur Terre [8]. La composante planétaire sera principalement constituée d'éléments qui répondent aux objectifs scientifiques (collecte de données ou d'échantillons) associés aux éléments présents dans le voisinage qui peuvent fournir un soutien de mission. Ils pourront être de différents types : rovers, avions, orbiteurs, atterrisseurs, balises, capteurs... La composante de soutien sera composée d'entités de contrôle (sur Terre ou dans l'avenir, déportées sur une station en orbite terrestre ou dans des stations de surface sur Mars ou la Lune) et d'équipements de communication (stations au sol, satellites relais, réseaux spatiaux de communication ...). Cette étude s'intéresse principalement aux interactions entre les composants du sous-ensemble planétaire et aux interactions entre ce sous-ensemble planétaire et celui de soutien, sur Terre.

A partir des missions existantes ou envisagées, décrites dans la littérature et d'hypothèses plausibles sur l'évolution attendue des véhicules et des missions d'exploration spatiale, il est possible de définir des scénarios réalistes, impliquant plusieurs engins.

Le contexte global sera l'exploration scientifique d'une planète disposant d'une atmosphère. L'utilisation de différents types d'engins sera possible (rovers de différentes tailles, véhicule aérien du type hélicoptère ou ballon, orbiteur ou sonde spatiale, atterrisseur). Certains d'entre eux pourront être étroitement liés, comme des véhicules captifs exploités par un engin "maître" (exemple de véhicule captif : microRover, ballon...). Un véhicule captif, une fois déployé, reste connecté physiquement à son véhicule "maître" pour le transfert de données ou l'alimentation en énergie. Ce contexte générique, compatible avec Mars ou Titan par exemple, permet d'envisager un large éventail de scénarios, basés autour de l'exploration de zones difficiles, présentant un intérêt scientifique important (failles, cratères ...). L'utilisation coordonnée des différents véhicules sera alors particulièrement appropriée [9]. L'autonomie des engins ainsi que leurs capacités d'interactions pourront être de niveaux différents.

1.3 Objectifs et contribution

Ce travail se focalise sur les problèmes de représentation et de transmission des connaissances, améliorant la coopération entre engins autonomes et les problèmes de contrôle qui en découlent. Il ne traite pas les aspects matériels et les problématiques liées à l'implémentation des fonctionnalités,

comme par exemple les déplacements ou bien l'acquisition de données.

L'objectif principal est donc de développer l'autonomie d'un système d'exploration multi-engins, en augmentant les interactions locales directes, tel que présenté sur la figure 1.8. Ceci en s'inspirant des travaux existants concernant les normes et l'interopérabilité des engins.

Actuellement, la gestion des missions est fondée sur la coordination indirecte, en passant par le contrôle mission sur la Terre.

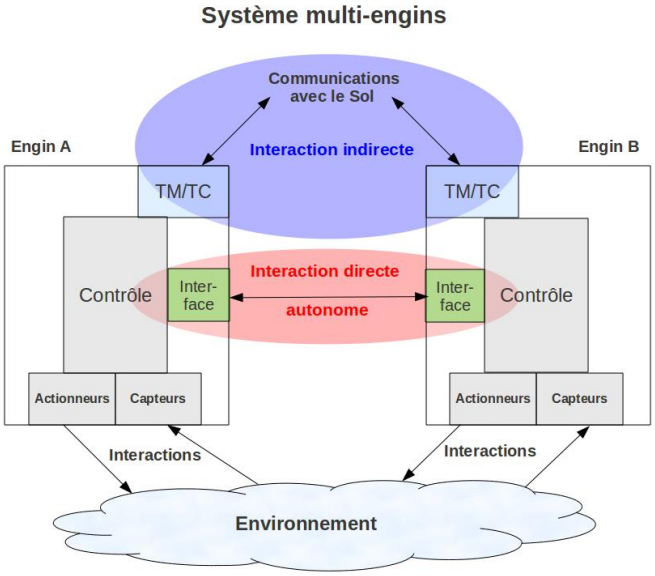

FIGURE 1.8 – Schéma de coopération entre engins d'exploration planétaire. Le but de cette étude est de minimiser les interactions indirectes entre les engins (*zone bleu*) et d'étudier les mécanismes permettant les interactions directes et autonomes (*zone rouge*)

1.3.1 Organisation

Globalement, l'ensemble des engins peut être pensé comme un réseau ouvert et dynamique, où les nœuds sont les engins et les arcs représentent les relations lâches entre les engins comme "EstConnuDe" ou "PeutCommuniquerAvec".

Le but final est de définir un service d'interaction locale "ISIS" (In-Situ Interaction Supervisor) composé du modèle de la connaissance et des fonc-

tions d'interaction de base, qui pourrait être intégré sur tous les véhicules pour soutenir les communications locales, les échanges et la coopération.

Chaque engin devra être capable de maintenir à jour sa description courante (intégrité structurelle, fonctionnalités disponibles ...) en réaction aux événements (demande de contrôle au sol ou demande d'un autre véhicule, panne interne...) et de partager une partie de cette information avec ses différents voisins.

1.3.2 Définir un manuel d'utilisation d'un engin

Chaque véhicule devra embarquer sa représentation des connaissances. Cette connaissance inclut la caractérisation de l'environnement de la mission (conditions météorologiques, carte, position, objectifs de mission...), les capacités propres de chaque véhicule ainsi que celles des véhicules voisins et/ou connus (fonctionnalités intégrées, capacités de déplacement, actionneurs intégrés, équipement, du niveau d'autonomie dans la prise de décisions...). Ces capacités seront couplées avec la notion de disponibilité temporelle des ressources (limitations matérielles, autorisations spécifiques, contraintes temporelles ou météorologiques ...).

La description des services disponibles sur un engin et utilisables par d'autres, s'apparente à un manuel d'utilisation de l'engin axès sur la notion de service.

Le but de ce manuel est donc de modéliser les actions ou capacités possibles d'un véhicule sur son environnement ainsi que les services qu'il peut offrir à d'autres véhicules (relais de communication, ressources informatiques...). En d'autres termes, décrire toutes les fonctionnalités des véhicules, avec leurs contraintes opérationnelles. Le partage de certaines de ces fonctionnalités, pourra permettre à d'autres robots de les exploiter, potentiellement de manière opportuniste, pour accomplir les objectifs de leur propre mission.

Ces différentes capacités d'autonomie et les dépendances entre les engins doivent aussi être discutées et décrites. Ce manuel permettra donc de décrire les services des véhicules, d'un point de vue but et action, pour leur permettre d'échanger leurs capacités d'action et pour soutenir une planification, avec le moins d'intervention du contrôle sol possible. La mise en œuvre proposée devra tenir compte des différents niveaux de collaborations possibles ainsi que des différences de générations possibles entre les véhicules. Ce mécanisme de partage de services entre les engins devra permettre de maintenir à jour la connaissance de manière continue. Car la disponibilité des services représentés va évoluer en permanence, en fonction des conditions de l'environnement, de l'état courant des véhicules et de leurs ressources, etc...

La gestion de ces fonctionnalités d'interactions additionnelles sera effectuée par le service d'interactions locales dénommé ISIS, qui sera connecté à l'architecture logicielle des véhicules et aura en charge la gestion des interactions locales et de la base de connaissances embarquée.

Chapitre 2

État de l'art

2.1 Objectifs

Ce chapitre traitera tout d'abord du domaine des systèmes multi-agents où une part importante des recherches concerne précisément les interactions entre agents. Cela permettra d'aborder les notions associées à ces systèmes comme les différents types d'architectures, les interactions et les communications. Seront ensuite abordées les notions permettant de caractériser l'autonomie d'un système automatisé. Puis, les concepts d'interopérabilité, ainsi que les communications associées, au sein des systèmes multi-engins seront présentés.

Une fois ces notions essentielles vues, il faudra étudier l'évolution des architectures orientées services. Dans ces architectures, qui couvrent les systèmes multi-agents artificiels et les systèmes multi-robots, la notion de service, fondamentale, y représente ce qu'un agent peut faire pour les autres. Toute l'organisation de ces systèmes est basée autour de cette notion de service. Ces points sont essentiels pour les recherches de cette thèse, puisque la notion de service sera la base du modèle de connaissance utilisé pour garantir l'intéropérabilité.

Pour terminer, les ontologies seront étudiées, de leur définition à leurs applications dans la robotique, où elles permettent de représenter la connaissance de manière formelle et non ambiguë. Un élément nécessaire pour permettre les interactions entre engins hétérogènes.

2.2 Panorama des Systèmes Multi-Agents

2.2.1 Historique

L'objectif des recherches sur les Systèmes Multi-Agents (SMA) est d'étudier des sociétés d'agents autonomes qui parviennent ensemble à accomplir des tâches et résoudre des problèmes qu'ils n'auraient pas été capables de traiter seuls, ou en tout cas pas avec la même efficacité. Alors que l'Intelligence Artificielle (IA) classique utilise la métaphore de l'intelligence humaine pour développer ses algorithmes, les SMA utilisent la métaphore sociale, que ce soit celle des insectes dits "sociaux" quand les agents sont réactifs (SMA réactifs) ou des organisations humaines quand les agents sont cognitifs (SMA délibératifs) [56].

La notion de SMA est issue du domaine de l'Intelligence Artificielle et plus particulièrement des études sur l'Intelligence Artificielle Distribuée. Les premières recherches, dans les années 80 portaient principalement sur la planification et sur la résolution d'un problème commun entre les agents. Aujourd'hui, l'objectif est de rendre les agents de plus en plus autonomes, individuellement, malgré l'augmentation de leur hétérogénéité. Il faut donc travailler sur le partage de la connaissance (buts, plans, capacités...) et sur la coordination [18].

2.2.2 Notions génériques

2.2.2.1 Un agent

Dans le domaine multi-agents, Ferber [30] définit un agent comme une entité physique ou artificielle qui :

- est autonome (capable d'action, raisonnement ou réaction),
- est capable d'agir dans un environnement,
- peut communiquer avec d'autres agents,
- est mue par un ensemble de buts, possède des ressources propres,
- est capable de percevoir de manière partielle son environnement,
- ne dispose que d'une représentation partielle de son environnement,
- possède des compétences et offre des services,
- peut éventuellement se reproduire,
- tend à satisfaire ses objectifs en tenant compte de ses ressources, des compétences et des informations dont il dispose.

En robotique un agent dit intelligent se définit par plusieurs critères. Il doit :

– Avoir des objectifs.
– Être capable d'actions.
– Disposer d'un domaine de connaissance.
– Évoluer dans un environnement.
– Être capable d'autonomie et de flexibilité face à des changements d'environnement.

La notion d'environnement ou de contexte d'un agent, peut avoir de nombreuses définitions dans la littérature selon les applications et les domaines de recherches. Par exemple, pour [15], le contexte d'un système multi-agents peut être décrit par rapport à la finalité de l'application, il sert à définir l'environnement et ses attributs nécessaires au fonctionnement des programmes.

Dans le cas présent, le contexte est défini comme l'état courant de l'environnement au sens large, l'environnement étant tout ce qui est extérieur à l'agent (milieu physique, autres agents...). Le contexte est donc représenté par l'ensemble de la connaissance d'un agent, celle-ci pouvant être incomplète, voire fausse, selon les moyens d'information dont dispose l'agent.

2.2.2.2 Un système multi-agents

Dans un système multi-agents, plusieurs agents interagissent. Souvent pour accomplir un but commun, plus complexe que ce qu'ils auraient pu accomplir individuellement. Du fait de ces interactions, chaque agent peut donc influencer les actions et les objectifs d'autres agents [7].

Selon Ferber toujours, on appelle multi-agents un système composé des éléments suivants :

– Un environnement, c'est à dire un espace disposant généralement d'une métrique.
– Un ensemble d'objets : ces objets sont situés, c'est à dire que à tout objet, il est possible à un moment donné d'associer une position dans l'environnement. Ces objets sont passifs, c'est à dire qu'ils peuvent être perçus, créés, détruits et modifiés par des agents.
– Un ensemble d'agents, tels que décrits ci-dessus, représentant les entités actives du système.
– Un ensemble d'actions permettant aux agents d'agir sur l'environnement.
– Un système de communication entre agents.
– Une organisation qui structure l'ensemble en définissant le rôle de chaque agent et potentiellement des équipes.
– Éventuellement une interface qui permet à des opérateurs d'interagir avec le système d'agents.

2.2.3 Les différentes architectures d'agents

Cette section s'appuie sur l'article de B. Chaib-Draa et al. [18].

2.2.3.1 Les architectures délibératives

Ces architectures sont composées des agents délibératifs. Ceux-ci sont issus du monde de l'Intelligence Artificielle. Ils tirent leur nom du fait qu'ils sont capables de délibération et de planification pour atteindre leurs objectifs.

La planification cherche à répondre à la question "Que doit on faire ?", c'est à dire : quelles actions réaliser et dans quel ordre. La planification repose généralement sur trois notions :

- un modèle de description de l'agent, de son environnement et de son but,
- une spécification symbolique des actions que l'agent peut accomplir (pré-conditions / actions / effets),
- un algorithme de planification qui utilise le modèle de l'environnement et la description des actions disponibles pour déterminer un plan d'action permettant d'atteindre le but.

Les algorithmes de planification sont efficaces sur des problèmes limités (monde fermé) mais ils font encore l'objet de nombreuses recherches pour améliorer leur résultats sur des problèmes du monde réel, plus complexes (monde ouvert, espace de recherche beaucoup plus grand, tâches plus complexes à modéliser...).

Inspirées par le raisonnement humain les architectures délibératives prennent en compte des "états mentaux". Elles sont nommées architecture BDI (Belief, Desire, Intention), ce qui correspond aux trois modalités possibles de raisonnement des agents : Les Croyances (ce que l'agent connaît de son environnement), Les Désirs (les états possibles que l'agent souhaite atteindre, qui peuvent être contradictoires), les Intentions (les états pour lequel l'agent a engagé des ressources afin de pouvoir les atteindre).

Après la perception de l'environnement, il y a donc une délibération sur les prises de décisions possibles, via le calcul de plans d'actions par exemple.

Chronologiquement, dans ces architectures, un agent acquiert des croyances sur son environnement, calcule des désirs possibles et met en place des actions pour réaliser ses intentions en fonction des nouvelles perceptions.

A l'inverse des architectures réactives, celles-ci sont moins rapides et moins robustes, mais elles favorisent l'exactitude et l'optimalité des comportements. Les actions, cohérentes, sont guidées par un but et non pas seulement par des stimuli extérieurs.

2.2.3.2 Les architectures réactives

Suite aux limitations du raisonnement symbolique (qui utilise la logique du premier ordre) appliqué aux problèmes complexes du monde réel, une alternative proposée est de développer des agents réactifs, définis par un ensemble de comportements. Les comportements sont des machines à états finis associant à une entrée (perception), une sortie (action) donnée. L'ensemble des interactions de ces agents "simples" crée l'émergence d'un comportement global plus complexe, à la manière des sociétés d'insectes sociaux (fourmis, termites...). Les principales limitations liées à ce type d'agents concernent :

- la difficulté pour les agents d'avoir une connaissance de l'environnement non local,
- l'absence d'apprentissage lié à l'expérience des agents,
- l'anticipation et la preuve du comportement global du système.

Dans les architectures réactives, les décisions sont prises de manière directe, en réaction à une acquisition d'information. Il y a un couplage direct entre l'action et la perception. Les mécanismes de décision sont donc très simples, l'objectif étant la robustesse et la rapidité du système plutôt que la précision et l'exactitude. Il n'y a donc, la plupart du temps, pas de représentation symbolique de l'environnement dans ce type d'architecture.

2.2.3.3 Les architectures hybrides

Comme l'expliquent B. Chaib-Draa et al. [19], les architectures hybrides associent les solutions "classiques" de l'IA, basées sur la planification, à celles des systèmes réactifs. L'objectif est d'exploiter les points forts de chacune.

Les agents y sont composés de plusieurs couches, hiérarchiques. Généralement trois couches suffisent : au plus bas niveau de l'architecture une couche purement réactive prend ses décisions en se basant sur des données brutes en provenance des senseurs. Une couche intermédiaire fait abstraction des données brutes et travaille avec une vision qui se situe au niveau des connaissances de l'environnement. La couche supérieure se charge quant à elle des aspects sociaux de l'environnement, c'est à dire du raisonnement tenant compte des autres agents.

2.2.4 Les interactions entre agents

Cette partie traite de la typologie des différentes interactions entre agents.

Le tableau 2.1, défini dans l'article de J. Ferber [29], propose un classement de plusieurs types d'interactions entre agents suivant différents critères.

Objectifs	Ressources	Capacités	Type de Situation	Catégori
Compatible	Suffisant	Suffisant	Indépendant	Indiffèrent
Compatible	Suffisant	Insuffisant	Collaboration Simple	Indiffèrent
Compatible	Insuffisant	Suffisant	Encombrement	Coopératio
Compatible	Insuffisant	Insuffisant	Coordination	Coopératio
Incompatible	Suffisant	Suffisant	Compétition individuelle	Coopératio
Incompatible	Suffisant	Insuffisant	Compétition Collective	Antagonisn
Incompatible	Insuffisant	Suffisant	Conflits individuels des ressources	Antagonisn
Incompatible	Insuffisant	Insuffisant	Conflits collectifs des ressources	Antagonisn

TABLE 2.1 – Classification des interactions entre agents selon J. Ferber [29]

Selon que les objectifs des agents sont compatibles ou non, que les ressources nécessaires sont en quantités suffisantes ou non et que les agents aient les capacités requises suffisantes ou pas, il distingue différents types de situations et catégories d'interactions.

D'après ces interactions et les architectures présentées dans la section précédente, L. Parker [66] présente trois paradigmes de conception qui peuvent être distingués dans les SMA :

- le paradigme des essaims bio-inspirés (comportements réactifs de nombreux agents "simples" et réactifs),
- le paradigme social, organisationnel (les agents font partie d'une organisation dont on dérive des modèles individuels et la dynamique du groupe),
- le paradigme fondé sur la connaissance, les ontologies, la sémantique (où des agents hétérogènes partagent une connaissance commune).

Le rapport du PEA Action [25] présente une synthèse des interactions présentes dans les SMA, qui peuvent être regroupées en 5 familles de comportements :

- Collectif : Les entités ne sont pas conscientes de la présence des autres agents, mais elles partagent des buts communs et leurs actions contribuent aux actions des autres entités (robotique en essaim, déplacement en formation...).
- Coopératif : Les entités sont conscientes de la présence des autres entités, elles partagent des buts communs et leurs actions contribuent aux actions des autres entités. Plusieurs agents travaillent ensemble pour réaliser un but commun (pousser une boite, explorer une zone...).
- Collaboratif : Les entités sont conscientes de la présence des autres

entités, elles ont des buts individuels et leurs actions contribuent à la satisfaction des buts des autres entités. Par exemple des robots partagent leur capacités de perception afin que chaque robot rejoigne sa propre position objectif.
- Coordination : Les entités sont conscientes de la présence des autres entités, elles n'ont pas de but commun et leurs actions ne contribuent pas à la satisfaction des buts des autres entités. Les robots doivent coordonner leurs actions pour éviter les interférences.
- Compétition : Les entités sont conscientes de la présence des autres entités, elles ont des buts individuels et leurs actions ont un effet négatif sur les actions d'autres entités. Par exemple lors d'une compétition de football où deux équipes s'affrontent.

Cette organisation reprend celle de J. Ferber, dans le cas d'application présent de la robotique d'exploration planétaire, il est considéré que les engins seront collaboratifs par défaut, mais qu'il pourront être ponctuellement coopératifs pour satisfaire des buts communs. Pour permettre ces interactions, ils devront respecter le troisième paradigme de L. Parker concernant le partage d'un connaissance commune.

2.2.5 Les communications entre agents

Les communications sont la base des interactions et de l'organisation dans les SMA. Une interaction entre des agents comprend une action de communication d'un agent (le locuteur) sur un ou plusieurs agents (auditeurs). Les communications peuvent être directes (échange de messages entre agents), mais aussi indirectes, par l'intermédiaire de modifications de l'environnement réalisées par des agents et perçues par d'autres [18].

Dans le cadre d'agents physiquement distribués, des protocoles de communications doivent être mis en place. Il en existe de nombreux, par exemple le protocole "Contract Net" [83]. Dans ce protocole, les agents peuvent avoir un rôle de manager ou de contractant. Les Managers découpent les tâches qu'ils souhaitent réaliser en sous-tâches et diffusent ensuite ces sous-tâches sur le réseau. Les agents qui peuvent exécuter ces sous-tâches font des propositions au Manager qui choisit alors parmi ces propositions les meilleurs agents pour les exécuter, qui deviendront des contractants. Ce protocole permet les négociations et l'allocation de tâches.

Le langage KQML (Knowledge Query and Manipulation Language) est conçu pour supporter la communication inter-agents. Ce langage définit un ensemble de types de messages ainsi que des règles qui définissent les comportements suggérés pour les agents qui reçoivent ces messages. Les types de messages de KQML sont de natures diverses : simples requêtes et asser-

tions (ex. “ask”, “tell”) ; instructions de routage de l’information (“forward” et “broadcast”) ; commandes persistantes (“subscribe”, “monitor”) ; commandes qui permettent aux agents consommateurs de demander à des agents intermédiaires de trouver les agents fournisseurs pertinents (“advertise”, “recommend”, “recruit” and “broker”). Plus récemment le langage ACL (Agent Communication Language), plus riche sémantiquement a été mis en avant par la FIPA [1]. ACL est basé également sur la théorie du langage naturel et a bénéficié grandement des résultats de recherche de KQML [18].

2.2.6 Synthèse

En raison des problématiques présentées dans le chapitre 1, liées au contexte de l’exploration spatiale, le système d’exploration multi-véhicules considéré dans cette thèse devra être le plus autonome possible, tant pour la mobilité que pour la prise de décision, dans la limite des ressources disponibles pour les communications, les déplacements, etc.

Naturellement distribués, les systèmes multi-agents, et en particulier leurs liens avec les systèmes multi-robots [66] seront le point de départ pour nos recherches, où des problèmes complexes seront à résoudre par une équipe d’engins d’exploration et non pas par par un seul engin.

D’une manière générale, les systèmes distribués semblent une solution naturelle pour les missions d’exploration complexes où plusieurs robots simples sont préférables à un robot complexe unique [65, 81]. Cependant, la contrepartie de ces systèmes réside dans leur plus grande complexité opérationnelle.

Les systèmes multi-engins opérant dans des milieux inconnus doivent être robustes et s’adapter aux situations changeantes rencontrées. Cela implique une forte charge de communication qui peut entraîner, en cas de surcharge des moyens de communication, un ralentissement global du système [84]. Il y a un compromis à trouver entre la charge de communication et la robustesse de l’ensemble afin de ne pas finir par apporter plus d’inconvénients que d’avantages au fait de faire travailler les agents en équipe.

2.3 L’autonomie d’un système

2.3.1 Définition

D’une manière générale, l’autonomie représente tout simplement la capacité d’un système à fonctionner seul, sans intervention d’un opérateur humain ou d’un autre système [74]. De ce point de vue un simple automate possède

1. FIPA : Foundation for Intelligent Physical Agents - http ://www.fipa.org

donc déjà une certaine autonomie, même s'il ne prend réellement aucune décision.

Dans la communauté des systèmes multi-agents, les agents sont considérés comme autonomes par définition, car capables d'actions ou de réactions en fonction de leur perception de l'environnement.

Le département de la défense américain a récemment défini l'autonomie, d'une manière plus précise, comme une capacité (ou un ensemble de capacités) permettant à une action particulière d'un système d'être automatique ou bien (dans les limites pré-programmées) indépendante d'un contrôle humain [73]. Mais il précise bien que tous les systèmes autonomes sont supervisés par des opérateurs humains à un certain point. L'autonomie "totale" d'un système artificiel n'existe pas.

2.3.2 Représentation et évaluation

L'ESA et la NASA ont chacune mis en place une grille de niveaux qui leur sont standards pour définir le degré d'autonomie de leurs systèmes spatiaux robotiques [95]. L'échelle de l'ESA est présentée sur la figure 2.1 et celle de la NASA sur la figure 2.2.

Lev.	Description	Functions
E1	Mission execution under ground control; limited onboard capability for safety issues	Real-time control from ground for nominal operations Execution of time-tagged commands for safety issues
E2	Execution of pre-planned, ground-defined, mission operations on-board	Capability to store time-based commands in an on-board scheduler
E3	Execution of adaptive mission operations on-board	Event-based autonomous operations Execution of on-board operations control procedures
E4	Execution of goal-oriented mission operations on-board	Goal-oriented mission re-planning

FIGURE 2.1 – Grille de l'ESA représentant les degrés d'autonomie des systèmes spatiaux robotiques

Level	Observe	Orient	Decide	Act
5	The data is monitored onboard without assistance from ground support	The calculations are performed onboard without assistance from ground support	The decision is made onboard without assistance from ground support	The task is executed onboard without assistance from ground support
4	The majority of the monitoring will be performed onboard with available assistance from ground support	The majority of the calculations will be performed onboard with available assistance from ground support	The decision will be performed onboard with available assistance from ground support	The task is executed onboard with available assistance from ground support
3	The data is monitored both onboard and on the ground.	The calculations are performed both onboard and on the ground.	The decision is made both onboard and on the ground and the final decision is negotiated between them.	The task is executed with both onboard and ground support.
2	The majority of the monitoring will be performed by ground support with available assistance onboard	The majority of the calculations will be performed by ground support with available assistance onboard	The decision will be made by ground support with available assistance onboard	The task is executed by ground support with available assistance onboard
1	The data is monitored on the ground without assistance from onboard	The calculations are performed on the ground without assistance from onboard	The decision is made on the ground without assistance from onboard	The task is executed by ground support without assistance from onboard

FIGURE 2.2 – Echelle de la NASA représentant les degrés d'autonomie des systèmes spatiaux robotiques

Dans le cas présent, le but est de soutenir un système d'exploration multi-engins capable d'exécuter de manière autonome des ordres mission de haut niveau, orientés objectifs. Le système désiré devra donc répondre au minimum aux critères du niveau E4 de l'ESA et à ceux du niveau 4 de la NASA, impliquant le minimum d'interventions externes possible. Ces niveaux sont un objectif pour le système d'exploration vu dans son ensemble, mais celui-ci pourra inclure dans sa composition des agents ayant des niveaux inférieurs.

Il faut cependant faire attention à la notion de "niveau d'autonomie" qui est de plus en plus remise en cause. Il est impossible de classer un système dans un niveau pré-défini car les capacités autonomes d'un système évoluent continuellement durant une mission, en fonction des ressources, des objectifs de mission, des contraintes environnementales, etc.

En effet, le dernier rapport du département de la défense américain sur le sujet prône donc de ne pas suivre la définition de "niveaux d'autonomie" [73]. Les auteurs définissent l'autonomie d'un système comme un continuum depuis le moment où l'humain a directement la main sur système, jusqu'au moment où certaines fonctions sont déléguées à un agent artificiel. L'autonomie est alors une capacité (ou un ensemble de capacités) qui permet à un

système homme-machine d'accomplir une mission donnée. Pour eux la notion d'autonomie d'un système risque de créer des "confrontations" et des malentendus hommes / machines, alors qu'ils pensent plus utile de chercher à développer les interactions hommes / machines. Cela diminuerait le risque de conflit et augmenterait l'efficacité globale du système.

Il recommandent donc d'abandonner le terme "niveau d'autonomie" et de le remplacer par un framework de référence sur les systèmes autonomes qui :

- Préciserait les décisions de conception sur l'attribution explicite des fonctions cognitives et des responsabilités entre l'humain et l'ordinateur pour atteindre des capacités spécifiques,
- Prendrait en compte le fait que ces allocations peuvent varier suivant les phases d'une mission,
- Rendrait visible les opérations de haut niveau du système inhérentes à la conception des capacités autonomes.

Comme un système peut être composé de sous-systèmes (que ce soit des composants matériels, logiciels ou des engins complets), pour pouvoir augmenter l'autonomie d'un système dans sa globalité il faut donc non seulement augmenter l'autonomie de tous ces sous-systèmes mais aussi améliorer les interactions entre ces sous-systèmes et garantir leur bon fonctionnement.

C'est pour cela que des standards de communication et d'interactions apportent une réelle plus -value dans les systèmes complexes et se développent donc, soutenus par des agences internationales (FIPA [2], CCSDS [3]...).

2.4 Interopérabilité des systèmes

Pour la réalisation de missions où plusieurs agences spatiales collaborent, une plus grande normalisation et interopérabilité est nécessaire. Et, à l'avenir, des normes internationales de communication et de représentation de la connaissance devront être utilisées pour permettre une collaboration efficace des différents véhicules lors de telles missions.

Dans cette section, les protocoles de très bas niveaux qui permettent d'utiliser et de communiquer directement avec le matériel d'un engin (e.g. allumer / éteindre un appareil, relever une mesure, actionner un moteur...) ne seront pas considérés. Ceci est trop éloigné de nos problématiques de recherches.

2. Foundation for Intelligent Physical Agents - www.fipa.org
3. Consultative Committee for Space Data Systems - www.ccsds.org

2.4.1 Niveau physique

Le premier niveau d'interopérabilité entre les agents communicants intervient sur la couche matérielle, sur les outils physiques et leur caractéristiques qui permettent d'échanger de l'information à distance. Ce peut être par des signaux électriques dans le cas d'une connexion filaire (e.g. connexion à internet par câble) ou bien par ondes électromagnétiques pour des liaisons sans fil (e.g. Wi-Fi).

Dans le domaine spatial, le **protocole Proximity-1** de la NASA [45] est un exemple récent de standard de communication. Il peut être mis en œuvre entre des sondes, des landers, des rovers ou des satellites en orbite, même s'ils proviennent d'agences spatiales différentes, afin de soutenir les communications locales. Il utilise une charge utile radio appelée "Electra" qui travaille dans la bande des 400 MHz. Il a déjà été implémenté et utilisé avec succès sur les rovers Spirit et Oportunity et sur la sonde MRO, tous trois opérés par la NASA ainsi que sur la sonde Mars Express de l'ESA.

Ce protocole montre que les communications, au niveau physique, entre engins d'origines différentes sont déjà possibles actuellement. L'hypothèse est faite dans cette thèse que les communications physiques entre engins ne seront pas une problématique bloquante dans le futur. Ce travail de thèse portera uniquement sur les protocoles de haut niveau. Cela ne veut pas dire qu'il ne faudra pas tenir compte des contraintes liées à la physique des communications (perte de signal, limitation de la bande passante...).

2.4.2 Niveau protocole

Ce niveau permet l'échange d'information ayant du sens pour les architectures logicielles des agents. Il utilise les équipements de la couche physique pour encapsuler des messages, les envoyer et garantir qu'ils sont délivrés correctement (dans un temps donné, sans altération...). Sur Internet, ce sont par exemple les protocoles TCP ou UDP, pour les plus connus, qui jouent ce rôle.

Dans le domaine du spatial, des équivalents existent comme le **Disruption Tolerant Networking** (DTN) [70], tout récemment testé entre la station spatiale internationale et un laboratoire au sol en Allemagne[43]. Durant cette expérience un robot au sol était contrôlé par un astronaute à bord de la station spatiale. Le DTN permet de garantir l'intégrité du réseau et la bonne distribution des messages, malgré les coupures qui peuvent intervenir au niveau physique (perte de visibilité de la station, retard dans les communications...). Ce protocole stocke les messages à chaque nœud du réseau,

pour pour les réexpédier en cas de pertes ou de coupure de connexion.

2.4.3 Niveau Service

A un plus haut niveau encore, il devient intéressant, pour des raisons d'opérabilité humaine et d'autonomie des systèmes, de représenter et d'échanger directement des informations à un niveau orienté service.

2.4.3.1 Les "Packet Utilisation Services"

La norme de l'ESA "Packet Utilisation Services" (PUS) [54] est utilisée par tous les satellites récents de l'ESA. Ce standard a pour but de décrire de manière standard les services disponibles sur un satellite, pour qu'ils soient utilisables par tous ceux qui respectent le protocole décrit, comme l'illustrent les exemples de la figure 2.3. C'est un environnement opérationnel qui est fourni par les concepteurs des satellites, il complète le standard plus bas niveau des Télécommandes/Télémesures (TM/TC) et permet de standardiser le contrôle et le monitoring.

Les PUS contiennent :

- Les concepts opérationnels (ce qui est requis pour l'utilisation d'un service)
- Le modèle des services (comment l'application embarquée qui gère le service doit réagir à l'exécution d'un service)
- La structure et la sémantique des paquets TM/TC associés

Cette philosophie des PUS correspond à notre approche, bien que de plus bas niveau. Elle intègre la notion de "service" ou de "contrat" réalisé par un engin, sous certaines conditions et en respectant un certain formalisme. Les PUS décrivent tout ce qui régit l'utilisation de ces services. Ils permettent une certaine interopérabilité entre les satellites et le contrôle Sol et l'utilisation des services d'un engin par un autre opérateur que celui qui opére le satellite habituellement.

2.4.3.2 Spacecraft Onboard Interface Services

La spécification "Spacecraft Onboard Interface Services" (SOIS) est issue du "Consultative Committee for Space Data Systems" (CCSDS)[4] [16]. Ce comité est composé de diverses agences spatiales et a pour but de développer des standards autours des méthodes de communication entre engins et entre agences.

4. Consultative Committee for Space Data Systems (CCSDS) - www.ccsds.org

No.	Service Name
1	Telecommand Verification Service
2	Device Command Distribution Service
3	Housekeeping and Diagnostic Data Reporting Service
4	Parameter Statistics Reporting Service
5	Event Reporting Service
6	Memory Management Service
8	Function Management Service
9	Time Management Service
11	On-board Operations Scheduling Service
12	On-board Monitoring Service
13	Large Data Transfer Service
14	Packet Forwarding Control Service
15	On-board Storage and Retrieval Service
17	Test Service
18	On-board Operations Procedure Service
19	Event-Action Service

FIGURE 2.3 – Exemples de service PUS et codes associés. Par exemple, pour utiliser le service "Telecommand Verification Service" il faudra employer le code "1".

Ce document décrit les concepts d'une interface de services embarquée. Il fournit une présentation de l'interface (services fournis et concepts utilisés) et donne quelques recommandations, justifiées, pour l'utilisation de certains services.

L'approche SOIS vise à standardiser les interfaces entre les différents équipements d'un engin spatial. Pour cela, elle spécifie des interfaces de services standards et des protocoles associés. L'intérêt est de permettre l'accès standard aux capteurs, actionneurs et autres fonctions génériques d'un engin, dans le but de pouvoir développer des applications utilisant ces services, indépendamment de la manière dont ils sont réalisés.

Un exemple intéressant est la spécification du "Device Discovery Service" [17]. Ce service permet de détecter si de nouveaux appareils deviennent actifs et changent la configuration matérielle d'un engin spatial. Cela peut arriver quand un appareil redondant est activé suite à une panne par exemple.

Ici encore c'est une approche très intéressante dans son principe, qui pourra être une source d'inspiration pour la détection de services disponibles entre engins autonomes. Mais elle ne concerne que la détection de nouveaux appareils à l'intérieur d'un même engin. Elle est donc très bas niveau, liée à des protocoles de communication et des bus de données spécifiques. Il faudra

l'adapter à l'utilisation de services public de hauts niveau, qui seront partagés entre différents engins.

2.5 Les Architectures Orientées Service (SOA)

2.5.1 Les notions génériques

Les SOA [4] sont développées autour de composants logiciels modulaires qui respectent des protocoles standardisés pour publier, découvrir ou invoquer des services qu'ils offrent les uns aux autres. Ces composants peuvent être de trois types : les fournisseurs de services, les utilisateurs de services et les gestionnaires de services qui centralisent la liste des services disponibles. Cela permet de réaliser des couplages souples entre différents agents logiciels. L'ensemble du système est donc plus flexible et adaptable [3]. Dans le monde du développement logiciel ce type de système est nommé "Service Oriented Computing" [47, 64], son avantage est de permettre des implémentations moins chères et plus rapides, même dans des environnements hétérogènes (plate formes, protocoles de communication, appareils...).

Dans une architecture orientée service, l'unité de base est le message plutôt que l'opération et la description des services est fondamentale. Dans leur rapport, Bloomberg et al. [11] définissent un service avec les éléments suivants :

Une entête
↪ Le nom du service
↪ La version
↪ Le fournisseur du service
↪ Un contrat de responsabilité (Le responsable du service, le décisionnaire, les entités à prévenir de l'exécution ou non du service)
↪ Le type du service

Une description fonctionnelle
↪ Les fonctionnalités du service, ce qu'il réalise
↪ Les opérations du service, comment il le réalise
↪ La méthode d'invocation du service (URL, interface...)

Une description non-fonctionnelle
↪ Les contraintes de sécurité
↪ Les contraintes / limites de qualité du service
↪ Les possibilités et conditions de transaction / collaboration avec d'autres services
↪ Le niveau d'autorisation à avoir pour l'utilisation du service
↪ La description des termes sémantiques utilisés pour décrire le service ou ses interfaces
↪ Le processus d'exécution du service

La définition d'un contrat de service, qui est la description d'un service qui engage un fournisseur de service et un utilisateur de ce service, sera également utilisée. Si l'utilisateur et le contexte courant respectent les conditions et contraintes décrites dans le contrat du service, le fournisseur de service doit réaliser l'objectif décrit dans le contrat.

Une méthode d'implémentation des SOA est décrite à travers le langage "Process Specification Modeling Language" [47]. Orienté service, il est destiné à des systèmes embarqués, comme par exemple les appareils domestiques pour les "maisons intelligentes" où un serveur centralise tous les services offerts par les différents appareils connectés.

Cette approche est très intéressante pour notre étude. Même si elle reste à un niveau d'interactions faible, car les appareils sont plus simples dans leur fonctionnement que des robots d'exploration et parce qu'il n'y a pas de collaboration automatique : tous les échanges possibles étant prévus et pré-programmés.

Il est ici présenté successivement l'utilisation des SOA dans : les "Web Services", puis les "Cyber Physical System" et enfin les "Service-Oriented Multirobot Systems".

2.5.2 Les Web Services

Même s'il est difficile de définir un Web Service, on peut les décrire comme une SOA respectant au moins les deux contraintes suivantes [38] :

- Les interactions sont basées sur des protocoles de communication Internet (HTTP, FTP, SMTP...).
- Les messages échangés (hors données binaires associées) sont au format XML.

Pour représenter les Web Services, le langage le plus utilisé est OWL-S [5] [52].

C'est un langage ontologique utilisé pour le web sémantique [6]. Il permet d'avoir une meilleure autonomie des agents logiciels pour l'utilisation des services car ils peuvent utiliser des mécanismes de découverte de services en fonction des besoins des utilisateurs. Cela se fait en cherchant des correspondances entre les entrées et les sorties définies dans la description des services puis en analysant leurs pré-conditions et leurs effets [46].

Selon la spécification OWL-S, un service est caractérisé par son profil (ce qu'il fait), par ses méthodes d'accès et par son modèle (comment il fonctionne), comme l'illustre la figure 2.4.

Exemple de Web Service Sémantique

Jinhan Kim et al. [46] utilisent OWL-S pour leur système logiciel orienté service. Le cas d'application présenté concerne la recherche de livre dans une bibliothèque, où un service de recherche par mot clé est automatiquement identifié puis configuré pour être utilisé par une interface graphique.

Leur logiciel permet de découvrir et de configurer de nouveaux services automatiquement dans un réseau. Un service y est décrit comme un concept associé à des entrées, sorties, pré-conditions et effets. Les web services sont décrits dans une ontologie des services qui utilise des instances de concepts qui sont eux décrits dans une ontologie qui définit leur domaine (par exemple dans le cas d'application citée : livre, prix, titre...).

5. OWL Web Ontology Language for Services – http ://www.w3.org/Submission/OWL-S

6. Selon le World Wide Web Consortium (W3C), « le Web sémantique fournit un Modèle qui permet aux données d'être partagées et réutilisées entre plusieurs applications, entreprises et groupes d'utilisateurs »

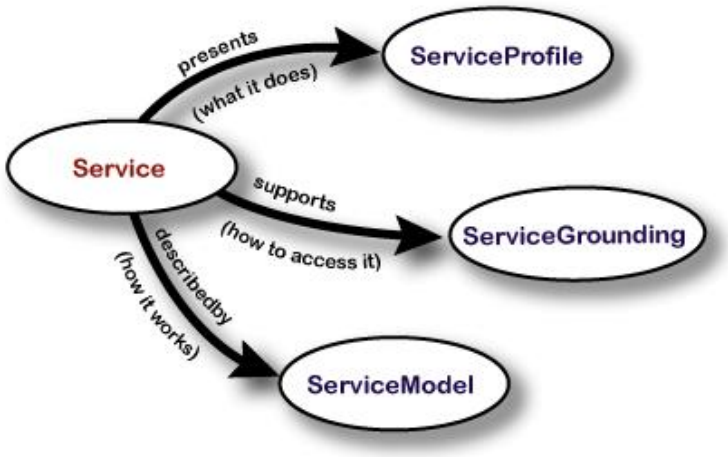

FIGURE 2.4 – Vue de haut niveau de l'ontologie OWL-S de description des services

Un agent responsable de l'ontologie gère cette connaissance en répondant aux requêtes des autres agents. Si un agent demande un service, il va rechercher les concepts correspondant à ce service dans l'ontologie du domaine puis rechercher le service correspondant dans l'ontologie des services. Ensuite il vérifie que les entrées, sorties, pré-conditions et effets correspondent bien à la demande initiale. Si c'est le cas, il retourne le bon service à l'agent qui en avait fait la demande.

Ces travaux illustrent la problématique des web services qui sont adaptés à des architectures purement logicielles, mais qui ne prennent pas en compte les contraintes des agents physiques (position, capacités variables, environnement changeant).

2.5.3 Les Cyber-Physical Systems

Les Cyber-Physical Systems (CPS) émergent de méthodologies et technologies pluridisciplinaires. Ce sont toujours des systèmes distribués, qui intègrent des entités physiques (capteurs, ordinateurs, communications ...), dans lesquels les principales problématiques viennent de l'hétérogénéité des composants et des contraintes des réseaux (délai, vitesse de transmission, perte de données...) [80].

Même si les web services peuvent prendre en compte la notion de contexte, les CPS, du fait des contraintes physiques de certains agents, sont plus précis sur ces problématiques [51].

Un exemple de CPS développé par Jian Huanga [41] concerne un scénario de mission de recherche et sauvetage impliquant des agents logiciels et physiques. L'auteur présente les CPS comme une combinaison d'entités physiques contrôlées par des agents logiciels. Il précise que des particularités sont à traiter quand il s'agit d'agents physiques : ils ne peuvent souvent fournir qu'un service à la fois ; la disponibilité et les résultats des services dépendent de l'environnement et du contexte ; un même service peut être offert par plusieurs agents, identiques ou non, qui ont des états et spécificités particuliers (position, niveau des ressources ...).

Dans cette étude, le modèle des SOA est étendu pour le rendre efficace avec les CPS. Une "Physical-Entity (PE) ontology" est utilisée pour décrire les agents et une PE-SOA pour spécifier leurs services. Le modèle OWL-S est lui aussi étendu pour séparer les pré et post-conditions qui dépendent du contexte de celles qui n'en dépendent pas. La figure 2.5 présente une "PhysicalEntity".

Les "Service Provision Constraints " ne contiennent pas d'information contextuelle. Elles décrivent seulement les aspects statiques du service, qui ne dépendent pas de l'état physique de l'engin, de ses objectifs, de son environnement... Les "Context Preconditions" elles, décrivent la partie dynamique, dépendante du contexte, des pré-conditions liées à l'exécution d'un service.

Un service peut avoir plusieurs effets possibles, mais un seul effet peut se produire après une exécution : pour cela, l'auteur utilise la notion "oneOf".

Pour trouver un service, le système recherche un service de référence dont la description correspond au besoin, puis il indique une entité physique (PE) qui peut exécuter ce service. Pour utiliser un service, un workflow temporaire est créé directement entre les deux agents concernés. Le workflow global est distribué, chaque agent n'en connaît qu'une partie, qui le concerne.

2.5.4 Les Service Oriented Robot Systems

Ces types de systèmes sont basés sur les principes des SOA, mais les agents déployés sont ici des robots.

Par exemple S. Ambroszkiewicz et al. [3] présentent un "Service Oriented Multirobot System" (SOMRS) basé sur une architecture centralisée autour d'un registre des services, voir figure 2.6.

Les auteurs classent les services en trois types :

- Les services physiques, qui ont une action dans le monde physique.
- Les services cognitifs, qui permettent de reconnaître, d'analyser, d'évaluer une scène ou une situation.
- Les services logiciels, qui traitent des données uniquement.

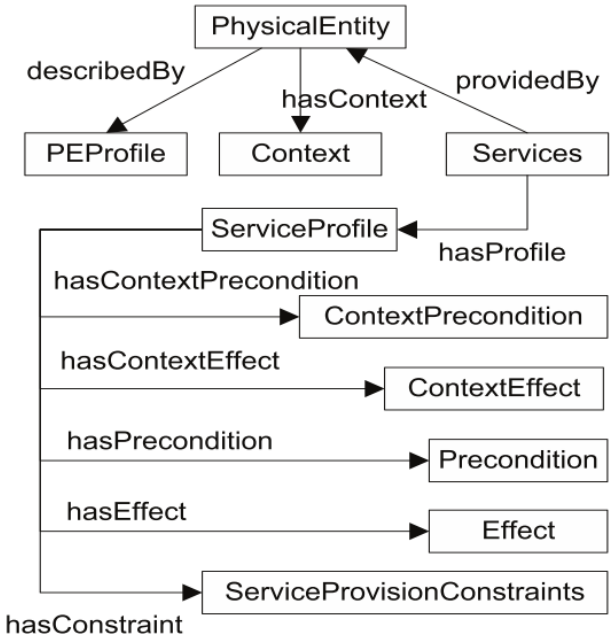

FIGURE 2.5 – Modèle de la PE-SOA

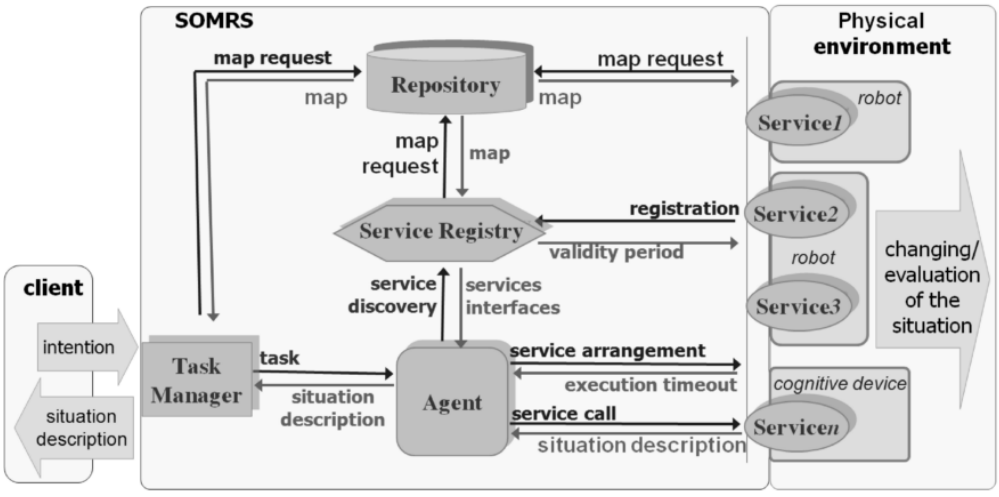

FIGURE 2.6 – Architecture du Service Oriented Multirobot System (SOMRS)

La figure 2.7 décrit la représentation d'un service physique dans cette architecture. On y retrouve 4 champs principaux :

- La description de la situation initiale (pré-conditions).
- La description de la situation finale (post-conditions).
- Le type de l'action réalisée (qui peut être attraper, transporter, pousser un objet...).
- Le périmètre physique dans lequel cette action peut être effectuée.

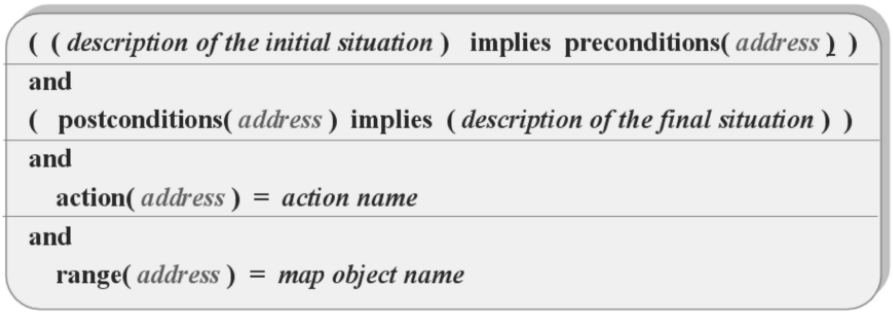

FIGURE 2.7 – Schéma d'un service physique dans SOMRS

Dans ce système, une tâche décrit une situation désirée par un client, voir figure 2.8. Une tâche peut nécessiter l'exécution de plusieurs services pour être réalisée. Un service représente une action (fonction), réalisée par un engin, pour laquelle une interface a été définie dans le langage commun et est publiée (enregistrée) auprès du système. Un service doit utiliser une représentation commune de l'environnement et des protocoles de communications prédéfinis.

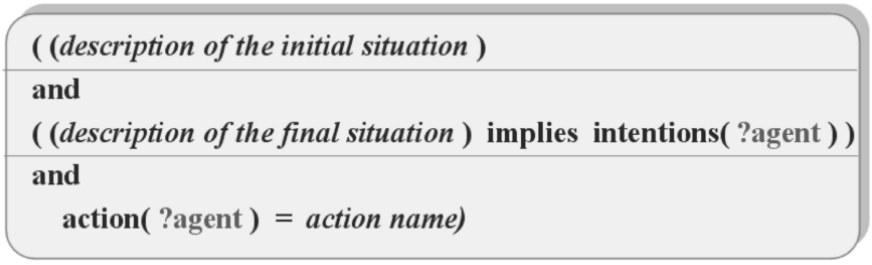

FIGURE 2.8 – Schéma d'une tâche dans SOMRS

Les auteurs discutent des services en général et de l'utilisation des web services pour les robots, mais rappellent que quand on parle de robots ou

d'agents qui peuvent agir sur leur environnement, on est obligé d'associer la connaissance de cet environnement à la description des services.

Ce travail rejoint nos problématiques car il considère plusieurs entités potentiellement physiques, mais il garde une approche de type infrastructure pour l'organisation du système, avec un registre des services qui centralise la connaissance des services. Dans le contexte de cette thèse, une approche purement distribuée des Service Oriented Robot Systems est visée.

2.6 Les ontologies

Pour permettre l'interopérabilité d'engins hétérogènes, il faut que ceux-ci se comprennent et partagent, au moins en partie, une même connaissance. Toutes les applications multi-robots ont donc besoin de représenter la connaissance d'une manière commune et de la partager entre les différents agents pour permettre une coopération efficace [28].

Pour cela, le choix a été fait d'employer les ontologies qui sont souvent utilisées dans les systèmes multi-agents et qui semblent d'un grand intérêt pour la représentation et le partage de la connaissance.

Il aurait été possible d'utiliser d'autres moyens de formalisation comme l'UML ou les bases de données. Mais les ontologies restent plus appropriées car moins génériques et moins orientées programmation.

2.6.1 Concepts génériques

2.6.1.1 Définition

Du point de vue de la philosophie, une ontologie représente la modélisation d'une théorie sur la nature de l'existant [24]. Il s'agit d'une représentation commune d'un domaine spécifique, qui permet à des individus différents de partager des concepts et des relations entre ces concepts [6, 96]. Ces relations syntaxiques n'ont pas de limites et peuvent lier n'importe quel concept à un autre, sans règle ou hiérarchie particulière. Une ontologie n'est donc pas à voir comme un arbre, mais plutôt comme un graphe complexe. C'est pourquoi il est toujours difficile de représenter une ontologie complète par un schéma en deux dimensions. L'utilisation des ontologies peut être décrite comme une méthode déclarative permettant de décrire un système logique.

2.6.1.2 Représentation

Des langages de spécification permettent de décrire les ontologies, ils sont proches de la logique du premier ordre, et représentent donc les connaissances sous forme d'assertion (sujet, prédicat, objet).

Actuellement la plupart des ontologies utilisées en robotique sont écrites en langage OWL (Web Ontology Language) [7], une spécialisation RDF XML, spécifiée par le WorldWide Web Consortium (W3C), qui utilise les fondements de la logique de description (sous ensemble de la logique du premier ordre) dans un monde ouvert.

Apparue dans les années 90, le Knowledge Interchange Format (KIF) [34], est lui aussi utilisé pour représenter les ontologies. C'est un langage pour l'échange de connaissances entre différents systèmes d'agents, basé lui aussi sur la logique des prédicats. Il utilise une syntaxe similaire à celle du LISP.

2.6.1.3 Utilisation

Pour les systèmes multi-agents, les ontologies permettent l'interopérabilité entre les agents et entre différents systèmes dans des environnements hétérogènes, la réutilisation de ces systèmes pour différents types d'applications et leur formalisation lors des développements [91].

En robotique, les ontologies sont utilisées pour spécifier et conceptualiser la connaissance acceptée par une communauté, tout en utilisant une description formelle lisible par une machine et échangeable entre des agents [79]. Dans l'ontologie tous les concepts doivent être associés à une définition, pour lever toute ambiguïté sur ce qu'ils représentent. Cette sémantique commune permet aux agents de coopérer. Elle est aussi un support au raisonnement et à la recherche sur cette connaissance, pour permettre de déduire des informations supplémentaires [77, 20].

Il est cependant important de préciser que chaque ontologie est une opinion préconçue de ses auteurs. Même si elle est conçue dans le but d'être générique, des choix précis doivent être faits pour définir un modèle, ce qui confère à chaque ontologie son unicité. L'ontologie ne garantit donc pas l'inter-compréhension entre agents, mais elle la permet [21].

7. Web Ontology Language spécification - http ://www.w3.org/TR/owl-features/

2.6.2 Applications à la robotique et aux multi-engins

A travers ces cas d'applications, les différents aspects suivants de la modélisation de la connaissance sont traités :

- La représentation de connaissance de l'environnement des engins.
- La représentation de la connaissance des agents eux même, de leur capacités.
- Le formalisme lié à la gestion des opérations

2.6.2.1 Description de l'environnement

Cette section passe en revue des cas d'utilisations des ontologies pour la modélisation de l'environnement des engins.

OpenRobots Ontology

Pour Severin Lemaignan et al. [49] un des enjeux majeurs de la représentation des connaissances dans les interactions homme-robot est le manque de connaissances du "sens commun" des objets (e.g. "la neige est froide", ou comment "ouvrir une porte").

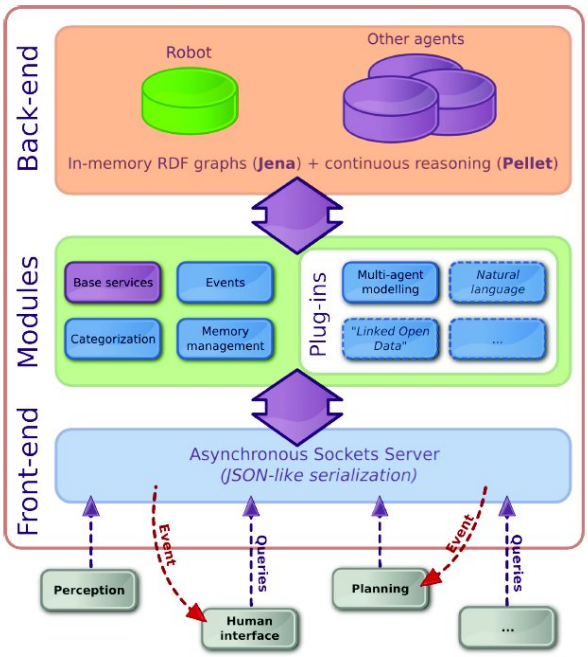

FIGURE 2.9 – Overview of the ORO architecture

Le serveur OpenRobots Ontology (ORO) [8] est un outil de conception pour gérer la connaissance (principalement d'un environnement humain). Il est basé sur OpenCyc [9] qui vise à définir les concepts du sens commun de la manière la plus générale possible et il propose une modélisation commune à toutes les ontologies. Il peut être une bonne base de départ pour créer une nouvelle ontologie de représentation de la connaissance, même si chaque ontologie doit être spécifique au problème qu'elle modélise.

L'architecture ORO permet de transformer des symboles acquis en langage commun (via une base de connaissances, ou des interactions avec un opérateur) en des concepts liés les uns aux autres (e.g. "Le lait est périssable. Le frigo sert à ranger des objets périssables. ⇒ La bouteille de lait doit être rangée dans le frigo."). Cela permet donc de raisonner sur le sens commun des choses.

Sur la figure 2.9 l'architecture ORO est présentée avec ses trois niveaux. En bas, une première couche permet d'interagir avec l'environnement en recevant des requêtes et en exécutant des événements. Le second niveau est constitué de modules qui permettent d'analyser ces requêtes et de générer des événements. Il est en relation directe avec le troisième niveau qui permet le stockage de l'information dans une ontologie et le raisonnement sur cette connaissance.

Grâce à cette architecture le robot doit pouvoir répondre aux questions d'un opérateur humain du type "Est ce qu'il y a de la vaisselle sur la table ?" en demandant si besoin des précisions, sur le type d'objet ("Bouteille" ou "Tasse" ?) recherché, ou sur sa couleur, pour pouvoir répondre plus précisément.

Exemple de concepts et relations représentés dans l'ontologies d'ORO : "Artefact" qui peut être du type "Furniture" qui contient les objets "Table", "Chair", "Shelf". Des relations entre les objets du type relations spatiales "on", "in", "front", "left" ou bien des couleurs existent également.

Le système KNOWROB-MAP et le projet RoboEarth

De la même manière, le système KNOWROB-MAP [87] (Knowledge-Linked Semantic Object Maps), permet de construire, dans une ontologie, un modèle de l'environnement humain et de l'associer à des connaissances encyclopédiques pour définir le type et les propriétés des objets enregistrés. KnowRob permet donc une description riche de la connaissance et utilise le langage Prolog (qui fonctionne sur le principe du monde fermé ; tout ce qui n'est pas dit est considéré comme faux) pour réaliser des inférences. Ces travaux concernent plus la cognition humaine que la coopération multi-robots,

8. OpenRobots Ontology - http ://www.openrobots.org/wiki/oro-server
9. OpenCyc http ://www.opencyc.org/

mais ils donnent un exemple des possibilités des ontologies concernant la modélisation et le raisonnement.

Ce langage KnowRob a été réutilisé dans le projet européen RoboEarth [97, 88] qui utilise les ontologies pour représenter la connaissance de robots hétérogènes. Ce projet a pour objectif de créer une base de données de référence universelle, où les robots pourront partager des informations au sujet de leurs expériences, avec une abstraction de leurs spécificités matérielles.

Le système utilise une architecture centralisée : un serveur accessible depuis le Web enregistre les services et la connaissance de tous les robots pour ensuite la restituer en fonction des requêtes. Toutes ces requêtes passent par une interface de type Web Service. Cette architecture est représentée sur la figure 2.10.

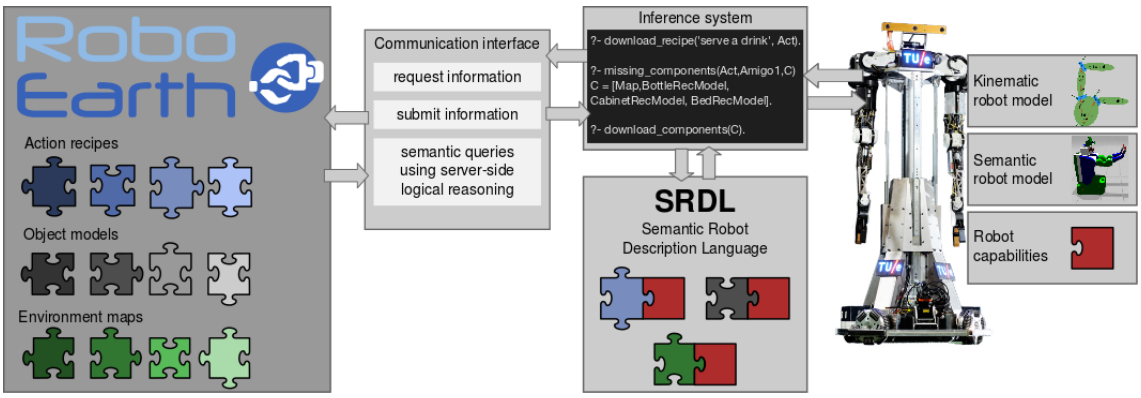

FIGURE 2.10 – RoboEarth architecture

Pour représenter les composants des robots (hardware et composants logiciels), le système utilise le Semantic Robot Description Language (SRDL). Ce langage de description est une extension du langage KnowRob [87, 86] qui est lui même défini par une ontologie utilisant les concepts de base d'OpenCyc.

Sur le serveur la connaissance est organisée en trois grands concepts fondamentaux : les "Action recipes", les "Object models" et les "Environment maps"

Une "Action recipe" est la description d'une action par (voir figure 2.11) :

- la liste des sous-actions qui la compose,
- les contraintes sur leurs enchaînements,
- les dépendances envers les composants nécessaires à l'exécution de la tâche.

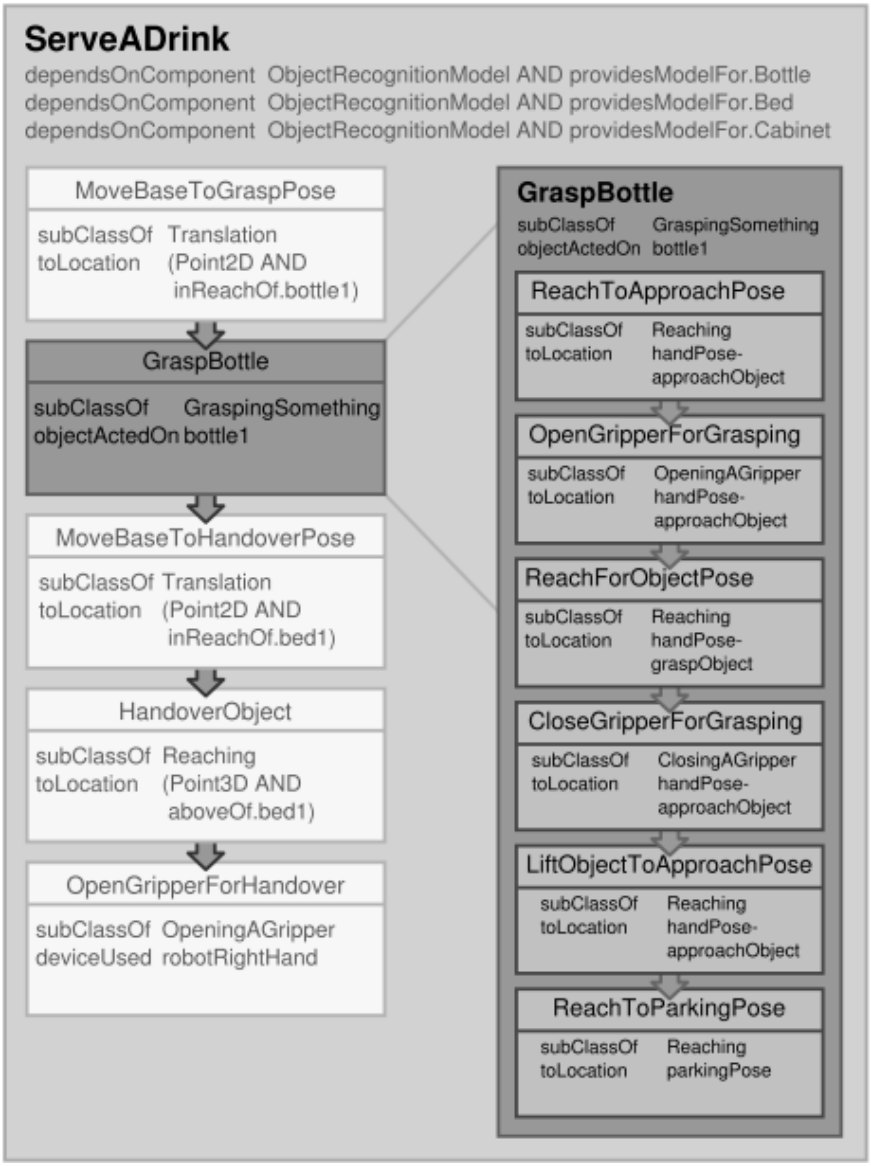

FIGURE 2.11 – L'action "ServeADrink" décrite dans RoboEarth

Chaque sous-action peut être elle-même décrite par une "Action receip". Avant de lancer l'exécution d'une action, une analyse est faite auprès de la base de données de connaissance pour savoir si le robot a toutes les capacités (ressources) requises pour la réaliser. Ces capacités peuvent dépendre des composants disponibles du robot, ainsi que d'autres capacités.

Exemple de description d'action :

```
Class: PuttingSomethingSomewhere
SubClassOf:
        Movement-TranslationEvent
        TransportationEvent

        subAction some PickingUpAnObject
        subAction some CarryingWhileLocomoting
        subAction some PuttingDownAnObject

        orderingConstraints value SubEventOrdering1
        orderingConstraints value SubEventOrdering2
```

Les "Object model" décrivent des classes d'objets, par exemple une armoire dans une chambre d'hôpital. C'est une description sémantique, qui inclut également les informations pour reconnaître l'objet (photo, schéma en 3D...).

Pour connaître leur environnement, les robots utilisent des cartes "Environment maps". Ces cartes d'environnement peuvent être de plusieurs types (topologique ou métrique, 2D ou 3D, correspondant à différents capteurs spécifiques). Les cartes sémantiques permettent de reconnaître et de positionner des instances d'objets, pour réaliser une tâche ou pour mettre à jour la connaissance commune.

Récemment, une mise à jour de la plate-forme du projet a mis en ligne "The RoboEarth Cloud Engine", un serveur partagé, accessible à distance, permettant de déporter les capacités de calculs d'un robot, dans un espace de calcul distant et sécurisé.

Ces recherches sont orientées vers le partage de la connaissance et sur les moyens permettant l'exploitation de cette connaissances par d'autres agents. L'objectif final est de rendre des robots autonomes pour répondre à des requêtes issues d'utilisateurs, dans un environnement humain, en offrant un certain nombre de services.

Cette approche n'est cependant pas orientée vers la collaboration multi-robots, du moins pour l'instant.

Le projet SWEET

L'environnement SWEET ("Semantic Web for Earth and Environmental Terminology") [57, 72] comprend lui une ontologie développée directement par la NASA. Son architecture est présentée dans la figure 2.12. SWEET est décrit comme très modulaire car il contient plus de 6000 concepts séparés en 200 ontologies distinctes, réunies en 8 concepts de hauts niveaux. La figure 2.13 présente quelques uns des sous-concepts. La partie concernant les SIG [10] est décrite dans les deux sous-concepts du concept "Représentation" : "Space" et "Time".

Le temps y est décrit principalement comme une échelle numérique, utilisant la terminologie propre au domaine temporel. Ces marqueurs de temps comprennent : la durée, la saison, la date, etc. Les relations temporelles comprennent : avant, après, etc.

De même ; l'espace est lui une échelle multi-dimensionnelle numérique utilisant la terminologie propre au domaine de la représentation spatiale. Les étendues spatiales comprennent par exemple : le pays, l'Antarctique, l'équateur, entrée, etc. Les relations spatiales comprennent : au-dessus, au Nord de, etc.

Ces systèmes de modélisation ne sont pas le cœur de notre étude, mais ils pourront être intégrés à notre représentation de la connaissance, pour pouvoir manipuler et représenter les notions d'espace et de temps qui serviront à faire le lien entre les services et les contraintes environnementales.

Selon les besoins, la forme la plus adaptée sera choisie pour être ajoutée au modèle.

L'environnement physique et temporel

La représentation du temps et de l'espace sont un besoin fort, traité de longue date, dans tous les systèmes informatiques et en robotique également. Ces notions sont nécessaires pour permettre à des agents de stocker et d'échanger, des informations datées et positionnées. Les ontologies, permettant de mettre en place des définitions communes et non ambiguës, sont naturellement appropriées à ce type de modélisation.

En ce qui concerne la représentation des événements temporels par exemple, les systèmes multi-agents et/ou basés sur la notion de services peuvent déjà implémenter des moyens de représentations temporelles. Comme c'est un besoin générique, F. Pan J.R. Hobbs [63] proposent de créer une sous-ontologie, la plus simple possible, intégrant la plupart des concepts et relations temporelles de base, dont même la plus simple des applications aurait besoin.

10. SIG : Système d'Information Géographique

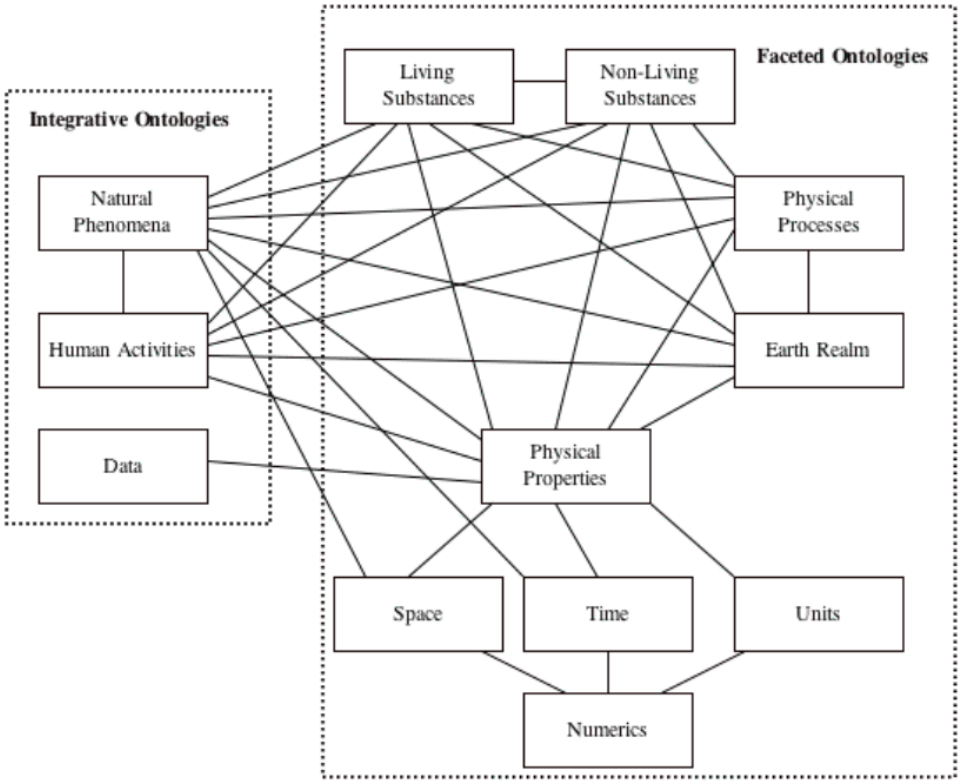

FIGURE 2.12 – Architecture de l'ontologie "SWEET" de la NASA

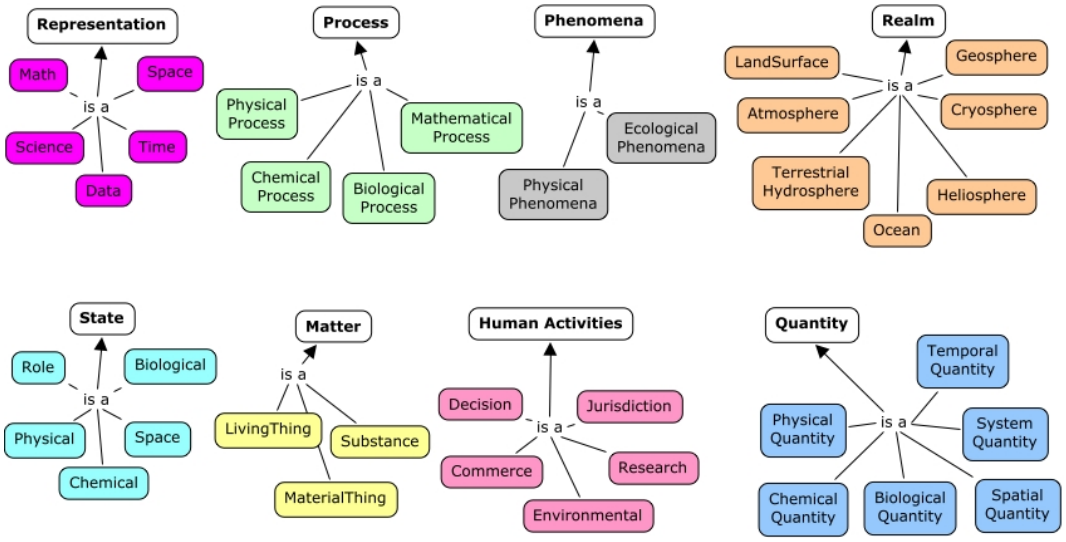

FIGURE 2.13 – Sous-concepts de l'architecture de l'ontologie "SWEET"

Le vocabulaire de cette ontologie permet de représenter des relations temporelles, des intervalles et des événements associés à des information de durée, de date et d'heure. La figure 2.14 représente l'organisation hiérarchique entre ces concepts.

Pour les aspects géographiques de l'information, de nombreux Système d'Informations Géographiques (SIG) existent dans le monde. Ce sont des moyens formalisés d'organiser et de présenter des information géolocalisées, repérées dans l'espace. Par exemple la fondation "Open Source Geospatial" [11] propose un standard de représentation de SIG. Une implémentation spécifique de ce standard pour les capteurs, nommé OpenGIS - Sensor Model Language (SensorML) [12] permet par exemple de représenter toute la géométrie et les caractéristiques temporelles d'un réseau de capteurs ainsi que le traitement de leurs données collectées. C'est un modèle général encodé en XML.

2.6.2.2 Description des engins

Cette section passe en revue des cas d'utilisation des ontologies pour la description des caractéristiques et des capacités des engins.

11. Open Source Geospatial Foundation - http ://www.opengeospatial.org

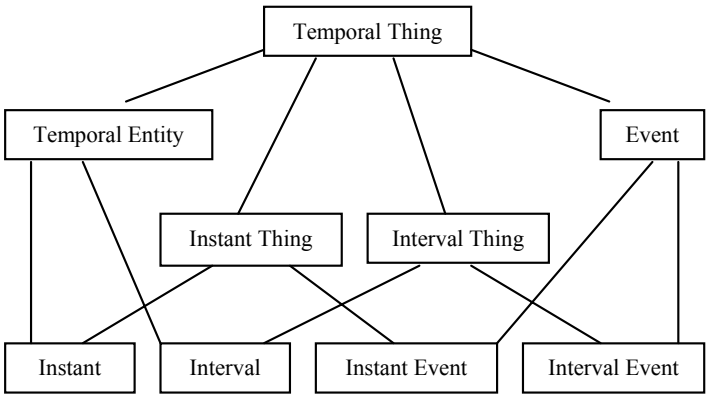

FIGURE 2.14 – Organisation hiérarchique des sous-classes des concepts temporels de l'ontologie de F. Pan J.R. Hobbs [63]

Le projet SensorML

Dans SensorML, comme présenté sur la figure 2.15, la vue d'un système de détection est basée sur la notion de Processus, qui est la superclasse de quatre grandes catégories : Système, Composant, Chaîne de transformation et analyse et Modèle de transformation et analyse. Pour la partie physique, le "Système" représente le niveau composé des appareils (qu'il est possible de comparer à la représentation d'un engin) et le "Composant" celui atomique (qu'il est possible de comparer aux constituants physiques d'un engin). La "Chaîne de transformation et d'analyse" représente le niveau composé des processus logiciels et le "Modèle de transformation et d'analyse" représente le niveau atomique du système. SensorML fournit un framework dans lequel les caractéristiques géométriques, dynamiques, et d'observation des capteurs et systèmes de capteurs peuvent être définies, comme pourraient l'être celles d'un engin ou de l'une de ses capacités.

Le projet A3ME

L'article "Agent-based Middleware approach for Mixed Mode Environments" (A3ME) [39] décrit des appareils mobiles hétérogènes avec une ontologie, pour permettre une communication et une interopérabilité entre eux, ceci indépendamment de la plate-forme matérielle, de l'OS ou du système de communication. Les capacités des appareils y sont présentées comme des services. L'ontologie permet de garantir une représentation et une classification communes, malgré le grand nombre de capacités pouvant exister.

Voici la modélisation utilisée par les auteurs pour décrire les agents :

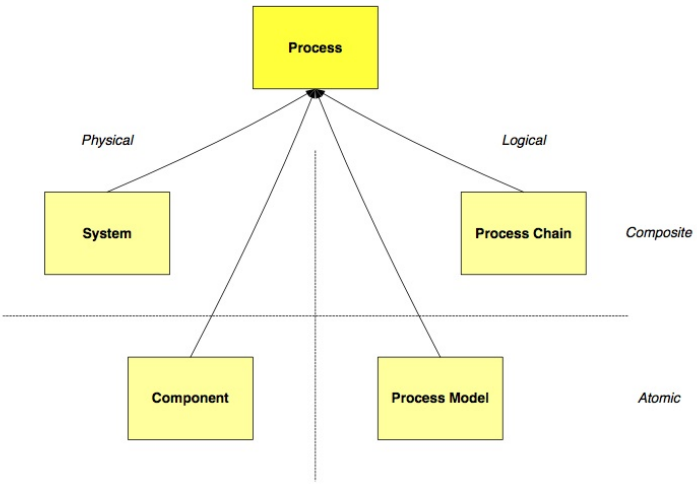

FIGURE 2.15 – Vue de SensorML de systèmes de capteurs

Type d'appareil : tag, télécommande, téléphone mobile, station de travail, véhicule, serveur, ensemble de périphériques, autre ;

Type de capacité : capteur, actionneur, IHM, énergie, communication, CPU, stockage, protocole, communication linguistique, autre ;

Type de service : démarrer, arrêter, invoquer, invoquer périodiquement, fonction, autre ;

Type de capteur : commutateur, température, lumière, humidité, accélération, tension, chimique, positionnement, ultrasons, vision, autre ;

Type d'actionneur : lumière, commutateur, mouvement, manipulateur, outil, autre ;

Type d'IHM : entrée, sortie, entrée/sortie, autre ;

Type d'énergie : non limitée, batterie, renouvelable, passive, autre ;

Les auteurs expliquent rajouter systématiquement le concept "autre" dans chaque catégorie pour permettre une extension possible de l'ontologie dans le futur.

C'est un travail intéressant qui peut être une source d'inspiration pour la partie description des agents. Mais il se concentre sur la description des capacités des appareils mobiles et ne traite pas de l'utilisation de services de plus haut niveau nécessaires à des robots autonomes.

Le projet SWAMO

Le projet de la NASA SWAMO (pour "Sensors Web for Autonomous Mission Operation") [99, 93] décrit lui les concepts et les relations d'un réseau de capteurs, ainsi que la représentation physique des capteurs et de leurs données, en utilisant des standards de représentation comme le formalisme SensorML [12], ou bien - SWEET [57, 72], présenté plus haut dans ce chapitre. Mais elle ne décrit pas comment l'information et la connaissance sont utilisées.

Ce projet et ses ontologies permettent également de définir un standard de représentation et d'échange de l'information entre les agents. Cela permet donc aux développeurs de systèmes utilisant ce formalisme d'implémenter et de garantir leur interopérabilité.

Une missions civiles de recherche et de sauvetage

Des travaux similaires, décrits dans l'article "Test Methods and Knowledge Representation for Urban Search and Rescue Robots" [76] ont été réalisés sur la représentation des capacités d'engins mobiles, dans le contexte de missions de recherches et secours. Les concepts définis se rapprochent de nos centres d'intérêts en ce qui concerne la description des services offerts par des engins. Ils sont présentés dans la figure 2.16, mais ils n'ont pas été complètement implémentés.

Comme pour l'article sur l'architecture "A3ME" l'étude est orientée vers la représentation des capacités uniquement et ne traite pas d'autonomie dans les interactions ni de la représentation de services de haut niveau.

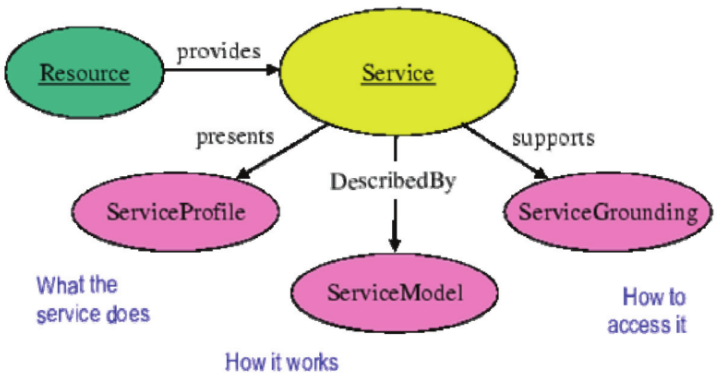

FIGURE 2.16 – Représentation schématique des services dans l'ontologie de Schlenoff et al. [76]

2.6.2.3 Gestion des opérations

Cette section passe en revue des cas d'utilisation des ontologies pour la modélisation et la gestion des missions.

Proteus

Le projet Proteus [53, 50] a pour but de faciliter le transfert de connaissances entre les différentes communautés de robotique de France. Pour cela il utilise les ontologies pour définir de manière non-ambiguë des scénarios (ou "challenges" à relever) et les outils associés permettant de les implémenter en simulation ou sur de vrais robots.

La figure 2.17, représente l'utilisation de l'ontologie dans la spécification des scénarios Proteus. L'ontologie sert ici aux modélisateurs à formaliser un environnement, puis un scénario.

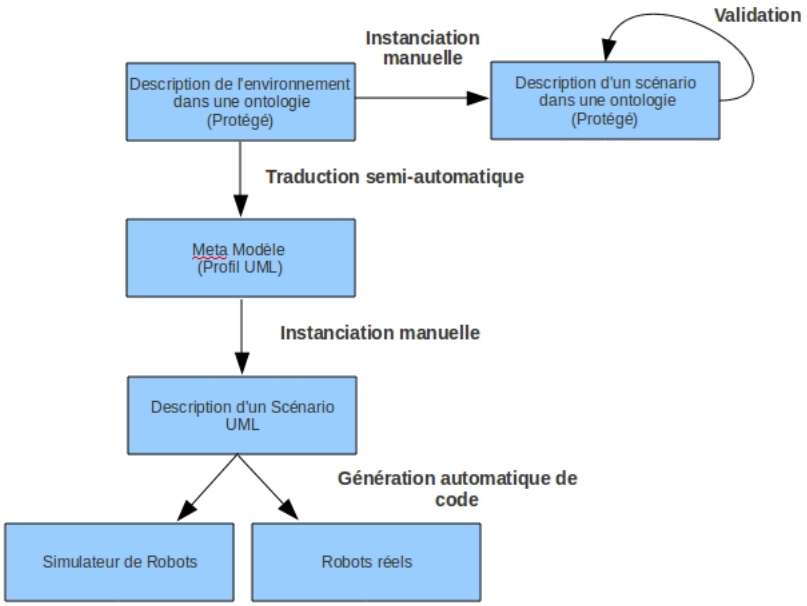

FIGURE 2.17 – Utilisation de l'ontologie dans Proteus

Ce projet n'est pas orienté vers la notion de service et l'échange de la connaissance entre robots. Il a pour but de décrire un environnement et un scénario à un moment donné, de manière formelle, pour que tous les utili-

sateurs soient d'accord sur cette description, ceci grâce à une ontologie qui spécifie les concepts et leurs relations permettant de définir un problème. L'ontologie est ensuite traduite en modèle UML pour permettre la génération de code automatique, qui sera utilisé pour de la simulation ou des applications réelles.

Le Battle Management Language

Le "BML Base Service" (BBS) définit des éléments d'un Battle Modeling Language (BML) comme "Qui", "Quoi", "Quand", "Où" et "Pourquoi" (appelée la règle des 5 W en anglais), ainsi que les informations requises pour la synchronisation et pour la coordination [71]. Ce langage permet d'exprimer de manière non-ambiguë les plans, les ordres, les requêtes et rapports dans le cadre de missions militaires. Pour cela il définit une grammaire formalisée (syntaxe, sémantique et vocabulaire). L'objectif final de ce langage est de spécifier une ontologie de gestion de conflits pour aller vers une interopérabilité conceptuelle de tous les acteurs [10].

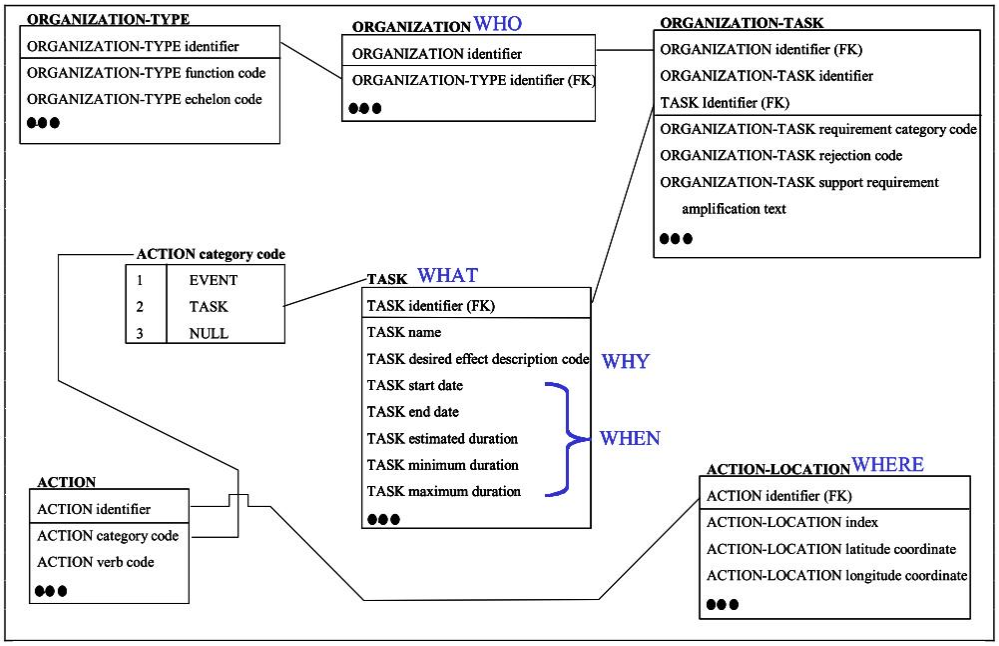

FIGURE 2.18 – Sous-ensemble "Joint Common Data Base " (JCDB), montrant les "5 W" (en français : "Qui", "Quoi", "Quand", "Où" et "Pourquoi") du BML

Dans un premier temps, les efforts de standardisation du BML ont consisté à définir les expressions de bases des mission (plans, ordres, requêtes, rapports). Une phase de conceptualisation s'est ensuite intéressée à formaliser une grammaire complète (syntaxe, sémantique et vocabulaire) permettant d'utiliser les expressions de bases d'une manière non ambiguë, documentée et parsable par des machines. La troisième et dernière phase a consisté à formaliser la sémantique pour améliorer l'interoperabilité, grâce à l'écriture d'une ontologie.

MECA Space

Le projet MECA Space[22] vise quant à lui à améliorer la résilience des équipes hommes/robots d'exploration planétaire, en utilisant les capacités cognitives des astronautes pour aider leurs partenaires robotisés. Les ontologies y fournissent le vocabulaire commun à utiliser dans la spécification des actions (par exemple "Bip", "Move"), de leurs propriétés et des types d'acteurs ("Robot" par exemple). Les auteurs ont intégré aux ontologies la notion d'équipe, d'acteur et des multiples actions qui composent un plan.

Pour spécifier les définitions des actions qui sont dépendantes du contexte, car présentant des niveaux d'abstraction et d'interprétation différents, ils ont introduit la relation *"counts-as"*.

Par exemple, il est possible d'écrire que X (i.e. le plan) *"counts-as"* Y (i.e. l'objectif du plan) dans le contexte C (i.e. les conditions dans lesquelles s'applique le plan), mais cet objectif pourra être différent dans un autre contexte.

L'ontologie KOPER

Pour le PEA Action (ONERA-LAAS), Thibaut Gateau a développé une architecture distribuée pour gérer la supervision d'une plate-forme multi-engins [33], notamment basée sur une planification HTN (Hierarchical Task Network [26]). Pour ses besoins de modélisation il a utilisé une ontologie, nommée KOPER ("KnOwledge base for Planning, Execution and Repair") qui décrit l'environnement des engins ainsi que leurs capacités et leurs variables internes, comme représenté sur la figure 2.19. L'implémentation a ensuite été réalisée en XML.

L'ontologie KOPER a également pour objectif de faire le lien entre le système de supervision multi-véhicules, la plate-forme robotique et la planification. D'une part, une partie des données contenues dans l'ontologie, représentant la connaissance nécessaire à la planification, peuvent être traduites en langage PDDL, directement interprétable par le planificateur. D'autre part le superviseur d'un engin exploite l'ontologie pour savoir comment appeler un service (quel protocole de communication utiliser, quel ordre envoyer...). Pour les développeurs et les utilisateurs, cela permet de définir

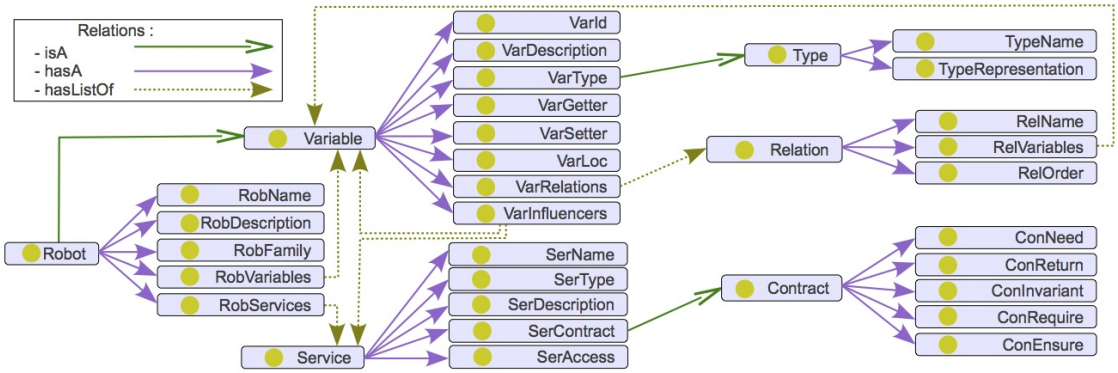

FIGURE 2.19 – Représentation simplifiée de l'ontologie KOPER [33]. Schéma librement adapté du visualiseur de Protégé, Ontograf.

et d'utiliser un formalisme pour la représentation des connaissances sur le système, son état et son environnement.

2.7 Synthèse de l'état de l'art

Cette étude de la littérature existante permet d'avoir une vision plus précise des définitions qui sont importantes pour cette thèse, notamment en ce qui concerne ce qu'est un agent et ce qui caractérise son autonomie. Le second aspect important de cet état de l'art concerne l'étude de l'interopérabilité entre les agents. Il a été présenté que celle-ci est indispensable à toute coopération et qu'elle nécessite que les agents partagent une connaissance commune, ce qui est possible grâce à la définition et à l'utilisation d'une ontologie commune par exemple.

L'environnement des systèmes multi-agents est très vaste et certains de ses aspects, concernant les agents purement logiciels, ne partagent pas les mêmes problématiques que les systèmes d'agents robotiques sur lesquels portent ce travail. A l'avenir, il est donc admis que le terme "engins" qualifiera les agents robotiques d'exploration, afin d'éviter toute confusion.

Suite à ce travail préliminaire, l'étape suivante se focalisera sur la modélisation de cette connaissance commune et sur son implémentation, ainsi que sur la gestion des interactions qui vont permettre à cette connaissance d'être échangée et utilisée par les engins.

Chapitre 3

Modélisation du système multi-engins

3.1 Objectifs

L'objectif global de l'étude est de permettre des interactions autonomes de haut niveau entre les engins robotiques d'un système d'exploration planétaire.

Dans ce chapitre, les thèmes suivant seront abordés :

- La notion de service : le but sera de déterminer comment faire pour décrire l'utilité (ce qu'il peut réaliser ou apporter) d'un engin pour d'autres engins. Que peut-il faire, exécuter ou produire et dans quelles conditions ?

- Les solutions techniques nécessaires à la mise en place et à l'exploitation de ces services à bord des engins (quels sont les modules à développer, les interactions et les protocoles à mettre en place...) ? Cela permettra d'implémenter un manuel d'utilisation des engins robotiques à destination d'autres engins.

- L'utilisation des ontologies comme modèle de représentation de la connaissance pour supporter une méthode de développement et d'utilisation des modules embarqués de gestion de la connaissance, ainsi que pour soutenir l'interopérabilité des engins en opération (mise en place d'un vocabulaire commun aux engins).

3.2 Hypothèses sur les engins et le système multi-engins

3.2.1 Un système dynamique et ouvert

Le système multi-engins d'exploration planétaire peut être modélisé comme un réseau où les nœuds sont des engins.

Ces engins pourront être mobiles ou immobiles, circuler sur le sol, dans les airs, dans l'eau ou sur l'eau. Ils présenteront de nombreuses capacités d'action et de calcul, avec des niveaux d'autonomie différents. Le contrôle par des opérateurs sera déporté mais limité (station en orbite, base installée à la surface de la Lune, contrôle depuis la Terre...). Ces moyens de contrôle seront également considérés comme des nœuds du réseau à part entière, ayant des services particuliers.

Concernant l'utilisation du système dans le temps, il est possible de faire le constat que les engins ont une durée de vie limitée et qu'ils sont envoyés en séquence sur site. Le système sera donc considéré comme ouvert car des engins pourront y être inclus (par exemple dans le cas d'une mission sur plusieurs années où les engins arrivent en séquence [1]) ou en partir (pour effectuer une autre mission ou suite à une panne). Les capacités de chaque engin seront également variables dans le temps, en fonction de l'environnement, des ressources embarquées, de son état interne, de la puissance électrique disponible ...

Dans le cas des missions d'exploration planétaire, où les opérateurs des missions sont peu nombreux et partenaires, il est considéré que tous les engins sont potentiellement coopératifs et bienveillants.

3.2.2 Le besoin d'autonomie

Tout système pour lequel une supervision humaine directe et permanente n'est pas possible, a besoin d'autonomie.

D'un point de vue organisationnel des missions, on supposera que le contrôle au sol peut communiquer directement avec chaque engin (ceci pour garantir un contrôle individuel en cas de besoin). L'objectif du point de vue gestion de mission est que des commandes de haut niveau soient transmises depuis le sol à certains engins seulement. Par exemple, pour lancer une mission d'exploration multi-engins le contrôle enverra un ordre de haut niveau

1. Comme par exemple dans la mission Exo-Mars où deux engins arriveront sur Mars à deux ans d'intervalle

à l'engin qui prendra le rôle de coordinateur de la mission. Celui-ci pourra créer virtuellement une équipe d'engins en invoquant les services nécessaires de ses coéquipiers.

On supposera que les engins pourront avoir des niveaux d'autonomie différents (ils pourront même être uniquement réactifs comme des balises par exemple). L'objectif est d'augmenter l'autonomie globale au niveau du système multi-engin, en exploitant au mieux les possibilités de chaque composante du système. Pour cela, il faut donc que les engins embarquent un module gestion et un équipement de communication qui leur permette de s'échanger leur connaissance sur leurs capacités et de mettre à jour cette connaissance au fur et à mesure qu'elle évolue.

Ce module sera intégré à l'architecture logicielle locale, durant la conception de l'engin et sera ensuite en charge des interactions locales.

3.2.3 La modélisation d'un engin

La description d'un engin sera décomposée en trois sous parties :

- La partie présentation publique

Publique, elle est partagée automatiquement dans les messages de présentation de nouveaux engins qui se rencontrent. Elle comprend l'identifiant unique de l'engin, le type de l'engin (rover, UAV, ballon dirigeable, robot marcheur, orbiteur...), la position de l'engin, la liste de ses certificats d'authentification (qui peuvent être requis pour l'utilisation de certains services) et la liste des services qu'il peut fournir.

- La partie Description Physique de l'engin

Elle est partagée à la demande d'un autre engin uniquement. Elle contient la description de l'encombrement de l'engin (poids, hauteur, longueur, largeur) et la liste de ses composants matériels (bras mécanique, roue, équipement scientifique, capteur, cartes de contrôle, antenne...) avec éventuellement leur description cinématique.

- La partie Informations Privées

La partie privée de la description de l'engin est un modèle de l'état courant de l'engin (équipement, fonction, état, mode, liste objectif mission individuel ou partagé ...). Ces connaissances sont issues de l'architecture locale.

3.3 La notion de Service

3.3.1 Définition

Dans le langage courant nous dirions :

"- Est ce que tu pourrais me rendre un service ? Je ne sais pas comment aller à l'aéroport, peux-tu m'y déposer, j'ai juste une valise ?"

Demander un service signifie que nous avons besoin d'aide ou d'assistance pour satisfaire un besoin sans connaître nécessairement le moyen d'y parvenir. Que ce soit par manque de ressource ou de connaissance sur la manière d'y parvenir, nous nous tourons alors vers d'autres acteurs, dans le voisinage, pour nous aider à atteindre l'objectif.

Dans ce cas présent nous nous positionnons en demandeur (ou "utilisateur") de service et nous recherchons une personne capable de rendre le service correspondant (autrement dit un "fournisseur" de service).

Nous pouvons formuler cette demande de service de la manière suivante :

- Objectif : Aller à l'aéroport
- Contraintes : Y aller tout de suite et emporter une valise.

A partir de cette demande, soit quelqu'un peut directement nous permettre d'atteindre notre objectif : "- Oui bien sûr, je t'y amène.", soit il peut nous aider à le faire, en nous indiquant un autre fournisseur de service : "- Non je ne peux pas, mais demande à Paul, il y va. Tu peux le rejoindre à son bureau" ou bien "- Non je ne peux pas, mais il y a une navette qui passe dans 10 min.".

Dans cet exemple, l'objectif peut être atteint et les conditions sont respectées.

Il est intéressant de noter qu'à aucun moment on n'a besoin de demander : "- Comment est ce que tu peux m'amener à l'aéroport ?".

Notre ami peut avoir une voiture rouge, verte ou un camion, peu importe. Même le fait d'y aller en bus ou avec Paul ne présente pas vraiment de différence tant que les contraintes sont respectées : être à l'heure et emporter une valise.

Par contre il est tout le temps nécessaire de préciser les contraintes d'utilisation du service proposé "Je t'y amène", "Paul est dans son bureau", "la navette passe dans 10 min", etc.

En définitive, exécuter un service, c'est le fait de réaliser un objectif prédéfini, en respectant certaines contraintes. Peu importe la manière dont cela est fait, il n'est pas nécessaire d'avoir connaissance du mode d'exécution.

Pour définir un service, il faut décrire l'objectif que l'on peut atteindre par

l'exécution de ce service, les conditions d'application, les contraintes d'utilisation. Il faut répondre aux questions : Qui, Quoi, Quand, Où (voire Pourquoi) ? de manière similaire à ce qui a été présenté concernant la règle des 5W du "Battle Management Language" (cf. 2.6).

Pour la robotique, un service peut également être vu comme une représentation sémantique d'un objet utile à la planification. Et S. Ambroszkiewicz , voir section 2.5.4, définit les services par la description de l'état initial du monde, l'état souhaité du monde, le type de l'action qui agira sur le monde et le périmètre accessible de cette action.

Un service est donc une unité de travail réalisée par un fournisseur de service pour répondre au besoin final d'un utilisateur de service [38]. Pour résumer, Randall Perrey [68] précise qu'un service doit avant tout pouvoir être décrit, à travers un contrat qui décrit le "Quoi" de ce qui est réalisé et implémenté, mais surtout pas le "Comment" cela est réalisé.

3.3.2 Pourquoi utiliser la notion de service ?

Les "Service Oriented Architectures" (SOA), détaillées dans la partie 2.5, sont principalement représentées par les Web-Services, les Cyber Physical Agents et les systèmes multi-robots. Ces architectures sont segmentées en modules ou engins indépendants, basés autour des services qu'ils offrent. Ce type de système présente les avantages suivants [64] :

L'interopérabilité.
: Du moment qu'un engin respecte le protocole d'utilisation d'un service, il sera en mesure d'interagir avec le système.

La réutilisation du code.
: Un service fonctionnel dans un système peut être plus rapidement adapté à un autre.

L'évolution et la maintenance du système.
: Le code permettant la réalisation d'un service peut être modifié sans remettre en cause ni perturber l'ensemble du système.

En robotique l'utilité des services va plus loin car les robots peuvent directement être définis par les services qu'il fournissent et par le mode d'interface avec les autres engins [14]. Les services et leurs modes d'utilisation permettent de décrire totalement l'utilisation d'un robot. Ils peuvent donc être considérés comme un véritable mode d'emploi.

Mais Yi Wei précise que la capacité pour une machine de donner un sens à ce qu'un service peut réaliser ou fournir, à partir de sa description, reste encore une question ouverte pour les scientifiques [98].

3.3.3 Spécificités de la notion de service dans le cadre applicatif

3.3.3.1 Hypothèses supplémentaires

Dans le cas de missions d'exploration spatiale réalisées par de multiples engins hétérogènes, ceux-ci arrivent sur site de manière successive, sur de longues périodes, et ils ont de longues durées de vie.

Tous les engins n'ont pas été conçus en même temps, pour une même mission. Mais l'objectif est de pouvoir les faire collaborer le jour où ils se retrouveront ensemble sur un même site d'exploration.

Pour permettre cela, une solution est que les engins implémentent tous un module d'interaction locale, basé sur un même modèle de connaissance s'appuyant sur la notion de service, qui leur permettra d'interagir.

Cette connaissance sera mise à jour au fil du temps, lors de l'évolution de l'état et des capacités des engins, lors de changements de situation de l'environnement ou bien lors d'événements particuliers comme l'arrivée ou le départ d'un engin, l'envoi d'ordres de missions de haut niveau par le sol, etc. Le contrôle au sol pourra également directement mettre à jour cette connaissance si besoin.

3.3.3.2 Différences entre Service et Capacité

Dans notre cas un service est avant tout une tâche de haut niveau que peut exécuter un engin pour un autre engin. Cette notion est différente de celle d'une capacité qui représente une compétence de plus bas niveau. Cette compétence peut être nécessaire à la réalisation d'un service, mais le détail de son utilisation ne concerne que l'engin qui exécute le service. Par exemple le service "CatchObject" utiliserait, entre autres, la capacité "MoveHandler" d'un engin.

Dans ce cas une action "GoTo" peut représenter une capacité nécessaire à la réalisation d'un service tel que "ExploreZone". Mais elle peut également être considérée comme un service, par exemple dans le cas où elle permettrait à un engin captif de positionner l'hôte qui le transporte dans le but de démarrer sa mission, ou bien si la nouvelle localisation d'un engin permettait de le mettre en bonne position pour pouvoir servir de relais de communication à un autre engin.

Cette notion de service est donc fortement liée à l'utilisation qui peut en être faite par d'autres engins. Si la tâche réalisée peut être utile à un engin, elle est à considérer comme un service.

Les services sont considérés comme des ressources. L'exécution d'un service entraîne l'exécution d'une tâche donnée ou d'une séquence de tâches par l'engin ou les engins fournisseurs.

3.3.3.3 L'utilisation des services

Dans les scénarios étudiés, les services seront exploités par les différents engins pour mener à bien leurs missions en réalisant les objectifs de haut niveau qui ont été envoyés par les opérateurs au sol.

Ils pourront être utilisés pour réaliser une tâche unique et précise, permettant à un engin de poursuivre sa mission, de débloquer une situation. Ou bien servir à mettre en place un travail coordonné, où plusieurs services de différents engins seront combinés et utilisés simultanément ou dans un ordre déterminé pour mener à bien une mission complexe, irréalisable par un engin unique.

Il faudra donc spécifier un protocole d'utilisation des services qui définira une phase de négociation (initialisation, mise en service), une phase d'exploitation et une phase fin d'utilisation. Ce protocole devra également tenir compte des contraintes sur la disponibilité des services. Il faudra pouvoir prendre en compte le fait qu'un service n'est disponible que sous certaines conditions, ou à un certain moment, et également intégrer le fait que plusieurs engins puissent demander le service à un même moment. Il faudra alors fournir des moyens de réservation ou de gestion des conflits et des priorités.

3.3.3.4 Typologie des services

Selon S. Ambroszkiewicz [3] les robots des systèmes multi-robots peuvent fournir 3 différents types de services :

- Les services physiques, qui modifient physiquement l'environnement.
- Les services cognitifs, qui permettent de reconnaître, d'analyser ou d'évaluer une situation (e.g. détecter la position d'un objet, reconnaissance d'un objet d'après une description sémantique...).
- Les services logiciels de traitement de données.

Dans notre contexte les services logiciels et cognitifs ne représentent pas de différence significative, on n' introduira donc pas de distinction. Seuls, les services physiques, qui agissent sur l'environnement, seront différenciés des services logiciels qui réalisent des fonctions de calculs, du stockage ou du transfert de données, ou bien qui renvoient simplement une information.

D'autre part, la différence sera également faite entre les services dont les résultats auront une influence sur la connaissance générale de l'environnement (e.g. "ExploreZone" ou "LocalizeAgent" qui mettent à jour une carte) de ceux qui ne font que fournir des données qui pourront ensuite être exploitées par l'engin utilisateur du service (e.g. si le service "GetAerialPicture" fournit une image, libre à l'utilisateur de la traiter, de la stocker ou de la transmettre, sans que cela ne change forcément la connaissance générale de l'environnement).

3.3.3.5 Exemples de services embarqués

Cette partie représente une liste des services types rencontrés dans les scénarios d'application des engins d'exploration planétaire. C'est une liste minimaliste qui a servi de base de réflexion, mais qu'il faudra développer et préciser pour des applications réelles.

Les services du type logiciel

MoveTo Se déplacer en un point, une orientation, une orbite, une zone, une altitude... Cas d'échec = Mauvaise position à l'arrivée.

ExploreZone Explorer un endroit donné de la carte. Cas d'échec = Zone impossible à atteindre.

MoveObject Déplacer un objet, un autre engin, modifier la configuration de l'environnement physique... Cas d'échec = Modification impossible, modification différente de celle escomptée.

RelayCommunication Servir de relais de communication à un autre engin. Cas d'échec = Message non transmis. C'est un cas particulier car ce service peut être géré par un protocole de bas niveau du type DTN.

Les services du type physique

StoreData Stockage de donnée par l'architecture locale. Cas d'échec = Mémoire pleine, stockage impossible.

GetData Acquisition de données en provenance d'un capteur (camera, sonde de température, traqueur d'étoile...). Cas d'échec = appareil de mesure indisponible, donnée erronée.

Compute Calcul déporté, on envoie le code et les paramètres, l'engin distant exécute le programme de calcul et renvoi le résultat. Cas d'échec =

calcul impossible, time-out (pas de réponse après un temps donné), erreur de code.

Analyze Analyses déportées, analyse d'image, calcul de trajectoire, de commande, planification... L'engin distant fournit la fonction d'analyse. Cas d'échec = Analyse impossible, time-out, erreur sur les données d'entrée.

ProvideData Transmission d'info / données sur une connaissance spécifique, un état interne, une position courante... Cas d'échec = info non disponible, Autorisation nécessaire.

LocalizeAgent (Aide au) Positionnement d'un engin. Cas d'échec = Localisation impossible.

3.3.4 Réflexions sur la modélisation d'un service et des notions associées

Cette partie est une discussion sur comment modéliser certaines parties spécifiques des services.

3.3.4.1 Identification d'un service

Comment savoir à qui s'adresse une demande de service, et comment identifier un service ?

Ici un service est unique et il appartient à l'engin qui le propose, ils est dit : engin dépendant.

Par exemple : les engins Rover1 et Rover2 peuvent proposer deux services présentant des descriptions totalement identiques (e.g. "MoveTo positionX") cela ne veut pas dire que ces services seront "identiques", car ils ne concernent pas le même engin. Même deux services logiciels identiques ("SaveData") ne vont pas monopoliser les ressources des mêmes engins et sont donc différents.

Pour nommer les services sans ambiguïté, il pourrait donc être intéressant d'utiliser la convention de nommage retenue par l'ESA qui spécifie l'appartenance des fonctions réalisées par des capteurs à travers leur nom. Par exemple le service GoTo proposé par le Rover1 sera appelé "Rover1_GoTo". Il n'y a alors pas de confusion possible sur qui doit exécuter le service. Ce type de convention peut également être retenu pour le nommage et l'identification des variables.

Une autre manière d'établir ces distinctions consiste à toujours associer un service à un engin, tant dans les messages que dans la représentation de la connaissance. C'est la solution qui a été choisie ici. Dans tous nos messages et description de la connaissance, un service est associé à l'identificateur unique de son fournisseur via un attribut.

3.3.4.2 Les entrées d'un service

Les entrées d'un service sont des données de différents types qui servent de paramètre au service.

Il faut faire la différence entre une donnée qui n'est qu'un argument d'un service, et que le service utilise directement pour son exécution, quelle que soit sa valeur, et une donnée dont le service est dépendant pour lancer ou non son exécution, c'est alors un critère qui doit être validé pour que le service soit possible.

Dans le premier cas il s'agira d'entrée du service (Input) qui représente des informations comme une position à atteindre, des données à transmettre...

Dans le second cas il s'agira d'une contrainte (Constraint), par exemple définissant le fait qu'un engin doit être joignable pour exécuter le service, ou bien qu'un niveau minimum d'énergie est requis.

Ces contraintes seront de quatre types :

"INIT" pour celles nécessaires à l'initialisation du service.

"EXEC" pour celles nécessaires durant l'exécution du service.

"END" pour celles nécessaire à l'arrêt du service.

"TIMELINE" pour les contraintes de temps qui pourront être du type : "Daily ; 11H47 :18H13", "Weekly ; Monday, 11H47 :18H13, Tuesday, 12h00 :13H00", "From :xxx" ou "To :xxx".

Il est complexe de représenter les préconditions des services qui dépendent d'une valeur donnée d'une variable dans une ontologie (e.g. Service accessible si "AgentCurrentMode" = "SurfaceOperationMode""), car la syntaxe des ontologie ne dépasse pas le stade des associations binaires (un concept lié à un second par une relation). Ici la modélisation de toutes les informations est gardée dans l'ontologie, sans ajouter de formules logiques. Une solution consiste à utiliser des concepts intermédiaire dans l'ontologie, par exemple "MustBeEqualTo" qui lierait le concept d'une variable à une instance d'une valeur.

Il a été choisi d'utiliser une relation "hasConstraint" qui lie un service à un champ de texte dans lequel sera utilisé un formalisme, choisi arbitrairement, qui reprend la hiérarchie de description de la variable concernée et y associe une valeur donnée. Par exemple il est possible d'écrire : *"ServiceProducer. Knowledge. Map(Service.Input : Zone). Weather = GoodWatherCondition"* pour définir que la variable "Weather" de la "Zone" donnée en entrée d'un service "exploreZone" doit être égale à "GoodWatherCondition" dans la base de connaissance du fournisseur du service.

3.3.4.3 Les sorties et les conséquences d'un service

Un service doit décrire la tâche qu'il exécute pour l'utilisateur de ce service, que ce soit une action sur l'environnement physique (e.g. "goTo"), ou la réalisation d'une fonction (e.g. "computePath", "relayCommunication").

Soit l'engin utilisateur du service connaît à l'avance le service et en a la même représentation que l'engin fournisseur de service, auquel cas un simple identifiant unique suffit à identifier et utiliser un service.

Soit l'engin utilisateur ne connaît pas ce service et il doit "comprendre" et interpréter ce que ce service réalise comme tâche. Il faut alors que le service décrive le résultat de son action sur l'environnement ou la connaissance des engins. Ceci peut être fait par un argument du type : "serviceAchievement" mais cela pose le même problème de la connaissance commune nécessaire à la compréhension de la tâche effectuée. On peut utiliser l'expression des "sorties" du service : qu'est ce qu'il résultera de l'exécution du service ? On ne parle alors plus de "tâche", mais de description des modifications de l'état de la connaissance d'un engin entraînées par l'exécution d'un service. Si les engins fournisseurs et utilisateurs du service ont la même représentation de la connaissance, ils peuvent comprendre et utiliser n'importe quel service. Mais cela signifie que la base de connaissances doit comporter, a priori, tous les concepts permettant de décrire les réalisations de futurs services.

De cette manière, les sorties d'un service peuvent être : des données à usage direct (image, trajectoire), des données décrivant une modification de l'info sur l'environnement (position, type de sol, météo...), un message qui permet de mettre à jour la connaissance ou d'être utilisé directement (accusé de réception dans le cas d'un relais de communication, erreur, problème...). Il est alors possible de modéliser les services par des relations vers ce qu'ils entraînent comme modification de la connaissance.

La difficulté est alors de trouver un moyen de représenter une modification de l'environnement qui pourrait avoir lieu lors de l'exécution d'un service, dans l'ontologie et dans l'implémentation du code embarqué. Par exemple le service 'goTo positionX', prend en entrée une donnée du type "cordonnée" et il a, en sortie attendue, une modification de la connaissance qui devrait donner la position de l'engin exécutant le service à la coordonnée "positionX".

3.3.4.4 Connaissances associées

Notons qu'il est également nécessaire d'associer à la représentation et à l'utilisation des services les notions de connaissance suivantes :

- Condition météo.
- Contraintes temporelles de disponibilité d'un service.
- Type et qualité du terrain / de l'environnement.
- Engin courant avec sa position et ses services.
- Engins voisins, leurs positions et leurs services associés.
- Ordre(s) de mission.

Les services peuvent avoir des contraintes liées à l'état de ces connaissances, ou bien modifier cette connaissance lorsqu'ils sont exécutés. Toutes les informations concernant cette connaissance doivent donc avoir des descripteurs associés, évolutifs, décrivant la qualité de l'information (fraîcheur, fiabilité de la source d'information, précision de l'information).

3.3.4.5 Compatibilité des représentations de l'information

Les situations d'échanges de services (apprentissage d'un nouveau service ou bien mises à jours de la description d'un service suite à son évolution) sont regroupées en trois catégories décrites ci-dessous. Dans celles-ci est appelé "engin E" l'engin qui émet la description d'un service à destination d'un "engin R" qui reçoit cette description et doit mettre à jour sa base de connaissance :

Service échangé et concepts associés connus
: L'engin R connaît déjà le concept de ce service, il doit seulement mettre à jour sa connaissance en intégrant le fait que l'engin E le propose selon les paramètres de disponibilité spécifiques, envoyés dans la description du service.

Service échangé inconnu mais concepts associés connus
: L'engin R ne connaît pas le service, il doit créer un nouveau service dans sa base de connaissance. Il est capable d'interpréter les dépendances et les actions de ce nouveau service grâce à la description de celui-ci qui ne contient que des concepts qu'il connaît.

Service échangé inconnu et concepts associés inconnus
: L'engin R ne connaît pas le service reçu, il doit créer un nouveau service associé à l'engin E, mais il ne peut pas interpréter les dépendances et/ou les actions de ce nouveau service car la description contient des concepts qu'il ne connaît pas. Par exemple un nouveau concept pourrait être une carte représentant l'environnement en 3D alors que l'engin R n'appréhende qu'une représentation 2D de son environnement, ou bien une mesure d'un nouveau capteur renvoyant un type de donnée inconnue.

Cette situation de découverte de service est plus complexe et ne sera pas traitée. Elle nécessitera une mise à jour plus profonde de la connaissance de l'engin E, avec sûrement des contraintes sur ses limitations matérielles (espace mémoire, puissance de calcul).

Note : Si le concept inconnu ne concerne que les dépendances du service, l'engin R pourra également se contenter de demander à l'engin E si le service est disponible, quand il en aura besoin, et l'utiliser le cas échéant. Sans pouvoir planifier sur la disponibilité de ce service, liée aux concepts qu'il ne comprend pas.

3.3.5 Description d'un service

3.3.5.1 Modélisation générale d'un service

Tous type de service est défini par au moins :

- L'action qu'il permet de réaliser.
- Les conditions et contraintes liées à son utilisation.
- Les protocoles d'utilisation.

Cela peut paraître simple, mais en présence de robots qui peuvent agir sur leur environnement ou en subir les conditions changeantes, il est obligatoire d'associer la connaissance de cet environnement à la description des services, les deux notions étant étroitement liées [3]. La représentation de la connaissance commune à tous les engins doit englober leurs services ainsi que l'environnement dans lequel ils évoluent.

Aux éléments précédemment listés, seront rajoutés :

- Une distinction sur les conditions et contraintes : il faudra séparer celles liées à la mise en place du service (initialisation), de celles nécessaires pendant son exécution.
- Une précision sur les contraintes du service qui sont dépendantes de l'environnement, dans lequel évolue l'engin, de celles qui ne le sont pas [42]. Il est également possible de parler des effets de bord et de leurs conditions d'occurrence.

Au cours des réflexions, d'autres éléments sont également apparus utiles à rajouter :

- Le mode d'utilisation du service, qui peut avoir deux états, actif ou passif. Pour un service actif l'utilisateur interroge régulièrement le fournisseur de service pour avoir un retour sur l'exécution. Pour un service passif l'utilisateur invoque le service et attend un retour du fournisseur quand celui-ci a terminé.
- La visibilité des services, qui définit la notion de service privé ou public, ainsi que le partage de services restreints à certains utilisateurs. Un service peut donc être : visible pour les voisins directs uniquement, transmis indirectement de voisin à voisin jusqu'à un certain niveau, ou bien visible et transmis à tous les engins du réseau.
- La différenciation des parties de la description du service qui sont destinées à un usage interne de celles qui devront être transmises aux autres engins.
- La différenciation des parties d'un service selon qu'elles sont nécessaires aux phases d'initialisation, d'exploitation ou de fin du service.

Au final, un service est donc décrit en utilisant les champs suivants :

Nom et identifiant du service et du fournisseur
Réalisation du service
Effets de Bords
    ↪ Tâches inconditionnelles
    ↪ Tâches conditionnelles (avec leurs conditions associées)
Contraintes
    ↪ Contraintes d'initialisation
    ↪ Contraintes d'exploitation
    ↪ Contraintes de fin de service
Utilisation du service (liée à l'état du service (Init, exec, end...))
    ↪ Fournitures en entrées
    ↪ Fournitures en sortie
État courant du service (disponibilité et mode de fonctionnement)
Visibilité du service

Les différentes valeurs de ces champs couplées à l'état interne d'un engin et à l'état de son environnement permettront de déterminer la disponibilité d'un service à un instant donné.

3.3.5.2 Décomposition du service en 3 parties

Pour minimiser les échanges d'information, la représentation des services a été structurée en trois parties : la définition publique du service, le mode d'emploi utilisateur et la partie réservée au fournisseur du service :

1 - La définition du service est l'information nécessaire à un engin robotique pour décider si le service peut lui être utile. Cette partie contient :

- L'identification du service et de son fournisseur comme par exemple "MonitoringZoneBlimp" pour un service à bord d'un dirigeable donnant des images de la zone qu'il survole.
- Le type du service : physique (utilise des actionneurs, avec un effet physique sur l'environnement) ou logiciel (informatique, stockage de données, relais de communication).
- La réalisation du service, qui décrit ce que le service est supposé faire, et le type de retour qui sera obtenu à l'exécution, comme par exemple : donner périodiquement des images d'une partie de son environnement.
- Le mode du service, actif ou passif, qui définit le type des interactions qui sont mises en œuvre entre le producteur et le consommateur.
- La visibilité des services.

2 - Le mode d'emploi utilisateur du service, qui doit répondre à des questions comme la façon d'invoquer et d'utiliser le service, du point de vue du consommateur. Cela inclut :

- L'initialisation et les contraintes d'exécution (les conditions requises pour invoquer et exécuter le service : les conditions environnementales, les délais, l'état de l'engin...).
- Les flux de données liés aux entrées du service (Description des données consommées par le service, qui peuvent être des messages ou des arguments) et aux sorties (Description des données fournies par le service).
- Les effets secondaires de la prestation qui décrivent ce qui peut éventuellement se produire et dans quelles conditions.
- Les cas de défaillances, c'est à dire une liste et une description des éventuelles défaillances de ce service. Par exemple une défaillance de la fonction *MonitoringZone* peut être *UnreachableZone*.

3 - La partie réservée au fournisseur du service qui doit offrir des informations sur la façon d'exécuter le service, du point de vue du fournisseur. Le terme générique de ressource représente tout ce qui est nécessaire pour exécuter le service (matériel, puissance, fonction, son état, le mode, etc.). Cette partie comprend, entre autres, l'état de service (disponible, indispo-

nible, init, exec, terminé ...) et les fonctions de l'engin qui sont requises pour exécuter le service.

3.4 Définition d'une ontologie générique pour l'exploration spatiale

Il a été choisi d'utiliser les ontologies, présentées dans le chapitre 2, pour modéliser le domaine de l'exploration spatiale, à savoir les engins et leur environnement.

La modélisation de l'ontologie développée ici a commencé en étudiant les ontologies existantes, présentées dans la partie 2.6. Pour la description des engins le projet A3ME a été une source d'inspiration particulièrement intéressante. Pour la partie description des services c'est l'ontologie décrite dans l'article "Test Methods and Knowledge Representation for Urban Search and Rescue Robots" qui a servi de point de départ, bien que limitée à des concepts de bas niveaux. Quand à l'utilisation de l'ontologie au sein de la mission, c'est le projet Proteus, qui a été pris en exemple.

L'ontologie développée ici a ensuite été adaptée aux besoins et spécificités de l'étude. La notion de service a notamment été travaillée pour décrire leur réalisation, le moyen de les utiliser et leurs contraintes.

D'une manière générale, elle a été conçue pour pouvoir représenter des engins d'exploration planétaire de différents types comme des rovers, des orbiteurs ou des engins volants ainsi que l'environnement dans lequel ils évoluent.

3.4.1 Les concepts, leur taxonomie et les relations entre les concepts

L'ontologie dont des extraits sont présentés sur les figures 3.1 et 3.2, propose une description des engins : leur structure, leurs capacités, les contraintes liées à ces capacités. Elle utilise une organisation hiérarchique entre les concepts (e.g. Descriptors → ServiceDescriptors → ServiceState).

L'ontologie définit également des relations binaires entre ces concepts. Comme la relation *hasDependency* qui lie un *Service* à un *AgentMode*, pour spécifier que le service en question dépend du mode courant de l'agent, par exemple le service *exploreZone* d'un rover ne sera disponible que quand celui-ci sera en mode *surface* et pas en mode *cruise*.

Ou bien la relation *hasAuthorization* qui relie un *Agent* à des certificats d'autorisation particulier, nécessaires pour demander l'exécution de certains

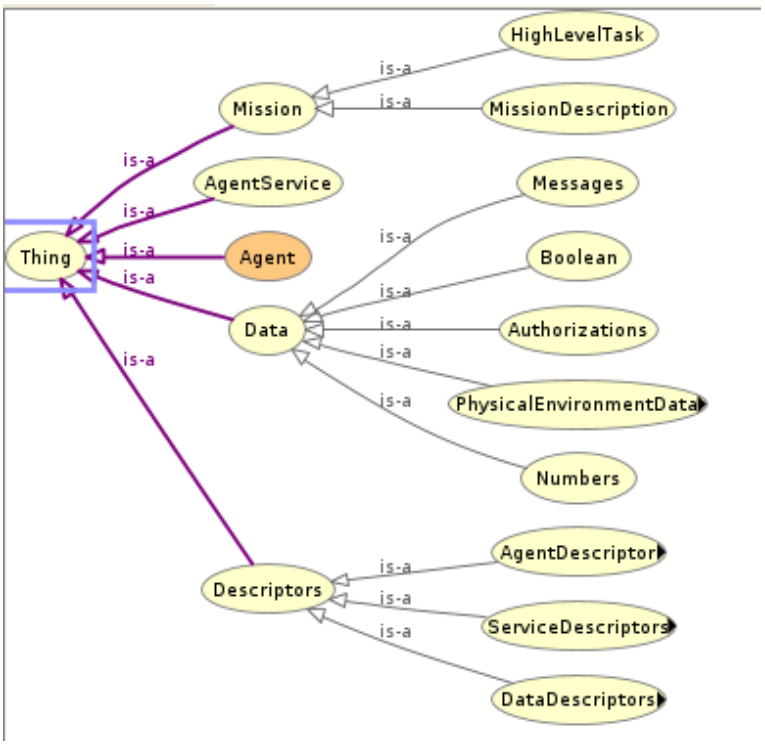

FIGURE 3.1 – Vue d'ensemble de l'ontologie

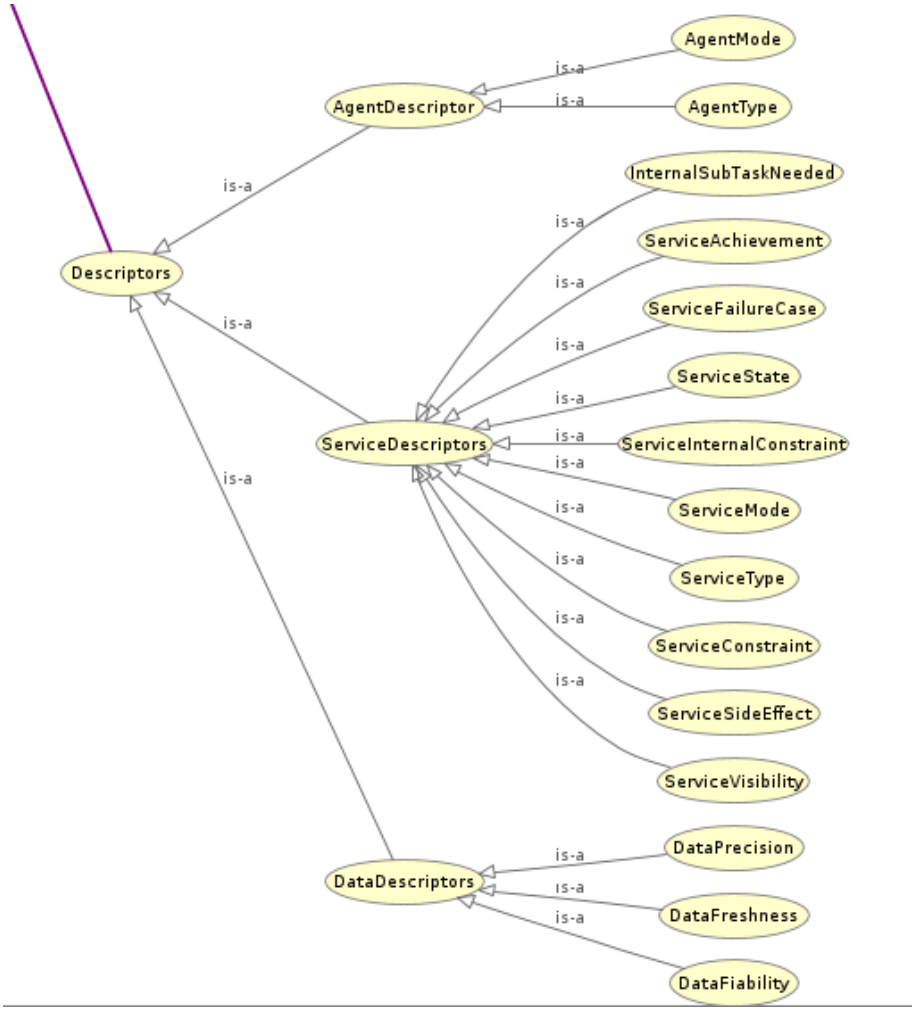

FIGURE 3.2 – Sous-partie "Descriptors" de l'ontologie

services, réservés à une agence spatiale donnée, ou limités en nombres d'utilisations et donc sujet à autorisations particulières du contrôle sol.

De la même manière des relations sont définies entre les concepts des Services, comme *hasServiceID* qui indique un identifiant de service ou *hasServiceInput* qui lie un service à un type de données nécessaire à l'exécution du service.

La figure 3.3 montre un schéma représentant ces relations. Dans l'ontologie il est également possible et utile de définir des regroupements de relations du même type comme par exemple *AgentProperties* ou *ServiceProperties*.

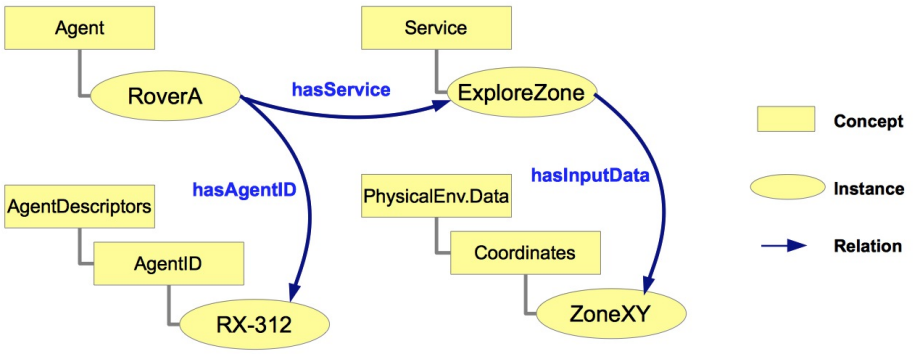

FIGURE 3.3 – Représentation schématique des relations reliant des instances de concepts dans notre ontologie

Au total, l'ontologie développée regroupe 34 concepts et 27 relations.

3.4.2 Les entrées d'un service

Il est complexe de représenter les préconditions des services qui dépendent d'une valeur donnée d'une variable dans une ontologie (e.g. Service accessible si $AgentCurrentMode = SurfaceOperationMode$), car les relations dans les ontologies sont limitées à des associations binaires (un concept lié à un second par une relation). L'adjonction de formules logiques entre les concepts, n'a pas été retenue, pour permettre l'implémentation simple et directe des relations à bord des engins. Une solution classique consiste à utiliser des concepts intermédiaires dans l'ontologie, par exemple *MustBeEqualTo* qui lierait le concept d'une variable à une instance d'une valeur.

Pour étendre cette représentation et avoir une solution plus souple, permettant de pointer vers une variable donnée d'un engin, il a été choisi d'utiliser une relation *hasConstraint*. Celle-ci lie un concept à un champ de texte dans lequel un formalisme reprend la hiérarchie de description de la variable concernée, pour y associer une valeur donnée.

Par exemple il est possible d'écrire que le service *exploreZone hasConstraint : ServiceProducer. Knowledge. Map(Service.Input : Zone). Weather = GoodWeatherCondition* pour définir que la variable *Weather* de la *Zone* donnée en entrée d'un service *exploreZone* doit être égale à *GoodWatherCondition* dans la base de connaissance du fournisseur du service.

3.4.3 Les sorties et des conséquences d'un service

Dans l'ontologie, suite aux contraintes de modélisation évoquées dans le paragraphe précédent, une solution possible est d'utiliser une troisième relation, interprétée comme "conditionnelle à l'exécution du service", qui lie les variables d'entrée et de sortie.

Cette relation, représentant les post-conditions, conditionnelles à l'exécution d'un service, serait alors modélisée grâce à un concept intermédiaire : *KnowledgeModification* (qu'il est possible de spécialiser en *PositionModification*, *MessageModification...*) qui lierait entre elles les valeurs d'entrées (pouvant provenir d'une variable également) qui seront affectées aux variables de sortie modifiées par l'exécution du service. Ce concept serait relié au service en tant que sortie.

De la même manière que pour représenter les entrées d'un service, une autre solution, retenue ici, consiste à décrire le *serviceAchievement* par une relation syntaxique et logique vers la variable de l'environnement qui doit être modifiée. Par exemple, le service *localizeAgent* qui a pour but de localiser l'engin dont l'identifiant est donné en entrée du service, est de type *UPDATE_KNOWLEDGE* (son résultat va modifier la connaissance générale de l'environnement) et sa description est *ServiceConsumer.AgentKnownList. getAgent(Service.Input :AgentID). Description.Presentation.Position.*

Ce qui signifie que la variable de la position de l'engin dont l'identifiant est donné en entrée du service va être modifiée.

Dans cet exemple on indique donc que c'est la position connue, enregistrée dans la description publique (*Description.Presentation.Position*) de l'engin dont l'identifiant est donné en entrée (*Service.Input :AgentID*) et dont la description est stockée dans la liste des engins connus (*AgentKnownList*) de l'engin utilisant le service (*ServiceConsumer*), qui va être mise à jour.

L'autre type possible de *serviceAchievement* est : *PROVIDE_DATA*, pour

les services qui donnent des information sur l'environnement, sans le modifier. La description de la réalisation du service décrit alors seulement la variable dont la valeur sera donnée.

Par exemple le service *takePicture* est du type *PROVIDE_DATA* car il fournit en sortie une image. Celle-ci est une donnée, qui sera utilisée par l'engin demandeur, ou pas, sans modifier la connaissance de celui-ci directement, ni sans modifier l'environnement.

3.4.4 Exemple de modélisation

Pour valider notre ontologie, des engins existants ont été modélisés.

Les figures 3.4 et 3.5 représentent une modélisation de l'instanciation de cette ontologie pour deux engins d'exploration existants., la figure 3.6 présente ce que pourrait être celle d'un engin futur.

3.5 Synthèse de la modélisation du système

Le système multi-engins sera modélisé comme un réseau ouvert, où les engins représenteront les nœuds.

Chaque engin embarquera un modèle de représentation de la connaissance. Il inclura l'environnement ainsi que les capacités des engins, construites autour de la notion de service. Ces services seront décrits par des concepts communs, compréhensibles par tous les engins, permettant de fournir l'équivalent d'un manuel d'utilisation des engins, à destination des engins. Mais cette représentation publique des services ne contiendra d'information concernant la manière dont les services seront exécutés. Il n'y aura donc pas de prédiction possible sur les actions précises d'un engin réalisant un service, cela limite les types d'interactions possibles entre les engins.

La proposition est de décrire la connaissance de l'environnement et les capacités des engins, représentées sous forme de services, dans une ontologie. Cette ontologie doit servir de formalisme et de standard de représentation pour les concepteurs des missions spatiales, afin de garantir l'interopérabilité entre tous les engins utilisant ce formalisme.

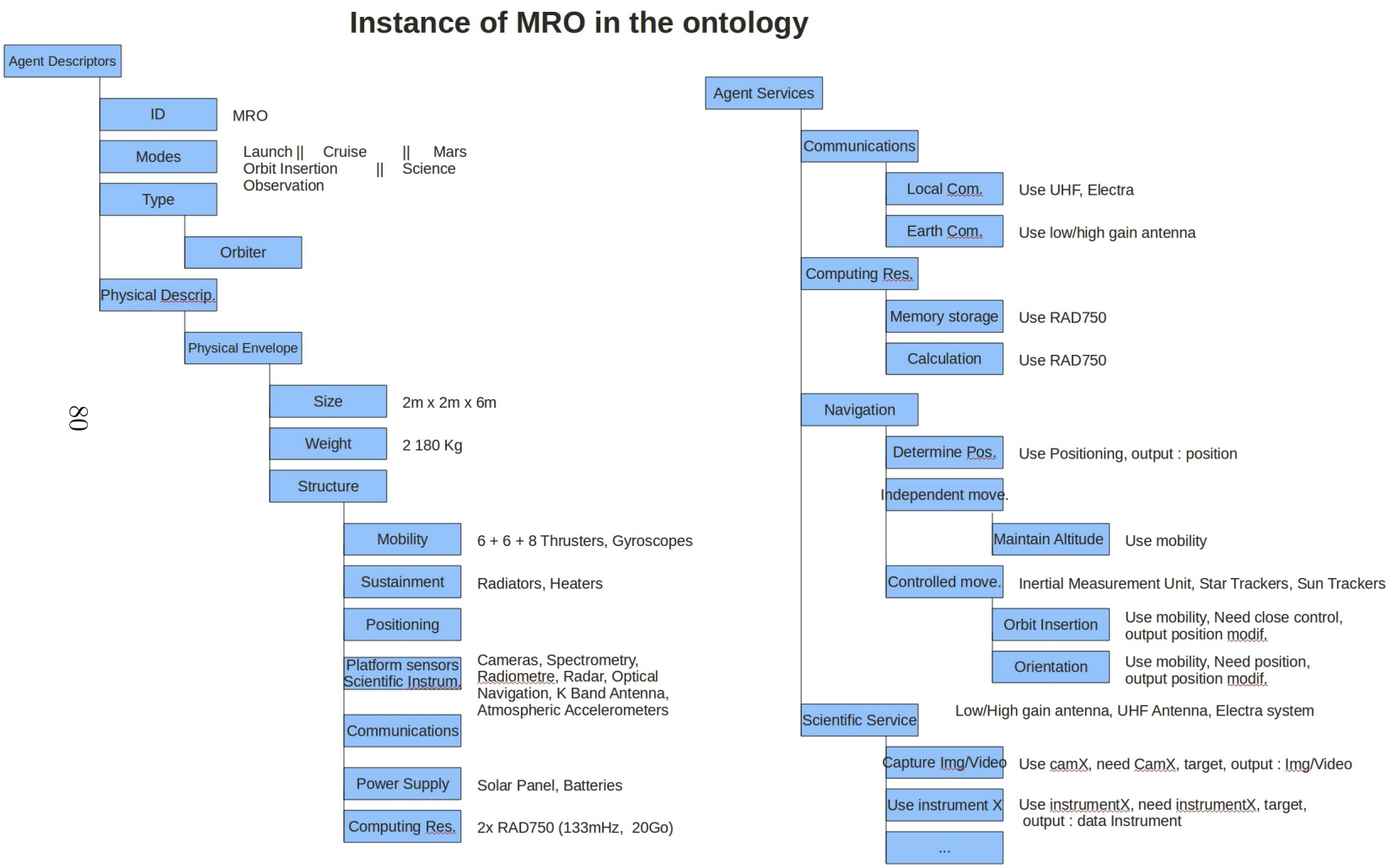

FIGURE 3.4 – Instanciation de Mars Reconnaissance Orbiteur dans l'ontologie

FIGURE 3.5 – Instanciation de Mars Science Laboratory dans l'ontologie

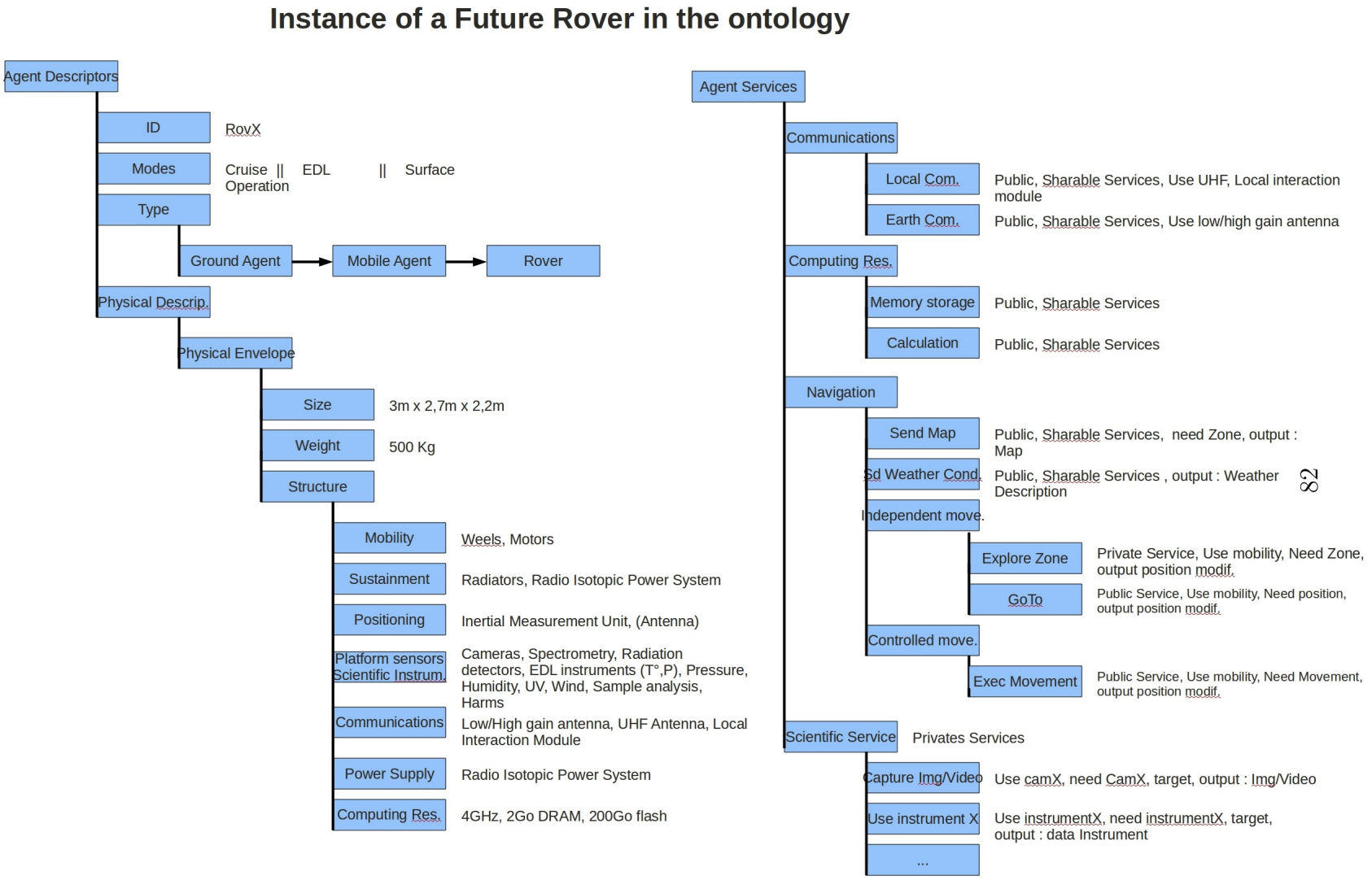

FIGURE 3.6 – Instanciation d'un possible futur rover dans l'ontologie

Chapitre 4

Gestion embarquée de la connaissance et des interactions

4.1 Objectif

Le but de ce travail est de proposer la mise en œuvre d'un système générique de gestion de la connaissance et des interactions locales (appelé "ISIS" pour In-Situ Interaction Supervisor), qui pourra être implémenté dans le logiciel embarqué au niveau des différents engins d'exploration. Ce système aura en charge de maintenir à jour la représentation des connaissances, et de partager cette connaissance avec les engins voisins, automatiquement, via des communications directes et locales.

4.2 Méthodologie de développement d'un module ISIS

Cette section propose une méthodologie de développement pour passer d'une ontologie à l'implémentation du module ISIS pour un engin donné, lors de la conception de son logiciel de bord.

L'ontologie générique, définie dans le chapitre précédent, permet de définir le domaine général des missions d'exploration planétaire.

Dans le cadre d'une mission particulière, les concepteurs d'un engin doivent créer une instance de cette ontologie pour cet engin avant son lancement, appelée *ontologie mission*.

L'utilisation de l'ontologie est consommatrice en ressource, et donc peu adaptée aux systèmes embarqués. L'ontologie ne sera donc pas directement

implémentée à bord des engins.

La méthodologie proposée pour aller de l'ontologie générique à l'implémentation spécifique d'un module ISIS comprend plusieurs étapes : développement d'une ontologie générique, adaptation de l'ontologie générique vers une ontologie mission, spécification du module ISIS, implémentation du module ISIS.

4.2.1 Développement d'une ontologie générique

Tout d'abord, une partie du travail consiste à définir une ontologie modélisant l'ensemble du contexte d'une mission d'exploration robotique. Cette ontologie est centrée autour de la notion d'engins et de services de hauts niveaux offerts par ces engins. Cela inclut également la connaissance associée aux notions d'environnement physique, description des engins, description des services. Cette ontologie doit être vue comme la composante principale du processus de standardisation et elle n'est pas restreinte à une seule mission, mais couvre autant que possible l'ensemble du domaine des missions d'exploration spatiale robotiques.

4.2.2 Adaptation de l'ontologie générique vers une ontologie mission

Quand un nouvel engin est conçu pour une mission déjà en cours ou bien à venir, une instance spécifique de l'ontologie est produite pour correspondre à la mission donnée et aux spécificités de cet engin et des services qu'il peut fournir. Cette instanciation particulière de l'ontologie, représentant un engin donné dans son environnement futur, utilisera les concepts de l'ontologie générique en l'étendant et en la raffinant si nécessaire pour correspondre aux caractéristiques de la mission et de l'engin. A ce stade, les interfaces de communication avec le logiciel de contrôle de l'engin devront être définies, ainsi que les informations à propos de l'implémentation dans l'architecture logicielle.

4.2.3 Spécification du module ISIS

Cette étape consiste à construire le module de gestion de la connaissance ISIS et la base de connaissance associée qui correspondent aux possibilités d'intégration sur l'engin. Cette transcription de l'ontologie mission, doit générer du code de manière aussi automatique que possible pour éviter les erreurs. Typiquement une ontologie peut être exportée dans un format de fichier OWL (du type XML), mais ce n'est pas une obligation d'utiliser

ce format directement dans le module. D'autres solutions sont possibles : l'une d'elle est d'utiliser un traducteur pour la transformer en représentation orientée objet, en implémentant les fonctions spécifiques associées pour exploiter cette représentation. Une autre option consisterait à utiliser des frameworks de représentation de la connaissance comme celui proposé par le projet "RoboEarth", qui permet d'encoder une ontologie en langage Prolog pour effectuer des tâches de raisonnement sur ce modèle logique.

4.2.4 Implémentation du module ISIS

Finalement le composant ISIS, destiné à être intégré dans l'architecture logicielle de l'engin, implémentera les fonctions standards chargées de gérer la connaissance locale, en la maintenant à jour et en la propageant sur le réseau d'engins. Ce module sera connecté à l'architecture spécifique de l'engin l'embarquant en utilisant des interfaces logicielles dédiées, permettant ainsi de garder une conception générique.

Il est important de noter que cette approche permet de respecter la diversité d'origines probable des différents engins. Elle impose seulement que la même ontologie soit utilisée comme point de départ (comme dans tous les standards) mais elle n'impose pas d'utiliser un moyen particulier pour créer et gérer le logiciel embarqué. La seule contrainte est qu'il puisse s'interfacer avec un module ISIS, qui sera considéré comme une brique additionnelle de l'architecture logicielle, en charge de gérer les interactions locales.

4.2.5 Problèmes de compatibilité ascendente

Dans le cas de la conception d'un nouvel engin utilisant la version courante de l'ontologie générique, par exemple l'engin Y, embarquant une version Vj de la représentation de la connaissance, sur la figure 4.1, se pose le problème de sa compatibilité avec les engins pour lesquels il avait été utilisé une version antérieure de cette ontologie, par exemple l'engin X, embarquant une version Vi de la représentation de la connaissance.

Trois situations sont à envisager :

– Si le nouvel engin embarque seulement une nouvelle instanciation de l'ontologie générique, il n'y aura aucun problème de compatibilité. Il suffira de partager la nouvelle instance avec les anciens engins pour qu'ils mettent à jour leur connaissance.

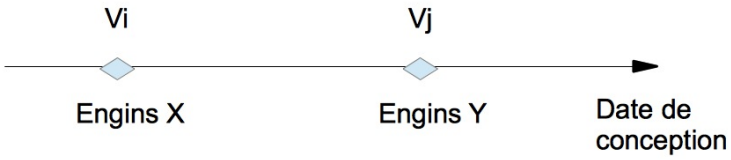

FIGURE 4.1 – Evolution de la version de la représentation de la connaissance au cours du temps

- Si le nouvel engin embarque une nouvelle version de l'ontologie, dans laquelle il a fallu définir de nouveaux concepts. Il faudra veiller à ce que ces nouveaux concepts étendent seulement la version précédente de l'ontologie, sans modifier les relations existantes. Cela permettra que le nouvel engin soit compatible avec les anciens et qu'ils puissent s'échanger des informations issues de la première version de l'ontologie. Ce sera alors à l'engin le plus récent de s'adapter à cette limitation.
- Une autre possibilité est que l'engin le plus ancien mette à jour sa version de l'ontologie, si ses ressources matérielles et logicielles le lui permettent, en intégrant les nouveaux concepts.

4.3 Intégration du module ISIS avec l'architecture locale de contrôle

Dans le cadre d'une architecture logicielle hiérarchique (Contrôleur, Superviseurs, Managers et Moniteur), le module ISIS fonctionne comme un superviseur, hiérarchiquement dépendant du Contrôleur global avec qui il communique par une interface dédiée. Comme présenté sur la figure 4.2.

L'organisation interne du module ISIS repose sur deux managers :

- Le manager des communications locales, qui est connecté à la charge utile de communication et qui a en charge la réception et l'envoi des

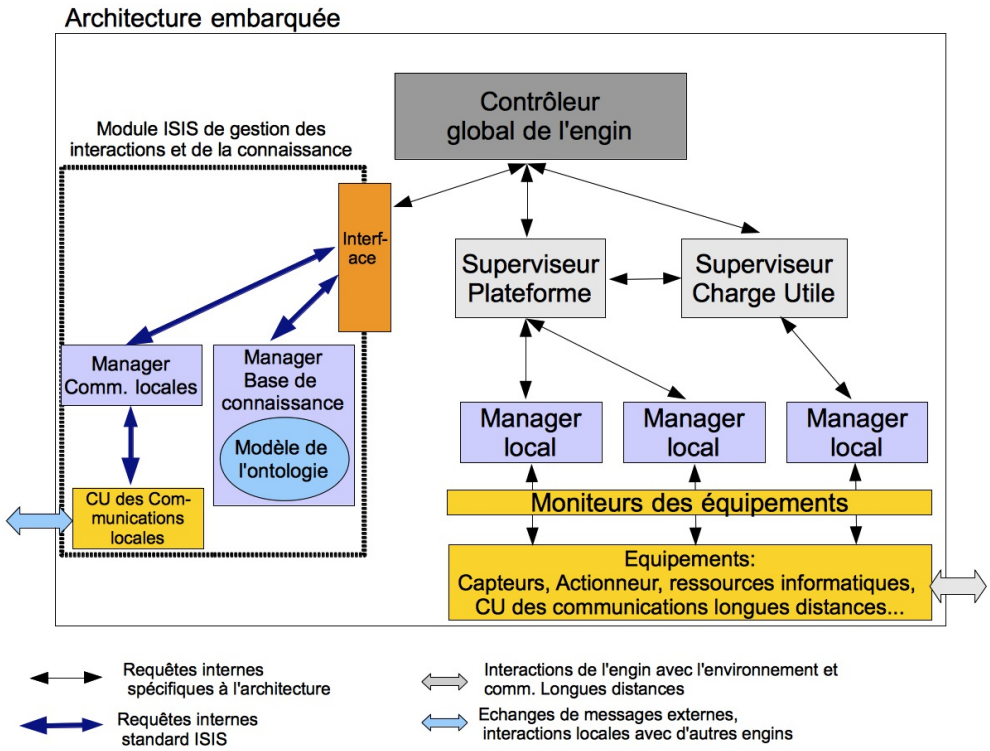

FIGURE 4.2 – Intégration du module ISIS à une architecture existante

messages en provenance ou à destination, des autres engins.
- Le manager de la base de connaissance, qui utilise, questionne et maintient à jour, la base de connaissance dans laquelle est implémenté le modèle de connaissance issu de l'ontologie mission. Ce module utilise des requêtes internes pour dialoguer le manager des communications locales et l'architecture locale de contrôle.

Les requêtes internes destinées à mettre à jour la connaissance de l'engin proviennent de plusieurs sources :
- Les messages reçus, provenant d'autres engins, via le manager de communication locales.
- Les interactions entre le gestionnaire de la connaissance et le contrôleur local, pour lesquelles deux modes sont possibles :
  - Les requêtes envoyées par le contrôleur local suite à une modification de l'état de l'engin ou bien à une nouvelle perception de l'environnement,
  - les requêtes reçues suite à l'exécution d'une routine du module ISIS, qui questionne périodiquement le contrôleur local sur l'état de l'engin.

Il est tout à fait possible que l'architecture logicielle de l'engin, si elle est déjà conçue de la sorte, garde une version de sa connaissance de l'environnement physique (terrain, carte...), sous son propre formalisme, via un de ses superviseurs ou managers. Ce qui est important c'est que celle-ci et le module ISIS se tiennent mutuellement informés de toute modification de leur connaissance.

4.4 Gestion des interactions

La figure présente une vue d'ensemble des différents types d'interactions : internes à un engin (requêtes) ou bien externes (message), entre deux engins ou entre un engin et le contrôle au sol.

4.4.1 Contrôle au sol et gestion des missions

Pour contrôler les engins, le contrôle au sol garde un lien radio direct avec ceux-ci, qui passe non pas directement par le module ISIS, mais par les antennes dédiées de la charge utile des engins, en utilisant des protocoles bas niveau du type TMTC.

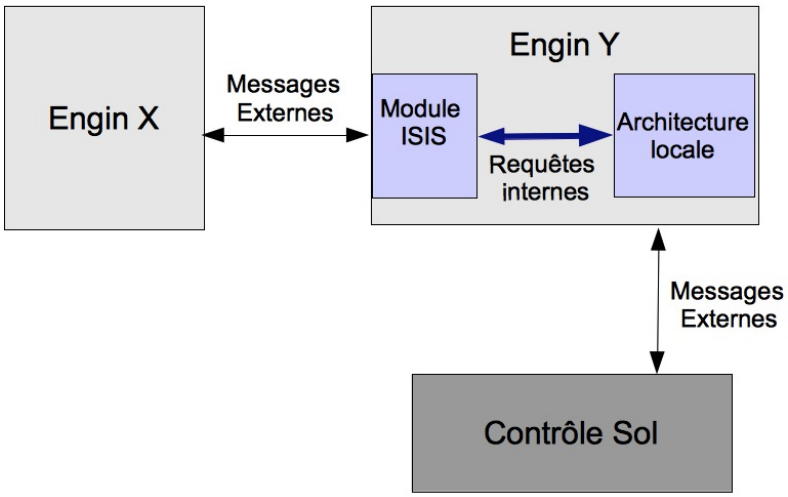

FIGURE 4.3 – Vue d'ensemble des différents types d'interactions : internes à un engin (requêtes) ou bien externes (message), entre deux engins ou entre un engin et le contrôle au sol.

Le module ISIS, en tant que gestionnaire des interactions directes et locales, est ensuite en mesure de faire transiter ces messages destinés à la planification et au contrôle des missions aux autres engins si nécessaire.

Les problématiques de synchronisation des tâches et de planification ne sont pas du ressort du module ISIS, mais seront laissées à la charge du contrôleur local de chaque engin.

4.4.2 Les interactions internes à l'engin

Les interactions internes concernent les communications qui ont lieu à l'intérieur de l'architecture logicielle de l'engin, principalement entre le module ISIS et les autres modules de l'architecture (par exemple pour envoyer des requêtes liées à l'utilisation d'un service (start, stop, resume...) ou acquérir des informations sur l'état de l'engin pour mettre à jour la base de donnée de la connaissance). L'architecture locale de contrôle dépend également du module ISIS pour envoyer des requêtes vers d'autres engins ou récupérer des informations enregistrées dans la base de connaissance (par exemple pour obtenir la liste des services disponibles offerts par les engins voisins, pour permettre de planifier une activité).

La figure 4.4 représente tous les types d'interactions qui peuvent avoir lieu un sein d'un engin entre l'architecture logicielle locale et le module ISIS.

Les types de requêtes que l'architecture locale peut envoyer au module ISIS sont : demander l'exécution d'un service à un autre engin, demander l'état d'une connaissance particulière, mettre à jour ou ajouter de la connaissance ou bien envoyer des messages de gestion d'une mission à un autre engin. Les requêtes du module ISIS destinées à l'architecture locale, de la même manière, servent à relayer des informations de mise à jour de la connaissance, de mission ou bien d'utilisation d'un service.

Au sein du module ISIS, nous trouvons les types de requêtes échangées entre le manager de la base de la connaissance et celui qui gère les communications externes, locales, à l'engin. Elles permettent de garder automatiquement à jour la connaissance de l'environnement au sein du réseau d'engin.

4.4.3 Les interactions externes entre engins

Les interactions externes concernent les communications directes entre les engins robotiques ainsi que le contrôle sol. La figure 4.5 représente tous les types de messages qui peuvent être échangés entre les engins du réseau.

Ils se décomposent en quatre grandes catégories :

– La gestion dynamique du réseau. Comme le système est ouvert, le réseau est dynamique et il faut intégrer des nouveaux engins et gérer les

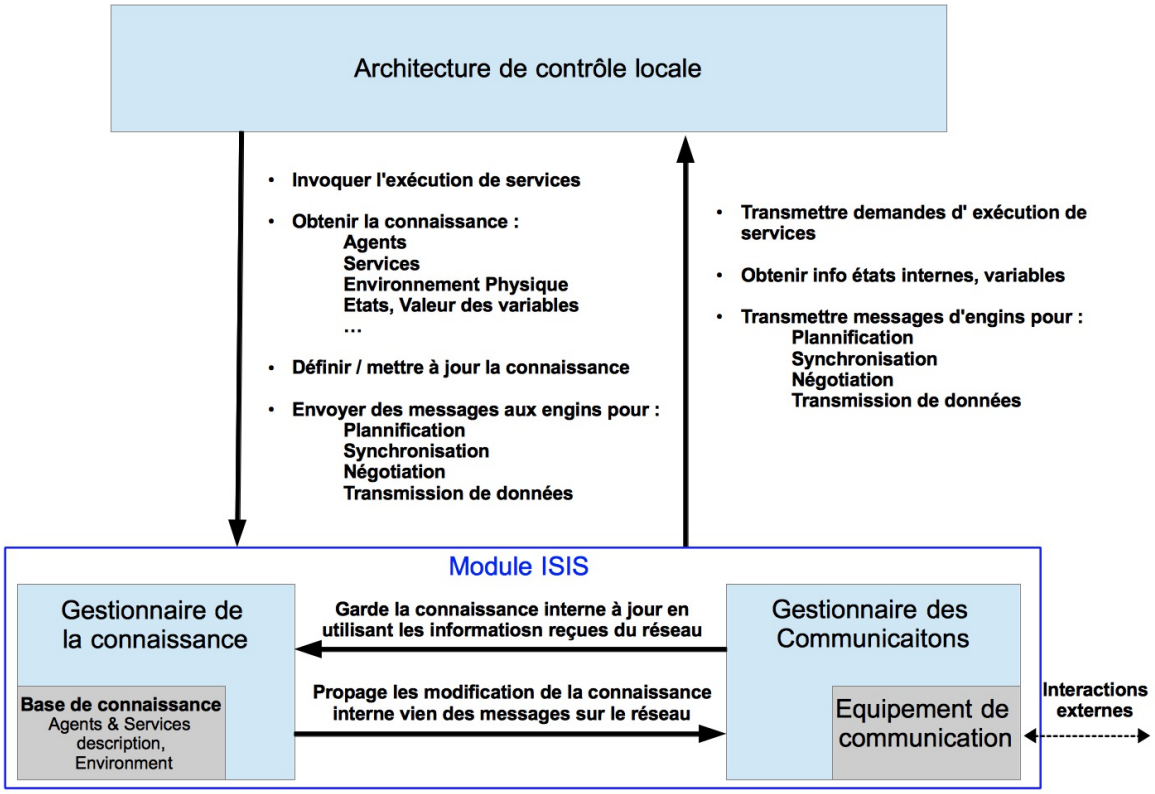

FIGURE 4.4 – Interactions internes à un engin

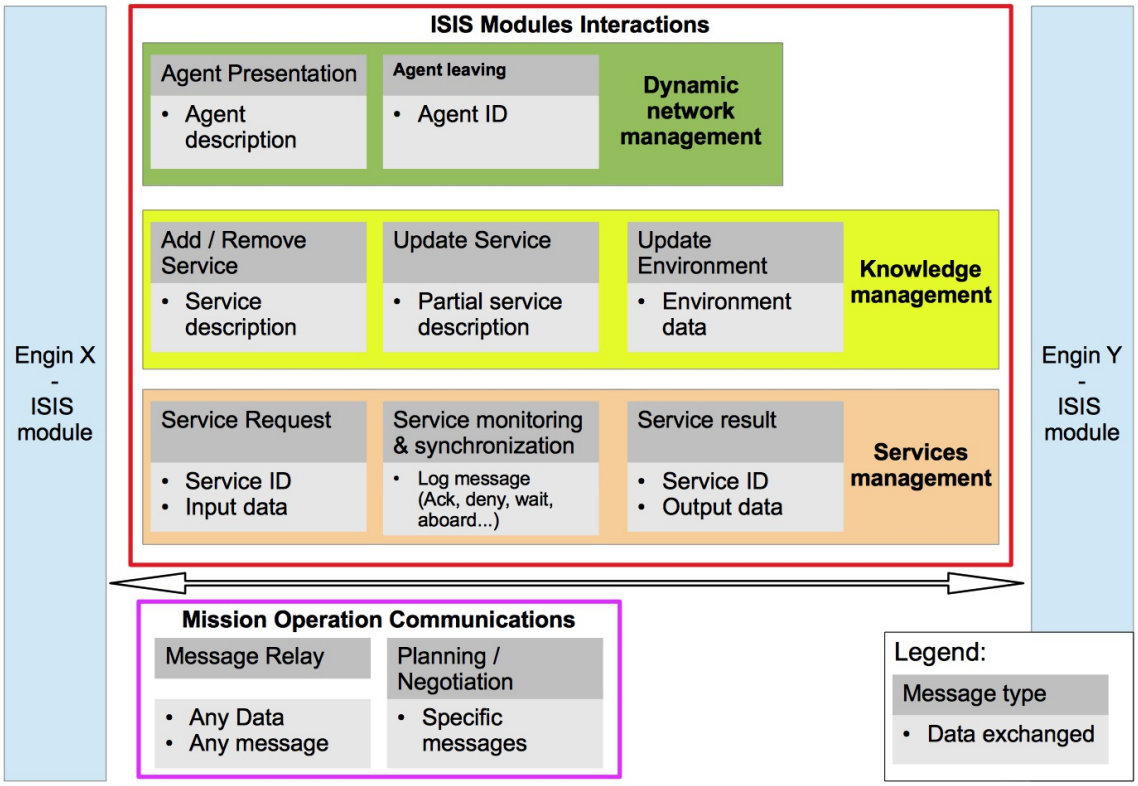

FIGURE 4.5 – Types des messages échangés, liés aux interactions externes entre engins

départs. Ici sont regroupés les messages de présentation et description des engins qui arrivent ainsi que ceux informant du départ d'un engin.
- La gestion de la connaissance qui porte sur l'état des services et de la connaissance associée à chaque engin (ajouter / supprimer un service, mettre à jour un service, mettre à jour la connaissance de l'environnement).
- L'utilisation des services ainsi que la surveillance de leur exécution et la réception de leurs résultats.
- La propagation d'objectifs mission et de messages sur le réseau afin de soutenir la planification et le travail collaboratif.

Les trois premières catégories concernent les interactions directes des modules ISIS entre eux. La dernière catégorie concerne les contrôleurs des architectures logicielles locales des engins, même si ces messages peuvent transiter via les modules ISIS.

Des protocoles d'échanges sont associés à ces catégories de messages, ils décrivent à la fois la forme des messages échangés et les actions / réactions qui sont liées aux différents types de messages.

4.4.3.1 La gestion dynamique du réseau

Quand deux engins sont dans la même zone de communication, à portée radio l'un de l'autre, ils peuvent s'envoyer un message de présentation pour s'identifier et décrire leur capacités l'un auprès de l'autre. Ce message peut être initié par le contrôle au sol, qui demande un envoi forcé du message d'un engin vers tous ses voisins. Mais il peut également être créé et envoyé automatiquement : si un engin reçoit un message d'un expéditeur qui n'est pas dans sa liste de voisins connus, il enverra automatiquement un message de présentation, auquel les autres engins qui le recevront répondront par leur propre message de présentation.

Ce type de protocole a l'inconvénient de générer beaucoup de messages et, dans un réseau de nombreux engins, il pourrait y avoir un risque de surcharge des communications. Il faudrait alors mettre en place des stratégies de propagation de l'information pour limiter les messages.

Ces stratégies pourraient consister en la limitation des messages à un certain nombre de sauts de nœuds sur le réseau par exemple, ou en durée de validité (on ne propage pas une information qui a plus de xx minutes).

Un autre type d'utilisation des interactions externes pourrait être la mise en place de routines de surveillances (ou "watchdog"). Par exemple, si un

engins n'a pas reçu de messages du contrôle sol depuis X temps, il peut décider de se mettre en mode survie pour attendre un retour à la normale de la situation.

4.4.3.2 La gestion de la connaissance

Quand les caractéristiques d'un service ou d'un élément de la connaissance évoluent au sein d'un engin, le module ISIS génère automatiquement un message de mise à jour qui est "broadcasté" à tous les engins voisins, qui modifieront alors leur propre connaissance en fonction. Un engin peut également demander à un autre la mise à jour d'une partie précise de la connaissance (carte, service, partie d'un service ou partie de la description d'un engin qu'il n'avait pas encore...).

Par exemple, si un engin détecte une panne interne qui lui rend impossible l'exécution de certains services, il va mettre à jour sa liste de services et ensuite notifier le module ISIS qui va propager cette information, via les mécanismes de mise à jour de la connaissance.

De même si un engin n'est plus disponible (mission prioritaire, autorisations modifiées ...), il va en informer ses voisins, pour que ceux-ci puissent mettre à jour leurs bases de connaissance en conséquence.

De manière indirecte, si un engin ne répond plus aux messages qui lui sont adressés, il sera automatiquement considéré comme hors service par les autres engins.

Bien qu'importants, les problèmes de consistance de la connaissance et de qualité de l'information ne seront pas abordés et devront faire l'objet d'études spécifiques. Que ce soit à cause de l'envoi par un engin d'une information fausse (mauvaise acquisition d'un capteur, bug...) ou bien à cause de la perte d'un message de mise à jour de la connaissance, un ou plusieurs engins peuvent se retrouver avec une croyance de l'état de leur environnement qui est fausse.

4.4.3.3 L'utilisation des services

Quand un engin décide d'utiliser un service, il va devoir en informer l'engin fournisseur du service. Il peut alors invoquer directement le service en fournissant les données nécessaires.

De son coté l'engin fournisseur du service va accuser réception de la demande et valider l'exécution ou la réservation du service, via son architecture de contrôle, ou bien il va la refuser, en renvoyant un message d'incapacité ou

d'indisponibilité, s'il ne peut pas satisfaire celle-ci.

4.4.4 Contrôle et la gestion des opérations entre engins

En plus des messages provenant du contrôle sol, les modules ISIS permettent de relayer des messages d'organisation et de gestion entre les engins, ces messages ayant pour but l'envoi d'objectif, la coordination ou la synchronisation de tâches entre engins.

L'information de base pour le contrôle de l'exécution des missions est le suivi de la bonne exécution des services. Les engins exécutant un service s'envoient donc des messages de monitoring entre eux. Ces messages peuvent être envoyés périodiquement tout au long de l'exécution ou bien uniquement à la fin de celle ci. Ils indiquent l'état courant de l'exécution du service.

4.5 Développement et intégration du module ISIS

Le système se doit d'être générique et modulaire pour correspondre à un standard multi-plate-formes compatible avec des architectures logicielles adaptées au domaine spatial comme Agata [2], Claraty [60], Aurora [69] ou bien celles basées sur le "Functional Reference Model" de l'ESA [95].

Un certain nombre de concepts d'architectures existantes ont donc été repris.

4.5.1 L'architecture du module ISIS

Comme présenté précédemment, les composants principaux du module ISIS sont le gestionnaire des communications et le gestionnaire de la connaissance. Le gestionnaire des communications s'occupe d'envoyer, de recevoir et de traiter les messages externes entre engins. Le gestionnaire de connaissance est lui en charge de maintenir à jour la base de connaissance.

La figure 4.6, montre l'implémentation orientée objet qui a été choisie pour le module ISIS. La classe *ISISModule* est l'objet principal. A sa création, il instancie le manager de la base de connaissance et le manager des communications.

La réception d'un message externe est notifiée aux équipements de communication de l'engin, par l'appel d'une méthode dédiée. Les messages sont modélisés par des instances de l'objet *Message*. A l'inverse, il peut créer des

messages et les envoyer sur le réseau d'engin via une méthode qui utilise les équipements de communication.

Le manager de la connaissances instancie lui l'objet *Knowledge* qui implémente la connaissance de l'engin. Il est initialisé avec la description de l'engin et de ses services, transcrit de l'ontologie mission.

Les managers communiquent via des requêtes, instances de l'objet *InternalRequest*, ils communiquent également de la même manière avec l'architecture logicielle locale.

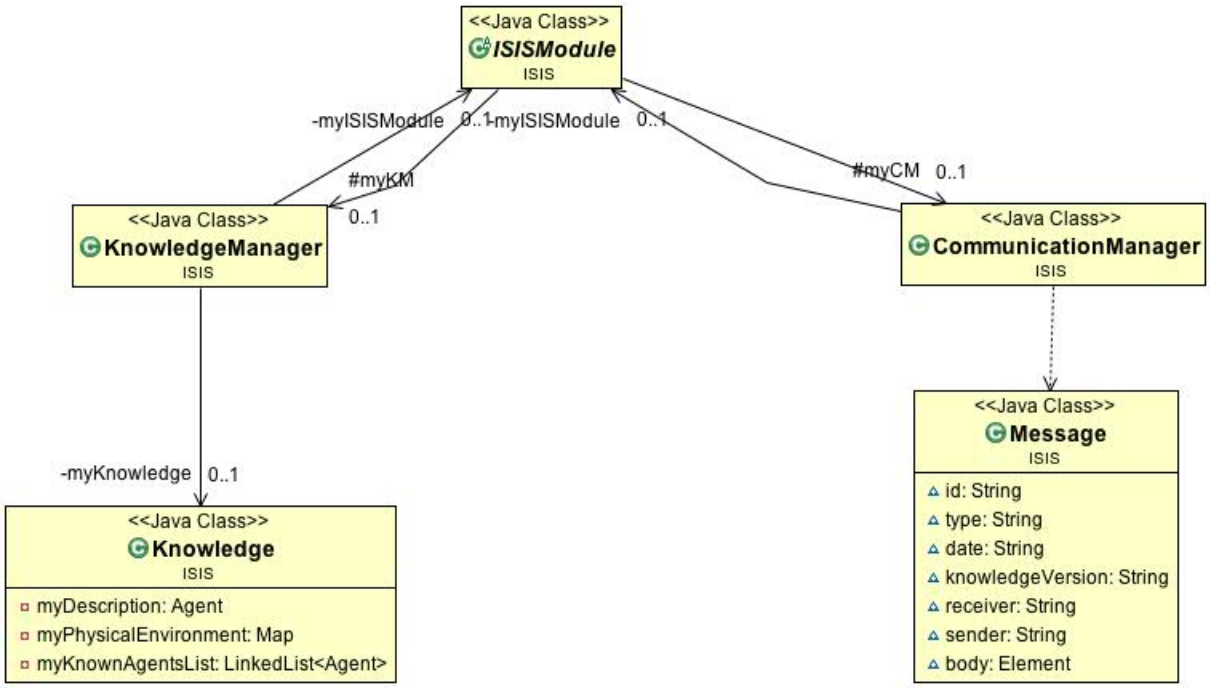

FIGURE 4.6 – Vue UML du module ISIS

4.5.2 Intégration de la connaissance dans le module ISIS

4.5.2.1 Solution idéale

A terme, il serait intéressant d'implémenter un traducteur automatique de l'ontologie mission.

Une fois l'ontologie mission dédiée à un engin, définie via un éditeur du type "Protégé", il est possible d'enregistrer l'instance de cette ontologie dans

un fichier au format XML. Ce fichier pourrait ensuite être parsé pour générer automatiquement la connaissance à embarquer dans le module ISIS.

Cette solution technique n'a pas pu être implémentée par manque de temps.

4.5.2.2 Solution adoptée

Actuellement, sans traducteur automatique, il faut définir manuellement la connaissance initiale de l'engin directement dans le code.

Une transcription de la connaissance modélisée dans l'ontologie mission est réalisée par le biais d'une implémentation en modèle objet, dont la figure 4.7 en montre une partie sous forme de diagramme UML. Pour réaliser cette implémentation nous avons appliqué les règles suivantes : les concepts de hauts niveaux, *Agent* et *Service*, ont été transformés en classes principales, chacune de ces classes contenant 3 sous classes, correspondant à chacune des sous parties de ces composants.

Dans l'ontologie ces trois sous parties sont modélisées par des regroupement de propriétés :

- *AgentProperties_Part1_Presentation* contient : *hasPosition*, *hasService*, *hasType*...

- *AgentProperties_Part2_ PhysicalDescription* contient : *hasHeight*, *hasLengh*...

Dans la modélisation objet ces descripteurs sont devenus des attributs de chacune des sous classes.

La connaissance générale d'un engin donné est modélisée de la manière suivante :

L'objet de référence est la classe *Knowldge*, chaque engin est donc équipé un module ISIS avec un *KnowledgeManager* qui possède une instance de l'objet *Knowledge*, qui contient la représentation de la connaissance. L'objet *Knowledge* contient la description de l'engin propriétaire du type *AgentDescription*, la description de l'environnement physique, ainsi qu'une liste de description des engins voisins connus. Chacune des instances d' *AgentDescription*, que ce soit celle de l'engin local ou celles des voisins, contient les trois sous classes de description d'un engin, ainsi que, pour chaque engin, de la même manière, la liste et la description des services qu'il propose.

L'organisation générale des packages du code source développé, ainsi qu'un exemple de l'implémentation de la connaissance d'un engin sont présentés en annexe C.3.

La carte géographique et les méta-données de l'environnement (le type de terrain, la météo, la position des engins ou de zones d'intérêt...) peuvent être représentées en utilisant des parties des standards des systèmes d'information géographiques mondiaux ("Geographic Information Systems" - GIS), comme ceux présentés dans le chapitre 2. Bien que ce soit complexe, il serait possible d'intégrer ces systèmes SIG dans l'organisation de la connaissance actuelle. Ils existent sous forme d'ontologies et s'intégreraient à la notre, au niveau de la description de l'environnement physique. Mais leur implémentation sous forme de modèle objet n'a pas été réalisée.

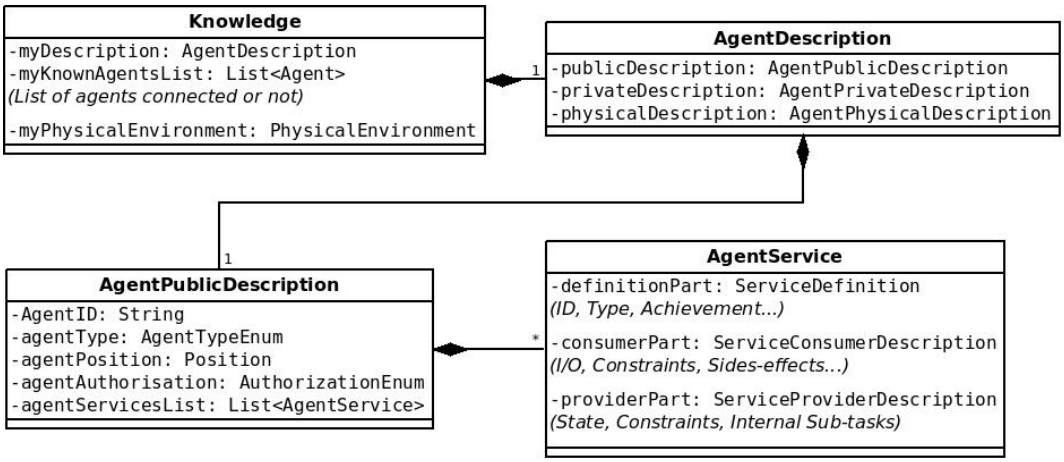

FIGURE 4.7 – Représentation UML d'une partie de l'implémentation la connaissance d'un engin

Des méthodes de parsing et de transcription ont été créées pour passer automatiquement d'une instance objet à un fichier XML et vice versa. Cette représentation garde tous les champs nécessaires à la description des services. En XML le découpage du service en 3 parties (présentation publique, partie réservée à l'utilisateur du service et partie réservée au fournisseur du service) n'est pas explicite. Celui-ci est recréé lorsque le fichier est parsé par le module ISIS pour instancier les classes définissant la connaissance et initialiser les valeurs des arguments.

Exemple de représentation XML du service "getMap" . On y trouve les informations publiques (ID du provider, ID du service, type, mode, visibilité et réalisation), et les informations réservées aux utilisateurs du service (contraintes à l'utilisation, données en entrée et en sortie, effets de bords, cas d'erreurs possibles).

```
<service providerId="BLIMP1" id="getMap" type="SOFTWARE"
mode="PASSIVE" visibility="NEIGHBORS_P0">
        <achievement type="PROVIDE_DATA" />
        <constraint type="EXEC" description="ServiceProducer.
        Knowledge.Map(Service.Input:Zone).
        Weather=GoodWatherCondition" />
        <input type="Zone" />
        <output type="Map" />
        <sideEffect type="UPDATE_KNOWLEDGE"
        description="ServiceProvider.Position" />
        <failureCase type="BAD_WEATHER_CONDITION" />
        <failureCase type="UNREACHABLE_ZONE" />
        <failureCase type="STRUCTURAL_FAILURE" />
        <failureCase type="SOFTWARE_FAILURE" />
  </service>
```

Exemple de représentation XML du service "ExploreZone". Avec également, les informations publiques (ID du provider, ID du service, type, mode, visibilité et réalisation), et les informations réservées aux utilisateurs du service (contraintes à l'utilisation, données en entrée et en sortie, effets de bords, cas d'erreurs possibles).

```
  <service providerId="ROVER1_gray" id="ExploreZone" type="
     PHYSICAL"
  mode="PASSIVE" visibility="PUBLIC">
    <achievement type="UPDATE_KNOWLEDGE" description="
       ServiceConsumer.
    Knowledge.Map(Service.Input:Zone)" />
    <constraint type="EXEC" description="ServiceProducer.
       Knowledge.
    Map(Service.Input:Zone).Weather=GoodWatherCondition" />
    <input type="Zone" />
    <output type="Map" />
    <sideEffect type="UPDATE_KNOWLEDGE"
    description="ServiceProvider.Position" />
    <failureCase type="BAD_WEATHER_CONDITION" />
    <failureCase type="UNREACHABLE_ZONE" />
    <failureCase type="STRUCTURAL_FAILURE" />
    <failureCase type="SOFTWARE_FAILURE" />
  </service>
```

4.5.3 Implémentation des interactions internes

Les interactions internes sont les échanges de requêtes entre les composants logiciels d'un même engin : entre le module ISIS et l'architecture logicielle locale de contrôle par exemple, ou entre deux parties d'un module ISIS.

4.5.3.1 Représentation

Les interactions internes sont effectuées par l'intermédiaire de requêtes et de réponses implémentées sous forme de messages XML. Comme présenté dans les exemples ci-dessous, ces requêtes contiennent les champs suivants :

- un identifiant unique reprenant l'identifiant de l'engin suivi d'un numéro de requête, par exemple ROVER1 :26, permet de distinguer les requêtes entre elles, notamment pour les phases de développement et de débuggage.
- l'identifiant du module qui a envoyé la requête : *communicationManager*, *knowledgeManager* ou *localControler*,
- le type de la requête qui peut être :
  - *isAgentAlreadyKnown*, pour savoir si un engin fait partie de la liste des voisins connus.
  - *getAgentPresentation*, pour obtenir la partie présentation publique d'un engin.
  - *addService*, pour ajouter un service à la liste des services de l'engin propriétaire.
  - *removeService*, pour enlever un service.
  - *updateService*, pour mettre à jour un service.
  - *agentPresentation*, pour décrire un engin.
  - *addAgent*, pour ajouter un engin.
  - *removeAgent*, pour enlever un engin.
  - *updateAgent*, pour mettre à jour un engin.
  - *invokeService*, pour utiliser un service.
  - *serviceMonitoring*, pour suivre l'exécution d'un service.
  - *requestServiceConsumerPart*, pour demander la partie utilisateur de la description d'un service.
  - *requestAgentPhysicalDescriptionPart*, pour demander la partie description physique d'un engin.
- les informations de date et heure de l'émission de la requête,
- un champ de données relatives à la requête (e.g. l'identifiant d'un service ou d'un engin)
- la version de la représentation de la connaissance utilisée
- et, de manière facultative, un corps contenant un élément supplémentaire

qui peut être la description de la connaissance d'une partie d'un engin ou d'un service, selon le type de requête.

Exemple d'une requête interne "simple", sans corps, envoyée par le gestionnaire des communications de l'engin "ROVER1", à destination du gestionnaire de la connaissance, afin de savoir si l'engin "ROVER2" est déjà connu ou non. :

```
<?xml version="1.0" encoding="UTF-8"?>
<internalRequest id="ROVER1:26" sender="communicationManager"
type="isAgentAlreadyKnown"
date="05/06/13.10:08:36"
knowledgeVersion="1.0"
requestData="ROVER2" />
```

Exemple d'une requête interne, avec un corps décrivant la partie publique d'un service nommé "TakePicture". Cette requête est envoyée par le gestionnaire des communication, au gestionnaire de la connaissance, pour qu'il ajoute ce service à la base de connaissance :

```
<?xml version="1.0" encoding="UTF-8"?>
<internalRequest id="ROVER1:105" sender="communicationManager"
type="addService" date="05/06/13.10:08:29" knowledgeVersion="
   1.0">
       <service providerId="ROVER2" id="TakePicture"
       type="PHYSICAL" mode="PASSIVE" visibility="PUBLIC">
              <input type="zone" />
              <output type="Picture" />
       </service>
</internalRequest>
```

4.5.3.2 Protocoles d'utilisation

Trois différents destinataires et émetteurs des requêtes internes existent, qui réagissent chacun de manière asynchrone en fonction des types des requêtes reçues :

- Le manager de la connaissance maintient à jour le contenu de la base de connaissance, à partir des requêtes en provenance manager des communications ou bien de l'architecture de contrôle qui l'informe de modifications d'état matériel, logiciel ou de changement dans l'environnement.

L'architecture locale peut questionner le manager de connaissance pour obtenir des informations sur son environnement ou sur l'état des engins voisins et de leurs services disponibles.
- Le manager des communications intervient lorsque :
  - l'architecture de contrôle veut envoyer un message à un autre engin, elle envoie une requête de transmission au manager de la communication.
  - le manager de la connaissance met à jour une information, il communique cette modification au manager des communications (si elle ne provient pas déjà de celui-ci), pour qu'il la propage au sein du réseau.
- L'architecture logicielle locale de contrôle peut recevoir des requêtes pour exécuter des services, communiquer des messages ou bien être questionnée sur l'état du monde et de l'engin.

Par exemple dans un fonctionnement nominal, comme le présente le diagramme de séquence de la figure 4.8, quand une modification de l'environnement ou de l'état de l'engin est détectée, l'architecture de contrôle envoie une requête de mise à jour au gestionnaire de connaissance, qui met à jour la représentation de la connaissance, puis envoie une requête au gestionnaire de communication pour qu'il propage cette mise à jour aux autres engins.

Les autres protocoles de gestion des interactions internes sont disponibles en annexe C.1 de ce document.

4.5.4 Implémentation des interactions externes

Les interactions externes sont les messages échangés entre les différents engins, via leurs module ISIS respectifs.

4.5.4.1 Représentation

Le formalisme de message ACL (Agent Communication Language, voir section 2.2.5) est à la base de notre implémentation et les interactions externes sont codées au format XML.

La balise message contient les champs descriptif suivants :
- un identifiant unique reprenant l'identifiant de l'engin émetteur suivi d'un nombre, par exemple ROVER1 :26,
- le type du message peut être, en reprenant les types des requêtes d'interactions internes possibles, :
  - *presentation*, pour obtenir la partie présentation publique d'un engin.

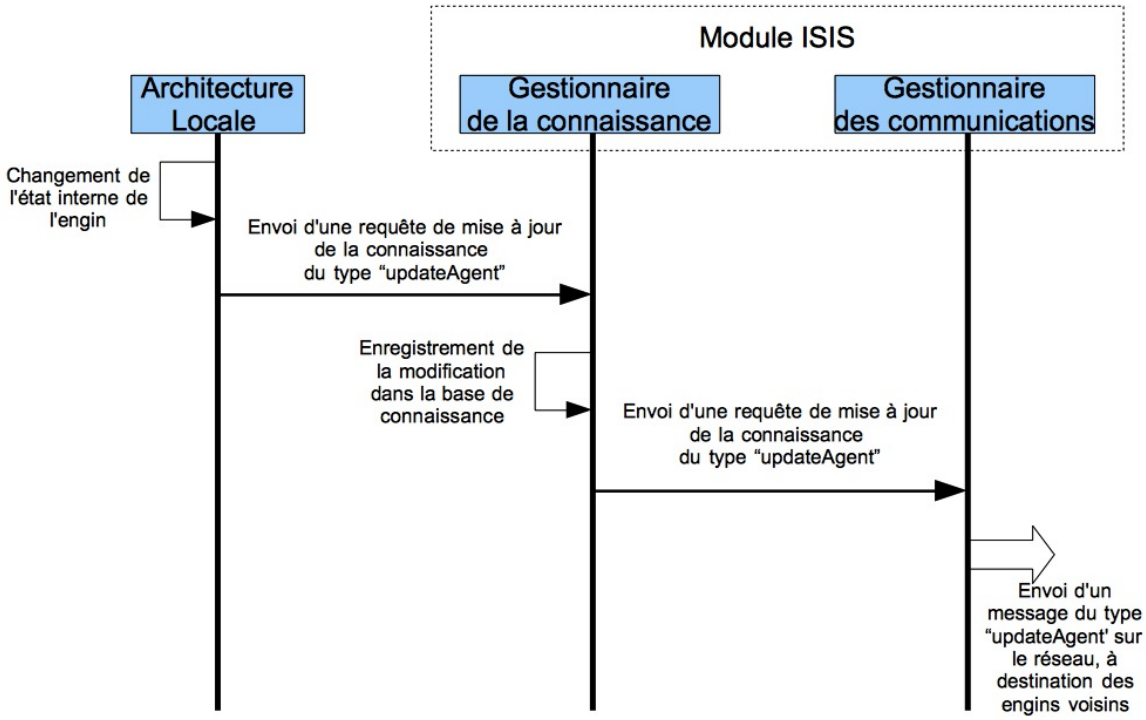

FIGURE 4.8 – Diagramme de séquence présentant les interactions internes relatives à la détection d'une modification de l'environnement par l'architecture de contrôle d'un engin.

  - *addService*, pour ajouter un service à la liste des services de l'engin propriétaire.
  - *removeService*, pour enlever un service.
  - *updateService*, pour mettre à jour un service.
  - *agentPresentation*, pour décrire un engin.
  - *addAgent*, pour ajouter un engin.
  - *removeAgent*, pour enlever un engin.
  - *updateAgent*, pour mettre à jour un engin.
  - *invokeService*, pour utiliser un service.
  - *serviceMonitoring*, pour suivre l'exécution d'un service.
  - *requestServiceConsumerPart*, pour demander la partie utilisateur de la description d'un service.
  - *requestAgentPhysicalDescriptionPart*, pour demander la partie description physique d'un engin.
- les informations de date et heure de l'émission du message,
- la version de la représentation de la connaissance utilisée

Un messages est composé de deux parties :

- Un entête (*header*), dans lequel se trouvent les identifiants de l'expéditeur du message et du destinataire. Comme tous les messages sont broadcastés à tous les voisins possibles, en utilisant un protocole et une fréquence de communication unique, il n'est pas nécessaire de préciser de port de communication ou d'adresse spécifique. Mais chaque engin qui reçoit un message doit être capable de déterminer s'il lui est destiné ou pas.
- Un corps (*body*), dans lequel se trouvent les informations utiles, de représentation d'une partie de la connaissance, qui dépendent du type du message.

Exemple de message externe, ou le ROVER1 se présente à d'autres engins. Il envoie sa description publique (ID, type, position, certificat d'autorisation) ainsi que la présentation publique du service "ExploreZone" qu'il fournit (ID, type, mode, visibilité, et réalisation) :

```
<?xml version="1.0" encoding="UTF-8"?>

<message id="ROVER1:362" type="presentation" date="
   05/06/13.10:07:58"
knowledgeVersion="1.0">
<header>
```

```
        <sender id="ROVER1_gray" />
        <receiver id="all" />
</header>
<body>
        <agent id="ROVER1" type="ROVER" position="
           5.0;139.94;0.0" >
                <authorization id="ESAAUT" />
                <service providerId="ROVER1" id="ExploreZone"
                type="PHYSICAL" mode="PASSIVE" visibility="
                   PUBLIC">
                        <achievement type="UPDATE_KNOWLEDGE"
                        description="ServiceConsumer.Knowledge.
                        Map(Service.Input:Zone)" />
                </service>
        </agent>
</body>
</message>
```

4.5.4.2 Protocoles d'utilisation

Les messages externes sont émis et reçus par les gestionnaires de communications de chaque module ISIS. Ceux-ci sont traités en fonction de leurs types et transformés en requêtes internes qui sont adressées soit au gestionnaire de la connaissance, soit au contrôleur de l'architecture logicielle locale.

Par exemple, comme représenté sur le diagramme de séquence de la figure 4.9, quand un gestionnaire de communication reçoit un message d'un engin qui n'est pas inscrit dans la base de connaissance, il envoie automatiquement un message de présentation à celui-ci. Qui répondra lui même par un message de présentation. Le gestionnaire de communication transmettra alors une requête de mise à jour de la connaissance au gestionnaire de connaissance. Ce mécanisme est appelé *protocole de présentation automatique*.

S'il reçoit un message de planification ou de demande d'exécution de service, il transmettra directement une requête correspondante à l'architecture de contrôle.

Les autres protocoles de gestion des messages externes sont disponibles en annexe de ce document.

Via l'utilisation de ces messages externes, un risque existe d'enregistrer une information fausse dans la connaissance, à cause de l'altération partielle d'un message. Cela pourrait être dangereux pour une mission, mais pourrait être évité relativement simplement en intégrant un calcul de somme de

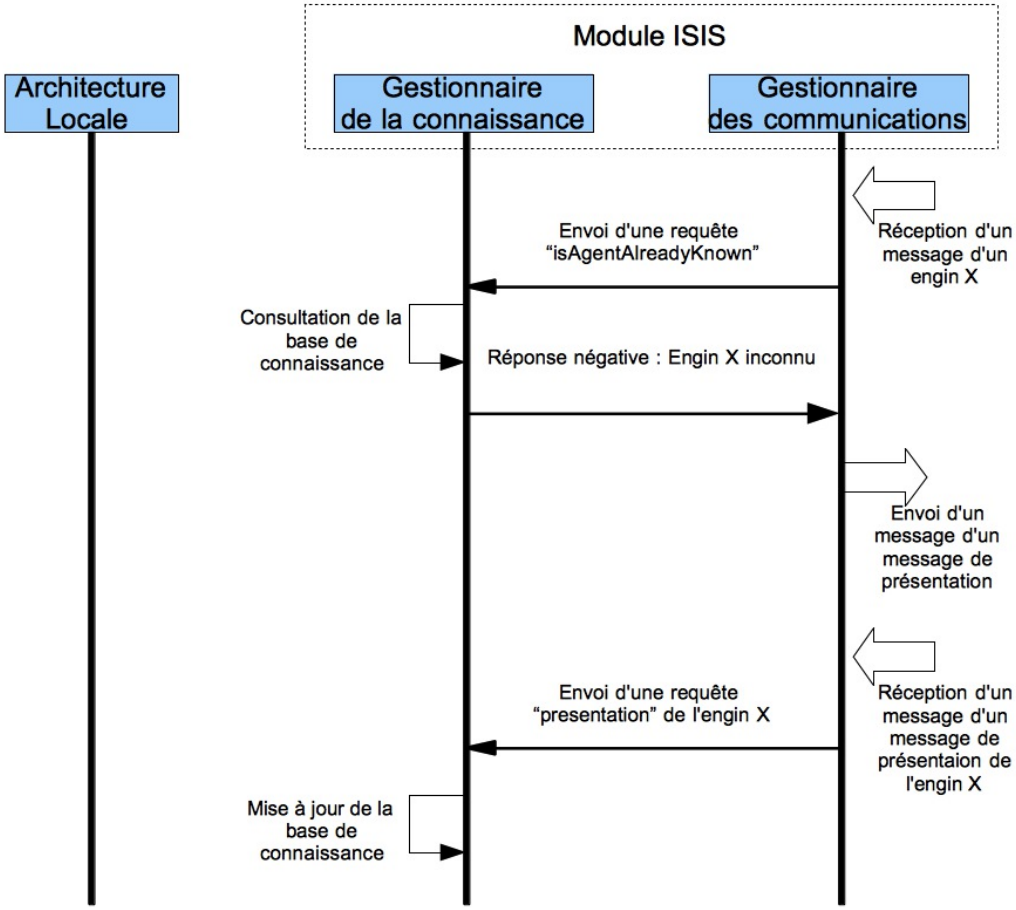

FIGURE 4.9 – Diagramme de séquence présentant les interactions externes relatives à la réception d'un message par un module ISIS d'un engin.

contrôle (l'empreinte du message, ou "checksum" en anglais). Ce type de correction est en général implémentée à un plus bas niveau dans les systèmes qui gèrent les communications.

4.6 Synthèse de la gestion de la connaissance

L'objectif de ce chapitre était de définir et de modéliser le module ISIS en charge de la gestion de la connaissance et des interactions au sein de chaque engin du réseau d'exploration.

Pour cela il a été proposé une méthodologie de création et de gestion de la connaissance des engins, comportant 4 étapes : le développement de l'ontologie générique, l'adaptation spécifique de l'ontologie, la transcription de l'ontologie et l'implémentation du module ISIS dans une architecture logicielle classique.

Un ensemble de messages et de protocoles associés à la gestion de ces interactions a été proposée, que ce soit celles internes à l'engin, ou bien celles externes entre les différents engins.

Une manière d'encoder les requêtes internes, et les messages externes a également été proposé. Un travail a été réalisé pour permettre l'échange de la représentation des services entre les différents engins par le biais de ces requêtes et messages.

Chapitre 5

Développement d'un environnement de simulation

5.1 Objectifs

Afin de valider la représentation du système sous forme de réseau ainsi que la présence d'un module ISIS sur chaque engin impliqué, un simulateur dédié est nécessaire.

Ce simulateur doit permettre de représenter un système d'exploration multi-engins, ainsi que les communications (et leur limitations) entre ces engins, afin d'étudier le comportement du système dans différentes situations.

Ces simulations serviront à vérifier que les trois points suivants permettent bien d'obtenir le comportement souhaité du système :

- la représentation de la connaissance choisie,
- le mode de gestion de cette connaissance avec ses fonctions associées,
- les différents types d'interactions entre les engins, basées sur la notion de service.

L'accent sera mis sur les fonctions embarquées au niveau de la gestion de la connaissance. Les moyens de déplacements des engins ainsi que la mise en œuvre réelle des différents services ne seront pas détaillés.

Ce travail comprend plusieurs étapes dont les principales sont :

- Modélisation de l'architecture interne d'un engin, qui inclut un module ISIS.
- La simulation de l'environnement dans lequel évoluent les engins, avec principalement la prise en compte de contraintes de communications.
- Le monitoring et la gestion de la simulation.

5.2 L'environnement de simulation

5.2.1 Organisation générale de l'architecture du simulateur

L'architecture globale du simulateur est modulaire, comme présenté sur la figure 5.1, avec une représentation de chaque engin et un simulateur de communication. Les engins seront simplement modélisés par une architecture locale de contrôle et une instance d'un module ISIS. Ils seront tous connectés directement au simulateur de communication qui s'occupera de gérer la distribution ou la non distribution de chaque message émis par un engin en fonction de la position et des caractéristiques de chacun.

Dans un souci de compatibilité et de standardisation, l'implémentation du simulateur repose sur ROS [1]. Les modules ISIS seront inclus dans des nœuds ROS, et la connexion avec le simulateur de communication se fait via des Topics [2]. Chaque engin aura deux Topics (émission/réception) dédiés au contrôle et au monitoring de la simulation des communications et deux topics par équipement de communication simulé qu'il embarque, comme détaillé sur la figure 5.2.

5.2.2 Architecture détaillée et implémentation du simulateur

Dans notre simulateur, l'architecture logicielle de l'engin est codée en Java. Comme représenté sur la figure 5.3 deux objets principaux la composent :

- l'architecture logicielle locale qui simule le fonctionnement de l'engin, ses prises de décision et l'exécution de ses services ;
- le module ISIS qui contient les sous-parties de gestion de la connaissance et des interactions, comme présenté dans le chapitre "Modélisation du système multi-engins".

La partie simulateur de communications entre engins d'exploration a été développé à l'ONERA, lors d'un stage par Sébastien Carrière. Elle utilise le moteur graphique et les outils de calcul de visibilité entre engins du logiciel

1. Robot Operating System (ROS) - Middleware robotique qui est présenté en B de ce document.
2. Topic - Moyen décharger des messages avec l'architecture ROS.

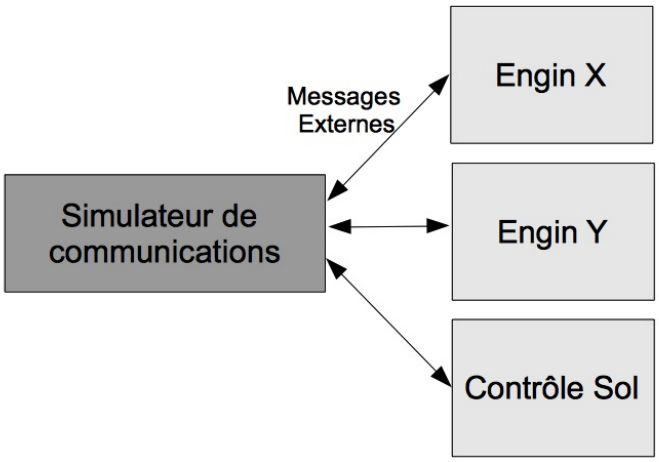

FIGURE 5.1 – Organisation générale de l'architecture de simulation, chaque engin est connecté uniquement au simulateur de communication, qui gère la redistribution des messages

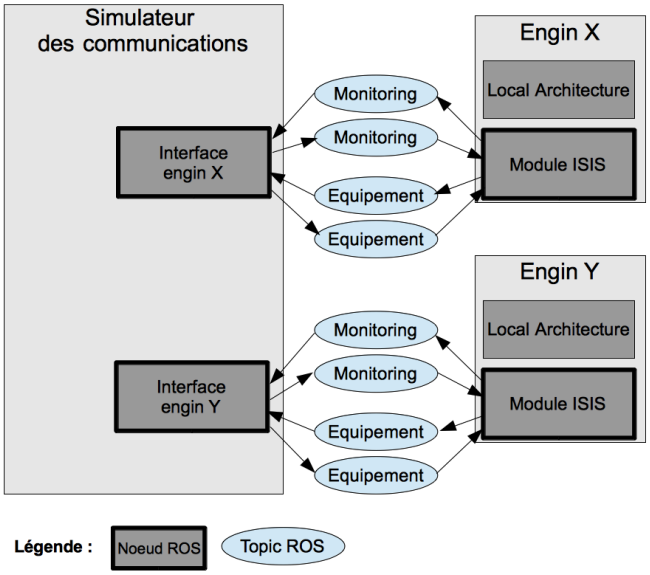

FIGURE 5.2 – Organisation de l'architecture de simulation, utilisant des Nœuds et Topics ROS pour les interfaces entre les engins et le simulateur de communications

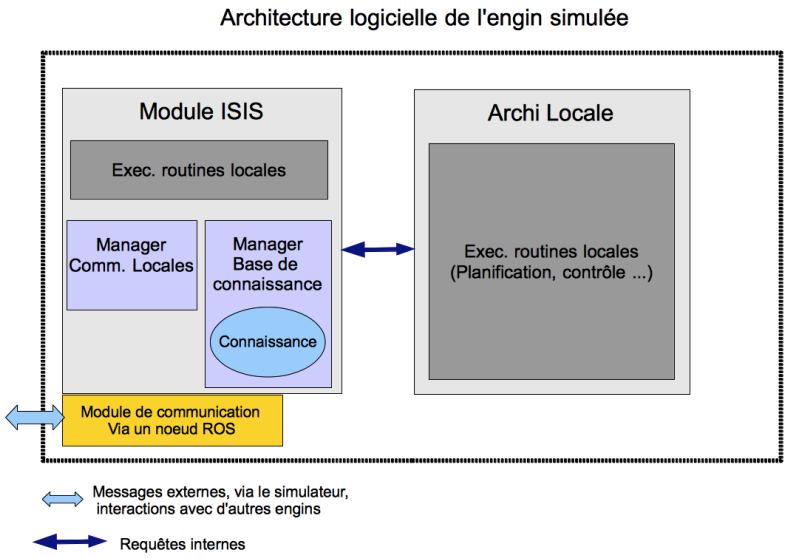

FIGURE 5.3 – Organisation logicielle d'un engin au sein de l'environnement de simulation

STK [3].

Il permet :

- de visualiser les engins qui peuvent être répartis à la surface et en orbite de différentes planètes,
- de simuler le comportement des communications radio entre eux.

Il est possible de simuler différents équipements de communication, utilisés par le module ISIS (antenne UHF, bande X...), avec différentes caractéristiques (portées des communications, émission directionnelle ou omnidirectionnelle, probabilité d'altération des messages).

Ces équipements sont modélisés dans le simulateur par un nœud ROS *Interface*, qui utilise des "Topics" *Equipement* (envoi / réception de messages) et *Monitoring* (gestion des paramètres de simulation) pour dialoguer avec le simulateur de communications, voir figure 5.2.

Le système de simulation des communications utilise un fichier de configuration où doivent être définies les caractéristiques de chacun de ces équipements. Leurs variables sont :

- L'identifiant de l'antenne (utilisé par les engins pour ajouter l'équipement de communication souhaité).
- Le type de l'antenne : directionnelle (communication avec un seul engin à la fois et longue portée) ou omnidirectionnelle.
- La prise en compte ou non du retard lié à la vitesse de propagation des ondes.
- La prise en charge ou non d'une portée limite des communications (différente pour les communications sol / sol sur une même planète et pour les communication sol / orbite ou sol / sol d'une planète à une autre).
- Le taux d'altération des données d'un message.
- Le taux de perte de données d'un message.
- Le taux de perte de messages complets.
- La prise en compte de la bande passante a été envisagée mais n'a pas été implémentée pour l'instant.

Bien que ces équipements comprennent en réalité à la fois des antennes et des systèmes électroniques de traitement de signal, ils sont nommés *antennes* dans notre simulateur. Chaque engin peut ensuite embarquer un ou plusieurs équipements de ce type.

3. Logiciel de simulation des communication et de visualisation graphique des engins, détaillé en annexes B de ce document.

Typiquement, chaque engin embarque une antenne longue portée pour les communication avec le sol et une antenne courte portée de communications locales.

5.3 La supervision de la simulation

5.3.1 Description d'un scénario

Dans nos simulations, l'objectif n'est pas de travailler sur la prise de décision ou l'exécution des services, mais de se focaliser sur l'étude des interactions liées à des situations données. Pour créer et rejouer ces interactions, il faut définir des scénarios de mission impliquant plusieurs engins.

Un scénario est décrit par un fichier en langage XML décrivant les actions des engins de manière séquentielle. Un scénario est interprété par chacun des engins impliqués.

Pour simplifier la gestion du temps et le séquencement des actions réalisées par les engins, l'hypothèse d'une horloge commune a été retenue. La gestion du temps est discrète et les problèmes de synchronisation d'horloges ne sont pas pris en compte. Chaque action, dans le scénario, est caractérisée par le moment auquel elle doit être exécutée et à l'engin qui doit l'exécuter.

Par exemple, l'envoi d'un message de présentation, par l'engin ROVER1, au temps 8 secondes, est modélisé par :

```
<time t="8">
        <action agentID="ROVER1" type="
            sendXMLPresentationMessage">
        </action>
</time>
```

De la même manière pour que l'engin ROVER1, demande l'exécution du service "goTo" à l'engin SCOOT1, au temps 26, il est spécifié :

```
<time t="26" >
        <action agentID="ROVER1" type="sendInternalRequest">
                <internalRequest sender="LocalControler"
                receiver="CommunicationManager"
                type="invokeService" displayAnswer="true">
                        <service providerId="SCOOT1" id="goto">
                                <input type="position" value="
                                    12;13;4" />
```

```
                    </service>
                </internalRequest>
            </action>
</time>
```

Les actions qu'il est possible de réaliser depuis un script de scénario sont les suivantes :

- Ajouter / enlever un équipement de communication à un engin.
- Afficher un message de suivi de l'exécution des actions.
- Envoyer une requête interne au module ISIS de l'engin.
- Faire sortir un engin de la simulation, pour simuler son départ ou son arrêt.

5.3.2 Contrôle des engins

Le comportement des engins en réaction à différentes situations est simulé de manière simplifiée.

Il y a deux moyens possibles pour contrôler les actions et réactions des engins lors de la simulation. Les décisions et actions d'un engin peuvent être décrites dans un scénario ou bien implémentées directement dans une routine de simulation des décisions au niveau de l'architecture locale de contrôle. Pour une même simulation, on peut utiliser les deux simultanément.

La première possibilité est de décrire les actions des engins par le biais du script de scénario. A son initialisation l'instance de l'architecture logicielle locale des engins parse le script à la recherche d'actions qui lui sont assignées, puis elle les enregistre pour les exécuter au temps souhaité.

La seconde possibilité est d'écrire une routine de contrôle au niveau de chaque architecture logicielle locale pour définir : le temps que mettra l'engin à exécuter le service, les messages de monitoring qu'il enverra durant ou à la fin de cette exécution, la mise à jour de la connaissance de l'engin (déplacement de l'engin par exemple), ainsi que le résultat final du service qui peut être celui escompté, ou pas (problème possible de déplacement, de communication, de calcul ...). Cette routine représente un modèle agrégé et simplifié de l'architecture locale de contrôle de l'engin.

La figure 5.4 schématise les relations qui existent entre le module ISIS, l'architecture locale de contrôle et le script du scénario, pour jouer un scénario donné.

Cette modélisation est volontairement simplifiée par rapport à une implémentation complète et réaliste d'une architecture de contrôle.

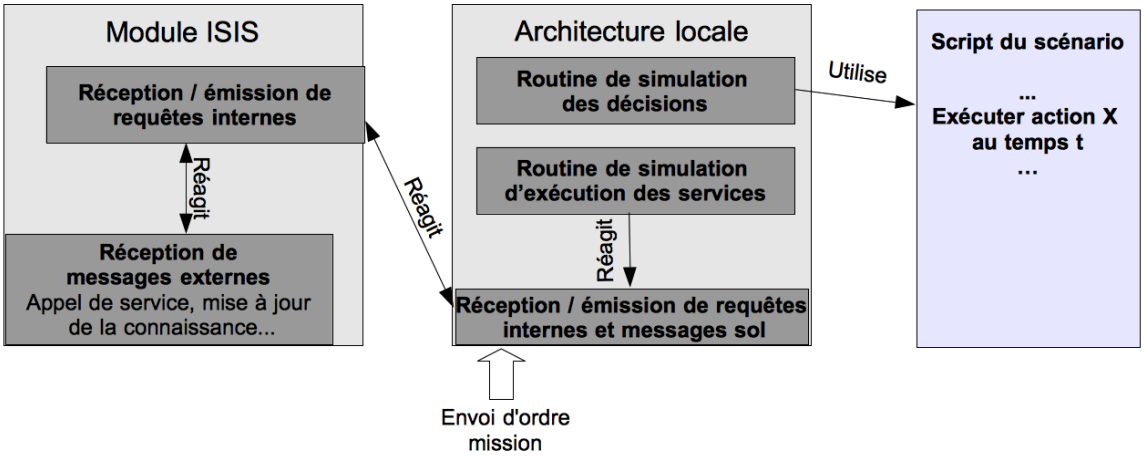

FIGURE 5.4 – Relations existantes entre le module ISIS, l'architecture locale de contrôle et le script du scénario, pour le jeu d'un scénario.

5.4 Exécution d'une simulation

Chaque engin simulé est défini par deux classes, l'une représentant son module ISIS, incluse dans un nœud ROS et l'autre représentant son architecture locale, comme présenté en annexe C.3.

Le diagramme de séquence de la figure 5.5 représente la phase d'initialisation de la simulation et un exemple de phase de simulation proprement dite.

5.4.1 Phase d'initialisation

Le simulateur de communication est lancé sur une machine dédiée, celui-ci attend alors que des engins se connectent. Le "ROSMaster" est lui aussi lancé, il est en charge des messages et des modules ROS.

Puis une interface ROS est créée pour chaque engin grâce à une méthode dédiée. Chacune de ces interfaces se connecte au serveur du simulateur des communications et crée deux topics d'administration, un pour recevoir et l'autre pour envoyer des informations sur la simulation des communications.

Pour démarrer effectivement la simulation, chaque classes du module ISIS, de chaque engin est appelée par un lanceur dédié de ROS ("RosRun").

Lors de leur démarrage, les modules ISIS chargent l'état de leur connaissance initiale, via une routine d'initialisation. Puis ils invoquent une ins-

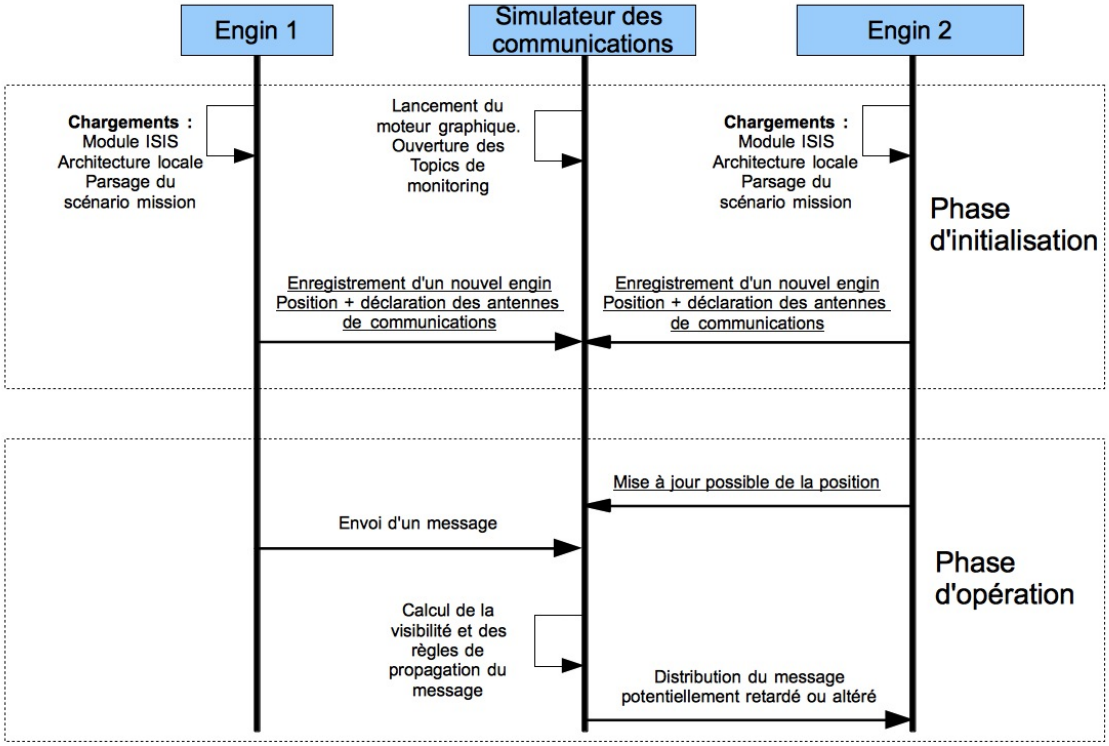

FIGURE 5.5 – Diagramme de séquence représentant les différentes étapes, du lancement à l'exploitation, d'une simulation. Les actions soulignées représentent les messages de monitoring de la simulation.

tance de simulation de l'architecture locale de l'engin sur lequel ils sont implémentés, avec laquelle ils pourront communiquer grâce aux requêtes internes. Cette architecture locale parse le script de scénario pour connaître les actions qu'elle aura a réaliser.

Les engins peuvent alors se connecter aux topics d'administration qui leurs sont dédiés (chaque topic à un ID unique, en fonction de l'ID de l'engin correspondant).

A leur connexion, les engins envoient un message d'identification qui spécifie le type de l'engin (orbiteur, véhicule ou station), la planète où il sera créé et sa position initiale (latitude et longitude ou bien éléments orbitaux).

Parmi les premières actions d'un scénario, il est ensuite possible d'ajouter à chaque engin des équipements de communication appelés "antennes". Ces "antennes" simulent les caractéristiques d'émission et de réception (débit, portée, retard, altération...) d'un équipement donné.

Chaque antenne ajoutée entraîne la création de deux topics, par l'interface ROS du simulateur de communications. Un pour émettre des messages l'autre pour recevoir des messages, selon cet équipement et ses caractéristiques.

La simulation est alors fonctionnelle et les modules ISIS des engins peuvent utiliser les topics simulant les équipements de communication pour envoyer et recevoir des messages.

5.4.2 Phase d'exécution

Durant la simulation, les engins peuvent dynamiquement interagir avec le simulateur.

Le format général des requêtes adressées au simulateur est le suivant : [Type de message] + [Paramètres/Contenu].

Les différentes actions possibles sont :

- Connexion d'un nouvel engin. En précisant son type, sa planète et sa position initiale.
- Mise à jour de la position d'un engin.
- Déclaration de l'utilisation d'un nouvel équipement de communication.
- Envoi d'un message.
- Départ d'un engin de la simulation.

En fonctionnement nominal, les engins envoient leurs nouvelles positions, via le topic ROS d'administration du simulateur, dès que celle-ci change, à une fréquence maximum 1Hz. Ensuite les modules ISIS utilisent les topics ROS associés à des équipements de communication ("antennes") pour

envoyer des messages.

Le simulateur utilise les positions des engins et les paramètres des antennes pour calculer la visibilité de ces équipements et les altérations ou délais à appliquer aux messages. Puis il publie les messages, éventuellement modifiés et retardés, sur les topics ROS correspondants aux antennes en visibilité de l'antenne émettrice.

Les messages peuvent être reçus depuis un autre engin (communications locales) ou bien depuis le contrôle sol (communications longues portées).

5.5 Synthèse des développements

A partir d'un modèle simplifié d'engins basé sur un module ISIS et une architecture locale de contrôle, un environnement dédié de simulation a été créé.

Cet environnement de simulation permet de gérer à la fois :

- les interactions internes avec l'architecture logicielle locale des engins,
- les interactions externes entre engins, via l'échange de messages.

La gestion de ces communications est simulée, le logiciel de simulation des communications permet de gérer des contraintes de visibilité des engins selon leurs positions et les équipements de communication qu'ils embarquent. Il est possible d'y simuler des délais et des altérations liés à la transmission des messages dans un contexte d'exploration planétaire.

On peut également visualiser l'ensemble du système avec la position des engins sur les planètes.

Il n'a pas été implémenté :

- de moteur physique de déplacement des engins et d'interaction avec leur environnement,
- de fonctions de prise de décision et de planification au sein des engins pour gérer leurs actions.

Chapitre 6

Expérimentations

6.1 Objectifs

Les tests et la validation du système développé et des mécanismes d'interactions se font par simulation. Des scénarios réalistes, comprenant plusieurs véhicules de différents types sont utilisés et joués par les engins.

Parmi les critères d'évaluation possibles, qualitatifs et quantitatifs, on s'intéressera en priorité à :

- La possibilité de modéliser la connaissance de la mission (les agents et leurs services) dans l'ontologie et de l'implémenter ensuite dans le module ISIS.
- La possibilité pour les modules ISIS de maintenir à jour et d'échanger leur connaissance durant l'exécution d'un scénario.
- Le nombres de messages échangés entre engins pour exécuter le scénario.
- Le nombre de requêtes internes nécessaires à l'exécution du scénario, entre les composants du module ISIS et entre le module ISIS lui même et l'architecture logicielle locale

6.2 Génération des modules ISIS

6.2.1 Création de l'ontologie mission

Comme expliqué dans la section 4.2, la première étape de la génération du module ISIS consiste à décrire la connaissance de chaque engin dans une instance d'*ontologie mission*.

A partir de l'*ontologie générique* décrivant les concepts de missions d'exploration (engins, services...), voir chapitre 3, on instancie une ontologie mission pour chaque engin, où des relations entre l'instance donnée d'un engin

et des instances représentant la valeurs d'une variable sont rajoutées.

Par exemple la figure 6.1 montre comment, dans l'outil d'édition d'ontologie Protégé, on spécifie que le type de l'engin nommé "Rov0" (concept de l'ontologie mission) possède la valeur "ROVER" (concept de l'ontologie générique). Sur la gauche de la fenêtre, sont affichés tous les types de propriétés possibles, organisés par groupes, suivant l'organisation proposée dans le chapitre 3. A droite de l'image sont affichées toutes les instances existantes de l'ontologie.

Un inconvenant de cette méthode est que cette interface de Protégé n'est pas ergonomique, en effet cette liste d'instance devient rapidement très importante et il est alors difficile de l'utiliser et de retrouver la valeur recherchée.

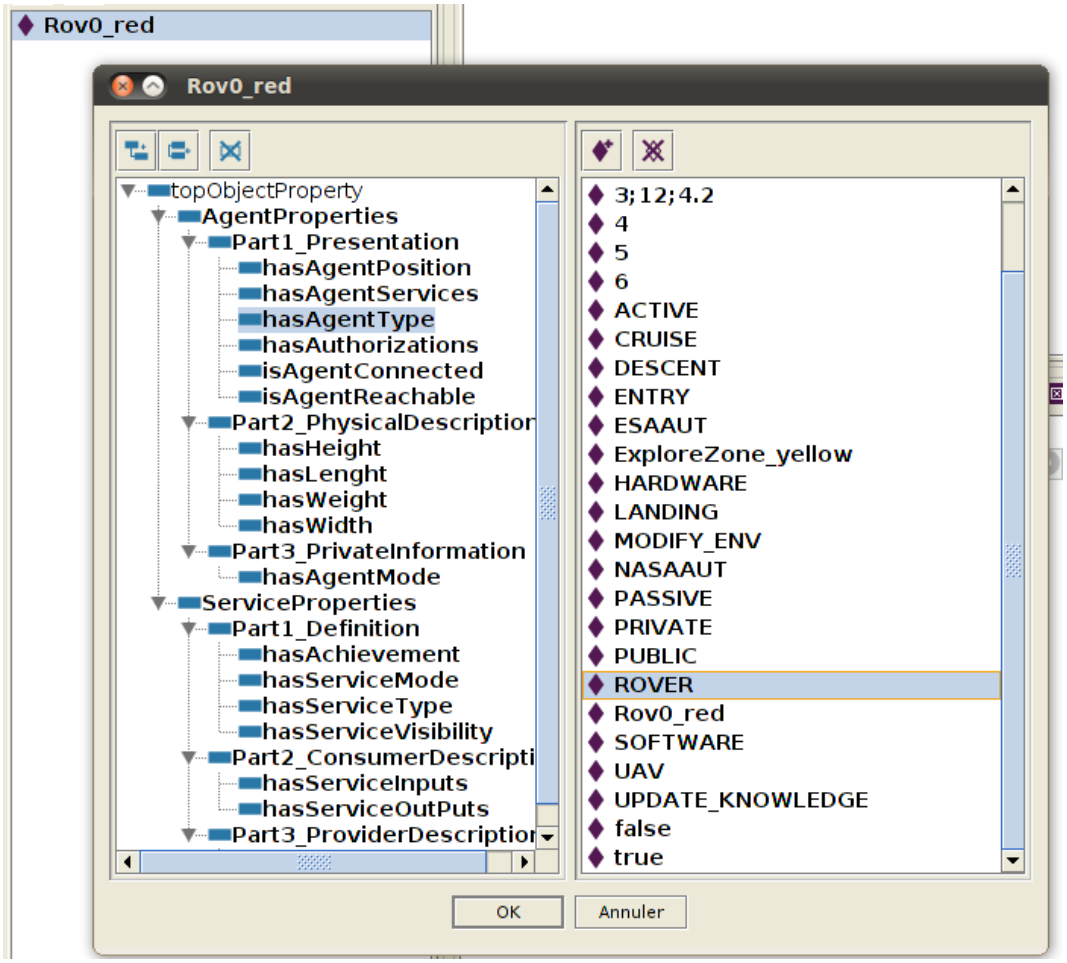

FIGURE 6.1 – Définition du type de l'engin "Rov0" = "ROVER" dans l'ontologie, via l'outil Protégé

On procède ainsi de suite pour toutes les propriétés des engins et de leur services disponibles. La figure 6.2 montre une vue générale de différentes propriétés ainsi définies pour l'engin "Rov0".

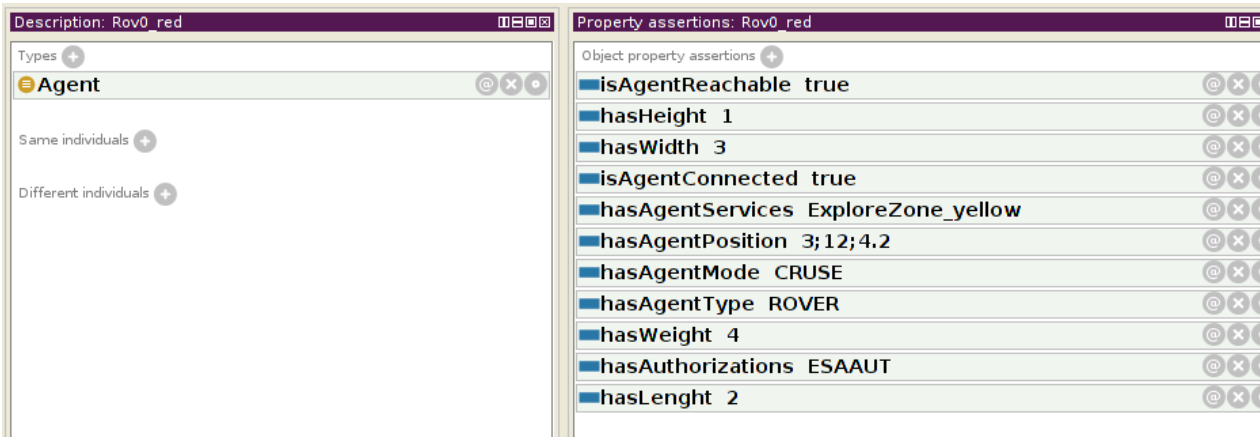

FIGURE 6.2 – Vue de propriétés de l'engin "Rov0", représentées dans une ontologie éditée avec l'outil Protégé

A la fin du travail de spécification, il est possible d'enregistrer *l'ontologie mission* ainsi définie au format OWL-XML. Ce format de fichier génère un fichier XML qui liste toutes les entités et relations définies dans l'ontologie. Chaque concept y est défini, comme par exemple ici le concept de mode de service :

```
<Declaration>
    <Class IRI="#ServiceMode"/>
</Declaration>
```

Ainsi que la liste des relations comme celles d'entrées d'un service :

```
<Declaration>
    <ObjectProperty IRI="#hasInput"/>
</Declaration>
```

Les instances créées sont listées de la manière suivante :

```
<ClassAssertion>
    <Class IRI="#AgentMode"/>
```

```
    <NamedIndividual IRI="#LANDING"/>
</ClassAssertion>
```

Et enfin nous trouvons dans ce fichier les relations liant des instances entre elles. Comme ici celle qui définit le mode de l'engin "Rov0" à la valeur "CRUISE".

```
<ObjectPropertyAssertion>
    <ObjectProperty IRI="#hasAgentMode"/>
    <NamedIndividual IRI="#Rov0"/>
    <NamedIndividual IRI="#CRUISE"/>
</ObjectPropertyAssertion>
```

Le fichier permet également de sauvegarder des commentaires concernant les concepts ou les relations :

```
<AnnotationAssertion>
    <AnnotationProperty abbreviatedIRI="rdfs:comment"/>
    <IRI>#AgentMode</IRI>
    <Literal datatypeIRI="&rdf;PlainLiteral">Vehicle
       configuration, depending on the mode some Services
       could be available or not.</Literal>
</AnnotationAssertion>
```

6.2.2 Codage de l'ontologie mission

Comme expliqué précédemment, le parseur automatique permettant d'extraite, à partir du du fichier OWL-XML, un modèle embarquable dans le module ISIS, n'a pas été développé.

L'ontologie mission est donc transcrite manuellement sous forme de modèle objet dans du code Java selon les principes présentés dans le chapitre 4. Ce modèle est intégré au module ISIS lors de sa création.

6.3 Scénario 1 - Maintien à jour de la connaissance

6.3.1 Description du scénario

Les engins impliqués dans ce scénario sont trois Rovers identifiés ROVER1, ROVER2 et ROVER3.

Le scénario présente plusieurs situations d'échange et de mise à jour de la connaissance au sein d'un réseau d'engins initialement composé de ROVER1 et de ROVER2 :

- L'insertion d'un nouvel engin ROVER3, avec de nouvelles capacités dans le réseau, du fait de son arrivée dans la zone où opèrent ROVER1 et ROVER2.

- La mise à jour de la connaissance suite à la modification des services proposés par un engin.

- Le départ d'un engin ROVER1 du réseau, du fait de l'exécution de sa propre mission.

Le schéma 6.3 représente l'organisation de ce scénario.

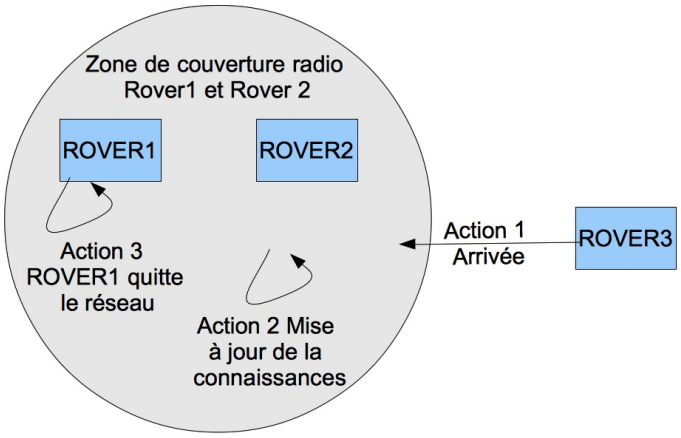

FIGURE 6.3 – Organisation du scénario 1

Dans ce scénario il n'y a pas d'utilisation des services, chaque rover ne possède qu'un service disponible : "ExploreZone", pour illustrer leurs échanges. Les mécanismes d'échange et de mise à jour de la connaissance sont automatiquement gérés par les composant *KnowledgeManager* et *CommunicationManager* du module ISIS. Ils ne nécessitent aucune action particulière de l'architecture de contrôle (planification, exécution de service...).

Integration de ROVER3

Au départ seuls les deux engins ROVER1 et ROVER2 sont à portée de communication l'un de l'autre. A l'initialisation de la simulation, l'architecture de contrôle de l'engin ROVER1 donne l'ordre d'envoyer un message de présentation (par exemple parce qu'il arrive dans une zone où il est sensé retrouver ROVER2). Les deux rovers échangent alors mutuellement les informations concernant leurs descriptions et la liste des services qu'ils peuvent fournir. Voir capture d'écran 6.4.

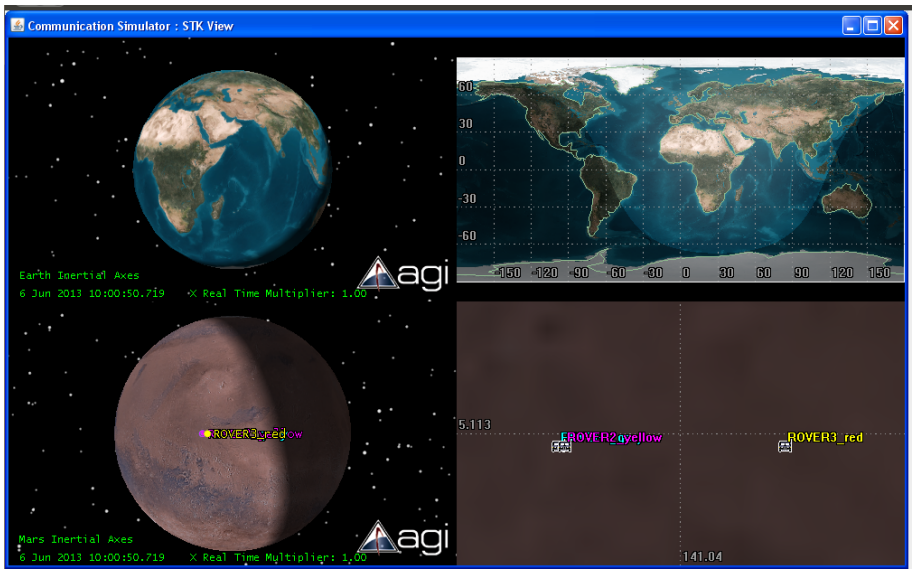

FIGURE 6.4 – Etape 1 du scénario 1 : Situation initiale

Puis l'engin ROVER3 lors de déplacements lié à sa mission, arrive à portée de communication des deux autres engins et son mécanisme de gestion de mission, simulant une demande du contrôle sol, décide d'envoyer un message de présentation. Ne le connaissant pas ROVER1 et ROVER2 enregistrent ce nouvel engin et renvoient à leur tour un message de présentation à l'attention de ROVER3.

Voir capture d'écran 6.5.

A ce stade de la simulation tous les engins se connaissent, avec leurs services respectifs.

Modification des services offerts par ROVER2

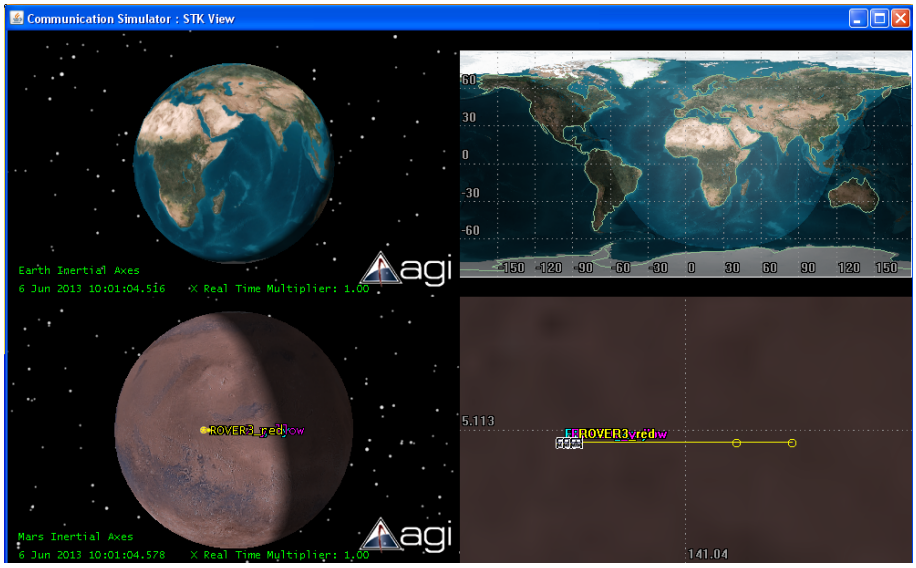

FIGURE 6.5 – Etape 2 du scénario 1 : Arrivée du Rover3

Dans la seconde partie de la simulation une requête est envoyée au module ISIS du ROVER2, via le script de scénario, pour ajouter le service "TakePicture" à la liste de services qu'il propose (simulant l'activation matérielle d'un nouveau service par exemple, ou la mise à disposition d'un service jusque là réservé à sa propre utilisation). Le *KnowledgeManager* du module ISIS enregistre cette modification puis le *KnowledgeManager* envoie un message sur le réseau pour diffuser l'information de mise à jour. Les engins ROVER1 et ROVER3 mettent alors également à jour leur connaissance sur l'engin ROVER2 en ajoutant le nouveau service à la liste des services disponibles et connus pour ROVER2. La modification est donc maintenant propagée à l'ensemble des engins concernés.

Puis l'architecture locale de l'engin ROVER2 émet une requête ISIS, cette fois pour demander la suppression du service "TakePicture" de la liste des services disponibles (simulation de la panne d'un capteur). Le module ISIS génère donc les messages externes nécessaires pour que les engins ROVER1 et ROVER3 mettent à jour leur connaissance sur ROVER2 et ainsi cette modification de la connaissance est à nouveau propagée dans leur réseau.

Départ de ROVER1

Dans la troisième partie de la simulation, une requête par exemple issue du contrôle sol, demande à l'engin ROVER1 de quitter le réseau pour prendre en

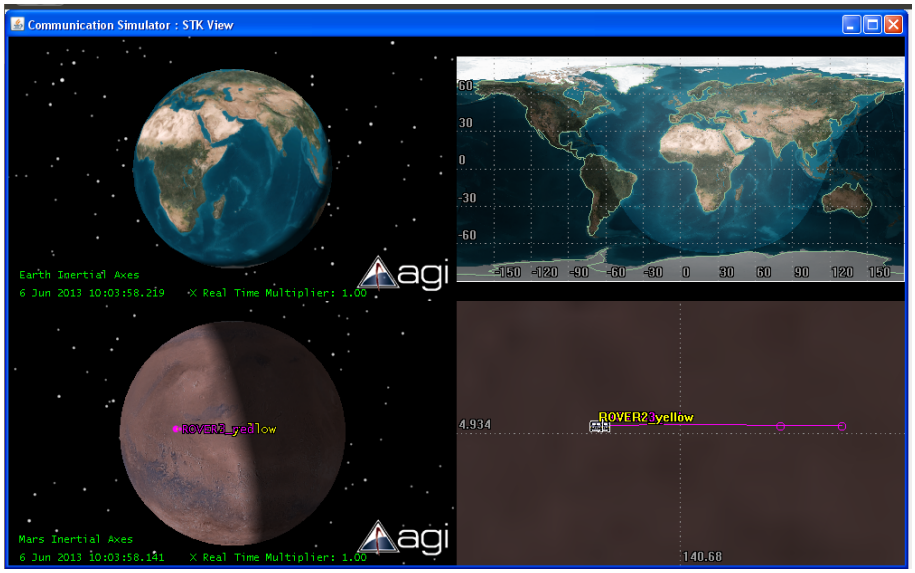

FIGURE 6.6 – Etape 3 du scénario 1

compte le fait qu'il démarre une mission privée et qu'il n'est plus disponible pour les autres engins. Avant de quitter le réseau, ROVER1 notifie ses voisins de son retrait et ceux-ci le retirent alors de leur base de connaissance. Voir capture d'écran 6.6.

6.3.2 Description de la simulation et des résultats

Une première étape de validation a consisté à tester les mécanismes d'interactions nécessaire à l'exécution de ce scénario sans contrainte sur les communications. Le scénario s'exécute nominalement et dès le début les trois engins ont pu entrer en communication et échanger leur connaissance.

Il nécessite au total l'envoi de 15 messages entre les engins (soit un volume cumulé d'information de 61 Ko). Le plus petit message échangé fait 267 octet et concerne la mise à jour de la position connue de l'engin ROVER3. Le message le plus gros fait 821 octet et concerne l'information de description du nouveau service mis à disposition en cours de simulation par ROVER2. 103 requêtes internes ont été émises au total, tous engins confondus.

La connaissance est bien maintenue à jour et les engins sont notifiés de toutes les modifications en quelques secondes seulement par les mécanismes d'interactions implémentés dans les modules ISIS.

La seconde étape d'évaluation des mécanismes d'interactions tient compte

des contraintes de visibilités des engins et de la portée de leurs équipements de communication, via l'utilisation du simulateur de communications.

Dans ces conditions la visibilité entre les équipements de communication est calculée en fonction de la position des engins (calcul de la distance euclidienne) et du type des antennes simulées. Selon la distance, le temps de transmission est également calculé et la réception du message décalée d'autant que nécessaire.

Dans ce scénario, où tous les engins sont proches, les délais de communications sont négligeables. La seule différence avec la simulation précédente est qu'au début de la simulation seuls ROVER1 et ROVER2 peuvent communiquer entre eux. Le ROVER3 est hors de portée de communication. Il ne peut intégrer le réseau qu'une fois suffisamment proche de ROVER1 et ROVER2.

Une troisième étape de simulation à consister à prendre en compte les altérations et les pertes possibles de messages . Dans ce mode ci de simulation, les messages ont une probabilité :

- d'altération de leur contenu de 10^{-4} (soit un caractère changé sur 10000),
- qu'un message soit totalement perdu de 10^{-1} (soit un message sur 10),
- qu'un message partiellement coupé de 10^{-1} (soit un message sur 10).

Lors des simulations, courtes et mettant en jeu un nombre de messages de l'ordre des dizaines, la perte ou l'altération ne concerne en moyenne que de 0 à 3 messages.

Les conséquences de la perte ou de l'altération d'un message sont identiques : le message est détecté comme invalide par le *CommunicationManager* et ne peut pas être exploité. En effet il n'a pas été implémenté de mécanisme de réparation des messages et un message reçu altéré, qui ne peut pas être lu, est donc ignoré. Un risque, faible, existe pour que l'altération d'un message ne rende pas celui-ci illisible, mais change uniquement son contenu (modification de la valeur de certain champs). Ce phénomène n'a pas été observé en simulation, mais il entraînerait l'enregistrement d'une information fausse dans la base de donnée de l'engin receveur.

Selon le type de message qui est altéré ou perdu, les conséquences sont différentes :

- Message de présentation : cela a peu de conséquences dans ce scénario. Dans ce scénario particulier, l'engin qui n'a pas eu l'information finit par recevoir un message de l'engin qu'il n'a pas encore enregistré dans sa base de connaissance (car celui ci va répondre à un autre agent ou envoyer un message de modification de sa connaissance). Il envoie alors spontanément son propre message de présentation, selons le *protocole*

de présentation automatique présenté dans la section 4.5.4.2, et les deux engins partagent leur connaissance.
Dans certaines situations, comme par exemple, s'il n'y a que deux engins, la mise en place du *protocole de présentation automatique* ne peut pas avoir lieu ci les messages de ce type sont perdus. Ce serait alors à l'architecture locale ou au contrôle au sol de décider de ré-émettre un message tant que l'engin n'a pas reçu de confirmation par exemple.

– Message de mise à jour de la connaissance (mise à jour d'un service, départ d'un engin...) : L'information est alors perdue. Il n'y a pas moyen de s'en rendre compte tant que l'engin émetteur du message perdu ne fournit pas une nouvelle mise en jour de cette connaissance (mise à jour d'un service par exemple).

6.4 Scénario 2 - Utilisation d'un service de traitement déporté d'une tâche

6.4.1 Description du scénario

L'objectif de ce scénario est d'illustrer les mécanismes liés à l'exécution déportée d'une tâche, par un engin, pour un autre. Cela peut être utile pour partager des ressources de calcul, de stockage, de communication, ou pour déléguer l'exécution d'une tâche impossible à réaliser par l'un des engins.

Les engins impliqués dans ce scénario sont un rover, nommé ROVER1 et un orbiteur nommé ORBITER1. Le rover possède un service *ExploreZone* qui nécessite en entrée les coordonnées d'une zone à explorer et qui fournit en sortie une carte de cette zone. L'orbiteur possède un service *ComputePathTo* qui prend en entrée une position de départ et une position d'arrivée et peut ensuite calculer et renvoyer un itinéraire pour relier ces deux points. Un argument supplémentaire du service pourrait être le critère de calcul du chemin : le plus rapide, le plus court, le plus sûr... L'orbiteur émet périodiquement des messages de présentation afin de pouvoir entrer en communication avec tous les engins qu'il survole.

Le schéma 6.7 représente l'organisation de ce scénario.

Le scénario de simulation commence alors que l'engin ROVER2 est en cours d'exécution d'une mission d'exploration une zone donnée (objectif fixé par le contrôle sol ou résultant d'une demande d'un autre engin).

Voir capture d'écran 6.8.

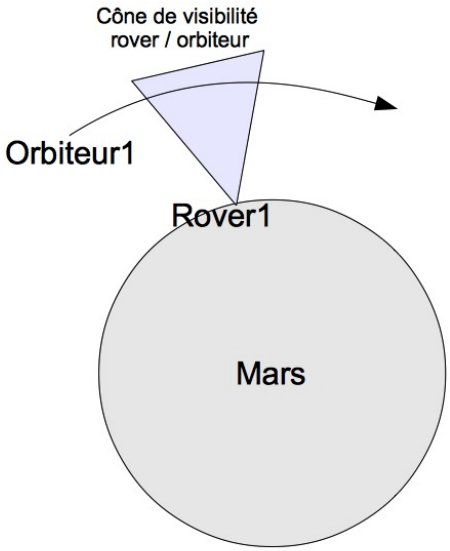

FIGURE 6.7 – Organisation du scénario 2

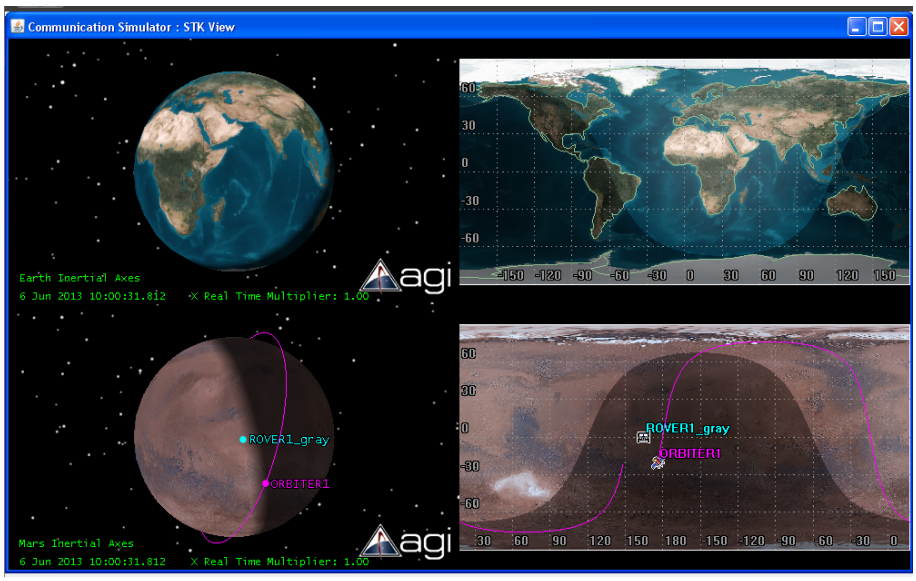

FIGURE 6.8 – Vue STK du scénario 2

A un moment donné, le contrôle au sol envoie une demande de changement de zone à explorer au rover. La nouvelle zone est éloignée et l'engin n'a pas les ressources de calcul ni la connaissance nécessaire pour calculer un itinéraire jusqu'à celle-ci.

Le rover cherche donc à faire réaliser la tâche de calcul d'itinéraire par un autre engin. Si un tel service est disponible parmi la liste des services proposés par ses voisins connus, le rover l'invoque, sinon il attend que celui-ci le soit.

Quand l'orbiteur arrive en visibilité du rover, après quelques dizaines de secondes de simulation, les deux engins échangent leur connaissance et le rover découvre le service *ComputePathTo* qui peut lui être utile. Il l'invoque alors auprès de l'orbiteur, en fournissant sa position courante et son objectif de position : point d'entrée dans la nouvelle zone à explorer.

L'orbiteur commence alors à calculer le meilleur chemin en utilisant les cartes de la planète à sa disposition. Après plusieurs dizaines de minutes (le temps de calcul est un paramètre de la simulation) l'orbiteur est en mesure de donner un résultat. Il attend de réaliser une révolution complète afin d'être de nouveau en visibilité du rover, pour lui envoyer le chemin calculé.

6.4.2 Description de la simulation et des résultats

Dans une première simulation, sans contrainte sur la portée ni la qualité des communications, l'exécution du scénario ne présente pas de problème. Dès le début de la simulation le rover et l'orbiteur se découvrent et échangent leurs connaissance.

Quand le rover cherche un service lui permettant de calculer un itinéraire, le service *ComputePathTo* proposé par l'orbiteur est disponible et il demande son exécution à l'orbiteur. Pour réaliser ce scénario les deux engins s'échangent 14 messages (soit un volume d'information cumulé de 57 Ko) et un total de 59 interactions internes sont générées.

En tenant compte des contraintes de visibilité des engins et de la portée de leurs équipements de communication, au départ de la simulation l'orbiteur n'est alors pas encore en visibilité et le rover attend.

Quand l'orbiteur arrive en visibilité la situation évolue et les échanges de connaissance ainsi que l'invocation du service *ComputePathTo* deviennent possibles. Le rover se met en attente de la réception du résultat du calcul (ici simulé avec une seule révolution de l'orbiteur autour de la planète).

Dans cette simulation les deux engins s'échangent 22 messages (soit un volume cumulé d'information de 90 Ko) et un total de 76 interactions internes

sont générées. Ces nombres sont plus importants que ceux de la simulation sans contrainte sur les communications car :

- L'orbiteur envoie périodiquement des messages de présentation, même quand aucun engin n'est à portée.
- Le rover génère périodiquement des requêtes internes vers sa base de connaissance pour vérifier si un service approprié est disponible ou non.

Dans un troisième temps, la simulation tient compte des altérations et pertes possibles des messages.

La perte ou la corruption d'un des messages de présentation ne remet pas en question l'exécution de la mission car l'orbiteur émet périodiquement des messages de présentations.

Par contre si c'est le message d'invocation du service, ou bien celui de retour du résultat qui est concerné, la mission ne sera pas remplie car : soit l'orbiteur ne recevra pas la demande, soit le rover ne recevra pas le résultat de l'exécution.

Une solution pour pallier ce problème serait de mettre en place au niveau de l'architecture locale de contrôle les mécanismes nécessaire à la vérification de la bonne exécution d'un service, via l'utilisation des messages du types *serviceMonitoring*. Une autre solution pourrait être d'intégrer ces mécanismes directement aux protocoles d'interactions du module ISIS. Les modules ISIS pourraient accuser la bonne réception et transmission aux architectures locales de la demande de service, mais il ne pourrait surveiller directement l'exécution des services.

Au vu des limitations rencontrées dans les simulations prenant en compte les contraintes sur la visibilité des équipements de communication, l'altération et la perte des messages, il serait donc intéressant pour améliorer le fonctionnement du système de mettre en place :

- Un mécanisme de réservation des services, pour prévoir par exemple quand l'orbiteur est en visibilité et ne pas générer de requêtes et de messages périodiques inutilement.
- Un mécanisme automatique d'accusé de réception à l'invocation ou à la réservation d'un service, pour être sûr que celui-ci à bien été démarré ou réservé.
- Un mécanisme de surveillance périodique et continu pour vérifier que le service est bien en cours d'exécution ou pour se rendre compte qu'il est terminé mais que le message de résultat n'a pas été reçu, et dans ce cas être capable de le re-demander. Mais cela générerait une augmentation du nombre de messages et de requêtes utilisés.

6.5 Scénario 3 - Utilisation coordonnée de plusieurs services

6.5.1 Description du scénario

L'objectif de ce scénario est de montrer la robustesse d'un système d'exploration multi-engins, lors de la réalisation d'une mission, malgré des événements imprévisibles, en mettant en avant le fait qu'une coopération opportuniste peut permettre de résoudre des situations bloquantes.

Ce scénario illustre la mise en place automatique d'un protocole d'interactions pour l'utilisation coordonnée de plusieurs services. Le contrôle au sol reste présent mais n'intervient que de manière très ponctuelle et uniquement pour envoyer des ordres de haut niveau à un seul des engins.

Voir capture d'écran 6.9 pour une vision initiale du système d'exploration.

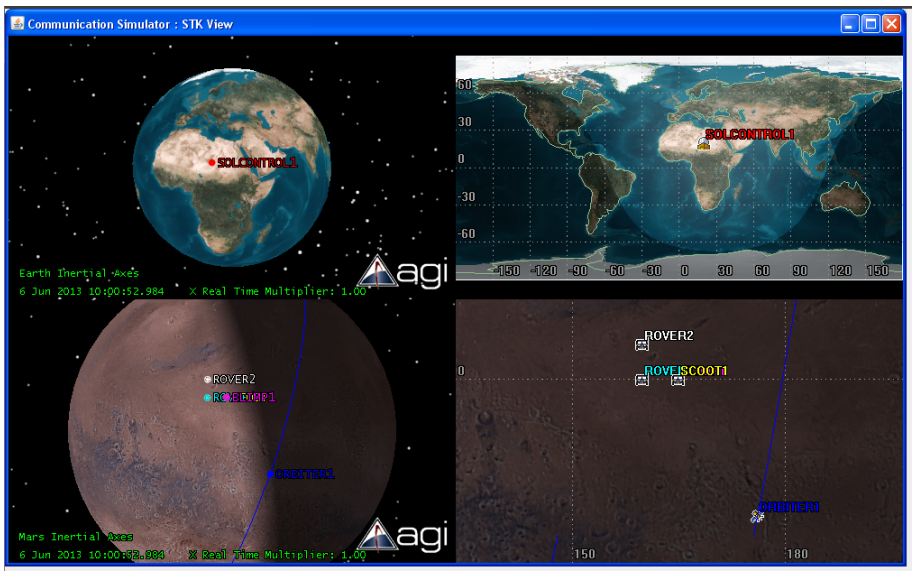

FIGURE 6.9 – Vue initiale du système d'exploration du scénario 3

Les engins impliqués dans ce scénario sont :

– Un rover nommé ROVER1 proposant les services *ExploreZone* et *ComputeTrajectory*. Le service *ComputeTrajectory* permet de générer, à partir d'une carte et des coordonnées d'un point de départ et d'un point d'arrivée, un ordre de déplacement, de bas niveau (déplacement ou mouvement simple, de faible distance), permettant à un engin du type

"robot marcheur", tel que SCOOT1, d'effectuer un mouvement. Avant chaque exécution du service le rover a besoin d'une carte avec la position précise de l'engin à diriger. Ce service est dédié au contrôle d'engins simples, comme le robot marcheur SCOOT1 qui n'embarquent pas de carte et ne peuvent pas se localiser.
- Un robot marcheur léger, nommé SCOOT1, embarqué et déployable par le ROVER1. Celui-ci propose le service *Move* qui prend en entrée des ordres de déplacement de bas niveaux (avancer / reculer, pendant X secondes, sur une des 8 directions discrétisées possibles), tels que ceux générés par le service *CouputeTrajectory* du ROVER1. Après l'exécution d'un ordre de déplacement le robot marcheur s'arrête et attend une nouvelle instruction.
- Un ballon dirigeable nommé BLIMP1, captif du ROVER1, qui est capable via l'utilisation du service *getMap* d'envoyer une carte de la région qu'il survole et de détecter et de localiser les engins qui s'y trouvent.
- Un orbiteur nommé ORBITER1, qui fournit un service de relais de données, *relayData*, vers les engins en visibilité et donc potentiellement vers le contrôle au sol.
- Un rover nommé ROVER2, qui fournit le service *getMap* qui prend en entrée les coordonnées d'une zone au sol et qui renvoie la carte de cette zone comprenant des informations sur le terrain dont les positions des engins détectés. Cette fonction peut s'appuyer sur un mat télescopique.
- Le contrôle au sol ne fournit pas de service ici, mais peut réagir à la réception d'un message d'alerte provenant d'un des engins.

Au départ ROVER1 reçoit l'ordre d'explorer une zone accidentée, il emporte à son bord le robot marcheur SCOOT1 et le ballon BLIMP1 est déjà déployé, compte tenu de la vitesse relativement lente du rover, le fait que le ballon vole au dessus de lui est envisageable. Les engins se connaissent déjà et ont échangé leurs connaissances respectives.

Durant l'exploration de la zone, le rover est stoppé par un terrain trop difficile d'accès pour lui (faille, éboulis, cratère...).

Le rover décide alors de déployer le robot marcheur, SCOOT1, pour qu'il poursuive l'exploration à sa place sur cette sous-zone (pour réaliser par exemple de la collecte d'échantillons ou des prises de vues). Cette décision pourrait résulter de la fonction planification embarquée sur ROVER1.

Il calcule la première trajectoire et demande l'exécution du service *Move* à l'engin SCOOT1, pour l'exécution du premier déplacement, qui se déplace alors à la position souhaitée. Voir capture d'écran 6.10.

Le rover invoque alors les services *getMap* du ballon BLIMP1, afin de

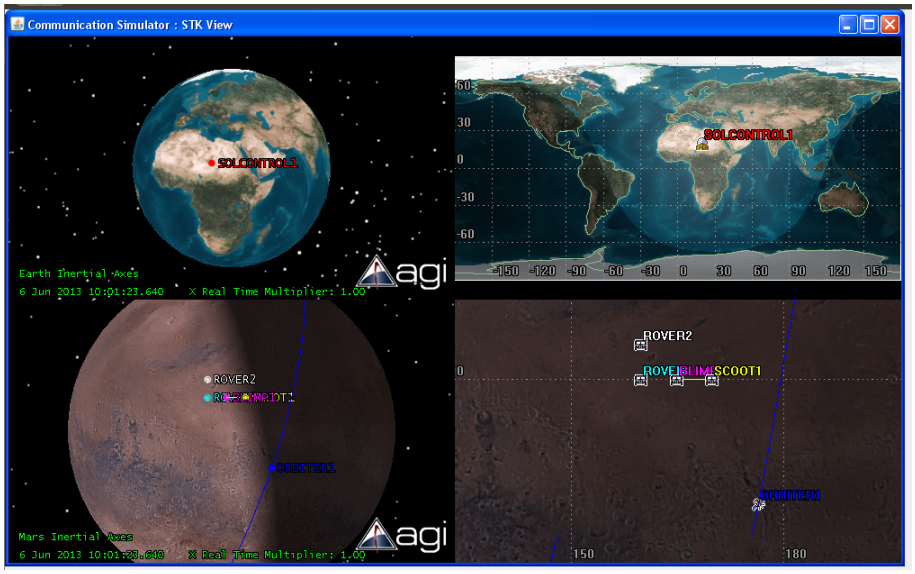

FIGURE 6.10 – Etape 1 du scénario 3

connaître l'environnement du robot marcheur et de pouvoir calculer les consignes de trajectoires suivantes.

Il s'ensuit la mise en place d'une "boucle de contrôle" du système multi-engin où le rover demande périodiquement les informations de position et d'environnement du robot marcheur au ballon, afin de calculer les nouvelles trajectoires d'exploration.

Après trois itérations de cette boucle le ballon BLIMP1 se met à renvoyer un message d'erreur, indiquant le service *getMap* est non fonctionnel.

Au bout de deux messages d'erreur consécutif, le système de gestion de suivi du service considère le service comme non fonctionnel. Le rover décide alors d'envoyer un message contenant une description de la situation et des derniers messages d'erreurs au contrôle au sol.
L'autre conséquence est que ROVER1 n'a plus d'information de position de SCOOT1 est ne peut plus ni poursuivre la mission, ni le récupérer.

Pour avoir une connexion immédiate avec le contrôle, celui-ci n'étant pas directement en visibilité, il utilise le service *relayData* de l'orbiteur.

Le contrôle au sol décide alors, pour débloquer la situation, d'envoyer un

message au ROVER2 pour mettre en pause sa mission courante, lui demander de se déplacer vers la zone où est ROVER1 et lui demander de se mettre à disposition de ses voisins une fois sur zone. Il dispose d'un service *getMap* qui devient disponible pour les autres engins.

Voir capture d'écran 6.11.

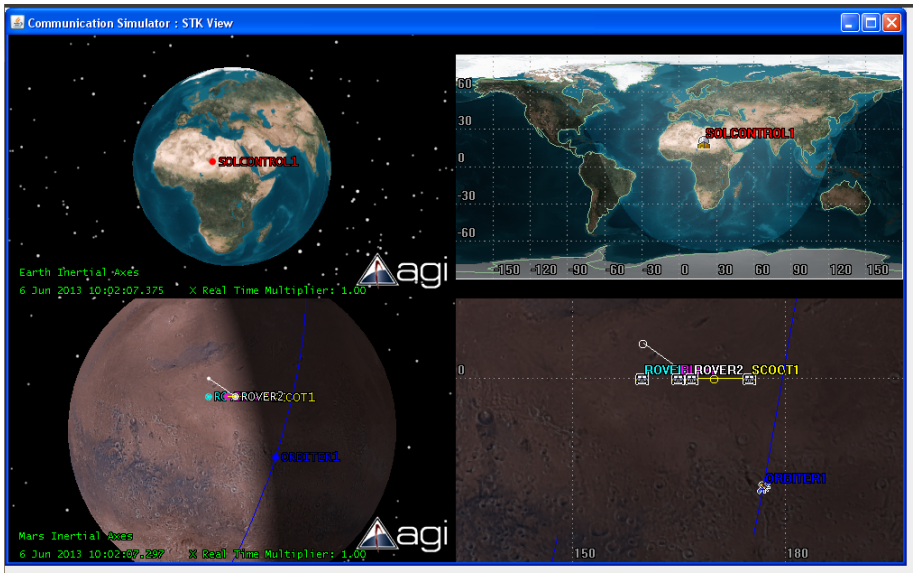

FIGURE 6.11 – Etape 2 du scénario 3

L'information de la disponibilité du nouveau service est ensuite propagée automatiquement sur le réseau d'engins par les modules ISIS.

Le ROVER1 peut alors demander la description complète du service *getMap* au ROVER2 et vérifier qu'elle est compatible avec celle précédemment fournie par BLIMP1. Il est alors capable d'utiliser de nouveau le service *getMap*.

Cela lui permet, après trois nouvelles itérations de la boucle d'acquisition de la carte / contrôle de l'engin SCOOT1, de ramener ce dernier à proximité de ROVER1 pour sa récupération.

Le ROVER1 prévient le contrôle au sol du retour de SCOOT1, pour que celui-ci puisse de nouveau désactiver le service *getMap* du ROVER2 et le renvoyer à sa mission initiale.

Voir capture d'écran 6.12.

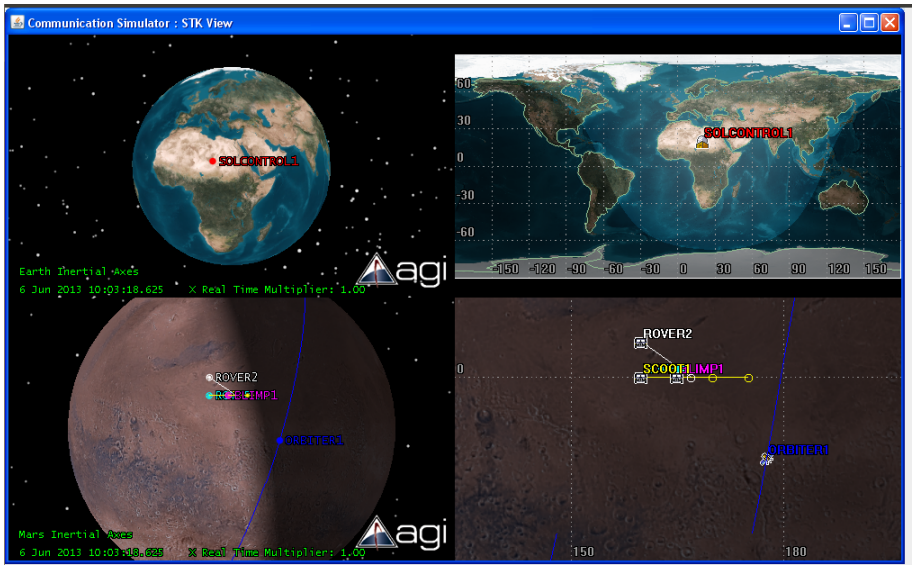

FIGURE 6.12 – Etape 3 du scénario 3

6.5.2 Description de la simulation et des résultats

Cette simulation a été réalisée sans le simulateur de communication. Les messages sont donc transmis de manière instantanée et sans erreur entre les engins.

La simulation nécessite l'échange de 92 messages entre les engins (soit un volume de 377Ko, hors volume des cartes d'environnement échangées qui ne sont pas modélisées) et un total de 1311 requêtes internes sont générées :

- 214 par BLIMP1
- 213 par ORBITOR1
- 284 par ROVER1
- 193 par ROVER2
- 206 par SCOOT1
- 201 par SOLCONTROL1

La mise en place de la boucle de contrôle est initiée par le rover qui joue le rôle de coordinateur des services car c'est lui qui supervise l'exécution des services du robot marcheur et du ballon.

Le rover attend que chaque engin ait terminé l'exécution des services demandés pour passer à l'étape suivante.

Le diagramme 6.13 présente le nombre de messages échangés entre les engins au cours du temps (en seconde) et le digramme 6.14 présente le nombre

total de requêtes internes aux engins générées.

Les premières secondes, où un nombre important de messages et de requêtes sont générés, correspondent à la phase d'initialisation de la simulation, durant laquelle les engins se présentent et échangent leur connaissance.

Au temps t=18s au temps 75s se déroule la première phase d'exploration conjointe : ROVER1, SCOOT1, BLIMP1, jusqu'à l'erreur sur l'utilisation du service du BLIMP1. Puis de t=75s à t=95s se déroule la phase d'attente de la solution provenant du contrôle au sol. De t=95s à t=138s la phase de secours par ROVER2 et du retour de SCOOT1.

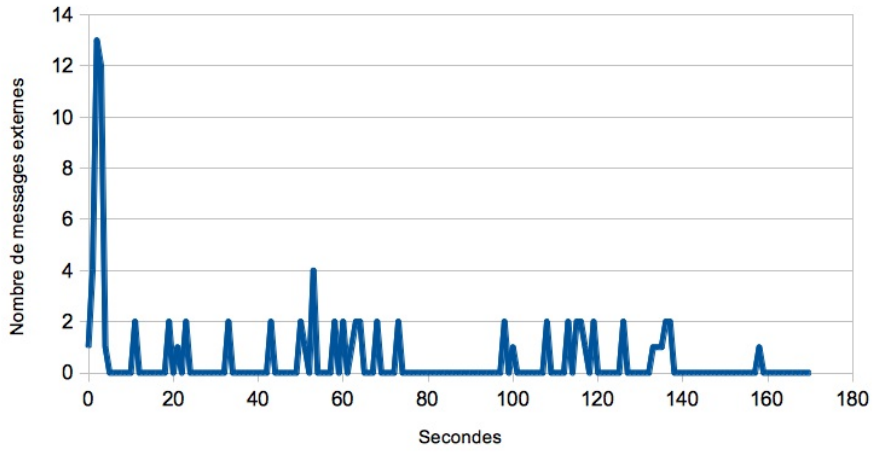

FIGURE 6.13 – Nombre de messages externes échangés entre les engins au cours du temps de la simulation.

Sans simulation d'erreurs et de délais dans les communications, la simulation se déroule nominalement. Avec une simulation d'erreur sur les communication, si un seul message avait été perdu, toute la mission aurait été bloquée, parce que nous n'avons pas implémenté de planificateur pour gérer les décisions des engins.

Dans une implémentation plus complète, ce pourrait être au planificateur du rover de gérer les situations où un engin, après un certain délai, n'a toujours pas complété l'exécution d'un service et de réagir en conséquence, en renvoyant une demande ou en cherchant une autre solution pour accomplir la mission (demande de l'intervention d'une autre engin, calcul d'une autre trajectoire d'exploration...). Il pourrait donc décider seul d'abandonner l'exécution de la mission, de la recommencer ou bien de demander l'assistance du contrôle mission au sol.

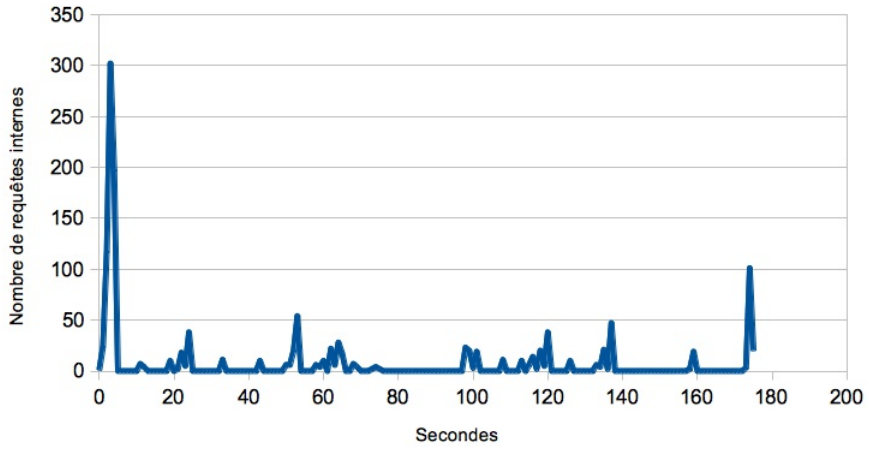

FIGURE 6.14 – Nombre de requêtes internes générées par les engins au cours du temps de la simulation.

6.6 Discussion des expérimentations

Ces simulations ont permis de valider sur des scénarios simples mais représentatifs, l'approche globale et le fonctionnement attendu concernant les poins suivants :

- La modélisation du contexte global d'exploration robotique ainsi que que sa modélisation dans une ontologie, où il est possible de représenter tant la connaissance des engins sur eux même (état, équipement, spécifications physiques...) que celle de leurs capacités (représentation des services).
- L'organisation et l'utilisation générale de la connaissance, depuis la modélisation dans une ontologie commune, garantissant l'interopérabilité, jusqu'à l'implémentation à bord d'un module même si cette dernière phase est manuelle.
- L'intégration du module ISIS en association avec les architectures locales de contrôle à bord des engins.
- Un premier ensemble de mécanismes d'interactions directes et automatiques entre engins voisins. Ces interactions directes permettent de limiter l'intervention du contrôle mission à l'envoi d'ordres de haut niveau et à la résolution de situations de crise.
- Des protocoles d'échange de l'information associés. Cela permet de partager une connaissance commune entre les engins. Et de disposer ainsi des informations nécessaires pour les engins à l'exécution de leur mission.

Il est à noter que le nombre de requêtes internes à l'architecture logicielle d'un engin, générées par l'utilisation du module ISIS, est important. Sur des architectures logicielle d'engins aux capacités de calcul limités, ce nombre important de requêtes internes pourrait être problématique. Il serait intéressant de les optimiser et de réduire leur nombre. Par exemple en ne demandant pas à chaque message reçu si l'on connaît l'engin émetteur, ou en n'envoyant pas toujours de réponses / accusés de réception aux requêtes.

Bien que les trois scénarios étudiés soient loin de représenter toute la diversité des missions robotiques d'exploration possibles, ils constituent une première étape dans les recherches sur ces problématiques et ont permis d'initier des expérimentations.

Chapitre 7

Conclusion et perspectives

7.1 Rappel des objectifs

L'objectif poursuivi est de faciliter la conduite de missions d'exploration planétaire robotique impliquant plusieurs engins, en autorisant des interactions directes entre engins. Une forte autonomie est donc laissée à ces engins dans la conduite de ces interactions locales, nécessitant un minimum d'interactions impliquant le contrôle mission humain.

Globalement, cela devrait permettre d'accroître l'autonomie du système d'exploration et de le rendre plus robuste et plus réactif.

Il s'agit en particulier d'étudier comment il est possible de caractériser les compétences d'un engin et comment les engins peuvent se partager ces informations entre eux, pour ensuite se coordonner et atteindre des objectifs mission, ou bien en collaborant de manière opportuniste.

7.2 Synthèse des travaux

A partir du constat que pour pouvoir communiquer et collaborer, les engins devaient tous partager une représentation commune de la connaissance, il a été montré qu'il était possible de caractériser les compétences et possibilités d'actions des engins sous forme de services. L'ensemble des services proposés par un engin forment un manuel utilisateur à destination des autres engins.

Pour organiser cette connaissance et afin de limiter les échanges entre engins pour chaque phase d'une mission, une modélisation structurée et progressive de la connaissance a été utilisée.

Pour assurer l'interopérabilité entre les différents engins, la connaissance embarquée dans chaque module est dérivée d'une ontologie générale spécifique au domaine de l'exploration planétaire. Cette ontologie permet de décrire l'environnement dans lequel évoluent les engins, mais surtout les capacités opérationnelles de ces engins. Une méthode a été proposée permettant de dériver, pour un nouvel engin et à partir de l'ontologie générale, la connaissance à embarquer ainsi que différentes fonctions de base associées.

Il a été proposé que pour échanger la connaissance de l'environnement des engins et l'état de leur services disponibles, un module en charge de gérer les interactions locales ainsi que la représentation de la connaissance devait être associé aux architectures logicielles embarquées sur les engins.

Une spécification de ce module, nommé ISIS, a été proposée, qui spécifie ses constituants et les protocoles de fonctionnement (interactions internes aux architectures logicielles locales de chaque engin et interactions externes, via des messages, entre différents engins).

En final un environnement de simulation a été développé. Cet outil a permis de mettre en oeuvre les concepts imaginés et de réaliser des tests et évaluation sur des scénarios simples et représentatifs, de situations rencontrées dans des systèmes multi-engins pour l'exploration planétaire.

Les points intéressants de cette approche sont :

- L'ontologie proposée permet de modéliser les engins des missions d'exploration actuelles ainsi que ceux de missions à venir.
- Les mécanismes d'interactions locales, et de gestion et partage de la connaissance, gérés par le module ISIS, permettent bien de partager et de maintenir à jour la connaissance des engins au sein du réseau. Les informations échangées peuvent donc être utilisées par un module de gestion de mission en ligne (planification).
- Bien que le contexte retenu ici soit celui de l'exploration planétaire (avec ses contraintes spécifiques), l'approche proposée peut être appliquée à d'autres systèmes multi-engins présentant des contraintes d'autonomie similaires (e.g. mission d'exploration et de secours, militaire ou civile). Dans le cas où ces travaux seraient repris pour les appliquer à des missions d'engins robotiques terrestres, les points les plus intéressants à reprendre seraient la représentation des capacités des engins sous forme de services ainsi que les protocoles automatiques de partage de la connaissance intégrés dans les modules ISIS.

Parmi les limitations de l'approche, on peut citer :
- L'ontologie générale de missions d'exploration ne peut pas être embarquée directement à bord des engins, la gestion des ontologies nécessite des ressources informatiques importantes, qui ne sont pas adaptées aux systèmes embarqués. Le passage automatisé d'une ontologie à une représentation "embarquable" n'a pas été automatisé.
- L'évolution de l'ontologie générique et des engins au cours du temps posent des problèmes relatifs à la garantie de l'interopérabilité entre engins présentant des versions différentes de modèles de la connaissance. Cette problématique n'a pas été abordée lors ce travail.
- Le modèle de connaissance fait abstraction de l'environnement physique et temporel. Et d'autres limitations liées à l'implémentation actuelle font que l'implémentation possible sur des engins réels est encore assez éloignée.

7.3 Perspectives

A court et moyen terme

Concernant la modélisation de la connaissance dans une ontologie :
- Il serait intéressant d'enrichir celle-ci avec les concepts décrivant l'environnement physique, les éléments de cartographie et l'environnement et les contraintes temporelles.
- L'outil de modélisation des ontologies utilisé, Protégé, n'est pas adapté à l'instanciation des ontologies telle que nous l'utilisons. Si cet outil est pratique pour décrire l'ontologie générique avec les concepts et leurs relations modélisant la connaissance générale, il ne l'est pas pour instancier la connaissance d'une mission particulière. Un autre outil plus adapté sera nécessaire.

Concernant l'implémentation des concepts proposés :
- Il serait utile de développer un outil informatique pour automatiser le passage de la connaissance de l'ontologie mission, à l'implémentation de cette connaissance dans le module ISIS.
- Le module ISIS pourrait être enrichi, en y transférant des protocoles de gestion des services (accusés de réception, monitoring des services, réservation) et de surveillance et de sécurité des engins (watchdog), actuellement supposés gérés par les architectures locales de contrôles dans notre implémentation.

Concernant l'augmentation du réalisme des simulations :

- Les architectures de contrôle en elles-même pourront être raffinées pour intégrer la modélisation et la gestion d'équipements de bord (capteurs, actionneurs, charge scientifique...) et utiliser une structure hiérarchique complète.
- Un algorithme de planification et de prise de décision pourrait être ajouté aux architectures locales de contrôle des engins. Afin de pouvoir jouer des scénarios de manière plus autonome, en modélisant la recherche et le choix des services utiles à un engin en fonction de leurs descriptions, leurs réservations puis leurs invocations et leurs utilisations dynamiques.
- Il serait nécessaire, pour poursuivre le travail de validation, d'implémenter les concepts proposés sur des réseaux d'engins réels.

A long terme

La problématique générale où les engins ne partagent pas totalement le même formalisme de représentation de la connaissance (soit parce qu'ils sont d'une génération différente, soit parce qu'ils proviennent de concepteurs différents) doit être abordée. Par exemple les recherches autour des système de médiation ou d'appariement pour réussir à faire du mapping d'ontologies ou de la fusion de base de connaissance pourraient être une piste de réflexion.

Une autre piste intéressante concerne l'apprentissage autonome, de nouveaux concepts par un engin. Est-il possible qu'un engin "apprenne" un nouveau type de carte, ou bien qu'il soit capable de faire une nouvelle distinction entre deux informations auparavant similaires de l'environnement (entre deux types de roches par exemple, alors qu'avant toutes étaient considérées comme un obstacle) ? Et ceci sans devoir mettre à jour complètement son logiciel de bord.

L'objectif final étant d'arriver à définir un standard de représentation et d'utilisation de la connaissance pour les engins des futures missions d'exploration robotiques.

Bibliographie

[1] FIPA Ontology Service Specification. Technical report, Fondation for Intelligent Physical Agent, 2000.

[2] PEA ACTION, LT1 - État de l'art, ONERA / LAAS-CNRS, RTS 1/12778 DCSD. Technical report, ONERA / LAAS-CNRS, 2007.

[3] S Ambroszkiewicz, W Bartyna, M Faderewski, and G Terlikowski. Multirobot system architecture : environment representation and protocols. *Bulletin of the Polish Academy of Sciences : Technical Sciences*, 58(1) :3–13, 2010.

[4] Rodrigo Amorim, Daniela Barreiro Claro, Denivaldo Lopes, Patrick Albers, and Aline Andrade. Improving Web Service Discovery by a Functional and Structural Approach. *2011 IEEE International Conference on Web Services*, pages 411–418, July 2011.

[5] JF and others Bell. (Nearly) Seven Years on Mars : Adventure, Adversity, and Achievements with the NASA Mars Exploration Rovers Spirit and Opportunity. *AGU Fall Meeting Abstracts*, 2010.

[6] Kenneth Baclawski and Artan Simeqi. Toward Ontology-Based Component Composition. In *10th OOPSLA Workshop Behavioral Semantics (OOPSLA 2001*, pages 1–11. Citeseer, 2001.

[7] Tucker Balch and Lynne Parker. *Robot Teams : From Diversity to Polymorphism*. A K Peteres, 2002.

[8] Eric Bensana. Robotique d'exploration : Architecture logicielle embarquée - Contexte de démonstration, Rapport Technique, RT 2/17423, ONERA, The french aerospace lab. *ONERA - DCSD*, 2010.

[9] Eric Bensana. Robotique d'exploration : Architecture logicielle embarquée - Contexte de mission, Rapport Technique, RI 1/17423, ONERA, The french aerospace lab. *ONERA - DCSD*, 2010.

[10] Curtis L. Blais. Application of Coalition Battle Management Language (C-BML) and C-BML Services to Live, Virtual, and Constructive

(LVC) simulation environments. In S. Jain, RR Creasey, J. Himmelspach, KP White, and M. Fu, editors, *2011 Winter Simulation Conference*, number Lvc, pages 2587–2599, 2011.

[11] Jason Bloomberg and Ronald Schmelzer. How do you Build a Service ? Technical report, ZapThink, 2005.

[12] Mike Botts and Alexandre Robin. Open Geospatial Consortium Inc . OpenGIS ® Sensor Model Language (SensorML) Implementation Specification. Technical report, Open Geospatial Consortium Inc, 2007.

[13] James Bruce, Stefan Zickler, Mike Licitra, and Manuela Veloso. CM-Dragons : Dynamic passing and strategy on a champion robot soccer team. In *2008 IEEE International Conference on Robotics and Automation*, pages 4074–4079. IEEE, May 2008.

[14] D. Brugali and P. Scandurra. Component-based robotic engineering (part i)[tutorial]. *Robotics & Automation Magazine, IEEE*, 16(4) :84–96, 2009.

[15] Oana Bucur, Philippe Beaune, and Olivier Boissier. Définition et représentation du contexte pour des agents sensibles au contexte. *Proceedings of the 2nd French-speaking conference on Mobility and uibquity computing - UbiMob '05*, page 13, 2005.

[16] Technical Report of the Consultative Committee for Space Data Systems (CCSDS). Spacecraft Onboard Interface Services. June 2007. Lien direct : http ://public.ccsds.org/publications/archive/850x0g1.pdf

[17] CCSDS. Spacecraft Onboard Interface Services—Subnetwork Device Discovery Service. Technical Report December, Consultative Committee for Space Data Systems, 2009.

[18] B Chaib-Draa, I Jarras, and B Moulin. Systèmes multi-agents : principes généraux et applications. *Agent et systèmes multi-agents*, 2001.

[19] Brahim Chaib-Draa, Imed Jarras, and Bernard Moulin. *Systèmes multi-agents : principes gènèraux et applications*. 2001.

[20] Michael Compton, Cory Henson, Laurent Lefort, Holger Neuhaus, and Amit Sheth. A survey of the semantic specification of sensors. *Proc. Semantic Sensor Networks*, page 17, 2009.

[21] Pascal Deplanques. *Vers le test de l'autonomie des robots : une ontologie de la robotique*. PhD thesis, Université Montpellier II, 1996.

[22] Jurriaan Van Diggelen, Jeffrey M. Bradshaw, Tim Grant, Matthew Johnson, and Mark Neerincx. Policy-Based Design of Human-Machine Collaboration in Manned Space Missions. *2009 Third IEEE International Conference on Space Mission Challenges for Information Technology*, pages 376–383, July 2009.

[23] S. Dubowsky, I. Hunter, and Gregory Chirikjian. Phase I Study of Self-Transforming Robotic Planetary Explorers, Final Report, Reporting Period : 11/98 - 5/99. *NASA Institute for Advanced Concepts*, 2000.

[24] Marc Ehrig and York Sure. Ontology mapping-an integrated approach. *The Semantic Web : Research and Applications*, pages 76–91, 2004.

[25] Equipe Projet Action (ONERA / LAAS-CNRS). Rapport LT1.2 - Etat de l'art relatif aux scénarios aéroterestres V et VI. Technical report, 2010.

[26] K. Erol, J. Hendler, and D. Nau. HTN planning : complexity and expressivity. In *National Conference on Artificial Intelligence (AAAI)*, 1994.

[27] ESA. ESA - ExoMars, http ://www.esa.int/ SPECIALS/ExoMars/index.html, September 2010.

[28] J.M. Evans, E.R. Messina, and J.S. Albus. Knowledge engineering for real time intelligent control. In *Intelligent Control, 2002. Proceedings of the 2002 IEEE International Symposium on*, volume 14, pages 421–427. IEEE, 2002.

[29] J. Ferber. *Multi-agent systems : an introduction to distributed artificial intelligence*. Addison-Wesley, 1999.

[30] J Ferber and JF Perrot. *Les systèmes multi-agents : vers une intelligence collective*. 1995.

[31] M. Fernandez, A. Gomez-Perez, and Natalia Juristo. Methontology : from ontological art towards ontological engineering. In *Proceedings of the AAAI97 Spring Symposium Series on Ontological Engineering*, pages 33–40, 1997.

[32] Jeremy Frank, A. Jónsson, and Paul Morris. On reformulating planning as dynamic constraint satisfaction. *Abstraction, Reformulation, and Approximation*, pages 271–280, 2000.

[33] Thibault Gateau, Gaetan Séverac, Charles Lesire, Magali Barbier, and Eric Bensana. Knowledge Base for Planning, Execution and Plan Repair. In *ICAPS PlanEx Workshop*, 2012.

[34] MR Genesereth and RE Fikes. *Knowledge interchange format-version 3.0 : reference manual*. Number January. 1992.

[35] J.H. Gennari, S.W. Tu, T.E. Rothenfluh, M.A. Musen, and Others. Mapping domains to methods in support of reuse. *International journal of human computer studies*, 41(3) :399–424, September 1994.

[36] CJ Hansen, JH Waite, and SJ Bolton. Titan in the Cassini-Huygens Extended Mission. *Titan from Cassini-Huygens*, pages 455—-477, 2010.

[37] Gautier Hattenberger. *Vol en formation sans formation : contrôle et planification pour le vol en formation des avions sans pilote.* PhD thesis, Université Toulouse III - Paul sabatier, January 2008.

[38] Hao He. What Is Service-Oriented Architecture. *O'Reilly - webservices.xml.com*, pages 1–5, 2003.

[39] Arthur Herzog, Daniel Jacobi, and Alejandro Buchmann. A3ME-an Agent-Based middleware approach for mixed mode environments. In *The Second International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies*, pages 191–196. IEEE, September 2008.

[40] Gerda Horneck, Angioletta Coradini, Gerhard Haerendel, May-Britt Kallenrode, Paul Kamoun, Jean Pierre Swings, Alberto Tobias, and Jean-Jacques Tortora. Towards a European vision for space exploration : Recommendations of the Space Advisory Group of the European Commission. *Space Policy*, 26(2) :109–112, May 2010.

[41] J Huang, F Bastani, IL Yen, and J Dong. Extending service model to build an effective service composition framework for cyber-physical systems. *Service-Oriented*, 00(c), 2009.

[42] Jian Huang. A SOA Model of Cyber Physical Systems. *utd.edu*, 2009.

[43] Andrew Jenkins, Sebastian Kuzminsky, Kevin K Gifford, Bioserve Space Technologies, Robert L Pitts, and Kelvin Nichols. Delay/Disruption-Tolerant Networking : Flight Test Results from the International Space Station. In *IEE Aerospace Conference*, 2010.

[44] Li Jin and K.S. Decker. Ontology Oriented Exploration of an HTN Planning Domain through Hypotheses and Diagnostic Execution. In *Workshop on Knowledge Engineering for Planning and Scheduling, ICAPS*, 2010.

[45] GJ Kazz and E Greenberg. Mars Relay Operations : Application of the CCSDS Proximity-1 Space Data Link Protocol. Technical report, JPL, California Institute of Technology - NASA, 2002.

[46] Jinhan Kim, Jaejeong Lee, and Byungjeong Lee. Runtime Service Discovery and Reconfiguration Using OWL-S Based Semantic Web Service. *7th IEEE International Conference on Computer and Information Technology (CIT 2007)*, pages 891–896, October 2007.

[47] YH Lee, Wu Li, WT Tsai, and YS Son. A code generation and execution environment for service-oriented smart home solutions. volume 00, pages 1–8, 2009.

[48] Francois Legras. *Organisation dynamique d'équipe d'engins autonomes par écoute flottante.* PhD thesis, SUPAERO, 2003.

[49] S. Lemaignan, Raquel Ros, L. Mösenlechner, Rachid Alami, and Michael Beetz. ORO, a knowledge management platform for cognitive architectures in robotics. In *Proceedings of the 2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2010.

[50] G Lortal and S Dhouib. Integrating ontological domain knowledge into a robotic DSL. In *RoSym'10*, 2010.

[51] Z Maamar, D Benslimane, P Thiran, C Ghedira, S Dustdar, and S Sattanathan. Towards a context-based multi-type policy approach for Web services composition. *Data & Knowledge Engineering*, 62(2) :327–351, August 2007.

[52] David Martin, Mark Burstein, Jerry Hobbs, Ora Lassila, Drew Mcdermott, Sheila Mcilraith, Srini Narayanan, Massimo Paolucci, Bijan Parsia, Terry Payne, Evren Sirin, Naveen Srinivasan, and Katia Sycara. OWL-S : Semantic Markup for Web Services. Technical Report November 2004, W3C, 2004.

[53] B.P.P. Martinet and Bruno Patin. Proteus : A platform to organise transfer inside french robotic community. In *3rd National Conference on Control Architectures of Robots (CAR)*, 2008.

[54] Mario Merri, Bryan Melton, Serge Valera, and Andrew Parkes. The ECSS Packet Utilization Standard and Its Support Tool. In *SpaceOps 2002 Conference, Huston-Texas USA*, volume 49, pages 1–9, 2002.

[55] Daniel Morgan. The Future of NASA : Space Policy Issues Facing Congress. In *Science And Technology*. Library of congress washington dc congressional research service, 2010.

[56] JP Müller. Des systèmes autonomes aux systèmes multi-agents : Interaction, émergence et systèmes complexes. Technical report, 2002.

[57] NASA. sweet ontologies.

[58] NASA. Sustainable cooperative robotic technologies for human and robotic outpost infrastructure construction and maintenance. *Autonomous Robots*, 20(2) :113–123, 2006.

[59] Dana Nau, T.C. Au, Okhtay Ilghami, Ugur Kuter, J.W. Murdock, Dan Wu, and Fusun Yaman. SHOP2 : An HTN planning system. *Journal of Artificial Intelligence Research*, 20(1) :379–404, 2003.

[60] I.a.D. Nesnas, a. Wright, M. Bajracharya, R. Simmons, and T. Estlin. CLARAty and challenges of developing interoperable robotic software. *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003) (Cat. No.03CH37453)*, pages 2428–2435, 2003.

[61] Shin-ichiro Nishida. JAXA's Space Exploration Vision -Lunar and Planetary Exploration- 21. *JAXA - Présentation*, 2010.

[62] NF Noy and DL McGuinness. Ontology development 101 : A guide to creating your first ontology. Technical report, Stanford University, 2001.

[63] Feng Pan and J.R. Hobbs. Time in owl-s. In *Proceedings of the AAAI Spring Symposium on Semantic Web Services*, pages 29–36, 2004.

[64] MP Papazoglou, Paolo Traverso, and Schahram Dustdar. Service-oriented computing : a research roadmap. In *Dagstuhl Seminar Proceedings*, number April, pages 1–29, 2006.

[65] L.E. Parker. ALLIANCE : An architecture for fault tolerant multi-robot cooperation. *Robotics and Automation, IEEE Transactions on*, 14(2) :220–240, 1998.

[66] L.E. Parker. Distributed intelligence : Overview of the field and its application in multi-robot systems. *Journal of Physical Agents*, 2(1) :5–14, 2008.

[67] L Pedersen, D Kortenkamp, D Wettergreen, and I Nourbakhsh. A Survey of Space Robotics. *Proceedings of the 7th International Symposium on Artificial Intelligent, Robotics and Automation in Space*, 2003.

[68] Randall Perrey and Mark Lycett. Service-oriented architecture. In *Symposium on Applications and the Internet Workshops*, 2003.

[69] Thierry Planche and Olivier Notebaert. A3SysDef : Aurora Avionics Architecture System Definition. Technical report, EADS Astrium SAS, 2005.

[70] Ioannis Psaras, Lloyd Wood, and Rahim Tafazolli. Delay-/Disruption-Tolerant Networking State of the Art and Future Challenges. Technical report, Technical Report, University of Surrey, UK, 2010.

[71] J Pullen, M Hieb, S Levine, A Tolk, and Curtis L. Blais. Joint Battle Management Language (JBML)-US Contribution to the C-BML PDG and NATO MSG-048 TA. *IEEE European Simulation*, 2007.

[72] Rob Raskin. Guide to SWEET ontologies. Technical report, NASA/Jet Propulsion Lab, Pasadena, CA, USA, 2006.

[73] Task Force Report. The Role of Autonomy in DoD Systems. Technical Report July, Department of Defense, U.S.A., 2012.

[74] Antonio Ceballos Roa, Michel van Winnendael, Alberto Medina Andres, Jorge Ocon Alonso, and Fernando Gandia Abellan. TOWARDS GOAL ORIENTED SPACE ROBOTICS OPERATIONS

USING AUTONOMOUS CONTROLLERS. In *i-SAIRAS : International Symposium on Artificial Intelligence, Robotics and Automation in Space*, number 1, 2012.

[75] Prabir Sarkar, S. Phaneendra, and Amaresh Chakrabarti. Developing Engineering Products Using Inspiration From Nature. *Journal of Computing and Information Science in Engineering*, 8(3) :031001–9, September 2008.

[76] CI Schlenoff, E Messina, A Lytle, and B Weiss. Test Methods and Knowledge Representation for Urban Search and Rescue Robots. *the Climbing and Walking Robots*, 2007.

[77] Craig Schlenoff and Elena Messina. A robot ontology for urban search and rescue. In *Proceedings of the 2005 ACM workshop on Research in knowledge representation for autonomous systems*, pages 27–34, New York, New York, USA, 2005. ACM.

[78] Gregory P. Scott and Alex Ellery. Design of a Biomimetic Walking Mars Explorer for the ESA Bionics & Space Systems Design - Report - Contract (AO/1-4469/03/NL/Sfe). *Surrey Space Centre*, 2005.

[79] Zied Sellami, V. Camps, N. Aussenac-Gilles, and S. Rougemaille. Ontology Co-construction with an Adaptive Multi-Agent System : Principles and Case-Study. *Knowledge Discovery, Knowlege Engineering and Knowledge Management*, pages 237–248, 2011.

[80] Jianhua Shi, Jiafu Wan, Hehua Yan, and Hui Suo. A Survey of Cyber Physical Systems. In *Int. Conf. onWireless Communications and Signal Processing*, pages 1–6. IEEE, 2011.

[81] A. Singh, Andreas Krause, C. Guestrin, and W. Kaiser. Efficient informative sensing using multiple robots. *Journal of Artificial Intelligence Research*, 34(1) :707–755, 2009.

[82] Beatrice G. R. Smith, Gregory P. Scott, and Chakravarthini M. Saaj. *Biorobotics : Innovative and low cost technologies for next generation planetary rovers*. IEEE, June 2009.

[83] RG Smith. The contract net protocol : High-level communication and control in a distributed problem solver. *Computers, IEEE Transactions on*, C(12), 1980.

[84] Milind Tambe. Towards flexible teamwork. 1997.

[85] TaskGroup-SCI-144. Integration of Systems with Varying Levels of Autonomy - Report AC/323(SCI-144)TP/144. *North Atlantic Treaty Organization*, (September), 2008.

[86] Moritz Tenorth and Michael Beetz. KNOWROB — Knowledge Processing for Autonomous Personal Robots. In *Intelligent Robots and Systems, 2009.*, 2009.

[87] Moritz Tenorth, Lars Kunze, Dominik Jain, and Michael Beetz. KNOWROB-MAP – Knowledge-Linked Semantic Object Maps. In *Proceedings of 2010 IEEE-RAS International Conference on Humanoid Robots*, 2010.

[88] Moritz Tenorth, Alexander Perzylo, Reinhard Lafrenz, and Michael Beetz. The RoboEarth language : Representing and Exchanging Knowledge about Actions, Objects, and Environments. In *IEEE International Conference on Robotics and Automation (ICRA)*, number 3, 2012.

[89] Gateau Thibault. Supervision de mission pour une équipe de véhicules autonomes hétérogènes. *Journée des thèses 2010 - ONERA*, pages 1–2, 2010.

[90] A. Tramutola and A. Martelli. Beyond the Aurora Architecture for the new challenging applications. The Enhanced Avionics Architecture for the Exploration Missions . Technical report, Thales Alenia Space, 2010.

[91] Q Tran and G Low. MOBMAS : A methodology for ontology-based multi-agent systems development. *Information and Software Technology*, 50(7-8) :697–722, June 2008.

[92] TRT. PROTEUS - Etat de l'art sur les ontologies pour la robotique. Technical report, ANR, 2010.

[93] Al Underbrink, Andrew Potter, K Witt, and Jason Stanley. Modeling sensor web autonomy. *Aerospace Conference, 2011 IEEE*, 2011.

[94] A Valero, G Randelli, P. de La Puente, D. Calisi, D. Rodriguez-Losada, F. Matia, and D. Nardi. UPM-SPQR rescue virtual robots team description paper. *Team Description Paper for RoboCup*, 2009.

[95] Gianfranco Visentin. Autonomy in ESA Planetary Robotics Missions. Technical report, ESA, Automation & Robotics Section (TEC-MMA), 2007.

[96] H Wache, T. Voegele, U Visser, H Stuckenschmidt, G Schuster, H Neumann, and S. H\\"ubner. Ontology-based integration of information-a survey of existing approaches. In *IJCAI-01 workshop : ontologies and information sharing*, volume 2001, pages 108–117. Citeseer, 2001.

[97] R. Waibel, M. and Beetz, M. and Civera, J. and D'Andrea, R. and Elfring, J. and Galvez-Lopez, D. and Haussermann, K. and Janssen, R. and Montiel, J.M.M. and Perzylo, A. and Schiessle, B. and Tenorth,

M. and Zweigle, O. and van de Molengraft. RoboEarth-A World Wide Web for Robots. *Robotics Automation Magazine, IEEE*, 18(June) :69–82, 2011.

[98] Yi Wei and M. Brian Blake. Service-Oriented Computing and Cloud Computing. *Ieee Internet Computing*, 2010.

[99] Kenneth J Witt, Jason Stanley, David Smithbauer, Dan Mandl, Vuong Ly, Al Underbrink, and Mike Metheny. Enabling Sensor Webs by Utilizing SWAMO for Autonomous Operations. In *NASA Earth Science Technology Conference (ESTC2008)*, 2008.

[100] Mark Yim, Wei-min Shen, Behnam Salemi, Daniela Rus, Mark Moll, Hod Lipson, Eric Klavins, and Gregory Chirikjian. Modular Self-Reconfigurable Robot Systems [Grand Challenges of Robotics]. *IEEE Robotics & Automation Magazine*, 14(1) :43–52, March 2007.

Annexe A

Méthodes et outils pour les ontologies

A.0.1 Outils de modélisation des ontologies

Il existe de nombreux outils de modélisation pour la création d'ontologies, dont un rapport du projet PROTEUS fait une liste détaillée [92].

Nous pouvons citer COE [1] basé sur l'outil CmapTools [2], Ontolingua [3] qui utilise le langage "Knowledge Interchange Format" (KIF) [4] [21], OpenCyc browser [5], the OWL API [6]. Il existe aussi des navigateurs en ligne qui permettent de parcourir et d'éditer des ontologies de manière simple mais limitée. Comme OntologyBrowser [7] ou OWLSight [8].

Nous avons choisi "Protégé [9]" dans sa dernière version 4.1, développé par l'université de Stanford. Il est open source et bénéficie d'une communauté de développeurs active. Il est mondialement utilisé, compatible avec le format OWL2, inclut des outils d'inférence et de vérification de cohérence, pratique et simple d'utilisation [10].

1. http ://www.ihmc.us/groups/coe/
2. http ://cmap.ihmc.us/
3. http ://www.ksl.stanford.edu/software/ontolingua
4. http ://logic.stanford.edu/kif/kif.html
5. http ://www.opencyc.org/
6. http ://owlapi.sourceforge.net/
7. http ://owl.cs.manchester.ac.uk/browser
8. http ://pellet.owldl.com/owlsight
9. http ://protege.stanford.edu
10. http ://protegewiki.stanford.edu/wiki/Protege4Features

A.0.2 Méthodes et conseils pour la création d'ontologies

De nombreux tutoriels existent pour apprendre à créer une ontologie. Le premier à regarder est sûrement celui proposé avec le logiciel Protégé [11] qui est un guide très clair et précis [62].

Le document "METHONTOLOGY"[31] est bien plus ancien mais explique également comment démarrer une ontologie à partir de rien. La FIPA a également édité un document sur la spécification d'une ontologie de service [1], dont l'annexe A discute de la conceptualisation d'une ontologie et l'annexe B est un guide sur comment définir une nouvelle ontologie.

Voici les conseils les plus simples et les plus utiles à respecter lors de la création d'une ontologie :

- Deux erreurs à éviter lors du développement [21] :
  - Une classe ne doit pas hériter d'elle-même à travers des héritages successifs.
  - Les définitions d'un axiome ne doivent pas être contradictoires (erreur de cohérence).
- Pour permettre une réutilisation simple et efficace, il faut isoler autant que possible la connaissance des méthodes de celle du domaine. Et pour les connecter il faut définir des relations déclaratives [35].
- Il n'y a pas de solution unique de modéliser un domaine, il y a toujours plusieurs alternatives viables. La meilleure solution dépend de l'application recherchée et de ses extensions futures qu'il faut anticiper [62].
- Le développement d'une ontologie est forcément un processus itératif [62].
- Les concepts d'une ontologie doivent être proches des objets (physiques ou logiques) et des relations du domaine représenté. Le plus souvent, ce seront des noms (objets) ou des verbes (relations) dans des phrases du langage courant décrivant ce domaine [62].
- Une information est une classe de l'ontologie si elle sert au raisonnement, sinon c'est un attribut d'une autre classe. Ce test est très utile pour vérifier l'utilité d'une classe [Nathalie Aussenac - IRIT].
- Pour être justifié, chaque nouveau concept modélisé doit être différent de ceux existants et partager des points communs avec son père et ses frères [Nathalie Aussenac - IRIT].

11. http ://protege.stanford.edu/doc/users.html

– Une relation de super-classe est équivalente au terme "peut" et une relation de "classe équivalente" (defined) est équivalente au terme "doit" [12].

12. Cours de Robert Stevens - www.cs.man.ac.uk/ stevensr/menupages/fhkb.php

Annexe B

Matériel et méthodes pour l'environnement de développement et de simulation

B.1 Outil ROS

L'implémentation du système dans un environnement de simulation se fait en utilisant ROS (Robot Operating System [1]). ROS est similaire pour de nombreux aspects à d'autres framework de robotique comme YARP [2], Orocos [3] ou GenoM [4] qui ont également pour but d'encapsuler du code modulaire et de standardiser les communications internes afin de permettre de réutiliser et de partager des modules robotiques.

Nous avons choisi ROS car il est largement utilisé par la communauté scientifique et bénéficie d'une communauté de développement très active.

ROS fournit une couche d'abstraction pour gérer les communications entre différents modules, basés autour de message très simples, construits sur le principe des structures du langage C. Un message est décrit par les types des variables ou les types d'autres messages qu'il encapsule. Les modules peuvent créer des "Topics" de communication, ou bien s'inscrire à des "Topics" existant en tant que "Talker" ou "Listener" (ou les deux). Tous les "Listener" d'un "Topic" reçoivent automatiquement les messages postés par les "Talker", comme schématisé sur la figure B.1.

Cet environnement est donc tout à fait approprié aux applications multi-

1. Robot Operating System (ROS) - www.ros.org
2. YARP - http ://eris.liralab.it/yarp/
3. Orocos - http ://www.orocos.org/
4. GenoM - http ://homepages.laas.fr/sara/RIA/RIA-genom.html.fr

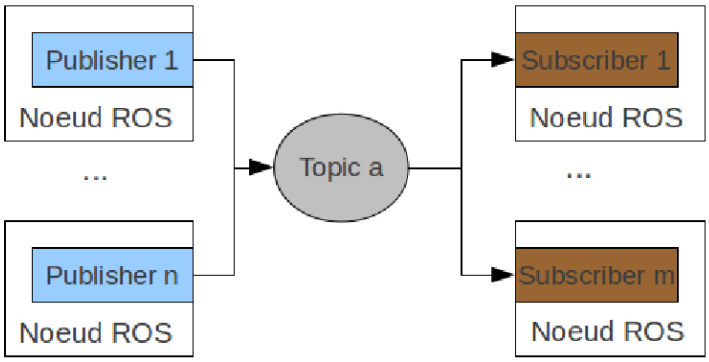

FIGURE B.1 – Listeners et Subscribers d'un Topic ROS

robots et aux architectures internes organisées en modules comme le système que nous souhaitons mettre en place.

ROS est implémenté en plusieurs langages (C++, Python, Lisp, Java, Lua), tous compatibles les uns avec les autres (e.g. un module écrit en Python peut communiquer avec un module écrit en Java, tous deux enregistrés auprès d'un "ROSMaster" écrit en C++).

B.2 Outil STK

STK, pour "System ToolKit" [5] (anciennement "Satellite ToolKit"), est une suite logicielle propriétaire et commerciale développée par AGI (Analytical Graphics Inc.). AGI développe des logiciels d'analyse et de modélisation pour les domaines du spatial, de la défense et du renseignement. STK est le logiciel phare d'AGI et sa vitrine technologique. Actuellement à sa 10 ème version, STK est disponible gratuitement à des fins "non productives" dans une version limitée qui nous permet de calculer la visibilité d'engins au sol ou en orbite de planètes du système solaire. Une version professionnelle, ainsi que des modules payants permettent d'ajouter des fonctionnalités avancées pour modéliser finement les déplacements et les communications des engins.

STK permet l'ajout de différents engins allant des satellites aux bateaux en passant par des zones d'intérêt. En ce qui nous concerne, nous avons utilisé 3 types d'engins : les véhicules, les satellites ainsi que les infrastructures.

Dans le cadre de notre simulateur, nous utilisons l'interface graphique fournie par la version de base et le module principal de STK, "STK Engine",

5. STK - http ://www.agi.com/products/stk/

par le biais de son API Java, pour réaliser deux actions principales :

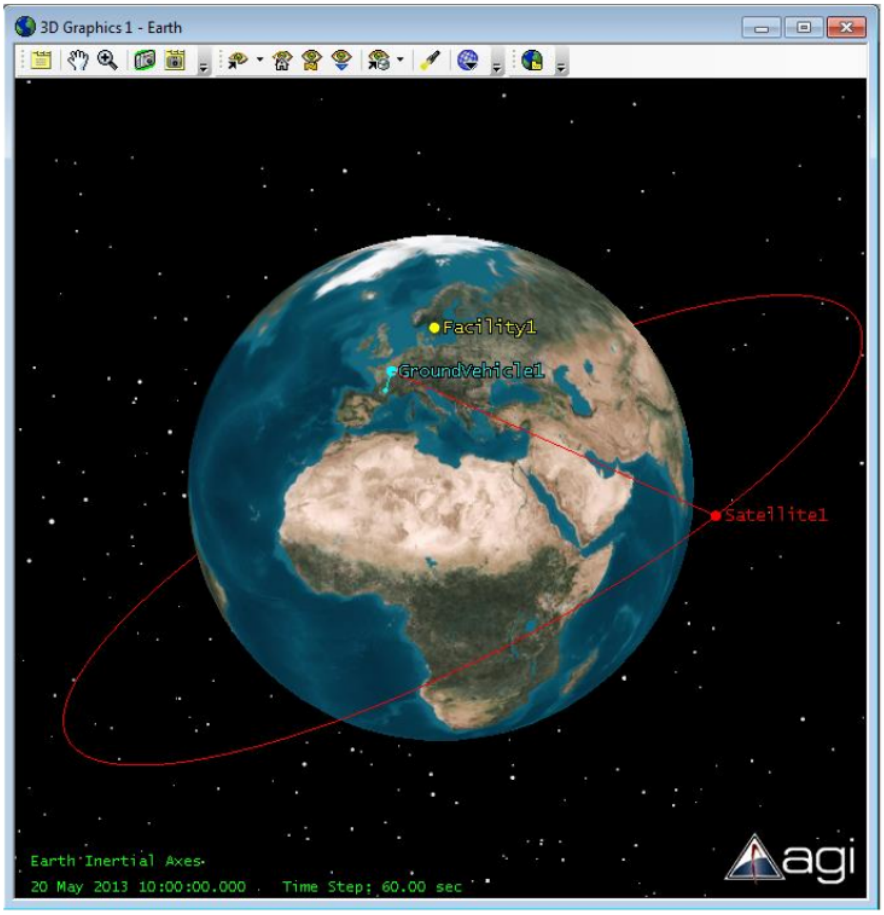

FIGURE B.2 – Illustration d'un calcul de visibilité via l'interface de STK

- Le calcul de visibilité
  C'est en effet la fonction qui nous permet de traiter les demandes de communication et de simuler la réception ou la perte d'un message. Pour STK, un calcul de visibilité s'effectue sur une période temporelle indiquée. Il est possible de demander en plus de la visibilité les positions relatives des deux engins dans l'espace (distance, azimut, élévation). Pendant la période temporelle correspondante et s'il y a effectivement visibilité, l'accès est représenté visuellement par une ligne continue joignant les deux entités en question, comme présenté sur la figure B.2.
- Les animations
  Les scénarios créés peuvent être animés en temps réel, pour permettre une visualisation en 3D. Chaque seconde de la simulation représentant

une seconde de la réalité. Il est possible d'appliquer un facteur de vitesse à l'animation pour l'accélérer. De sorte qu'un message émis depuis Mars vers la Terre ne prenne pas une dizaine de minutes comme dans la réalité, mais quelques secondes.

Annexe C

Détails de l'implémentation

C.1 Méthodes implémentées pour la gestion des interaction

Nous décrivons ici la liste des principales méthodes utilisées ainsi que leur rôle dans la simulation.

C.1.1 Pour l'architecture locale - Classe LocalControler

- TreatInternalRequest (Request) Parse les requêtes internes provenant d'autres modules et réagit en fonction.
- ExecutionScenario () Permet de déclencher des actions conformément au script du scénario de mission à jouer.
- MangementServiceExecution () Permet de simuler l'exécution d'un service et ses réactions/décisions associées.

C.1.2 Pour la gestion de la connaissance - Classe KnowledgeManager

- treatInternalRequest (Request) Parse les requêtes internes provenant d'autres modules et réagit en fonction.
- getKnownAgent(AgentID) Renvoie la description de l'engin dont l'ID est passé en paramètre s'il est connu.
- getKnownAgentList() Renvoie la liste de description de tous les engins connus
- getMyDescription() Renvoie la propre description de l'engin courant.

– getKnownService(ServiceID) Renvoie la description du service dont l'ID est donné en argument.
– getKnownServiceList() Renvoie la liste de descriptions de tous les services connus
– isAgentAlreadyKnown(AgentID) Renvoie "vrai" si l'engin dont l'ID est donné en argument est connu, sinon faux.
– add/remove KnownAgent(AgentDescription/agentID) Ajoute / retire un engin de la connaissance.
– add/remove Service(AgentID, ServiceDescription / AgentID, ServiceID) Ajoute / retire un service de la connaissance associée à l'engin qui le fourni.
– createAgent/ServiceXMLDescription(Agent/ServiceID) Créer une description XML du service / de l'engin dont l'ID est donné en paramètre.
– setAgent/ServiceFromXMLDescription(XMLAgent/ServiceDescription) Utilise la description XML pour mettre à jour ou créer un la connaissance d'un engin ou d'un service.

C.1.3 Pour la gestion des messages externes - Classe CommunicationManager

– treatInternalRequest (Request) Parse les requêtes internes provenant d'autres modules et réagit en fonction.
– treatExternalMessage (Message) Parse les messages internes provenant d'autres engins et réagit en fonction.
– sendMessage(Message) Envoie le message donné en paramètre aux engins à proximité.

C.2 Liste des interactions gérées par le module ISIS

Voici la liste des interactions qui sont initiées par l'architecture locale, puis par le gestionnaire des communications :

- mise à jour d'un service (initiée par l'architecture locale, puis par le gestionnaire des communications),

- invocation d'un service,

- monitoring d'un service,

- demande d'envoi de la partie de description d'un service réservée à l'utilisateur du service,

- demande d'envoi de la partie de description physique d'un engin.

C.3 Présentation du code source

C.3.1 Organisation des paquets

La figure C.1 présente l'organisation générale des paquets du code source. On y retrouve dans l'ordre les paquets suivants :

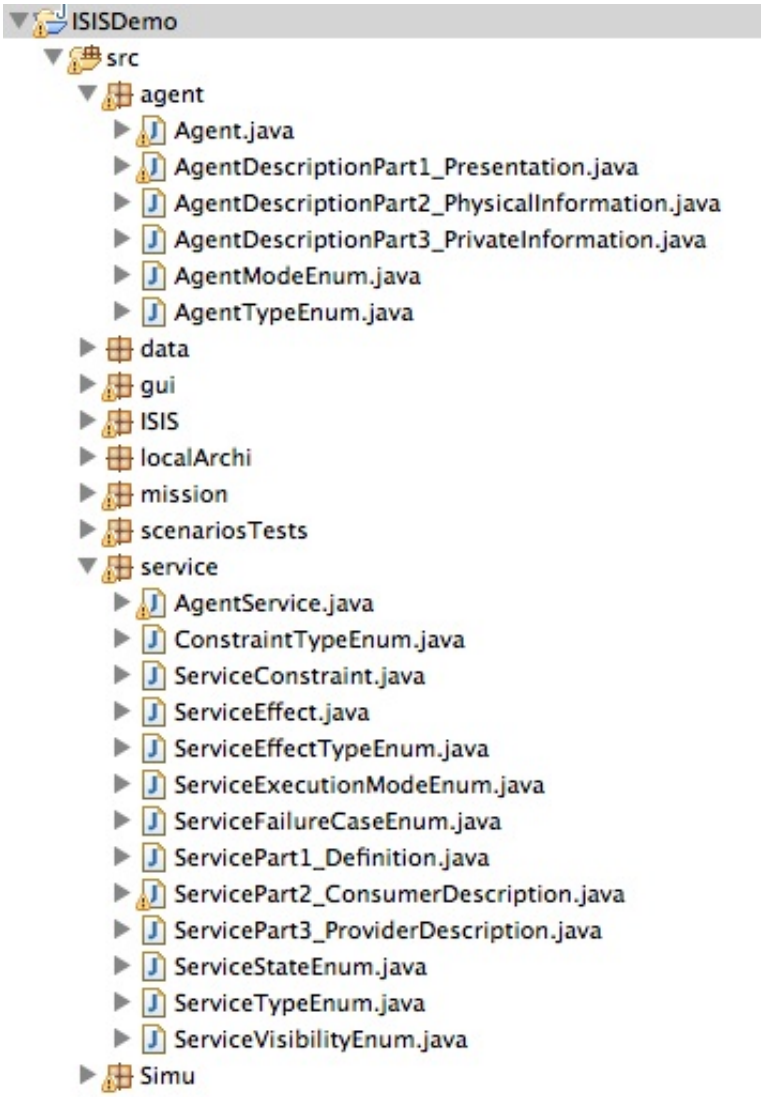

FIGURE C.1 – Organisation générale du code source, vue des paquets

– agent - un paquet pour les classes liées aux engins.

– data - pour les données bas niveau comme les positions, cartes, autorisation...
– gui - pour la gestion de l'affichage de la connaissance de engins.
– ISIS - pour les classes liées au module ISIS.
– localArchi - pour les classes simulant l'architecture locale des engins.
– mission - description des missions.
– scenariosTests - Pour le jeu des simulations.
– service - pour les classes liées aux services.
– situ - pour les classes liées à l'utilisation du simulateur des commmunications.

Plus précisément la figure C.2 montre les classes constituantes du module ISIS : la classe principale ISISModule qui instancie les classes CommunicationManager et KnowledgeManager, et qui utilise les classes : Knowledge pour manipuler la connaissance, InternalRequest pour créer les requêtes internes et Message pour créer les messages externes.

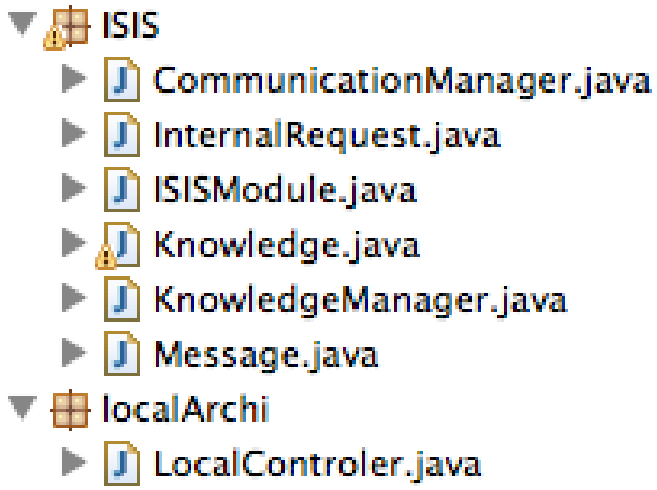

FIGURE C.2 – Organisation du code source pour les paquets ISIS et localArchi

La figure C.3 montre, elle, l'organisation du code lié aux simulatiosw. Pour chaque simulation deux classes sont créées par engin impliqué. Une pour le module ISIS et une pour l'architecture locale, chacune étendant les classes mères ISISModule et LocalArchi. Une classe, Scenario1UpdateKnowledge dans l'exemple de la figure, sert à lancer chaque agent dans l'environnement ROS.

Plus bas, le paquet Simu, inclut les classes permettant aux engins de communiquer via le simulateur des communications.

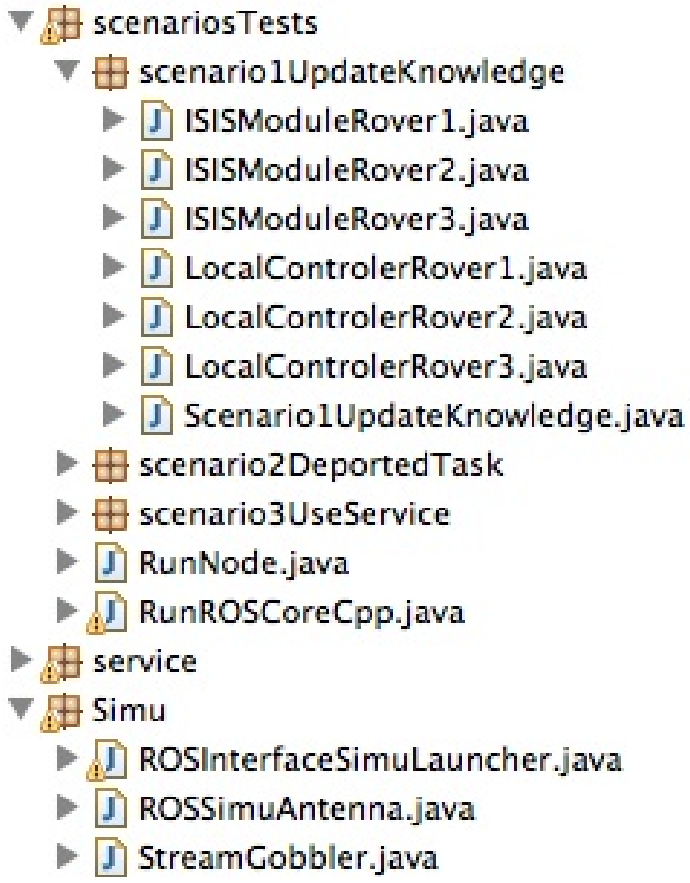

FIGURE C.3 – Organisation du code source pour les paquets liés à la simulation