Intergiciels pour systèmes multi-robots: état de l'art

Stefan-Gabriel Chitic, Julien Ponge, Olivier Simonin

Intergiciels pour systèmes multi-robots : état de l'art

Stefan-Gabriel CHITIC, Julien PONGE, Olivier SIMONIN
Université de Lyon, INSA-Lyon, CITI-INRIA
F-69621, Villeurbanne, France
stefan.chitic@insa-lyon.fr, julien.ponge@insa-lyon.fr, olivier.simonin@insa-lyon.fr

ABSTRACT
Les flottes de robots autonomes sont des systèmes complexes qui nécessitent des interactions et des communications entre des éléments matériels et logiciels hétérogènes. Malgré l'évolution du domaine robotique, il existe encore un manque d'architectures logicielles de référence et d'intergiciels éprouvés, en particulier pour les systèmes multi-robots. De nombreuses équipes continuent d'élaborer des logiciels orientés spécifiquement vers le matériel lié à leurs robots. Cette vision rend le partage des modules ou des codes existants difficile. Un intergiciel robotique doit être conçu pour abstraire l'architecture matérielle de bas niveau, pour faciliter la communication et l'intégration avec des briques logicielles tierces. Dans cet article, nous présentons et comparons les intergiciels les plus susceptibles de s'appliquer aux systèmes multi-robots. Nous présentons et discutons également deux solutions de type cloud dédiées aux plate-formes multi-robots.

Keywords
Systèmes multi-robots, intergiciel, cloud robotique

1. INTRODUCTION
Une flotte de robots autonomes est constituée de plusieurs robots capables de partager des données et d'effectuer ensemble une ou plusieurs tâches. Ces systèmes sont de plus en plus envisagés en connexion et en coopération avec d'autres objets / capteurs fixes et mobiles dans l'environnement. Un robot est un système complexe et hétérogène qui exige des communications entre ses divers composants (capteurs, actionneurs et composants logiciels).

Les travaux en intelligence artificielle distribuée ont montré que le partage des tâches réduisait la complexité et la difficulté d'un problème, même si cela requiert des mécanismes de coordination [13]. Le même concept s'applique dans le monde robotique. Il est nécessaire dans les systèmes multi- robots d'être capable de diviser en plusieurs sous-tâches un problème complexe. Il existe également un besoin de partage des informations entre les robots et les objets extérieurs. La communication à l'intérieur d'une flotte peut être faite en utilisant une infrastructure de réseau centralisée comme les points d'accès WiFi ou une architecture décentralisée avec un réseau ad- hoc.

Les robots ont un grand potentiel dans de nombreuses applications et ils offrent de nouvelles approches à des problèmes comme les missions de surveillance, l'assistance ou le sauvetage dans des zones où l'accès peut ne pas être possible pour les hommes.

Les systèmes multi-robots peuvent augmenter leur puissance de calcul en utilisant des architectures externes, comme des data-grids ou des clouds pour robots. Le principal avantage d'un cloud pour robots est la diminution de la durée d'une tâche, puisque le calcul est déporté vers un data-center où la puissance de calcul n'est typiquement pas un problème. Cette approche a également ses inconvénients, puisque chaque système robotique doit communiquer et partager des données avec un système centralisé en utilisant Internet.

Malgré l'évolution du domaine robotique, il n'y a pas ou peu d'architecture logicielle de référence ou d'intergiciel standardisé et eprouvé [41]. Un intergiciel est une couche logicielle qui agit comme un pont entre une couche réseau ou un système d'exploitation et d'autres applications. Le logiciel développé est lié à l'architecture et au matériel utilisé. Sans intergiciel le partage des modules et des algorithmes est presque impossible.

Cependant une tendance récente est à la convergence du monde robotique et de celui de l'intergiciel (ou *middleware*), afin de construire des solutions intergicielles efficaces pour la robotique. Cette tendance est vraie avec ROS [1] qui a émergé du monde du génie logiciel.

Un intergiciel robotique doit gérer l'hétérogénéité du matériel, faciliter la communication à l'intérieur et à l'extérieur d'un robot, améliorer la qualité des logiciels et réduire les délais et les coûts lors de la création de nouvelles applications. Il doit aussi permettre aux robots de s'auto- configurer, de s'auto-adapter et de s'auto-optimiser face à l'évolution de l'environnement. La combinaison des composants et de la programmation orientée services simplifie considérablement la mise en œuvre d'applications constamment en évolution [16]. Les robots doivent être capables d'auto-provisionnement (auto-découverte et auto-installation des modules logiciels et des bibliothèques utilisées par les autres robots de la flotte) et d'auto-profilage via un système de configuration et de lancement des applications et services sans intervention directe de composants externes.

1. Robot Operating System [37]

Il existe déjà des intergiciels qui tentent d'intégrer une partie des besoins souhaités. La plupart d'entre eux sont conçus pour des systèmes mono-robot mais peuvent être utilisés pour une flotte. Il existe aussi de nouvelles approches cloud conçues pour le multi-robots. Cet article examine les besoins d'intergiciel pour les systèmes multi-robots et la façon dont ils facilitent le développement des logiciels embarqués. Nous comparons les différentes solutions existantes en présentant leurs avantages et leurs inconvénients sur la base de plusieurs critères qui couvrent l'architecture, l'infrastructure et l'usage des plate-formes. Nous présentons et discutons également deux solutions cloud pour robots.

Ce document est structuré comme suit. La Section 2 définit les défis d'un intergiciel multi-robots, puis la Section 3 décrit les solutions existantes. La Section 4 présente nos critères de comparaison, qui sont utilisés en Section 5 pour comparer les principales solutions. La Section 6 présente deux approches cloud pour flotte robotique tandis que la Section 7 synthétise notre analyse des solutions existantes. La Section 8 conclut l'article.

2. LES DÉFIS DES INTERGICIELS ROBOTIQUES

Pourquoi un intergiciel pour la robotique ?.

L'intergiciel est un élément important dans le processus de développement, de déploiement et d'exploitation du logiciel. Aujourd'hui, les robots sont utilisés dans un contexte de flotte, étant capables d'avoir une perception commune de l'environnement et aussi de communiquer à l'intérieur de la flotte et avec des objets externes tels que des capteurs, des passerelles réseau, ou encore des appareils mobiles dotés de communications sans fil [28]. On parle alors de robots ubiquitaires et hétérogènes. Ces dispositifs et robots sont constitués de matériels variés qui sont contrôlés par des logiciels développés dans différents langages de programmation, utilisant de multiples normes et protocoles de communication.

L'un des défis que doit relever un intergiciel est la modularité du logiciel tel que présenté dans [10]. Le développement d'applications robotiques a besoin d'adopter une vision modulaire. Le processus de développement doit être simplifié par l'intégration de couches plus élevées d'abstraction avec des interfaces d'application (API) [28]. De plus, l'intergiciel doit soutenir des mécanismes *plug-and-play* pour les nouveaux modules développés, afin de pouvoir les changer dynamiquement.

Enfin, un intergiciel devrait être en mesure de partager des connaissances et des modules avec un référentiel des connaissances tel que RoboEarth [43] (Voir Section 6).

En outre, un intergiciel robotique devrait intégrer des rôles multiples d'un intergiciel classique [20]. Il s'agit de l'évolutivité des intergiciels orientés messages. L'intergiciel devrait être orienté service afin de permettre aux services robotiques d'être publiés par les fournisseurs et découverts par les consommateurs.

L'infrastructure et la communication.

Les modules logiciels doivent pouvoir fonctionner sur n'importe quelle infrastructure, ce qui implique que l'intergiciel doit proposer une couche d'abstraction matérielle afin de faciliter la réutilisation des modules. L'intergiciel doit rendre le robot conscient de ses capacités en découvrant automatiquement ses capteurs et ses actionneurs. Ces capacités devraient être organisées dans des services robotiques pouvant être diffusés pour permettre à chaque robot de savoir de quoi les membres de son équipe sont capables. Ces mécanismes automatiques de configuration et de découverte des ressources augmenteraient ainsi le potentiel de chaque robot.

La mobilité des robots nécessite une auto-organisation [2] de la flotte reposant sur un réseau de communication décentralisé. De plus, cette mobilité peut diviser la flotte au niveau de la couche communication (couche physique) alors que l'on souhaite garder la même configuration au niveau de la couche applicative ; il faut alors que les membres s'adaptent automatiquement à la nouvelle configuration de la flotte [44].

L'intergiciel doit également fournir un soutien à la collaboration entre les robots en s'assurant que tous les robots partagent les mêmes informations. Il doit également fournir des APIs permettant de faciliter le développement de comportements collaboratifs.

Les inconvénients.

Même les intergiciels les plus aboutis peuvent avoir des limites. Comme mentionné dans [41] le fait d'avoir une couche d'abstraction matérielle qui cache l'hétérogénéité des capteurs et des actionneurs a ses inconvénients. La spécificité des capteurs, leur position, leurs limites, et la forme du robot augmente la complexité du logiciel de contrôle. En extrapolant, cette variabilité rend l'intergiciel plus complexe à définir. Si dans les intergiciels cloud classiques le réseau peut être considéré comme presque totalement fiable, dans une flotte robotique le réseau est susceptible d'être interrompu. L'intergiciel ne doit pas essayer de traiter une exception de défaillance du réseau mais accepter que le réseau est temporairement inaccessible et passer en mode dégradé jusqu'à son rétablissement. La même logique devrait s'appliquer en cas de panne du matériel puisque les robots sont généralement déployés dans des environnements où ils doivent rester autonomes.

L'ensemble des défis que nous venons de mentionner pour un intergiciel multi- robots sont élevés. Il existe de nombreuses techniques d'intergiciel de cloud qui peuvent être appliquées dans le contexte des flottes robotiques, mais il reste de fortes différences entre un cloud et une flotte de robots dues en particulier aux spécificités de mobilité et de communication à l'intérieur de la flotte. Jusqu'à présent, plusieurs intergiciels pour la robotique ont été proposés et apparaissent comme prometteurs pour la gestion de flottes de robots.

Les sections suivantes présentent et comparent les intergiciels les plus utilisés et adaptés aux systèmes multi-robots ainsi que des plate-formes cloud pour les flottes robotiques.

3. LES INTERGICIELS EXISTANTS

Dans cette section, nous présentons les intergiciels les plus utilisés en considérant leur application aux flottes robotiques. Un état de l'art complet de tous les intergiciels mono-robot est clairement impossible en raison du grand nombre d'intergiciels existants et l'apparition constante de nouveaux. Pour réduire le nombre d'intergiciels présentés, nous avons sélectionné en premier lieu ceux qui sont compatibles avec une plate-forme multi-robots, puis nous avons considéré le nombre de citations dans des publications. Sur

2. c'est un processus où l'organisation globale ou la coordination entre les robots découle de leurs interactions locales (par exemple l'élection d'un responsable à l'aide d'un système de vote)

la base de ces critères nous avons retenu 7 intergiciels pour la robotique : *Player/Stage*, *ROS*, *Miro*, *MRDS*, *Marie*, *Orca* et *Pyro*. Mentionnons d'autres intergiciels disponibles : *Claraty* [31], *OpenRT-Maist* [2], *OPRos* [21], *Carmen* [29], *Orocos* [6], *ERSP* [12], *RoboFrame* [33], *WURDE* [17], *Aseba* [25], *Skilligent* [40], *MissionLab* [11], *SmartSoft* [38], *iRobotAware* [19], *Yarp* [15], *Spica* [4], *Babel* [14], *DROS* [42], *IRSP* [46], *K- MIDDLEWARE* [8], *OpenRDK* [7], *OpenJAUS* [32], *ORCCAD* [39], *RIK* [5], *MRPT* [30], *Webots* [26], etc.

Le résumé qui suit donne un aperçu et une description de chaque logiciel sélectionné. Il précise également les plate-formes robotiques compatibles et les caractéristiques les plus pertinentes.

Player/Stage [24]. est conçu pour fournir une infrastructure, des pilotes et une collection de bibliothèques de périphériques partagés pour les applications robotiques. C'est l'un des premiers intergiciels qui a émergé pour les systèmes robotiques, et d'autres intergiciels utilisent Player comme fondation. Player/Stage ne considère pas un robot comme une unité, mais traite séparément les dispositifs, ce qui en fait un serveur de référentiel pour actionneurs, capteurs et robots. Les principales caractéristiques de Player sont le serveur de référentiel des périphériques, la variété des langages de programmation, le protocole de transport basé sur des sockets, la modularité et le fait d'être open-source. Cet intergiciel est composé de deux éléments : Player et Stage. Player représente l'intergiciel lui même et Stage est un simulateur 2D.

Les plate-formes qui peuvent exécuter Player/Stage incluent MobileRobots, Segway, Acroname, K-Team robots, iRobot's RFLEX-based, Botrics et Evolution Robotics.

ROS [35]. (Robot operating system) est un intergiciel récent et souple pour les applications robotiques. C'est une collection d'outils, de bibliothèques et de conventions qui visent à simplifier la définition de comportements complexes et robustes pour une grande variété de plate-formes robotiques. Il fournit une abstraction du matériel, des pilotes de périphériques, des visualiseurs, une infrastructure pour communiquer à travers des messages et une gestion des paquets.

ROS est livré avec une série de bibliothèques contenant des services robotiques comme des fonctions de SLAM [3], de navigation autonome, ou de suivi d'objets. ROS est conçu pour être multi-plateforme.

Les plate-formes qui prennent en charge ROS incluent le PR2, Turtlebot, Kobuki, Husky et Dr. Robot Jaguar V4 with Manipulator Arm.

Miro [23] . est un intergiciel orienté objet, distribué, développé pour améliorer le processus de développement de logiciels en augmentant l'intégrabilité des logiciels hétérogènes, la modularité et la portabilité des applications robotiques. Il a été développé en C++ pour Linux basé sur CORBA (*Common Object Request Broker Architecture*). Cela permet l'interopérabilité entre plateformes, rendant l'intergiciel applicable dans un contexte distribué multi-robot.

3. *Simultaneous Localization and Mapping*

Les plate-formes qui supportant Miro sont iRobot B21 et MobileRobots Pioneer. Miro est très flexible et peut facilement être étendu pour supporter de nouveaux dispositifs ou applications robotiques.

Microsoft Robotics Developer Studio [22]. (MRDS) est un intergiciel basé sur Windows pour le contrôle et la simulation du robot créé par Microsoft.

Visual Programming Language, qui est un élément clé du MRDS, est un environnement de développement graphique qui utilise un catalogue de service et d'activité.

MRDS vise un public universitaire, amateur et les développeurs commerciaux. Il gère une grande variété de matériels et robots comme Eddie Robot, ABB Group Robotic, CoroWare CoroBot, Lego Mindstorms NXT, iRobot Create, Parallax Boe- Bot etc.

Marie [9]. (Mobile and Autonomous Robotics Integration Environment) est un intergiciel conçu pour permettre l'intégration et la distribution de logiciels pour les systèmes robotiques.

Il a été créé en C++ et utilise *Adaptive Communication Environment* (ACE) comme infrastructure de communication. Le composant central fourni par l'intergiciel, appelé *Mediator Design Pattern* (MDP), permet aux composants logiciels de se connecter à MARIE.

MARIE peut fonctionner sur MobileRobots Pioneer 2. Ses principales caractéristiques sont l'interopérabilité et la réutilisation des modules logiciels robotiques.

Orca [1]. est un intergiciel open-source pour le développement de systèmes à base de composants. Il fournit des mécanismes pour créer des blocs de construction qui peuvent être assemblés pour former des systèmes robotiques complexes.

Pour mettre en œuvre un système à base de composants distribués, CORBA a été choisi dans la première version d'Orca, mais a été rapidement modifiée avec ICE [27], une nouvelle approche d'intergiciel orientée objet.

Les plate-formes qui peuvent exécuter l'intergiciel Orca sont : MobileRobots, Segway, K-Team robots, iRobot's RFLEX-based et Evolution Robotics.

Pyro [34]. (Python Robotics) a pour objectif de fournir un environnement de programmation pour explorer facilement des sujets avancés de l'intelligence artificielle et de la robotique, sans avoir à se soucier des détails de bas niveau du matériel sous-adjacent ni de l'environnement de programmation de robot. Il incorpore l'intergiciel Player/Stage, ce qui permet de réutiliser des composants développés pour ce dernier.

Il existe de nombreuses bibliothèques pour Pyro qui fournissent des services robotiques spécifiques. L'intergiciel est compatible avec MobileRobots Pioneer, Sony Aibo et tous les robots pris en charge par Player/Stage.

Intergiciel	*Système d'exploitation*	*Services de stockage de données durable*	*Tolérance aux pannes*
`Player/Stage`	Linux, Windows	$\sim$	+
`ROS`	Des dépôts : Ubuntu, Debian. From source : generic Linux, Windows, MacOS	**+** (Rosbags)	+
`Miro`	Linux	$\sim$	+
`MRDS`	Windows	$\sim$	+
`Marie`	Linux	$\sim$	+
`Orca`	Linux	$\sim$	+
`Pyro`	Linux	$\sim$	− (Ni mode dégradé, ni isolement du composant)

Table 1: L'architecture

4. LES CRITÈRES COMPARATIFS

Nous allons comparer les sept intergiciels robotiques présentés ci-dessus d'un point de vue génie logiciel, car le concept d'intergiciel est apparu dans ce domaine et qu'il fournit de nombreuses caractéristiques qui peuvent être transférées aux applications robotiques. Nous avons regroupé les critères de comparaison en trois grands groupes : *architecture*, *infrastructure* et *usage*. Chaque groupe est composé de différents critères présentés ci-dessous.

L'architecture évalue l'impact qu'a l'intergiciel sur le système d'exploitation hôte, qui se décompose en :

Système d'exploitation - la dépendance du système d'exploitation. Ce critère exprime la portabilité d'un système au travers de multiples plate-formes et systèmes.

Services de stockage de données durable - des outils qui permettent de conserver les données de capteurs et d'autres robots de la flotte. La couche de persistance des données est importante pour la sauvegarde des résultats de la mission, pour les validations de données expérimentales, pour le traitement de données hors ligne ainsi que pour rejouer les données dans un simulateur.

Tolérance aux pannes - elle correspond à la détection d'une panne de logiciel, d'un fonctionnement en mode dégradé et à l'exécution de processus de récupération. Le fait qu'un intergiciel puisse détecter des défaillances est essentiel pour les applications robotiques. En outre, il est important que les robots continuent d'effectuer leurs tâches dans un mode dégradé jusqu'à ce que le système répare la défaillance.

L'infrastructure évalue les outils et les API fournies par l'intergiciel, qui se décompose en :

Processus de gestion et de surveillance - des outils qui permettent de gérer, de déboguer, de configurer et de surveiller les composants de l'intergiciel. Il est important de faciliter la tâche de surveillance en offrant une vision complète des capteurs, des actionneurs et du statut des autres robots.

Services de coordination multi-robot - outils pour le partage de données entre robots et de distribution des tâches. Il est important de disposer d'outils de gestion de distribution des algorithmes afin de réduire la complexité du développement.

Communication - La communication est très importante entre les différents composants d'un robot afin de lui permettre d'effectuer avec succès sa tâche, ainsi qu'à l'intérieur d'une flotte de manière à permettre à des robots d'interagir avec les autres.

L'usage évalue l'apport de l'intergiciel pour la définition de nouvelles applications, qui se décompose en :

Déploiement et cycle de vie - la possibilité de déployer des comportements/ configurations sur l'ensemble de la flotte, d'intégrer les chaînes de compilation et la gestion du cycle de vie pour de nouvelles applications. Il est très utile d'avoir une système de compilation et un environnement de test automatisé pour simplifier le déploiement et la définition de tâches distribuées.

Modèle de programmation - les types de programmation disponibles : synchrone [4], asynchrone [5], etc. Le fait d'avoir des approches différentes pour les modèles de programmation permet d'employer simultanément différentes techniques selon les problèmes traités.

Services d'intégration du code et des données - facilite l'intégration de nouveaux services et de modules via des APIs dans le logiciel embarqué. La présence d'interfaces permet au développeur d'enrichir les applications et de faciliter le développement.

5. COMPARAISON DES INTERGICIELS

Cette section analyse chaque intergiciel selon les critères présentés dans la section précédente. Chaque grand groupe est commenté dans un paragraphe. L'évaluation des critères est notée comme suit : un **+** représente que toutes les exigences du critère sont satisfaites, un + représente que la plupart des exigences sont présentes, un $\sim$ montre le fait que le critère n'est satisfait que partiellement et un − représente une absence des exigences.

5.1 L'architecture

Le Tableau 1 synthétise le groupe *Architecture*. Il est composé des critères : *système d'exploitation*, *services de stockage de données durable* et *tolérance aux pannes*.

Système d'exploitation.

Les robots composants une flotte peuvent avoir des systèmes d'exploitation hétérogènes, donc ce critère est très important dans le

4. Le modèle de programmation synchrone est utilisé pour effectuer des tâches d'exécution séquentielles bloquantes

5. Le modèle de programmation asynchrone est utilisé pour la programmation de systèmes interactifs qui interagissent en permanence avec leur environnement, à leur propre rythme.

L'intergiciel	*Processus de gestion et de surveillance*	*Services de coordination multi-robot*	*Communication*
`Player/Stage`	$\sim$	+ (Coordination par tiers)	$\sim$
`ROS`	**+** (Gestion et surveillance)	$\sim$	**+** (Sync. & async.)
`Miro`	– (Aucun)	$\sim$	$\sim$
`MRDS`	**+** (Visual Studio plugins)	$\sim$	**+** (Sync. & async.)
`Marie`	$\sim$	$\sim$	$\sim$
`Orca`	– (Aucun)	$\sim$	$\sim$
`Pyro`	$\sim$	$\sim$	$\sim$

Table 2: L'infrastructure

choix d'un intergiciel. A part *MRDS* qui ne fonctionne que sur Windows, les autres intergiciels fonctionnent sous Linux. *Player/Stage* et *ROS* sont eux multi-plateforme.

Services de stockage de données durables.
ROS est le seul intergiciel qui fournit des services de stockage de données durables. Tous les messages produits peuvent être conservées dans le nœud rosbags. Les autres intergiciels ne fournissent aucune API native pour enregistrer les informations des capteurs.

Tolérance aux pannes.
Aucun intergiciel ne dispose d'un mode dégradé ni d'un système de redémarrage automatique des processus après un échec. A part *Pyro*, tous les intergiciels offrent une isolation des composants [6]. *ROS* a besoin d'une adresse IP à l'initialisation pour exécuter le nœud de démarrage roscore. Une fois que tous les nœuds sont lancées, l'échec de roscore n'affectera pas les autres nœuds.

5.2 L'infrastructure

Le Tableau 2 résume le groupe *Infrastructure*. Il est composé de : *processus de gestion et de surveillance*, *services de coordination multi-robot* et *communication*.

Processus de gestion et de surveillance.
À part *Miro* et *Orca* qui ne fournissent ni une surveillance ni une interface de gestion, les autres intergiciels disposent d'un logiciel de surveillance graphique. *MRDS* utilise Visual Studio comme IDE. *ROS* a plusieurs outils de gestion et un tableau de bord graphique QT.

Services de coordination multi-robot.
Aucun des intergiciels considérés ne fournit de services natifs de coordination multi-robot. *Player/Stage* dispose d'algorithmes de coordination externe développés pour lui. *ROS*, *Miro*, *MRDS*, *Orca*, *Marie* et *Pyro* délèguent les services de coordination à la couche d'application.

Communication.
La communication entre les couches de l'infrastructure dans *Player/Stage* et *Pyro* est faite en utilisant des connexions socket directes. Le partage des données dans *Miro* est affecté à la IIOP [7] de CORBA. *Marie* utilise une mémoire partagée et des sockets. *MRDS* et *ROS* disposent de communications à la fois synchrones et asynchrones.

5.3 Usage

Le Tableau 3 résume le groupe *Usage*. Il est composé de : *déploiement et cycle de vie*, *modèle de programmation* et *services d'intégration du code et des données*.

Déploiement et cycle de vie.
Aucun des intergiciels ne fournit un système de déploiement multi-robots. *ROS* ne possède pas de système de référentiel de déploiement mais dispose d'une chaîne de compilation a base de CMake appelé Catkin. Il utilise le simulateur Gazeboo comme environnement de test. *Miro* a un compilateur IDL, qui permet de générer le code pour la communication entre l'intergiciel et les services sous-adjacents. Il utilise Stage et Gazeboo pour les simulations. *MRDS* utilise Visual Studio comme IDE qui fournit une chaîne de compilation, un outil de déploiement, ainsi qu'un simulateur. *Orca* utilise la compilation CMake et fournit un simulateur graphique pour les tests. *Pyro* fournit plusieurs simulateurs pour les tests de code : Stage, Gazeboo et Khepera mais il ne dispose pas d'un outil de déploiement. *Player/Stage* n'a pas de chaîne de compilation native mais il existe des chaînes de compilation dans les IDE pour compiler le code source de l'application. Il fournit un environnement de test dans Stage. *Marie* n'a pas d'outil de compilation spécifique ou de système de déploiement.

Modèle de programmation.
Les applications pour *Player/Stage* peuvent être écrites dans n'importe quel langage de programmation. *Pyro* est basé sur Python et ses applications peuvent être écrites en Python. Les applications *Miro* peuvent être écrites dans tous les langages fournissant des implémentations de CORBA. Les échanges de données sont déclenchés par un événement. *MRDS* utilise un langage de programmation visuel, un environnement de développement graphique qui utilise un catalogue de services et d'activités. Le langage de pro-

6. L'isolement de composants ou de sand-boxing est un mécanisme de sécurité pour séparer les processus en cours d'exécution. Les espaces des codes et de données sont séparés pour chaque processus.

7. Le protocole IIOP (Internet Inter-ORB Protocol) est le protocole de communication utilisé par CORBA. C'est une implémentation s'appuyant sur un transport TCP/IP du protocole de plus haut-niveau GIOP (General Inter-ORB Protocol).

Table 3: L'usage

L'intergiciel	*Déploiement et cycle de vie*	*Modèle de programmation*	*Services d'intégration du code et des données*
Player/Stage	∼	∼	+
ROS	+ (Catkin)	+ (Async et sync)	+ (roslaunch, rosrun)
Miro	+ (IDL compiler, Gazeboo)	− (CORBA)	∼
MRDS	+ (Visual Studio)	+ (C#)	+ (VPL)
Marie	− (Aucun)	∼	+ (compatible Player)
Orca	+ (CMake)	+ (Multiple langages de programmation)	∼
Pyro	+ (Gazeboo)	∼	+

grammation principal dans MRDS est C#. *Orca* supporte essentiellement C/C++ sous Linux, mais il peut être utilisé en développant en Java, Python, PHP, C# et Visual Basic. *ROS* supporte à la fois des modèles de programmation synchrones et asynchrones. Les applications peuvent être écrites nativement en Python et C++ , mais il existe des intégrations pour Java, LISP et d'autres langages. *ROS* reçoit la meilleure évaluation en raison de la variété des langages et des modèles de programmation dont il dispose.

Services d'intégration du code et des données.
Tous les intergiciels examinés supportent une architecture modulaire et permettent une intégration facile ou une réutilisation du code. *Miro* fournit des abstractions de service des capteurs et des actionneurs via le langage de définition d'interface (IDL) CORBA. *Orca* maximise la réutilisation des logiciels et la modularité pour les applications robotiques. *MRDS*, avec l'utilisation de VPL [8], permet de générer le code de nouvelles "macro" de services à partir des diagrammes créés par les utilisateurs. Un service ou une activité est représentée par un bloc avec des entrées et des sorties qui a juste besoin d'être glissé du catalogue au diagramme. *Marie* fournit des services de traduction tels que les composants écrits pour Player / Stage peuvent être utilisé. *Pyro* supporte des modules créés pour Player / Stage. *ROS* a un système de packages bien conçu et un système du lancement gérant les dépendances.

En synthèse des tableaux établis ci-dessus, il apparaît que ROS semble être l'intergiciel le plus approprié pour les systèmes multi-robots, suivis par MRDS. Les deux satisfont totalement ou presque les différents critères proposés. Dans la section suivante, nous présentons une autre approche émergente, le cloud pour les robots.

6. PLATEFORMES DE CLOUD ROBOTIQUE EXISTANTES

En parallèle du développement d'intergiciels pour la robotique, une autre vision du génie logiciel émerge dans le contexte des flottes robotiques : le cloud. De façon générale, un cloud robotique suppose la présence de robots, d'objets communicants et d'autres éléments d'infrastructure qui partagent des informations et des ressources de manière transparente pour le développeur. Le principal avantage, en dehors du partage d'informations, est l'apport en puissance de calcul de l'infrastructure et la distribution des processus vers les différents robots selon leur capacités. Nous présentons ci-après les deux plate-formes de cloud robotique DAvinCi [3] et Rapyuta [18]. Les deux utilisent l'intergiciel ROS.

6.1 DAvinCi

DAvinCi [3] est un système qui fournit des avantages tels que l'évolutivité, le parallélisme des tâches et le partage de l'infrastructure. Il peut être utilisé dans des scénarios où il existe de multiples robots exécutant des tâches en parallèle. Les données collectées sont fusionnées dans un environnement de cloud computing puis renvoyées à chaque robot.

DAvinCi est basé sur ROS comme moyen de messagerie entre robots et sur Hadoop [45] pour la composante cloud. L'hypothèse faite par la plate-forme est que chaque robot dispose d'une communication WiFi et que l'infrastructure comporte un point d'accès WiFi centralisé qui sert de passerelle entre la flotte robotique et les services de cloud computing.

Nous pouvons noter la forte dépendance à l'infrastructure cloud pré-existante et l'accès Wi-Fi centralisé qui réduit fortement les possibilités de déploiement. En outre, le besoin d'envelopper des messages binaires ROS dans des requêtes HTTP induit une surcharge notable en termes de trafic.

6.2 Rapytua

Rapytua [18] est une plate-forme de services open source qui offre un environnement cloud personnalisable. Il est également connu comme le "Cloud Engine de RoboEarth" car il facilite l'accès des robots au référentiel de connaissances RoboEarth [36].

Rapytua est composé d'un modèle de calcul élastique qui alloue dynamiquement des environnements informatiques pour les robots. Cela permet aux robots de partager des services et des informations. Les environnements informatiques sont mis en œuvre en utilisant Linux Container. La plate forme est compatible avec ROS et elle bénéficie des protocoles de communication identique au ROS, permettant à tous les paquets ROS d'être exécutés directement sans une adaptation préalable.

On peut remarquer la surcharge de l'utilisation de JSON [9] pour sérialiser les messages binaires ROS et les dépendances d'un accès

8. Visual Programming Language

9. JavaScript Object Notation

centralisé WiFi. D'autre part, la plate-forme n'est pas liée à une plate-forme de cloud spécifique ce qui rend le système portable.

7. DISCUSSION

Concernant l'analyse des intergiciels susceptibles d'être utilisés dans un contexte multi-robot, nous pensons que ROS est celui qui a le plus fort potentiel pour devenir un logiciel de référence. Il présente toutefois encore des manques dans cette direction puisqu'il ne dispose ni d'un système de coordination / déploiement multi-robot, ni d'un environnement de test automatisé. En revanche, il bénéficie d'une large communauté qui développe de nouveaux paquets robotiques pour lui. Un autre élément clé de ROS est son mécanisme de communication. Il supporte les communications synchrones et asynchrones et peut facilement être personnalisé avec de nouveaux types de messages. Il a une grande base de données de pilotes permettant une bonne abstraction de la couche matérielle. De nouveaux modules et paquets peuvent être développés et intégrés rapidement. Il est très permissif du point de vue du développement puisqu'il permet de travailler avec différents langages de programmation.

Nous avons vu que de nouvelles infrastructures logicielles réparties émergent, comme le concept de cloud pour les robots. Il permet aux robots de communiquer avec l'infrastructure de cloud externe et de déporter les opérations informatiques lourdes ainsi que d'interagir avec l'Internet des objets. L'inconvénient de ce nouveau concept est l'infrastructure de communication qui suppose un accès Wi-Fi centralisé. Cela réduit les cas d'utilisation puisque de nombreuses applications robotiques se font en environnements non contrôlés, c'est-à-dire sans une infrastructure de communication.

Nous considérons qu'une flotte de robots doit être organisée comme un cloud de robots et non comme un cloud pour les robots. Il existe un besoin de distribution des tâches d'un problème complexe ou lourd sur plusieurs robots. Notre vision est de transférer les avantages d'un environnement cloud au contexte des flottes robotiques en permettant à des robots (hétérogènes) de partager des informations, des ressources, et de la puissance de calcul. Le robot est alors considéré comme une machine dans le nuage. Nous visons à utiliser ROS et/ou MRDS comme une couche de communication entre les composants internes des robots. Nous voulons développer un intergiciel capable de gérer la flotte, de faire de la coordination multi-robot, de partager des ressources et simplifier la parallélisation des tâches.

Dans cette perspective, nous devons prendre en compte la problématique induite par un système de cloud, qui suppose que les communications sont fiables, pour l'appliquer à des environnements de communication instables à cause de la mobilité des robots. L'intergiciel doit pouvoir fonctionner dans une infrastructure à la fois centralisée et ad hoc. De plus, nous devons prendre en considération les modes dégradés lorsque des robots sont isolés du reste de la flotte (en terme de communication). Une approche que nous visons à développer sera d'effectuer la tâche de façon indépendante et de rejoindre la flotte lorsque les conditions de fonctionnement reviennent à la normale.

8. CONCLUSION

Dans cet article, nous avons présenté les défis que les intergiciels robotiques doivent relever pour répondre au contexte des systèmes multi- robots. Nous avons examiné sept intergiciels très utilisés en robotique et nous les avons comparés en définissant les principales exigences de tels systèmes, en terme de génie logiciel. Ainsi nous avons montré que leurs capacités de robustesse, de modularité logicielle et de communication sont très variables. Cet état de l'art montre qu'aucun des intergiciels existants ne répond complètement au besoin des systèmes multi-robots, en particulier en terme de services de coordination. Toutefois les solutions proposées par *ROS* et *MRDS* apparaissent comme potentiellement les plus adaptables.

Par ailleurs, nous avons présenté deux solutions de cloud computing pour la robotique tout en soulignant leurs inconvénients. En perspective de cette analyse, nous proposons de développer une vision différente qui consiste à définir un cloud de robots plutôt qu'un cloud pour les robots. Dans ce cadre, nous avons commencé à travailler sur une évolution de *ROS* qui apparaît dans notre analyse comme l'intergiciel le plus adapté aux systèmes multi-robots.

9. REFERENCES

[1] A. B. Alexei Makarenko and T. Kaupp. On the benefits of making robotic software frameworks thin. In *POn the Benefits of Making Robotic Software Frameworks Thin IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'07), San Diego CA, USA, 29 Oct. - 02 Nov. 2007*, 2007.

[2] N. Ando, T. Suehiro, and T. Kotoku. A software platform for component based rt-system development : Openrtm-aist. In S. Carpin, I. Noda, E. Pagello, M. Reggiani, and O. Stryk, editors, *Simulation, Modeling, and Programming for Autonomous Robots*, volume 5325 of *Lecture Notes in Computer Science*, pages 87–98. Springer Berlin Heidelberg, 2008.

[3] R. Arumugam, V. R. Enti, L. Bingbing, W. Xiaojun, K. Baskaran, F. F. Kong, A. S. Kumar, K. D. Meng, and G. W. Kit. Davinci : A cloud computing framework for service robots. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 3084–3089. IEEE, 2010.

[4] P. A. Baer, R. Reichle, and K. Geihs. The Spica Development Framework – Model-Driven Software Development for Autonomous Mobile Robots. In W. Burgard, R. Dillmann, C. Plagemann, and N. Vahrenkamp, editors, *IAS-10*, pages 211–220. IAS Society, Proceedings of the 10th International Conference on Intelligent Autonomous Systems, 2008.

[5] D. J. Bruemmer, D. A. Few, M. C. Walton, and C. W. Nielsen. The robot intelligence kernel. In *AAAI*, 2006.

[6] H. Bruyninckx. Simulation, modeling and programming for autonomous robots : The open source perspective. In S. Carpin, I. Noda, E. Pagello, M. Reggiani, and O. Stryk, editors, *Simulation, Modeling, and Programming for Autonomous Robots*, volume 5325 of *Lecture Notes in Computer Science*, pages 1–1. Springer Berlin Heidelberg, 2008.

[7] D. Calisi and A. Censi. Robotic software development and interoperability using the openrdk framework. In *ICAR'09 Workshop on "Rapid Application Development in Robotics : On the role of re-use and adaptation of system components, middleware, and control architectures"*, Munich, Germany, June 2009.

[8] D.-H. Choi, S.-H. Kim, K.-K. Lee, B.-H. Beak, and H.-S. Park. Middleware architecture for module-based robot. *SICE-ICASE International Joint Conference*, 0 :4202–4205, 2006.

[9] C. Côté, Y. Brosseau, D. Létourneau., C. Raïevsky, Y. Brosseau, and F. Michaud. Using marie for mobile robot

component development and integration. *Software Engineering for Experimental Robotics Book Series Springer Tracts in Advanced Robotics Publisher Springer Berlin / Heidelberg*, 30/2007 :211–230, April 2007.

[10] A. Elkady and T. Sobh. Robotics middleware : A comprehensive literature survey and attribute-based bibliography. *Journal of Robotics*, 2012, 2012.

[11] Y. Endo, D. MacKenzie, and R. Arkin. Usability evaluation of high-level user assistance for robot mission specification. *IEEE Transactions on Systems, Man, and Cybernetics*, 34 :168–180, 2004.

[12] ERSP. Ersp 3.1 software development kit. Online : http ://www.evolution.com/products/ersp/, 2010.

[13] J. Ferber. *Multi-Agent Systems. An Introduction to Distributed Artificial Intelligence.* Addison Wesley, London, 1999.

[14] J. Fernandez-Madrigal, C. Galindo, and J. Gonzalez. Integrating heterogeneous robotic software. In *Electrotechnical Conference, 2006. MELECON 2006. IEEE Mediterranean*, pages 433 –436, July 2006.

[15] P. Fitzpatrick, G. Metta, and L. Natale. Towards long-lived robot genes. *Robot. Auton. Syst.*, 56(1) :29–45, 2008.

[16] S. Frénot, F. Le Mouël, J. Ponge, and G. Salagnac. Various Extensions for the Ambient OSGi framework. In *Adamus Workshop in ICPS*, Berlin, Allemagne, July 2010.

[17] F. Heckel, T. Blakely, M. Dixon, C. Wilson, and W. D. Smart. The wurde robotics middleware and ride multi-robot tele-operation interface. 2006.

[18] D. Hunziker, M. Gajamohan, M. Waibel, and R. D'Andrea. Rapyuta : The roboearth cloud engine. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 438–444. IEEE, 2013.

[19] iRobotware. Aware 2 robot intelligient software. Online : http ://www.irobot.com/gi/developers/Aware/, 2010.

[20] V. Issarny, M. Caporuscio, and N. Georgantas. A perspective on the future of middleware-based software engineering. In *Future of Software Engineering, 2007. FOSE '07*, pages 244–258, May 2007.

[21] C. Jang, S.-I. Lee, S.-W. Jung, B. Song, R. Kim, S. Kim, and C.-H. Lee. Opros : A new component-based robot software platform. *ETRI Journal*, 2010.

[22] K. Johns and T. Taylor. *Professional Microsoft Robotics Developer Studio*. Wrox Press Ltd., Birmingham, UK, UK, 2008.

[23] G. K. Kraetzschmar, H. Utz, S. Sablatnög, S. Enderle, and G. Palm. Miro - middleware for cooperative robotics. In *RoboCup 2001 : Robot Soccer World Cup V*, pages 411–416, London, UK, 2002. Springer-Verlag.

[24] M. Kranz, R. B. Rusu, A. Maldonado, M. Beetz, and A. Schmidt. A player/stage system for context-aware intelligent environments. 2006.

[25] S. Magnenat, P. Retornaz, M. Bonani, V. Longchamp, and F. Mondada. Aseba : A modular architecture for event-based control of complex robots. *Mechatronics, IEEE/ASME Transactions on*, PP(99) :1 –9, 2010.

[26] O. Michel. Cyberbotics ltd. webots tm : Professional mobile robot simulation. *Int. Journal of Advanced Robotic Systems*, 1 :39–42, 2004.

[27] M. S. Michi Henning. Distributed programming with ice. http ://www.zeroc.com/doc/Ice-3.4.0/manual/, February 2010.

[28] N. Mohamed, J. Al-Jaroodi, and I. Jawhar. A review of middleware for networked robots. *International Journal of Computer Science and Network Security*, 9(5) :139–148, 2009.

[29] M. Montemerlo, N. Roy, and S. Thrun. Perspectives on standardization in mobile robot programming : The carnegie mellon navigation (carmen) toolkit. In *In Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)2003*, pages 2436–2441, 2003.

[30] MRPT. The mobile robot programming toolkit. Online : http ://www.mrpt.org/, 2010.

[31] I. A. Nesnas, R. Simmons, D. Gaines, C. Kunz, A. Diaz-Calderon, T. Estlin, R. Madison, J. Guineau, M. McHenry, I.-H. Shu, and D. Apfelbaum. Claraty : Challenges and steps toward reusable robotic software. *International Journal of Advanced Robotic Systems*, 2006.

[32] openJaus. Openjaus. http ://www.openjaus.com/, 2010.

[33] S. Petters, D. Thomas, and O. von Stryk. Roboframe - a modular software framework for lightweight autonomous robots. In *Proc. Workshop on Measures and Procedures for the Evaluation of Robot Architectures and Middleware of the 2007*, San Diego, CA, USA, Oct. 29 2007.

[34] Pyro. Website. http ://pyrorobotics.com/, 2012.

[35] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng. Ros : an open-source robot operating system. In *ICRA workshop on open source software*, volume 3, 2009.

[36] RobotEarth. http ://roboearth.org/, 2014.

[37] ROS. Robot operating system. http ://www.ros.org/, 2014.

[38] C. Schlegel, T. Hassler, A. Lotz, and A. Steck. Robotic software systems : From code-driven to model-driven designs. In *Advanced Robotics, 2009. ICAR 2009. International Conference on*, pages 1 –8, 22-26 2009.

[39] D. Simon, B. F. Pissard-Gibollet, R., and S. Arias. Orccad, robot controller model and its support using eclipse modeling tools. In *5th National Conference on "Control Architecture of Robots"(2010) CAR'10*, 2010.

[40] Skilligent. Skilligent. Online : http ://www.skilligent.com/index.shtml, 2010.

[41] W. D. Smart. Is a common middleware for robotics possible ? In *Proceedings of the IROS 2007 workshop on Measures and Procedures for the Evaluation of Robot Architectures and Middleware*. Citeseer, 2007.

[42] D. R. O. System. http ://dros.org/.

[43] M. Tenorth, A. C. Perzylo, R. Lafrenz, and M. Beetz. The roboearth language : Representing and exchanging knowledge about actions, objects, and environments. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 1284–1289. IEEE, 2012.

[44] D. Valle, E. Nuno, L. Basañez, and N. Arana-Daniel. Consensus of networks of nonidentical robots with flexible joints, variable time-delays and immeasurable velocities. In *IROS*, pages 5878–5883, 2013.

[45] G. Xu, F. Xu, and H. Ma. Deploying and researching hadoop in virtual machines. In *Automation and Logistics (ICAL), 2012 IEEE International Conference on*, pages 395–399, Aug 2012.

[46] J. young Kwak, J. Y. Yoon, and R. Shinn. An intelligent robot architecture based on robot mark-up languages. In *Engineering of Intelligent Systems, 2006 IEEE International Conference*, pages 1 –6, 0-0 2006.