ROS : une solution middleware adaptée pour l'industrie du futur ?

Cyril Briand, Olivier Stasse, Michel Taïx, Thierry Germa, Fabien Marco, Nathan Moret

ROS : une solution middleware adaptée pour l'industrie du futur ?

Cyril BRIAND
LAAS-CNRS,
Université de Toulouse, UPS,
Toulouse, France
cyril.briand@laas.fr

Thierry GERMA
Magellium,
Université de Toulouse, UPS,
Toulouse, France
thierry.germa@gmail.com

Fabien MARCO
S-MART AIP-PRIMECA Occitanie,
Université de Toulouse, UPS,
Toulouse, France
fabien.marco@univ-tlse3.fr

Nathan MORET
S-MART AIP-PRIMECA Occitanie,
Université de Toulouse, UPS,
Toulouse, France
nathanp.moret@gmail.com

Olivier STASSE
LAAS-CNRS,
Université de Toulouse
Toulouse, France
ostasse@laas.fr

Michel TAÏX
LAAS-CNRS,
Université de Toulouse, UPS,
Toulouse, France
michel.taix@laas.fr

Résumé— *Depuis 6 ans, le pôle S-MART Occitanie a fait le choix d'utiliser le middleware Robot Operating System (ROS) pour intégrer les équipements de sa plateforme robotique et pour former les étudiants des diverses formations aux diverses facettes de la robotique. Cet article relate cette expérience et, au-delà, s'interroge sur le potentiel de ROS en tant que solution open-source pour l'accélération de la transition vers l'industrie du futur.*

Mots-clés— *Middleware, ROS, Jumeaux Numériques, Industrie du futur.*

I. INTRODUCTION

La robotique est considérée comme un pilier majeur de la 4ème révolution industrielle [1]. Les avancées récentes en robotique ouvrent en effet de nouvelles perspectives en rupture avec le modèle industriel traditionnel dans lequel des robots manipulateurs exécutent à haute cadence des tâches pré-programmées en environnement contrôlé. Même si ce modèle reste tout à fait pertinent aujourd'hui, les progrès récents dans la conception mécanique des robots, les technologies de perception, les algorithmes de traitement des données basés sur l'apprentissage, les algorithmes de planification de mouvements, etc. permettent de mettre en œuvre un nouveau modèle dans lequel les robots sont désormais capables de se mouvoir de manière autonome et sûre dans leur environnement de travail, en présence d'obstacles dynamiques, pour la réalisation de tâches diverses et complexes. Ces nouvelles facultés peuvent être exploitées pour développer des applications innovantes telles que l'assemblage robotisé de grandes structures [2], le contrôle visuel de leur conformité [3,4], la co-exécution de tâches par/avec un humain [5], la logistique basée sur l'utilisation de flottes de robots mobiles ou de drones [6,7], etc.

Innover repose en grande part sur la capacité à intégrer avec succès différentes briques technologiques et prototyper rapidement les applications en prenant en compte les diverses contraintes liées au respect des contraintes temporelles de l'application, au partage rapide de volume conséquent de données, à la sûreté de fonctionnement, à la qualité logicielle, etc. Les middlewares ont été conçus pour répondre à cette problématique d'intégration en permettant de s'abstraire des couches logicielles basses, rendant ainsi les applications plus claires, modulaires et maintenables [8].

Parmi les solutions middleware, l'usage de ROS (Robot Operating System) est de plus en plus répandu. ROS est le premier projet collaboratif open-source de robotique à grande échelle. Né en 2000 à l'Université de Stanford, puis porté à maturité par la startup Willow-Garage en 2007, ROS fournit aujourd'hui un ensemble de librairies, d'outils et de concepts informatiques permettant le développement de systèmes robotiques complexes. L'écosystème ROS comporte aujourd'hui des dizaines de milliers d'utilisateurs dans le monde et est soutenu par des acteurs industriels majeurs.

Cet article présente tout d'abord les principales caractéristiques de ROS. Les différents avantages et inconvénients de son utilisation sont ensuite discutés au travers un retour d'expérience de deux applications réalisées au sein du pôle S-MART Occitanie : la réalisation d'un jumeau numérique dédié à la contrôle/commande d'une cellule flexible de production, l'intégration de robots propriétaires dans l'environnement ROS. Une conclusion dresse un bilan et évoque les perspectives d'évolution.

II. LE MIDDLEWARE ROS EN ROBOTIQUE

A. *Pourquoi un middleware ?*

Un système robotique est un ensemble de systèmes mettant en jeu de nombreuses compétences relatives à l'électrotechnique, la mécanique, l'automatique, la vision, le son et l'informatique. Chacun de ces domaines étant en constante évolution, il est nécessaire de capitaliser l'expérience, de réutiliser les outils mis en place et de coordonner/synchroniser efficacement les différents sous-systèmes. Les middlewares permettent de répondre à cette problématique et de réduire les temps et les coûts de production en :

- facilitant la gestion de la complexité et de l'hétérogénéité des applications,
- simplifiant la conception logicielle,
- masquant la complexité de la communication de bas niveau et l'hétérogénéité des capteurs,
- favorisant la réutilisation de l'infrastructure logicielle qui capitalise des années d'efforts de recherche.

B. Le middleware ROS

ROS est un middleware fonctionnant exclusivement dans l'environnement Linux Ubuntu. Il s'appuie sur une *plomberie* efficace qui permet la communication entre des composants logiciels quelle que soit leur répartition au sein de l'architecture distribuée. Ces mécanismes de communication sont illustrés sur la figure 1. Les processus de ROS sont appelés *nœud*. Les nœuds communiquent entre eux avec des messages via des *topics*. Quand un nœud veut publier un message sur un topic, il avertit dans un premier temps le *master* puis, dans un second temps, le publie, il est alors appelé *publisher*. Lorsqu'un nœud veut consulter un message, il doit s'abonner au topic correspondant. Pour cela il avertit le master de sa volonté de souscrire à un topic et le master crée la connexion. Ce nœud est alors appelé *suscriber*. Le suscriber a accès au message du topic via une fonction *callback*, c'est-à-dire une fonction appelée à chaque fois qu'un message arrive. Un nœud peut être à la fois suscriber et publisher. ROS présente donc une structure de communication entre les processus de type publisher / suscriber permettant d'échanger des messages de type différent. Cette architecture lui confère une grande modularité et un caractère d'interopérabilité lui permettant d'interagir avec des systèmes robotiques différents.

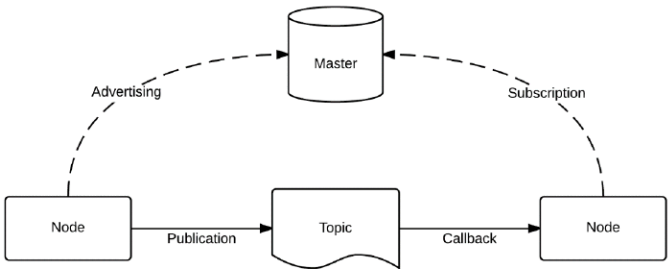

Figure 1. Schéma du protocole de communication de ROS

ROS dispose également d'une suite intégrée d'outils de développement permettant d'analyser, afficher et débugger une application répartie complexe :

- catkin : un système de gestion de paquets, de génération de code automatique et compilation .
- rqt_graph : une interface graphique permettant d'analyser le graphe d'applications et les transferts de données via les topics.
- rosbag : un programme permettant d'enregistrer et de rejouer des séquences topics (la fréquence, la durée et les topics à enregistrer pouvant être spécifiés).
- rqt : une interface de contrôle incrémentale qui se base sur le système de plugins de la bibliothèque de GUI Qt.

Une grande force de ROS réside dans le niveau d'abstraction du matériel utilisé. Cela permet de limiter fortement la portée des modifications en cas d'un changement de matériel. Cette abstraction permet de coupler son application à des librairies puissantes telles que :

- OpenCV pour le traitement du signal,
- OctoMap pour la représentation de l'environnement,
- SMACH pour la planification de tâches,
- MoveIt pour la planification de mouvement,
- ROS control pour le contrôle des boucles de commande,
- Rviz pour l'affichage des modèles de robots, des cartes de navigation.

ROS permet également, via des outils de simulation comme Gazebo (http://gazebosim.org - totalement intégré à ROS) ou CoppeliaSim (https://www.coppeliarobotics.com/), de valider une application de commande. Gazebo et CoppeliaSim sont tous deux des simulateurs 3D, cinématiques, dynamiques et multi-robots, permettant de simuler des robots articulés dans des environnements complexes, intérieurs ou extérieurs, réalistes. Il est ainsi possible de valider le comportement d'une application contrôlant un ou plusieurs robots en interaction au sein d'un processus d'assemblage ou de fabrication et, grâce à l'abstraction offerte par ROS, d'exécuter ensuite cette même commande sur le système physique réel, cela sans modifier aucune ligne de code.

III. RETOURS D'EXPERIENCE DU POLE S-MART OCCITANIE

Depuis 6 ans, le pôle S-MART Occitanie a fait le choix d'utiliser ROS pour les formations liées à la robotique mobile. L'idée d'utiliser ce middleware pour intégrer complètement les ressources de la plateforme de robotique industrielle a germé il y a 4 ans. Plusieurs projets étudiants et stages ont été réalisés pour valider le principe dont les chapitres suivants résument les résultats.

A. Présentation succincte de la plateforme robotique industrielle

La cellule flexible (cf. Fig. 2) est un démonstrateur d'une chaîne de production industrielle composée de quatre postes de travail robotisés ; chacun équipé d'un bras manipulateur et d'un système de vision. Ces postes de travail sont connectés entre eux par un système de transport. Il s'agit d'une ligne transitique Montrac commercialisée par la société Allemande Montratec. Elle est composée d'un circuit monorail sur lequel circulent des navettes. Ce système est commandé par des automates programmables industriels Schneider Modicon M340. La ligne permet de transporter des pièces manufacturées d'un poste de travail à l'autre pour leur faire subir différentes opérations industrielles.

Figure 2. Photo de la cellule flexible

Dans le cadre de ce travail, seulement deux postes robotisés sont considérés. Le premier comporte un bras manipulateur RX60L du constructeur Stäubli et un système de vision à deux caméras Cognex (cf. Fig. 3). L'autre est composé du même système de vision et d'un bras manipulateur KR6 R700 du constructeur Kuka (cf. Fig. 4). Aucun de ces robots n'est compatible nativement avec l'environnement ROS.

Figure 3. Bras robot RX60L et son contrôleur CS8

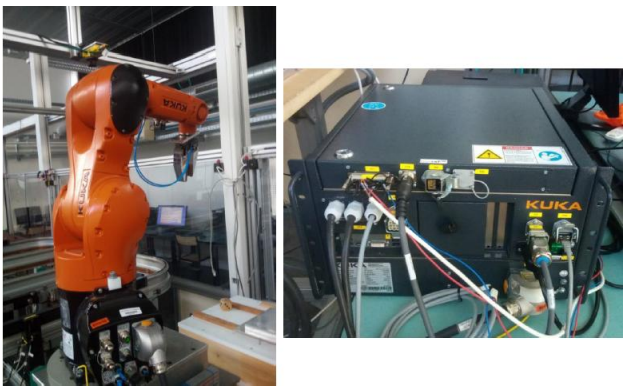

Figure 4. Bras robot Kuka KR6 R700 et son contrôleur KR C4 compact

B. *Intégration des robots Stäubli et Kuka dans l'environnement ROS*

Pour chacun des deux robots, l'objectif est de réaliser trois preuves de concept (POC) à complexité incrémentale pour démontrer l'intérêt de ROS pour commander des robots industriels en utilisant une grande part de leur potentiel. ROS permettrait ainsi de s'affranchir des environnements propriétaires des constructeurs. Ces POCs ont été choisies de façon à être représentatives de cas typiques industriels. Le POC n°1 est une tâche de pick and place classique, très fréquente dans l'industrie, consistant à prendre un objet à un endroit pour le déplacer à un autre endroit avec une certaine orientation. Cette tâche est facile à mettre en œuvre et permet d'utiliser différents types de mouvement du robot comme un mouvement linéaire, circulaire ou encore d'approche. De plus, cette tâche permet de saisir les objets véhiculés par les navettes de la cellule flexible. Le POC n°2 est une tâche de suivi de contour de forme. Le suivi de contour est classique dans l'industrie pour réaliser un dépôt de colle ou une soudure entre deux surfaces. Il s'agit de montrer la capacité de l'application ROS à suivre une trajectoire donnée à plus ou moins grande vitesse. Le POC n°3 est un évitement d'obstacle dont l'intérêt est de montrer la capacité de l'application à s'interfacer efficacement avec l'environnement ROS tel que le planificateur de trajectoire MoveIt [9].

Les deux baies des contrôleurs disposant de ports Ethernet et USB, une architecture logicielle basée sur la réalisation d'un programme écrit en langage propre faisant l'interface entre l'OS du contrôleur et la couche ROS a été retenue. Il s'agit ainsi de réaliser un *driver* ayant pour rôle de réceptionner, d'interpréter et de transférer les informations entre l'OS du contrôleur et la couche ROS. Ce principe est illustré sur la figure 5. Cette solution impose le développement d'un driver différent pour chaque contrôleur de robot mais possédant des interfaces (API) identiques avec la couche ROS. Il est donc intéressant de travailler avec deux contrôleurs pour vérifier la généricité de l'API proposée et de l'approche.

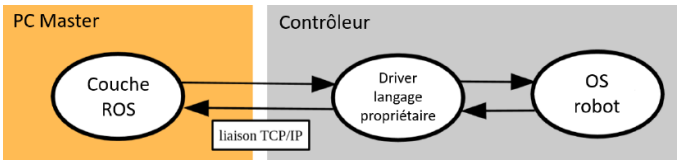

Figure 5. Schéma de l'architecture de la solution

Le driver Val3 est le driver dédié à l'utilisation du robot Stäubli dont le langage propriétaire est Val3. Il repose sur des applications composées de fonctions de bases propres à ce langage, de programmes qui sont une séquence d'instructions VAL3 à exécuter, de variables et de tâches. Une tâche est un programme en cours d'exécution qui peut fonctionner en parallèle d'autres tâches de manière synchrone ou asynchrone. Le driver Val3 est une application dont les tâches sont dédiées à la communication avec la couche ROS en utilisant un protocole de communication établi. Trois tâches ont été implémentées : *control*, *retour* et *erreur*. Ces tâches sont asynchrones et n'ont pas la même priorité. La tâche *control* réceptionne les messages envoyés par la couche ROS et les traite. La tâche *retour* lit les informations du robot réel et les renvoie à la couche ROS. Enfin, la tâche erreur traite les erreurs bloquantes mineures pour éviter une intervention de l'opérateur. La priorité maximale est accordée à la tâche *retour* car le retour de l'état du robot est primordial à la couche ROS pour pouvoir adapter son contrôle sur le robot. La tâche *control*, a une priorité haute (moindre que la tâche *retour*) afin de limiter les temps de latence.

Dans la tâche *control*, le driver ne traite qu'un message à la fois, c'est-à-dire que, à la réception d'un nouveau message, il le traduit, l'interprète et réceptionne le message suivant que lorsque les instructions ont été envoyées à l'OS du contrôleur. Le robot mettant un temps non négligeable à réaliser une action, en comparaison avec le temps de traitement d'un message, les instructions d'action sont accumulées dans une pile de consigne de l'OS. Cela permet au contrôleur d'exécuter les consignes au robot en tenant compte des consignes futures.

Les deux couches applicatives communiquent entre elles via des messages. La structure des messages est la suivante : `Id X donnée X … donnée X F`, où *Id* est un identifiant du message décrivant son contenu, *X* est le marqueur de fin de valeur, *F* est le marqueur de fin de message.

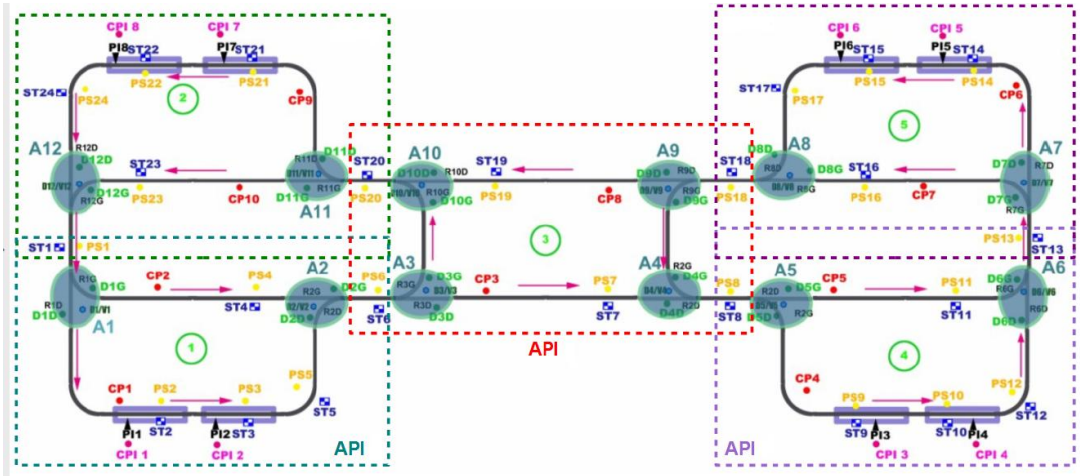

Figure 6. Capteurs et actionneurs

Les messages utilisés ont été déterminés par l'étude des informations que peut recevoir et communiquer le contrôleur du robot. Ils ont être regroupés en six familles différentes :

- Message de mouvement : contient la consigne pour les différents types de mouvement (point à point, linéaire et circulaire),
- Message de paramètres de mouvement : contient les paramètres de vitesse et de lissage,
- Message dédié à l'outil : contient la consigne de contrôle de l'outil ainsi que ces paramètres,
- Message de configuration du robot : contient la consigne de configuration et de la vitesse moniteur,
- Message d'état du robot : contient les informations de l'état du robot réel,
- Message stop : averti de la fermeture d'une application.

Les tables 1 et 2 décrivent les différents types de message par *Id* et les structures de ces messages.

	ID	Description
Mouvement	11	Coordonnée articulaire pour mouvement point à point
	121	Coordonnée cartésienne pour mouvement point à point
	122	Coordonnée cartésienne pour mouvement linéaire
	1231	Coordonnée cartésienne du point d'arrivée d'un mouvement circulaire
	1232	Coordonnée cartésienne du point intermédiaire d'un mouvement circulaire
Paramètres mouvement	2	Consigne de vitesse et de lissage du mouvement
Outil	31	Consigne des paramètres de l'outil
	32	Commande de l'action de l'outil
Configuration robot	41	Consigne de la configuration des articulations du robot
	42	Consigne de la vitesse moniteur du robot
Etat	5	Retour de l'état du robot
Stop	99	Consigne pour vider la pile de consigne du contrôleur

Tableau 1. Identifiants des différents messages

L'architecture de contrôle a également été implémentée pour le robot Kuka. Les drivers sont naturellement spécifiques à chacun des environnements du fait que les environnements VAL 3 (Stäubli) et KRL (Kuka) fonctionnent selon des logiques distinctes. Néanmoins, les mêmes structures de messages ont pu être implémentées dans les deux environnements. Les trois POCs ont ainsi pu être implémentées et validées sur chacun des robots. Les drivers pourraient certainement être enrichis pour mieux profiter de tout le potentiel des robots mais ils couvrent aujourd'hui une bonne part du besoin. Le chapitre suivant décrit comment ROS permet de faire le lien entre les robots et la cellule.

ID	Message
11	11 X j1 X j2 X j3 X j4 X j5 X j6 X F
121	121 X x X y X z X rx X ry X rz X F
122	122 X x X y X z X rx X ry X rz X F
1231	1231 X x X y X z X rx X ry X rz X F
1232	1232 X x X y X z X rx X ry X rz X F
2	2 X accel X vel X decel X tvel X rvel X blend X leave X reach X F
31	31 X x X y X z X rx X ry X rz X otime X ctime X F
32	32 X action X F
41	41 X shoulder X elbow X wrist X F
42	42 X speed X F
5	5 X isPowered X isCalibrated X workingMode X esStatus X speed X shoulder X elbow X wrist X isSettled X F
99	99 X F

Tableau 2. Structures des différents messages

C. *Développement d'un jumeau numérique de la cellule flexible de production*

La figure 6 décrit les capteurs et actionneurs du système Montratec. La ligne transitique est composée de sections et d'aiguillages sur lesquels circulent des navettes autonomes. Elle est commandée à l'aide des trois automates M340. Les aiguillages permettent de dériver les navettes vers les postes de travail robotisés. Les navettes activent à leur passage des capteurs de présence PS_i (i=1..24) disposés en amont des aiguillages A_i (i=1..12), CP_i (i=1..20) disposés en aval des aiguillages A_i (i=1..12) et CPI_i (i=1..8) situés devant les zones d'indexage. Les stops fonctionnent en logique inverse : si tous les stops sont activés, les navettes avancent sans s'arrêter. Chaque aiguillage A_i (i=1..12) comporte 4 actionneurs et 2

capteurs : les actionneurs pneumatiques *Rid*, *Rig* permettent de commander une rotation des aiguillages à droite ou à gauche, les actionneurs pneumatiques *Di*, *Vi* permettent de déverrouiller / verrouiller les aiguillages, enfin, les capteurs de *Did*, *Dig* indiquent le positionnement de l'aiguillage.

L'objectif est de commander la ligne dans l'environnement ROS en s'affranchissant de l'environnement de programmation Schneider. Il est possible d'utiliser l'outil de simulation CoppeliaSim (ex-VREP) pour créer un jumeau numérique de la ligne transitique et permettre aux étudiants de visualiser en temps réel le résultat de leurs programmes en simulation. Une fois le programme validé en simulation, il peut être directement testé sur le système physique réel (sans modification de code). La figure 7 décrit l'architecture de commande. Le jumeau numérique implémenté est représenté sur la figure 8. Le nœud *ROS-Commande* réalise l'interface entre le programme étudiant (nommé *ROS-Planificateur*) et, soit le jumeau numérique, soit le système réel. La liaison avec le système réel est assurée par le nœud *ROS-Communication* qui communique avec les automates avec un protocole Modbus TCP/IP grâce à la librairie libmodbus. Le nœud *ROS-Communication* lit et écrit dans une mémoire partagée pré-configurée dans chacun des automates. Les automates recopient périodiquement les valeurs des capteurs dans leur mémoire partagée, et actualisent les actionneurs en fonction des valeurs mises à jour dans cette même mémoire partagée.

Comme on peut le voir sur la figure 8, les robots implémentés dans le jumeau numérique ne sont pas différenciés. Ils sont pour l'instant assimilés à des procédés à événements discrets élémentaires, munis de capteurs / actionneurs, réalisant des opérations simples. Dans un premier temps, les robots réels ont été connectés aux automates par leurs modules d'entrée/sortie. Les programmes implantés au sein des robots sont déclenchés par mise-à-un d'un actionneur sur automate. Une fois la tâche réalisée, le robot l'acquitte auprès de l'automate par mise-à-un d'une de ses sorties.

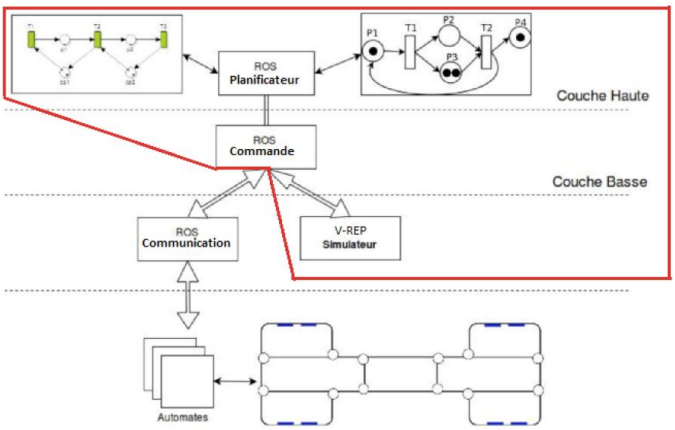

Figure 7. Architecture ROS

L'ambition à plus long terme est d'intégrer les deux travaux présentés dans cet article. Ainsi, la simulation CoppeliaSim de la cellule flexible pourrait être couplée à la simulation Gazebo de chacun des robots. Il sera ainsi possible de mixer les systèmes simulés et ceux réels afin de valider progressivement le bon fonctionnement. Il sera également possible, dans une logique de redimensionnement ou d'évolution du système, de tester virtuellement l'introduction de nouveaux matériels au sein du système réel.

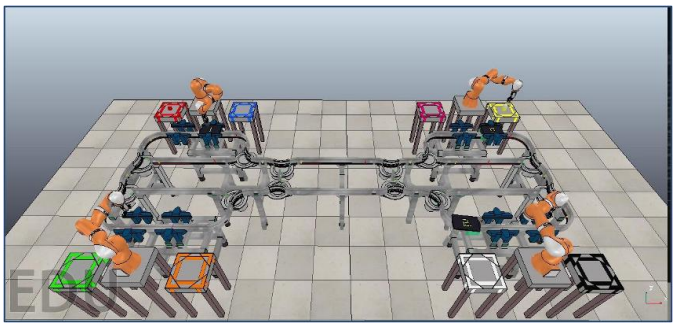

Figure 8. Un jumeau numérique de la cellule

D. Des avantages, mais aussi des limites

Les chapitres précédents ont démontré l'intérêt d'un middleware tel que ROS pour l'intégration de systèmes cyber-physiques. ROS capitalise l'expérience d'années d'ingénierie en robotique et offre une vaste collection d'applications robotiques : calcul de trajectoire, contrôle de robot par joystick, cartographie 3D, simulation, etc. Dans la cadre de l'industrie 4.0 où l'agilité est le maître-mot, disposer d'un outil offrant gratuitement des technologies aussi abouties peut être décisif dans bien des situations où il est nécessaire de prototyper très rapidement de nouvelles solutions. ROS ne nécessitant pas des ressources informatiques énormes, il peut également fonctionner sur des systèmes embarqués utilisant des microcontrôleurs (e.g., Raspberry Pi 3).

ROS a cependant les inconvénients de ses avantages : sa nature open-source. Développé par une communauté très active, mais aussi très hétérogène, il existe aujourd'hui plus de 2000 packages ROS et, malheureusement, tous ne sont pas au même niveau de maturité, ne sont pas maintenus, ni n'utilisent la même version de ROS. Il est donc souvent difficile d'identifier quels sont les packages les plus aboutis /prometteurs sans être spécialiste, voire engager une ingénierie relativement coûteuse pour les tester. De plus, ROS fonctionnant uniquement sur le système Linux Ubuntu, la quasi-totalité des packages sont incompatibles avec des OS classiques, notamment Windows, l'OS le plus répandu dans l'industrie. Enfin, les systèmes Linux et ROS évoluant tous les 3-4 ans en moyenne, il est nécessaire de mettre à jour les développements réalisés en interne afin d'éviter leur obsolescence, cela sans autre support que celui de la communauté. Là aussi des coûts d'ingénierie sont à prévoir.

Un autre aspect est le temps réel souvent indispensable pour la mise en œuvre de systèmes de contrôle-commande. Celui-ci n'est pas natif dans ROS. Il est nécessaire d'ajouter une couche telle que OROCOS (Open RObot COntrol Software) qui permet de gérer les communications efficacement. Utilisé en concomitance avec le package *ROS control*, il est possible de mettre en œuvre des systèmes de contrôle de robot intégrant des contraintes temps réel et angulaires, en ayant une IHM offerte par défaut. Néanmoins, de solides bases en informatique et robotique sont utiles pour aborder ce genre de développement.

La prochaine version de ROS, nommée ROS 2 (https://design.ros2.org/), devrait permettre de pallier la plupart des faiblesses précédemment évoquées. Ainsi qu'illustré sur la figure 9, la majeure avancée de ROS 2 consiste en l'utilisation du standard DDS (Data Distribution Service) pour la gestion des communications bas niveau. DDS est un standard ouvert produit par l'OMG (Object Management Group : https://omg.org) conçu pour répondre aux besoins de l'IoT industriel, de l'industrie des communications, de la robotique. Le principal avantage pour les développeurs de ROS est de rester centrés sur les aspects métiers exclusivement, en s'affranchissant des aspects communications. En même temps que la définition des paradigmes de diffusion de l'information, DDS permet de spécifier les besoins de l'application. Cela peut permettre de configurer le réseau dont les éléments de commutation (switches) sont programmables. Ce type de réseau est appelé Software Defined Network (SDN) et est contrôlé par un élément central (le contrôleur SDN) qui diffuse dans le réseau les règles de commutation. Un des atouts majeurs de ROS 2 est sa disponibilité sur la plupart des plateformes (Windows, Linux, MacOS). L'obsolescence de ROS 1 au profit de ROS 2 est annoncée pour 2025. Toutefois, l'utilisation de ROS 2 pourrait s'avérer plus complexe que celle de ROS 1 du point de vue informatique.

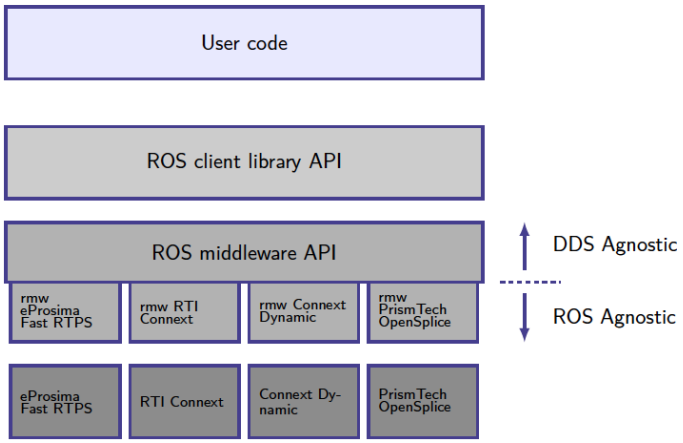

Figure 9. Architecture de ROS 2

IV. Conclusion

Les technologies robotiques évoluent de plus en plus rapidement et doivent être repensées sans cesse. Dans cette perspective, ROS offre une richesse sans précédent et apparaît être une option viable pour développer les systèmes cyber-physiques de demain et capitaliser les principales innovations industrielles et académiques. Les paquets ROS peuvent être propriétaires, les développeurs de tels paquets ayant alors à charge de mettre eux-mêmes en œuvre les actions de maintenance nécessaires à éviter l'obsolescence de leurs développements, cela pouvant être coûteux. De nombreux acteurs industriels de la robotique, rassemblés dans le consortium ROS Industrial (https://rosindustrial.org/), ont fait le choix de faire confiance à ROS pour supporter leurs développements logiciels (ABB, BOSH, PAL Robotics, YASKAWA, etc.). ROS est encore "jeune" mais ses défauts vont s'atténuer avec le temps, notamment avec l'avènement de ROS 2 en 2025.

Le pôle S-MART Occitanie a fait le choix de ROS pour l'avenir et développé, avec le support du projet européen ROSIn (https://www.rosin-project.eu/), une ROS Academy (https://www.aip-primeca-occitanie.fr/ros-in-occitanie/). L'ambition est de continuer à former les étudiants sur ROS mais aussi de contribuer à l'accélération de la diffusion de ROS au sein de l'industrie en développant des offres de formation tout-au-long-de-la-vie.

Remerciements

Ce travail a été partiellement financé par le projet ROSIn, développé dans le cadre du programme européen Horizon 2020, sous le no d'agrément 732287. Il n'aurait pas existé sans le soutien de nombreux étudiants, en projet ou stage, qui ont joué un rôle moteur dans le développement de ROS au sein du pôle SMART-Occitanie. Merci à Nathan, Ahmed, Abdoul, Thomas, l'équipe SALLAG (Steve, Anthony, Lucas, Lucie, Antonin, Guillaume), Jean-Baptiste, Claire, Enrique, Maxime, Aurélie, Céline, Bruno, Alexandra, … et tous les autres.

References

[1]. Industrie du futur : enjeux et perspectives pour la filière aéronautique. Rapport final / DGE, PIPAME, GIFAS .- in : DGE, 10/12/2018, 188p., (Etudes économiques) - En ligne sur le site de la DGE.

[2]. Sebastian Rendon Fernandez. Accuracy enhancement for robotic assembly of large–scale parts in the aerospace industry. Robotics. HESAM Université, 2020.

[3]. Fiorenzo Franceschini, Luca Mastrogiacomo, Barbara Pralio. An Unmanned Aerial Vehicle–based System for Large Scale Metrology Applications. International Journal of Production Research, Taylor & Francis, 2010, 48 (13), pp.3867–3888.

[4]. Zheng Wang. (DET2009) High accuracy mobile robot positioning using external Large Volume Metrology instruments. International Journal of Computer Integrated Manufacturing, Taylor & Francis, 2011.

[5]. Gennaro Raiola, Susana Sanchez Restrepo, Pauline Chevalier, Pedro Rodriguez–Ayerbe, Xavier Lamy, et al.. Co–manipulation with a Library of Virtual Guiding Fixtures. Autonomous Robots, Springer Verlag, 2017, pp.1–15.

[6]. Moritz Pöting, Stefan Schaudt, Uwe Clausen. A Comprehensive Case Study in Last-Mile Delivery Concepts for Parcel Robots. WSC 2019: 1779-1788.

[7]. Voker Grabé, Martin Riedel, Heinrich H Bülthoff, Paolo Robuffo Giordano, and Antonio Franchi. The TeleKyb Framework for a Modular and Extendible ROS-based Quadrotor Control. In European Conference on Mobile Robots, ECMR 2013, Barcelona, Catalonia, Spain, September 2013.

[8]. Stefan-Gabriel Chitic. Middleware and programming models for multi-robot systems. Theses, Université de Lyon, March 2018.

[9]. Michael Görner, Robert Haschke, Helge J. Ritter, Jianwei Zhang. MoveIt! Task Constructor for Task-Level Motion Planning. ICRA 2019: 190-196