P-RC2: Platform for Robot Controller Construction

B. Milville, B. Gradoussoff, B. Morinière

***Abstract*— Cet article présente le positionnement et les résultats préliminaires du projet P-RC2 (Platform for Robot Controller Construction) dont l'objectif principal est de faciliter la mise au point de contrôleurs robotiques de qualité industrielle en fournissant une solution de développement intégrée. L'approche proposée exploite les méthodes de conception orientée par les modèles pour permettre de déployer aisément un éventail de fonctionnalités indépendemment du middleware robotique exploité. L'architecture, la méthodologie et l'outillage développés dans le cadre de ce projet sont présentés dans cet article.**

I. Introduction

A. Problématique

Comment adresser la complexité des applications robotiques académiques et industrielles à forte innovation ? Comment accélérer les temps de développement de ces applications et diminuer les coûts associés ? Comment capitaliser les développements réalisés ? Comment garantir la qualité de service et la sécurité pour un environnement industriel ? Voici les questions qui motivent les travaux présentés dans cet article.

Le projet P-RC2 propose une architecture, une méthodologie et un outillage visant à adresser ces problèmes tout gardant les bénéfices apportés par les middlewares robotiques.

B. Les middlewares robotiques

Les applications robotiques académiques comme industrielles sont de plus en plus complexes. L'usage de plateformes logicielles adaptées au développement et à la mise au point de telles applications devient alors incontournable. Les enjeux principaux sont de disposer d'une architecture, d'outils et de méthodes permettant de gérer la complexité et l'hétérogénéité intrinsèques des applications, de conserver la flexibilité, d'accélérer le développement en facilitant la réutilisation de code existant tout en garantissant la qualité de service et les performances de l'applicatif final.

Un middleware est une couche d'abstraction qui est placée entre le système d'exploitation et les applications logicielles. Largement utilisé en robotique [1], [2], cet élément intermédiaire apporte de nombreux avantages : il fournit des outils, des pilotes matériels et des algorithmes dédiés à la robotique et facilite la communication entre les éléments de l'applicatif. Les middlewares robotiques les plus utilisés exploitent de plus un paradigme "orienté composant" [3] qui encourage la réutilisation de fonctionnalités d'une application à l'autre. Bien que ROS [4] et, dans une moindre mesure, OROCOS [5] disposent de communautés d'utilisateurs importantes, aucun middleware robotique ne s'est imposé pour l'instant. Ceci s'explique en particulier par le fait qu'aucun middleware ne propose à la fois un vaste inventaire de fonctionnalités robotiques, des performances *temps réel* et une large communauté.

L'offre continue aujourd'hui d'évoluer [6] si bien qu'il reste stratégiquement risqué pour un industriel d'opter pour un middleware particulier. La solution de coupler plusieurs middlewares pour palier aux lacunes de chacun n'est pas idéale car elle nécessite la mise en place de *bridges*, uniquement pour faire communiquer les différents middlewares, qui sont difficiles à maintenir et qui pénalisent généralement les performances globales du système [7].

C. Conception orientée par les modèles

La conception orientée par les modèles (MBD pour *Model-Based Design*) [8], [9] est une méthode de développement largement utilisée dans l'automobile [10] et l'avionique. Cette méthode d'ingénierie s'appuie sur le constat pragmatique que les systèmes complexes ne peuvent être développés de manière déterministe et maintenable que s'ils sont d'abord modélisés, analysés et vérifiés de façon abstraite et si leur code source est généré. En pratique, cette méthode permet ainsi :

- le test au plus tôt des choix stratégiques et techniques
- l'implémentation et la validation des fonctionnalités avant la disponibilité d'un prototype
- la détection précoce des erreurs grâce à une vérification à tous les stades de la conception
- le prototypage accéléré par un redéploiement rapide après tout changement de conception
- la décorrélation des compétences entre experts du domaine et experts en programmation
- le travail en équipes délocalisées

L'approche MBD est encore assez peu exploitée dans le domaine de la robotique car le manque d'outillages adaptés aux spécificités du domaine conduit les développeurs à considérer la modélisation exhaustive comme un "détour" coûteux [11].

II. Le positionnement de P-RC2

P-RC2 s'appuie sur le paradigme de la conception orientée par les modèles pour proposer des outils de mise au point de

B. Milville (benoit.milville@cea.fr),
B. Gradoussoff (baptiste.gradoussoff@cea.fr),
B. Morinière (boris.moriniere@cea.fr)
CEA, LIST, Laboratoire de Robotique Interactive, 91190 Gif sur Yvette, FRANCE

Ces travaux sont financés par l'appel à projet "Logiciel Embarqué et Objets Connectés" (LEOC), dans le cadre du Programme Investissements d'Avenir du gouvernement français

contrôleurs robotiques flexibles vis à vis du middleware robotique exploité. La plateforme doit en particulier permettre au développeur de :

— Modéliser la structure de son contrôleur

— Intégrer facilement du code métier existant

— Déployer les fonctionnalités sur le middleware de son choix pour bénéficier de ses spécificités

— Apporter des outils supplémentaires de supervision génériques

— Faciliter la validation et la vérification à l'échelle du composant et à l'échelle du contrôleur

A. Axes prioritaires et objectifs

Trois axes prioritaires ont guidé les choix technologiques du projet P-RC2 :

a) Modularité et généricité: L'objectif est de concevoir une architecture polyvalente et flexible pour permettre à l'utilisateur de se focaliser sur l'innovation en simplifiant au maximum les tâches d'implémentation qui sortent de son domaine d'expertise, et en lui permettant d'exploiter simplement les développements capitalisés de projets précédents.

b) Sûreté de fonctionnement: Une des difficultés rencontrées lors la mise au point de contrôleurs robotiques est la validation du comportement pour passer du prototype de laboratoire au produit exploitable sur une application industrielle.

L'objectif de P-RC2 est de mettre à disposition du concepteur les outils et la méthodologie nécessaires à la validation de ses développements.

c) Ergonomie et facilité de prise en main: L'ergonomie de la plateforme est assurée par une intégration complète dans un environnement, Eclipse, et par le développement d'interfaces dédiées au flux de travail proposé, associées à une documentation exhaustive et des tutoriels détaillant les bonnes pratiques.

B. Les produits de P-RC2

Le projet P-RC2 a pour objectif de mettre au point une plateforme logicielle modulaire dédiée aux roboticiens. Elle contient quatre éléments complémentaires :

1) Une architecture de contrôleur: Cette infrastructure standardisée est conçue pour accueillir les briques *métier* du contrôleur tout en intègrant les fonctions services nécessaires aux applications industrielles (services de gestion des erreurs, de journaux, d'enregistrement de données, etc). L'architecture est détaillée dans la section V.

2) Une boîte à outils: Elle groupe à la fois des ressources utiles à la création de contrôleurs robotiques (cf section III-B) et des outils utiles à leur construction (cf section III-A) et à leur exploitation (cf section III-C).

3) Une méthodologie: Adossée à l'architecture et aux outils, elle guide le développeur dans la mise au point de son contrôleur. Elle garantit le respect et la pérennité des bonnes pratiques. La méthodologie est détaillée dans la section IV.

4) Des ressources documentaires: Au moins aussi important que les trois éléments techniques précédents, un corpus documentaire permet de garantir à tous une prise en main rapide de la plateforme et, aux utilisateurs expérimentés, l'accès aux fonctions les plus avancées.

C. Les concepts-clé de P-RC2

Cette partie présente les concepts utilisés dans le projet P-RC2 et explicite par la même occasion le vocabulaire technique employé dans cet article.

1) Conception orientée composants: La conception orientée composants est un paradigme de programmation qui vise à favoriser la réutilisation de fonctionnalités existantes en s'appuyant sur la séparation des cinq aspects de la fourniture d'une fonctionnalité que sont le calcul, la coordination, la communication, la configuration et la composition. Un composant logiciel est ainsi défini [12] comme une unité de composition dotée essentiellement d'une interface explicitement spécifiée et d'un contexte déclaré de dépendances. Ainsi, un tel composant peut être déployé de manière indépendante pour être intégré en l'état dans des applications tierces. Ce concept est fondamental dans P-RC2 car il permet de garder une architecture modulaire et générique.

2) Abstraction du middleware: Nous avons choisi de ne pas construire P-RC2 sur un middleware particulier pour des raisons de généricité et de pérennité de la plateforme. L'abstraction du middleware associée à la conception orientée modèles permet de séparer totalement la représentation fonctionnelle du contrôleur de son déploiement, spécifique au middleware. Le code source *métier* des composants, qui implémente les algorithmes robotiques, devient alors portable et facilement réutilisable. L'abstraction du middleware permet également de représenter au niveau fonctionnel des concepts qui ne sont pas fournis par tous les middlewares, c omme la notion de macro-composant.

3) Conception orientée modèles: Le second paradigme utilisé est donc celui de la conception orientée modèles (MBD pour *Model Based Design*), qui permet l'implémentation concrète du concept d'abstraction middleware. Par ailleurs, la conception orientée modèles augmente l'intérêt du paradigme composant en permettant de construire graphiquement le déploiement des composants dans le contrôleur. C'est une fonctionnalité qui n'est pas toujours proposée par les middlewares robotiques.

D. Travaux similaires

OpenRTM [13] est un middleware japonais qui contrairement à ROS ou OROCOS propose un méta-modèle explicitement défini pour la composition hierarchique des composants. Les macro-composants d'OpenRTM sont cependant responsables de la coordination des exécutions de leurs sous-composants si bien qu'il n'est pas possible de distribuer les sous-composants d'un macro-composant sur plusieurs processus ou machines.

BRICS [14] est un projet européen qui a abouti au dévelopement de BRICS Component Model, un modèle de

composant indépendant du middleware et de BRIDE, un plugin Eclipse associé.

HyperFlex [15] est une extention de BRICS Component Model qui ajoute un concept de promotion d'interface pour faciliter la composition à un plus large niveau de granularité.

La plateforme P-RC2 se démarque des travaux de BRICS et HyperFlex en ajoutant en particulier la fourniture d'une bibliothèque de fonctionalités et de services robotiques indépendante des middlewares exploités.

III. La boîte à outils de P-RC2

La plateforme P-RC2 veut proposer un ensemble d'outils complet pour le développement et l'exploitation de contrôleurs. Cela se traduit par un ensemble de ressources et d'outils mis à disposition des utilisateurs.

A. Outils de construction (du contrôleur)

1) Éditeur de modèles: L'édition des modèles est réalisée sur la base de RobotML [16], [17], un outil de modélisation robotique qui s'appuie sur l'éditeur UML [1] Papyrus [18].

La version actuelle de RobotML permet de modéliser une application robotique indépendemment du middleware cible, à l'aide de diagrammes d'architecture et de machines à états. La modélisation s'appuie sur une ontologie de composants dédiée à la robotique mobile, et la génération de code est possible vers quatre middlewares (RTMaps, Urbi, Arrocam et OROCOS).

Dans le cadre de P-RC2, une évolution du méta-modèle de RobotML est prévue pour intégrer une nouvelle ontologie de composants, avec un périmètre élargi aux différentes thématiques de la robotique, et une granularité des composants qui soit adaptée aux besoins identifiés.

2) Générateur de code: La génération de code est réalisée à l'aide des outils Acceleo ou Xtend, tous deux intégrés à l'environnement Eclipse, et pouvant être utilisés depuis RobotML. Les deux outils se basent sur la notion de *template* de génération, qui sont des fichiers propres au langage de destination. On retrouve donc pour chaque middleware cible un ensemble de fichiers *templates* permettant de générer les différents sources du projet.

3) Outils de vérification: L'analyse de sûreté de fonctionnement peut être réalisée à deux niveaux complémentaires :

Au niveau du modèle du contrôleur, l'outil SOPHIA [19] permet de réaliser des analyses de type FTA (*Fault Tree Analysis*) et FMEA (*Failure Modes and Effects Analysis*).

Au niveau du composant, l'outil Frama-C [20] permet de valider le comportement de chaque composant individuellement en réalisant une analyse statique du code source. Cette analyse permet par exemple de compléter ou remplacer les tests unitaires, et peut être automatisée pour éviter les régressions lors du développement de codes complexes.

B. Ressources logicielles

La capitalisation et la réutilisation des connaissances est une priorité de P-RC2. Elle s'appuie sur trois bibliothèques distinctes, destinées à organiser et faciliter les échanges communautaires.

1) Bibliothèque de composants: La bibliothèque de composants a pour rôle de mettre à disposition publiquement les composants P-RC2 existants, afin de permettre leur réutilisation dans d'autres projets. Ces composants sont indépendants du middleware cible, et sont organisés selon la classification présentée en partie IV.B.

2) Bibliothèque de patrons de contrôleurs: De la même manière, une bibliothèque de modèles de contrôleurs est prévue. Elle pourra contenir des contributions allant du patron de sous-système robotique jusqu'au modèle complet de contrôleur qui contient l'ensemble des patrons de déploiement pour les différents modes de fonctionnement, ainsi que la machine d'états associée.

3) Patrons de génération de code: La troisième bibliothèque contient les patrons de génération de code utilisés pour la conversion M2T (Model to Text) permettant de générer les fichiers sources middleware à partir des modèles.

C. Outils d'exploitation (du contrôleur)

Plusieurs IHM [2] sont prévues pour permettre à l'utilisateur final de travailler avec son contrôleur :

a) Pilotage: Interface permettant d'opérer un système robotique en service. eg : Mise sous puissance, déplacements, changements de mode de fonctionnement, acquittement des erreurs, ...

b) Réglage: Cette interface permet de manipuler directement les paramètres du contrôleur et de réaliser des manipulations qui requièrent plus de connaissance sur le fonctionnement du contrôleur et du matériel piloté.

c) Scénarisation: Cette interface permet d'éditer et de jouer des *programmes*, sous forme de scripts, qui pourront être executés par le contrôleur.

IV. La méthodologie de P-RC2

A. Flux de travail

L'utilisation de la plateforme P-RC2 a été pensée pour être accessible à tous. Elle repose sur des outils sélectionnés et adaptés aux besoins spécifiques du projet. Le *flux de travail* d'utilisation mis au point est présenté Figure 1.

L'utilisateur applique les étapes suivantes.

(i) Sélection d'un patron de contrôleur au plus proche de l'architecture souhaitée

(ii) Modification de ce patron pour l'adapter à ses besoins

- Ajout, remplacement et/ou suppression de composants
- Sélection des implémentations *métier* désirées pour chaque composant
- Modification des connexions inter-composants

1. Unified Modeling Language

2. Interface Homme-Machine

— Configuration des paramètres de chaque composant

(iii) Configuration du déploiement, de manière spécifique au middleware sélectionné

(iv) Génèration du code middleware nécessaire à la compilation

(v) Modification éventuelle du code généré :

— Modification du déploiement ou de l'ordonnancement

— Ajustement de paramètres

— Modification de la machine d'état du contrôleur

— ...

(vi) Compilation et empaquettage, automatiquement, de l'ensemble du contrôleur

(vii) Déploiement du contrôleur final sur son matériel

(viii) Configuration du contrôleur final via les outils/services mis à sa disposition

La génération de code est complémentaire à la modélisation du contrôleur et à l'abstraction du middleware. On génère à partir du modèle les fichiers sources suivants :

— squelettes des composants (indépendants du middleware cible), ce qui permet de simplifier la création de nouveaux composants.

— fichiers de déploiement de l'architecture pour le middleware cible

— fichiers d'encapsulation middleware des composants

— fichiers de description des machines à états

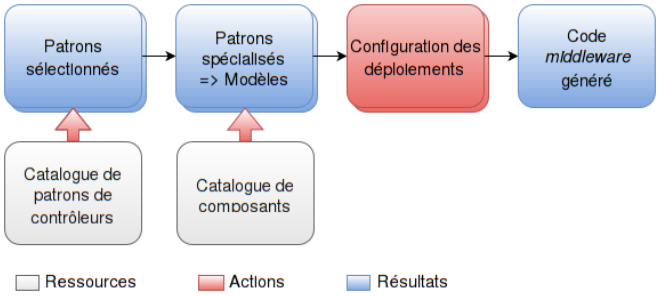

Fig. 1: Flux de travail P-RC2

B. Standardisation

Le flux de travail tire son efficacité de la réutilisation de patrons et de composants existants. Cette réutilisation n'est possible que si composants et patrons partagent une API[3] standardisée. La philosophie de P-RC2 est d'encourager les utilisateurs à appliquer des bonnes pratiques plutôt que de les enfermer dans un cadre imposé. En suivant cette ligne directrice, nous avons choisi de fournir des recommandations sous forme de spécifications d'API, que l'utilisateur final est libre de respecter ou non. Ces recommandations sont également un standard *defacto*, puisqu'elles sont appliquées aux composants développés dans le cadre des démonstrateurs du projet.

Dans P-RC2, la standardisation est abordée à deux niveaux, qui sont présentés ci-après.

1) Classification des composants: La classification des composants découle d'un besoin simple : avant même de consider les questions d'API, on ne peut espérer interchanger que des composants ou sous-systèmes fonctionnellement équivalents. Elle permet de délimiter le périmètre fonctionnel des composants métier. Ainsi, deux composants d'une même catégorie de la classification ont de fortes chances d'être conceptuellement interchangeables. En plus de ce rôle de standardisation, elle a également un rôle documentaire pour l'organisation de la bibliothèque de composants, en permettant d'indexer et de rechercher thématiquement les composants métier pouvant répondre au besoin.

La classification des composants a été réalisée par une démarche itérative, en croisant deux approches complémentaires. Une approche *top-down*, basée sur une étude bibliographique des ontologies et des architectures existantes, a permis d'identifier les axes possibles de classification. Une approche *bottom-up*, basée sur l'analyse de schémas de contrôleurs robotiques existants, a permis de lister les composants et catégories de composants les plus pertinents. Pour garantir la pertinence de l'analyse, nous avons étudié des familles de contrôleurs robotiques très différentes : bras manipulateurs, AGVs[4], robots humanoïdes et robots quadrupèdes.

La solution proposée s'inspire de l'architecture *3 Tier* [21], avec un positionnement plus bas-niveau, centré sur le contrôleur du robot. La classification résultante est basée sur un découpage à 4 couches principales :

— Supervision Layer : regroupe les couches *Planning* et *Sequencing* de l'architecture 3T.

— Control Layer : regroupe l'ensemble du contrôle commande d'un mécanisme robotique, dans trois sous-catégories *Motion Generation*, *Motion Control*, et *State Observation*.

— Hardware Abstraction Layer : regroupe les composants d'interfaçage avec le matériel (drivers et composants de communication) dans les sous-catégories *Sensors*, *Actuators*, *Robots and Machines* et *Communication*.

— Core Layer : couche de services génériques regroupant les composants d'infrastructure de P-RC2, tels que présentés dans la partie V-B.2.

2) Définition des API composants: La spécification des API composants est complémentaire à leur classification, en permettant de garantir que deux composants fonctionnellement équivalents seront interchangeables s'ils respectent tous deux la spécification de leurs APIs. Ce travail est en cours de réalisation, et consiste à définir un ensemble de recommandations sous la forme d'un standard qui sera appliqué aux composants développés dans P-RC2. La standardisation porte sur la description des ports, paramètres et méthodes publiques des composants. Elle explicite les conventions à respecter en termes de nommage, de typage informatique, et de typage métier (structuration des données pour la représentation des concepts robotiques usuels, tels

3. Application Programming Interface

4. Automated Guided Vehicles

que la cinématique cartésienne et articulaire, les torseurs d'efforts, les images et nuages de points...)

V. L'ARCHITECTURE DU CONTRÔLEUR

Cette partie présente les choix d'implémentation réalisés sur P-RC2, ainsi que les premiers résultats obtenus.

Un objectif majeur de P-RC2 est de permettre *au maximum* la réutilisation de code. Cela se traduit principalement en deux idées :

— Les composants *métier* développés ne doivent en aucun cas dépendre d'un quelconque middleware : c'est la matérialisation du concept d'*abstraction middleware* tel qu'expliqué dans la section II-C.2.

— Les outils de déploiement et de supervision d'un contrôleur développé avec P-RC2, doivent eux aussi être génériques.

A. Abstraction Middleware

Les concepts utilisés dans P-RC2 ont étés sélectionnés de façon à pouvoir se *traduire naturellement* en concepts communs aux différents middlewares évalués. L'idée est de pouvoir, dans la mesure du possible, utiliser les outils natifs du middleware avec un composant P-RC2.

Cela implique de réellement s'adapter au middleware, en respectant les types de données disponibles par exemple, pour en exploiter au mieux les capacités qu'il fournit et ainsi réduire la *surchage* (*overhead*) induite par une couche d'abstraction, eg : limiter la copie de données entre le middleware et le composant P-RC2.

Pour ce faire, une couche d'abstraction middleware, MAL (pour *Middleware Abstraction Layer*), a été développée. Un composant utilise les concepts de la MAL qui doit, lors de l'édition des liens, fournir une implémentation middleware pour chaque concept utilisé.

Tous les concepts sont implémentés suivant la même structure : ils dérivent d'une base *Objet* (indépendante du middleware) qui possède et instancie une implémentation (différente selon le middleware). Lorsqu'un *Objet* est associé à un *Objet* parent (eg : ajout d'un port à un composant), son implémentation fait de même avec l'implémentation du parent. L'implémentation de cette association est différente selon le middleware choisi.

Par la suite, un appel à une méthode d'un *objet* est simplement relayée à la méthode de l'implémentation qui réalise l'appel à la méthode correspondante du middleware. Cette indirection supplémentaire (entre un *objet* et son implémentation) permet de relâcher le couplage avec le middleware.

Exemples pratiques: Parmi les différents middlewares disponibles, deux ont été retenus pour servir d'exemples d'implémentation :

— OROCOS [5] de part ses propriétés *temps réel* et le fait qu'il est déjà utilisé par des partenaires du projet.

— ROS [4] car son architecture relativement éloignée de celle d'OROCOS est intéressante pour valider le concept d'abstraction middleware mis en place pour P-RC2. Par ailleurs, ROS dispose d'outils pratiques pour la simulation et les tâches *haut niveau*.

Des implémentations vers d'autres middlewares viendront par la suite en fonction des besoins et des évolutions des middlewares disponibles.

Les développements réalisés jusqu'alors ont permis de valider le concept d'abstraction middleware en faisant correspondre les concepts P-RC2 aux concepts natifs des deux middlewares sélectionnés [5]. TABLE 1 présente la correspondance effectuée entre les concepts P-RC2 et les concepts middleware.

P-RC2	**OROCOS 2.8**	**ROS (Jade)**
Component	TaskContext	Node
InPort	InputPort	Subscriber
OutPort	OutputPort	Publisher
Parameter	Property	Parameter
APICallProvider	Operation	Service / Action
APICaller	OperationCaller	ServiceClient / ActionClient
APICallReq	SendHandle	GoalHandle

TABLE 1: Correspondance des concepts

B. Architecture d'un contrôleur P-RC2

La problématique du déploiement se décompose en différentes étapes :

1) Outils et Services génériques: La majorité, voir la totalité, des outils et services proposés par P-RC2 doivent être génériques et ne pas dépendre d'un middleware donné. Cela permet entre autres de :

— Limiter le développement initial et la duplication de code

— Simplifier la maintenance et les évolutions des outils

— Proposer une expérience utilisateur similaire quelque soit le middleware utilisé.

L'objectif n'est en aucun cas de se soustraire aux outils d'introspection propres aux middlewares, mais de venir les compléter.

Différents outils et services doivent être développés dans le cadre du projet, à savoir :

— Un service de journaux (*logs*) permettant d'enregistrer des messages en provenance des composants

— Un service de surveillance (*monitoring*) pour enregister de façon périodique des variables internes aux composants, afin de simplifier la mise au point des algorithmes et du contrôleur.

— Un service de gestion des *APICalls* permettant d'agréger les API des composants et d'exposer l'API contrôleur résultante.

— Des outils de supervision pour commander le système robotique

5. Certains types d'APICall ne sont, par nature, pas supportés dans l'implémentation ROS existante. Des solutions "alternatives" sont envisagées pour contourner "au mieux" cette limite du middleware

2) Architecture envisagée: La définition des services proposés par la plateforme associée à d'autres contraintes, comme la possibilité de faire cohabiter et communiquer plusieurs contrôleurs, permettent de définir une architecture pour la plateforme qui soit modulable et flexible.

La Figure 2 illustre l'architecture envisagée pour le déploiement de plusieurs contrôleurs qui seraient amenés à communiquer entre eux et avec des outils de supervision déportés.

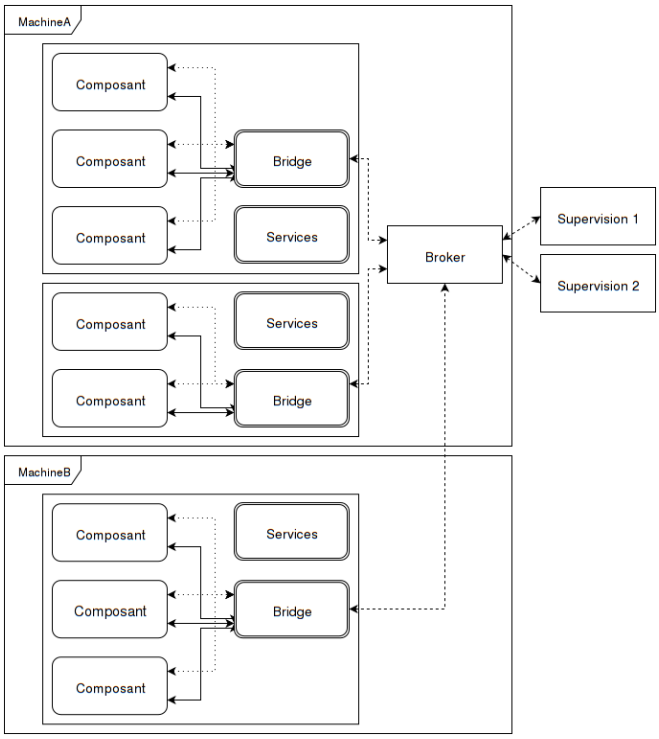

Fig. 2: Architecture d'un contrôleur P-RC2

Différents éléments, présents en *standard* au sein d'un contrôleur P-RC2, diffèrent d'une architecture de middleware robotique typique.

Des composants particuliers, dits "de service", sont ajoutés dans le contrôleur, notamment :

— Le *bridge* : Les composants utilisent les standards du middleware pour communiquer, tandis que les outils de supervision sont destinés à être générique. Un service, spécifique à chaque implémentation middleware, est donc responsable de faire les conversions entre les concepts du middleware et ceux propres à P-RC2. Il est le "point d'entrée" pour interagir avec un contrôleur.

— La machine d'état du contrôleur : Gérer les déploiement et comportement du contrôleur dans ses différents modes de fonctionnement passe par l'utilisation d'une machine à états hiérarchique, HFSM (pour *Hierarchical Finite State Machine* [22]). Cela permet par exemple de gérer simplement les transitions entre les différents modes de fonctionnement.

— Le gestionnaire de défauts : Fortement couplé à la machine d'état, ce composant reçoit les défauts levés par les composants et déclenche des actions ou comportements réactifs au niveau du contrôleur. Contrairement à un événement simple, le défaut est persistant, et doit être acquitté (automatiquement ou par l'utilisateur) avant que le contrôleur ne retourne dans un état de fonctionnement standard.

Une communication additionnelle est prévue entre les composants et le *bridge*, en supplément des fonctions standard des middlewares. Elle permettra aux composants de pouvoir remonter des informations à la supervision et aux services de journalisation et de surveillance. Elle sera compatible avec un fonctionnement *temps réel* du contrôleur.

Un service de *broker*, par machine ou unique pour plusieurs contrôleur, permettra de jouer un rôle d'annuaire ou de point d'entrée unique pour connecter une supervision unique ou plusieurs contrôleurs entre eux (selon un couplage faible).

Leur présence permet de standardiser des concepts importants au sein de contrôleurs robotiques mais non présents ou trop peu adaptés dans les différents middlewares évalués.

Un utilisateur avancé de la plateforme P-RC2 peut néanmoins simplifier l'architecture de son contrôleur pour l'adapter à son déploiement.

VI. Conclusion

A. Avancement du projet

P-RC2 est un projet de trois ans actuellement au milieu de son terme. Les spécifications fonctionnelles et le flux de travail ont été définis, et un premier prototype de l'architecture logicielle a été réalisé, intégrant l'implémentation des concepts de l'abstraction middleware. Ce prototype a été validé à l'aide d'un maquettage déployé sur deux middlewares différents (ROS et OROCOS). Concernant les composants robotiques, une classification a été réalisée, et une spécification des API est en cours. La prochaine étape consiste à développer les composants robotiques fonctionnels qui seront utilisés dans les démonstrateurs prévus au projet.

B. Travaux à venir

Plusieurs tâches restent à terminer ou à réaliser dans le cadre du projet afin d'aboutir à une plateforme exploitable en contexte industriel.

1) Modélisation: Le travail principal à venir est l'adaptation du méta-modèle de RobotML pour les besoins de P-RC2. Cette tâche consiste à implémenter dans le méta-modèle la classification des composants et les spécifications d'API des composants, de manière à permettre leur utilisation directe lors de la conception de nouveaux composants et de la construction de contrôleurs à partir de composants existants.

2) Architecture logicielle: L'implémentation représente une part importante des travaux à réaliser.

Une des premières étapes était de définir une architecture logicielle en accord avec la philosophie et les contraintes de performances du projet P-RC2. Un soin particulier y a donc été apporté. L'implémentation logicielle de cette architecture

au niveau des composants étant faite, il faut maintenant réaliser l'implémentation logicielle concernant le contrôleur (infrastructures liées au déploiement du contrôleur pour un middleware donné) puis développer les services et outils permettant son exploitation.

3) Caractérisation: Une fois la plateforme implémentée, une étape de caractérisation devra être menée afin de montrer que les mécanismes permettant l'*abstraction middleware* ne nuisent pas aux performances de la plateforme, par rapport aux contraintes *temps-réel* en particulier.

4) Documentation: L'ergonomie est une des trois priorités du projet. Cela passe par la rédaction de plusieurs documentations adaptées aux différentes catégories d'utilisateurs de la plateforme, ainsi que par la mise à disposition d'outils de partage des connaissances eg : listes de diffusions, site d'entraide communautaire.

Différents formats sont prévus, dont des

- tutoriels de prise en main, basés sur des exemples concrets
- guides méthodologiques de conception de contrôleur, destinés aux développeurs qui utilisent P-RC2
- guides avancés pour l'évolution de la plateforme, destinés aux développeurs souhaitant ajouter le support d'un middleware particulier, ou enrichir l'abstraction proposée.
- documentations d'API du code du projet, générées par l'outil Doxygen, permettant de développer avec et pour la plateforme P-RC2
- manuels des outils de construction, et des outils d'exploitation

Les bonnes pratiques mises en avant par le projet encourageront les développeurs de composants à documenter à la fois leur code, via Doxygen, mais aussi le rôle et l'API de leur composant, via une courte documentation textuelle.

5) Démonstrateurs: Dans le cadre du projet, trois démonstrateurs industriels seront réalisés par les partenaires afin d'évaluer et d'illustrer les produits du projet.

a) Cobot mobile: Il s'agit d'un prototype de robot de co-manipulation de charges lourdes associant une base mobile et un bras manipulateur. Ce cobot devra intégrer deux modes de fonctionnement : un mode autonomone et un mode où l'effecteur est co-manipulé à la main par un opérateur.

b) Téléopération: Ce démonstrateur couple un bras maitre avec un robot manipulateur esclave pour permettre de réaliser des tâches de maintenance en milieu hostile avec un retour en effort.

c) Bin-picking: Ce démonstrateur est centré sur une activité de prise de pièces en vrac automatisée utilisant un système de vision 3D et un bras manipulateur industriel.

REMERCIEMENTS

La plateforme P-RC2 est développée dans le cadre d'un projet collaboratif mené par Akeoplus [23], Arcure [24], BA Systèmes [25], le CEA-LIST [26] et Sarrazin Technologies [27].

Nous tenons à remercier tout particulièrement B. Boquet, C. Lecerf et J.-B. Tredez pour leur participation à la définition de l'architecture logicielle de la plateforme P-RC2.

REFERENCES

[1] A. Elkady and T. Sobh, "Robotics middleware : A comprehensive literature survey and attribute-based bibliography," May 2012.

[2] N. Mohamed, J. Al-Jaroodi, and I. Jawhar, "Middleware for robotics : A survey," in *2008 IEEE Conference on Robotics, Automation and Mechatronics*, pp. 736–742, Sept 2008.

[3] A. Shakhimardanov, N. Hochgeschwender, and G. K. Kraetzschmar, "Component models in robotics software," in *Proceedings of the 10th Performance Metrics for Intelligent Systems Workshop*, pp. 82–87, 2010.

[4] M. Quigley, K. Conley, B. P. 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*, 2009.

[5] H. Bruyninckx, "Open RObot COntrol Software : the OROCOS project," 2001.

[6] E.Fernandez, T. Foote, D. Thomas, and W. Woodall, "Why you want to use ROS 2," in *ROSCon*, 2014.

[7] J. Wienke, A. Nordmann, and S. Wrede, *A Meta-model and Toolchain for Improved Interoperability of Robotic Frameworks*, pp. 323–334. Springer Berlin Heidelberg, 2012.

[8] C. Atkinson and T. Kuhne, "Model-driven development : a metamodeling foundation," *IEEE Software*, vol. 20, pp. 36–41, Sept 2003.

[9] J. C. Jensen, D. H. Chang, and E. A. Lee, "A model-based design methodology for cyber-physical systems," pp. 1666–1671, July 2011.

[10] S. Fürst, J. Mössinger, S. Bunzel, T. Weber, F. Kirscheke-Biller, P. Heitkämper, G. Kinkelin, K. Nishikawa, and K. Lange, "Autosar - a worldwide standard is on the road," in *International Congress on Electronic Sysems for Vehicles*, 2009.

[11] J.Sprinkle, J. M. Eklund, H. Gonzalez, E. I. Grøtli, B. Upcroft, A. Makarenko, W. Uther, M. Moser, R. Fitch, H. Durrant-Whyte, and S. S. Sastry, "Model-based design : a report from the trenches of the darpa urban challenge," *Software & Systems Modeling*, vol. 8, no. 4, pp. 551–566, 2009.

[12] D. Brugali and P. Scandurra, "Component-based robotic engineering (part i) [tutorial]," *IEEE Robotics Automation Magazine*, pp. 84–96, Dec. 2009.

[13] N. Ando, S. Kurihara, G. Biggs, T. Sakamoto, H. Nakamoto, and T. Kotoku, "Software deployment infrastructure for component based rt-systems," *Journal of Robotics and Mechatronics*, vol. 23, no. 3, 2011.

[14] H. Bruyninckx, M. Klotzbücher, N. Hochgeschwender, G. Kraetzschmar, L. Gherardi, and D. Brugali, "The brics component model : a model-based development paradigm for complex robotics software systems," in *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, 2013.

[15] L. Gherardi and D. Brugali, "Modeling and reusing robotic software architectures : The hyperflex toolchain," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 6414–6420, May 2014.

[16] "Robotml." `http://robotml.github.io/`, 2012.

[17] S. Dhouib, S. Kchir, S. Stinckwich, T. Ziadi, and M. Ziane, "RobotML, a Domain-Specific Language to Design, Simulate and Deploy Robotic Applications," in *Third international conference on Simulation, Modeling, and Programming for Autonomous Robots (SIMPAR'12)*, pp. 149–160, Nov 2012.

[18] Papyrus. `https://eclipse.org/papyrus/`, 2015.

[19] N. Yakymets, H. Jaber, and A. Lanusse, "Model-based system engineering for fault tree generation and analysis," in *Proceedings of the 1st International Conference on Model-Driven Engineering and Software Development*, pp. 210–214, 2013.

[20] F. Kirchner, N. Kosmatov, V. Prevosto, J. Signoles, and B. Yakobowski, "Frama-c : A software analysis perspective," *Formal Aspects of Computing*, pp. 573–609, 2015.

[21] E. Gat, "Artificial intelligence and mobile robots," ch. Three-layer Architectures, pp. 195–210, Cambridge, MA, USA : MIT Press, 1998.

[22] D. Harel, "Statecharts : A visual formalism for complex systems," *Sci. Comput. Program.*, pp. 231–274, Jun 1987.

[23] Akeoplus. `http://www.akeoplus.com/`, 2015.

[24] Arcure. `http://www.arcure.net/`, 2015.

[25] BA Systemes. `http://www.basystemes.com/`, 2015.

[26] CEA-LIST. `http://www-list.cea.fr/`, 2015.

[27] Sarrazin Tech. `http://www.sarrazin-technologies.com/`, 2015.