Architectures Logicielles pour la Robotique Autonome

Félix Ingrand
LAAS/CNRS,
7 Avenue du Colonel Roche,
F-31077 Toulouse Cedex 04, France
felix@laas.fr
Draft version
Final version : `http://www.laas.fr/~felix/publis/`

Résumé

La mise en place de l'ensemble des logiciels nécessaires au déploiement de robots autonomes, c'est-à-dire l'architecture de ces systèmes, reste un problème ouvert. D'un côté, les architectures réactives de type "subsumption" ne semblent être que peu utilisées dans les domaines nécessitant un grand nombre de fonctionnalités et des prises de décision critiques. En effet, les interactions entre ces fonctionnalités et la planification de leurs activités nécessitent des mécanismes de délibération logique et temporelle. D'un autre côté, on trouve les architectures en couches, qui restent largement utilisées, même si l'on voit ici ou là des variations apparaître. Enfin, de nouvelles approches remettent le "modèle" (unique ou distribué) au coeur de l'architecture, et présentent l'avantage d'assurer une cohérence forte entre les différents composants.
Nous présenterons un rapide panorama de ces architectures, ainsi que leurs avantages et inconvénients respectifs. Au delà de ces architectures, il semble aussi nécessaire de considérer et de discuter les outils de communication et de développement disponibles, ainsi que les acquis des différentes approches sur les diverses plateformes utilisées.

Mots clefs

Architecture logicielle, Robots Autonomes.

1 Introduction

Qu'est ce qu'une architecture pour robot et plus précisément une architecture logicielle pour robots autonomes ?
Une architecture robotique précise autant que possible les composants logiciels et matériels utilisés pour mettre en place un robot, et précise les modalités d'interaction de ces composants. Dans le cadre qui nous intéresse ici, nous nous limiterons aux aspects logiciels de cette architecture (bien que certains aspects matériels puissent être évoqués), et nous considérerons des robots mettant en avant des aspects d'autonomie décisionnelle.
Ceci étant précisé, la question de la nécessité d'une architecture pour la mise en œuvre de robots autonomes reste "entière". La réponse tient principalement dans le fait que les robots sont généralement des systèmes complexes. Ils font intervenir un grand nombre de capteurs et d'effecteurs, un grand nombre de traitements tels que du traitement du signal, du raisonnement géométrique, des processus de planification d'actions ou des boucles de contrôles. Certains de ces traitements doivent se faire en temps réel, alors que d'autres, qui sont en général de complexité exponentielle, prennent un temps "difficilement prévisible".
Quoiqu'il en soit, une architecture logicielle indique comment ces différents composants sont mis en œuvre (c'est à dire dans quel cadre on définit leur contenu), comment ils sont organisés, comment ils communiquent et interagissent.
Quelles sont les principales propriétés attendues d'une architecture ?

Programmabilité Une architecture doit permettre aux robots d'être des machines hautement et facilement programmables (tant du point de vue du programmeur de la machine que celui de l'utilisateur final). Du niveau fonctionnel au niveau décisionnel, il doit être possible de programmer des boucles de contrôles et des traitements de bas niveau, des contraintes de fonctionnement, des procédures d'affinement de buts, et aussi de spécifier des modèles d'actions et pour l'utilisateur des missions à un niveau tâche.

Intégration Les logiciels qui s'exécutent sur le robot sont très divers. Il est important de proposer les bons outils et les bons environnements de développement pour chaque type de composant, et surtout d'offrir des mécanismes de communication et d'échange des données transparents.

Autonomie, Adaptabilité et Cohérence Le robot exécute les actions, affine et adapte ses plans et ses comportements en fonction de ses objectifs et de l'environnement tel qu'il le perçoit. Son comportement et ses réactions sont guidés par ses objectifs.

Réactivité Les différents composants de l'architecture doivent être capables de réagir de façon appropriée aux stimuli spécifiques qu'ils reçoivent.

Robustesse L'architecture doit permettre d'exploiter la redondance des sources d'information, des traitements,

et la multiplicité des processeurs.

Sûreté L'évolution de la robotique de service et l'utilisation de robots dans des situations critiques (pour le robot lui même ou son environnement) requièrent l'utilisation de méthodes qui garantissent certaines propriétés de sûreté.

Extensibilité La modularité de l'architecture doit permettre d'ajouter de nouvelles fonctionnalités, sans remettre en cause l'existant, et ce à quelque niveau que cela soit.

2 Architecture, outils et modèles

Il semble nécessaire de faire un distinguo entre une architecture, les outils et les environnements de développement qui permettent sa mise en œuvre, et les modèles et langages utilisés pour en programmer les composants.

Au premier niveau, une architecture propose une méthode de définition et d'organisation des différents composants qui sont nécessaires à la mise en place d'un robot. Par exemple, elle précisera les couches s'il y a lieu, les fonctions satisfaites à leur niveau, les contraintes temporelles qui leur sont liées, etc.

Au delà de ces spécifications générales qui restent insuffisantes pour mettre en œuvre un robot réel, certaines approches proposent des outils pour la mise en place et la programmation des composants de l'architecture. L'expérience montre que ces outils ont in fine une très grande importance. C'est souvent d'eux que dépend l'acceptation ou pas d'un concept architectural, puisqu'ils doivent en général faciliter la tâche du programmeur, et aussi garantir qu'il restera dans le cadre fixé par l'architecture.

En dernier lieu, la mise en place d'architectures se fait avec l'écriture de modèles pour les éléments et composants formels (modèles d'actions, modèles de pannes, etc) et de codes qui sont intégrés et compilés. Là aussi, le type de modèle et/ou de langage utilisé importe pour faciliter l'expression d'une fonctionnalité particulière.

3 Panorama

On retrouve souvent une classification qui va des architectures purement réactive, Subsumption Architecture [Brooks 1986; Arkin 1990], aux architectures purement délibératives (NASREM[Albus, McCain, & Lumia 1987], NASA/NIST), avec entre les deux les architectures mixtes (RAPs, TCA, Saphira, etc)... Toutefois cette classification ne capture qu'un des aspects (délibération/réaction) de ces architectures, en délaissant les autres (mono-multi robots, outils et environnement de développement, etc).

Il existe de nombreuses architectures logicielles pour des robots autonomes, toutefois, peu d'entre elles prétendent couvrir l'ensemble du spectre "des capteurs effecteurs à la décision", et si certaines le prétendent, encore moins le montre sur des exemples concrets.

Certains systèmes sont présentés dans une optique architecturale mais restent des outils ou des langages plus que des architectures. Ainsi, TDL [Simmons & Apfelbaum 1998] issue du CMU reste plutôt au niveau du langage de contrôle d'exécution et dans une certaine mesure de planification (par un mécanisme d'expansion de plans). Toutefois, malgré ses très grandes qualités TDL n'aide en rien au développement des couches fonctionnelles (même si la couche de communication IPC facilite grandement les interactions entre ces deux niveaux.)

Les approches synchrones sont aussi a l'origine de travaux qui ont influencé les architectures pour robot autonomes : ORCCAD [Simon *et al.* 1993; Coste-Maniere, Espiau, & Simon 1992] repose sur le langage synchrone ESTEREL, mais reste cantonné aux couches de bas niveau de l'architecture. On peut faire une critique similaire à l'approche Controlshell[Schneider *et al.* 1998], peu adaptée à la mise en place de systèmes décisionnels. Ces méthodes peuvent être envisagées comme composant d'une architecture plus générale, mais ne sont pas en soi des architectures.

Notre objectif n'est pas de présenter un tableau exhaustif des architectures existantes et passées. Nous nous proposons plutôt, dans les sections suivantes, de présenter trois architectures qui nous semblent à la fois représentatives de ce que l'on trouve aujourd'hui dans l'état de l'art et qui sont aussi significatives de l'évolution du domaine.

Nous les abordons dans un ordre chronologique qui illustre l'évolution des approches et des outils utilisés. Ainsi nous débutons avec l'architecture LAAS[1] [Alami *et al.* 1998], architecture à trois couches, en développement au LAAS depuis de nombreuses années. La deuxième architecture présentée est CLARAty [Volpe *et al.* Dec 2000]. Issue de travaux de la NASA JPL et du CMU, elle est présentée par ses auteurs comme une évolution des architecture à trois niveaux qui doit faciliter l'infusion du décisionnel dans le système robotisé. Enfin la troisième architecture évoquée est l'architecture IDEA [Muscettola *et al.* 2002] (NASA Ames), plus radicale dans le type d'organisation qu'elle propose : un moteur unifié de planificateur/contrôleur d'exécution temporel déployé dans chaque composant du système, utilisant un modèle de relations/contraintes temporelles pour spécifier le fonctionnement du système.

4 L'architecture LAAS

L'architecture LAAS [Alami *et al.* 1998] a été originellement conçue pour les robots autonomes. Le but de cette architecture est d'offrir une solution générique à la conception et l'intégration de tels systèmes. Elle est supportée par des outils et méthodes de conception offrant des garanties quant à la mise en œuvre de systèmes autonomes au niveau de la spécification, de l'intégration et dans une certaine mesure, de la validation de ceux-ci.

4.1 Une architecture à trois niveaux

Comme l'indique la FIG. 1, l'architecture LAAS est composée de trois niveaux. Chacun d'entre a ses propres

[1] LAAS : *LAAS Architecture for Autonomous Systems.*

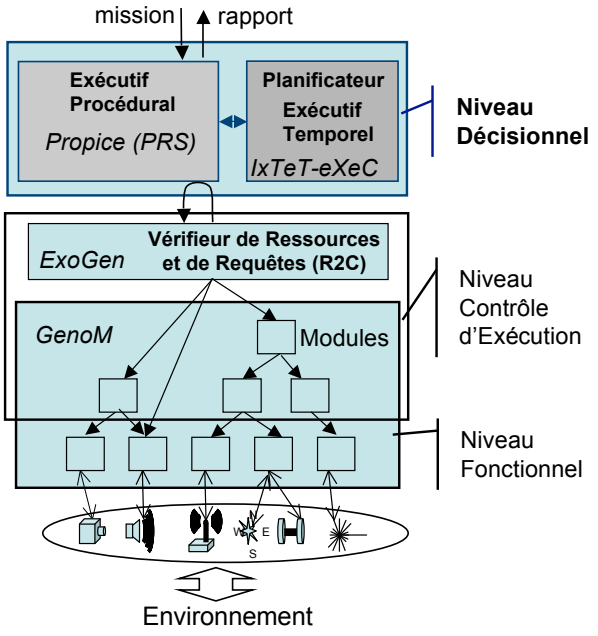

FIG. 1 – L'architecture LAAS

contraintes temporelles et représentations de l'état du système :

– *Le niveau décisionnel :* Ce plus haut niveau intègre les capacités délibératives de l'agent, par exemple : produire des plans de tâches, reconnaître des situations, détecter des fautes, etc. Dans notre cas, il comprend :
  - un exécutif procédural (PRS/Propice [Ingrand *et al.* 1996]) connecté au niveau inférieur auquel il envoie des requêtes qui vont lancer des actions (capteurs/actionneurs) ou démarrer des traitements. Il est responsable de la supervision des actions tout en étant réactif aux événements provenant du niveau inférieur et aux commandes de l'opérateur. Cet exécutif a un temps de réaction garanti.
  - un planificateur/exécutif temporel (dans notre cas IxTeT-eXeC, extension de IxTeT [Ghallab & Laruelle 1994]) chargé de produire et exécuter des plans temporels. Ce système doit être réactif et prendre en compte les nouveaux buts ainsi que les échecs d'exécution (échec d'une action et time-out).
– *Le niveau fonctionnel :* Situé à la base de l'architecture, il est l'interface entre les composants des couches supérieures et la partie physique du système. Il est le siège des fonctions de bases du robot. On y trouve en particulier ses fonctions sensori-motrices, les fonctions de traitement (planificateur de trajectoires, etc) ainsi que les boucles de contrôle (navigation, traitement d'images, capteurs, ...). Chacune de ces fonctions est encapsulée dans un module généré par l'outil GenoM. Chaque module offre un ensemble de services, liés à la fonctionnalité du module, accessibles par ses clients via des requêtes. Les algorithmes sont décomposés en unités de code insécables appelés codels.
– *Le niveau de contrôle des requêtes :* Situé entre les deux niveaux précédents, le R^2C *"Requests and Resources Checker"* [Ingrand & Py 2002] vérifie les requêtes envoyées aux modules fonctionnels (par l'exécutif procédural ou entre modules) et l'utilisation des ressources. Il est synchrone avec les modules (il connaît toutes les requêtes envoyées et tous les bilans retournés et construit en ligne l'état du niveau fonctionnel). Il agit comme un filtre qui rejette éventuellement des requêtes en fonction de l'état et d'un modèle formel donné par l'opérateur spécifiant les états autorisés ou interdits. Les bilans retournés par le niveau fonctionnel sont transmis à l'exécutif procédural après mise à jour de l'état interne. Les contraintes temporelles sont de type temps réel dur.

Détaillons plus avant ces différents niveaux.

4.2 Niveau Fonctionnel

Le niveau fonctionnel regroupe l'ensemble des fonctions opératoires qu'un robot peut être amené à exécuter pour accomplir des tâches et interagir avec son environnement par le biais de capteurs et d'actionneurs. Ces fonctions, dont beaucoup présentent de fortes contraintes temporelles, sont regroupées dans des *modules* fonctionnels temps-réel indépendants : chaque module est responsable d'une ressource (physique ou logique) et dispose à ce titre de l'ensemble des fonctions (algorithmes) nécessaires à son contrôle (y compris les procédures de reprise d'erreur), et d'un contexte d'exécution qui lui est propre. Le module est lui-même contrôlé au moyen de *requêtes asynchrones* qui permettent de démarrer, d'interrompre ou de paramétrer les fonctions dont il prend en charge l'exécution. Le module informe son client de la fin d'un traitement (e.g. surveillance, calcul, filtre, commande, etc) en lui retournant une *réplique* associée à un bilan qualitatif d'exécution. En complément de ce mécanisme de contrôle, un autre protocole de communication assure les flux de données.

La réalisation d'une tâche consiste alors à séquencer, coordonner, paramétrer ces requêtes en fonction des besoins propres de la tâche et de l'état du système caractérisé par les répliques.

Ainsi, nous avons élaboré un *modèle générique de module* capable d'intégrer tant des fonctions synchrones qu'asynchrones : chaque module est une instance particulière de ce modèle. Grâce au générateur de modules GenoM ([Fleury, Herrb, & Chatila 1997]), les modules sont automatiquement synthétisés à partir d'une description formelle des éléments spécifiques du module considéré.

4.3 Niveau Décisionnel

La production et le contrôle de l'exécution des plans sont répartis entre le planificateur/exécutif temporel d'IxTeT-eXeC, et l'exécutif procédural Propice. IxTeT-eXeC est chargé de la planification de la mission globale de l'agent, gérant les objectifs et les ressources sur un horizon à long terme. Son exécutif temporel décide des dates de lancement et éventuellement d'arrêt des tâches (relativement

abstraites) de ce plan global. De la flexibilité est laissée à l'exécutif procédural pour mener à bien l'exécution des tâches fournies par IxTeT-eXeC.

On remarque dans la description du niveau décisionnel que l'exécutif procédural est le composant qui communique avec le niveau inférieur (envoi de requêtes, réception de bilans) et avec l'opérateur (au niveau buts et missions). Ce composant est effectivement conçu pour réagir aux événements et buts et agir en conséquence (voir [Ingrand *et al.* 1996] pour plus de détails sur ces fonctionnalités). Il peut cependant avoir besoin d'un nouveau plan pour accomplir un nouveau but ou récupérer d'un échec quand aucune procédure prédéfinie n'est disponible. Ce plan est fourni (tâche par tâche) par IxTeT-eXeC.

Lors de la réception d'une demande de lancement de tâche par IxTeT-eXeC, Propice la détaille (si nécessaire), et/ou choisit la procédure adéquate pour la réaliser en fonction du contexte courant. Cela entraîne généralement l'envoi de requêtes aux modules fonctionnels. Des échecs peuvent se produire au niveau fonctionnel. Propice dispose d'une certaine flexibilité pour palier ces échecs avant qu'ils ne soient transmis, en dernier recours, à IxTeT-eXeC :

– Tout d'abord, dans Propice, tout sous-but est automatiquement repris en compte tant qu'on n'a pas atteint un échec "complet" (quand toutes les procédures/ unificateurs applicables ont échoué). Par exemple, la tâche *Prendre une image* peut échouer en utilisant une première caméra (défaillante) mais réussir en utilisant une caméra de secours. Ce choix peut être fait par Propice, si cette flexibilité lui a été donnée.
– Ensuite, les procédures sont généralement écrites de façon à tester en ligne la meilleure stratégie d'exécution à suivre en fonction du contexte.

Ces recouvrements locaux d'échecs ne sont pas le fruit de processus de planification, mais correspondent plus à de bonnes pratiques d'ingénierie et de programmation procédurale. De toutes façons, ils participent à la robustesse globale de l'approche.

Propice envoie un bilan à IxTeT-eXeC à chaque fin de tâche (succès ou échec complet). En cas d'échec ou bien, si la tâche n'est pas terminée dans les temps impartis, le plan global n'est plus valide et doit être réparé par IxTeT-eXeC.

Avant de passer à une description détaillée du fonctionnement d'IxTeT-eXeC, nous faisons un bref rappel des principes de base du planificateur.

4.4 Niveau Contrôle d'Exécution

Dans cette architecture on remarque que les couches fonctionnelle et décisionnelle sont en quasi interaction directe et pourraient se suffire à elles-mêmes. Toutefois, nous pensons pour des raisons de robustesse, et de sûreté qu'il est nécessaire d'avoir une couche qui contrôle les requêtes envoyés aux modules.

En effet, le niveau décisionnel est basé sur des outils s'appuyant sur des concepts principalement issus de l'intelligence artificielle. Ceci est nécessaire compte tenu du rôle

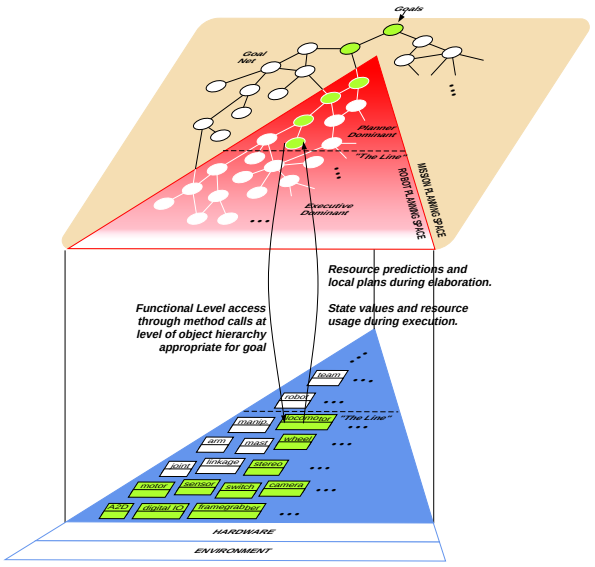

FIG. 2 – L'architecture CLARAty (extraite de [Volpe *et al.* Dec 2000])

de prise de décisions de cette couche, mais présente des contreparties.

De tels outils sont complexes et les développements qui les utilisent – même par des experts du domaine – peuvent mener à des comportements non prévus initialement. De plus, même un planificateur correctement conçu ne donnera qu'un plan valide étant donné sa connaissance de l'état courant et sa spécification du domaine. Une erreur de développement sur un de ces deux points peut avoir des conséquences fatales pour le bon fonctionnement du système.

5 CLARAty

L'architecture CLARAty (Coupled Layer Architecture for Robotic Autonomy) [Volpe *et al.* Dec 2000] développé par NASA JPL et CMU est une architecture à deux niveaux (voir FIG. 2) et se positionne comme une évolution des architectures à trois niveaux. Les arguments principaux avancés par les auteurs sont les suivants :

– Les architectures à trois niveaux ont tendance à cantonner et à cloisonner les activités de type décisionnel au plus haut niveau, sans permettre la mise en œuvre d'algorithmique de ce type au plus bas niveau.
– Il faut un couplage fort entre la couche décisionnelle et la couche fonctionnelle qui peuvent ainsi interagir à tous les niveaux de granularité.

Ainsi, l'architecture CLARAty se compose de deux couches :

5.1 Couche Fonctionnelle

Elle repose fortement sur une approche objets qui doit permettre une forte réutilisablité du code et une extension aisée. La couche fonctionnelle présente trois caractéristiques importantes selon les auteurs :

– elle propose une décomposition modulaire et hiérarchique du système robotique à différents niveaux d'abstraction. Ainsi, une classe pourra fournir une fonction de locomotion et être de plus en plus spécialisée au fur et mesure qu'on s'approche des effecteurs (roues, jambes, etc).
– elle permet dans une certaine mesure de découpler les algorithmes génériques des capacités spécifiques d'un système.
– elle permet d'avoir un modèle d'exécution flexible et non fixé a priori.

5.2 Couche Décisionnelle

Même si cette couche est présente dans la description de l'architecture, et son rôle à peu près bien défini (planifier, ordonnancer et exécuter les plans), son contenu est beaucoup plus "flou". Elle propose l'utilisation d'outils et approches tels que CASPER [Chien *et al.* 2000] ou TDL [Simmons & Apfelbaum 1998] (voir plus haut).
Les interactions avec le niveau fonctionnel ce font à travers un protocole client serveur relativement classique. Ainsi, le niveau décisionnel peut enquérir le niveau fonctionnel sur l'état réel de ressources, mais peut aussi envoyer des commandes à différents niveaux de granularité.

6 IDEA

L'architecture IDEA [Muscettola *et al.* 2002] est proposée par l'équipe de la NASA Ames qui a mis en place le planificateur/contrôleur d'exécution Rax-PS de la sonde DS1. D'une certaine manière, cette architecture résulte et bénéficie de l'expérience acquise lors de cette mission. L'un des principaux enseignements de cette expérience, est la nécessité d'utiliser des modèles cohérents et si possible "similaires" pour les différentes activités (planification, contrôle d'exécution, diagnostic). Ainsi, il a été montré que l'un des problèmes les plus critiques rencontrés sur la sonde DS1 lors de sa mission résultait d'une incohérence entre les modèles utilisés par le planificateur réactif et par le système de diagnostic.
L'architecture IDEA se distingue radicalement des deux premières architectures que nous avons présentées dans le sens ou elle place résolument le décisionnel au "centre". Elle présente les caractéristiques suivantes (FIG. 3 et 4) :
– Elle propose une architecture multi-agents, un agent pouvant être un module fonctionnel, un planificateur, un système de diagnostic, etc.
– Chaque agent repose sur la mise en œuvre d'un couple planificateur/planificateur réactif qui gère un ensemble de variables d'état évoluant dans le temps.
– Chaque agent dispose d'un modèle de contraintes et de compatibilités qui spécifie les évolutions possibles de ces différentes variables d'état.

Ainsi, pour la mise en place d'une expérimentation de robotique mobile [Lemai, Diaz, & Muscettola 2003], on décrit pour les différents modules fonctionnels, les relations de causalité et de dépendances temporelles qui lient les divers traitements. Par exemple, une localisation relative par stéréo odométrie nécessite une séquence de deux images stéréo corrélées, qui elles mêmes nécessitent des paires d'images, etc... La mise en place d'un tel modèle et sa mise en œuvre dans le cadre d'une planification réactive reposant sur ce modèle déroulera la bonne séquence d'actions permettant de satisfaire l'objectif donné.

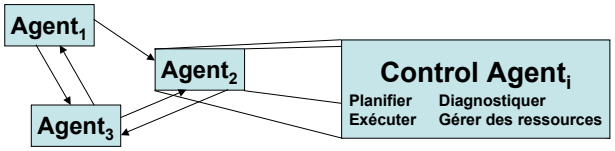

FIG. 3 – Collection d'Agents IDEA

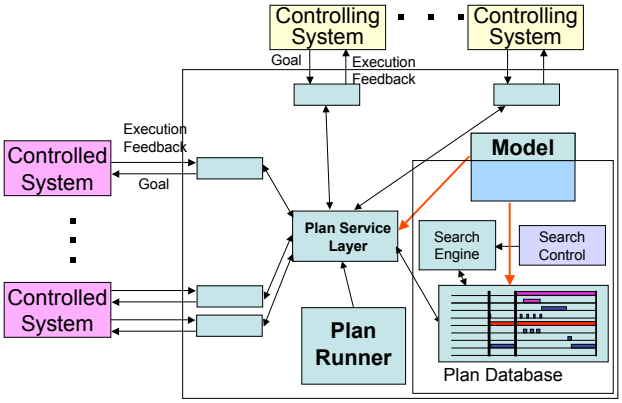

FIG. 4 – Structure d'un Agent IDEA

7 Analyse

Il est toujours difficile de comparer des architectures logicielles sur des aspects objectifs. Après tout, si une jeune équipe de robotique est sensible à la rapidité de mise en œuvre d'une architecture et au très grand nombre de modules fonctionnels déjà disponibles, une équipe de la NASA est sans doute plus attachée aux aspects validation et vérification offerts par ses composants.
Une architecture, pour être utile et utilisable, ne peut pas proposer uniquement des concepts d'organisation. Les outils et les environnements de développement offerts deviennent aussi importants et fondamentaux que l'architecture elle-même.
L'acquis autour d'une approche donnée est aussi important. Les architectures les plus récentes ont peu de "passif" à mettre en avant, alors que les plus anciennes, lorsqu'elles ont été utilisées de façon régulière et soutenue proposent de très nombreuses fonctionnalités. Toutefois, tant qu'une architecture n'a pas atteint une masse critique, elle a peu de poids dans la communauté. Ainsi pour reprendre les trois exemples que nous avons présentés, les auteurs eux même avancent les chiffres suivant :

LAAS Cette architecture a été déployée sur tous les robots du LAAS (7), ainsi que sur des robots de la

NASA (Gromit, et K9). À ce jour, plus de cent modules fonctionnels sont "officiellement" enregistrés, et soixante sont activement utilisés. Ils se répartissent comme suit :

- 13 sont des modules communs a plusieurs plateformes (`platine, camera, sick, locext, nd, ...`)
- 10 modules spécifique au robot H2/H2BIS (`loco, trPilo, us, ...`)
- 14 modules pour le XR4000 et les SCOUTS (`band, teleop, locPost, ra, gvg, ...`)
- 17 modules pour LAMA (Marsokod), DALA (ATRV) et Gromir (ATRV-Jr NASA) (`lloco, p3d, scorrel, steo, nav, ...`)
- 5 modules pour le bras GT6A et le robot de laparoscopie (`arm, piloArm, girobo, coord, endo`)
- 5 modules fonctionnels de démos et TPS (`demo, away, dloco, dmon, dzone`)
- Autres : 11 pour MARTHA (`smc, ci, irc, ...`), 13 pour SYDRE, 12 caducs (`trLoco, site, azi, tele3d, loca2d, ...`)

CLARAty CLARAty est vue à ce jour comme l'architecture du futur pour les robots de la NASA. Ainsi, elle est devenue en quelques années un passage quasiment obligé, et de nombreuses plateformes de la NASA l'utilisent : Rocky 7 et 8, K9 et Fido, ATRV avec des taux de réutilisabilité de l'ordre de 80%.

IDEA Compte-tenu de son développement récent, cette architecture n'a été utilisée que sur quelques systèmes : deux robots de la NASA (K9 et un ATRV : Gromit). Toutefois, il est intéressant de noter que l'équipe qui la développe réimplémente complètement le planificateur, le contrôleur d'exécution et le module de FDIR de DS1.

Une architecture doit aussi s'adapter et évoluer en fonction des outils paradigmes de développement de logiciels (objets, CORBA, UML, etc).

Entre CLARAty et l'architecture LAAS, il y a in fine relativement peu de différences. Le souhait louable de permettre une plus grande interaction entre le décisionnel et le fonctionnel ne semblait pas un réel frein dans l'architecture LAAS, et ne semble pas avoir fait progresser notablement la présence du décisionnel dans les applications de CLARAty.

A contrario, l'architecture IDEA se pose résolument comme une nouveauté dans le paysage des architectures pour robots autonomes. Elle replace le décisionnel au centre du robot, et pousse ses modèles vers les modules fonctionnels. Toutefois, de nombreux problèmes restent à résoudre pour déployer une telle architecture sur un robot (en particulier l'encapsulation des modules fonctionnels, la communication entre les différents agents, et les performances...). Par contre, la mise en avant du décisionnel, permet de le déployer beaucoup plus facilement. Car il faut bien reconnaître que même si la difficulté n'est pas insurmontable, encore peu d'applications de robotique mettent en place des systèmes, par exemple, de planification de tâches.

Un point important à noter dans les architectures récentes, est qu'elles bénéficient en général des dernières innovations en matière d'outil de développement. Ainsi CLARAty repose largement sur une approche objet. De la même façon, IDEA utilise ACE/TAO pour les communications inter processus et pour diverses autre fonctionnalités de base.

8 Conclusion

Nous avons justifié la nécessité d'utiliser une architecture logicielle pour déployer des robots autonomes. Cette nécessite est d'autant plus forte que la complexité de la plate forme et la complexité des décisions prises sont importantes.

Du panorama rapide que nous avons présenté, nous avons dégagé trois architecture représentatives de l'état de l'art, et de son évolution. L'objectif n'est pas d'en sélectionner une par rapport à une autre, mais plutôt de dégager les axes forts de développement de ce domaine de recherche.

Les architectures qui présente une structure en couches décisionnelle/fonctionnelle semblent encore être les plus utilisées. Il faut reconnaître que cette séparation correspond aussi à un plan de coupe dans la communauté scientifique (robotique "pure"/intelligence artificielle).

D'autres leçons peuvent être tirées de ce tour d'horizon :

- Compte tenu de l'évolution de la robotique (vers la robotique de service, et vers des missions critiques), et compte tenu des résultats de recherche dans le domaine du génie logiciel et des méthodes formelles de spécification et validation, il est souhaitable que les architectures logicielles pour la robotique considèrent d'avantage certaines de ces approches (e.g. automates temporisés [Asarin *et al.* 1998]).
- Le succès d'une architecture dépend aussi de sa disponibilité, et de celle de ses composants. La mise à la disposition de la communauté des outils de développement liés à CLARAty et à IDEA n'est pas encore clarifiée, et ne peut que retarder l'acceptation de ces approches. A contrario, des projets comme le projet européen Orocos (`http ://www.orocos.org/`) propose des logiciels libres pour la mise en œvre d'architecture (du contrôle de bas niveau dans le cas d'Orocos).

Au delà de ces considérations, il semble aussi opportun de considérer l'intégration de l'apprentissage et d'un meilleur traitement de l'incertitude dans les architectures pour les robots autonomes. Enfin un dernier point est celui de l'interaction avec l'humain dans le cadre d'un partage de l'environnement mais aussi d'un résolution interactive des problèmes.

Références

Alami, R. ; Chatila, R. ; Fleury, S. ; Ghallab, M. ; and Ingrand, F. 1998. An architecture for autonomy. *Internatio-*

nal Journal of Robotics Research, Special Issue on Integrated Architectures for Robot Control and Programming 17(4) :315–337.

Albus, J. ; McCain, H. ; and Lumia, R. 1987. NASA/NBS standard reference model for telerobot control system architecture (NASREM). Technical Report 1235, NBS.

Arkin, R. C. 1990. Motor Schema-Base Mobile Robot Navigation. *International Journal of Robotics Research*.

Asarin, E. ; Maler, O. ; Pnueli, A. ; and Sifakis, J. 1998. Controller synthesis for timed automata. In *Proc. System Structure and Control*, 469–474. IFAC.

Brooks, R. A. 1986. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation* RA-2(1) :14–23.

Chien, S. ; Knight, R. ; Stechert, A. ; Sherwood, R. ; and Rabideau, G. 2000. Using iterative repair to improve the responsiveness of planning and scheduling. In *Proceedings of the International Conference on Artificial Intelligence Planning and Scheduling (AIPS)*.

Coste-Maniere, E. ; Espiau, B. ; and Simon, D. 1992. Reactive objects in a task level open controller. In *Proceedings of the International Conference on Robotics and Automation*.

Fleury, S. ; Herrb, M. ; and Chatila, R. 1997. Genom : a tool for the specification and the implementation of operating modules in a distributed robot architecture. In *Proceedings of the International Conference on Intelligent Robots and Systems*.

Ghallab, M., and Laruelle, H. 1994. Representation and Control in Ixtet, a Temporal Planner. In *Proceedings of the International Conference on Artificial Intelligence Planning and Scheduling (AIPS)*, 61–67.

Ingrand, F., and Py, F. 2002. An Execution Control System for Autonmous Robots. In *IEEE International Conference on Robotics and Automation*.

Ingrand, F. ; Chatila, R. ; Alami, R. ; and Robert, F. 1996. PRS : A High Level Supervision and Control Language for Autonomous Mobile Robots. In *IEEE International Conference on Robotics and Automation*.

Lemai, S. ; Diaz, B. ; and Muscettola, N. 2003. A real-time rover executive based on model-based reactive planning. In *Proceedings of International Conference on Advanced Robotics*.

Muscettola, N. ; Dorais, G. A. ; Fry, C. ; Levinson, R. ; and Plaunt, C. 2002. Idea : Planning at the core of autonomous reactive agents. In *Proceedings of the 3rd International NASA Workshop on Planning and Scheduling for Space*.

Schneider, S. A. ; Chen, V. W. ; Pardo-Castellote, G. ; and Wang, H. H. 1998. Controlshell : a software architecture for complex electromechanical systems. *International Journal of Robotics Research* 17.

Simmons, R., and Apfelbaum, D. 1998. A task description language for robot control. In *Proceedings of the International Conference on Intelligent Robots and Systems*.

Simon, D. ; Espiau, B. ; Castillo, E. ; and Kapellos, K. 1993. Computed-aided design of a generic robot controller handling reactivity and real-time control issues. *IEEE Transaction on Control Systems Technology* 1(4).

Volpe, R. ; Nesnas, I. ; Estlin, T. ; Mutz, D. ; Petras, R. ; and Das, H. Dec. 2000. Claraty : Coupled layer architecture for robotic autonomy. *Technical Report, Jet Propulsion Laboratory*.