PAR AKIM DEMAILLE (90)

directeur du développement d'Urbi à Gostai

Urbi : un « operating system » de conception **française**

Les défis du logiciel sont encore plus grands que ceux du matériel, même s'ils sont moins spectaculaires ou médiatiques. Les robots sont bien autre chose qu'un ordinateur « auto-mobile ». Ils sont composites, assemblages de composants matériels ou logiciels d'origines diverses qu'il faut interconnecter et asservir. Un monde complexe d'interactions et de dépendances dont la partition est multiple, mais dont l'ensemble, correctement orchestré, aboutit au comportement global du robot.

Comme un ordinateur, un robot a besoin d'un « OS », ou *Operating System*, qui, à l'image de Linux ou Windows, va rythmer le cœur du système. Mais il a besoin d'un OS bien spécial, ouvert et adapté aux besoins de la robotique. C'est ce que propose *Gostai* avec le système « Urbi ».

La motivation première a été de proposer un système unique pour contrôler des robots très différents, afin d'apporter un socle commun pour le développement d'applications robotiques génériques, un peu sur le modèle des PC dans l'informatique.

Stimuler la créativité des très nombreux passionnés

REPÈRES

Urbi n'est pas le seul projet qui cherche à proposer un standard pour le logiciel robotique. Microsoft a lancé une offre de *middleware* robotique. Des initiatives académiques comme Player/Stage existent de longue date. Des concurrents apparaissent régulièrement aux États-Unis, au Japon et en Corée. Cette diversité témoigne de l'intérêt porté par l'industrie pour les *middlewares robotiques*, et de l'enjeu important qui y est lié. Il n'y a toujours pas aujourd'hui de standard « de fait » pour les robots, mais des progrès très importants ont été réalisés.

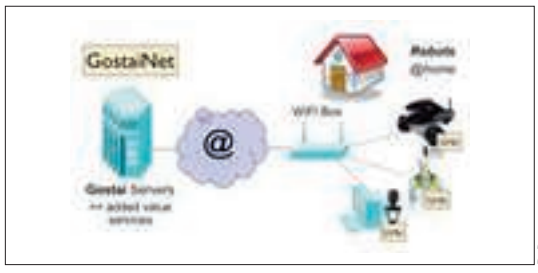
Cette capacité d'adaptation suppose une flexibilité et une ouverture importante, pour épouser les interfaces et morphologies extrêmement diverses que l'on peut retrouver dans les robots : robots à roues, robots humanoïdes, robots industriels, drones volants, etc.

Une jeune société

Urbi vient du monde de la recherche. Il a été développé à partir de 2004 par Jean-Christophe Baillie (94) au laboratoire de robotique cognitive de l'ENSTA ParisTech, avant d'être porté ensuite par la jeune pousse *Gostai* qu'il a fondée en 2006. La société compte aujourd'hui une équipe de vingt personnes. Elle a créé une première filiale à Palo Alto en Californie et entretient de nombreux contacts au Japon et en Corée. Urbi est aujourd'hui appliqué à des robots industriels, des robots grand public tels que le *Nao* ou le *Spykee*, et dans le cadre de recherches académiques et de projets européens.

Simple et accessible

Une autre motivation était de construire un outil de programmation simple et accessible. Ce souci de simplicité dans la conception d'Urbi est resté au cours du temps, alors que le système se dotait de fonctionnalités de plus en plus avancées, et reste aujourd'hui encore une ligne directrice importante. Le principe est qu'il doit être possible de prendre en main un robot en moins de cinq minutes sur la base de connaissances élémentaires en informatique. Le but est de permettre le développement



D.R.

d'une communauté qui ne soit pas réservée à des experts mais ouverte à tous, pour stimuler la créativité des très nombreux passionnés qui s'intéressent aujourd'hui à la robotique dans les clubs, les écoles ou les universités. Nous avons pour cela ouvert un site, urbiforge.org, dédié aux échanges entre utilisateurs et permettant de mettre en avant les projets les plus significatifs.

L'approche « composants »

Dans un robot, on peut segmenter les fonctions fondamentales en unités, que l'on appelle « composants », ou parfois aussi « services ». Le premier travail d'un OS robotique est de fournir une base pour décrire ces composants. Dans l'univers Urbi, cet outil s'appelle « UObject » et permet d'écrire en langage C++ ou Java une brique élémentaire de fonctionnalité, comme, par exemple, la synthèse de la parole, le calcul de l'équilibre, ou le contrôle d'un moteur.

Pour la plupart des développeurs, le monde de la « programmation orientée objet » (POO) est un monde familier. « UObject » s'en inspire pour permettre très facilement d'intégrer un objet (au sens de la POO) dans Urbi et de le rendre immédiatement disponible pour le robot. Cela permet de réutiliser des milliers de composants existants très facilement et d'éviter de démarrer sur une table rase.

On regroupe habituellement les composants en deux catégories : d'une part les composants bas niveau, ou *drivers,* et d'autre part les composants fonctionnels, fournissant une interface vers une fonctionnalité particulière, comme le traitement d'images ou de la voix, ou un calcul complexe.

Pour adapter Urbi à un robot particulier, il suffit de développer tout ou partie des composants bas niveau spécifiques à ce robot pour donner accès aux moteurs, caméra et autres capteurs. Un ensemble de conventions publiques permet ensuite de définir la forme et les nommages à adopter de telle sorte que le résultat soit standard. Il faut également définir et adapter des fonctions dites de haut niveau pour spécifier comment avancer, reculer, se positionner, etc., pour que le programme final soit le moins dépendant possible de la morphologie du robot.

L'orchestration

Une fois tous ces composants définis et intégrés, il reste une étape importante pour que le robot s'anime : il faut les interconnecter et définir leurs relations, leurs liens et dépendances. C'est ce qu'on appelle l'orchestration. L'orchestration permet de définir les flux de données d'un composant à un autre, les priorités d'exécution, les réactions du système à tel ou tel événement, en un mot : le comportement du système. Bien souvent, l'orchestration est réalisée « en dur », de manière figée et directement liée aux composants : chaque composant connaît les autres composants et s'adresse à eux directement d'une manière décidée dès le départ.

Une partition d'orchestre

D'autres approches privilégient la souplesse et définissent l'orchestration à l'aide d'un langage de script séparé des composants eux-mêmes. Un peu comme une partition d'orchestre. On retrouve souvent pour cette tâche des langages comme Python ou LUA, qui sont également utilisés dans ce cadre pour l'orchestration de jeux vidéo, domaine qui partage de nombreux points communs avec la robotique.

L'approche proposée dans Urbi, et qui constitue le cœur de son innovation, est d'utiliser un langage spécialisé pour cette tâche d'orchestration : le langage *urbiscript.* Ce langage est semblable à ses vénérables collègues sur de nombreux points (orienté objet, dynamique et réflexif, syntaxe proche de C, etc.), mais il propose deux nouveautés cruciales pour faciliter son travail de chef d'orchestre : le parallélisme et la programmation événementielle.

Un parallélisme omniprésent

Le parallélisme est essentiel car un robot gère communément des dizaines, voire des centaines d'activités en même temps. Bien sûr, nos ordinateurs actuels en sont déjà capables, mais ils le font en général au niveau des applications elles-mêmes. La distinction est liée ici à la finesse dans le niveau de granularité du parallélisme. Ce que propose *urbiscript* c'est une gestion du parallélisme à un niveau plus fin, celui du code d'exécution lui-même, ce qui n'est classiquement possible que *via* des techniques telles que la programmation par *threads.* Or, l'approche *threads* est notoirement complexe et pose de véritables problèmes de montée en complexité, ce que propose de résoudre *urbiscript* en fusionnant ces concepts directe-

L'orchestration est nécessaire pour que le robot s'anime

Le suivi de balle

```
balltag: whenever (ball.visible)
  {
     headYaw.val   += camera.xfov * ball.x
   & headPitch.val += camera.yfov * ball.y
  };
```

Ces quelques lignes comprennent certains des aspects les plus notables de l'architecture Urbi et du langage *urbiscript*. Le mot-clé *whenever* introduit une sorte de *if* continu et événementiel : à chaque fois que la condition est vérifiée, le corps est exécuté. Celui-ci est composé ici de deux instructions mises en parallèle par le connecteur *&*. Chacune des deux instructions asservit un axe de la tête du robot (en angle) à la position d'une balle dans l'image (en coordonnées cartésiennes). L'UObject *camera* est utilisé pour la conversion cartésien/angulaire, les UObjects *headYaw*, *headPitch* et *camera* sont des abstractions du matériel. Ils sont utilisés en lecture pour l'acquisition *(camera.xfov)*, en écriture pour le mouvement *(headYaw.val')*. L'UObject *ball* est algorithmique ; il fournit des événements (tels que la détection de la balle) et des procédures telles que le calcul de sa position. Enfin, *balltag* est ici un *tag* qui permet à tout moment d'interrompre l'exécution du code qui le suit par un appel du type : « balltag.stop » ou « balltag.freeze/unfreeze » pour geler et reprendre l'exécution. Les *tags* forment un outil puissant dans *urbiscript* pour contrôler le flux d'exécution de centaines de portions de code qui fonctionnent en parallèle.

Un robot réagit sans cesse à des événements, extérieurs ou internes

ment dans le langage. Par exemple, en *urbiscript*, pour exécuter A et B l'un après l'autre on écrira, comme dans beaucoup de langages, « A ; B » et pour les exécuter en parallèle, on écrira tout simplement « A & B ».

Décrire des comportements robotiques

Le rôle d'Urbi et *d'urbiscript* est donc de fournir des abstractions (composants, parallélisme, événements, contrôle de flux) afin de simplifier la tâche de l'ingénieur ou du chercheur lorsqu'il écrit des programmes pour des robots. On peut aller encore plus loin et, au-delà des abstractions fonctionnelles, proposer des abstractions structurelles : comment décrire un comportement de robot ? Le sujet est très vaste et fait l'objet de thèmes de recherches très variés depuis de nombreuses années. Parmi les nombreuses approches existantes, celle que l'on a retenue est celle des « graphes d'états finis hiérarchiques », une technique classique par ailleurs pour décrire le fonctionnement d'un système complexe, et qui est souvent utilisée en robotique. *Gostai Studio* est une application conçue pour permettre de créer et de manipuler ces graphes.

La programmation événementielle

La programmation événementielle est également un fondement de n'importe quel programme robotique. Un robot réagit sans cesse à des événements, extérieurs ou internes, et doit gérer les interactions entre toutes ces sollicitations… en parallèle. Les deux aspects, événementiel et parallélisme, se complètent naturellement. Par exemple, en *urbiscript*, pour faire une action A quand x est égal à 42, on écrira simplement : « at (x==42) A ».

La robotique à bas coût

Déployer une application robotique complexe pose des problèmes pratiques : comment faire tenir dans un robot à la puissance limitée les algorithmes sophistiqués nécessaires pour traiter l'image, la voix, le son, la localisation, etc. ? La solution que nous avons imaginée est de déporter ces algorithmes, intégrés dans UObject, et de les faire fonctionner sur des serveurs puissants, à distance. C'est ce qu'on appelle communément le *cloud computing*, mais appliquée ici à la robotique.

D.R.

D.R.

© D.R.

Un robot gère des centaines d'activités en même temps

Demain l'open source

Urbi se veut donc une tentative de fournir un ensemble complet pour les besoins logiciels de la robotique : une architecture de composants, un système d'orchestration, des outils de développement graphiques et une plateforme d'exécution type *cloud computing* pour les besoins hors embarqué. Le premier niveau de l'édifice, les composants et leur orchestration, correspond globalement à un *Operating System* robotique. Or, aujourd'hui, à l'heure d'*Android* et de *Maemo,* l'industrie exige de l'ouverture pour les composants clefs des systèmes embarqués, ce qui est compréhensible, en particulier pour un OS, car il s'agit d'une partie centrale et vitale du système.

Dans cet esprit, nous avons annoncé le passage d'Urbi en *open source* (licence compatible GPL), ce qui va permettre d'asseoir l'ensemble des solutions proposées par *Gostai* sur une base ouverte, pérenne et favorable au développement d'une communauté. Alors que les États-Unis, le Japon et la Corée préparent chacun leur propre OS *open source* pour les robots, Urbi se positionne aujourd'hui comme une alternative européenne sur un marché qui sera probablement demain un enjeu clé pour l'industrie. ■