MASL, langage de contrôle multi-agents robotiques

Michel Dubois

8 Décembre

Laboratoire VALORIA, Université Européenne de Bretagne

sous la direction de Dominique Duhaut
et de Yann Le Guyadec

Plan

1. Introduction & problématique
   - Domaines de la robotique collective
   - Problématique
   - Contraintes du domaine
2. Modèles d'exécution d'un composant robotique.
   - API et objets
   - Architectures logicielles de contrôle
3. Le modèle de programmation MASL
   - Agents hétérogènes
   - Synchronisation
   - Communication
   - Changement dynamique de rôle
   - Évolution des capacités
   - Extensibilité
4. Implantation MASL
   - Scénarios de déploiements
   - Traduction de MASL vers langage cible
5. Perspectives et conclusions

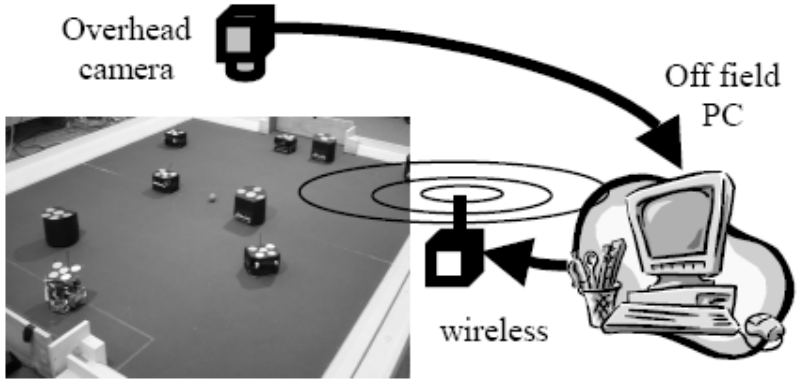
1. Définitions Robot / Agents / SMA / SMR

- Nous voulons programmer des robots. Notre travail se focalise sur les SMR (systèmes Multi-Robots).

- Les agents sont la bonne abstraction pour programmer des robots.
- Un agent est une entité logicielle qui fonctionne continuellement d'une manière autonome dans un environnement.
- Il peut être basé
  - sur le modèle réactif (il interagit avec son environnement via des capteurs et des actionneurs)
  - sur le modèle proactif (il raisonne à partir de sa propre représentation du monde).
- Lorsque l'environnement contient d'autres agents, on parle de SMA (Système Multi-Agents) : le caractère social des agents recouvre leur aptitude à communiquer, coopérer et collaborer à une mission commune.

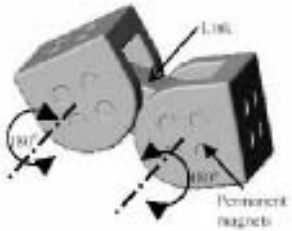
1. Motivation : RoboCup

- Robocup ?
- Problème : exprimer le programme pour N éléments



1. Motivation : la robotique collective

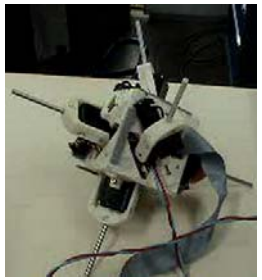
PolyPod Xerox



MTRAN AIST

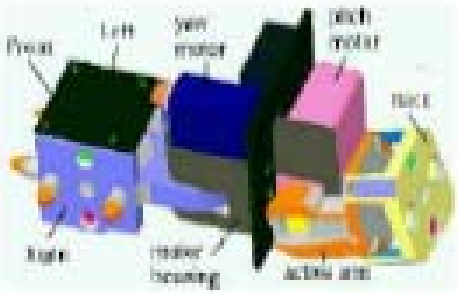
Molecule robot

Darmouth College



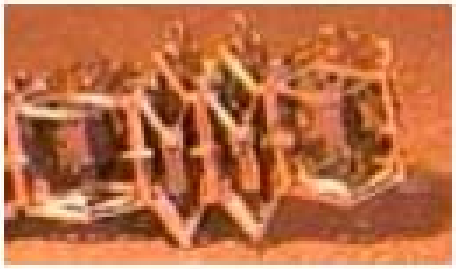
MAAM

UBS



Conro USC

I-CUBE CMU

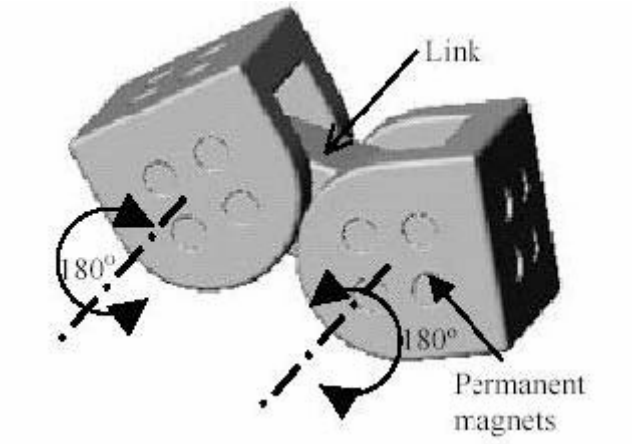


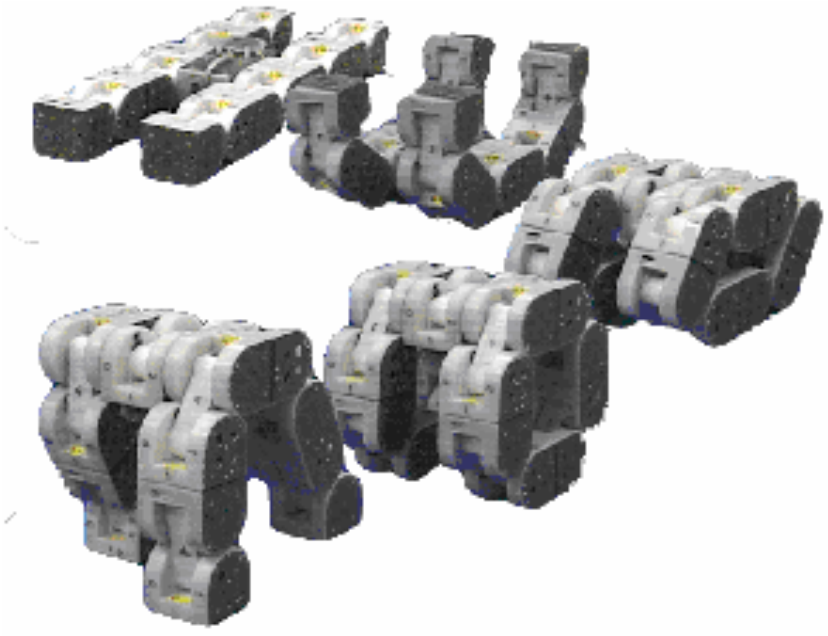
PolyPod Stanford

1. Systèmes auto reconfigurables

Pourquoi reconfigurer ?

Problème exprimer le programme pour N éléments





1. Problème

Définir un langage de programmation pour exprimer le comportement d'un ensemble de robot (d'agents)

1. Définir un langage de contrôle ?

Mise en place du cahier des charges

- Identifier des critères de comparaisons
- Pour chaque langage SMA existant pour la robotique, utiliser ces critères pour son positionnement.
- Chercher le(s) langages SMA pour la robotique qui satisfait ces critères.

Les critères de comparaison ?

- Ce qui manque au programmeur pour faciliter sa tâche.

1. C1 : Hétérogénéïté des agents

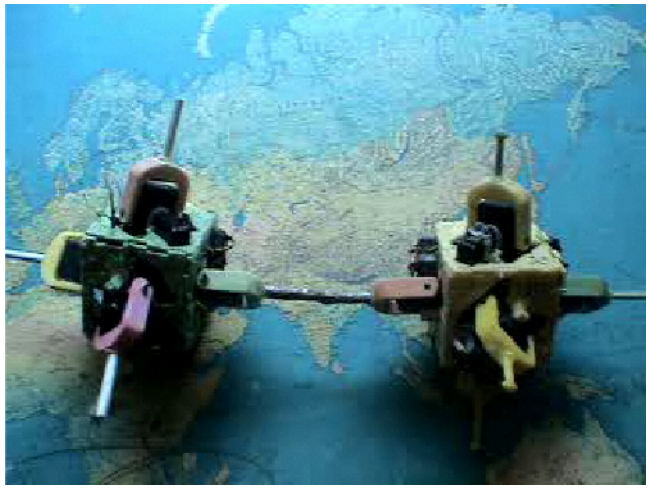
- La nécessité de faire travailler les robots existants, que l'on à sa disposition.
- L'hétérogénéité des agents complique la programmation d'une flotte de robots.
- Le risque pour le programmeur est de se focaliser trop sur les différences d'agents, ce qui complique la programmation de la mission.

Ex : missionLab [MissionLab, 97]

1. C2 : La synchronisation

- Les agents situés physiquement nécessitent des synchronisation dans leurs mouvements.

- Dans un programme d'agent, on souhaite alterner des phases
  - où l'on souhaite faire travailler de concert un sous groupe d'agents et des
  - où on laisse travailler les agents de manière indépendantes.



1. C3 : Moyens de communication

- Offrir plusieurs moyens de communications
  - Communication indirecte (agents réactifs, peu de capacité de communication)
  - Communication directe par variables partagées (Blackboard)
  - Communication directe par diffusion d'évènements (ROM [Boussinot & all, 95] (Reactive Object Model))
  - Communication directe par envoi synchrone de messages (modèle objet classique)
  - Communication directe par envoi asynchrone de messages (modèle d'acteurs [Hewitt, 77])

1. C4 : Changement dynamique de rôle (groupe).

- Les agents doivent jouer plusieurs rôles dans la mission.
- La reconfiguration rend nécessaire la réaffectation des rôles au sein d'un groupe.

- Au sein d'une tactique, les robots footballeurs peuvent changer de rôle
- Les tactiques d'une équipe de robots footballeurs changent en fonction de la perception de celle en cours chez l'équipe adverse.

1. C5 : Évolution des capacités

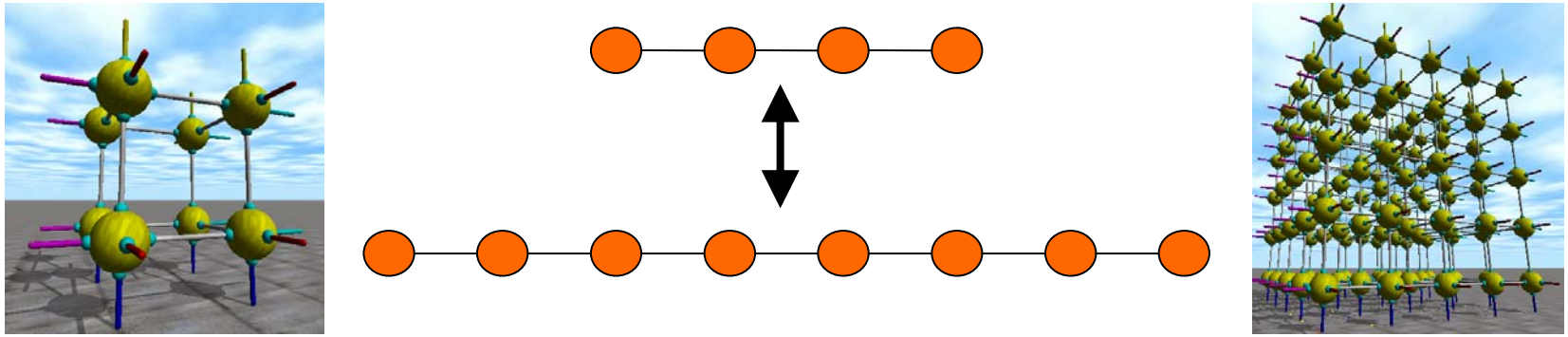
- Les agents ont leur propre programme qui consiste à jouer des rôles en collaboration avec d'autres au sein de la formation.

- Les agents connaissent leur état et maintiennent leur vitalité. Ils peuvent fonctionner **en modes dégradés**.

- La collaboration avec les autres agents est soumise à sa **capacité actuelle**, à sa « proximité » avec les agents initiateurs de collaboration qui lui permet de juger de son **caractère prioritaire**.

1. C6 : Extensibilité

- Les algorithmes ne devraient pas être dépendants du nombre de robots



- Facilité de déboguage, de test, permet la réutilisation d'algorithme.

1. Langages existants

Légende : E: évènements, SV: Variables partagées, MP: envoi de messages, +: supporté, -: non supporté, rien: non documenté.	Hétéro--généïté	Synchro--nisations	Communi--cations	Changement dynamiques de rôles et de groupes	Capa--cités dyna--miques	Extensi--bilité
CHARON : [Alur & all, 00] (Coordinated control, Hierarchical design, Analysis, and Run-time mONitoring of hybrid systems)	+	Pas aussi précis	SV, MP, E-	Pas aussi précis	+	-
CCL [Klavins, 03], [Waydo & all, 03] (Computation and Control Language)	+	Pas aussi précis	SV, MP	Pas aussi précis	+	+
MRL [Nishiyama & all, 98] (Multiagent Robot Language)	+	Pas aussi précis	E, SV, MP		+	-
Tapir [King, 02]	+	Pas aussi précis	MP, SV		+	-
GOLOG (alGOl in LOGic) [Levesque & all, 97]	+	Pas aussi précis	E, SV		+	-
CDL [MacKenzie & all, 97] (Configuration Description Language)	+	Pas aussi précis	E, MP	-	-	+
XABSL [Lötzsch, 04] (eXtensible Agent Behavior Specification Language)	+	Pas aussi précis	E, MP	Pas aussi précis	-	+

Plan

- Introduction
  - Domaines de la robotique collective
  - Problématique
  - Contraintes du domaine

- **Modèles d'exécution d'un composant robotique.**
  - **API et objets**
  - **Architectures logicielles de contrôle**

- Le modèle de programmation MASL
  - Agents hétérogènes
  - Synchronisation
  - Communication
  - Changement dynamique de rôle
  - Évolution des capacités
  - Extensibilité

- Implémentation MASL
  - Scénarios de déploiements
  - Traduction de MASL vers langage cible

- Perspectives et conclusions

2. Les exécutifs robotiques

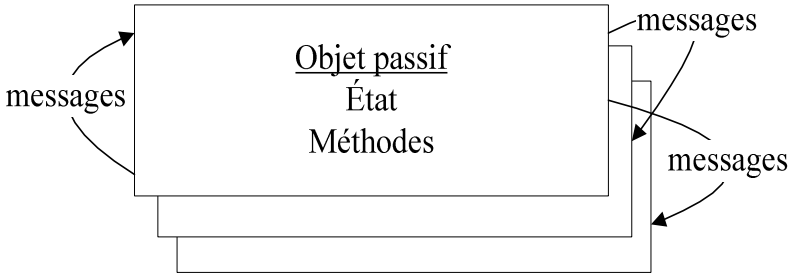
- Hétérogénéïté de matériel mais il existe aussi une hétérogénéïte logicielle :
  - du point de vue des API
  - du point de vue des architectures logicielles

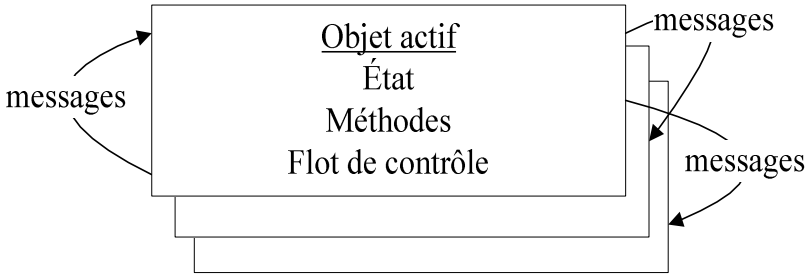
2. API : Trois types d'objets

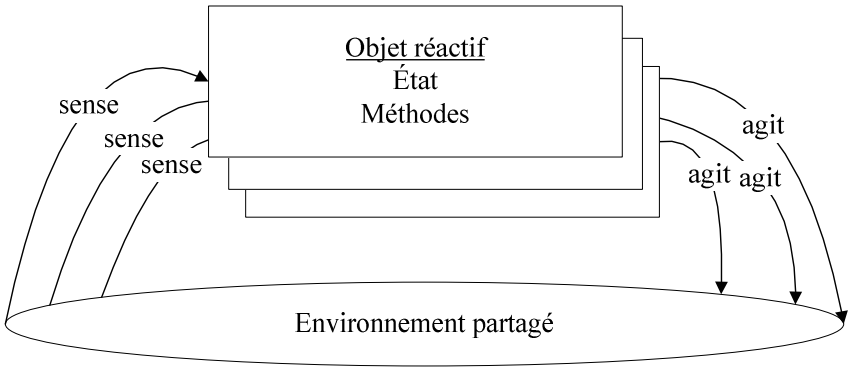
- Un objet passif exécute ses méthodes suite à la réception d'un message. Le flot de contrôle est partagé. C'est le cas classique.

- Un objet actif possède son propre flot de contrôle pour son autonomie.

- Un objet réactif exécute une méthode suite à la perception d'un évènement. Tous les objets du système sont réactifs même si leur cycles réaction/action peuvent ne pas être synchronisés.

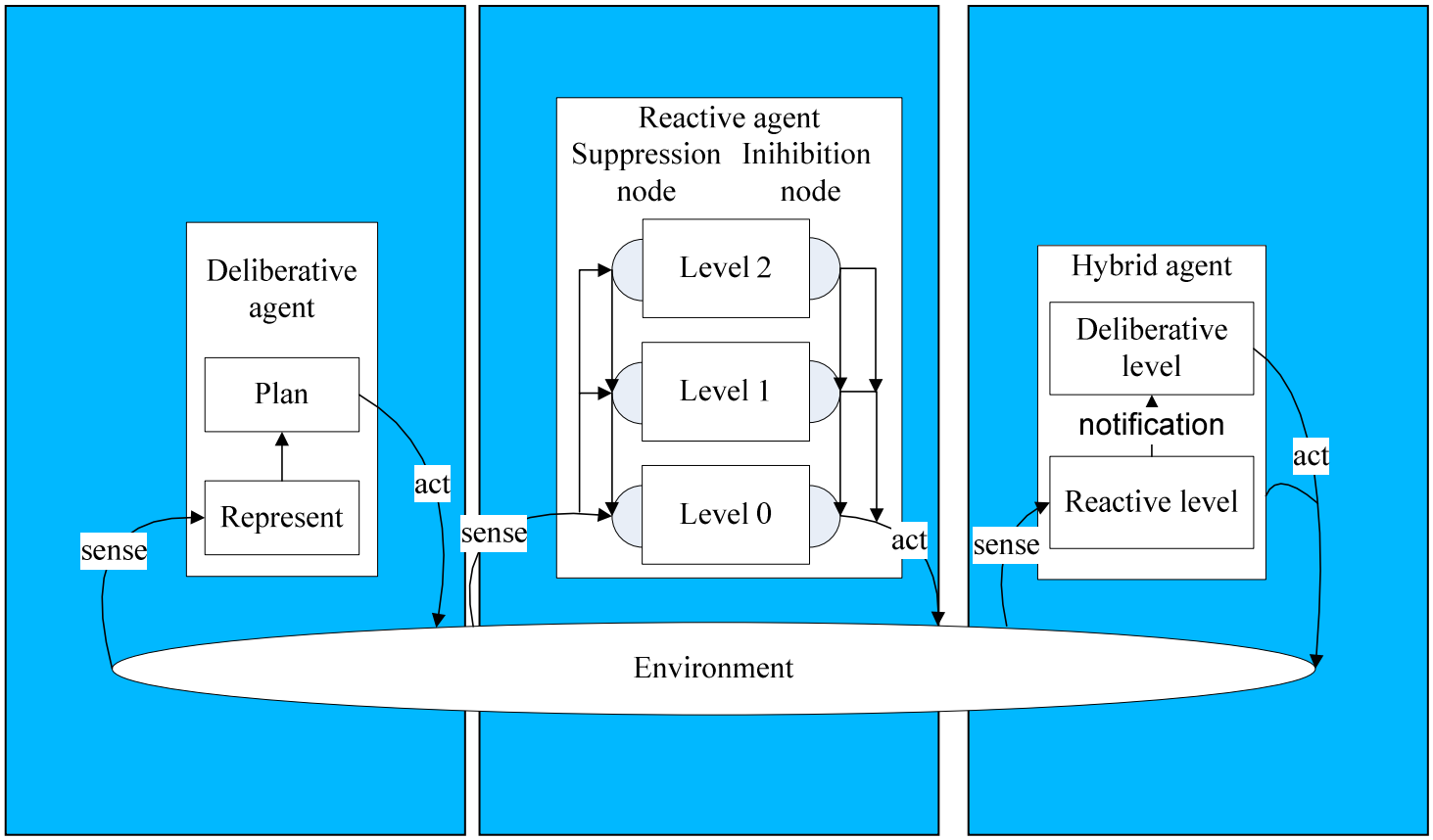






2. Architectures logicielles

- Trois types d'architectures logicielles d'agents



Plan

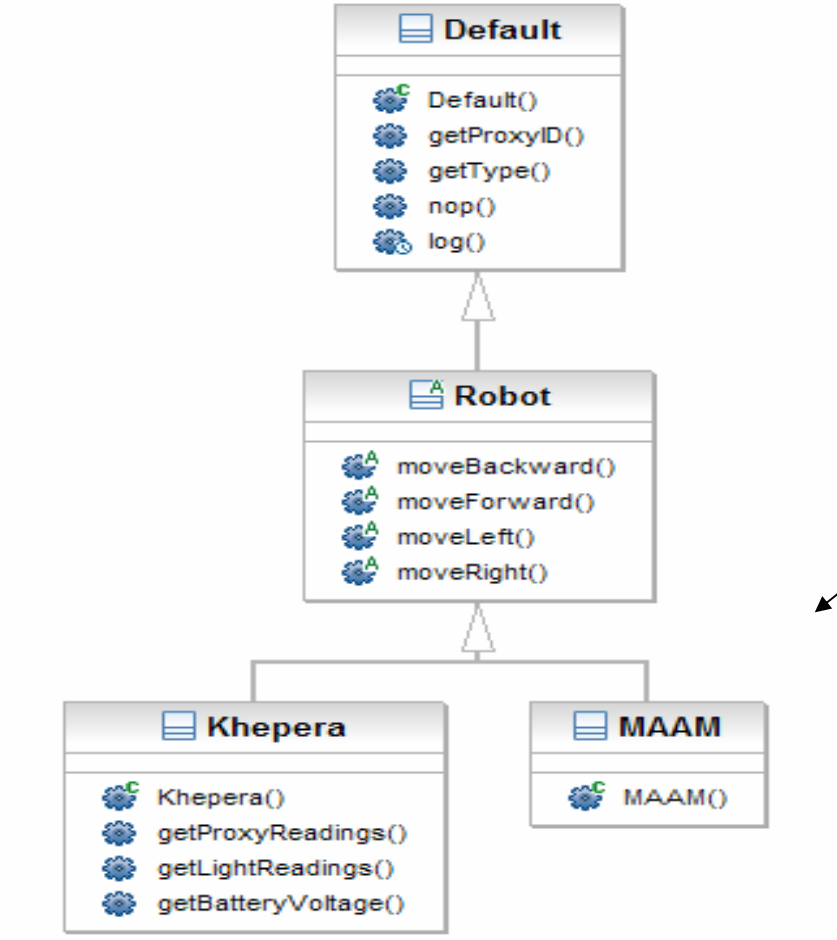
- Introduction
  - Domaines de la robotique collective
  - Problématique
  - Contraintes du domaine
- Modèles d'exécution d'un composant robotique.
  - API et objets
  - Architectures logicielles de contrôle
- **Le modèle de programmation MASL**
  - **C1 Agents hétérogènes**
  - **C2 Synchronisation**
  - **C3 Communication**
  - **C4 Changement dynamique de rôle**
  - **C5 Évolution des capacités**
  - **C6 Extensibilité**
- Implémentation MASL
  - Traduction de MASL vers langage cible
  - Scénarios de déploiements
- Perspectives et conclusions

3. C1 : Agents hétérogènes

Objectif : nous voulons programmer un ensemble d'agents hétérogènes qui travaillent ensemble.

Pour cela on importe la description XML des fonctionnalités des agents (A.P.I.) dans un type MASL.

3. C1 : Agents hétérogènes



Des primitives de même noms sont possibles

3. C1 : Agents hétérogènes

Hypothèse forte : ce que savent faire les agents est connu au départ.

```
01| import Khepera.xml as Khepera;
02| import MAAM.xml    as MAAM;

03| Khepera k1,k2    = newAgent(Khepera);
04| MAAM    m1,m2,m3 = newAgent(MAAM);
```

Nouveau type

Type

Variables

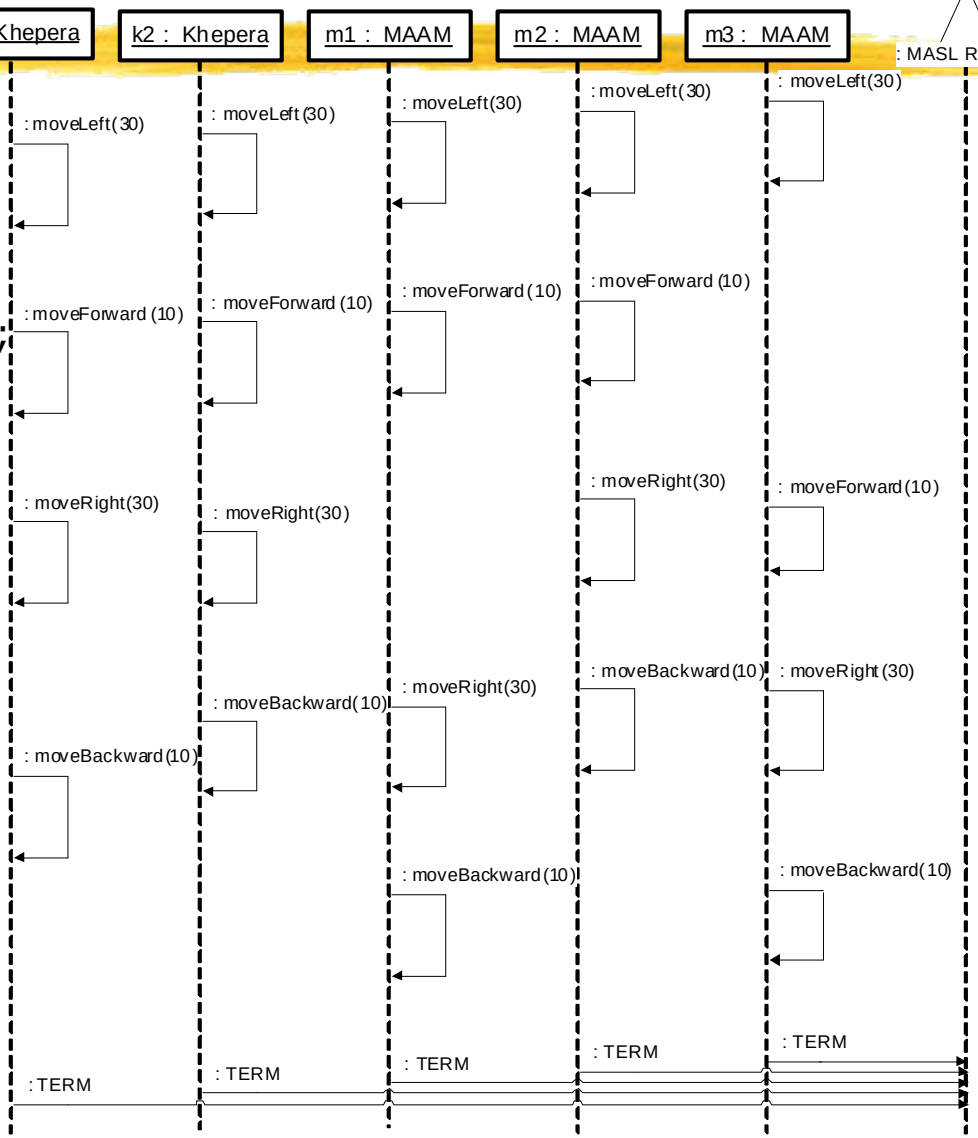
Instanciation

3. C1 : Agents hétérogènes

```
01| import Khepera.xml as Khepera;
02| import MAAM.xml    as MAAM;

03| Khepera k1,k2   = newAgent(Khepera);
04| MAAM    m1,m2,m3 = newAgent(MAAM);

05| asynchonous entry main (true) {
06|    .moveLeft(30);
07|    .moveForward(10);
08|    .moveRight(30);
09|    .moveBackward(10);
10| }
```

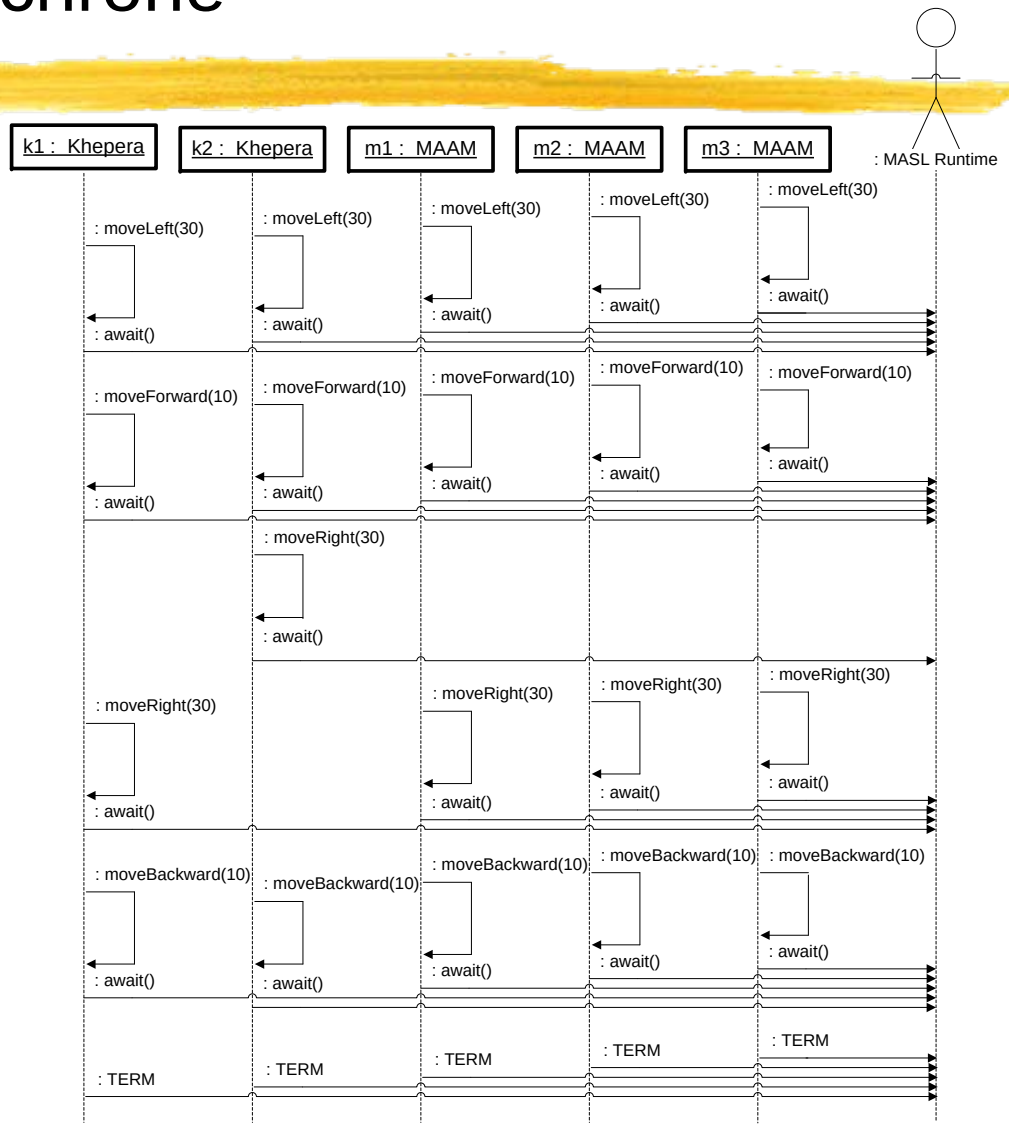
3. C2 : Composition asynchrone

```
01| import Khepera.xml as Khepera;
02| import MAAM.xml as MAAM;

03| Khepera k1,k2 = newAgent(Khepera);
04| MAAM m1,m2,m3 = newAgent(MAAM);

05| asynchronous entry main (true) {
06|    .moveLeft(30);
07|    .moveForward(10);
08|    .moveRight(30);
09|    .moveBackward(10);
10| }
```



3. C2 : Composition synchrone

```
01| import Khepera.xml as Khepera;
02| import MAAM.xml as MAAM;

03| Khepera k1,k2 = newAgent(Khepera);
04| MAAM m1,m2,m3 = newAgent(MAAM);

05| synchronous entry main (true) {
06|    .moveLeft(30);
07|    .moveForward(10);
08|    .moveRight(30);
09|    .moveBackward(10);
10| }
```



3. C2 : Séquence

```
01| import Default.xml as Default;
02| Default a1,a2 = newAgent(Default);

03| scalar entry main (true) {
04|    .log("Hello ");
05|    .log("World !\n");
06| }
```


Hello World !

3. C3 : Communication par variables partagées

```
01| import Default.xml as Default;
02| Default a1,a2,a3 = newAgent(Default);

03| asynchronous entry main (true) {
04|   shared int svar=0;
05|   local  int lvar=0;
06|   lvar++;
07|   svar++;
08|   .log("lvar="+lvar+"\n");
09|   .log("svar="+svar+"\n");
10| }
```

3. C2 : Synchronisation par variables partagées

```
01| import Default.xml as Default;
02| Default a1,a2,a3,a4 = newAgent(Default);
03| asynchronous entry main (true) {
04|   shared int ssynchro = 0;
05|   scalar entry e0 (true) {
06|     .log("do something…\n");
07|     .log("alone\n");
08|     ssynchro=1;
09|   }
10|   while (ssynchro!=1) {}
11|   .log("ready to do something…\n");
12|   .log("all together\n");
13| }
```

Seulement 1 exécution

Un seul dans le bloc

Libération des autres

Attente active

3. C2 : Synchronisation par imbrications de e-blocs

Synchronisation par e-bloc synchrones

```
01| import Default.xml as Default;
02| Default a1,a2,a3,a4 = newAgent(Default);
03| synchronous entry main (true) {
04|   scalar entry e0 (true) {
05|     .log("do something…\n");
06|     .log("alone\n");
07|   }
08|   .log("ready to do something…\n");
09|   .log("all together\n");
10| }
```

3. C3 : Communication par évènements

```
01| import Default.xml as Default;
02| Default a1 = newAgent(Default);
03| asynchronous entry main (true) {
04|   shared event sevent;
05|   emit sevent;
06|   react (sevent) {
07|     .log("Reaction to an event");
08|   }
09| }
```

3. C3 : Communication par évènements

Emission d'un agent et perception par un autre

```
02| Default a1,a2 = newAgent(Default);
03| asynchronous entry main (true) {
04|   shared event sevent;
05|   asynchronous entry e0 (.isOne()) {
06|     emit (sevent);
07|   }
08|   asynchronous entry e1 (.isTwo()) {
09|     do {.log("Ready to react");} loop
10|     react (sevent) {
11|       .log("Here is the reaction to event");
12|     }
13|   }
14| }
```

3. C3 : Communication par évènements

restart et resume

```
02| Default a1,a2 = newAgent(Default);
..
04|    shared event sevent;
..
08|    asynchronous entry e1 (.isTwo()) {
09|      do { .log("Ready to react"); } loop
10|      react (sevent) {
11|         .log("Here is the reaction to event")
12|         resume;
13|      }
14|    }
```

3. C4 : Changement dynamique de rôle

Objectif : Durant son exécution, un agent peut changer de comportements. Un comportement est décrit dans un bloc entry.

Moyens : La politique de retour au programme interrompu reelect vérifie si l'agent doit recommencer le même comportement ou en changer et les instructions lock/release régule l'entrée des agents au niveau d'un point d'entrée.

3. C4 : Changement dynamique de rôle

Changement de comportement avec reelect

```
02| Default a1,a2 = newAgent(Default);
..
04| shared event sevent;
..
07| asynchronous entry e1 (.isTwo()) {
08|   while(true){.log("Bip"); .log("Bop");};
09|   react (sevent){
10|   .log("event");reelect;
11|   }
12|  }
13| }
```

3. C3 : Envoi (a)synchrones de messages

Envoi synchrone de messages : La primitive bloque l'appelant durant l'exécution.

Envoi asynchrone de messages : La primitive ne bloque pas l'appelant durant l'exécution. Il est souhaitable d'offrir un moyen à l'appelant de savoir quand l'exécution de la primitive est terminée.

Un label permet de connaître si tous les éléments d'un ensemble d'appels non bloquant à des primitives ont terminé.

3. C3 : Envoi (a)synchrones de messages

```
01| import Default.xml as Default;
02| Default leader,a1,a2,a3 = newAgent(Default);
03| asynchronous entry main (true) {
04|   local int lleaderID;
05|   local int lID;
06|   Label llabel;
07|   llabel.lleaderID=leader.getfProxyID();
08|   lID=.getProxyID();
09|   llabel.flog("Calls of synchronous methods\n");
10|   while (! llabel.isFinished()){
11|     .log("Waiting 1 second…\n");
12|     .sleep(1000);
13|   }
14|   .log("---\n");
15|   .log("["+lID+"] "] The leader ID is: "+lleaderID+"\n");
16| }
```

Non bloquant

bloquant

Attente de la fin d'exécution non bloquante

3. C5 : Perméabilité

- Les accès internes et externes aux services de l'agent sont conditionnés par des droits :
  - en lecture (r) et en écriture (w) pour les attributs ;
  - en exécution (x) pour les méthodes.
- Ces droits distinguent
  - si le demandeur de l'accès est l'agent lui-même (u=user);
  - un agent s'exécutant dans le même bloc entry (g=group,un collègue) ;
  - un autre agent (o=others).

- État de perméabilité : ensemble de droits applicables à un moment donné et qui couvrent toute l'interface de l'agent.

- Une liste d'états est définie dans le fichier XML de manière statique.

3. C5 : Changement dynamique de perméabilité

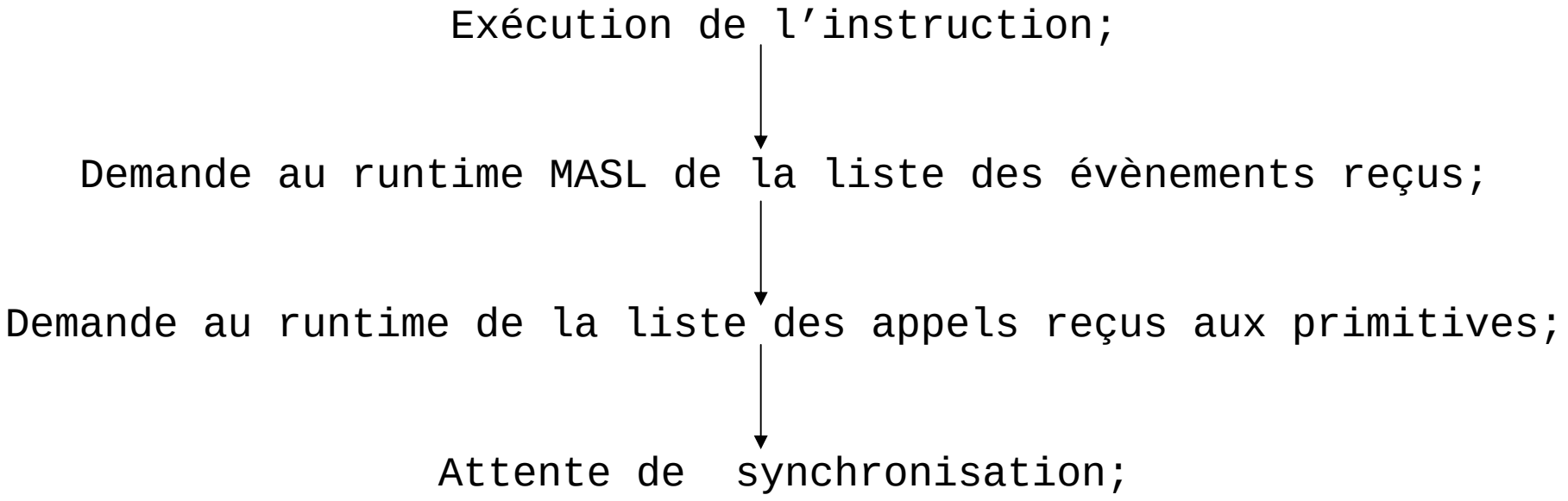
Objectif : Seules certaines primitives sont exécutables pour un agent à un moment donné du fait d'un mode dégradé.

Moyen : Un agent peut basculer dans un état de perméabilité à un autre. Il est le seul à pouvoir changer son état de perméabilité courant.
En fonction de son état courant de perméabilité, les demandes de services seront traitées ou ignorées.
Tout agent peut demander à un autre agent son état de perméabilité courant.

3. C6 : Extensibilité

```
05| asynchonous entry main (true) {
06|    .moveLeft(30);
07|    .moveForward(10);
08|    .moveRight(30);
09|    .moveBackward(10);
10| }
```

3. Principe d’exécution de la boucle MASL

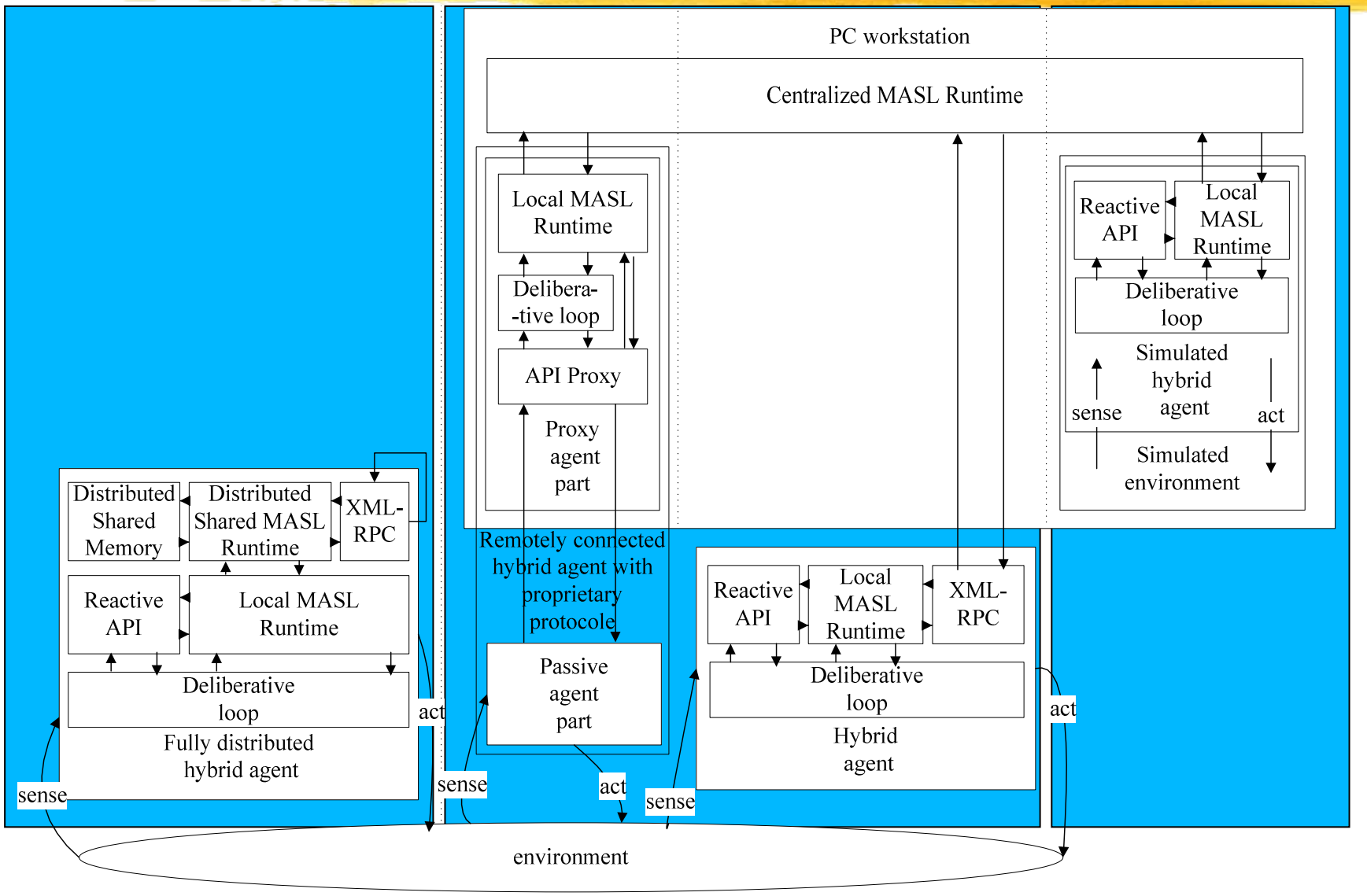


MASL code source ⟶ (Target language, MASL local runtime)

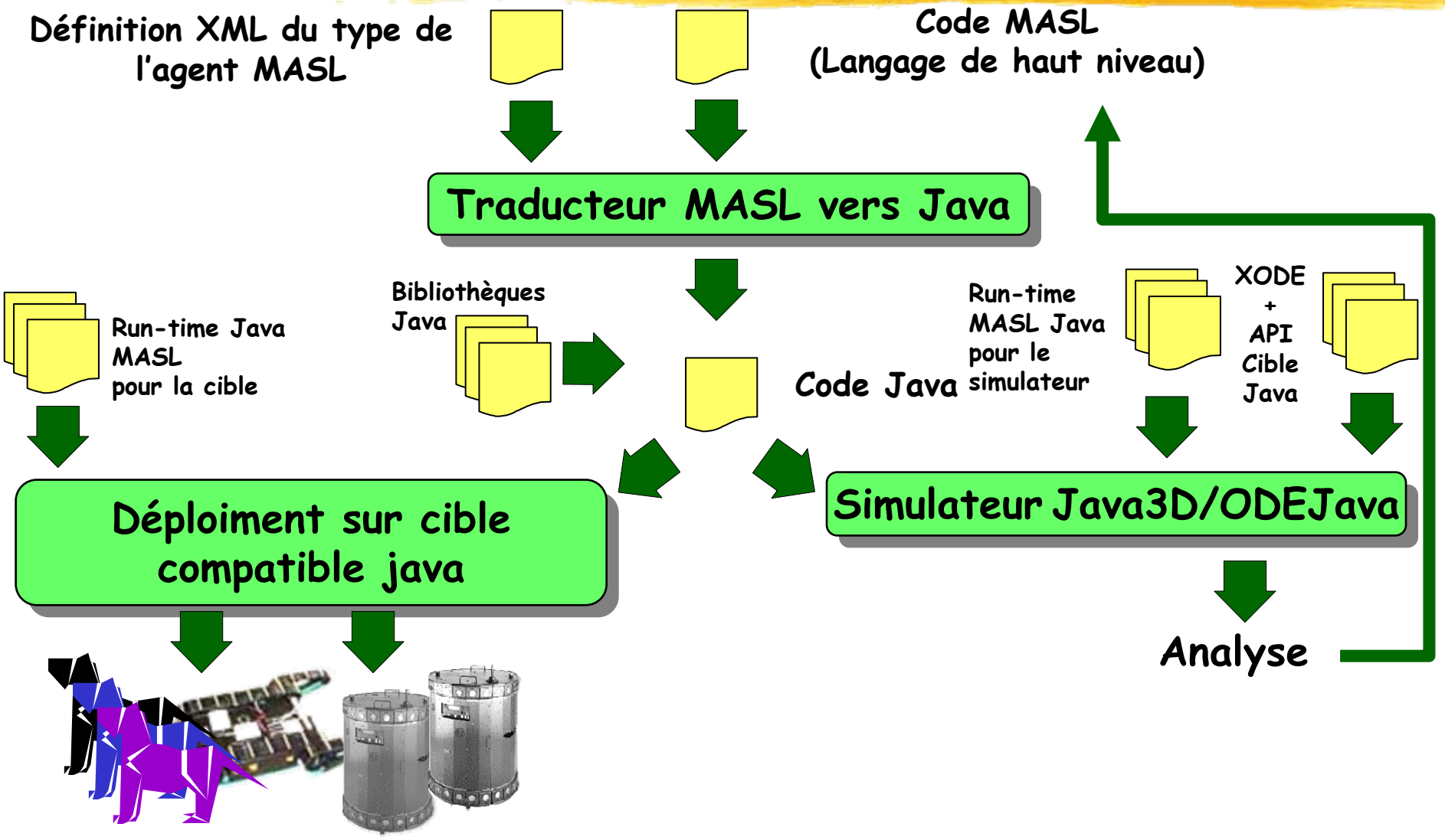
Plan

- Introduction
  - Domaines de la robotique collective
  - Problématique
  - Contraintes du domaine
- Modèles d'exécution d'un composant robotique.
  - API et objets
  - Architectures logicielles de contrôle
- Le modèle de programmation MASL
  - Agents Hétérogènes
  - Synchronisation
  - Communication
  - Changement dynamique de rôle
  - Évolution des capacités
  - Extensibilité
- **Implémentation MASL**
  - **Traduction de MASL vers langage cible**
  - **Scénarios de déploiements**
- Perspectives et conclusions

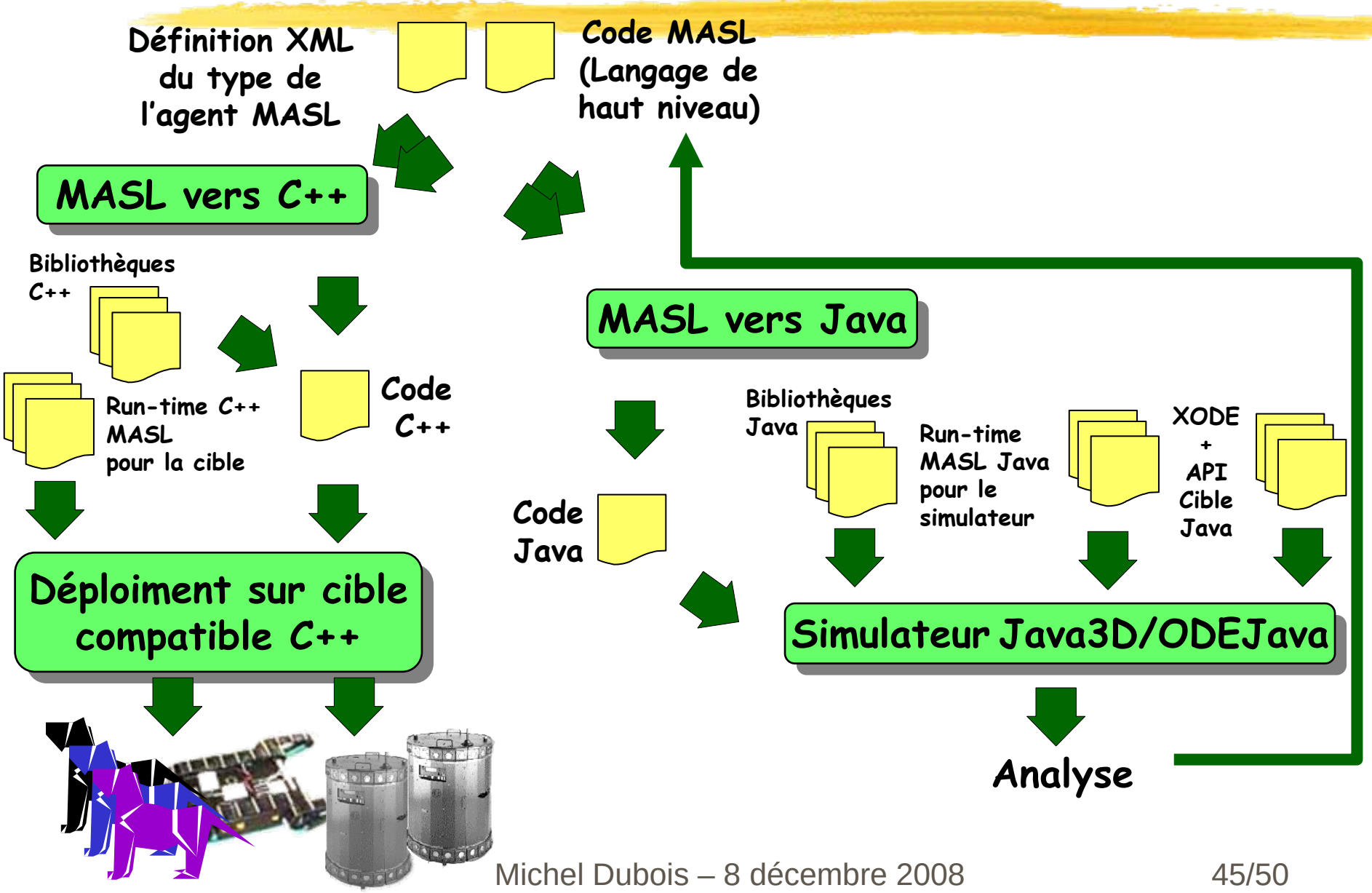
4. Scénarios de déploiements



4. MASL vers une cible compatible Java



4. MASL vers une cible compatible C++



Plan

- Introduction
  - Domaines de la robotique collective
  - Problématique
  - Contraintes du domaine
- Modèles d'exécution d'un composant robotique.
  - API et objets
  - Architectures logicielles de contrôle
- Le modèle de programmation MASL
  - Agents hétérogènes
  - Synchronisation
  - Communication
  - Changement dynamique de rôle
  - Évolution des capacités
  - Extensibilité
- Implémentation MASL
  - Traduction de MASL vers langage cible
  - Scénarios de déploiements
- **Perspectives et conclusions**

5. Conclusion

- Proposition du langage MASL
  - C1 Hétérogénéïté
  - C2 Synchronisation
  - C3 Communication
  - C4 Changements de rôles
  - C5 Évolution des capacités
  - C6 Extensibilité

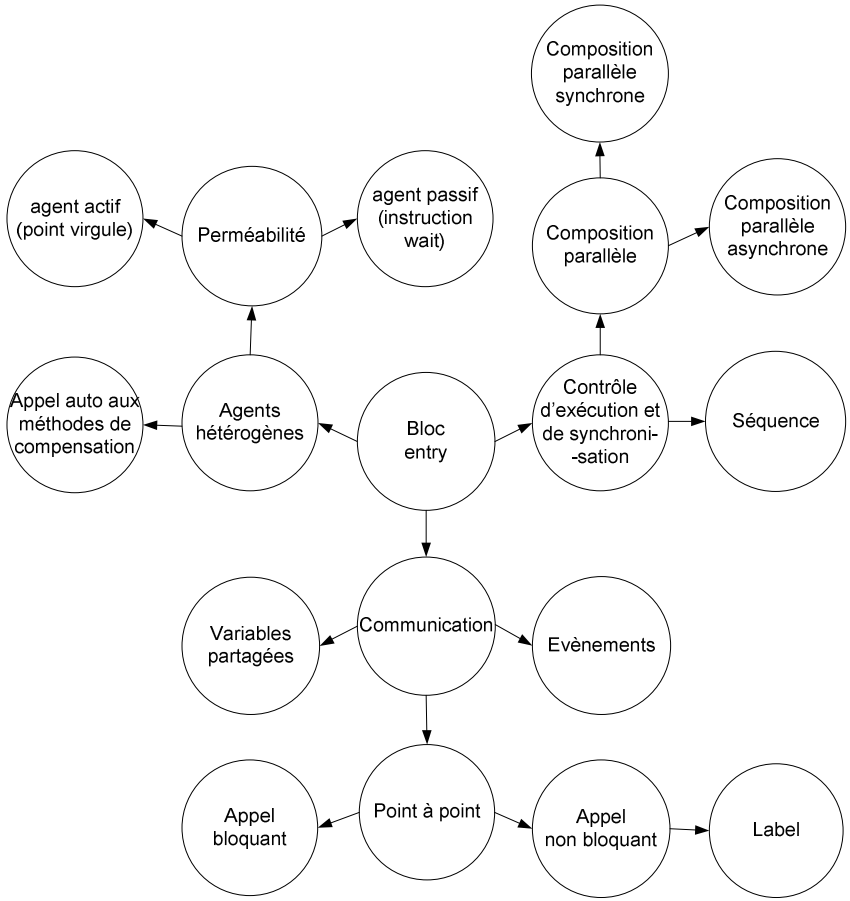
+

Support des API d'objets passifs, actifs, réactifs

Adaptation aux trois types d'architecture : réactive, délibérative, hybride.

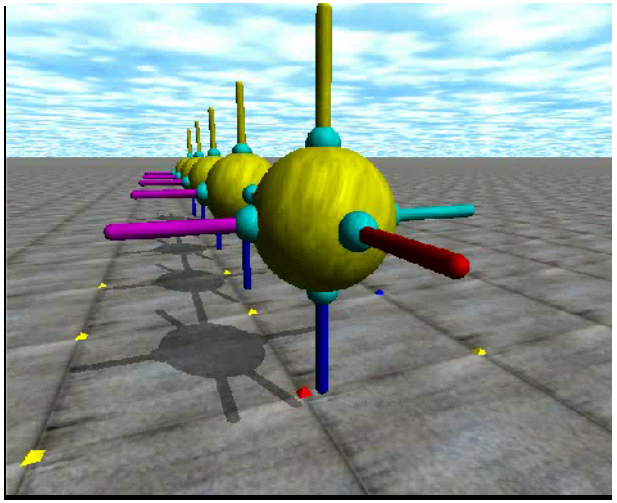
5. Cartes des fonctionnalités d'un bloc entry

- MASL est une unification dans le contexte multi-robots :
  - des modèles d'agents (délibératifs, réactifs, hybrides);
  - des modèles de communication (par mémoire partagée, par évènements, par envoi de messages synchrones, par envoi de messages asynchrones) ;
  - des modèles de contrôle d'exécution et de synchronisation (séquence, exécution parallèle asynchrone, exécution parallèle synchrone).
- Pour une vision offerte au programmeur simplifiée, une seule construction « entry » qui peut être imbriquée, permet de mixer les différentes approches.



5. Perspectives

- Portage de MASL sur un simulateur



- Mise en œuvre d'un traducteur :
  - MASL -> java
  - MASL -> d'autres cibles
- La prise en compte de nouvelle méthodes (primitives) pendant l'exécution (découverte de nouveau service) ?
- Définition de la sémantique formelle

5. Généralisation à la robotique collective

- Projet européen Swarmanoid