AU313 - Application Robotique Dronique

Charles Lesire-Cabaniols (ONERA / DCSD)
charles.lesire@onera.fr

3A-SEM - 2010-2011

Définition

Robot (étym. : *robota* (tchèque), travail, corvée)

un robot est un système mécanique poly-articulé mû par des actionneurs et commandé par un calculateur qui est destiné à effectuer une grande variété de tâches.

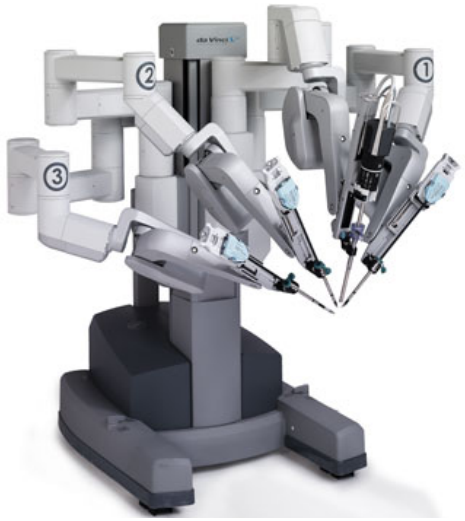
Origine

- *Robot* utilisé pour la première fois en 1921 par Karel Capek dans sa pièce *Rossum's Universal Robots* ;
- *Robotique* employé pour la première fois par Isaak Asimov en 1941
  - *I, robot*, 1950
  - *Foundation*, 1951

Isaak Asimov (1965)

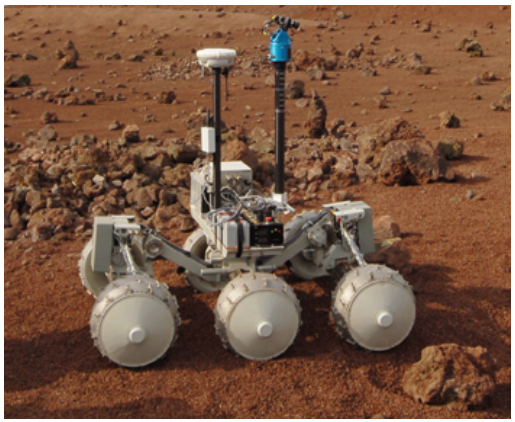
Robots manipulateurs

- Robots industriels : chaînes de montage, manipulation de produits chimiques, ...
- Robots d'assistance médicale



Robots d'exploration

- Exploration planétaire
- Exploration d'épaves ou de décombres
- Déminage, zones radioactives, ...



Robots de service

- Transport de marchandises
- Robots ménagers
- Aide aux personnes

Robots ludiques

Boucle de décision

Un robot est capable d'extraire de l'information à partir de son environnement et d'utiliser ses connaissances pour décider comment agir. Un robot est équipé de capteurs et d'effecteurs.

Capteurs / Effecteurs

Capteurs :

- Caméra
- Sonar
- Détecteur de lumière
- Boussole
- GPS
- Détecteur de chaleur
- ...

Effecteurs :

- Roues
- Bras
- Jambes
- Pinces
- ...

Tâches

- Les robots ont un ensemble de tâches à réaliser ;
- Leur exécution consomme du temps et des ressources ;
- Des contraintes (temporelles, spatiales, ...) peuvent leur être associées.

Programmation

L'intelligence artificielle d'un robot se résume à un ensemble de programmes écrits sur un ordinateur :

- les programmes sont écrits dans un langage de programmation ;
- ils s'exécutent grâce au contrôleur du robot ;
- ils prennent en entrée les informations obtenues des capteurs et en sortie envoient des ordres aux effecteurs.

Programmes

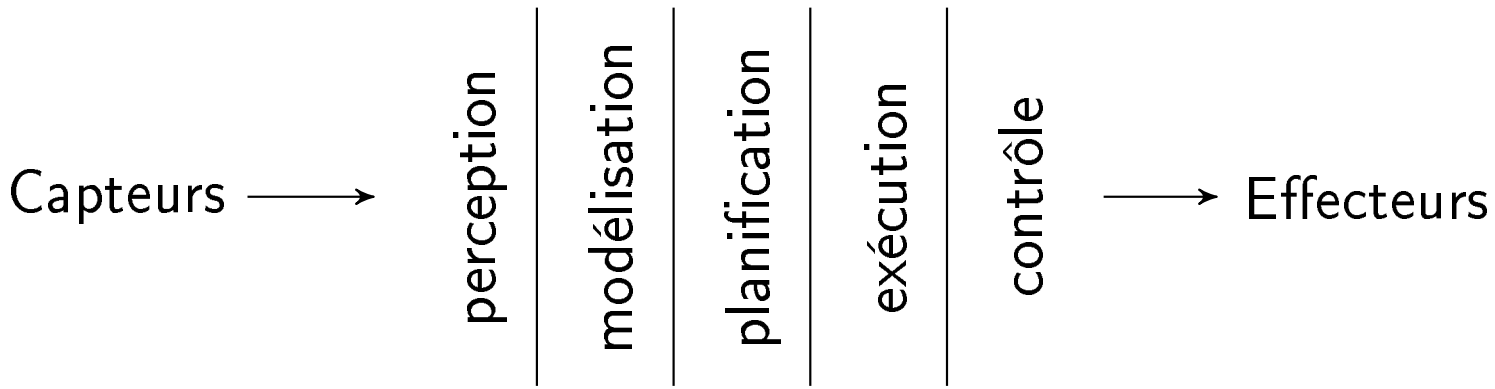
L'intelligence artificielle d'un robot permet par exemple :

- l'analyse d'images ;
- sa localisation et sa navigation ;
- la gestion des interactions (communications, interfaces) ;
- la planification et la prise de décision ;
- le contrôle de l'exécution des tâches.

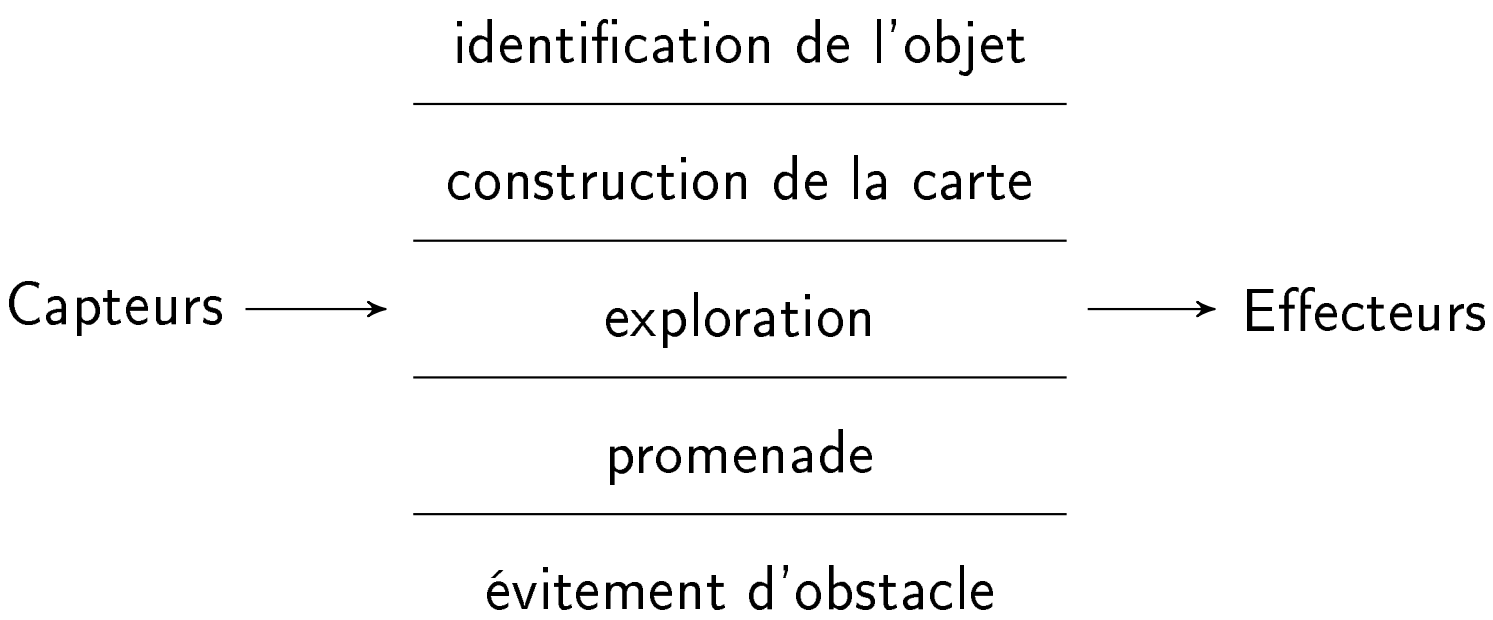
Approche sub-symbolique, ascendante ou *bottom-up*

- 1986, Rodney Brooks : *"Elephants don't play chess"*
  - L'essentiel pour un robot est d'abord de survivre
  - Des composants réactifs plutôt que cognitifs
  - La complexité peut émerger de la somme de comportements simples
- Vision modeste mais réaliste
  - Objectifs modestes : labyrinthes, autonomie énergétique, ...
  - Etude de la boucle perception-action
  - La réactivité et l'adaptation deviennent des enjeux cruciaux

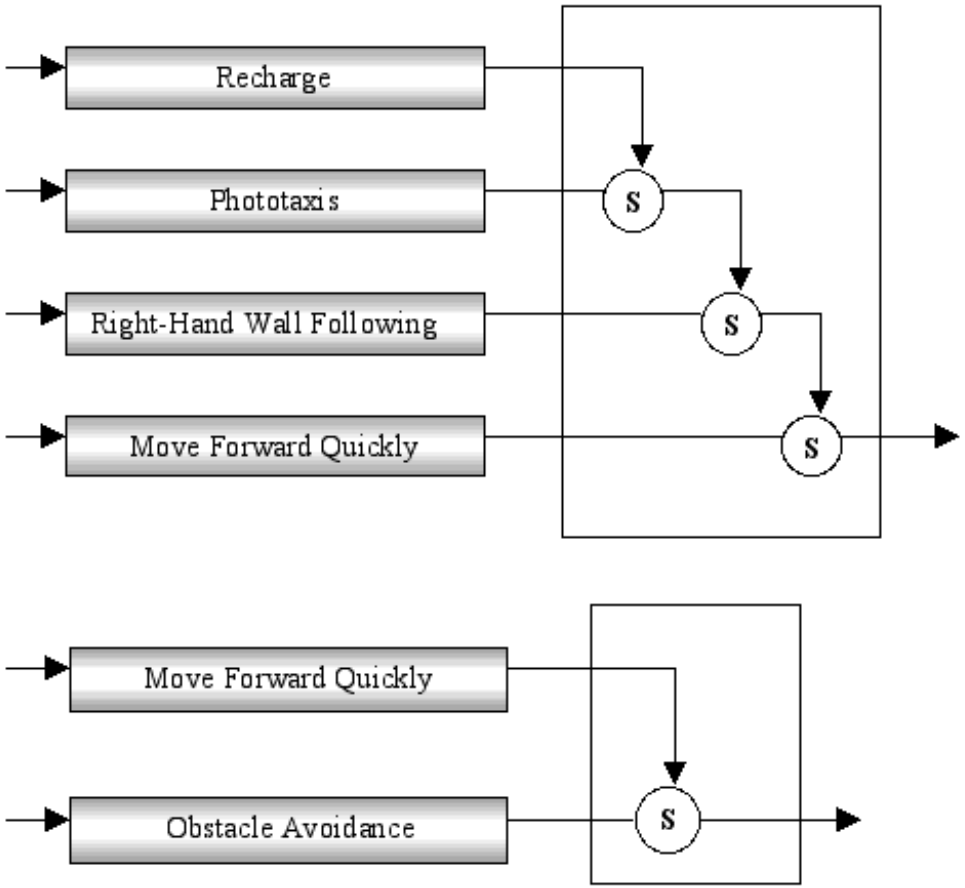
Approche traditionnelle



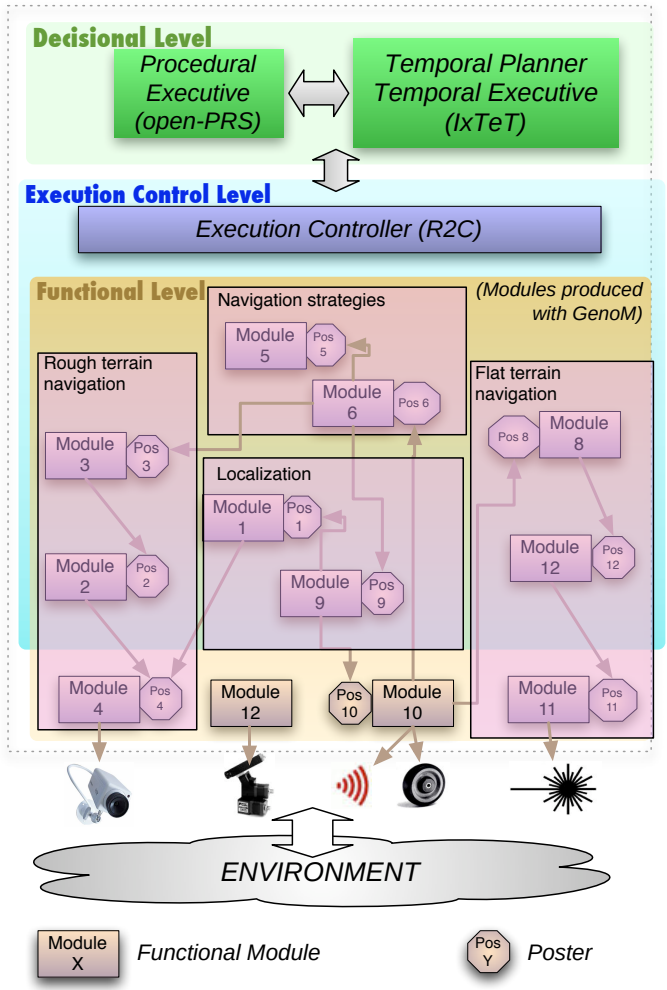
Approche comportementale



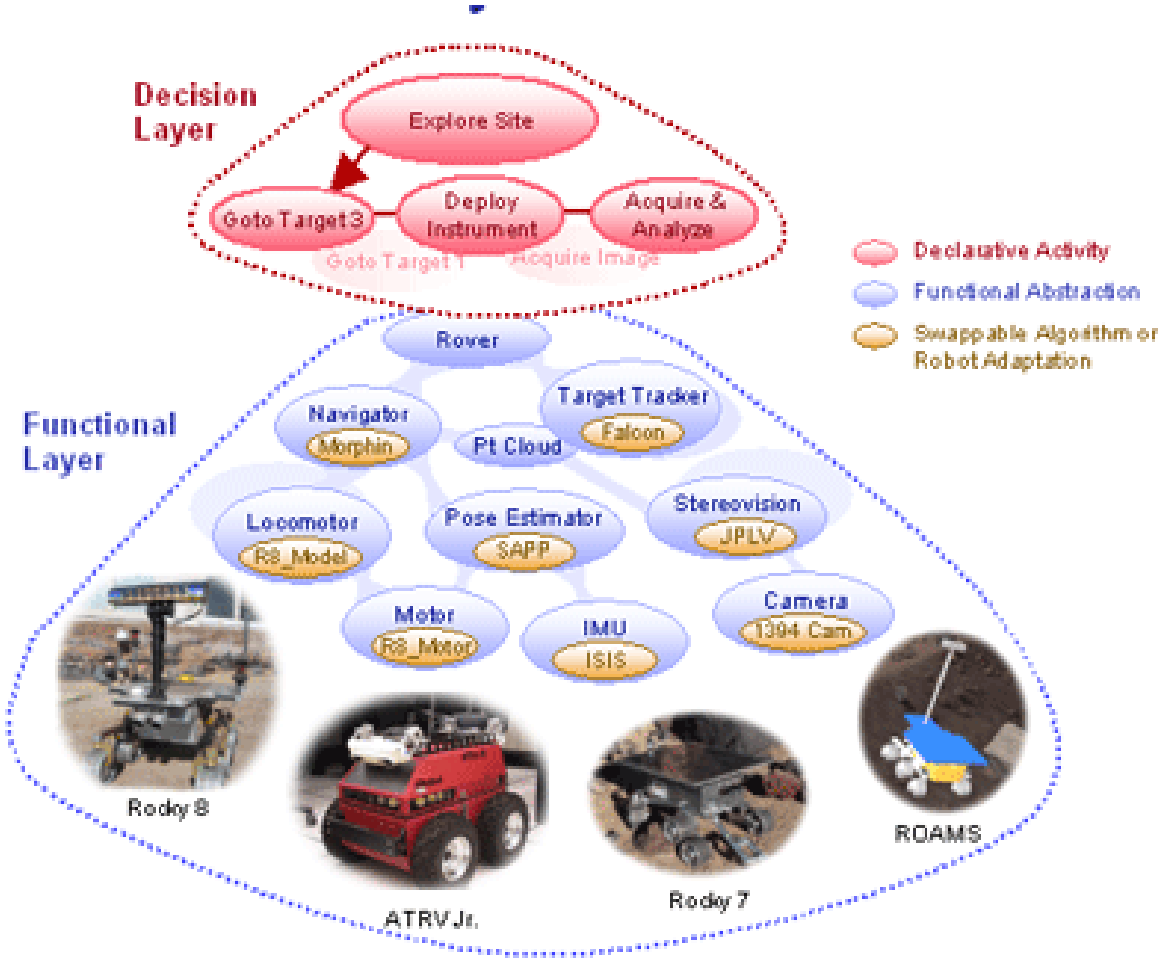
Approche comportementale



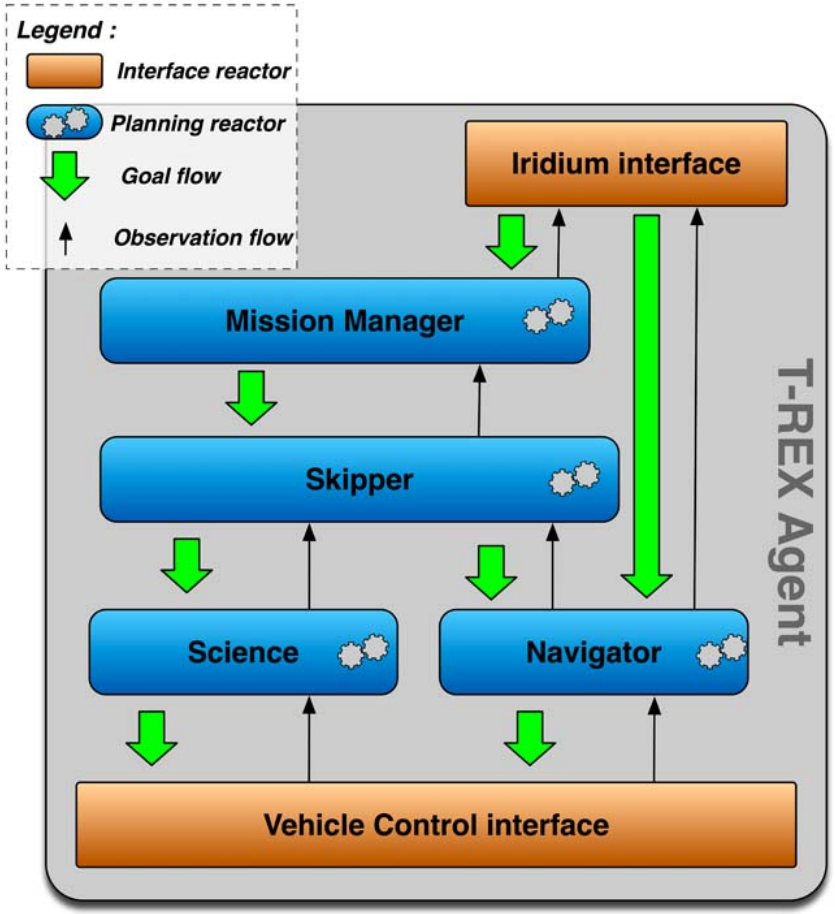
Architecture LAAS



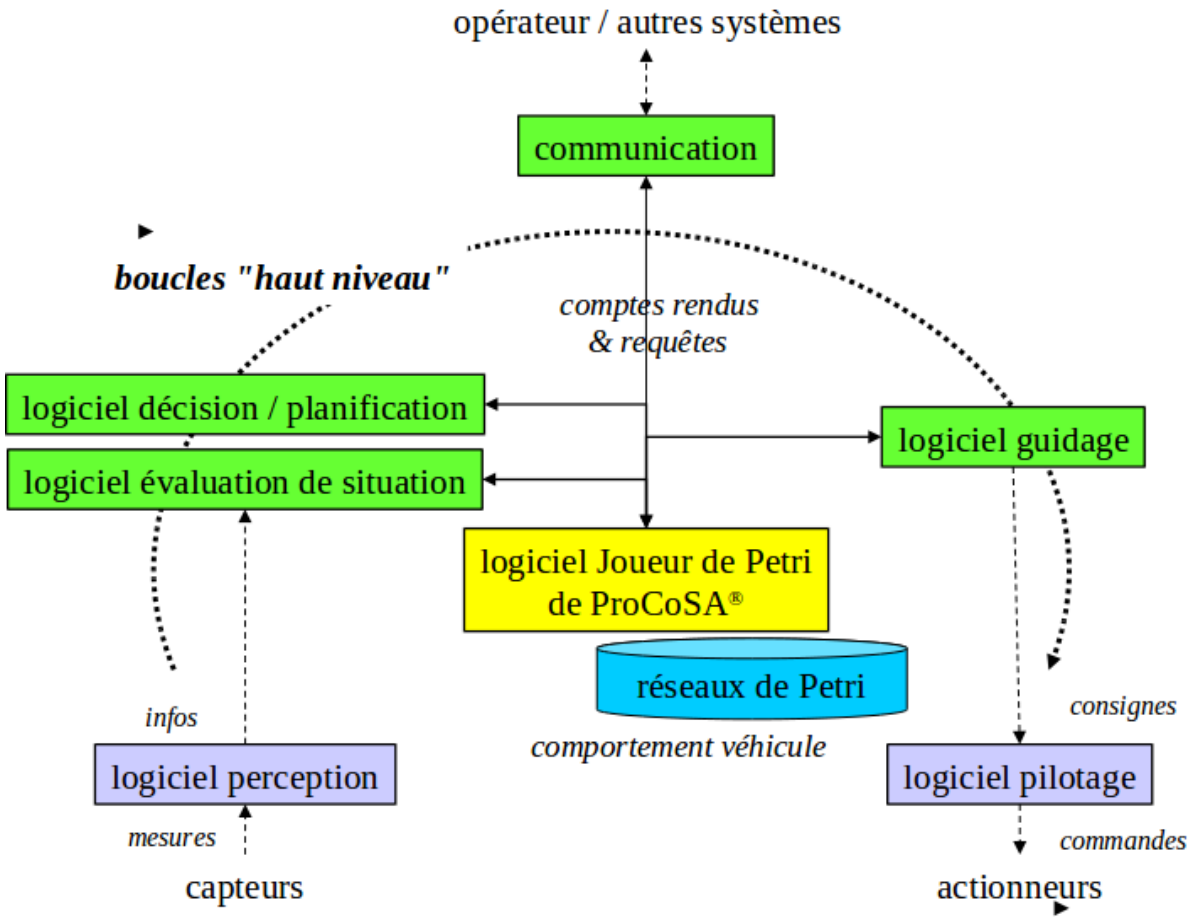
Architecture Claraty (NASA)



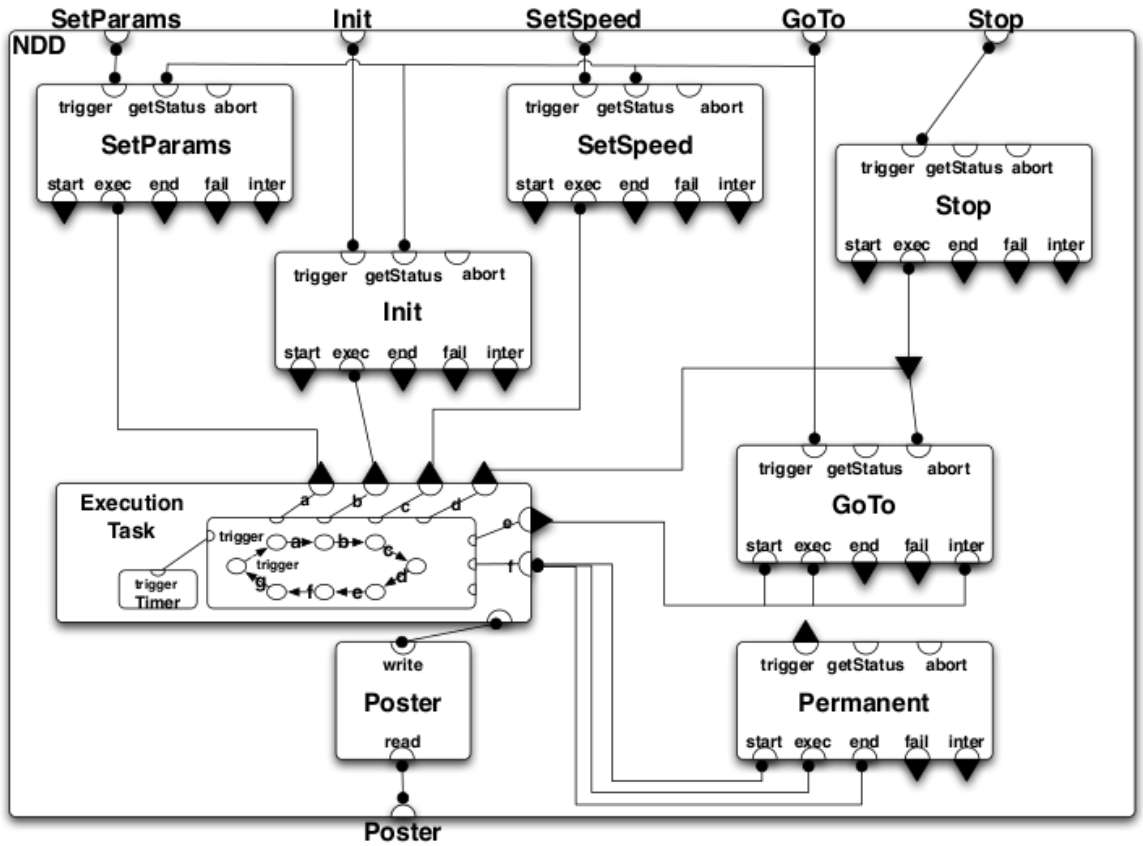
Architecture T-REx (MBARI)



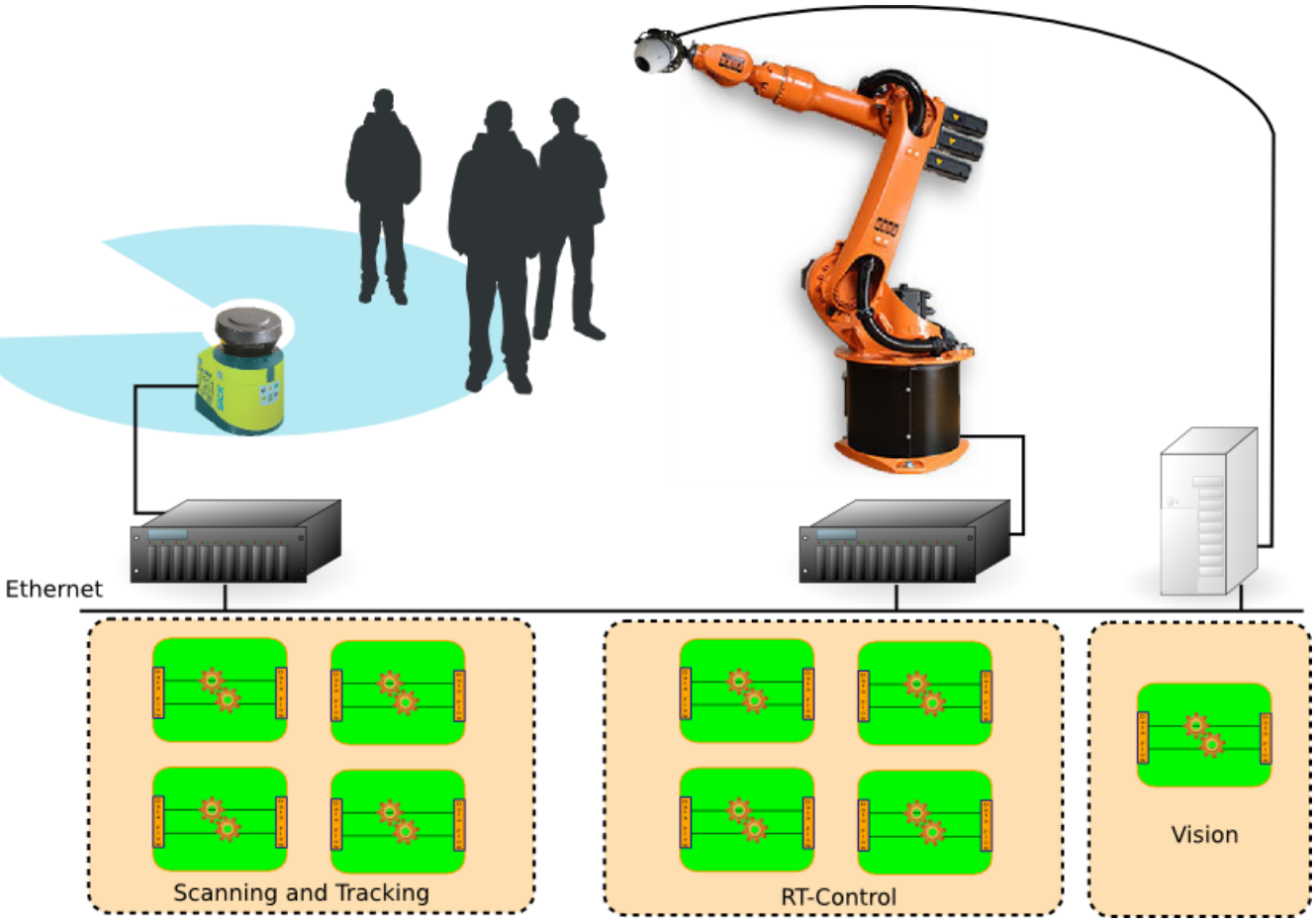
Architecture ProCoSA (Onera)



Architecture BIP (LAAS/VeriMAG)

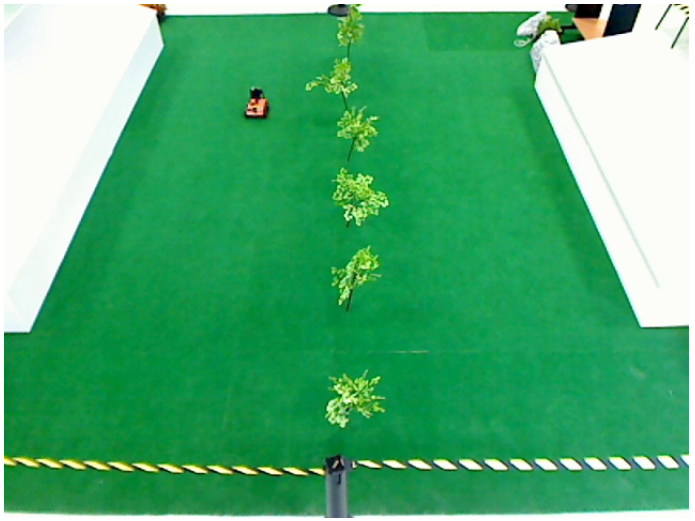


Architecture Orocos (Univ. Leuven, Onera, NASA, ...)



Sujet du BE

- Mission d'exploration de zones, et d'extinction d'incendies
  - navigation
  - exploration
  - analyse d'images
  - prise de décision, planification



Sujet du BE

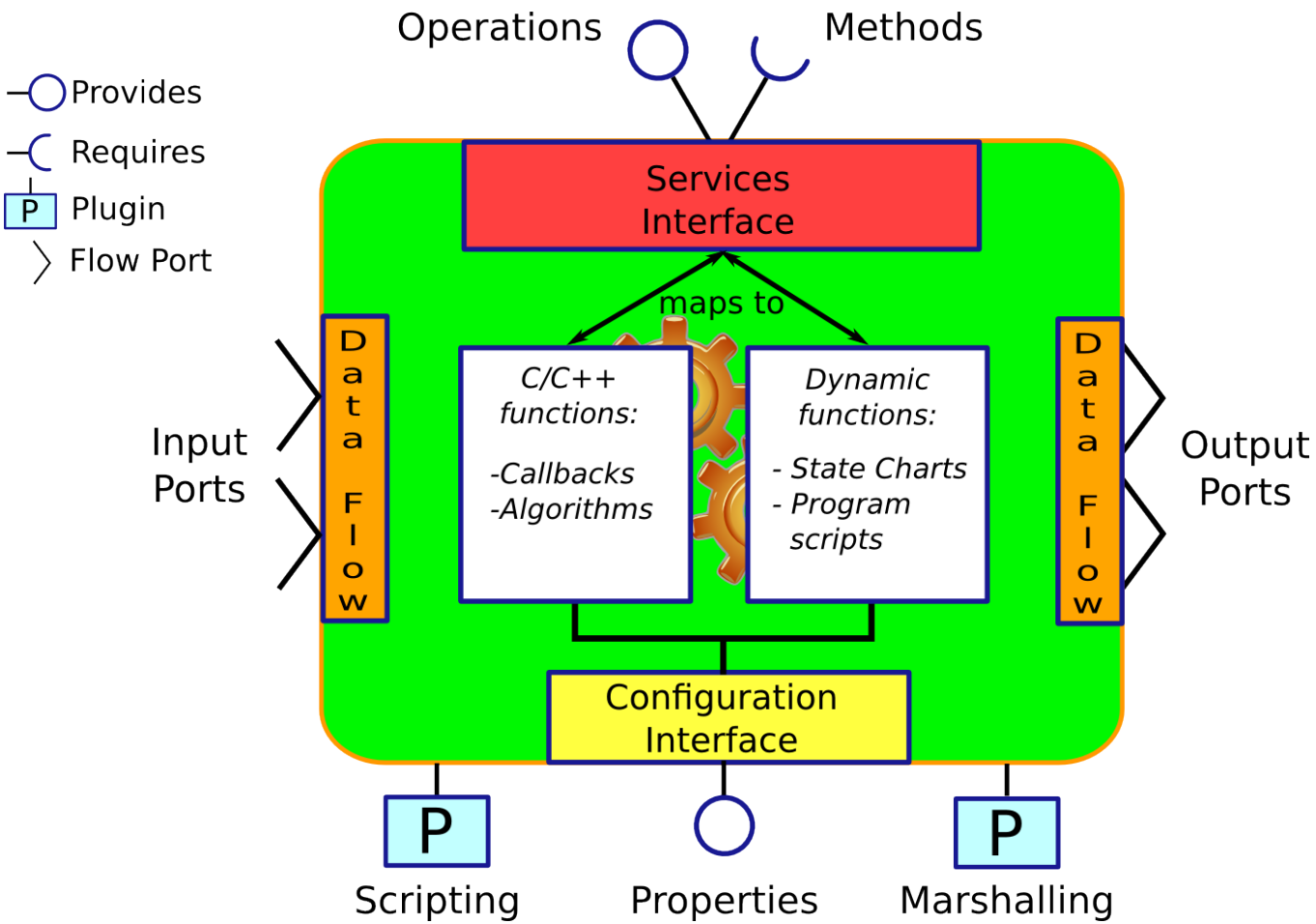
- Développement de composants robotiques
  - Analyse d'images simplifiée
  - Sous l'environnement Orocos
- Déploiement d'une architecture robotique
  - Navigation, Prise d'images, Analyse d'images
- Supervision de mission

Orocos

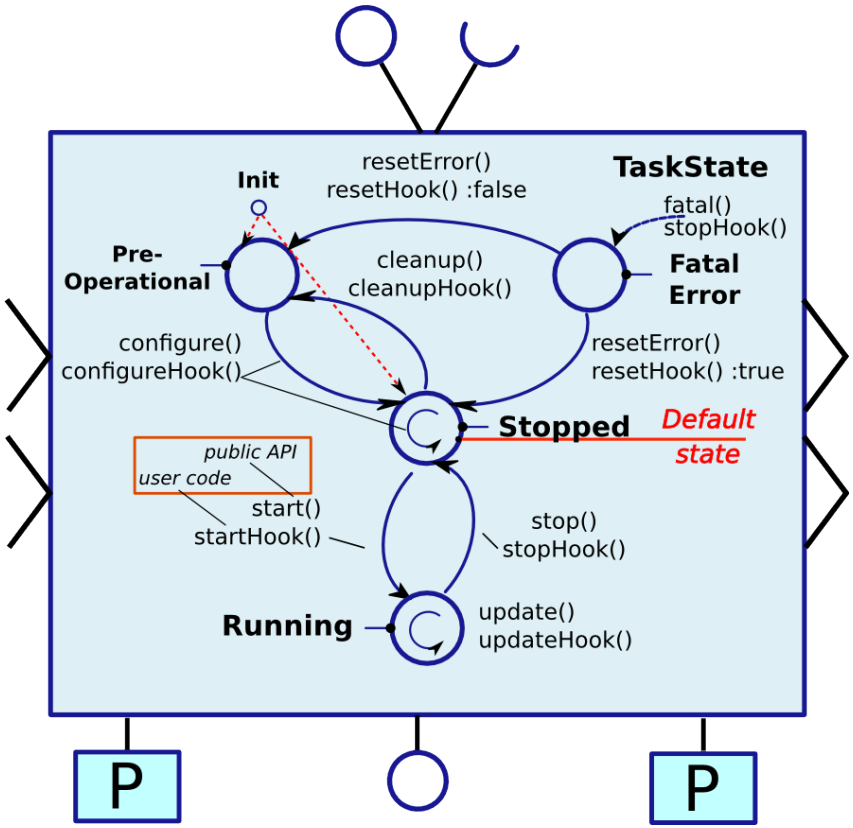
Une librairie en C++ qui permet :

- de créer des composants exécutables, distribuables ;
- de spécifier des communications temps-réel et "thread-safe" entre composants ;
- de charger et d'exécuter des scripts (programmes / machines à état) en temps-réel ;
- d'accéder aux différents attributs des composants et des communications.

Interface d'un composant



Etats d'un composant



Code

```
class Mapping : public RTT::TaskContext, MappingAlg {
    MatchingParameters pMatch;
    RTT::Property<std::string> CalibrationFile;
    RTT::Property<std::vector<double> > MapFrame;
    RTT::ReadDataPort<image_t> image_port;
    RTT::ReadDataPort<Vector> position_port;
    RTT::ReadDataPort<Vector> attitude_port;
    RTT::WriteDataPort<std::vector<int> > obstacles;
    RTT::WriteDataPort<image_t> map_port;
    RTT::Command<bool(void)> build_command;
```

Code

```
Mapping(const std::string& name) :
    RTT::TaskContext(name, PreOperational),
    CalibrationFile("CalibrationFile", "/comment/", ""),
    MapFrame("MapFrame", "/comment/", vector<double>(5,0)),
    position_port("Position"),
    attitude_port("Attitude"),
    image_port("Image"),
    map_port("MapImage"),
    obstacles("MapCounter"),
    build_command("build", &Mapping::build, this)
{
    ports()->addEventPort(&image_port);
    ports()->addPort(&position_port);
    ports()->addPort(&attitude_port);
    ports()->addPort(&obstacles);
    ports()->addPort(&map_port);
    properties()->addProperty(&pMatch);
    properties()->addProperty(&CalibrationFile);
    properties()->addProperty(&MapFrame);
    commands()->addCommand(&build_command, "Build map.");
};
```

Code

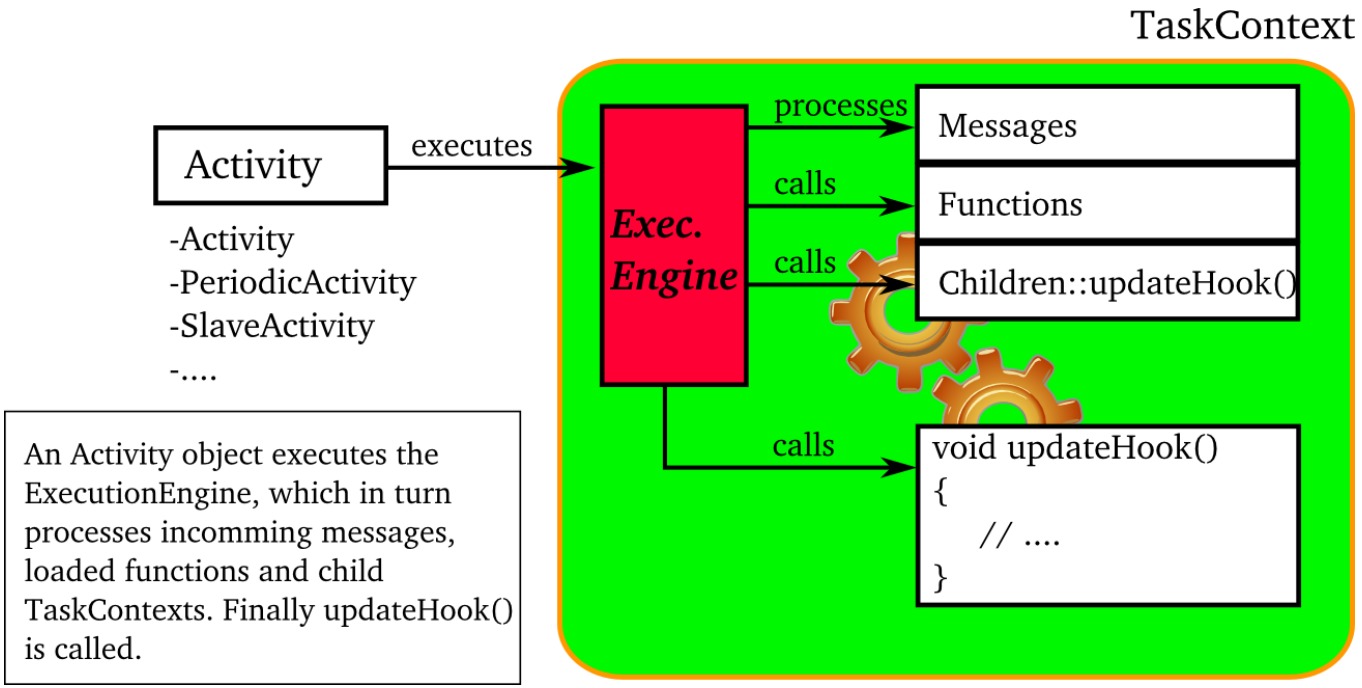
```
virtual bool startHook() {
    // EVA properties
    pMatch.fill(pObsDetect);
    // Init EVA parameters
    initParameters(calibration);
    if (log().getLogLevel() >= Logger::Info)
        dtim_Camera_showIntrinsicParam(&pIntrin);
    // Init Map
    eva_cartoInitialisation(origin_north, ..., &map);
    return true;
};

virtual void stopHook() {
    if (!flag1st) freeEVA();
    flag1st = true;
};
```

Code

```
virtual void updateHook() {
    img = image_port.Get();
    if (!img) {
        log(Error) << "Input image is empty!" << endl;
        return;
    }
    Vector v = position_port.Get();
    Vector w = attitude_port.Get();
    setExtrinsicParameters(v, w);
    if (flag1st) {
        flag1st = false;
        createEVA();
        return;
    }
    // Detection
    double pct = detect();
    log() << pct << " \% of pixels are obstacles" << endl;
    if (log().getLogLevel() >= Logger::Debug)
        eva_logEva_print2screen(&perfo);
    // Mapping
    mapping();
}
```

Execution



An Activity object executes the ExecutionEngine, which in turn processes incomming messages, loaded functions and child TaskContexts. Finally updateHook() is called.

Interconnexion des composants

Flot de données

- Connexion entre deux ports,
  - Lock-free
- Politique de la connexion :
  - donnée unique partagée (DATA) ou bufferisée (BUFFER)
  - taille du buffer
  - valeur initiale
- Chaque composant peut :
  - Lire ou écrire dans son port,
  - Connaitre l'état de la connexion,
  - Savoir si la donnée reçue est nouvelle.

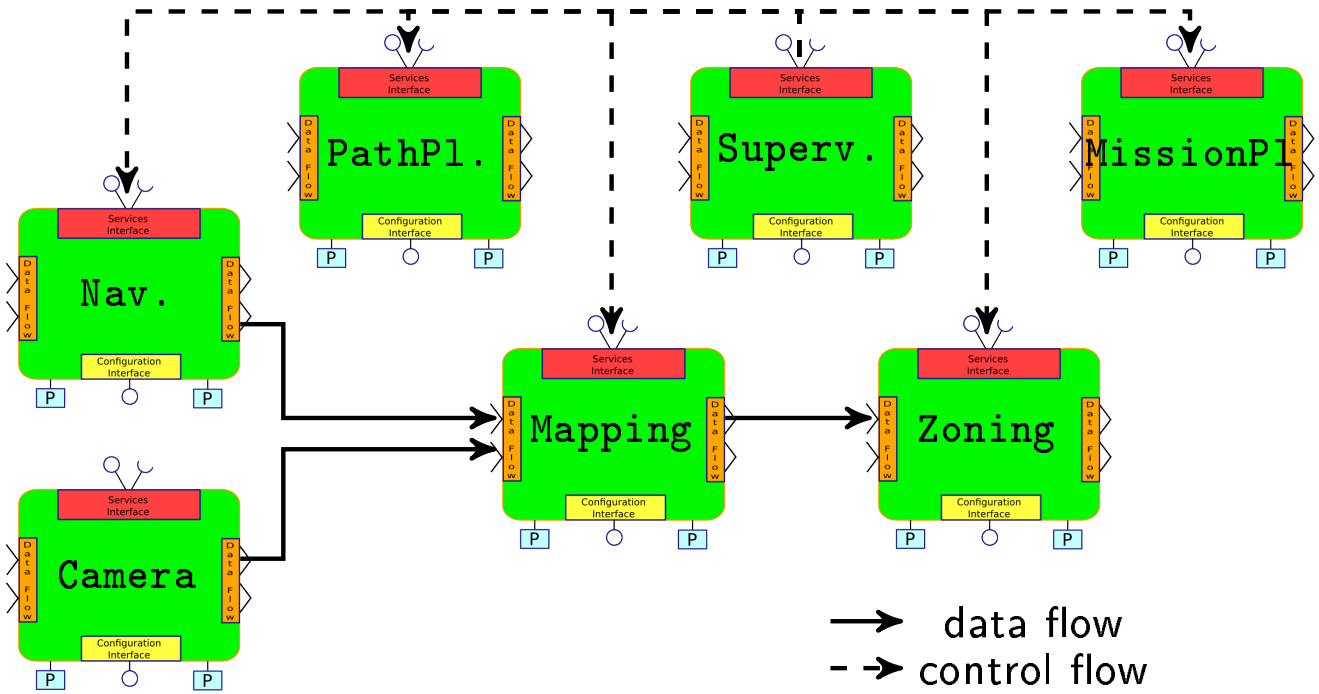
Interconnexion des composants

Flot de services

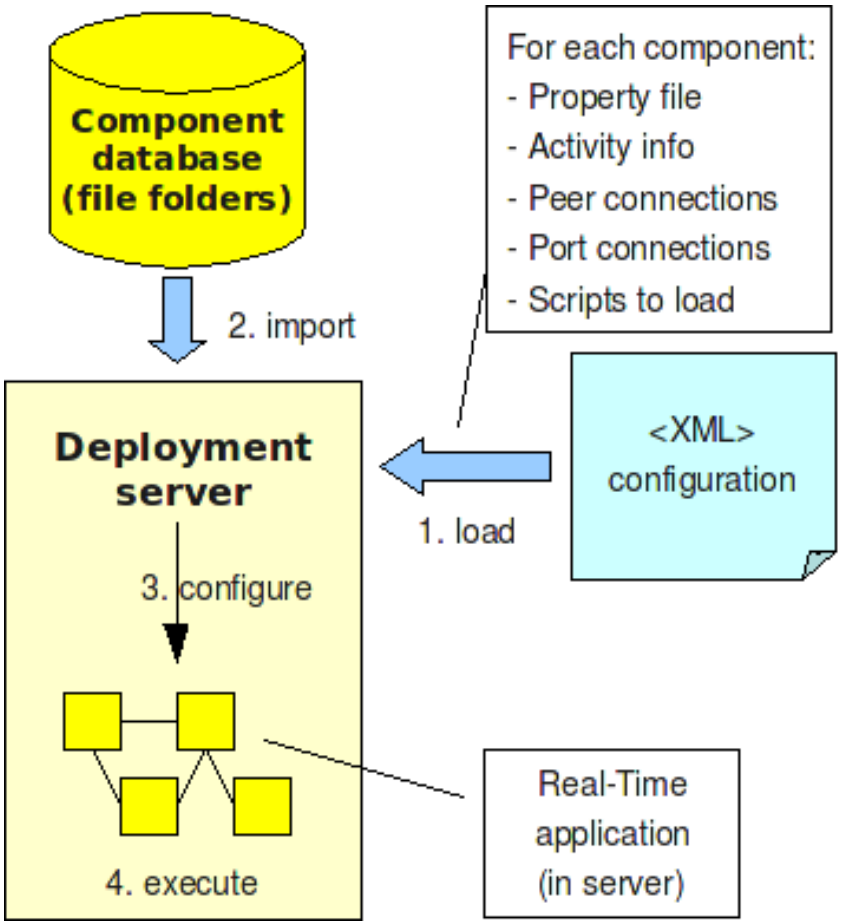
- Connexion des opérations (services fournis) d'un composant aux méthodes (services requis) d'un autre,
- Utilise le nom du service et la signature des fonctions,
- Le code associé (la fonction C++) est exécuté :
  - Dans la tâche du fournisseur (le fournisseur doit l'autoriser),
  - Dans la tâche du demandeur (le demandeur doit l'autoriser),
  - Dans une tâche de fond de la RTT (si personne ne veut l'exécuter).
- Le demandeur peut choisir d'attendre le retour de la fonction (bloquant) ou non.

Déploiement

Architecture



OCL : :DeploymentComponent



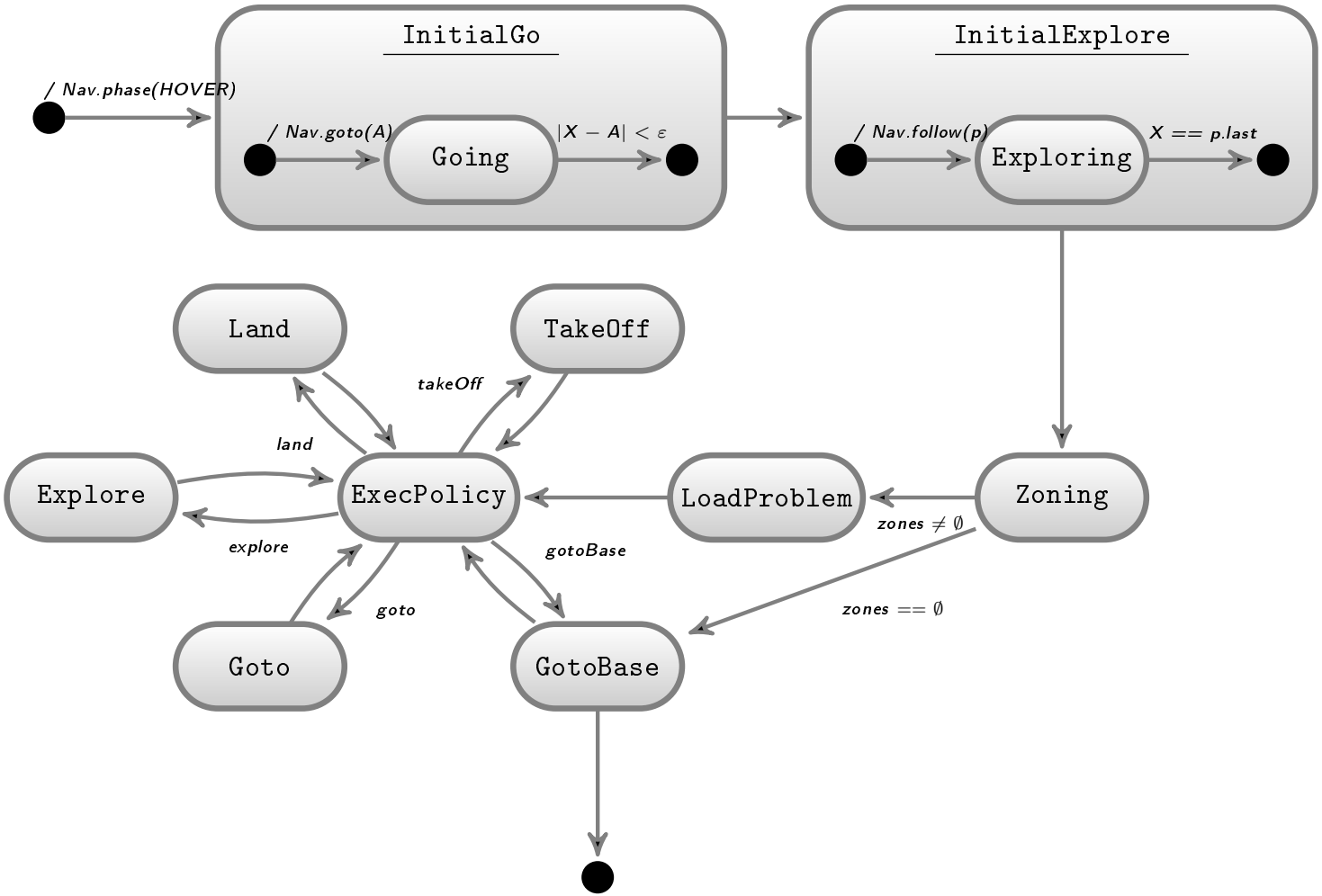
Fichier XML

```
<properties>
 <!-- Imports -->
 <simple name="Import" type="string">
  <value>libressac-mapping</value>
 </simple>
 ...
 <!-- Components -->
 <struct name="Mapping" type="Ressac::Mapping">
  <struct name="Activity" type="Activity">
   <simple name="Period" type="double"><value>0</value></simple>
   <simple name="Priority" type="short"><value>0</value></simple>
   <simple name="Scheduler" type="string">
    <value>ORO_SCHED_OTHER</value></simple>
  </struct>
  <struct name="Properties" type="PropertyBag">
   <struct name="Matching" type="PropertyBag">
    <simple name="rSearch" type="short"><value>100</value></simple>
   </struct>
  </struct>
  <simple name="AutoConf" type="boolean"><value>1</value></simple>
  <simple name="AutoStart" type="boolean"><value>0</value></simple>
 </struct>
```

Fichier XML

```
<struct name="Camera" type="RoboTIS::Vision::FirewireCamera"></struct>
<struct name="Zoning" type="Ressac::Zoning"></struct>
<struct name="Planning" type="Planning::PlannerHMDP"></struct>
<struct name="Navigation" type="Ressac::NavigationOutSerial"></struct>
<struct name="Ressac">
 <simple name="StateMachineScript" type="string">
  <value>search_and_rescue.osd</value>
 </simple>
</struct>
```

Machine à états



Fichier OSD

```
StateMachine SearchAndRescue {
 param zone z
 var zones zone_list

 initial state Init {
  transition select InitialGo
 }

 state InitialGo {
  entry {
   do Navigation.goto(z.center)
  }
  transition select InitialExplore
 }

 state Zoning {
  entry {
   do Zoning.extract()
   set zone_list = Zoning.zone_list.Get()
  }
  transition if zone_list.size != 0 then select LoadProblem
  transition if zone_list.size == 0 then select GotoBase
 }
 ...
```