# Agents mobiles et Programmation Orientée Agent

Master 1 Androide – UE FoSyMa

Avril 2019

Cédric Herpson
cedric.herpson@lip6.fr

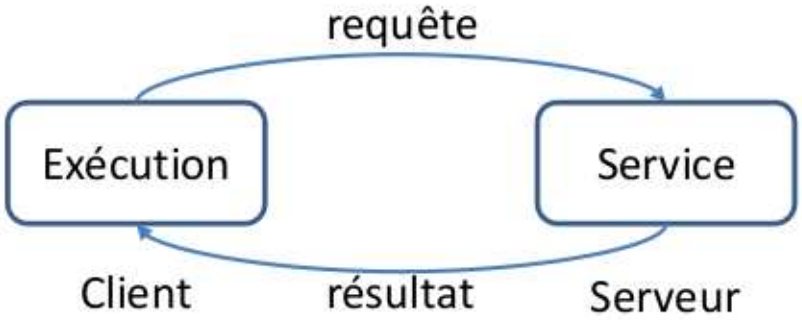
# Plan

- Agents mobiles VS Clients/Serveurs
  - Modèle d'exécution répartie
  - Migration
- Programmation Orientée Agent (AOP)
  - AgentSpeak, Jason, et 3APL
  - GOAL
  - CLAIM/S-CLAIM
- Bilan

# Agents mobiles VS Client-serveur

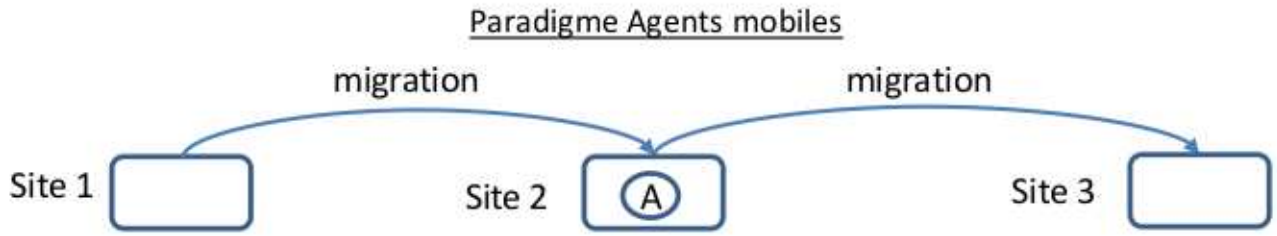
Paradigme Client-serveur

Extension de la notion de service des environnements centralisé




- Interaction synchrone
  - Le client est bloqué tant que le serveur n'a pas répondu
- Expertise et données coté serveur
- Pannes courantes :
  - Perte de la requête, de la réponse
  - Panne du serveur, du client

# Agents mobiles VS Client-serveur




- Interaction Asynchrone
  - Pas de problème de latence
  - Déconnexion possible des sites distants ( localisation explicite)
- Peu de consommation de bande passante
  - Adapté aux gros volumes/flux de données
- Réactivité et Adaptativité
  - Découplage de l'expertise et des données, gestion des ressources plus aisée (par d'accès distant)
  - Tolérance aux pannes par reconfiguration
  - Robustesse (indépendance par rapport au système hôte et support de l'hétérogénéité matérielle)
  - Gestion des imprévus
  - Fonctionnalités évoluant dynamiquement avec la mobilité des agents
- Problèmes
  - Mobilité du code (Capture d'état, Hétérogénéité des environnements soft/hard)
  - Identification
  - Sécurité (des agents et des hôtes)
  - Notion de voisinage (jeux)

(A)

# Paradigme Agents mobiles

Infrastructure d'accueil des agents

- Support pour l'exécution
- Communication agent/hôte
- Migration
  - Admission (Gatekeeper)
  - Réception
- Désignation/localisation
- Contrôle des ressources et sécurité
  - Coté agent : Confidentialité et non modification du code
  - Coté serveur : Vérification, isolation
- Gestion des erreurs
  - Isolation, destruction, modifications
  - Migration
  - Notification

# Paradigme Agents mobiles

## Migration

De quoi parle t'on ?

Code, données, état d'exécution

Problème : L'état est partiellement dépendant de la plateforme sur lequel le code s'exécute

- pile d'exécution,
- registre d'adresse,
- ..

On distingue deux types de ressources : Transférables et non transférables

- Mobilité/Migration forte : Code + données/paramètres + unité d'exécution
- Mobilité/Migration faible : Code + données/paramètres

c

**Forte**

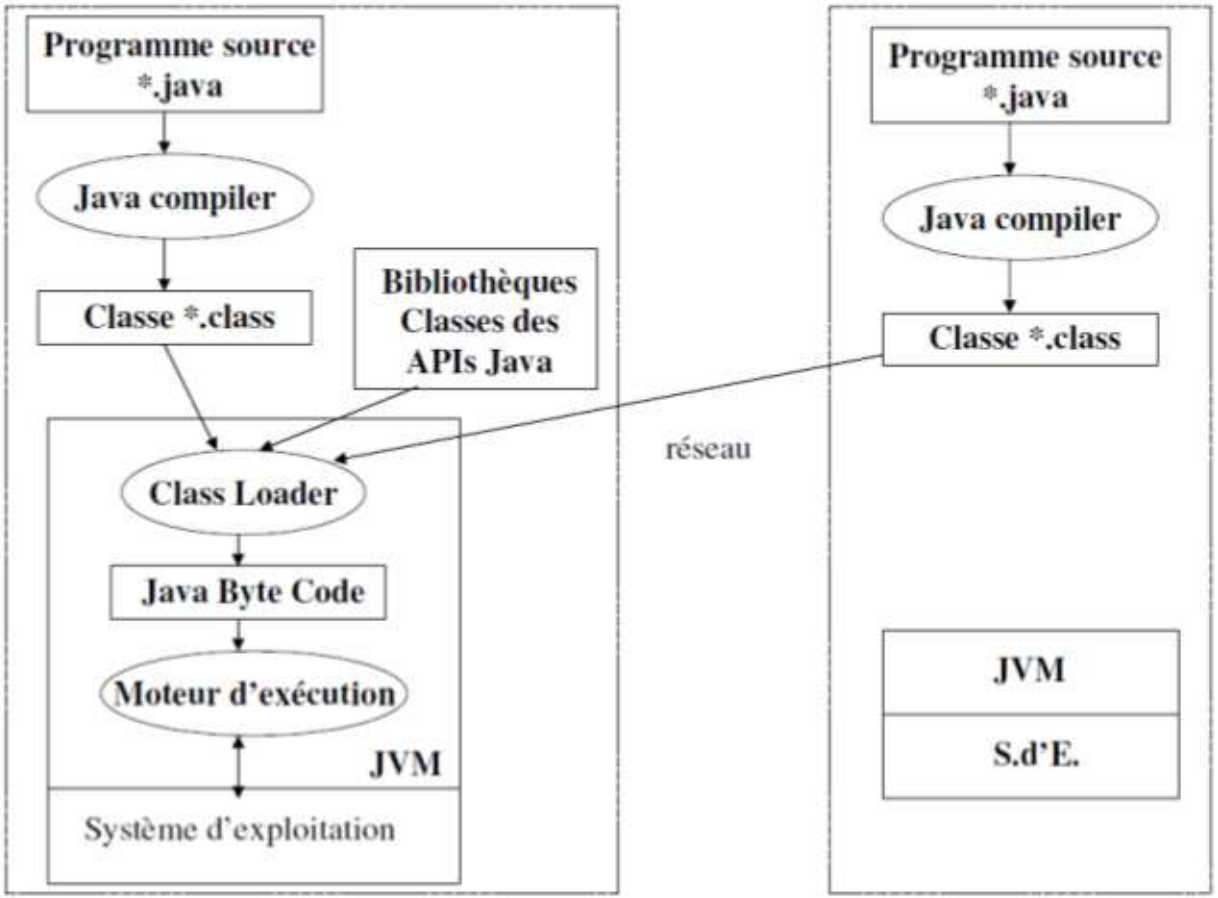
-> Instruction i
Migration
->Instruction i+1

**Faible**

-> Sauvegarde partielle du contexte d'exécution
-> Reprise de l'exécution à une méthode spécifiée

JVM (jade) : Migration faible

# Migration faible avec Java



# Migration faible avec Jade

- doMove()
  - beforeMove()
  - afterMove()

- protected void beforeMove() {
  gui.dispose();
  gui.setVisible(false);
  System.out.println(getLocalName()+" is now moving elsewhere.");
}

- protected void afterMove() {
  System.out.println(getLocalName()+" is just arrived to its new location.");
  // creates and shows the GUI
  gui = new MobileAgentGui(this);
  //if the migration is via RMA the variable nextSite can be null.
  gui.setVisible(true);
  addBehaviour(new GetAvailableLocationsBehaviour(this));
}

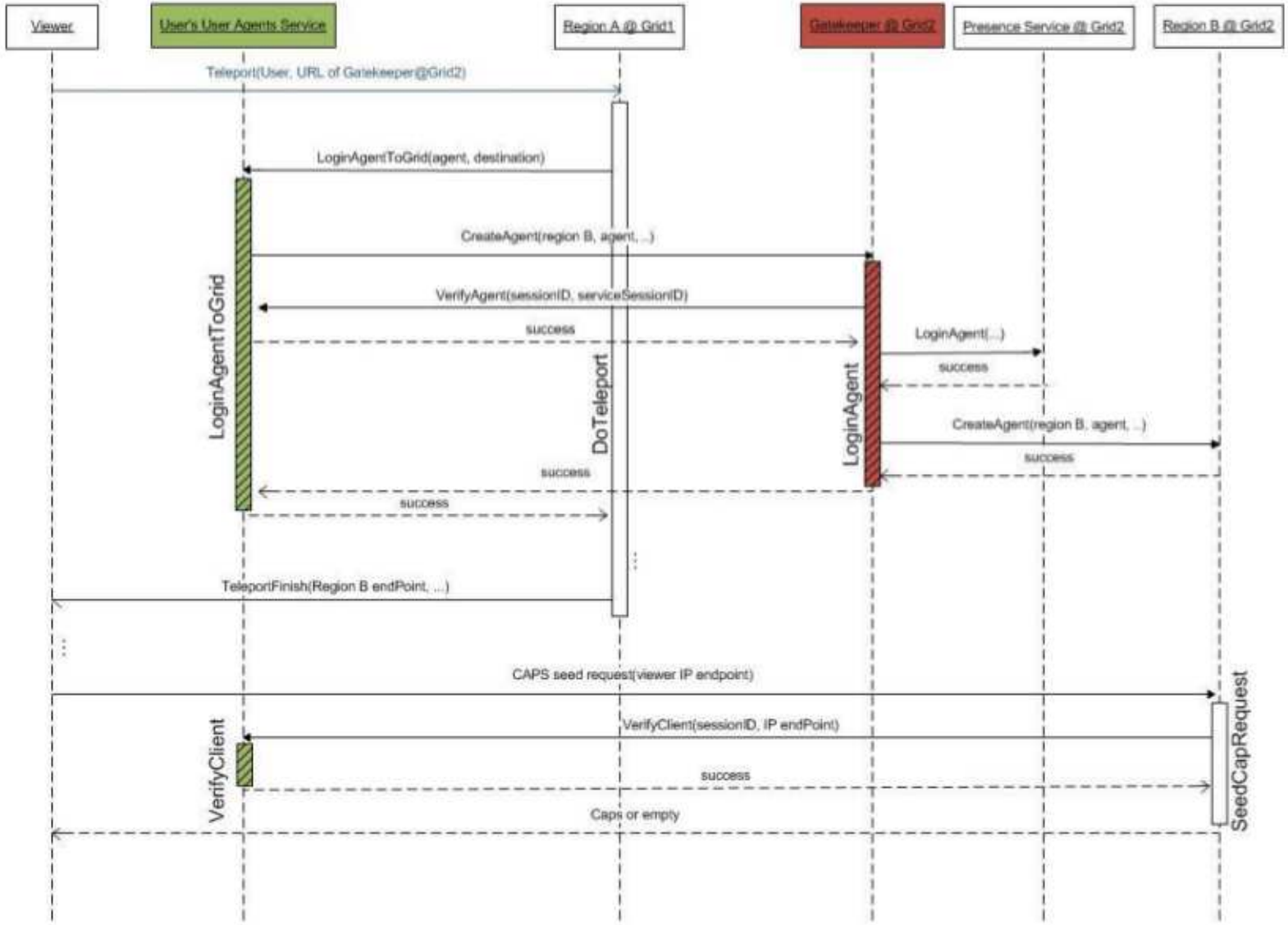
# Paradigme Agents mobiles

## Migration

- **Localement :**
  - **Sérialisation/Marshalling** du code, des données et de l'état d'exécution
  - Contact GateKeeper distant
  - Suppression de l'agent
  - Envoi de l'agent sur le réseau
  - Récupération des ressources
- **Site distant :**
  - **Désérialisation/Unmarshaling** du code, des données et de l'état d'exécution
  - Allocation des ressources
  - Reprise de l'exécution (à une méthode spécifiée ou à l'instruction donnée)

JVM (jade) : Migration faible

User is in Region A @Grid1 and wants to teleport to Region B @ Grid2



# Programmation Orientée Agent

Agents, Objets, quelle(s) différence(s) ?

- Un objet ne peut refuser d'exécuter une méthode lorsque celle-ci est invoquée.
  - Il n'a pas de contrôle sur ton état interne
- Un agent reçoit un message l'invitant à réaliser une action.
  - Il contrôle son cycle d'exécution et son état.

| | Approche « monolithique » | Approche modulaire | Approche objet | Approche agent |
|---|---|---|---|---|
| Comportement | Non modulaire | Modulaire | Modulaire | modulaire |
| Etat | Externe | Externe | Interne | interne |
| Invocation (choix) | Externe | Externe (appel) | Externe (appel de méthode / message) | Interne (but) |

- La programmation objet est un paradigme se focalisant sur les données; comment les représenter et les manipuler. Les concepts principaux y sont les classes, objets et méthodes.
- La programmation agent fait apparaitre la notion d'*action persistante.* Les concepts principaux y sont les buts, rôles, décisions et (inter)actions. [Problème de la sélection de l'action]

# Programmation Orientée Agent

D'un point de vue génie logiciel, pas de « magie »

- Une méthode d'analyse et de conception d'applications sous la forme d'un ensemble d'entités autonomes coopérant (le plus souvent)
- Simplifie la représentation et la résolution de certaines classes de problèmes

| Framework | OOP | AOP |
|---|---|---|
| Unité fondamentale (UF) | Objet | Agent |
| Modèle structurant l'état de l'UF | Inexistant | Croyances, Désirs, Intentions, Capacités, Choix,.. |
| Mécanismes d'exécution | Passage de message et appel de méthodes | Passage de message et appel de méthodes |
| Types de messages | Non structurés | Structurés : inform, request, propose,.. |
| Contraintes sur les méthodes | Aucune | Notions de « haut-niveau » : honnêteté, cohérence,.. |

# Programmation Orientée Agent

- De nombreux langages
  - **Jason (Hübner, Bordini, ...)**
  - **AgentSpeak**
  - **3APL/2APL (Dastani, Meyer, ..)**
  - MetateM (Fisher,...) – logique temporelle
  - ConGoLog/DTGolog (Lesperance/ Boutilier ) – non BDI, pas de mobilité, mais bonne manipulation des plans
  - Teamcore/ MTDP (Milind Tambe, ...)
  - IMPACT (Subrahmanian, Krauss,..)
  - **CLAIM/S-CLAIM (Amal El Fallah Seghrouchni, ...);**
  - **GOAL (Hindriks) - rational agents**
  - BRAHMS (Sierhuis, ...);
    [NASA] Notion d'activité - un agent ne fait jamais "rien" - dormir,respirer... Modélisation humaine facilitée
  - STAPLE (Kumar, Cohen, Huber);
  - Go! (Clark, McCabe);
  - MINERVA (Leite, ...);
  - FLUX (Thielscher);
  - JIAC (Hirsch, ...);
  - **JADE(Agostino Poggi, ...); (Java, proche POO)**
  - **Jadex (Braubach, Pokahr)- XMLbased;**
    perform goals that call for an action to be performed, achieve goals to meet certain criteria, query goals to gather information and maintain goals to make sure certain facts remain applicable
    https://docs.google.com/document/d/1wI0Q6z1BRH_X8S5m4_AzrIpgLuzeNcKjR8C7Z4woQ5g/edit# récupérer les 4
  - **JACK (AOS)**
  - Agentis (Agentis Software);
  - ...

# Programmation Orientée Agent

- 3APL/2APL/JASON, GOAL
  - Xml pour la définition du SMA
  - Programmation déclarative (pas d'état interne)
  - On décrit le problème (quoi) mais pas la structure de contrôle relative à l'implémentation de la solution.
  - Dans le cas présent, c'est le moteur d'inférence qui a la charge de cette tâche (Prolog)

Remarque : hypothèse du monde clos (c)

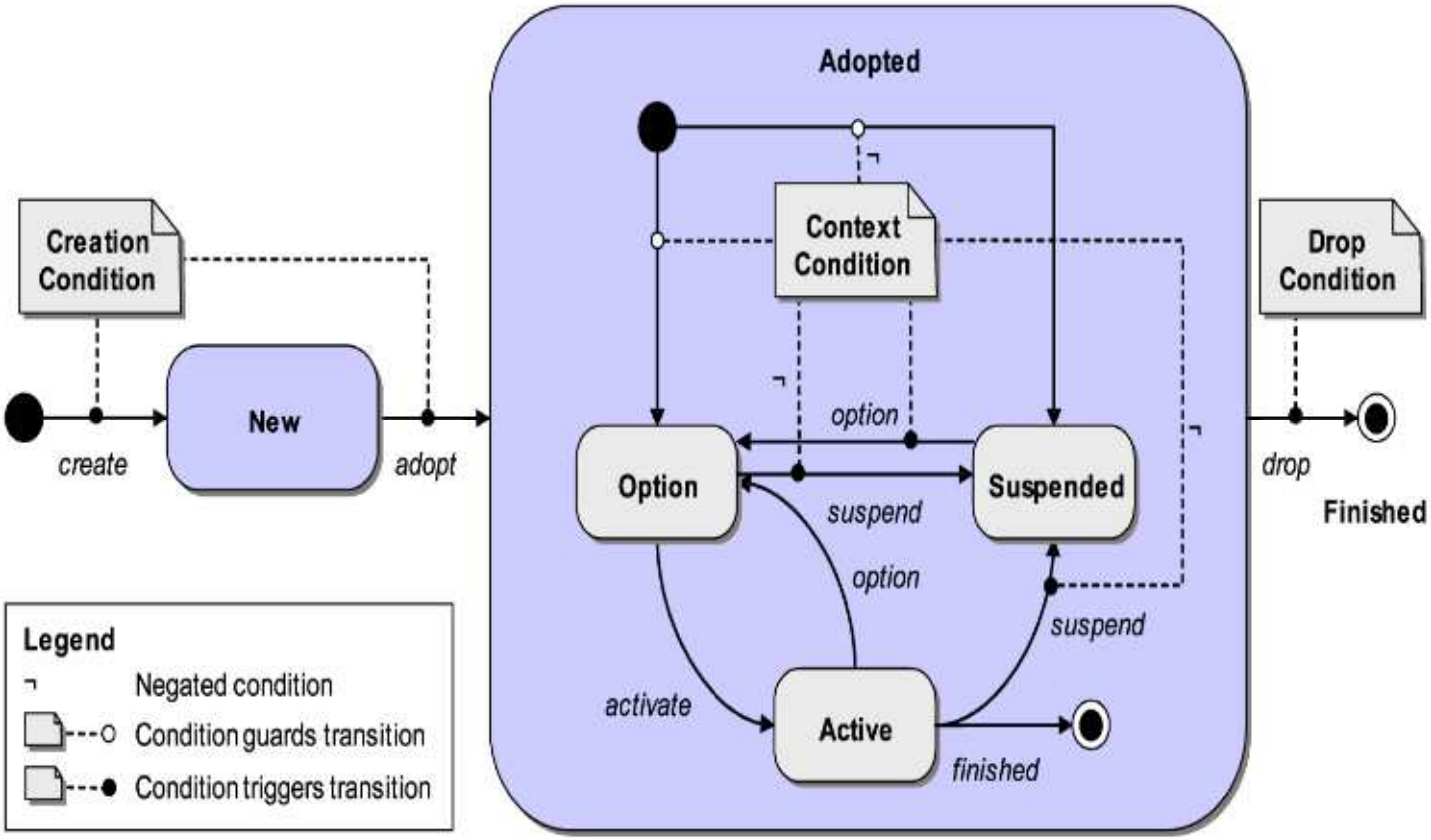
```
PROGRAM "cleaning"

CAPABILITIES{
    { pos(P) }                Goto(R)          { NOT pos(P) , pos(R) },
    { pos(P) AND dirty(R) }   Vacuum(R)        { NOT dirty(R) },
    { pos(P1) AND box(P1)}    Movebox(P1,P2)   { NOT pos(P1), NOT box(P1), pos(P2), box(P2)},
    {TRUE}                    IsClean()        {clean()},
    {TRUE}                    Transported()    {transport()}
}

BELIEFBASE{
    dirty(room1).
    dest(room1).
    box(room2).
    pos(room3).
}

GOALBASE{ clean(), transport() }

PLANBASE{ }

PG-RULES{
     clean() <- dirty(Room) |
                {   Goto(Room);
                    Vacuum(Room);
                    if not dirty(R) then IsClean()
                },
     transport() <- box(Room) AND dest(Dest) |
                {   Goto(Room);
                    Movebox(Room,Dest);
                    if box(Dest) then Transported()
                }
}

PR-RULES{}
```

# Programmation Orientée Agent

- Jadex (Braubach *et al*)
  - Basé sur XML
  - Quatre types de buts
    - Achieve : Atteinte d'un état donné
    - Perform : Une action a réaliser
    - Query : Récupération d'information
    - Maintain : Atteinte de faits relativement au temps

# Représentation et cycle de vie des buts

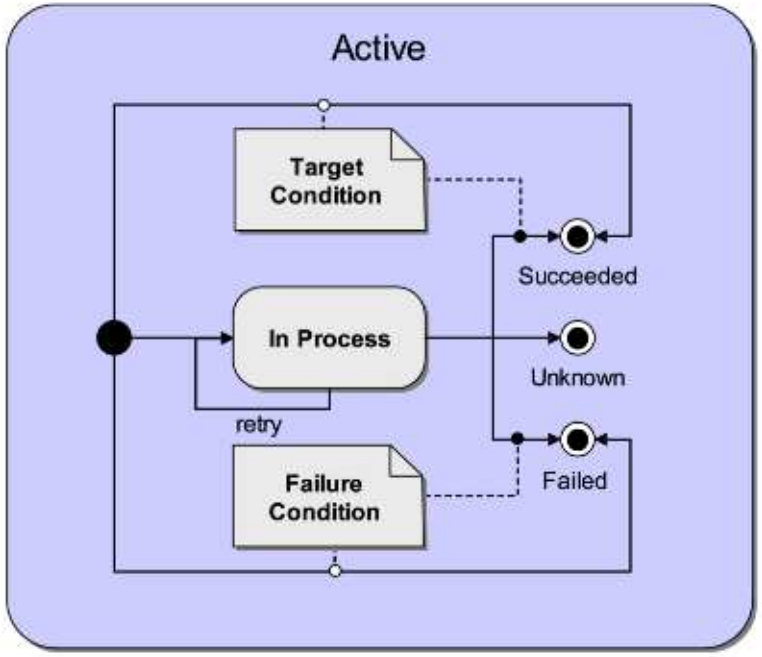
[Braubach]



# Représentation et cycle de vie des buts

[Braubach]

- Achieve goal

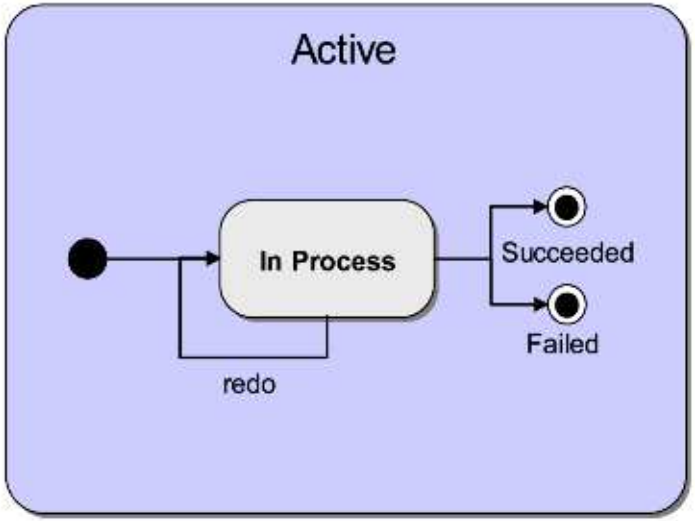



- AchieveGoal(t, P, f ) :
  - Atteindre la condition t en utilisant l'ensemble de Plan P; L'opération est un échec si la propriété f est vérifiée.

# Représentation et cycle de vie des buts

[Braubach]

- Perform goal

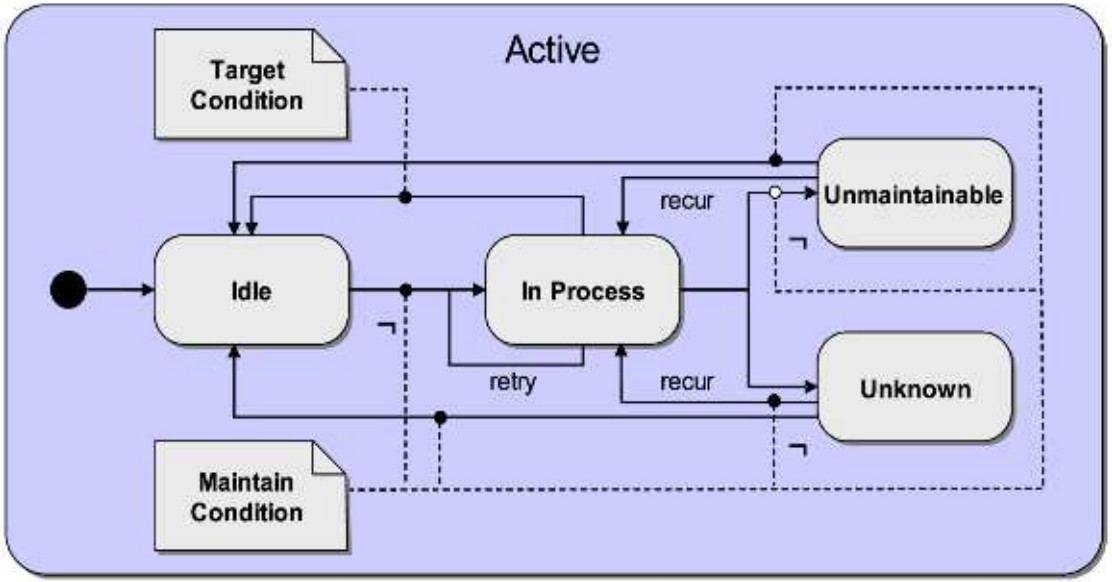



- Contrairement à un « achieve goal », les conséquences d'un « perform goal » dépendent du contexte et de l'exécution du plan.
- Ce processus est un succès lorsque un plan correspondant au but donné est exécuté; le but est abandonné si aucun plan ne permet de le réaliser.

# Représentation et cycle de vie des buts

[Braubach]

- Maintain goal




- L'agent observe l'état de l'environnement et agit lorsque la condition à maintenir n'est plus vérifiée.
- Une durée ou une condition cible peut être associée afin de définir une condition d'abandon de ce but.

# Représentation et cycle de vie des buts

- Problèmes ouverts
  - Délégation de buts (cas coopératifs/compétitifs)
  - Gestion des priorités
    - Quel but poursuivre en premier
      - Environnement simple : Définition d'un ordre total top-down
      - Environnement complexe : Génération automatique
        - » Prise en compte des poids des sous-buts.
        - » Fonction de cout à charge du programmeur
        - » Chemin optimal = séquence de sous-but de poids le plus élevé
    - Problèmes d'interdépendance des buts et sous-buts
      - Situations d'interblocage

# Programmation Orientée Agent

- Langages d'agents « classiques »
  - Grande expressivité relativement :
    - À l'état mental d'un agent
    - Au processus de réflexion humain tel qu'un certain nombre de personne l'imagine aujourd'hui
  - Ne supportent pas :
    - la représentation explicite de la concurrence [CLAIM]
    - la mobilité [2APL,..]

# Programmation Orientée Agent

- CLAIM/S-CLAIM [Suna, El Fallah Seghrouchni,..]
  - Approche cognitive
    - Connaissance, buts et capacités
  - Primitives de mobilités
    - Calcul des ambiants (CLAIM)
  - Sensible au contexte
    - Structure de représentation hierarchique des agents

[ANT]