JADE

Java Agent DEvelopment Framework

Κινητοί Πράκτορες (mobile agents)

- Ένας πράκτορας λογισμικού είναι μια προσανατολισμένη σε κάποιο σκοπό υπολογιστική οντότητα που μπορεί να αντιδρά σε ερεθίσματα του περιβάλλοντός της και η οποία έχει ένα μεγάλο βαθμό αυτονομίας στο να επιλέξει τις λειτουργίες που θα εκτελέσει για να επιτύχει αυτό το σκοπό
- Προγραμματιστικά:
  - αντικείμενο με δικό του νήμα εκτέλεσης,
  - έχει τη δυνατότητα να μεταφέρεται (migrate) ανάμεσα σε διαφορετικές τοποθεσίες ενός δικτύου
  - διατηρεί κάποια πληροφορία κατάστασης
- Χαρακτηριστικά: αυτονομία (autonomy), ικανότητα αντίδρασης (reactivity), κοινωνικότητα (sociability), προσαρμοστικότητα (adaptability), αξιοπιστία (reliability), κινητικότητα (mobility)

Συστήματα κινητών πρακτόρων

- ύπαρξη κάποιας υποδομής (agent platform) - παρέχει λειτουργικότητα σχετική με τις απαιτήσεις που έχει ένα σύστημα πολλαπλών πρακτόρων (π.χ. κινητικότητα)
- διαλειτουργικότητα μεταξύ διαφορετικών πλατφορμών
  - ορισμός της βασικής δομής και συστατικών μια πλατφόρμας
  - διάφορα πρότυπα για τον τρόπο δόμησης μιας τέτοιας πλατφόρμας
    - OMG MASIF (Mobile Agent System Interoperability Facilities)
    - FIPA (FOUNDATION FOR INTELLIGENT PHYSICAL AGENTS)

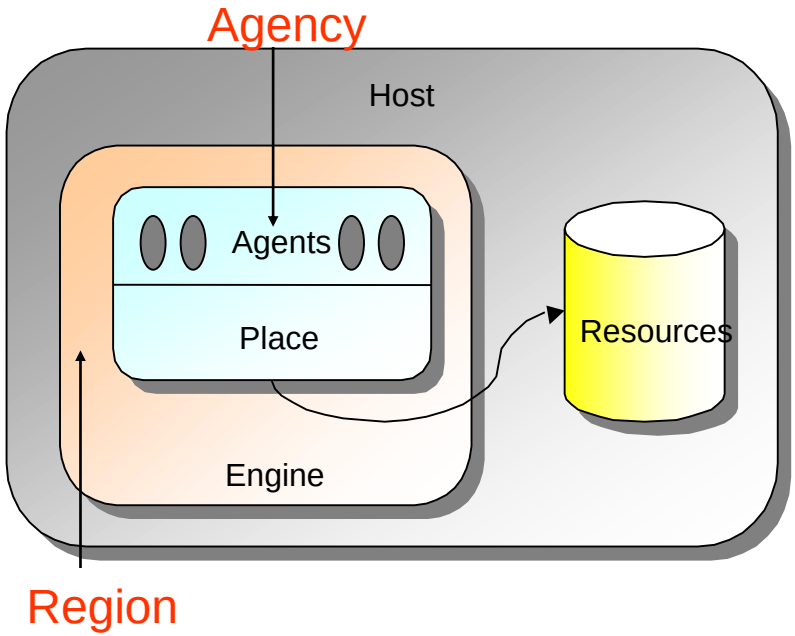
OMG MASIF

Κινητοί πράκτορες - το πρότυπο MASIF

- **Πρακτορείο (Agency):** Παρέχει το περιβάλλον εκτέλεσης των κινητών πρακτόρων. Κάθε πρακτορείο εκτελείται σε ξεχωριστή εικονική μηχανή της Java και αποτελείται από μία ή περισσότερες τοποθεσίες. Το πρακτορείο παρέχει τις βασικές λειτουργίες, όπως διαχείριση του κύκλου ζωής του πράκτορα, μεταφορά, επικοινωνία και ασφάλεια.

- **Τοποθεσία (Place):** Επιτρέπει την οργάνωση των πρακτόρων, που παρέχουν ίδιες υπηρεσίες. Οι τοποθεσίες ανήκουν στα πρακτορεία. Κάθε τοποθεσία μπορεί να συσχετιστεί με μια πολιτική ασφάλειας (security policy), για τη διευκόλυνση της διαχείρισης της ασφάλειας.

- **Πράκτορας (agent):** Οντότητα λογισμικού, που έχει τη δυνατότητα να δρα αυτόνομα, αποτελούμενη από μία ή περισσότερες Java κλάσεις. Υπάρχουν δύο είδη πρακτόρων: οι κινητοί (mobile) και οι στάσιμοι (stationary).

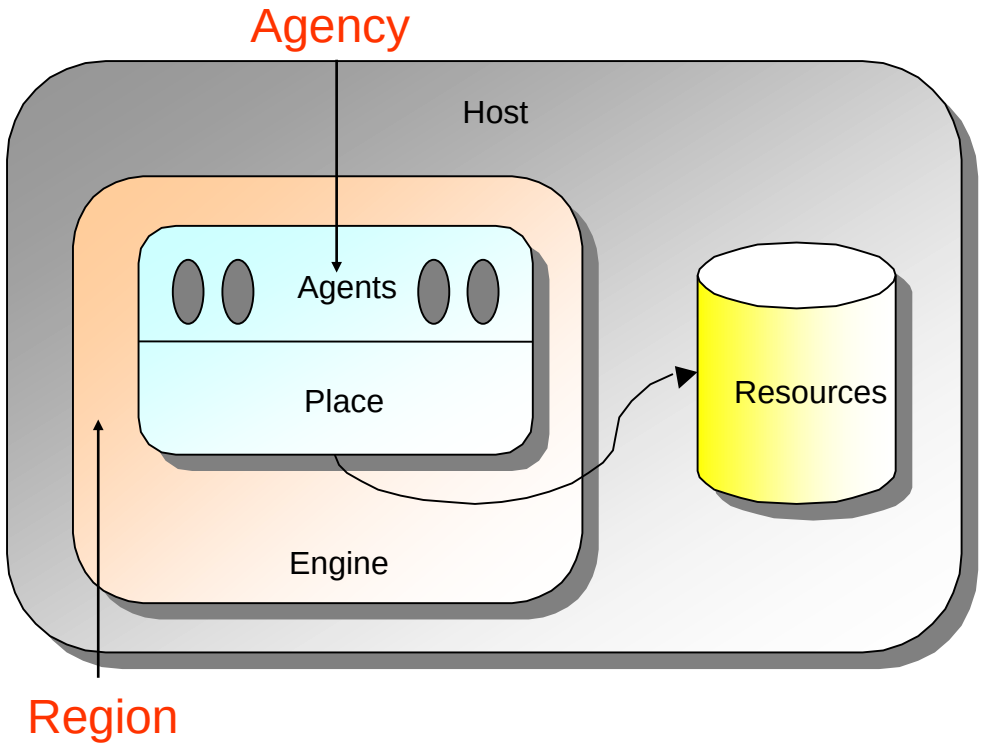


OMG MASIF

Κινητοί πράκτορες – το πρότυπο OMG MASIF

Περιοχή Καταχώρησης (Region Registry): Παρέχει τη βάση δεδομένων με πληροφορία σχετική για τα πρακτορεία, τοποθεσίες και τους πράκτορες μιας περιοχής.

Περιοχή (Region): Διευκολύνει τη διαχείριση των πρακτορείων (agencies), τοποθεσιών (places) και πρακτόρων (agents). Μία περιοχή περιέχει μία μόνον περιοχή καταχώρησης (region registry) και πολλά πρακτορεία.

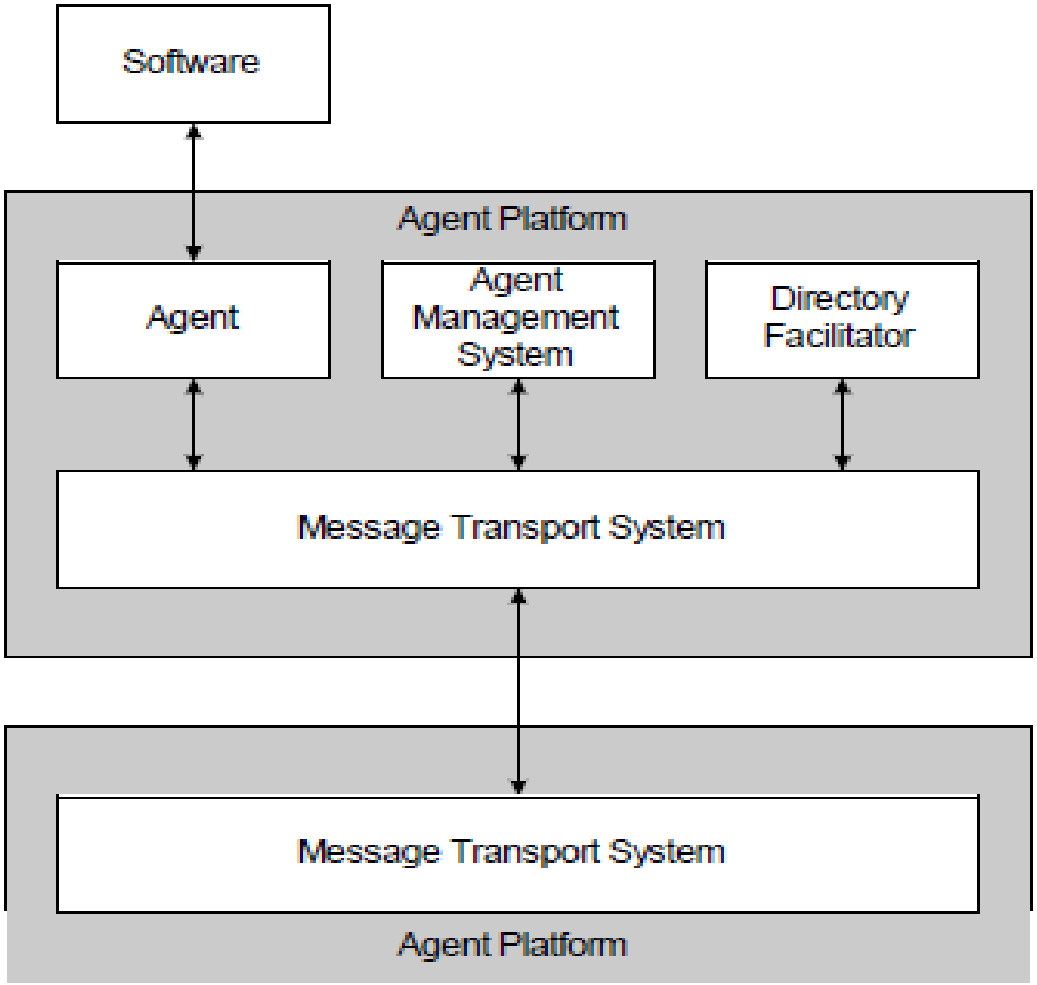


Το πρότυπο FIPA (1)

Μια πλατφόρμα κατά το πρότυπο FIPA αποτελείται από τις εξής λογικές οντότητες:

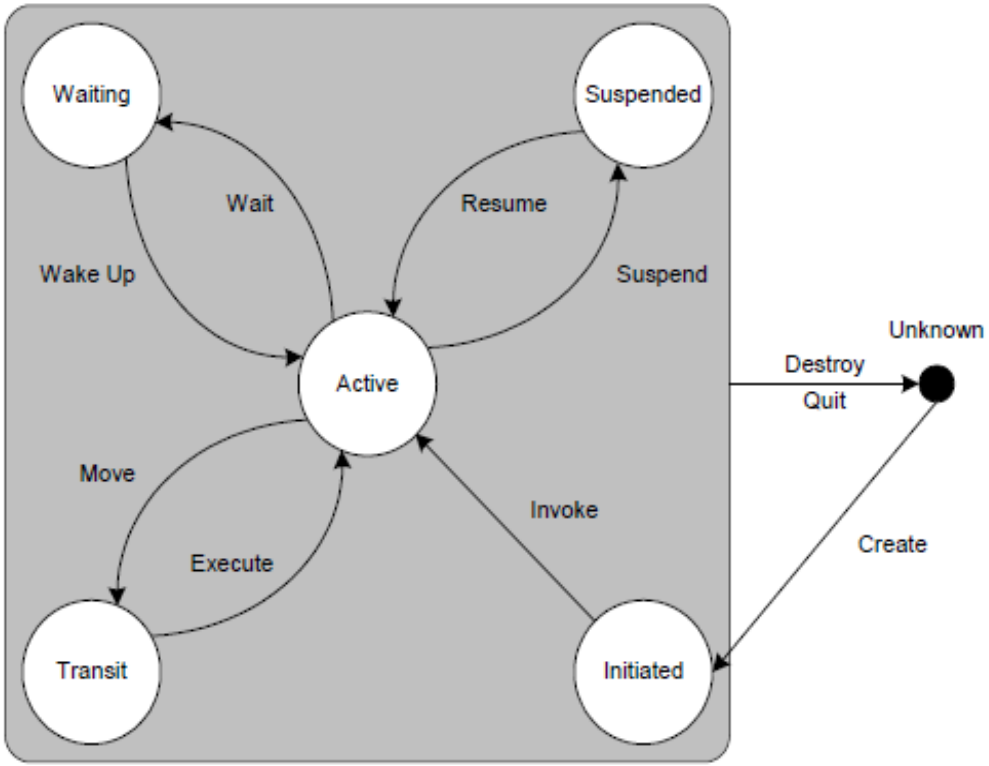
- agent:
  - υπολογιστική διεργασία που υλοποιεί τη λειτουργικότητα μιας εφαρμογής.
  - πρέπει να έχει ένα στοιχείο ταυτότητας
- directory facilitator:
  - λειτουργεί ως υπηρεσία καταλόγου
  - χρησιμοποιείται για τον εντοπισμό των agents
- agent management system:
  - μοναδική οντότητα σε κάθε πλατφόρμα
  - υπεύθυνη για τον έλεγχο της πρόσβασης και τη χρήση της πλατφόρμας
- message transport system:
  - ο τρόπος επικοινωνίας μεταξύ διαφορετικών agents που βρίσκονται μέσα σε μια πλατφόρμα ή σε διαφορετικές πλατφόρμες

Το πρότυπο FIPA(2)



Το πρότυπο FIPA - Agent Life Cycle

- INITIATED: the Agent object is built, but hasn't registered itself yet with the AMS, has neither a name nor an address and cannot communicate with other agents.
- ACTIVE : the Agent object is registered with the AMS, has a regular name and address and can access all the various JADE features.
- SUSPENDED : the Agent object is currently stopped. Its internal thread is suspended and no agent behaviour is being executed.



- WAITING : the Agent object is blocked, waiting for something. Its internal thread is sleeping on a Java monitor and will wake up when some condition is met (typically when a message arrives).
- DELETED : the Agent is definitely dead. The internal thread has terminated its execution and the Agent is no more registered with the AMS.
- TRANSIT: a mobile agent enters this state while it is migrating to the new location. The system continues to buffer messages that will then be sent to its new location.

JADE - Εισαγωγή

- υποδομή λογισμικού που παρέχει λειτουργίες για την ανάπτυξη συστημάτων πολλαπλών αντιπροσώπων
- συμφωνεί με τα FIPA (Foundation for Intelligent Physical Agents) πρότυπα
- είναι εξ' ολοκλήρου υλοποιημένο σε Java
- περιλαμβάνει τα εξής:
  - ένα περιβάλλον εκτέλεσης
  - μια βιβλιοθήκη κλάσεων
  - μια σειρά γραφικών εργαλείων για την επίβλεψη και τον έλεγχο των δραστηριοτήτων των αντιπροσώπων

- Home page: **http://jade.tilab.com**

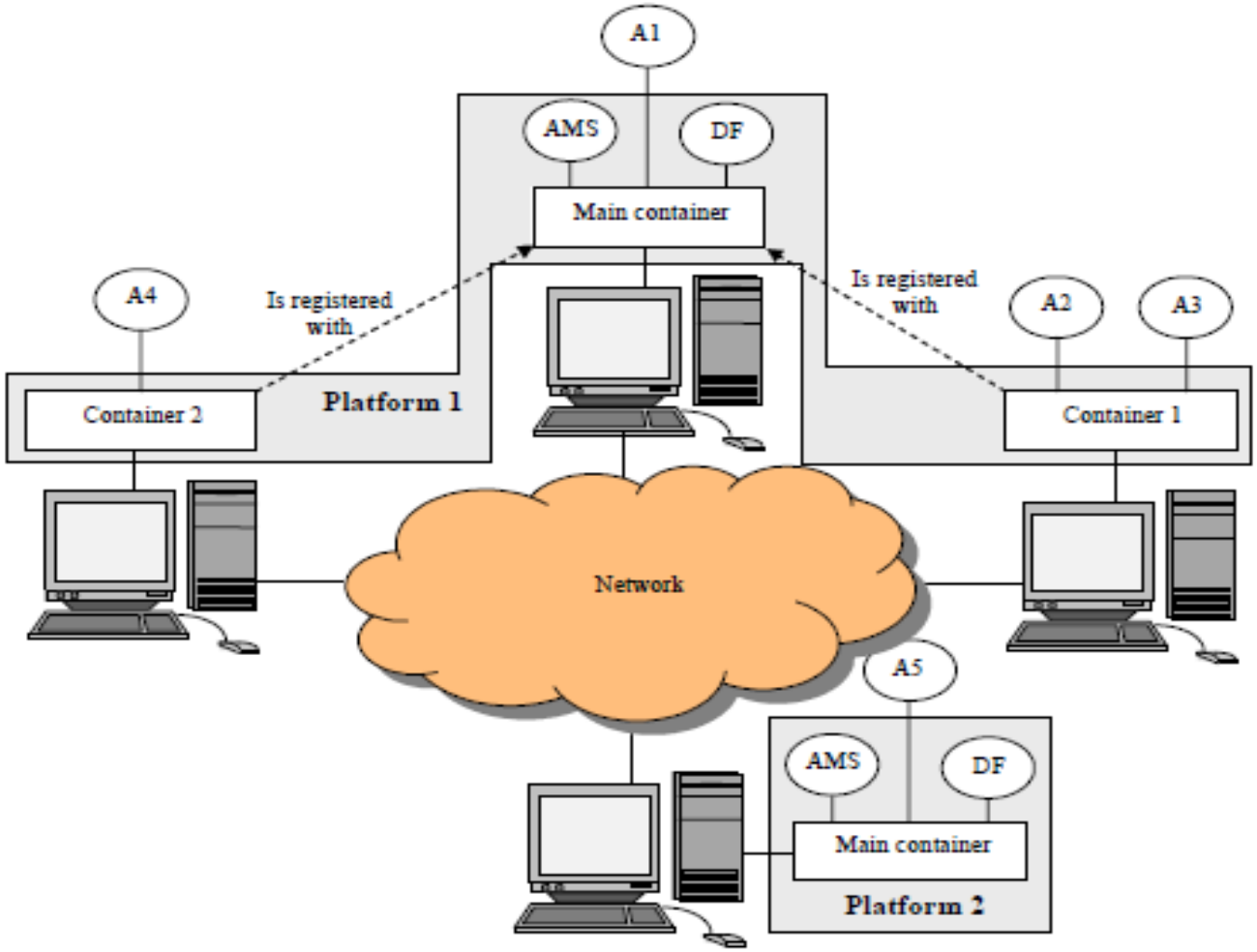
Jade - Βασικά δομικά στοιχεία(1)

- **Container**: ένα στιγμιότυπο του περιβάλλοντος εκτέλεσης του Jade
  - *main container*:
    - το πρώτο container που δημιουργείται
    - απαραίτητο και μοναδικό σε κάθε πλατφόρμα
    - περιέχει δύο ειδικούς αντιπρόσωπους για διαχείριση του συστήματος και για παροχή υπηρεσιών καταλόγου (AMS και DF)
  - *secondary containers*:
    - δημιουργούνται μετά το main container
    - υποχρεωτικά καταχωρούνται σε ένα main container
- **Platform**: το σύνολο των ενεργών containers - αποτελείται από ένα μοναδικό main container και κανένα ή περισσότερα secondary containers

Jade - Βασικά δομικά στοιχεία(2)

- **AMS** (Agent Management System)
  - παρέχει υπηρεσίες ονοματολογίας (π.χ. φροντίζει κάθε πράκτορας σε μια πλατφόρμα να έχει μοναδικό όνομα)
  - αναπαριστά την διευθύνουσα αρχή μιας πλατφόρμας (π.χ. ελέγχει τη δημιουργία και τον τερματισμό των αντιπροσώπων)
- **DF** (Directory Facilitator):
  - παρέχει υπηρεσίες καταλόγου, μέσω των οποίων είναι δυνατόν ένας αντιπρόσωπος να εντοπίσει άλλους αντιπροσώπους

Jade - Βασικά δομικά στοιχεία(3)



Agent Identifier (AID)

- δίνεται σε κάθε agent από τον AMS κατά τη δημιουργία του
- χρησιμοποιείται ως αναγνωριστικό για το διαχωρισμό του δεδομένου agent από τους υπόλοιπους agents της πλατφόρμας
- ένα AID αποτελείται από τα εξής:
  - ένα καθολικά μοναδικό όνομα
  - ένα σύνολο από διευθύνσεις για τον agent:
    - κληρονομούνται από τις διευθύνσεις της πλατφόρμας στην οποία αυτός δημιουργήθηκε
    - χρησιμοποιούνται από κάποιον άλλο agent που επιθυμεί να επικοινωνήσει με αυτόν
  - διάφορα άλλα πεδία για τον προσδιορισμό άλλων χαρακτηριστικών (π.χ. κατάσταση) του agent

Δημιουργία αντιπροσώπων

- Η δημιουργία μιας κλάσης αντιπροσώπου γίνεται κάνοντας extend την κλάση jade.core.Agent και override την μέθοδο setup() που ορίζεται σε αυτήν.

```
import jade.core.Agent;
public class HelloWorldAgent extends Agent{
        protected void setup(){
        System.out.println(“Hello World! my name is”
                                +this.getAID().getName());
    }
}
```

- Κάθε στιγμιότυπο ενός agent ταυτοποιείται από ένα αντικείμενο της κλάσης jade.core.AID.

  - ένα AID αποτελείται από ένα μοναδικό όνομα και κάποιες διευθύνσεις

  - το AID μπορεί να ανακτηθεί μέσω της μεθόδου Agent.getAID().

Ονοματολογία αντιπροσώπων

- Τα ονόματα των agents είναι της μορφής

  <localName>@<platformName>
- Το πλήρες όνομα ενός agent πρέπει να είναι καθολικά μοναδικό
- Μέσα σε μια Jade πλατφόρμα η αναφορά σε κάποιον agent γίνεται μόνο μέσω του localName
- Γνωρίζοντας το όνομα ενός agent, μπορούμε να ανακτήσουμε το AID του ως εξής:

  AID id = new AID(localname, AID.ISLOCALNAME);

Πέρασμα ορισμάτων σε έναν agent

- Κατά την εκκίνηση ενός agent (είτε από γραμμή εντολών, είτε προγραμματιστικά), είναι δυνατόν να περαστούν σε αυτόν κάποια ορίσματα.
- Ο agent έχει πρόσβαση σε αυτά τα ορίσματα μέσω της μεθόδου getArguments() της κλάσης jade.core.Agent

```
protected void setup() {
        System.out.println("Hello World! my name is " + this.getAID().getName());
        Object[] args = this.getArguments();
        if (args != null) {
                System.out.println("My arguments are:");
                for (int i = 0; i < args.length; ++i) {
                        System.out.println("- "+args[i]);
                }
        }
}
```

Τερματισμός αντιπροσώπων

- Ένας agent τερματίζει όταν κληθεί η μέθοδος doDelete()
- Κατά τον τερματισμό ενός agent καλείται η μέθοδος takeDown() χρησιμοποιείται για αποδέσμευση πόρων που πιθανώς χρησιμοποιεί ο agent

```
protected void setup() {
        System.out.println("Hello World! my name is "  +  getAID().getName());
        Object[] args = getArguments();
        if (args != null) {
                System.out.println("My arguments are:");
                for (int i = 0; i < args.length; ++i) {
                        System.out.println("- "+args[i]);
                }
        }
        this.doDelete();
}
protected void takeDown() {
        System.out.println("Bye...");
}
```

Η κλάση Behaviour

- Οι κυρίως εργασίες ενός agent εκτελούνται μέσα σε “behaviours”
- Οι behaviours δημιουργούνται κάνοντας extend την κλάση jade.core.behaviours.Behaviour
- Για να εκτελέσει ένας agent μια behaviour χρειάζεται η δημιουργία ενός αντικειμένου της συγκεκριμένης υποκλάσης και η κλήση της μεθόδου addBehaviour() της κλάσης Agent.
- Κάθε υποκλάση της κλάσης Behaviour πρέπει να υλοποιεί τις ακόλουθες μεθόδους:

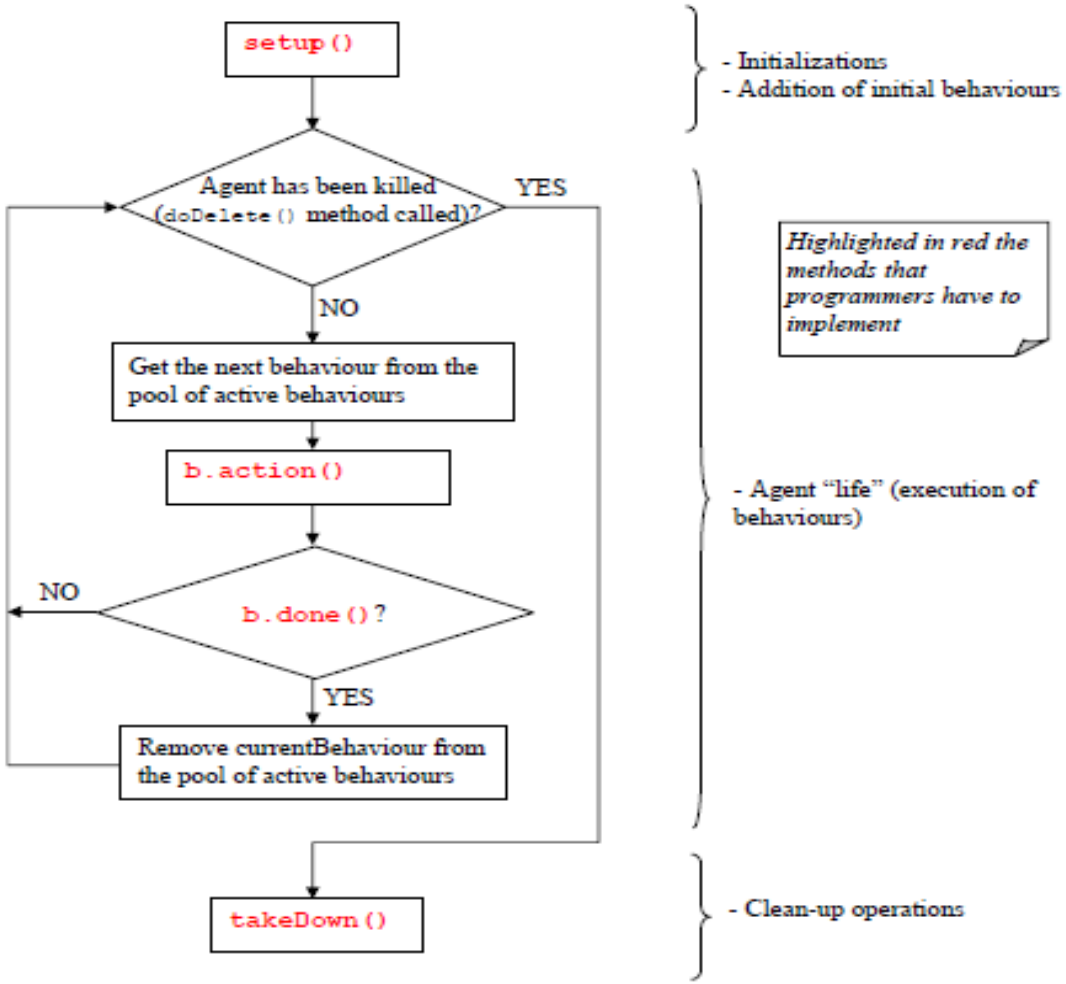
  - public void action(): ορίζει τις ενέργειες που πραγματοποιεί η δεδομένη behaviour

  - public boolean done(): καθορίζει αν η behaviour έχει τερματίσει και εκτελείται μετά τον τερματισμό της action()

Εκτέλεση των behaviours

- Ένας agent μπορεί να εκτελεί παράλληλα πολλές διαφορετικές behaviours
- Η προσέγγιση του Jade είναι ότι κάθε agent τρέχει σε ένα μοναδικό νήμα – γι’ αυτό το λόγο οι behaviours δεν εκτελούνται ταυτόχρονα σε διαφορετικά νήματα
- Αλλαγή στην εκτελούμενη behaviour γίνεται μόνο όταν η μέθοδος action() της αρχικά εκτελούμενης τερματίσει
- Οι behaviours που δεν έχουν ολοκληρωθεί διατηρούνται σε ένα behaviour pool και εναλλάσσονται κυκλικά, με τη σειρά που προστέθηκαν
- Μια behaviour απομακρύνεται από το behaviour pool όταν η μέθοδος done() της behaviour επιστρέψει “true”

Μοντέλο εκτέλεσης ενός JADE agent



Βασικοί Τύποι Behaviours

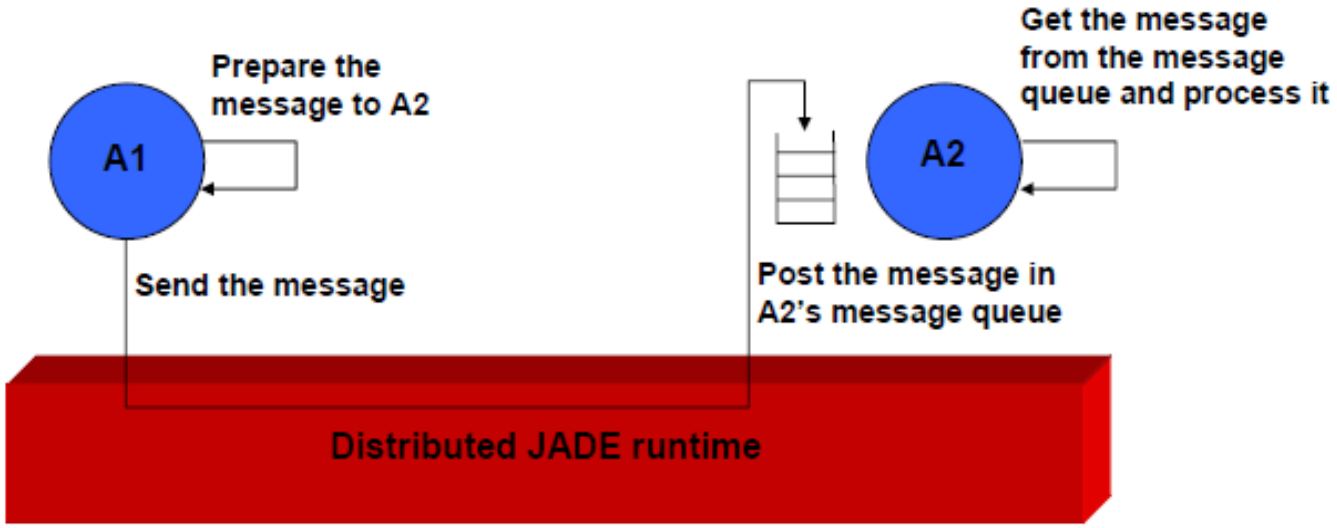
- “One shot” behaviours:
  - επεκτείνουν την κλάση jade.core.behaviours.OneShotBehaviour
  - η μέθοδος done() επιστρέφει “true”, επομένως η μέθοδος action() εκτελείται μόνο μία φορά
- “Cyclic” behaviours:
  - επεκτείνουν την κλάση jade.core.behaviours.CyclicBehaviour
  - η μέθοδος done() επιστρέφει πάντα “false”, επομένως η μέθοδος action() εκτελείται επαναληπτικά μέχρι να τερματιστεί ο agent
- “Complex” behaviours:
  - περιέχουν κάποιες μεταβλητές κατάστασης και η μέθοδος action() μπορεί να εκτελεί διάφορες λειτουργίες ανάλογα με την κατάσταση
  - η μέθοδος done() επιστρέφει “true” όταν ικανοποιείται κάποια συνθήκη

Περισσότερα για τις behaviours

- μέθοδος onStart():
  - καλείται μόνο μία φορά πριν από την πρώτη κλήση της action()
  - μπορεί να χρησιμοποιηθεί για ενέργειες αρχικοποίησης
- μέθοδος onEnd(): καλείται μόνο μία φορά, όταν η μέθοδος done() επιστρέψει “true”
- protected πεδίο myAgent - αναφορά στον agent στον οποίο ανήκει η δεδομένη Behaviour
- μέθοδος Agent.removeBehaviour() - μπορεί να χρησιμοποιηθεί για την απομάκρυνση μιας behaviour από το pool των active behaviours
- Όταν το pool των active behaviours ενός agent αδειάσει, τότε ο agent περνάει σε κατάσταση IDLE

Μοντέλο επικοινωνίας

- Η επικοινωνία μεταξύ των αντιπροσώπων βασίζεται στην ασύγχρονη ανταλλαγή μηνυμάτων
- Τα μηνύματα είναι αντικείμενα της κλάσης jade.lang.acl.ACLMessage, στην οποία ορίζονται μέθοδοι για τον καθορισμό διαφόρων ιδιοτήτων (π.χ. αποστολέας, αποδέκτες, σκοπός επικοινωνίας κτλ)



Αποστολή και λήψη μηνυμάτων (1)

- Αποστολή μηνυμάτων:

```
ACLMessage msg = new ACLMessage(ACLMessage.INFORM);
msg.addReceiver(new AID(“Peter”, AID.ISLOCALNAME));
msg.setLanguage(“English”);
msg.setOntology(“Weather-Forecast-Ontology”);
msg.setContent(“Today it’s raining”);
send(msg);
```

- Λήψη μηνυμάτων:

```
public void action() {
        ACLMessage msg = myAgent.receive();
        if (msg != null) {
                // Process the message
        }
        else {
                block();
        }
 }
```

Η μέθοδος block() βγάζει τη συγκεκριμένη behaviour από το pool των active behaviours και αλλάζει την κατάστασή της σε “blocked”

Αποστολή και λήψη μηνυμάτων (2)

- Λήψη μηνυμάτων σε blocking mode:
  - μέθοδος blockingReceive(): επιστρέφει μόνο όταν υπάρχει κάποιο μήνυμα διαθέσιμο στην ουρά του agent
  - χρειάζεται προσοχή στην χρήση της γιατί πρόκειται για blocking call, οπότε αν δεν υπάρχει μήνυμα στην ουρά μπλοκάρει η ροή όλων των behaviours
- Κοινή πρακτική: χρήση του συνδυασμού receive()-block() μέσα σε behaviours και η χρήση της blockingReceive() μέσα στις setup() και takedown()
- δυνατότητα επιλογής μηνυμάτων με συγκεκριμένα χαρακτηριστικά από την ουρά, περνώντας ένα αντικείμενο της κλάσης jade.lang.acl. Message Tem- plate σαν όρισμα στην receive()/blockingReceive() – χρήσιμο όταν δύο ή περισσότερες behaviours ενός agent αναμένουν τη λήψη διαφορετικών μηνυμάτων

Κινητικότητα (1)

- Το JADE υποστηρίζει κινητικότητα κατάστασης και κώδικα:

  - κατάσταση: ένας agent έχει τη δυνατότητα i) να σταματήσει την εκτέλεσή του σε έναν container, ii) να μετακινηθεί σε έναν απομακρυσμένο container iii) να επανεκκινήσει την εκτέλεσή του από το σημείο στο οποίο διακόπηκε αρχικά η εκτέλεσή του

  - κώδικας: αν ο κώδικας του agent που μετανάστευσε δεν υπάρχει διαθέσιμος στον απομακρυσμένο container, τότε ανακτάται αυτόματα
- Για να μπορεί ένας agent να μεταναστεύσει, πρέπει να υλοποιεί το Serializable interface της Java
- Η μετανάστευση μπορεί να ξεκινάει είτε από αίτηση του ίδιου του agent (με κλήση της μεθόδου doMove()), είτε να επιβάλλεται από τον AMS (π.χ. λόγω αίτησης κάποιου άλλου agent)

Κινητικότητα (2)

- doMove(): μετανάστευση agent σε απομακρυσμένο container
- doClone(): δημιουργία αντιγράφου του agent με διαφορετικό όνομα σε απομακρυσμένο container
- και οι δύο μέθοδοι παίρνουν σαν παράμετρο ένα αντικείμενο της κλάσης jade.core.Location (abstract κλάση, πρέπει να ανακτηθεί από τον AMS)
- beforeMove() – beforeClone(): εκτελούνται ακριβώς πριν εκτελεστεί η αντίστοιχη μέθοδος doMove ή doClone
- afterMove() – afterClone()

Πρωτόκολλα Αλληλεπίδρασης

- Καθορισμένα πρότυπα πρωτοκόλλων αλληλεπίδρασης για την συνομιλία ανάμεσα σε agents
- για κάθε συνομιλία, γίνεται διάκριση σε *initiator* (ο αντιπρόσωπος που ξεκινά τη συνομιλία) και σε *responder*
- το Jade παρέχει διάφορες κλάσεις συμπεριφορών, τόσο για το ρόλο του initiator, όσο και για το ρόλο του responder
- όλες οι initiator behaviours είναι one shot behaviours, ενώ οι responder behaviours είναι cyclic behaviours
- κλάσεις AchieveREInitiator/Responder: υλοποιούν με ομογενή και μοναδικό τρόπο τα πρότυπα πρωτοκόλλων αλληλεπίδρασης

SimpleAchieveREInitiator
(RE=**R**ational **E**ffect)

- πρόκειται για behaviour που δημιουργείται δίνοντας σαν όρισμα στον constructor ένα μήνυμα (το οποίο θα χρησιμοποιηθεί για την επικοινωνία με τον responder)
- μπορεί να χρησιμοποιηθεί για την επικοινωνία με τον AMS (π.χ. για τη λήψη των διαθέσιμων containers μιας πλατφόρμας)
- κάνοντας override τις μεθόδους handleNotUnderstood, handleRefuse, handleFailure, handleAgree, handleInform μπορούν να οριστούν οι ενέργειες που πρέπει να γίνουν ανάλογα με το είδος του μηνύματος που θα ληφθεί σαν απόκριση από τον responder

SimpleAchieveREInitiator - παράδειγμα

```
ACLMessage request = new ACLMessage(ACLMessage.REQUEST);
request.setProtocol(FIPANames.InteractionProtocols.FIPA_REQUEST);
request.addReceiver(new AID(“receiver”, AID.ISLOCALNAME));
myAgent.addBehaviour( new AchieveREInitiator(myAgent, request)
{
   protected void handleInform(ACLMessage inform)
   {
         System.out.println(“Protocol finished. Rational Effect achieved.
         Received the following message: ”+inform);
   }
});
```

SimpleAchieveREResponder

- Πρόκειται για behaviour που δημιουργείται δίνοντας σαν όρισμα στον constructor το είδος του μηνύματος που θα εξυπηρετείται από την behaviour
- Κάνοντας override τις μεθόδους prepare… (π.χ. prepareResponse, prepareResultNotification κτλ) μπορούν να οριστούν οι τρόποι χειρισμού των αιτήσεων και δημιουργίας των μηνυμάτων-αποκρίσεων
- Χρησιμοποιείται από τον AMS για την παροχή διάφορων πληροφοριών

SimpleAchieveREResponder - παράδειγμα

```
AchieveREResponder.createMessageTemplate(FIPANames.InteractionProtocols.FIPA_REQUEST);
myAgent.addBehaviour( new AchieveREResponder(myAgent, mt)
{
   protected ACLMessage prepareResultNotification(ACLMessage request, ACLMessage response)
   {
            System.out.println(“Responder has received the following message: ” +request);
            ACLMessage informDone = request.createReply();
            informDone.setPerformative(ACLMessage.INFORM);
            informDone.setContent(“inform done”);
            return informDone;
   }
});
```

Χρήση της πλατφόρμας Jade

- Εκκίνηση main container: java jade.Boot –gui
- Εκκίνηση main container μαζί με agent: java jade.Boot –gui *agent_name*:*agent_class*(*arguments*)
- Εκκίνηση container: java jade.Boot –container –host *host_name*

RMA (Remote Monitoring Agent)