A Flexible and Lightweight Agent Deployment Architecture

Andrei Olaru, Alexandru Sorici and Adina Magda Florea

andrei.olaru@cs.pub.ro

AI-MAS Group, University Politehnica of Bucharest

29.05.2019

A Flexible and Lightweight Agent Deployment Architecture

overview

Research Objective

· MAS Frameworks are directed towards

- deployment of loosely coupled systems in a distributed computing environment.
- simulation of a large number of entities on a local machine or a computing cluster.

Current state of the art contains

Jade	Java	very popular	but relatively slow
Jiac	Java EE	industrial-oriented	based on Java EE
JaCaMo	Jason & co	A&A-oriented approach	steep learning curve
AImpulse Spectrum	Java	great performance	no distribution
Repast Suite	Java / C++		

FLASH-MAS – A Fast Lightweight Agent Shell

- fast
- flexible
- lightweight

FLASH-MAS – A Fast Lightweight Agent Shell

- fast
- flexible
- lightweight

- Java
- FIPA-compliant
- target both distributed setups and local simulation
- support for different modes of communication and framework service provision

FLASH-MAS – A Fast Lightweight Agent Shell

- fast
- flexible
- lightweight

- Java
- FIPA-compliant
- target both distributed setups and local simulation
- support for different modes of communication and framework service provision

"Easy for beginners, powerful for experts."

- **fast**
  - fast (easy) configuration
  - fast deployment (start-up)
  - fast execution (e.g. messaging)

- **flexible**
  - flexible means of communication
    [but respecting FIPA standards]
    - different sets of agents can use different means to communicate between each other;
    - the same agents may be able to communicate using different means of communication.
    - e.g. TCP/IP, Jade, using common web services, or WebSockets
  - flexible agent structure
    [but respecting FIPA standards]
  - flexible system structure
  - flexibility in deployment platforms

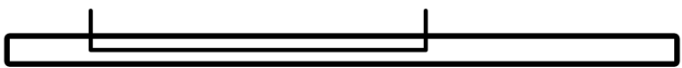
- **flexible**
  - flexible means of communication
    [but respecting FIPA standards]
  - flexible agent structure
    [but respecting FIPA standards]
    - e.g. Jade agents, Composite Agents, sequential agents
  - flexible system structure
  - flexibility in deployment platforms

- **flexible**
  - flexible means of communication
    [but respecting FIPA standards]
  - flexible agent structure
    [but respecting FIPA standards]
    - e.g. Jade agents, Composite Agents, sequential agents
  - flexible system structure
    - e.g. agent arrays, various types of infrastructure
  - flexibility in deployment platforms

- **flexible**
  - flexible means of communication
    [but respecting FIPA standards]
  - flexible agent structure
    [but respecting FIPA standards]
    - e.g. Jade agents, Composite Agents, sequential agents
  - flexible system structure
    - e.g. agent arrays, various types of infrastructure
  - flexibility in deployment platforms
    - e.g. PCs, Android devices, Rasberry Pi, cloud deployment

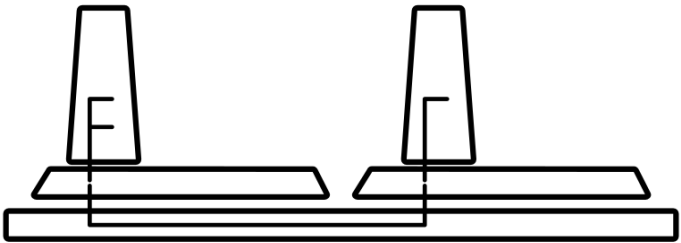
- **lightweight**
  - execute on resource-constrained devices
  - ability to run a large number of agents on the same machine

- Nodes represent the FLASH-MAS presence on a machine;
- Support infrastructures offer services (messaging, mobility, directory);
- Infrastructures are manifested on nodes as pylons;
- Agents run inside nodes, in the context of pylons;
- Agents contain agent shards, which encapsulate agent functionality.

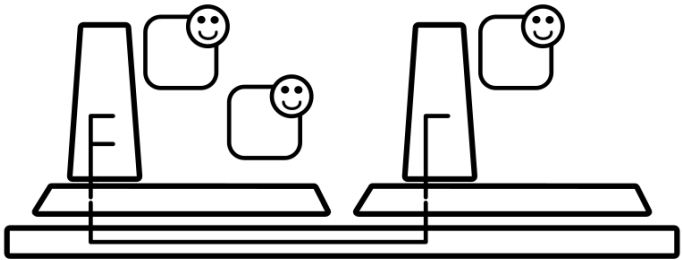


underlying network & machines

underlying network & machines · nodes

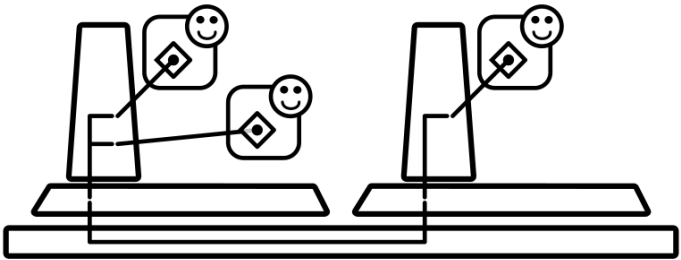


underlying network & machines · nodes · pylons form support infrastructure(s)



underlying network & machines · nodes · pylons form support infrastructure(s)

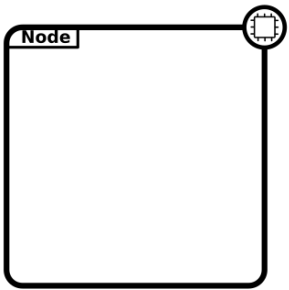
agents run on nodes



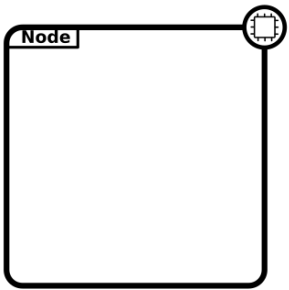
underlying network & machines · nodes · pylons form support infrastructure(s)

agents run on nodes · pylon-specific agent shards enable service use

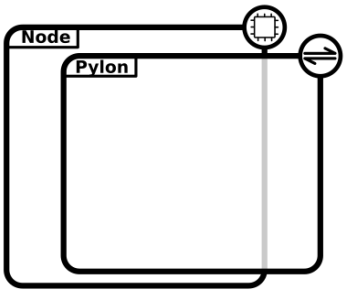
- the system is formed of entities;



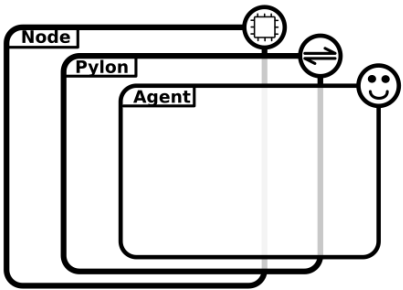
- the system is formed of entities;
- each entity exists in the context of another entity; [the *deployment* is the root entity]
- rules can be configured to control the visibility of an entity as context for nested entities.



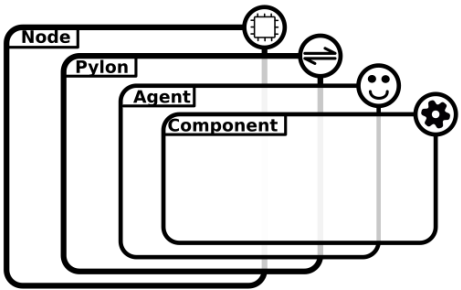
- the system is formed of entities;
- each entity exists in the context of another entity; [the *deployment* is the root entity]
- rules can be configured to control the visibility of an entity as context for nested entities.



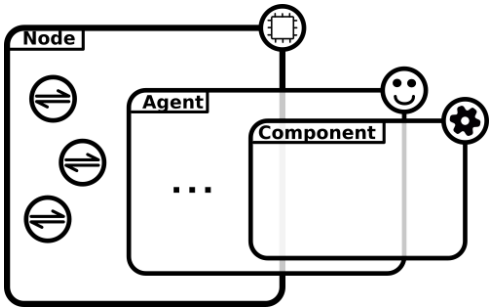
- the system is formed of entities;
- each entity exists in the context of another entity; [the *deployment* is the root entity]
- rules can be configured to control the visibility of an entity as context for nested entities.



- the system is formed of entities;
- each entity exists in the context of another entity; [the *deployment* is the root entity]
- rules can be configured to control the visibility of an entity as context for nested entities.



- the system is formed of entities;
- each entity exists in the context of another entity; [the *deployment* is the root entity]
- rules can be configured to control the visibility of an entity as context for nested entities.



- the structure of the system is flexible;
  - any type of entity can be added to the deployment (e.g. artifacts, groups).
  - the only *required* entities are nodes and agents.

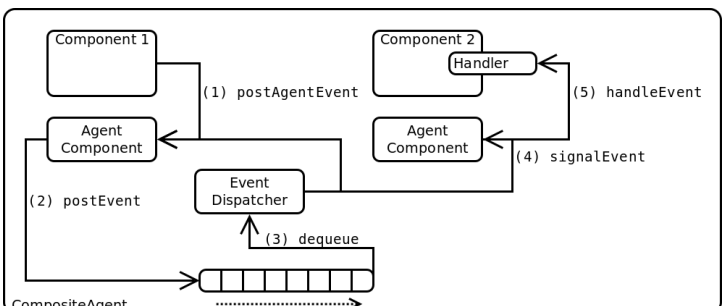
· Example: sending messages without depending on a specific means of communication

- an agent that wishes to communicate contains a messaging shard;
- all messaging shards offer the same API;
- the implementation of the shard is specific to the communication infrastructure used – the agent loads the appropriate implementation, as recommended by its context pylon;
- the shard knows how exactly to interact with the pylon;

· Example: sending messages without depending on a specific means of communication

- an agent that wishes to communicate contains a messaging shard;
- all messaging shards offer the same API;
- the implementation of the shard is specific to the communication infrastructure used – the agent loads the appropriate implementation, as recommended by its context pylon;
- the shard knows how exactly to interact with the pylon;
- each entity may exist within the context of multiple entities simultaneously, e.g. the same agent can use multiple communication infrastructures.

- Composite Agents contain
  - one thread
  - an event queue
  - a set of shards that subscribe to various events



- Sequential Composite Agents are a lighter version of Composite Agents
- all agents run in the same thread, and at each step process a given number of events.

- Shards can be part of any type of agent
- Agents can contain any type of shards

- Shards can be part of any type of agent
  - the agent must be able to accept events posted by the shard;
  - optionally, the agent can provide to the shard a list of the other shards.
- Agents can contain any type of shards

- Shards can be part of any type of agent
  - the agent must be able to accept events posted by the shard;
  - optionally, the agent can provide to the shard a list of the other shards.
- Agents can contain any type of shards
  - the shard provides its designation to the agent, so it can be found by other shards¡
  - the shard should subscribe to agent events.

· The deployment of the system can be configured via two different methods, depending on the desired balance between being fast versus being complex / readable:

- an XML file
- a command-line interface

· Both method describe a hierarchical structures of key-value pairs, with the CLI taking precedence.

· Any of the two methods has the power to completely define the deployment.

```
<?xml version="1.0" encoding="UTF-8"?>
<deployment xmlns="http://flash.xqhs.net/deployment-schema" [...]>
  <package>examples.composite</package>
  <loader for="agent:composite"/>

  <agent name="AgentA" kind="composite">
    <shard name="messaging" />
    <shard name="monitoring" classpath="MonitoringTestFeature" />
  </agent>
  <agent>
    <parameter name="name" value="AgentB" />
    <parameter name="kind" value="composite" />
    <shard name="messaging" />
    <shard name="monitoring" classpath="MonitoringTestFeature" />
  </agent>
</deployment>
```

- A simple deployment

```
flash -package simple -agent agentA -agent agentB
```

- A simple deployment

```
flash -package simple -agent agentA -agent agentB
```

- Deployment of one agent able to communicate and to have knowledge

```
flash
  -loader agent:composite
  -pylon websocket:  server:ws.org
    -agent
      -shard messaging
      -shard knowledge -pair state:initial
```

```
  ...
  <loader for="agent:composite"/>
  <agent name="AgentA" kind="composite">
    <shard name="messaging" />
    <shard name="monitoring" classpath="MonitoringTestFeature" />
  </agent>
  <agent>
    <parameter name="name" value="AgentB" />
    <parameter name="kind" value="composite" />
    <shard name="messaging" />
    <shard name="monitoring" classpath="MonitoringTestFeature" />
  </agent>
</deployment>
```

```
flash deployment.xml
  -package additional.goal
  -agent agentA
    -shard goalOriented
      -goal survive(15, seconds)
  -agent agentB
    -monitoring_server:ws.org
```

- We can have a MAS deployment framework that is fast, flexible and lightweight.
- We aim for building a structure ont which a MAS developer can easily create a MAS with almost any structure and using any services implementation one whishes, with some implementations already given.
- Future work aims to deploy FLASH-MAS onto a varied set of platforms, and provide developers with several pre-implemented support infrastructures, including decentralized means of discovery and communication.

Thank You!

Any Questions?

andrei.olaru@cs.pub.ro