A Framework for Integrating Heterogeneous Agent Communication Platforms

Andrei Olaru and Adina Magda Florea

cs@andreioraru.ro

AI-MAS Group, University Politehnica of Bucharest

23.09.2015
A Framework for Integrating Heterogeneous Agent Communication Platforms

overview
There are currently many MAS development & deployment frameworks to choose from.
- E.g. Jade [Bellifemine et al., 2001], JIAC [Lützenberger et al., 2013], Jason [Bordini et al., 2007], Agent Factory [Russell et al., 2011], or XJAF/Siebog [Mitrović et al., 2014].

Choosing a framework likely means restricting the architecture to:
- a messaging platform (e.g. Jade, JMS, etc);
- an agent architecture (based on goals, behaviors, logic, etc);
- [sometimes] a specific AOP language.

Our target: Create a framework in which agents developed and deployed using different platforms and means of communication are able to co-exist and communicate.
Example usage: Agents communicating through a wireless sensor network, using specific protocols, are able to send messages to mobile devices that use WebSockets to connect to a local server. A device part of the WebSockets platform coordinates a Wireless Body-Area Network.
A Framework for Integrating Heterogeneous Agent Communication Platforms

Motivation
Experience
Elements
Requirements

Introduction

Motivation

CLAIM & SymPa

A. Suna

[CLAIM & SymPa]

Experience

2010

tATAmI-1
S-CLAIM
Jade

2013

tATAmI-2
S-CLAIM
ContextKB
Jade

2015

tATAmI-2.5
multiple platforms

A. Olaru

M.T. Benea

T.T.N. Nguyen

[Olaru et al., 2015]

A. Olaru

M.T. Benea

E. Sevastian

[Olaru, 2015]

A. Olaru

C. Mihai

C.M. Toma

[Baljak et al., 2012,

[tATAmI: towards Agent Technologies for Ambient Intelligence

• The tATAmI project was started together with LIP6

Suna & El Fallah 2004]
An agent is an autonomous entity with various functionality.
An agent component runs inside an agent and implements specific functionality (e.g. messaging).
An agent executes on a machine, or node.
A platform instance executes locally on a node to offer platform-specific services. The platform link enables platform-specific components to offer these services to agents.
A platform spans multiple machines and offers communication, discovery and mobility services to agents, by means of [platform-specific] components.
The tATAmI-2 system (or framework) connects all platforms and agents, across multiple machines.
Communication between platforms is done through **Frontier Agents**, that live on **Frontier nodes**, and are able to communicate through multiple platforms.
· Nodes:
  ▶ System Central (runs System Management)
  ▶ Platform-Central (run Central* agents)
  ▶ Frontier (run Frontier agents)
press one button to deploy all agents.

- specify the minimal set of parameters in an XML file or at the command line. Configure everything on the system-central node.
- only use the command line and a minimal set of parameters on every node that is not the system center.

▸ deploy with ease ←

▸ visualizable

▸ flexible platform services

▸ backwards-compatibility
- **deploy with ease**
  
  the location, status, and execution logs of all agents

- **visualizable**  ← should be visualizable from a single machine (the system center).

- **flexible platform services**

- **backwards-compatibility**
deploy with ease

- agent code* ⊥ messaging/mobility platform;
  *all components except for the messaging component

visualizable

- agent configuration ⊥ platform used;
  (if the messaging component of the agent has no special configuration itself)

- platform services ⊥ agent architecture, limiting the requirements to a platform-specific component that links the agent to the platform.

- component implementation ⊥ implementation of other components.

flexible platform services

backwards-compatibility
deploy with ease

visualizable

flexible platform services

backwards-compatibility ← full compatibility with tATAmI-2 and partial compatibility with tATAmI-1.
in order to use (or to implement) a functionality, an agent uses an agent component (that may be already implemented). Components can be used for communication, mobility, user interface, etc.
### Components Providing Services

- In order to use (or to implement) a functionality, an agent uses an **agent component** (that may be already implemented). Components can be used for communication, mobility, user interface, etc.

- All components offering similar services implement the same **interface**. E.g.:
Components Providing Services

- in order to use (or to implement) a functionality, an agent uses an agent component (that may be already implemented). Components can be used for communication, mobility, user interface, etc.

- all components offering similar services implement the same interface. E.g.:
  - for messaging, components implement the methods to send messages and add handlers for incoming messages;
### Components Providing Services

- In order to use (or to implement) a functionality, an agent uses an **agent component** (that may be already implemented). Components can be used for communication, mobility, user interface, etc.

- All components offering similar services implement the same **interface**. E.g.:
  - For messaging, components implement the methods to send messages and add handlers for incoming messages;
  - For mobility, components implement a method to move the agent to a node with a specified identifier;
in order to use (or to implement) a functionality, an agent uses an agent component (that may be already implemented). Components can be used for communication, mobility, user interface, etc.

all components offering similar services implement the same interface. E.g.:
- for messaging, components implement the methods to send messages and add handlers for incoming messages;
- for mobility, components implement a method to move the agent to a node with a specified identifier;
- for UI, components implement methods for input and output.
Components Providing Services

- in order to use (or to implement) a functionality, an agent uses an agent component (that may be already implemented). Components can be used for communication, mobility, user interface, etc.

- all components offering similar services implement the same interface. E.g.:
  - for messaging, components implement the methods to send messages and add handlers for incoming messages;
  - for mobility, components implement a method to move the agent to a node with a specified identifier;
  - for UI, components implement methods for input and output.

- when an agent is loaded on a platform, the platform gives to the agent a reference – the platform link – that components can use.
in order to use (or to implement) a functionality, an agent uses an agent component (that may be already implemented). Components can be used for communication, mobility, user interface, etc.

all components offering similar services implement the same interface. E.g.:

- for messaging, components implement the methods to send messages and add handlers for incoming messages;
- for mobility, components implement a method to move the agent to a node with a specified identifier;
- for UI, components implement methods for input and output.

when an agent is loaded on a platform, the platform gives to the agent a reference – the platform link – that components can use.

a platform may also recommend a specific implementation for a certain type of component.
· in order to use (or to implement) a functionality, an agent uses an agent component (that may be already implemented). Components can be used for communication, mobility, user interface, etc.

· all components offering similar services implement the same interface. E.g.:
  · for messaging, components implement the methods to send messages and add handlers for incoming messages;
  · for mobility, components implement a method to move the agent to a node with a specified identifier;
  · for UI, components implement methods for input and output.

· when an agent is loaded on a platform, the platform gives to the agent a reference – the platform link – that components can use.

· a platform may also recommend a specific implementation for a certain type of component.

· such a platform-specific component will use the platform link to communicate with the local platform instance in a specific way.
· **Challenge:** enable all agents in the system to communicate, even across platforms, based on name only (or on platform name and agent name).

· **Solution:** use a *System Graph* as a structure that contains information about the relations between nodes.

  ► the System Graph contains Platform-Central nodes, the System Central node, and Frontier Nodes;
  
  ► it is built during the bootstrap process, by System Management;
  
  ► it is disseminated to all platform-central nodes, by means of frontier nodes, which disseminate it to other frontier nodes or to “smart” nodes.
  
· updates will be disseminated when frontier agents are added or removed from the network.
A Framework for Integrating Heterogeneous Agent Communication Platforms

Motivation | Architecture | Routing | Bootstrap | Conclusion

System Graph (2)

· Example:

network topology

system graph
From a messaging perspective, a platform can be:

- “silent” (for backwards compatibility) – deployed using tATAmI-2, but contain Frontier Agents; the implementation of the platform remains unchanged. Thanks to Frontier Agents, they will receive messages from the rest of the system, but they are not adapted to send messages to other platforms (they don’t understand the concept);

- “silly” – tATAmI-2-based implementation, which doesn’t use the System Graph. Messages to unknown destinations go to the Central* agent, which sends them in turn to a Frontier Agent on the path to the central node of the other platform;

- “smart” – the platform is able to route messages to various Frontier Agents, using the System Graph received from System Management. These platforms may contain “smart” nodes, which are also able to understand the System Graph and use it to route messages without the help of the Central* agent.
The bootstrap process must ensure that all agents are connected to their platforms and that all platforms are connected in the system.

There are 4 phases:

1. on each platform the Platform-Central agent gathers information from the Frontier Agents in the platform;

2. System Management disseminates its platform identifier to everybody else;

3. all Central* agents send information about their platform to System Management;

4. System Management sends the System Graph to all the platforms, which disseminate it internally.

* Phase 1 can happen simultaneously with phases 2-4.
Booting using a minimal number of command-line arguments:

- scenario file is always the first argument (if any)
- `-iscentral [main-platform-id]` ← is this the System Central node
- `-center IP port other...` ← central node to connect to
- `-here node IP port other...` ← local node info
- `-platformID type settings...` ← name & settings for the platform
- `-platformType settings...` ← settings for the platform
- `-wh width height` ← other local parameters
Booting using a minimal number of command-line arguments:

- scenario file is always the first argument (if any)
- `-iscentral [main-platform-id]` ← is this the System Central node
- `-center IP port other...` ← central node to connect to
- `-here node IP port other...` ← local node info
- `-platformID type settings...` ← name & settings for the platform
- `-platformType settings...` ← settings for the platform
- `-wh width height` ← other local parameters

Examples:

```
tATAmI scenario.xml -center <IP1> -here <IP2>
```
Booting using a minimal number of command-line arguments:

- scenario file is always the first argument (if any)
- `-iscentral [main-platform-id] ← is this the System Central node`
- `-center IP port other... ← central node to connect to`
- `-here node IP port other... ← local node info`
- `-platformID type settings... ← name & settings for the platform`
- `-platformType settings... ← settings for the platform`
- `-wh width height ← other local parameters`

Examples:

```
tATAmI scenario.xml -center <IP1> -here <IP2>
tATAmI -websockets -center <IP1> -here Node2
```
Booting using a minimal number of command-line arguments:

- scenario file is always the first argument (if any)
- `-iscentral [main-platform-id]` ← is this the System Central node
- `-center IP port other...` ← central node to connect to
- `-here node IP port other...` ← local node info
- `-platformID type settings...` ← name & settings for the platform
- `-platformType settings...` ← settings for the platform
- `-wh width height` ← other local parameters

Examples:

```
  tATAmI scenario.xml -center <IP1> -here <IP2>
  tATAmI -websockets -center <IP1> -here Node2
  tATAmI -jade1 jade showGUI -here centralNode <IP>
```
Scenario XML file – configures platforms, agents and execution:

```xml
<scen:platform>
<scen:parameter name="name" value="jade1" />
<scen:parameter name="type" value="jade" />
<scen:parameter name="GUI" value="true" />
<scen:parameter name="localIP" value="<IP>" />
<scen:parameter name="mainContainer" value="centralNode" />
</scen:platform>

<scen:initial>
<scen:container name="centralNode">
<scen:agent>
<scen:component name="visualizable" />
<scen:component name="messaging" />
<scen:parameter name="loader" value="composite" />
<scen:parameter name="name" value="AgentA" />
</scen:agent>
</scen:container>
</scen:initial>

– is equivalent to tATAmI -jade1 jade showGUI -here centralNode <IP>
```
In order to avoid the long-term effects of an initial choice of agent framework and messaging platform, it is useful to have a method of deploying agents over multiple platforms in the same system.

The tATAmI-2.5 architecture was presented that supports this, in which agents are able to be deployed on various platforms with no changes. The main features of this architecture are a routing method and a bootstrap process.

Some changes must be introduced in tATAmI-2 in order to support multiple messaging components per agent, and the improved bootstrap process.

Testing must be performed to evaluate performance in a strongly heterogeneous setup.
Thank You!

Any Questions?

cs@andreiolaru.ro
A Framework for Integrating Heterogeneous Agent Communication Platforms

S-CLAIM: An agent-based programming language for AmI, a smart-room case study.

Developing multi-agent systems with JADE.
Intelligent Agents VII Agent Theories Architectures and Languages, pages 42–47.

John Wiley & Sons.

JIAC V: A MAS framework for industrial applications.

Extensible java ee-based agent framework in clustered environments.

tATAmI-2 – a flexible framework for modular agents.
In Dumitrache, I., Florea, A. M., Pop, F., and Dumitrescu, A., editors, Proceedings of AgTAmI 2015, the International Workshop on Agent Technology for Ambient Intelligence, the 20th International Conference on Control Systems and Computer Science, May 27-29, Bucharest, Romania, volume 2, pages 703–710. IEEE Computer Society.


Thank You!

Any Questions?

cs@andreiorlaru.ro