A Framework for Integrating Heterogeneous Agent Communication Platforms

Andrei Olaru and Adina Magda Florea

cs@andreiolaru.ro

AI-MAS Group, University Politehnica of Bucharest

23.09.2015

- Motivation
- Architecture
- Routing
- Bootstrap
- Conclusion

A Framework for Integrating Heterogeneous Agent Communication Platforms

overview

AI-MAS Group

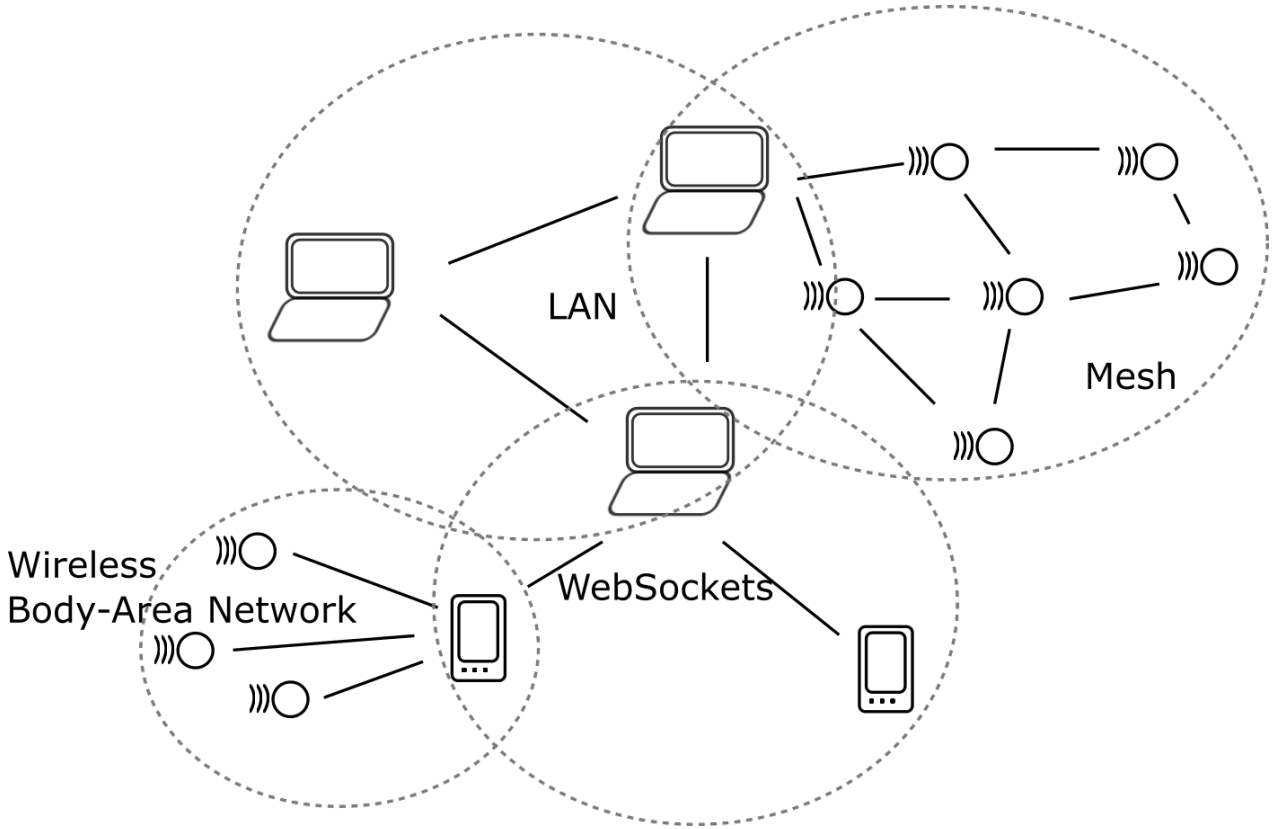
. Andrei Olaru and Adina Magda Florea
. cs@andreiolaru.ro

- 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.

AI-MAS Group
. Andrei Olaru and Adina Magda Florea
. cs@andreiolaru.ro
. ACSys Workshop | SYNASC 2015, Timişoara, Romania 23.09.2015



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.

Introduction

CLAIM & SymPa

A. Suna

[Suna & El Fallah 2004]

2010

tATAmI-1
S-CLAIM
Jade

 A. Olaru

 M.T. Benea

 T.T.N. Nguyen

[Baljak et al., 2012,

Olaru et al., 2015]

2013

tATAmI-2
S-CLAIM
ContextKB
Jade

 A. Olaru

 M.T. Benea

 E. Sevastian

[Olaru, 2015]

2015

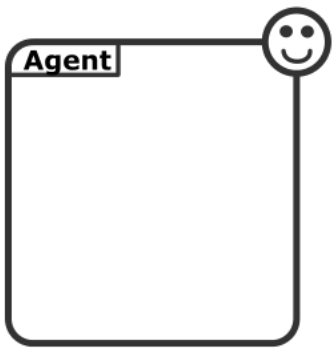
tATAmI-2.5
multiple
platforms

 A. Olaru

 C. Mihai

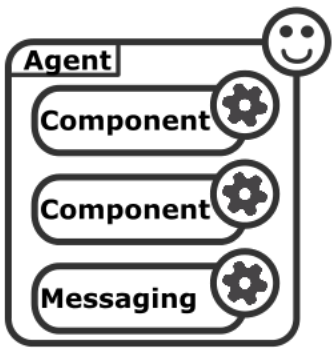
 C.M. Toma

- tATAmI: towards Agent Technologies for Ambient Intelligence
- The tATAmI project was started together with LIP6

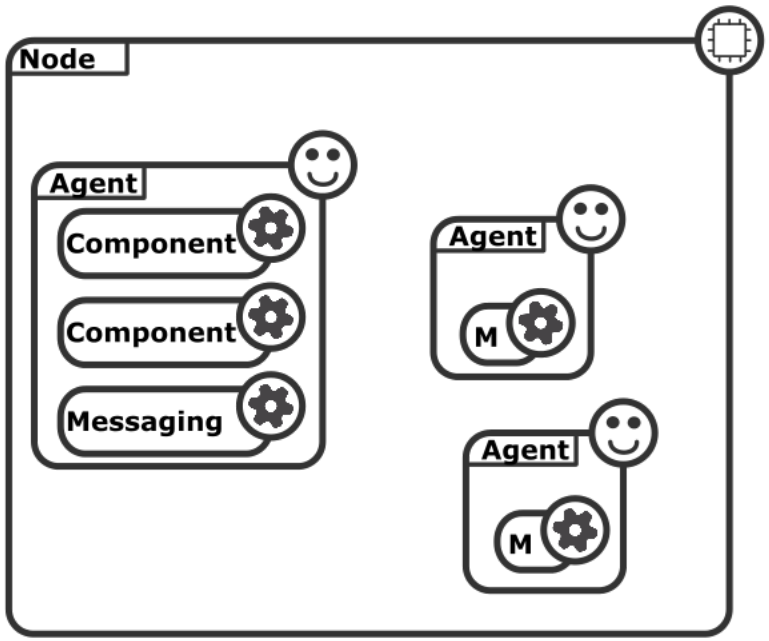


An agent is an autonomous entity with various functionality.

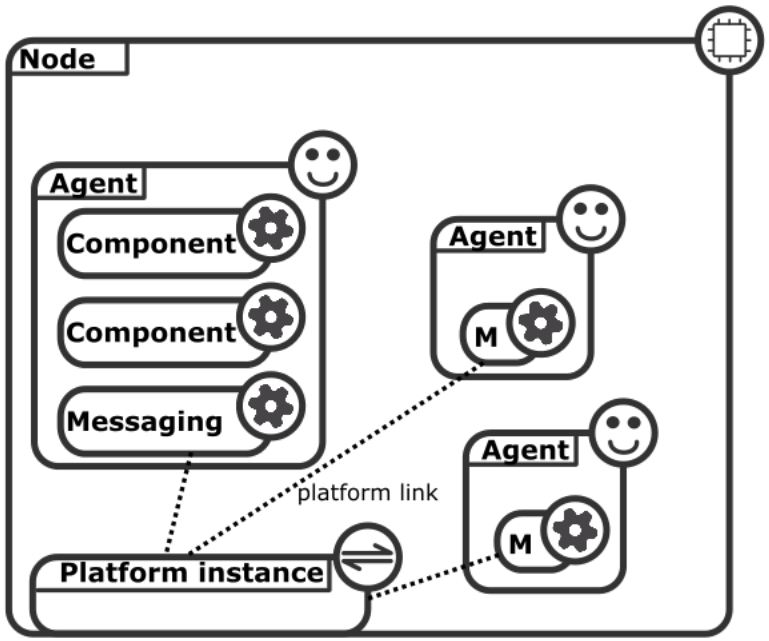
AI-MAS Group
. Andrei Olaru and Adina Magda Florea
. cs@andreiolaru.ro
. ACSys Workshop | SYNASC 2015, Timişoara, Romania 23.09.2015



An agent component runs inside an agent and implements specific functionality (e.g. messaging).

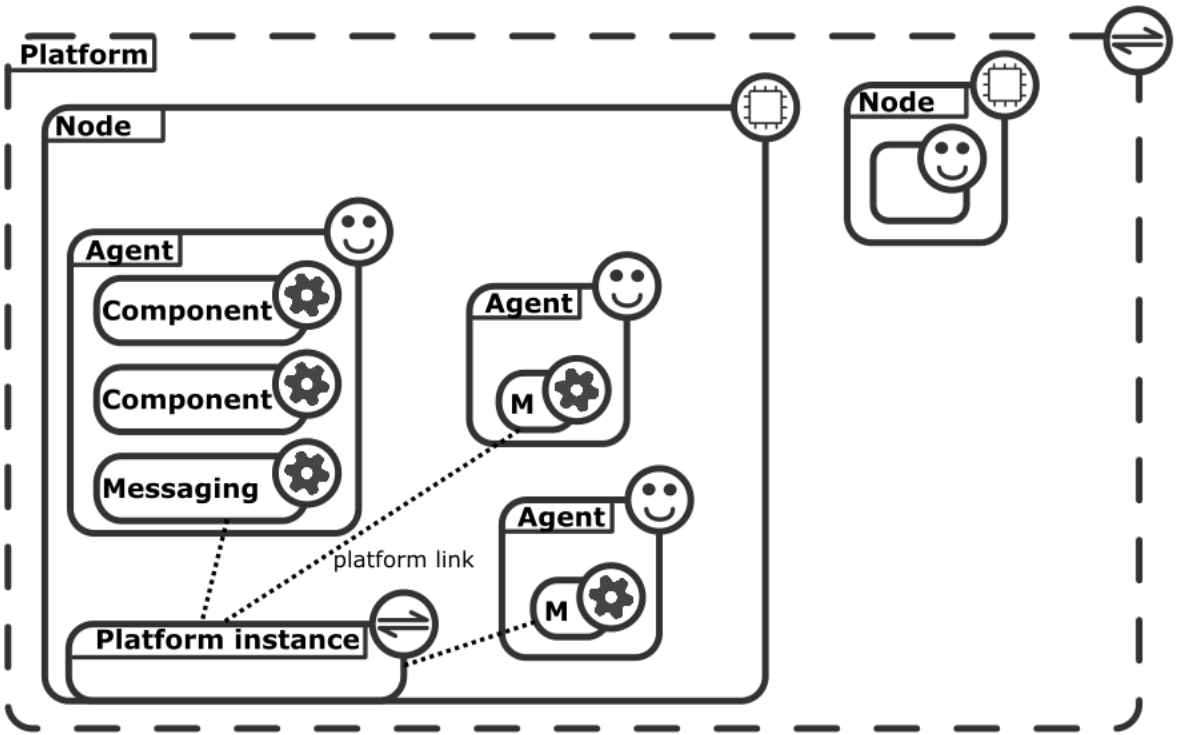


An agent executes on a machine, or node.



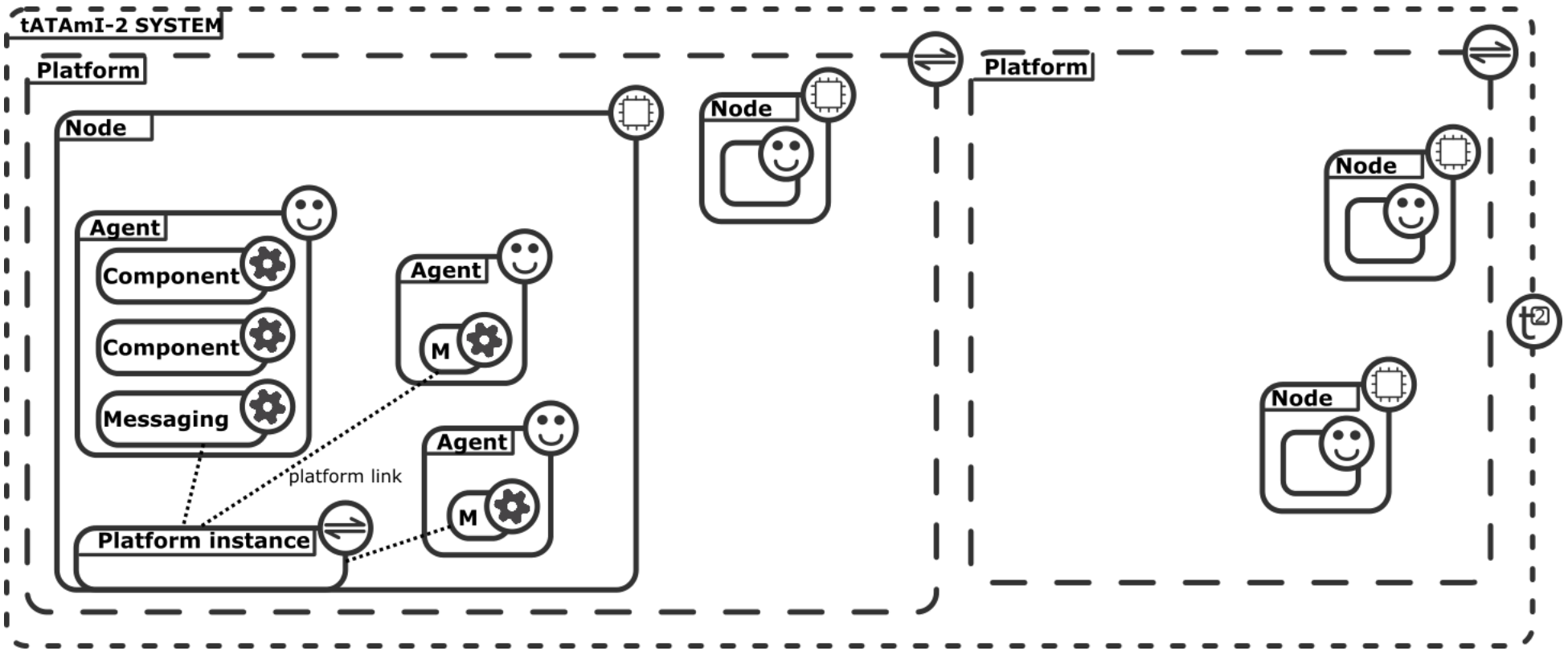
A platform instance executes locally on a node to offer platform-specific services. The platform link enables platform-specific components to offer these services to agents.

AI-MAS Group
. Andrei Olaru and Adina Magda Florea
. cs@andreiolaru.ro
. ACSys Workshop | SYNASC 2015, Timişoara, Romania 23.09.2015



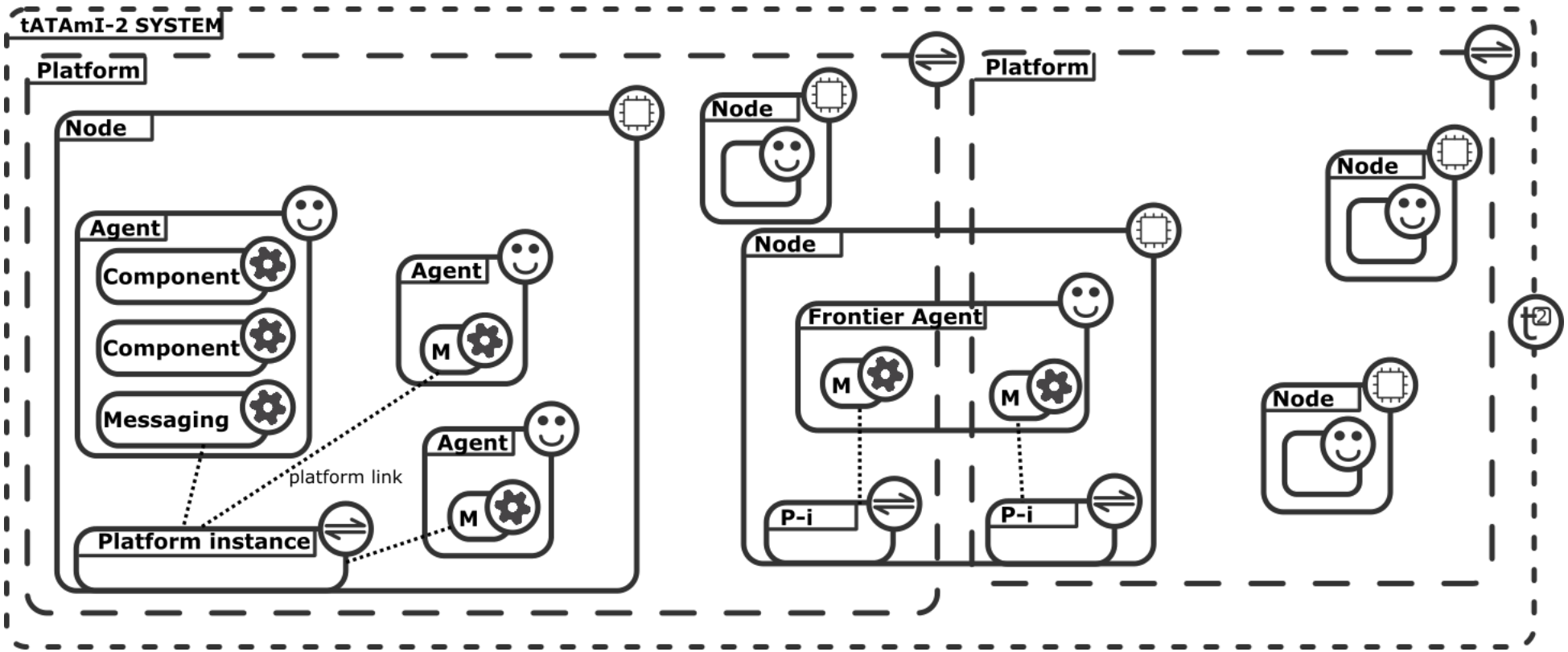
A platform spans multiple machines and offers communication, discovery and mobility services to agents, by means of [platform-specific] components.

AI-MAS Group
. Andrei Olaru and Adina Magda Florea
. cs@andreiolaru.ro
. ACSys Workshop | SYNASC 2015, Timişoara, Romania 23.09.2015



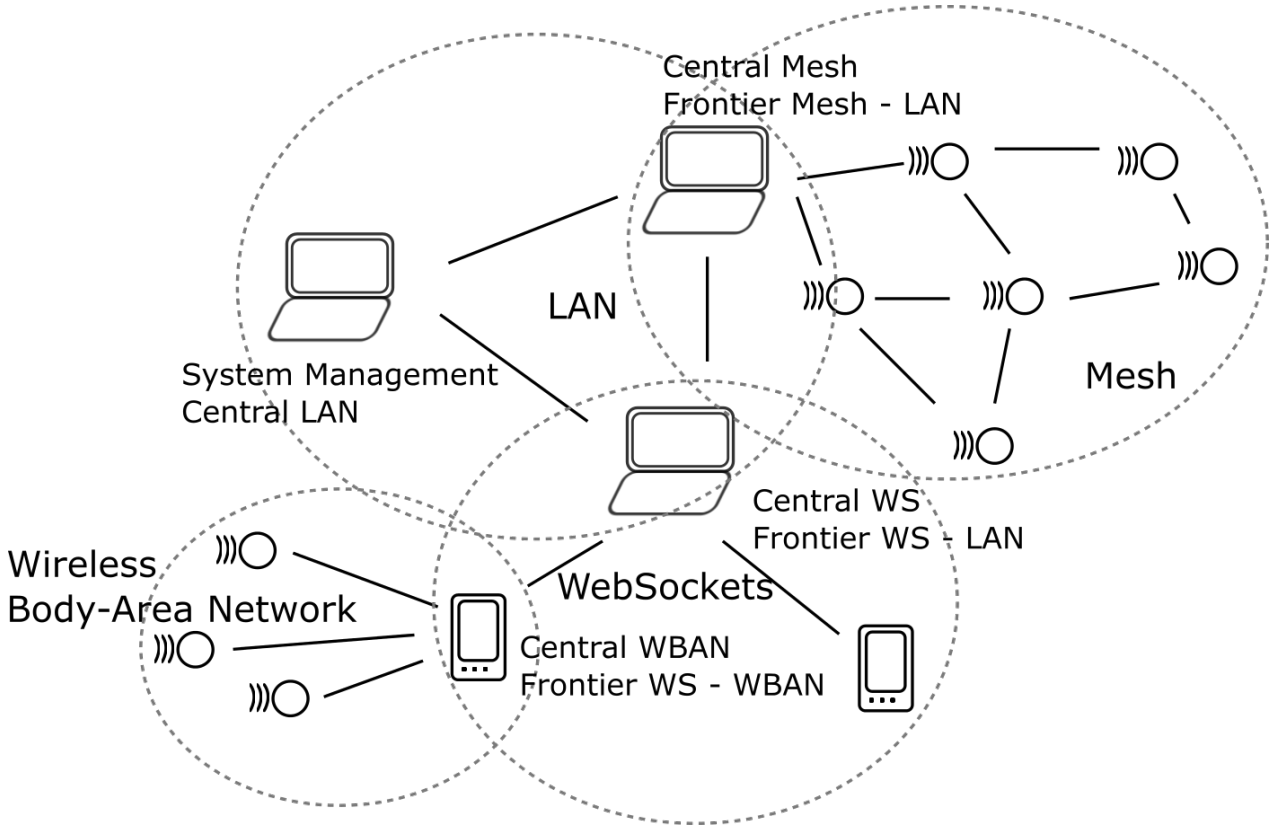
The tATAmI-2 system (or framework) connects all platforms and agents, across multiple machines.

AI-MAS Group
. Andrei Olaru and Adina Magda Florea
. cs@andreiolaru.ro
. ACSys Workshop | SYNASC 2015, Timişoara, Romania 23.09.2015



Communication between platforms is done through Frontier Agents, that live on Frontier nodes, and are able to communicate through multiple platforms.

AI-MAS Group
. Andrei Olaru and Adina Magda Florea
. cs@andreiolaru.ro
. ACSys Workshop | SYNASC 2015, Timişoara, Romania 23.09.2015



- Nodes:
  - System Central (runs System Management)
  - Platform-Central (run Central* agents)
  - Frontier (run Frontier agents)

AI-MAS Group
. Andrei Olaru and Adina Magda Florea
. cs@andreiolaru.ro
. ACSys Workshop | SYNASC 2015, Timişoara, Romania 23.09.2015

press one button to deploy all agents.

- deploy with ease ←
  - 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.
- visualizable
- flexible platform services
- backwards-compatibility

- deploy with ease
- visualizable ← the location, status, and execution logs of all agents should be visualizable from a single machine (the system center).
- flexible platform services
- backwards-compatibility

Introduction

- deploy with ease
- visualizable
- flexible platform services ←
  - agent code* ⊥ messaging/mobility platform;
    *all components except for the messaging component
  - 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.

$\perp \equiv$ "independent of"

- backwards-compatibility

AI-MAS Group
. Andrei Olaru and Adina Magda Florea
. cs@andreiolaru.ro
. ACSys Workshop | SYNASC 2015, Timişoara, Romania 23.09.2015

- 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.

- 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.:

- 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;

- 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.

- 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.

Architectural Principle

- 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.

Message Routing

System Graph (1)

· Challenge: enable all agents in the system to communicate, even across platforms, based on name only (or on platform name and agent name).

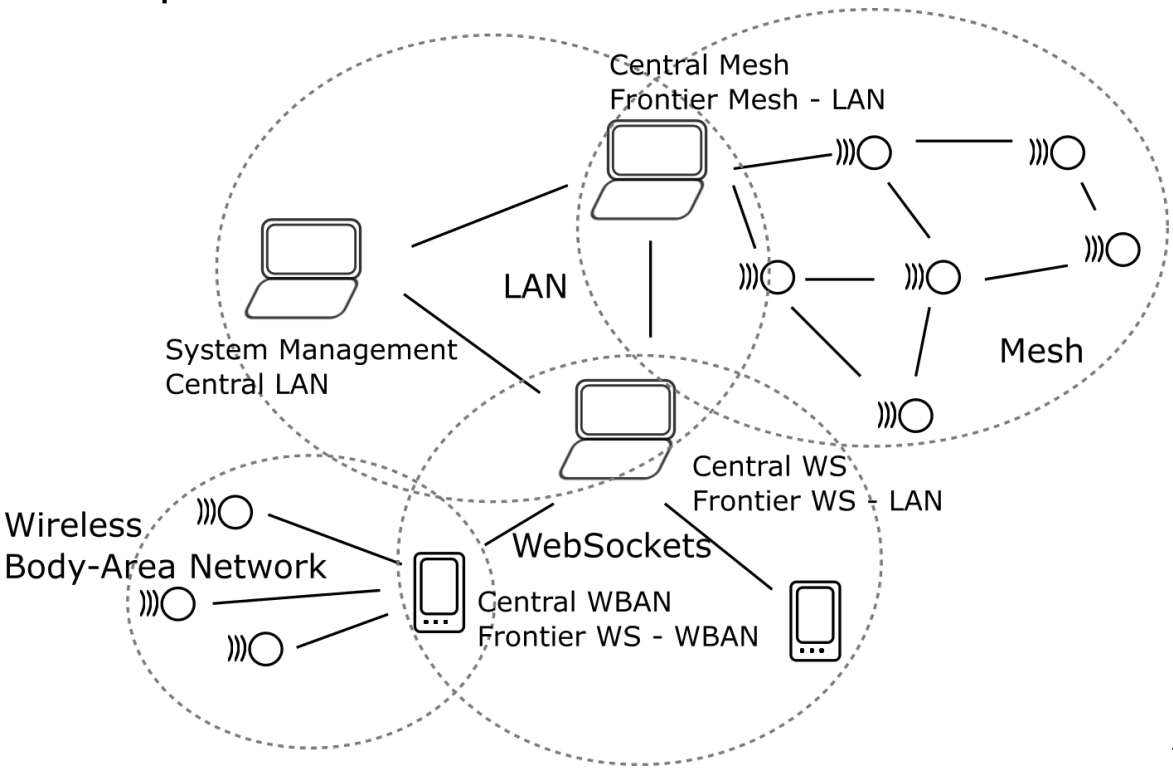
· Solution: use a *System Graph* as a structure that contains information about the relations between nodes.

- 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.

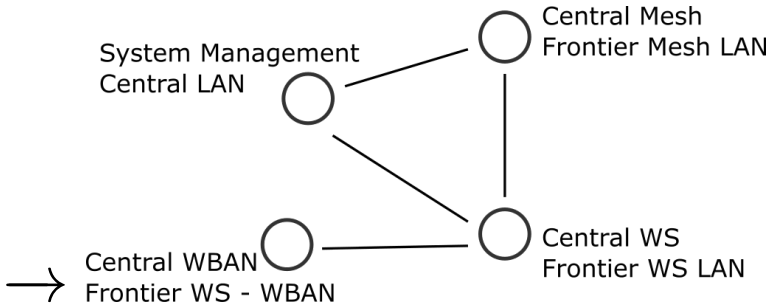
System Graph (2)

Message Routing

- Example:



network topology



system graph

AI-MAS Group
. Andrei Olaru and Adina Magda Florea
. cs@andreiolaru.ro
. ACSys Workshop | SYNASC 2015, Timişoara, Romania 23.09.2015

System Graph (3)

Message Routing

- 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.

Bootstrap

· 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:

```
<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>`

AI-MAS Group
. Andrei Olaru and Adina Magda Florea
. cs@andreiolaru.ro
. ACSys Workshop | SYNASC 2015, Timişoara, Romania 23.09.2015

Conclusion & Future Work

- 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

Baljak, V., Benea, M. T., El Fallah Seghrouchni, A., Herpson, C., Honiden, S., Nguyen, T. T. N., Olaru, A., Shimizu, R., Tei, K., and Toriumi, S. (2012).
S-CLAIM: An agent-based programming language for AmI, a smart-room case study.
In Proceedings of ANT 2012, The 3rd International Conference on Ambient Systems, Networks and Technologies, August 27-29, Niagara Falls, Ontario, Canada, volume 10 of Procedia Computer Science, pages 30–37. Elsevier.

Bellifemine, F., Poggi, A., and Rimassa, G. (2001).
Developing multi-agent systems with JADE.
Intelligent Agents VII Agent Theories Architectures and Languages, pages 42–47.

Bordini, R. H., Hübner, J. F., and Wooldridge, M. (2007).
Programming multi-agent systems in AgentSpeak using Jason, volume 8.
John Wiley & Sons.

Lützenberger, M., Küster, T., Konnerth, T., Thiele, A., Masuch, N., Heßler, A., Keiser, J., Burkhardt, M., Kaiser, S., and Albayrak, S. (2013).
JIAC V: A MAS framework for industrial applications.
In Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems, pages 1189–1190. International Foundation for Autonomous Agents and Multiagent Systems.

Mitrović, D., Ivanović, M., Vidaković, M., and Budimac, Z. (2014).
Extensible java ee-based agent framework in clustered environments.
In Multiagent System Technologies, pages 202–215. Springer.

Olaru, A. (2015).
tATAmI-2 – a flexible framework for modular agents.
In Dumitrache, I., Florea, A. M., Pop, F., and Dumitrascu, 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.

Olaru, A., Benea, M.-T., El Fallah Seghrouchni, A., and Florea, A. M. (2015).

tATAmI: A platform for the development and deployment of agent-based ami applications.
In Shakshuki, E., editor, Proceedings of ANT-2015, the 6th International Conference on Ambient Systems, Networks and Technologies, June 2-5, London, United Kingdom, volume 52 of Procedia Computer Science, pages 476–483. Elsevier.

Russell, S., Jordan, H., O'Hare, G. M., and Collier, R. W. (2011).

Agent factory: a framework for prototyping logic-based AOP languages.
In Multiagent System Technologies, pages 125–136. Springer.

Suna, A. and El Fallah Seghrouchni, A. (2004).

Programming mobile intelligent agents: An operational semantics.
Web Intelligence and Agent Systems, 5(1):47–67.

Thank You!

Any Questions?

cs@andreiolaru.ro