S-CLAIM: An Agent-Based Programming Language for AmI, A Smart-Room Case Study

Valentina Baljak[b], **Marius-Tudor Benea[a,c]**, Amal El Fallah Seghrouchni[a], Cédric Herpson[a], Shinichi Honiden[b], Thi Thuy Nga Nguyen[a], Andrei Olaru[c], Ryo Shimizu[b], Kenji Tei[b], Susumu Toriumi[b]

[a]LIP6 - University Pierre and Marie Curie (UPMC), France
[b]NII - National Institute of Informatics, Japan
[c]University Politehnica of Bucharest (UPB), Romania

August 27, 2012

Outline

Introduction and Problem Statement

Ambient Intelligence (AmI) is the vision of a future ubiquitous electronic environment that supports people in their daily tasks, in a proactive and context-aware, but "invisible" and non-intrusive manner. Ramos et al., 2008, Ducatel et al., 2001

AmI applications – characterized by:

- intrinsic distribution of the architecture;
- dynamic topology;
- frequent changes in execution context $\Rightarrow$ context sensitivity is a key element of AmI applications.

Therefore, an agent-oriented approach for AmI becomes a good choice.

Introduction and Problem Statement

The problem: A better agent-oriented programming language is needed for the development of AmI applications. This language should:

- allow representation of cognitive elements (goals, knowledge, capabilities);
- support mobile computation and execution in smart environments;
- offer a good solution to achieve context-sensitivity.

State Of the Art

- Agent-Oriented Programming (AOP) languages: AgentSpeak, 3APL;
  - Advantages: allow development of intelligent agents;
  - Disadvantages: do not support mobility for the agents.
- Concurrent languages: Ambient calculus – Cardelli et al (2000);
  - Advantages: formalize concurrent and mobile processes in distributed environments;
  - Disadvantages: impossible to represent intelligent agents.

State Of the Art

- Agent-Oriented Programming (AOP) languages: AgentSpeak, 3APL;
  - Advantages: allow development of intelligent agents;
  - Disadvantages: do not support mobility for the agents.
- Concurrent languages: Ambient calculus – Cardelli et al (2000);
  - Advantages: formalize concurrent and mobile processes in distributed environments;
  - Disadvantages: impossible to represent intelligent agents.
- CLAIM – Suna and El Fallah Seghrouchni (2004).
  - Advantages: combines in a unified framework the main advantages of AOP languages with those of the concurrent languages;
  - Drawbacks: a complex, difficult to follow syntax, an application layer that needed many resources to execute and no possibility to deploy applications on heterogeneous device networks.

Our Approach – S-CLAIM

S-CLAIM (Smart Computational Language for Autonomous, Intelligent and Mobile agents):

- Combines the advantages of the CLAIM language, like:
  - Cognitive elements – knowledge, goals and capabilities;
  - Mobility primitives – inspired from ambient calculus;
  - Hierarchical organization of agents – offers a natural solution to achieve context-sensitivity.
- With a series of new features and improvements to the existing ones:
  - A simplified and easier-to-follow syntax – Lisp-like;
  - A simplified semantics – focused on agent-specific functionality;
  - All algorithmic functionality is exported to external implementations (implemented, for instance, in Java);
  - Supports various representations of the KB (representable by relations);
  - Allows deployment of applications on heterogeneous device networks, including devices with limited resources, like mobile devices.

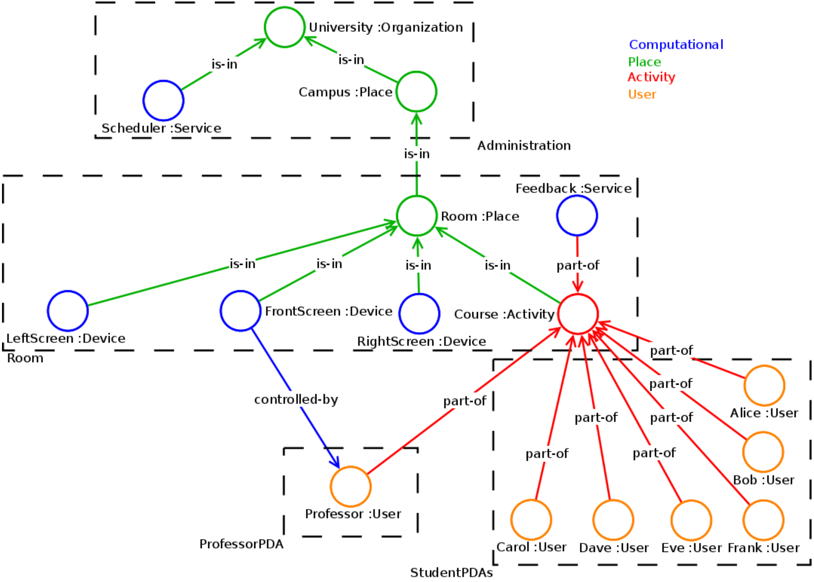
Outline

Scenario – Syamisen

- Alice is informed that the room for the CS course that she attends has changed;
- At the hour set for the course, the professor is in the room;
- Based on a global situation of the students, available on his PDA, he decides to start the course;
- The room is configured for the presentation and the presentation begins;
- After the course the students are involved in some hands-on activities;
- After a pre-established interval of time, the teacher evaluates the results of the activities;
- The course ends and everything turns off;
- The students leave feedback when the Feedback agent comes to their PDAs in order to ask for it.

Smart Room in NII, Japan

Agentification of the scenario



Outline

Agent behaviors in S-CLAIM

Behaviors – one of the most important parts of an agent. They define what an agent can do in certain situations. In S-CLAIM, the behavior types are the following ones:

- Initial: triggered at agent creation;
- Reactive: triggered by the reception of messages;
- Cyclic: infinitely repeating;
- Proactive (developed by Simons and Garella from Delft University): uses the following goal types:
  - perform;
  - achieve;
  - maintain.

Vocabulary and Syntax

Example (Agent Class Definition)

```
(agent Course ?courseName ?parent
    (behavior
        ...




    )
)
```

Vocabulary and Syntax

Example (Agent Class Definition)

```
(agent Course ?courseName ?parent
    (behavior
        ...

        (initial register
            (send ?parent (struct message managesCourse this ?courseName))
        )

        ...
    )
)
```

Vocabulary and Syntax

Example (Agent Class Definition)

```
(agent Course ?courseName ?parent
    (behavior
        ...
        (reactive changeRoom /*reacts to a message that informs about the new room*/
            (receive scheduling ?courseName ?roomName)
            (addK (struct knowledge scheduling ?courseName ?roomName))
            (if (readK (struct knowledge roomAgent ?roomName ?roomAgentName))
                then
                    (forAllK (struct knowledge userAgent ?userName ?userAgentName)
                        (send ?userAgentName (struct message scheduling ?courseName
                            ?roomAgentName))
                    )
                    (in ?roomAgentName)
                else
                    (send ?parent (struct message whoManagesRoom this ?roomName))
            )
        )
        ...
    )
)
```

Vocabulary and Syntax

Example (Agent Class Definition)

```
(agent Course ?courseName ?parent
    (behavior
        ...

        (cyclic verifyStartingCondition
            (condition (not (readK (struct knowledge courseStarted))))
            ... // assign values to ?studentsInRoom, ?minNoOfStudents
                        // and ?professorAgent based on the KB
            (if (greaterOrEqual ?studentsInRoom ?minNoOfStudents)
                then
                    (calculatePercent ?result ?studentsInRoom ?minNoOfStudents)
                    /*the professor is informed that the course can start*/
                    (send ?professorAgent (struct message presentStudents ?result))
            )
            (wait 60000)
        )

        ...
    )
)
```

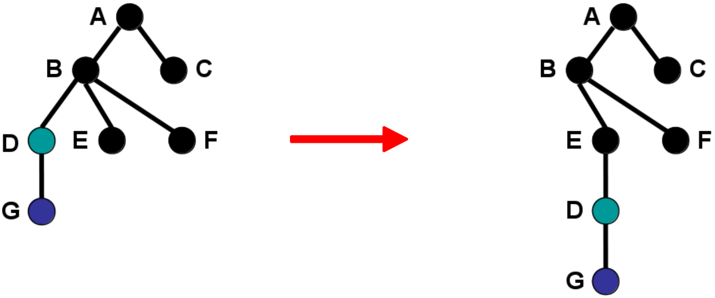
Semantics and Agent Hierarchies

S-CLAIM primitives	
Messaging primitives	Control primitives
Mobility primitives	Agent management primitives
Knowledge management primitives	Goal-oriented primitives

- Messaging primitives: *send*, *receive*;
- Mobility primitives: *in*, *out*;
- Knowledge management primitives: *addK*, *removeK*, *readK*, *forAllK*;
- Agent management primitives: *open*, *acid*, *new*;
- Control primitives: *condition*, *if*, *wait*, *while*;
- Goal-oriented primitives: *aGoal*, *pGoal*, *mGoal*;

Semantics and Agent Hierarchies

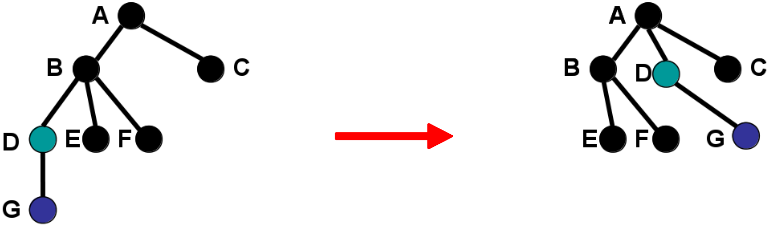
S-CLAIM primitives	
Messaging primitives	Control primitives
Mobility primitives	Agent management primitives
Knowledge management primitives	Goal-oriented primitives



D.in(E)

Semantics and Agent Hierarchies

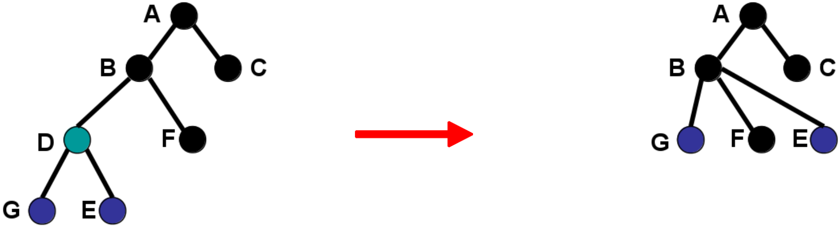
S-CLAIM primitives	
Messaging primitives	Control primitives
Mobility primitives	Agent management primitives
Knowledge management primitives	Goal-oriented primitives



D.out(B)

Semantics and Agent Hierarchies

S-CLAIM primitives	
Messaging primitives	Control primitives
Mobility primitives	Agent management primitives
Knowledge management primitives	Goal-oriented primitives



B.open(D), D.acid

Semantics – Web Services

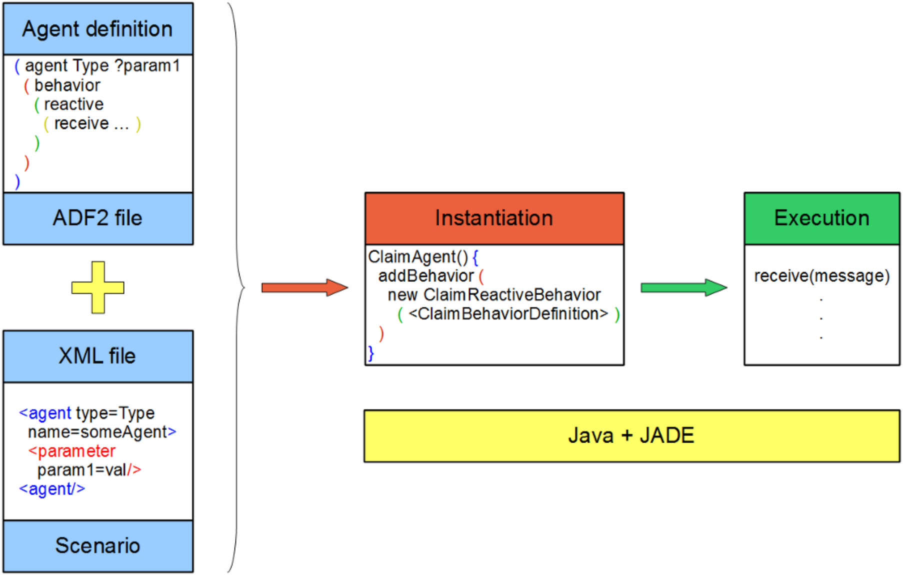
Web services – strong feature that facilitates the deployment of applications on heterogeneous device networks.

- They are integrated in the existing language constructs – modified messaging primitives;
- The reactive behaviors of the agents are exposed as web services (using WSIG);
- Agents could invoke web services, just as they would send messages (thanks to WSDC).

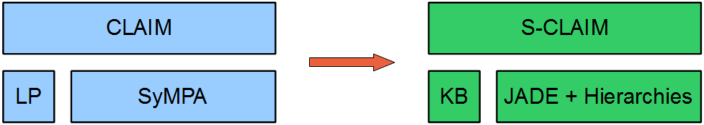
Example (Web Services – adapted *send* primitive)

```
(
   send ?service (struct message echo)
          http://localhost/wsig/ws/
          (struct message ?back)
)
```

Outline


Agent definition
(agent Type ?param1
(behavior
(reactive
(receive ...)
)
)
)
ADF2 file
XML file
<agent type=Type
name=someAgent>
<parameter
param1=val/>
<agent/>
Scenario
Instantiation
ClaimAgent() {
addBehavior (
new ClaimReactiveBehavior
(<ClaimBehaviorDefinition>)
)
}
Execution
receive(message)
.
.
.
Java + JADE

Improvements to the platform



- Based on JADE (Java Agent Development Framework): More stable and higher degree of interoperability with other platforms (like Android);
- Support for Android mobile devices;
- Scenario read from an XML file;
- Easier to extend (modularity);
- Web services support (WSIG and WSDL JADE add-ons);
- A centralized logging system for all the agents;
- Various possible implementations of the agent's knowledge base (CLAIM supported only propositional logic).

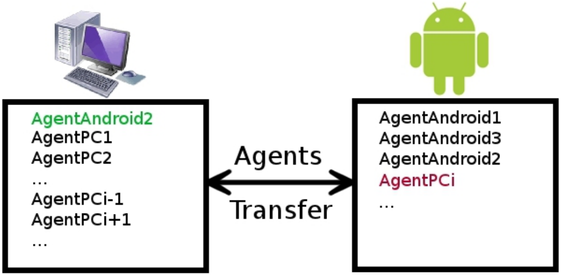
Android support

- Reasons to consider a mobile platform:
  - Very useful in developing AmI applications:
    a) Could be used as an interface with the user;
    b) Newer, powerful, devices could run complex user assistant agents;
    c) Valuable data provided by the features of the smartphones or tablets.

Android support

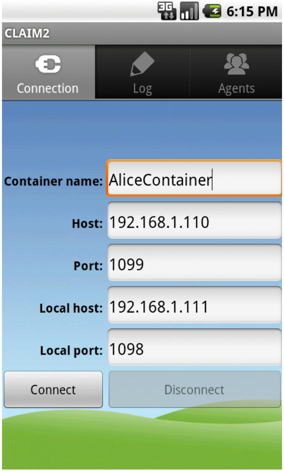
- Reasons to consider a mobile platform:
  - Very useful in developing AmI applications:
    a) Could be used as an interface with the user;
    b) Newer, powerful, devices could run complex user assistant agents;
    c) Valuable data provided by the features of the smartphones or tablets.
- Reasons to consider Android :
  - World's bestselling Smartphone platform;
  - Easy to:
    a) Access the core functionality of the Android devices;
    b) Interact with the OS;
    c) Control the hardware;
  - Applications constructed from components – easy to integrate already developed components from other applications;
  - Supported by JADE .

Android support



Agent migration *PC* $\longleftrightarrow$ *Android*

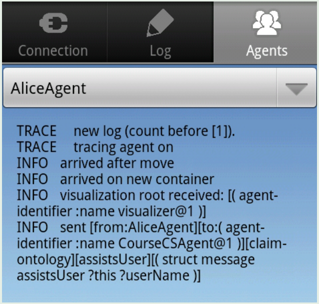
- Based on JadeAndroid add-on for JADE;
- Agents could move to / from any types of devices supported by the system.



Screenshot of S-CLAIM Android

Android support

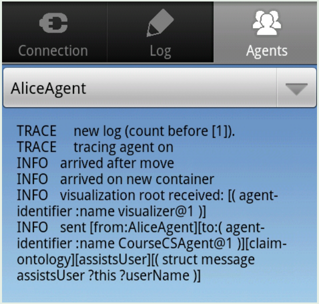
Example (Migration *PC* $\longleftrightarrow$ *Android* of an agent)



Android support

Example (Migration *PC* ⟷ *Android* of an agent)

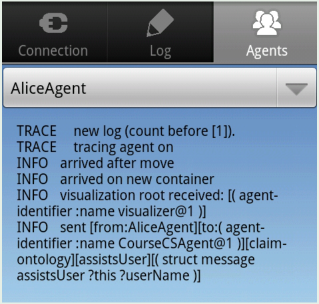
```
<scen:timeline>
  <scen:event time="2000" >
    <scen:CLAIMMessage>
      <scen:to>SchedulerUPMCAgent</scen:to>
      <scen:protocol>newSchedule</scen:protocol>
      <scen:content>
      ( struct message newSchedule
        ( struct knowledge scheduledTo CSCourse Room04 )
      )
      </scen:content>
    </scen:CLAIMMessage></scen:event>
</scen:timeline>
```



Android support

Example (Migration *PC* ⟷ *Android* of an agent)

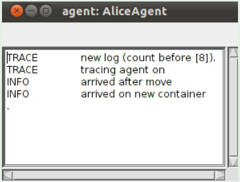
```
<scen:timeline>
  <scen:event time="2000" >
    <scen:CLAIMMessage>
      <scen:to>SchedulerUPMCAgent</scen:to>
      <scen:protocol>newSchedule</scen:protocol>
      <scen:content>
      ( struct message newSchedule
        ( struct knowledge scheduledTo CSCourse Room04 )
      )
      </scen:content>
    </scen:CLAIMMessage></scen:event>
</scen:timeline>
```

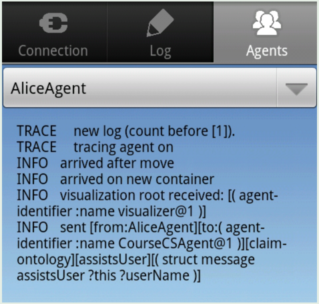



```
20:39:05:0624 INFO  [AliceAgent]:   sent [from:AliceAgent][to:( agent-identifier :name CourseCSAgent@1 )]
20:39:13:0767 INFO  [AliceAgent]:   AliceAgent must migrate to RoomContainer
20:39:13:0773 INFO  [AliceAgent]:   moving to [RoomContainer]
20:39:13:0777 INFO  [AliceAgent]:   moving to [RoomContainer@<Unknown Host>]
20:39:13:0783 TRACE [AliceAgent]:   log out (logs remaining [0]).
```

Example (Migration *PC* $\longleftrightarrow$ *Android* of an agent)

```
<scen:timeline>
  <scen:event time="2000" >
    <scen:CLAIMMessage>
      <scen:to>SchedulerUPMCAgent</scen:to>
      <scen:protocol>newSchedule</scen:protocol>
      <scen:content>
      ( struct message newSchedule
        ( struct knowledge scheduledTo CSCourse Room04 )
      )
      </scen:content>
    </scen:CLAIMMessage></scen:event>
</scen:timeline>
```

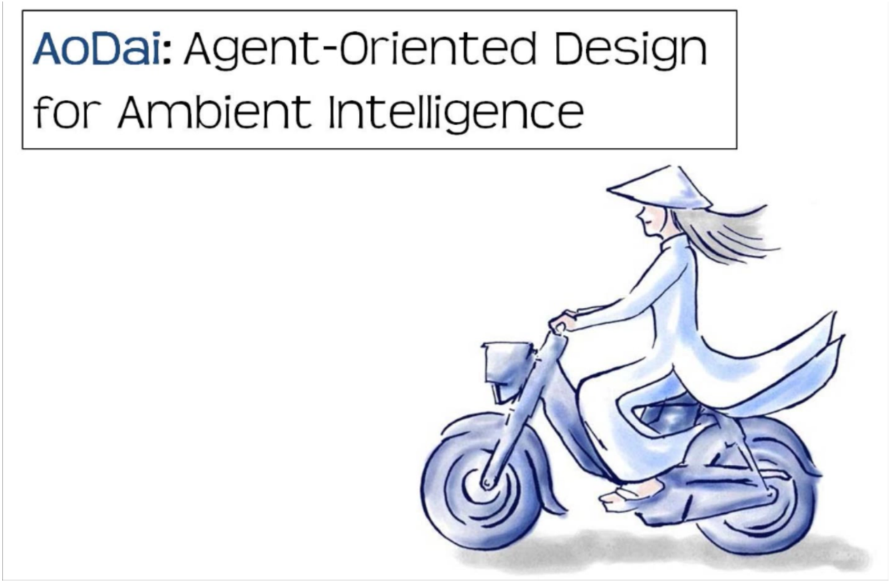





```
20:39:05:0624 INFO  [AliceAgent]:   sent [from:AliceAgent][to:( agent-identifier :name CourseCSAgent@1 )]
20:39:13:0767 INFO  [AliceAgent]:   AliceAgent must migrate to RoomContainer
20:39:13:0773 INFO  [AliceAgent]:   moving to [RoomContainer]
20:39:13:0777 INFO  [AliceAgent]:   moving to [RoomContainer@<Unknown Host>]
20:39:13:0783 TRACE [AliceAgent]:   log out (logs remaining [0]).
```

Outline

Smart Room Demo



Outline

Conclusion

- S-CLAIM
  - Allows programming cognitive and mobile agents in a simple and intuitive manner;
  - Facilitates the development of complex and expressive mental states of the agents, by means of relation-based knowledge bases;
  - Allows to successfully understand and control complex contexts, thanks to the hierarchical representation of agents;
  - Separates the agent-related components and operations, leaving algorithmic processes aside.
- The platform
  - Built on top of JADE, which handles communication, mobility, and agent management.
  - Integrates web services, which are of a great importance for AmI applications;
  - Allows cross-platform deployment and mobile device compatibility.
- Smart Room Scenario;
  - A first scenario was successfully developped and tested, in order to prove the qualities of S-CLAIM.

Future work

- Short term improvements:
  - Designing and implementing more complex scenarios, adapting S-CLAIM in order to completely support their development and rigorously testing the reliability of the platform;
  - Improving the cognitive parts of the agents;
  - Approaching the problem of security.
- Long term improvements:
  - Improving the modularity of S-CLAIM (in order to allow users to easily add new desired features) and developing a way to interact directly with existing function libraries;
  - Designing a better way for the agents to interact with their environment.

Selected references

Amal Fallah-Seghrouchni and Alexandru Suna.
An unified framework for programming autonomous, intelligent and mobile agents.
In Vladimr Mark, Michal Pechoucek, and Jrg Mller, editors, *Multi-Agent Systems and Applications III*, volume 2691 of *Lecture Notes in Computer Science*, pages 1067–1067. Springer Berlin / Heidelberg, 2003.

Amal Fallah Seghrouchni and Alexandru Suna.
Claim and sympa: A programming environment for intelligent and mobile agents.
In Rafael Bordini, Mehdi Dastani, Jrgen Dix, Amal Fallah Seghrouchni, and Gerhard Weiss, editors, *Multi-Agent Programming*, volume 15 of *Multiagent Systems, Artificial Societies, and Simulated Organizations*, pages 95–122. Springer US, 2005.

A. Suna and A. El Fallah Seghrouchni.
Programming mobile intelligent agents: An operational semantics.
Web Intelli. and Agent Sys., 5(1):47–67, January 2007.

Questions and Answers

Thank you!