Ao Dai: Agent-Oriented Design for Ambient Intelligence

Andrei Olaru, Thi Thuy Nga Nguyen, Marius Tudor Benea and Amal El Fallah Seghrouchni

University Pierre and Marie Curie
University Politehnica of Bucharest
French-speaking Institute for Informatics, Hanoi

10.10.2011

Ao Dai: Agent-Oriented Design for Ambient Intelligence

overview

· Ambient Intelligence – or 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]

· We focus on an implementation for the application layer of an AmI system, based on:

- mobile, cognitive agents;
- context-aware transfer of information;
- context-related topology of the agent system;
- decentralization and robustness.

Why build a new platform?

- need of a platform for testing AmI applications;
- requirements:
  - agent-oriented development;
  - hierarchical mobility;
  - mobile agents;
  - compliance with standards, interoperability, modularity;
  - support for deployment on mobile devices
- positive experience with CLAIM and Sympa: successful demonstration of the Ao Dai prototype at the 5th NII-LIP6 workshop in 2010;
- however, Sympa is non-standard and difficult to extend, and CLAIM has a difficult syntax.

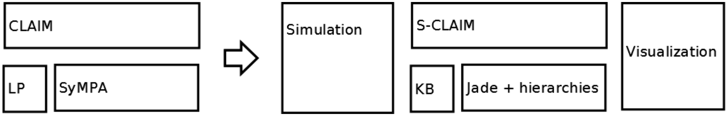
· CLAIM – Computational Language for Autonomous, Intelligent and Mobile agents. [Suna and El Fallah Seghrouchni, 2004]

- agent-oriented programming;
- directed toward agent designers without significant programming skills;
- build on a formal operational semantics;
- inspired by ambient calculus: mobile agents are placed in a hierarchy and move together with their children;
- agent components:
  - parent, authority
  - knowledge, goals
  - messages
  - capabilities
  - processes

· Building blocks for the new platform:

- JADE Java Agent Development Framework – for agent management, inter-agent communication, agent mobility;
- WSIG and WSDL JADE add-ons for compatibility with web services;
- JADE-LEAP for compatibility with mobile devices (e.g. Android smartphones);
- log4j – used, through a wrapper, for centralized logging of the activity of agents and other components;
- SAX Parser – for parsing XML in scenario definition and web-services messages;
- BYACC parser generator – for building the parser for the S-CLAIM language;

· Similar basic structure, but improved in all components:



· Smart-CLAIM

- more simple;
- more focused on agent-specific functionality;
- all algorithmic functionality is exported to external implementations (implemented, for instance, in Java);
- cleaner and easier-to-learn syntax.

· Syntax is Lisp-like:

```
(send ?parent (struct message report ?content))
```

· S-CLAIM semantics:

- based on CoCoMo, a simplified semantics originating from CLAIM.
- reduced number of primitives:
  - messaging: `send, receive`;
  - mobility: `in, out ;`
  - agent management: `new, open, acid`;
  - knowledge management: `addK, removeK, readK, forAllK`;
  - flow control: `condition, if, wait`.
  - user interaction: `input, output`.

```
defineAgentClass Agenda(){
  authority=null; parent=null; knowledge=null; goals=null;
  messages=null; capabilities={ ...
  checkTask{
    message=checkTask();
    condition=hasKnowledge(task(?tn,?h,?m,?p));
    do{forAllKnowledge(task(?tn,?h,?m,?p)){ send(this,check(?tn,?h,?m,?
      .Java(PDA.wait(60)).send(this,checkTask()) } effects=null;
  }
  check{
    message=check(?tn,?h,?m,?p);
    condition=Java(Agenda.isInInterval(?h,?m));
    do{send(parent,task(?tn,?p)).removeKnowledge(task(?tn,?h,?m,?p))}
    effects=null; } }
```

```
(agent Agenda ?schedule ?parent ?owner
  (authority ?owner)
  (behavior ...
    (reactive checkTasks
      (message checkTasks)
      (condition (readK (struct knowledge task ?tn ?h ?m ?p)))
      (forAllK (struct knowledge task ?tn ?h ?m ?p)
        (if (isInInterval ?h ?m)
          (send ?parent (struct message task ?tn ?p)))
          (removeK (struct knowledge task ?tn ?h ?m ?p)))
        )
      )
      (wait 60) (send this (struct message checkTask))
    ) ) )
```

- scenarios are XML files that contain all information on:
  - container and agent creation (local & remote);
  - initial agent knowledge and CLAIM agent parameters;
  - event timeline – sending of messages to agents, etc.

- assures repeatable and easy running of simulations

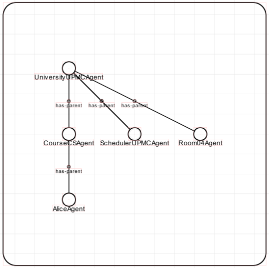
- Example:

```
<?xml version="1.0" encoding="UTF-8"?>
<scen:scenario ...>
 <scen:jadeConfig .../>
 <scen:adfPath>scenario/phase2</scen:adfPath>
 <scen:initial>
  <scen:container name="Administration">
   <scen:CLAIMAgent name="CourseCSAgent" type="CourseAgent">
    <scen:parameter>
     <pr:param name="parent" value="UniversityUPMCAgent" />
     <pr:param name="courseName" value="CSCourse" />
    </scen:parameter>
    <scen:knowledge> ...</scen:knowledge>
   </scen:CLAIMAgent>
  </scen:container>
</scen:scenario>
```

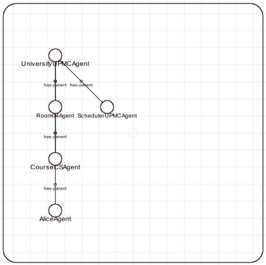
· Visualizable agent – logs all activity locally; reports activity to visualization agent.

· Visualization agent – central entity that gathers activity logs from all agents, as well as movement / hierarchy information.

Example:

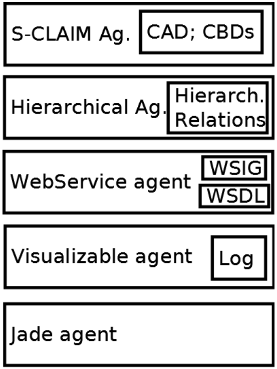


before agent movement



after agent movement

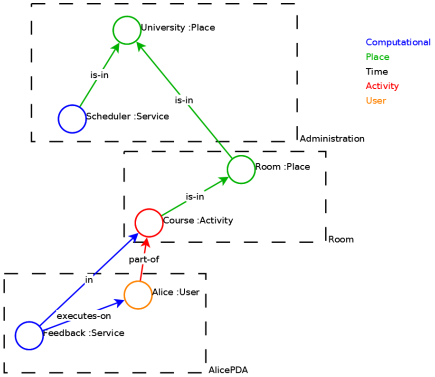
· Window Layout component – places agent windows on the screen automatically.



- S-CLAIM code interpretation
- hierarchical relations and mobility
- expose agents as web services; access external web services
- logging; reporting to the Visualization agent
- agent mobility, communication and management

· Scenario developed in collaboration with the Syamisen team from NII, Tokyo.

- the application interoperates with the SmartRoom and the user detection WSN to assist the students and the professor in course-related activities.



- we have developed a platform for the implementations of AmI applications, that is also suited for other agent-based scenarios;
- it uses a successor of the CLAIM language for the implementation of agents;
- it is based on existing, standard-compliant technologies;
- the platform features replicable scenarios and visualization tools;
- the implementation of the platform has been a collaboration between students from LIP6, University Politehnica of Bucharest (AI-MAS team), and IFI Hanoi;
- it has been tested in collaboration with Honiden-Lab from NII Tokyo.

· Future work:

- implementation of pro-active / goal-driven behavior;
- more powerful scenario format and improved visual representation;
- implementation of function libraries for common algorithmic functionality;
- improved Knowledge Base representation and inclusion of graph-based representations featuring pattern-matching.

Thank You!

Any Questions?

Ducatel, K., Bogdanowicz, M., Scapolo, F., Leijten, J., and Burgelman, J. (2001).
Scenarios for ambient intelligence in 2010.
Technical report, Office for Official Publications of the European Communities.

Ramos, C., Augusto, J. C., and Shapiro, D. (2008).
Ambient intelligence - the next step for artificial intelligence.
IEEE Intelligent Systems, 23(2):15–18.

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?