

Exploiting Agent Observability in Multi-Agent System Programming

A Support in CArtAgO

Alessandro Ricci a.ricci@unibo.it

Michele Piunti michele.piunti@unibo.it

ALMA MATER STUDIORUM – Università di Bologna – DEIS, Italy



MATES 2009

Hamburg

Interaction among Agents

Agents interact in order to coordinate and influence each other

There are (at least) 3 ways for agents to interact:

Direct Explicit message passing, i.e., *speech acts* and *ACLs*

Mediated Mediated by environment and enabled by permanent changes on external entities, i.e., *stigmergy*

Observation Based Agents allocate resources to observe, recognize and possibly reason about others' actions and states

Indirect Interaction

Over-hearing [Dignum and Vreeswijk, 2004]

Agents listening to conversations without the status of addressee

- Monitoring teams of agents and plan recognition [Kaminka et al., 2002]
- Dialogic Scenes in E-insitutions []

Over-sensing [Platon et al., 2005]

Mechanism to let agents observe other agents at work in the same environment

- generalises over-hearing by introducing observations related to agent actions/state
- use of soft-body and symbolic tags to signal additional information

- In MAS Programming frameworks, few concrete technologies have been devised to exploit agent observability in practice.
- For doing this we envisage a comprehensive approach to MAS
 - including both *agents* and *environments*
- We describe a support for agent observability composed by agents that interact and work inside *artifact-based* computational environments running on top of CArtAgO
 - Enabling both over-sensing and stigmergic approaches

1 A&A and CArtAgO

2 Support for Over-sensing

3 Examples

4 Conclusion

Outline

- 1 A&A and CArtAgO
- 2 Support for Over-sensing
- 3 Examples
- 4 Conclusion

A&A meta-model for MAS

Rationale

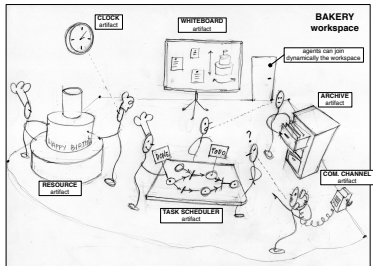
A&A and CArtAgO

Support for Over-sensing

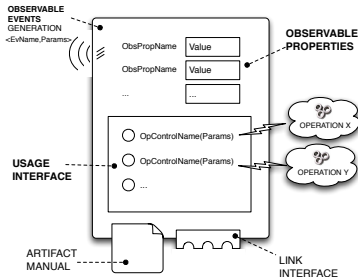
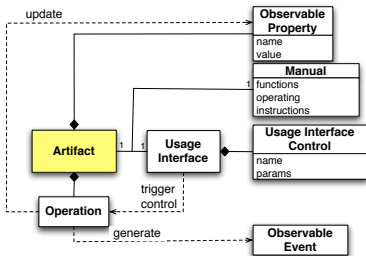
Examples

Conclusion

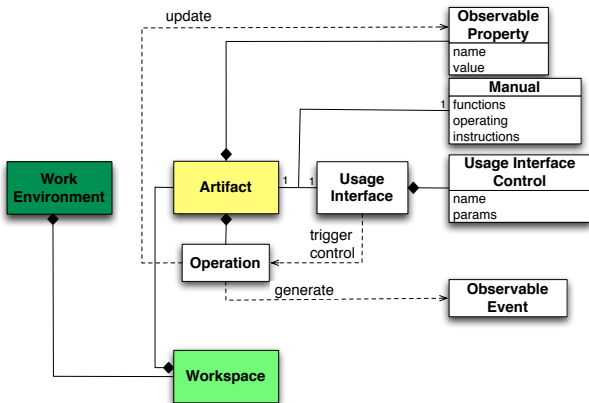
- **Agents** Autonomous, goal-oriented, situated, social, reactive and pro-active entities
- **Artifacts** Non-autonomous, automatic/reactive, function-oriented and stateful entities (controllable and observable by agents)
- **Workspaces** Virtual containers of agents and artifacts, defining the topology and the properties of the work environment



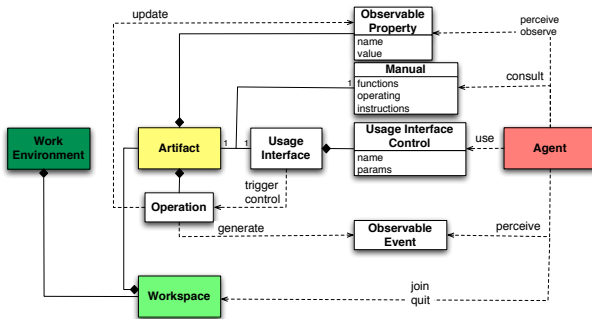
Artifact Computational Model



Workspaces and Work Environments



Agent-Artifact Interaction



Basic Actions for playing in CArtAgO Work Environments

Rationale

A&A and
CArtAgO

Support for
Over-sensing

Examples

Conclusion

(1) `joinWorkspace(+Workspace [,Node])`

(2) `quitWorkspace`

(3) `makeArtifact(+Artifact,+ArtifactType [,ArtifactConfig])`

(4) `lookupArtifact(+ArtifactDesc,?Artifact)`

(5) `disposeArtifact(+Artifact)`

(6) `use(+Artifact,+UIControl([Params]) [,Sensor] [,Timeout] [,Filter])`

(7) `sense(+Sensor,?PerceivedEvent [,Filter] [,Timeout])`

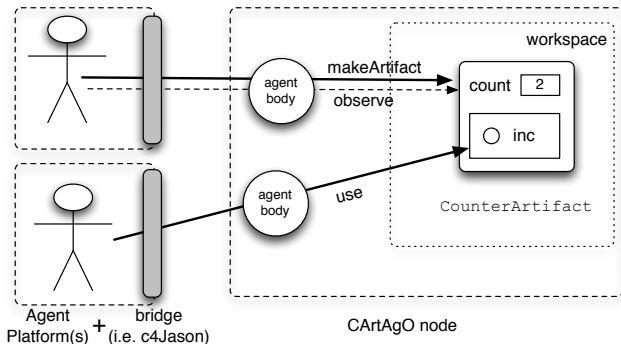
(8) `focus(+Artifact [,Sensor] [,Filter])`

(9) `stopFocussing(+Artifact)`

(10) `observeProperty(+Artifact,+Property,?PropertyValue)`

Table: Agent actions for managing workspaces (1–2), creating, disposing and looking up artifacts (3–5), using artifacts (6–7), and observing artifacts (8–10). Syntax is expressed in a logic-like notation, where italicised items in square brackets are optional [Ricci et al., 2008].

CArtAgO Architecture

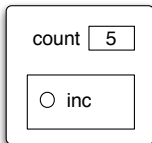


Agent Bodies

As architectural element of the infrastructure to integrate agents minds (belonging to agent platforms) in work environments (belonging to CArtAgO nodes).

- Means for enabling the interaction

A Counter Artifact in CArtAgO



OBSERVABLE PROPERTIES:
count: int

USAGE INTERFACE:
inc: [prop_updated(...),
 op_exec_completed(...)]

```
import alice.cartago.*;

public class Counter extends Artifact {

    void init(){
        defineObsProperty("count",0);
    }

    @OPERATION void inc(){
        int count = getObsProperty("count").intValue();
        updateObsProperty("count",count+1);
    }
}
```

A *Jason* agent using the counter artifact:

```
// Discover Counter in a Workspace
+!discover_counter(C): true
  <- cartago.lookupArtifact("counter",Cid).

-!discover_counter(C) : true
  <- .wait(1000);
  !discover_counter(C).

// Increment Counter 4 times
+!use_counter(C) : cycle(N) & N < 4
  <- cartago.use(C,inc,s0);
  cartago.sense(s0,op_exec_completed(.));
  -+cycle(N+1) ;
  !use_counter(C).

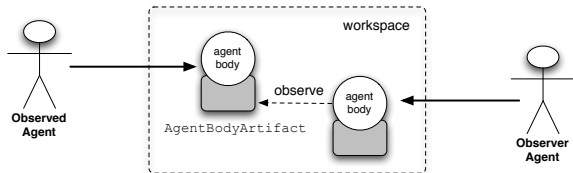
+!use_counter(C) : cycle(4).
```

Outline

- 1 A&A and CArtAgO
- 2 Support for Over-sensing
- 3 Examples
- 4 Conclusion

Enabling Agent Observability

...by making it observable the *agent bodies* as special artifacts:



- 1 Bodies are artifact-ised, through fully-fledged, observable artifacts belonging to the workspace
 - **AgentBodyArtifact** is automatically created once an agent joins the workspace
- 2 Other agents can be aware of one's action and state by observing its **AgentBodyArtifact**

Improving Agents' repertoire

AgentBodyArtifact automatically launch events
signalling ongoing activities

doing(What, When)

- 'What' is an ongoing operation (i.e., use, focus, observeProperty)
- 'When' is a system time stamp

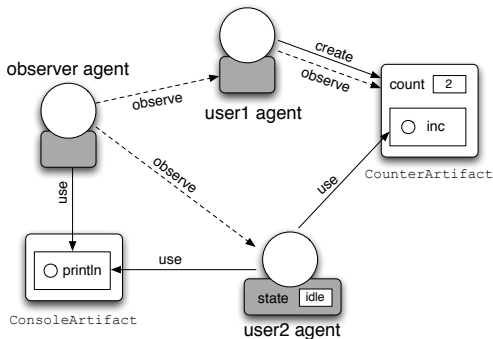
New primitives to manifest an internal state as observable
properties in AgentBodyArtifact

- Agents may explicitly *add* / *update* / *remove* observable properties on own body artifacts
- Read observable properties on others' AgentBodyArtifact

Outline

- 1 A&A and CArtAgO
- 2 Support for Over-sensing
- 3 Examples
- 4 Conclusion

Example



user2 Agent manifest its state and modifies it before using the counter:

```
#!/work : true
<- cartago.updateMyObsProperty(state,"working");
!discover_counter(C);
!use_counter(C);
cartago.updateMyObsProperty(state,"idle").
```

Observer Agent in Jason

```
// observer initial goal
+!observe_agents : true
  <- !discover_agent("user1");
    !discover_agent("user2").

// plans to discover an agent
+!discover_agent(AgName)
  <- cartago.lookupArtifact(AgName,AgBody);
    cartago. focus(AgBody);
      .send(AgName,tell,go_on).
-!discover_agent(AgentName)
  <- .wait(1); !discover_agent(AgName).
```

Observer Agent in Jason

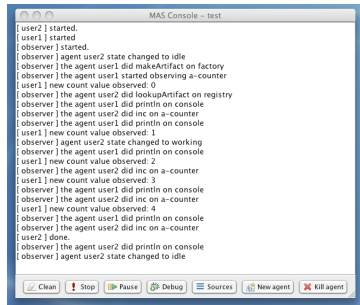
```
// observer initial goal
+!observe_agents : true
  <- !discover_agent("user1");
    !discover_agent("user2").

// plans to discover an agent
+!discover_agent(AgName)
  <- cartago.lookupArtifact(AgName,AgBody);
    cartago. focus(AgBody);
      .send(AgName,tell,go_on).
-!discover_agent(AgentName)
  <- .wait(1); !discover_agent(AgName).

// plans to react to events related to
// observed agents' actions
+doing(use(ArName,OpName,Params),LocalTime)
  [source(Who)]
  <- cartago.use(console,println("Agent ",Who,
    " did ",OpName," on ",ArName)).

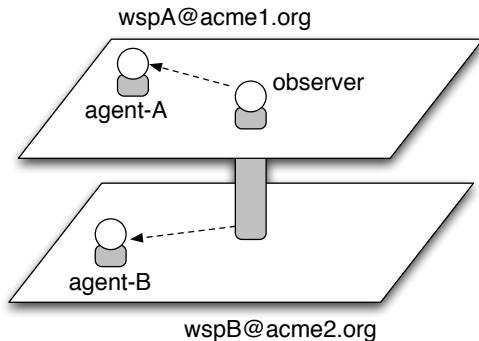
+doing(focus(ArName),LocalTime)
  [source(Who)]
  <- cartago.use(console,println("Agent ",Who,
    " started observing ",ArName)).

// plans to react to events related to
// observed agents' states
+state(X)[artifact(Who)]
  <- cartago.use(console,println("Agent ",Who,
    " state changed to ",X)).
```



```
user2] started.
[user1] started
[observer] started.
[observer] agent user2 state changed to idle
[observer] the agent user1 did makeArtifact on factory
[observer] the agent user1 started observing a-counter
[user1] new count value observed: 0
[observer] the agent user2 did lookupArtifact on registry
[observer] the agent user1 did println on console
[observer] the agent user2 did inc on a-counter
[observer] the agent user1 did println on console
[user1] new count value observed: 1
[observer] agent user2 state changed to working
[observer] the agent user1 did println on console
[user1] new count value observed: 2
[observer] the agent user2 did inc on a-counter
[user1] new count value observed: 3
[observer] the agent user1 did println on console
[observer] the agent user2 did inc on a-counter
[user1] new count value observed: 4
[observer] the agent user1 did println on console
[observer] the agent user2 did inc on a-counter
[user2] done.
[observer] the agent user2 did println on console
[observer] agent user2 state changed to idle
```

A distributed application



Agents may have several body-artifacts, for each joined workspace

Outline

- 1 A&A and CArtAgO
- 2 Support for Over-sensing
- 3 Examples
- 4 Conclusion

Conclusions

- Mechanism to let agents observe other agents at work in the same environment (*over-sensing* [Platon et al., 2005])
 - Generalises over-hearing by introducing observations related to agent actions/state
 - Alternative approach to communication, complementary to message passing
 - Challenging for improving interactions in Massive MAS and Organizations

Conclusions

- Mechanism to let agents observe other agents at work in the same environment (*over-sensing* [Platon et al., 2005])
 - Generalises over-hearing by introducing observations related to agent actions/state
 - Alternative approach to communication, complementary to message passing
 - Challenging for improving interactions in Massive MAS and Organizations
- Observability is limited to agent using artifacts and workspaces
 - Not to actions addressed to other entities of the system, i.e. messages

Conclusions

- Mechanism to let agents observe other agents at work in the same environment (*over-sensing* [Platon et al., 2005])
 - Generalises over-hearing by introducing observations related to agent actions/state
 - Alternative approach to communication, complementary to message passing
 - Challenging for improving interactions in Massive MAS and Organizations
- Observability is limited to agent using artifacts and workspaces
 - Not to actions addressed to other entities of the system, i.e. messages
- Enabling Behavioral Implicit Communication (*BIC* [Castelfranchi, 2006])
 - Agents acting with the aim (goal) to be observed, hence at changing the mind/behavior of other agents
 - Plan Recognition abilities to integrate observation in a time scale
 - To be implemented within agent “cognitive” model

Exploiting Agent Observability in Multi-Agent System Programming

A Support in CArtAgO

Alessandro Ricci a.ricci@unibo.it

Michele Piunti michele.piunti@unibo.it

ALMA MATER STUDIORUM – Università di Bologna – DEIS, Italy



MATES 2009

Hamburg