Agents & Artifacts at the Knowledge Level

Samuele Burattini [1] Andrei Ciortea [2] Meshua Galassi [1]
Alessandro Ricci [1]

Dipartimento di Informatica - Scienza e Ingegneria,
Alma Mater Studiorum, Università di Bologna, Cesena Campus, Italy
meshua.galassi@studio.unibo.it,
{samuele.burattini|a.ricci}@unibo.it

School of Computer Science,
University of St.Gallen, Switzerland
andrei.ciortea@unisg.ch

29th May, 2023

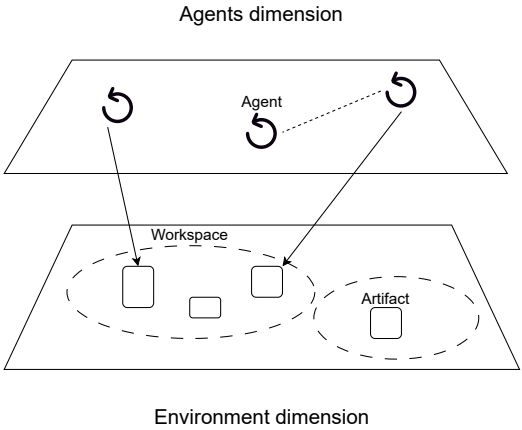
Table of Contents

1. Agents & Artifacts
2. Proposed extension
3. Prototype and Supporting Technologies
4. Conclusions and Future Directions

The Agents & Artifacts meta-model

The Agents & Artifacts (A&A) meta-model[1] is a way to define the *environment* in MAS based on:

- **Artifacts** as resources and tools that can be created, shared, observed and used by agents
- **Workspaces** as logical containers of Artifacts



[1] Ricci A., Viroli M. and Omicini A., 2007. Give agents their artifacts: the A&A approach for engineering working environments in MAS.

Agents & Artifacts: current limitations

Artifacts are inspired by how humans use tools and have:

- **Observable properties** that can be perceived by agents
- **Operations** that are exploitable by agents' actions
- **Events** that can notify the agents of changes

Agents & Artifacts: current limitations

Artifacts are inspired by how humans use tools and have:

- **Observable properties** that can be perceived by agents
- **Operations** that are exploitable by agents' actions
- **Events** that can notify the agents of changes

But...

... in the real world, human agents also leverage their **domain knowledge** and the **relationships** among artifacts to efficiently use them.

For example...

Building a smart-room system:

- an agent has the goal of turning on a Lamp
- the domain knowledge is that Lamp and Switches exist in rooms and Switches control Lamps

Then we would like to program the agent behaviour to:

1. enter a room
2. find a lamp
3. find the switch that controls it
4. use the switch to achieve its goal

1. `joinWorkspace("room-001", WP)`
2. $\simeq$ `lookupArtifactByType("LampClass", L)`
3. `????`
4. `turnOn()[artifact_id(S)]`

Table of Contents

1. Agents & Artifacts
2. Proposed extension
3. Prototype and Supporting Technologies
4. Conclusions and Future Directions

Extending A&A

Why?

Agents benefit from having a **common level of abstraction** to describe both their internal knowledge and the domain entities in the environment, including their **relationships**.

With this explicit description they could:

Query the environment to find Artifacts

- of a given kind
- in a given state
- following relationships

Observe the environment and be notified of changes

- of a single Artifact
- of connected ones
- of new connections

Manipulate the environment

- using domain operations
- without dealing with low-level details

A&A at the Knowledge Level

We identify the **common level of abstraction** as the *Knowledge Level* proposed by Newell[2] to be the highest level in the hierachy of computer systems

As done by Jennings[3] for the social dimension of MAS we further extend this to include the environment dimension.

We do that following some design principles.

Knowledge-level systems Medium: Knowledge Laws: Principle of Rationality
Program-level systems Medium: Data structures, programs Laws: Sequential interpretation of programs
Register-transfer system Medium: Bit vectors Laws: Parallel logic
Logic circuits Medium: Bits Laws: Boolean algebra
Electric circuits Medium: Voltage/current Laws: Ohm's law, Kirchhoff's law
Electronic devices Medium: Electrons Laws: Electron physics

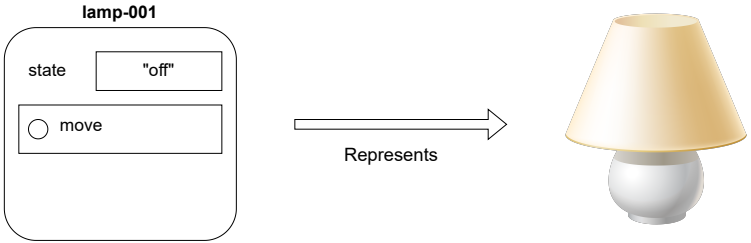
[2] Newell A., 1982. The knowledge level.

[3] Jennings N.R., 2000. On agent-based software engineering

Design principle I

Artifacts are Domain Entities

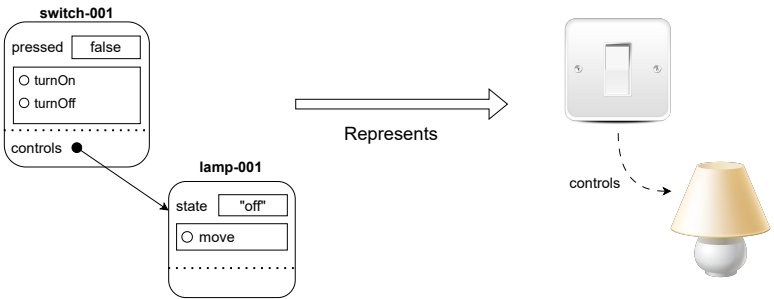
Artifacts should be semantically ground to domain entities: their affordances and their manuals should be described at that same level of abstraction.



Design principle II

Explicit Relationships

Relationships among entities at the domain level should be explicitly represented and reified at the artifact level so that agents can reason about them



Design principle III

Vocabulary consistency

Workspaces can be used to define logical (bounded) contexts that share the same domain vocabulary to describe the entities within them.

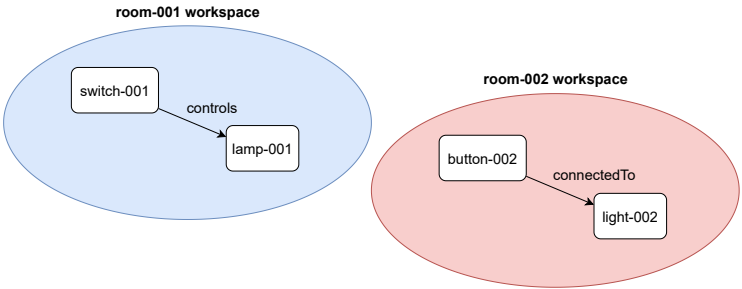


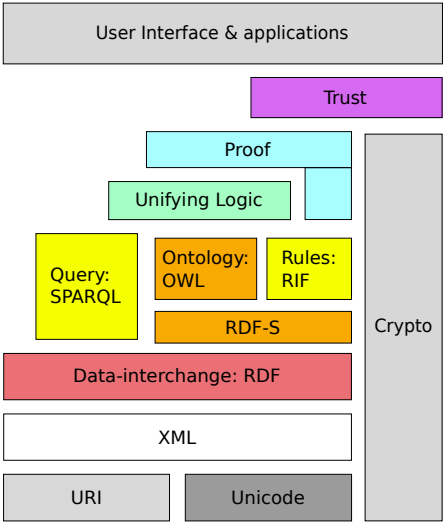
Table of Contents

1. Agents & Artifacts
2. Proposed extension
3. Prototype and Supporting Technologies
4. Conclusions and Future Directions

Supporting Technologies

We looked into how knowledge is represented in the Semantic Web:

- **Knowledge Graphs** (KG) allow to convey knowledge about entities in the world and relationships among those entities
- **RDF** is the standard way to represent KGs on the Web
- **SPARQL** is the standard language for querying RDF graphs



CArtAgO extension

We developed a prototype adding an *explicit semantic layer* on top of CArtAgO[4]:

One KG for Workspace

Each workspace manages a centralized KG containing all its Artifacts

Artifact RDF description

Each Artifact generates and mantains up-to-date its RDF description in the KG

Agents can query the KG

Using an API to express SPARQL queries and acquire new knowledge

[4] the reference implementation of A&A

... back to our Lamp Agent

Artifacts will automatically populate the Workspace's KG with their RDF description:

```
@prefix : <http://example.org/> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .

:lamp-001      a owl:NamedIndividual, :Lamp ;
               :state "off" .

:switch-001 a owl:NamedIndividual, :LightSwitch ;
               :controls :lamp-001 ;
               :pressed false .
```

At runtime, the agent will be able to query the KG and achieve its goal.

A solution using Knowledge

Here a Jason agent performs SPARQL queries on the environment to find a Lamp and then the Switch that controls it.

```
1 +!turnOnLamp : true
2 <-  joinWorkspace("room-001", WP)
3     query("SELECT ?l WHERE { ?l rdf:type :Lamp }", R1);
4     getValue(0, "l", R1, LampID);
5     .concat("SELECT ?s WHERE { ?s :controls :", LampID, "}", Q);
6     query(Q, R2);
7     getValue(0, "s", R2, SwitchID);
8     turnOn()[artifact_id(SwitchID)]
```

Note how, differently from before, the agent deals only with the *domain knowledge* and does not need to know any implementation details about the artifacts (e.g. the class name)

Table of Contents

1. Agents & Artifacts
2. Proposed extension
3. Prototype and Supporting Technologies
4. Conclusions and Future Directions

The Road Ahead

This vision introduces many challenges and open issues.
Among those, we highlight:

- How to support querying in large, distributed artifact graphs?
- How to work with multiple existing domain ontologies?
- How to bring all the MAS dimensions to the Knowledge Level in a coherent fashion?
- How to devise or adapt methodologies to build MAS at the Knowledge Level?

We look forward to continue researching solutions to these problems!