simpA and CARTAGO

PROTOTYPING TECHNOLOGIES FOR AGENT-ORIENTED COMPUTING

Alessandro Ricci
a.ricci@unibo.it
DEIS, Università di Bologna
Sede di Cesena

AGENDA

- simpA and CARTAGO Background:
  - Revisiting the notion of environment in Multi-Agent Systems (MAS)
  - The A&A Basic Conceptual Framework
- A&A prototyping technologies
  - simpA
    - Java based agent-oriented framework
  - CARTAGO
    - Java based framework for bulding artifact-based MAS environments

BACKGROUND:
THE NOTION OF ENVIRONMENT IN MAS AND THE A&A APPROACH

"Artifacts *play a critical role in almost all human activity [...]. Indeed [...] the development of artifacts, their use, and then the propagation of knowledge and skills of the artifacts to subsequent generations of humans are among the distinctive characteristics of human beings as a specie*", Donald Norman [1]

"*The* use *of* tools *is a hallmark of intelligent behaviour. It would be hard to describe modern human life without mentioning tools of one sort or another*", Robert Amant [2]

[1] D. Norman. Cognitive artifacts. In J. Carroll, editor, *Designing interaction: Psychology at the human–computer interface*, pages 17–38. Cambridge University Press, New York, 1991.

[2] Tool use for autonomous agents. In M. M. Veloso and S. Kambhampati, editors, *AAAI/IAAI'05 Conference*, pages 184–189, Pittsburgh, PA, USA, 9–13 July 2005. AAAI Press / The MIT Press.

BACKGROUND: ON THE ROLE OF *ARTIFACTS* IN HUMAN SOCIETY

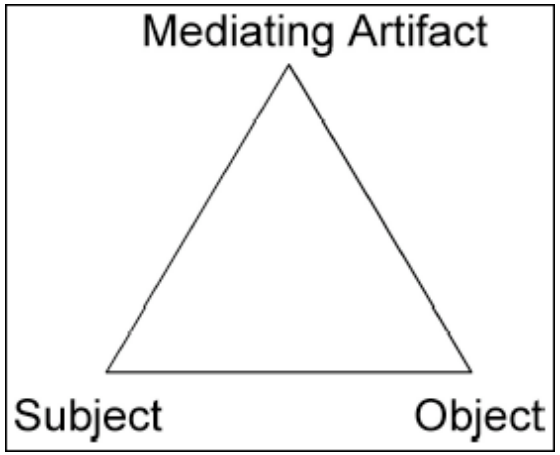
- Artifacts & tools as essential elements for *mediating* & *supporting* human what-ever-kind-of activities
  - especially social activities
- A first characterisation
  - artifacts as what-ever-kind of device explicitly *designed* to embed some kind of functionality, that can be properly *used* by humans to do their work
    - tools perspective
  - artifacts as objectification of human goals and tasks
    - target of the activity
- Basic bricks of human *environment*
  - in particular *working environments*
  - *fields-of-work notion* in CSCW

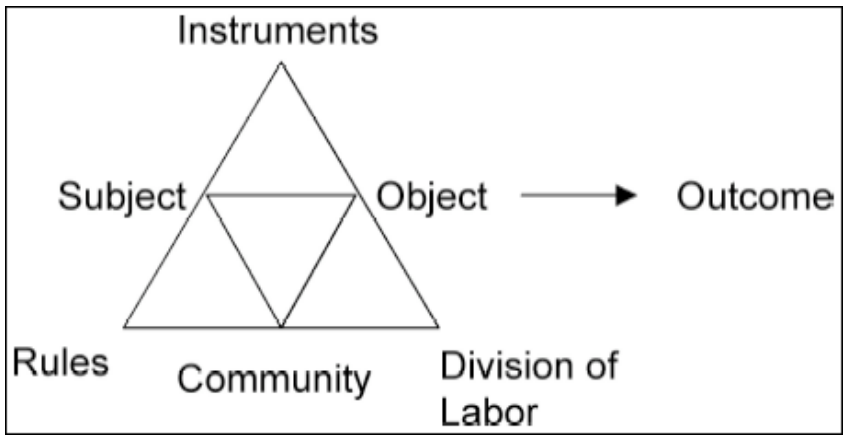
PERVASIVE NOTION

- Activity Theory
  - [context: psychology & human sciences]
  - artifacts as mediators of any (complex) activity
- Distributed Cognition
  - [context: cognitive science]
  - cognition spread among humans *and the environment where they are situated*
- CSCW
  - artifacts as bricks of human fields of works
  - coordinative artifacts..
- Distributed Artificial Intelligence
  - the role of environment for supporting intelligence
  - “Lifeworld Analysis” by Phil Agre and Horswil (Journal of Artificial Intelligence, 1997)
- …

ACTIVITY THEORY IN PARTICULAR...

- basic mediating element of *social activities*



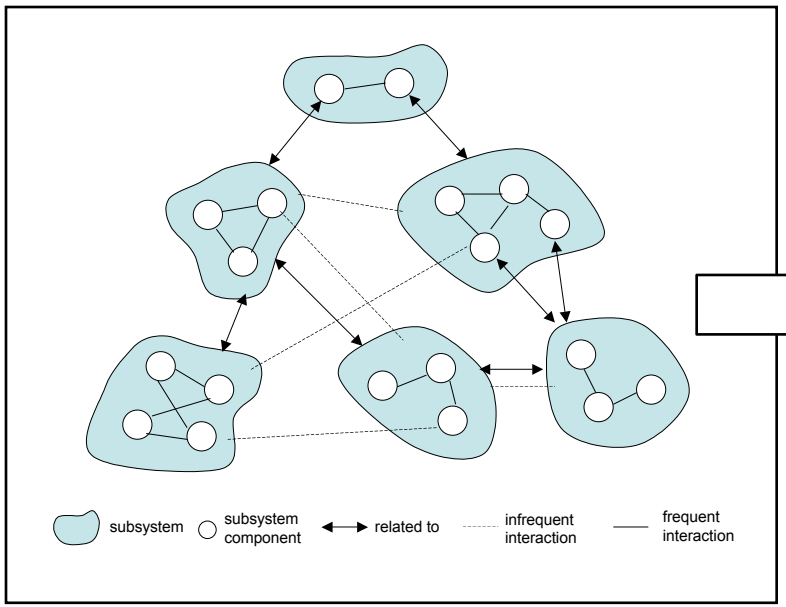


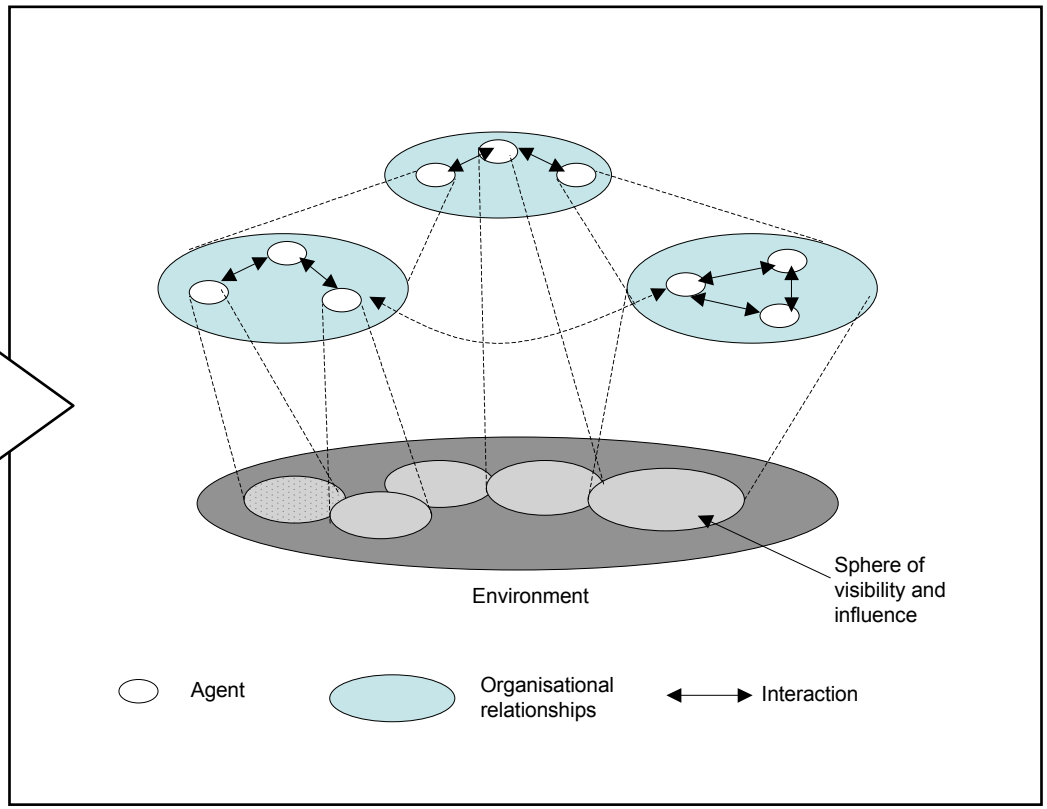
FROM HUMAN SOCIETY TO AGENT SOCIETIES (?)

"If it is such a fundamental concept for human society and human working environments *could it be useful also in agent societies and MAS for conceiving agent "working environments"?*

- Analogy between human and agent societies
  - agent societies / MAS conceived as systems of autonomous cognitive entities working together within some kind of *social activities,* situated within some kind of *working environment...*
- A&A conceptual framework
  - defining a notion of artifact and working environment within MAS
  - foundational & engineering aims
    - finally useful for building MAS and agent-based systems

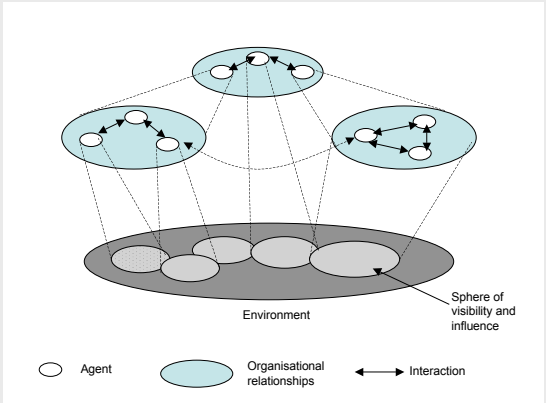
AGENT AND MAS AS A PARADIGM FOR ENGINEERING SYSTEMS





THE ROLE OF THE "ENVIRONMENT"...

- For most definition and models, typically something "already there"
  - *"out of the agents",*
  - *"out of the MAS"*
- Out of the design scope of MAS engineers



ENVIRONMENT IN THE LOOP OF MAS ENGINEERING

- Revisiting the role of the environment in MAS and MAS engineering
  - useful for designing & engineering MAS
    - E.g. environment-based coordination
      - pheromone-based environment for ant-based MAS (Parunak)
      - fleld-based middleware (Mamei & Zambonelli)
- State-of-the-art research in MAS
  - E4MAS workshops in AAMAS conference
    - 2006,2005,2004
    - Lecture Notes in Computer Science (Springer)
  - D.Weyns and H.D. Parunak, editors. *Journal of Autonomous Agents and Multi-Agent Systems. Special Issue: Environments for Multi-Agent Systems*, volume 14(1), Springer, 2007

A&A OBJECTIVES

- Abstraction and generality
  - defining a basic set of abstractions & related theory *general enough* for designing & engineering general-purpose working environments
    - basic engineering principles in mind
      - encapsulation, reuse, extendibility, etc
  - *effective / expressive* enough to capture the essential properties of specific domains
- Cognition (="beyond environments for ants")
  - the properties of such environment abstractions should be conceived to be suitably and effectively exploited by *cognitive agents*
    - agents as intelligent constructors, users, manipulators of such environment
  - analogy with human society

A&A BASIC ELEMENTS

- MAS ~ set of agents working together in the same *working environment*
- Working environment as an set of *artifacts*, collected in workspaces
- Agents dynamically create, share and use artifacts to support their social/individual activities
  - mediation role of artifacts

ARTIFACT ABSTRACTION

- Basic building blocks of working environments
  - different kinds / types of artifacts, with possibly multiple instances for each type
    - similar to *objects* but in an agent world
  - representing any kind of resource or tool that agents can dynamically create, share, use, manipulate
- Passive, dynamic, *function*-oriented entities
  - designed to encapsulate some kind of *function*
    - the intended purpose of the artifact
    - vs. agent as goal/task-oriented entities
  - typically state-full

NOTION OF USE AND USAGE INTERFACE

- Agents-artifacts interaction based on a notion of *use*
  - agents *use* artifacts so as to exploit their functionalit
  - vs. communication-based interaction
    - agents *do not* communicate with artifacts
- Artifact *usage interface*
  - set of *operations* that agent can trigger on the artifact
  - it can change depending on the working state of the artifact
- Artifact *observable state* and *events*
  - artifacts as sources of observable events that can be perceived by agents using or observing them

THE NOTION OF MANUAL

- Equipping each artifact with the formal description of its functionality and usage modalities
  - *function description*
    - *"why* to use the artifact"
  - *operating instructions*
    - "how to use the artifact"
- Enabling a cognitive use of artifacts
  - enabling dynamic discovery, selection and learning of artifacts for cognitive agents
    - open systems perspective

WORKSPACE ABSTRACTION

- Logical containers of artifacts
- Useful to define *the topology* of the working environment
  - localities
  - scopes constraining artifacts usage and observations

KINDS OF ARTIFACTS: EXAMPLES

- What ever kind of "mechanism in MAS" can be suitably re-casted as an artifact
  - raising the level of abstraction
- Special purpose...
  - e.g. game-board in any MAS-based games
    - usage interface enabling agents to make a move and observe the state of the game
    - encapsulating the rule of the game
  - interface & resource wrappers
    - enabling and ruling the access to (possibly) existing specific resources

KINDS OF ARTIFACTS: EXAMPLES (II)

- General purpose
  - a simple example: a shared knowledge base
    - usage interface for inserting, retrieving, query knowledge
    - ensuring consistency among updates / queries
  - *coordination artifacts*
    - artifacts designed to provide some kind of coordination service
      - managing agent dependencies
      - constraining agent interaction
      - encapsulating social norms and rules
    - pervasive examples
      - synchronizers, maps, ticket dispensers,...
      - mailboxes, blackboards, tuple spaces, tuple centres,...
      - workflow engines, auction engines,...

simpA

simpA

- Framework for prototyping A&A-based application on top of the Java platform
  - it requires the JDK 5.0
    - heavily usage of annotations
- available at
  - http://www.alice.unibo.it/projects/simpa
  - current version 0.4.1
    - major release with the forecoming 0.5.0

simpA OBJECTIVES

- Two basic objectives
  - providing a programming environment for fast prototyping A&A-based applications
    - Agents, artifacts, workspaces
  - reusing as much as possible the support provided by a mainstream OO environment
    - Java platform
- Orthogonality & separation of concerns
  - agents and artifacts are at a different level of abstraction than objects
    - agents and artifacts are not objects or “extended objects”
  - OO abstractions (classes, interfaces,..) used for defining abstract data types
    - used to define agents & artifacts structure & behaviour
    - used to define types of objects involved in interactions

A simpA APPLICATION

- An application as a multi-agent system
  - a (dynamic) set of agents working together inside the same *working environment*
  - a working environment is composed by a (dynamic) set of artifacts, structured in workspaces
- Programming a simpA applications involves:
  - defining basic agent and artifact types
  - defining the application (MAS) main to:
    - create a working environment
    - define the initial set of artifacts
    - spawn the initial sets of agents

HELLO WORLD

```
import alice.simpa.*;

public class HelloAgent extends Agent {
  @ACTIVITY void main() throws SimpaException {
    ArtifactId aid = getArtifactId("stdout");
    execOp(new Op(aid,"println","Hello, world! from "+getId().getName()));
    shutdown();
  }
}

public class STDOutput extends Artifact {
  @OPERATION void println(String msg){
    log(msg);
  }
}

public class HelloWorld {
  public static void main(String[] args) throws Exception {
    Environment env = new Environment("hello-world-app");
    env.createArtifact("stdout",STDOutput.class);
    env.spawnAgent("simpa",HelloAgent.class);
  }
}
```

HELLO PHILOSOPHERS!

- [check sources on the distribution]

AGENTS IN simpA: OVERVIEW

- State-full entities with an identity
  - logic name + unique identifier
  - agent memory
    - internal state & knowledge base
- Activity-oriented behaviour
  - behaviour structured in terms a hierarchy of *activities*
    - unit of work inside the working environment
  - activities can be *simple* or *complex*
    - either or not structured in sub-activities
      - notion of *agenda* with TODO for defining the structure of complex activities
- Interaction side: sensors and effectors
  - actions to act construct, use, manipulate artifacts
  - sensors to perceive stimuli from the working environment
    - observable events generated by artifacts

DEFINING AN AGENT

- Agent templates (=types)
  - defining agent structure and behaviour
- Implemented as classes extending the base class `alice.simpa.Agent`
  - private field as agent memory / state
  - annotated methods to define activities body
    - `@ACTIVITY` to define simple activities
    - `@ACTIVITY_WITH_AGENDA` to define complex activities
    - no return parameter
    - `main` as default main activity
  - *basic actions* available as protected method provided by the base class
  - private methods can be used to structure the algorithmic part

BASIC EXAMPLES

- [check sources on the distribution]
  - alice.simpa.examples.basic

ACTIVITIES: DETAILS

- Activity Agenda
  - basic element: TODO
    - as a work (sub-activity) to be done
    - TODO pre-conditions defining when it can be done
  - Inherent concurrency
    - in each activity frame, whenever a TODO pre-conditions hold, the related sub-activity is executed and the TODO removed
  - Coordinating sub-activities: NOTEs
    - memo that can be attached to the agenda
      - agent internal actions
    - can be used to trigger the execution of TODOs
    - can be used to synchronise sub-activities...
- Activities can succeed or fail
  - possibility to manage failures

AGENDA: BASIC EXAMPLES

- [check sources on the distribution]

ACTIONS & PERCEPTIONS

- Actions
  - atomic constituents of activities
  - internal or external
    - inspecting and updating the state of the agent (internal)
    - constructing, using, manipulating artifacts (external)
- Perception
  - sensing stimuli from the working environment
    - observable events generated by the artifacts
  - *active sensing*
    - perception as the result of an internal action of sensing on sensors
  - pattern / data-driven sensing
    - defining filters

BASIC EXAMPLE

- [check sources on the distribution]

AVAILABLE ACTIONS (I)

ARTIFACT CONSTRUCTION & USE

```
createArtifact(String name,String|Class template,{ArtifactConfig config}):ArtifactId
disposeArtifact(ArtifactId aid)
```

Primitives for dynamic instantiation and disposal artifacts

```
execOp(Op op,{SensorId aid}):OpId
```

Primitive action for executing an operation on an artifact. The artifact identifier, operation names and possible parameters are specified with the `Op` objects

---

```
getArtifactId(String name):ArtifactId
createArtifactId(String name,String location):ArtifactId
```

Gets the identifier of an existing artifacts

```
getArtifactOI(ArtifactId aid):String
getArtifactFD(ArtifactId aid):String
getArtifactState(ArtifactId aid):String
```

Gets artifact static and dynamic properties.

ARTIFACT MANIPULATION

```
acquireSession(ArtifactId aid):SessionId
releaseSession(ArtifactId aid)
```

Acquires and releases a (private) session for interacting with an artifact.

AVAILABLE ACTIONS (II)

SENSING

```
sense({SensorId sid},{IPerceptionFilter filter},{int duration}):Perception
```

Basic primitive to sense perceptions.

```
linkSensor(Sensor s):SensorId
unlinkSensor(SensorId aid)
```

Links and unlinks sensors to agent body.

AGENTS' INTERACTION

```
getAgentId(String name):AgentId
createAgentId(String name,String location):AgentId
```

Gets the identifier of existing agents.

```
spawnAgent(String name,String|Class template,{AgentConfig config}):AgentId
```

Spawns a new agent.

```
speak(Message msg)
```

Communicative act to send a message to another agent.

AVAILABLE ACTIONS (III)

SELF-INSPECTION & CONTROL

`getId():AgentId`
Gets agent own identifier.

`sleep(int duration)`
Suspend agent current activity for the specified amout of time.

`shutdown()`
Terminates agent execution.

`log(String msg)`
Prints on standard output the specified message.

AGENDA INSPECTION & MANIPULATION

```
memo(Memo memo)
removeMemo(Memo template):boolean
collectMemos(Memo template):Memo[]
removeMemos(Memo template):Memo[]
```

Primitives to insert, remove and collect memos.

ARTIFACTS in simpA: OVERVIEW

- State-full entities with an identity
  - logic name + unique identifier
  - protected / hidden dynamic state
- Function-oriented entities
  - behaviour structured in terms of *operations*
    - changing artifact inner & observable state
    - generating observable events
  - Usage interface
    - set of operations that agent can execute on an artifact
    - can change according to artifact *observable state*
  - Manual
    - defining artifact function description, operating instructions, observable states & other descriptive parts

DEFINING AN ARTIFACT

- Artifact templates (=types)
  - defining artifact basic structure and behaviour
- Implemented as classes extending the base class `alice.simpa.Artifact`
  - private fields used to define artifact state
  - annotated methods to define operation bodies
    - `@OPERATION` annotation
    - no return parameter
    - operation parameters as method parameters
  - private methods to for structuring the algorithmic behaviour of operations
  - class annotation to define artifact manual
    - `@ARTIFACT_MANUAL` annotation

BASIC EXAMPLES

- [check sources on the distribution]

OPERATIONS & OBSERVABLE EVENTS

- Operation execution
  - atomic
    - monitor-like behaviour of artifacts, without condition variables & without synch primitives
  - can succeed or fail
    - generating exceptions as special events
- Observable events
  - can be generated either related or not to a specific operation execution instance
    - getEvent primitives
  - implemented as classes extending the base class `alice.simpa.Event`

BASIC PRIMITIVES FOR PROGRAMMING ARTIFACTS

EVENT GENERATION

```
getEvent([OpId id],Event ev)
```

Generates an event, related to an operation request.

OPERATION INSPECTION

```
getCurrentOpId():OpId
getCurrentOpRequestDescriptor():OpId
```

Primitives an inspect information about operation requests.

STATE INSPECTION & CHANGE

```
getCurrentObservableState():String
setCurrentObservableState(String state)
```

Primitives to inspect and change current observable state.

PROPERTIES' INSPECTIONS

```
getId():ArtifactId
getOI():String
getFD():String
```

Primitive to inspect artifact identifier, operating instructions and function description.

MISCELLANEA

```
log(String msg)
```

Prints on standard output the specified message.

KINDS OF ARTIFACT: EXAMPLES

- Resource / Interface artifacts
  - artifacts used to wrap and rule the access to resources
  - E.g. dbase interface artifact
- Coordination artifacts
  - artifact providing coordinating functionalities
  - E.g: Synchronizers, Latches, Spaces,...
- Artifacts mediating human / agents interaction
  - E.g. GUI Artifacts
    - [check the examples on the distribution]

AN OVERALL EXAMPLE: HELLO PHILOSOPHERS

- Dining philosopher problem
  - a society of N philosophers sitting at the same round table, sharing chopsticks, interleaving thinking and eating activities
- Solution in simpA
  - restaurant working environment
  - Table artifact
    - implementing the table abstraction
      - chopstick management
    - embedding and enacting the rules for coordinating chopstick access
  - Philosophers agent
    - implementing the philosopher abstraction
  - [check the solution in alice.simpa.examples.philo]

ONGOING WORKS

- Major release preview
  - new model for artifact operation definition
    - support for non atomic operations
      - composed by atomic operation steps
  - new threading model for activity scheduling & execution
    - thread pool for executing (sub)-activities
  - workspace support
    - primitives for moving artifacts
  - new primitive for artifact manipulation
    - to grab artifacts
  - artifact-support based on CARTAGO
    - simpA based on CARTAGO

CARTAGO

CARTAGO FRAMEWORK

- Framework / infrastructure for prototyping artifact-based working environments
  - Common ARTifact for AGent Open Environment
- Conceived / designed to be integrated to existing MAS programming environment
  - Enabling MAS designers / programmers to develop their own kinds of artifacts
  - Enabling agents to participate in working environment
- Fully Java-based technology, available as open-source project at
  - http://www.alice.unibo.it/projects/cartago

CARTAGO ABSTRACT MODEL

- Working environment
  - workspaces
  - artifacts
  - *agent bodies*
- artifacts as instances of artifact types
  - defining usage interface, manual, state and behaviour
  - programming model on-top-of Java
- The notion of agent body
  - what makes it possible for an agent to be *situated* in the working environment
    - sensors and effectors for acting and perceiving observable events from the environment
  - ‘piloted’,controlled by the agent mind (outside CARTAGO)

BASIC ACTIONS AVAILABLE TO AGENTS

- Artifact discovery, construction and disposal
  - createArtifact(Name,TypeID,Conf,WspID)
  - getArtifactID(Name,WspID):AID
  - disposeArtifact(AID)
- Artifact use
  - execOp(AID,Op(Name,Params),SID)
  - sense(SID,Filter,Timeout):Perception
  - focus(AID,SID)
  - stopFocus(AID,ID)
- Artifact inspection
  - getFD(AID):FD
  - getOI(AID):OI
  - getObsState(AID):ObsState
- Sensor management
  - linkSensor(SensorType):SID
  - unlinkSensor(SID)

PROGRAMMING MODEL FOR DEFINING ARTIFACT TYPES

- Reusing as much as possible the OO support of Java
  - annotation
- Basic model for implementing artifacts
  - state
  - operations
  - observable state and events
  - manual

INTEGRATION WITH EXISTING MAS PROGRAMMING PLATFORM

- Integrating CARTAGO with existing agent-oriented programming platform
  - on the agent side: extending the basic set of agent actions with the new ones dealing with working environments
  - mapping observable events sensed within working environments as agent perceptions
- Examples
  - simpA
  - JASON (ongoing)