Agent & Artifact (A&A)
ARTIFACT-BASED ENVIRONMENT FOR MAS

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

EMSE, November 2006
AGENDA

• Revisiting the notion of environment in Multi-Agent Systems (MAS)

• The A&A Basic Conceptual Framework

• CARTAGO framework / infrastructure
ON THE NOTION OF ENVIRONMENT IN MAS
“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]


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)
      – field-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)
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 OVERVIEW
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 functionality
  – 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,...
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)
  – unfocus(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)
simpA

- simple agent-oriented programming environment on top of Java
  - directly implementing the A&A framework
    - activity-oriented characterisation of agents
    - based on CARTAGO for the artifact side

- available at
  - http://www.alice.unibo.it/projects/simpa
JASON DINING PHILOSOPHERS USING CARTAGO TABLE

• “Hello philosophers”
  – implementing the dining philosophers as a MAS
  – Agent philosophers implemented in Jason and simpA
  – The “Table” artifact implemented in CARTAGO
THE TABLE

- Two operations
  - `getChops`
  - `releaseChops`
- Observable events
  - `chops_acquired`

```java
import alice.cartago.*;
import java.util.*;

public class Table extends Artifact {
    private boolean[] chops;
    private List<PendingReq> reqs;

    public Table(int nchops) {
        chops = new boolean[nchops];
        reqs = new LinkedList<PendingReq>();
        for (int i = 0; i < chops.length; i++) {
            chops[i] = true;
        }
    }

    @OPERATION void getChops(int c0, c1) {
        if (chops[c0] && chops[c1]) {
            chops[c0] = chops[c1] = false;
            genEvent("chops_acquired");
        } else {
            PendingReq req = new PendingReq(c0, c1, getOpId());
            reqs.add(req);
        }
    }

    @OPERATION void releaseChops(int c0, c1) {
        chops[c0] = chops[c1] = true;
        Iterator<PendingReq> it = reqs.listIterator();
        while (it.hasNext()) {
            PendingReq r = it.next();
            if (chops[r.c0] && chops[r.c1]) {
                it.remove();
                chops[r.c0] = chops[r.c1] = false;
                try {
                    genEvent(r.reqId, "chops_acquired");
                } catch (Exception ex) {
                }
            }
        }
    }

    private static class PendingReq {
        public int c0, c1;
        public OpId reqId;
        public PendingReq(int c0, int c1, OpId id) {
            this.c0 = c0;
            this.c1 = c1;
            reqId = id;
        }
    }
}
```
• agent goals:
  – repeatedly eating and thinking

• agent interaction with the table
  – using the table to get and release the chopsticks
  – sensing table observable events as perceptions

```
waiting_chopsticks_timeout(1000).
+start_living(C1,C2): true
  <- getArtifactId('theTable', TableID);
  linkSensor('DefaultSensor', mySensorID),
  +my_chopsticks(C1,C2, TableID);
  +time_to_think.
+time_to_think: not(hungry)
  <- +hungry.
+hungry:
  my_chopsticks(C1,C2, TableID) &&
  not got_chopsticks(C1,C2) &&
  not chopsticks_requested(C1,C2)
  <- execOp(TableID, getChops(C1,C2), mySensorId);
  +chopsticks_requested(C1,C2);
  ?waiting_chopsticks_timeout(DT),
  sense(mySensorId, DT).
+perception(mySensorId, chops_acquired):
  chopsticks_requested(C1,C2)
  <- +got_chopsticks(C1,C2).
+exception(sensing_timeout(SensorID)):
  chopsticks_requested(C1,C2)
  <- +starved.
+got_chopsticks(C1,C2):
  chopsticks_requested(C1,C2)
  <- -chopsticks_requested(C1,C2);
  !eat(C1,C2);
  !release_chopsticks(C1,C2);
  +time_to_think.
+:eat(C1,C2):
  hungry && got_chopsticks(C1,C2)
  <- -hungry.
+:release_chopsticks(C1,C2):
  not hungry && got_chopsticks(C1,C2) &&
  my_chopsticks(C1,C2, TableID)
  <- execOp(TableID, releaseChops(C1,C2));
  -got_chopsticks(C1,C2).
```
CONCLUSION

• New way to look to MAS environment
  – working environments

• A&A basic meta-model
  – modelling and exploiting artifact-based working environments

• CARTAGO framework / infrastructure
  – first prototyping technologies for creating and exploiting A&A artifact-based working environments