Coordination in Multi-agent Systems

Elena Nardini    Andrea Omicini

Alma Mater Studiorum – Università di Bologna
{elena.nardini, andrea.omicini}@unibo.it

November 20, 2008
A Notion

- An **agent** is an encapsulated computer system *situated* in some environment and capable of *flexible, autonomous action* in that environment in order to meet its *design objectives* [?].

- A **Multi-agent System (MAS)** contains a number of agents that interact (cooperate, coordinate or negotiate) with one another either:
  - to achieve a common objective or
  - because this is necessary for them to achieve their own objectives.
The Typical Structure of MASs [?]
Environment as a First-class Abstraction

- The agent environment should be a first-class abstraction entity [?].
  - Structures the MAS;
  - Is responsible of the activity in the MAS.

- This is even more true if we have to engineer modern software systems that are characterised by [?]:
  - A lot of interconnected computing systems (components);
  - Situatedness;
  - Openness;
  - Locality in control;
  - Locality in interactions.

- Ruling inter-agent and agent-environment interactions is an environment concern [?].
Coordination Meta-model

- Coordination is the activity of managing/costraining the agent interaction space, through the adoption of first class abstractions, from design to runtime.

- These first class abstractions (coordination media), enable interaction (including communication) among agents (the coordinables), and govern it by means of suitable coordination laws [?].
Data Vs. Control Driven [?] 

Data / information-driven models

- Coordinables synchronise, cooperate, compete by accessing, consuming and producing information available in the *shared data spaces.*

Control-driven models

- Coordinables synchronise, cooperate, compete by generating and reacting to signals / events in ports connected by *channels.*
Data-driven Models [?]

Almost all coordination model belonging to this category have evolved around the notion of a shared dataspace [?].

A shared dataspace is a common, content-addressable data structure. All processes involved in some computation can communicate among themselves only indirectly via this medium.

⇒ Data-driven models seems to be suitable for open applications, where a number of possibly a-priori unknown and autonomous entities have to cooperate.

⇒ Data-driven models better fit our application scenario than control-driven ones.
A Notable Example: Tuple Space Model

- Coordination medium: **Tuple Space.**
  - Multiset / bag of data object / structures called **tuples.**

- Communication Language: **Tuples.**
  - Tuple = ordered collection of (possibly heterogeneous) information items.

- Coordination Language: set of operations to put and retrieve associatively tuples to / from the space.
A coordination language is the materialisation or the linguistic embodiment of a coordination model.

⇒ It offers syntactical means with which a coordination model can be used for implementing an application.

Linda is a coordination language for shared tuple space coordination model.
Linda Coordination Language II [?]

- Communication Language
  - Tuple, Templates (anti-tuples), and Tuple Matching.

- Coordination Primitives
  - $\text{out}(T)$
    - Puts in the space the tuple $T$.
  - $\text{in}(TT)$
    - Removes from the space a tuple matching the template $TT$.
  - $\text{rd}(TT)$
    - Reads (without removing) from the space a tuple matching the template $TT$. 
Languages Ortogonality

- Separation of coordination and computation languages that makes a program easier to be implemented and understood.

Generality

- The same general purpose coordination language can be used in different coordination contexts, gluing different kind of computation.

⇒ Heterogeneity

- Gluing computation of heterogeneous computational models, all in the same coordination context.

⇒ Portability / Reusability

- Reusability in reusing application, implementation, tools and heterogeneous programmer expertise in the same coordination context.
Generative Communication

- Until explicitly withdrawn, the tuples generated by coordinables have an independent existence in the tuple space. A tuple is equally accessible to all the coordinable, but is bound to none.
  - Both senders and receivers can interact even without having prior knowledge about each other.
  - Space uncoupling.
  - Time uncoupling.

Associative Access

- Accessing tuples through content, not through address.
Towards An Hybrid Approach

- Linda-like coordination lacks the flexibility typical of direct and control-driven coordination models.
  - Both agent-to-agent coordination and access to local data are bound by the built-in-data-access mechanisms integrated in the shared dataspace.

- Any coordination policy not directly supported by the model has typically to be charged on agents, which are forced to implement in their code the coordination protocol required.
  - It increases the agent complexity.
  - It makes the coordination rules distributed between shared dataspaces and the agents.
  - It breaks the logical separation between coordination and algorithmic issues.
TuCSoN Coordination Model

- **Tuple Centre Spread over the Network** [?].
- It starts from the above considerations, by proposing a notion of an associative shared dataspace whose behaviour can be tailored according to the specific application needs.
  - From *tuple spaces* to *tuples centres* [?].
  - Tuple centres are *programmable* tuple spaces.
    - Programmable coordination media.
    - The coordination model it is the same.

- Tuple centres are distributed over the network, collected in *nodes*.
  - Distributed coordination media.
TuCSoN Coordination Space

- Set of distributed nodes.
  - Each TuCSoN node is an Internet node identified by the IP (logic) address.
- TuCSoN topology.
  - Here, Internet topology.
  - HiMAT [?]: hierarchical, dynamic, configurable topology.
Each TuCSoN node defines a coordination context, providing an open / dynamic set of tuple centres as coordination media.

- Identified by means of a logic name (term).
  Ex: `mail(aricci),
       room('2.3'),
       ticket_dispenser, ...`

- Full tuple centre identifier:
  `<name>@<node>.
   Ex: `mail(aricci)@myhome.org,
        room('2.3')@ingce.unibo.it,
        ticket_dispenser@137.204.191.188, ...`
In order to access and use the tuple centres of a node, an agent must enter the coordination context, either logically or physically (mobile agents).

**Agent Coordination Context (ACC)** [?].
Tuple Centres Features I

- **Programmable**

  - Tuple centre behaviour can be programmed to enact the desired coordination policies.

  - **ReSpecT** is an example of programming language for specification of the behaviour.
    
    - It programs as a set of logic tuples (reactions, first order logic terms such as Prolog terms) specifying medium behaviour reacting to interaction events.

  - Tuple centres as a general purpose coordination media customisable by means of a specification language like ReSpecT.
**Tuple Centres Features II**

- **Adaptable at runtime**
  - Tuple centre behaviour can be changed / adapted dynamically, at runtime, by reprogramming the coordination media.

- **Locality / encapsulation**
  - Tuple centres embed coordination laws.
    - A tuple centre can be a full-fledged coordination abstraction.
  - Reaction model ensure encapsulation of low-level coordination policies.
Tuple Centres Features III

- Inspectable at runtime
  - Tuple centre behaviour can be inspected dynamically, at runtime.

- Uniformity of languages
  - Same structure / primitives for communication and coordination

```plaintext
reaction(in(q(X)),
   no_r(q(X)),
   rd_r(p(Y)),
   out_r(q(Y)) ))
```
Simple Examples I

Current behaviour of the tuple centre (pseudocode):

When a tuple $T$ is inserted, produce a tuple $\text{backup}(T)$

When a tuple $p(X)$ is inserted, update the tuple $\text{total\_tuple}(N)$ (retrieve and store the tuple with $N$ incremented)...

in ReSpecT:

reaction(out(T), (out_r(backup(T)))).

reaction(out(p(X)), (in_r(\text{total\_tuple}(N)), N1 is N + 1, out_r(\text{total\_tuple}(N1)))).
Simple Examples II

Tuple Centre $T$

$q(5)$

$\text{n\_hits (303)}$

agent $A$

$\text{in(q(X))}$

**Tuple centre behaviour:**

When an $\text{in}$ operation is executed with template $q(X)$, update the tuple $\text{n\_hits}$

$\text{n\_hits (304)}$

$q(5)$

**in ReSpecT:**

reaction($\text{in(q(X))}$),

\[
\begin{align*}
\text{in}_r(\text{n\_hits}(Y)),
\text{out}_r(\text{n\_hits}(Y+1)).
\end{align*}
\]
TuCSoN Technology I

- **TuCSoN API**
  - Virtually any hosting language, currently Java and Prolog.
    - Support for Java and Prolog agents.
  - Heterogeneous hardware support.

- **TuCSoN Service**
  - Booting the TuCSoN Service daemon.
    - The host becomes a TuCSoN node.
    - With current version (1.4.5):
      ```
      ```
TuCSoN Technology II

- **TuCSoN Tools**
  - **Inspector**
    - Fundamental tool to monitor tuple centre communication and coordination state, and to debug tuple centre behaviour.
    - With current version (1.4.5):
      ```
      java -cp dir/tucson.jar alice.tucson.tools.Inspector.
      ```
  - **TuCSoN Shell**
    - Shell interface for human agents.
    - With current version (1.4.5):
      ```
      java -cp dir/tucson.jar alice.tucson.tools.CLIAgent.
      ```

- **TuCSoN technology is freely available in**
  http://alice.unibo.it/xwiki/bin/view/TuCSoN/
TuCSoN on the Fly

- Booting a TuCSoN node.
- Using a tuple centre (as a human agent) by exploiting TuCSoN shell node.
- Inspecting and debugging tuple centres by exploiting TuCSoN inspector.
Development in TuCSoN

- Building simple systems
  - Experiments with the "Hello world" simple Java agent.
  - Creating simple coordination among Java, human and Prolog agents.
import alice.tucson.api.*;
import alice.logicTuple.*;

public class Test {
    public static void main(String[] args) throws Exception {
        TucsonContext cn = Tucson.enterDefaultContext();
        TupleCentreId tid = new TupleCentreId("test_tc");
        cn.out(tid, new LogicTuple("p", new Value("hello world")));
        LogicTuple t = cn.in(tid, new LogicTuple("p", new Var("X")));
        System.out.println(t);
    }
}
import logictuple.*;
import tucson.api.*;

class Test2 {
    public static void main(String args[]) throws Exception{
        AgentId aid = new AgentId("agent-0");
        TucsonContext cn = Tucson.enterContext(new DefaultContextDescription(aid));

        // put the tuple value(1,38.5) on the temperature tuple centre
        TupleCentreId tid = new TupleCentreId("temperature");
        LogicTuple outTuple = LogicTuple.parse("value(1,38.5)"");
        cn.out(tid, outTuple);

        // retrieve the tuple using value(1,X) as a template
        LogicTuple tupleTemplate = new LogicTuple("value",
            new Value(1),
            new Var("X"));
        LogicTuple inTuple = cn.in(tid, tupleTemplate);

        cn.exit();
        System.out.println(inTuple);
    }
}
TuCSoN in Java III

```java
import alice.tucson.api.*;
import alice.logictuple.);

class MyAgent extends Agent {
    protected void body()
    {
        try {
            TupleCentreId tid = new TupleCentreId("test_tc");
            out(tid, new LogicTuple("p", new Value("hello world"))); 
            LogicTuple t = in(tid, new LogicTuple("p", new Var("X"))); 
            System.out.println(t);
        } catch (Exception ex) {
        }
    }
}

public class Test {
    public static void main(String[] args) throws Exception {
        AgentId aid = new AgentId("alice");
        new MyAgent(aid).spawn();
    }
}
```
TuCSoN in Prolog (tuProlog)

:- load_library('alice.tuprologx.lib.TucsonLibrary').
:- solve(go).

go:-
    test_tc ? out(p('hello world')),
    test_tc ? in(p(X)),
    write(X), nl.
Remind

- Ruling inter-agent and agent-environment interactions is an environment concern [?].

⇒ TuCSoN it is a part of agent environment.

- Until now, we have seen TuCSoN as an infrastructure supporting inter-agent interactions ...

- ... but we can also see TuCSoN as an infrastructure supporting agent-environment interactions.
  - Internal environment (work environment).
  - External environment (see Situated ReSpecT [?]).
We call context the physical / virtual and social situation in which an agent is situated [?].

⇒ In open world components need some form of context awareness in order to interact with both other agents and environment.

When an agent enters in a new context, the environment should provide it with a sort of control room that provides agents with context awareness [?].

▶ Is the only way in which the agent can perceive the environment as well as ...
▶ ... the only way in which the agent can interact.

⇒ It is possible to scale with openness of modern software systems.
⇒ While the environment manages social coordination, the control room manage coordination of the particular agent.
Agent Coordination Context (ACC) [?] 1

- Should be works as a model for the agent environment, by describing the environment where an agent can interact.
  - **Subjective viewpoint**: an ACC should provide agents with a suitable representation of the environment where they live, interact, and communicate.
  - **Objective viewpoint**: an ACC should provide a framework to express the interaction within a MAS as a whole, i.e. the space of MAS interaction, that is, the admissible interactions occurring among the agents of a MAS, and between the agents of a MAS and the MAS environment.
Agent Coordination Context (ACC) [?] II

- Should enables and rules the interactions between the agent and the environment by defining the space of the admissible agent interactions.
  
  - **Subjective viewpoint:** the coordination context enables in principle agents to perceive the space where they act and interact, reason about the effect of their actions and communications, and possibly affect the environment to accomplish their own goals.

  - **Objective viewpoint:** the coordination context would allow engineers to encapsulate rules for governing applications built as agent systems, mediate the interactions amongst agents and the environment, and possibly affect them so as to change global application behaviour incrementally and dynamically.
Agent Coordination Context (ACC) [?] III

- Should be conceived not only as a tool for human designers, but also as a run-time abstraction provided as a service to agents by a suitable infrastructure.
  - agent model or behaviour is not constrained \textit{a priori}.

⇒ Two basic stages characterize the ACC dynamics [?]:

- \textbf{ACC negotiation}. An ACC is meant to be negotiated by the agents with the MAS infrastructure, in order to start a working session inside an organisation, specifying which roles to activate.

- \textbf{ACC use}. The agent then can use the ACC to interact with the organisation environment, by exploiting the actions / perceptions enabled by the ACC.
Agent Coordination Context (ACC) [?] IV

- Should be dynamically configurable and inspectable by both agents and humans.
  - Configurability would allow a MAS to evolve at run time, by suitably adapting its behaviour to changes.
  - Inspectability would allow both humans and intelligent agents to reason about the current laws of coordination as represented and embodied within coordination contexts, and to possibly change them by properly reconfigure coordination contexts according to new application needs.
ACC framework is orthogonal both to the specific computational model(s) adopted to define agent behaviour, and to the interaction model(s) adopted to specify how agents communicate, and more generally, interact within the environment.

⇒ It is possible to extend any MAS infrastructure with the ACC framework in order to support the organisation and security features.

As minimum requirements, the infrastructure must explicitly define two different models:

- It must provide a model of interaction, expressing agent / perceptions (including eventually communication).
- It should specify a basic organisational model, at least including the notion of agent identity.
A general model of ACC can be defined, by describing three distinct aspects characterizing the ACC concept:

- **ACC Interface.** It defines the set of admissible operations provided by the infrastructure for interacting with the (social and resource) environment.

- **ACC Contract.** It is a description of the relationships between the agent and the (organisation) environment where the agent is playing, in particular of the policy enacted by the ACC ruling agent actions and interaction protocols.

- **ACC Configuration.** The ACC configuration is a description of the run-time state of the ACC, concerning the evolution of ongoing interaction protocols.
ACC can be exploited as a unifying abstraction to face a number of otherwise heterogeneous issues in the modelling and engineering of MASs where MASs are seen as structural / social settings. In particular:

- Modelling Organisation;
- Modelling Access Control;
- Modelling the Quality of Interaction;
- Modelling Relationships between Agents and Institutions.
**Modelling Organisation.** When engineering complex software systems by adopting agent-oriented abstractions, *organisation* emerges a fundamental dimension [?]. The ACC abstraction makes it possible to explicitly model the presence of an agent in an organisational context where specific structures and rules are defined.

**Modelling Access Control.** Agent autonomy and system openness are among the main features that make the engineering of security particularly challenging in the context of MASs [?]. The governing behaviour enacted by the ACC on the agent actions makes this abstraction suitable to model forms of dynamic access control to environment resources.
ACC as a Unifying Abstraction for Organisation and Security [?] III

- **Modelling the Quality of Interaction.** As an interface, the ACC is the conceptual framework place where non-functional properties related to the quality of the interaction / communication can be suitably modelled [?].

- **Modelling Relationships between Agents and Institutions.** The ACC represents a contract between the agent and the institution (organisation) that released it.
Conceiving and representing different issues exploiting a unified abstraction have several benefits. In particular:

- Conceptual economy is obviously the first benefit.
- A common framework is the most obvious way to consistently support adaption and evolution of such issues.
- There are system aspects that can be modelled and engineered in their complex articulation only by considering such issues at the same time.
Experiments in TuCSoN Infrastructure

- In [?] TuCSoN was extended in order to deal with security and topology issues.
- The access control model adopted, however, was unrelated from organisation specification and management.
  ⇒ In [?] the previous approach was integrated with RBAC-like architecture, by explicitly considering access control as linked to organisation structures and rules.
Coordination in Multi-agent Systems

Elena Nardini       Andrea Omicini

Alma Mater Studiorum – Università di Bologna
{elena.nardini, andrea.omicini}@unibo.it

November 20, 2008