Tuple Centre Spread over the Network (TuCSoN)

Laboratory of Distributed Systems

Elena Nardini

Alma Mater Studiorum – Università di Bologna
elena.nardini@unibo.it

May 16, 2011

Outline

Coordination Meta-model [Ciancarini, 1996]

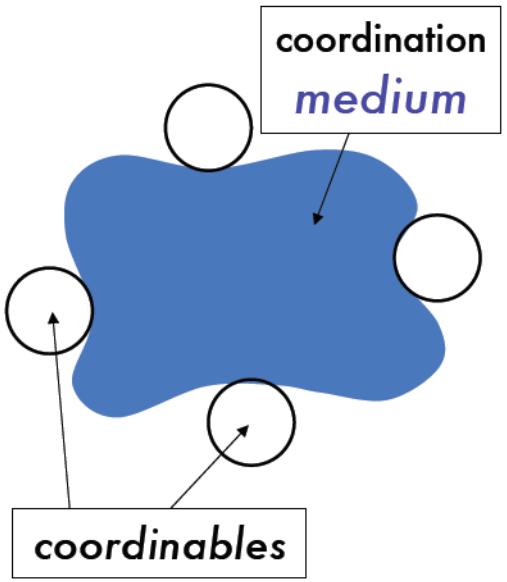
- Interaction is one of the main sources of complexity in the software system engineering

→ Coordination is the activity of managing/costraining the interactions occurring dynamically among software components

→ Coordination models and languages are a set of approaches for the engineering of a system interaction space based on the following abstractions:

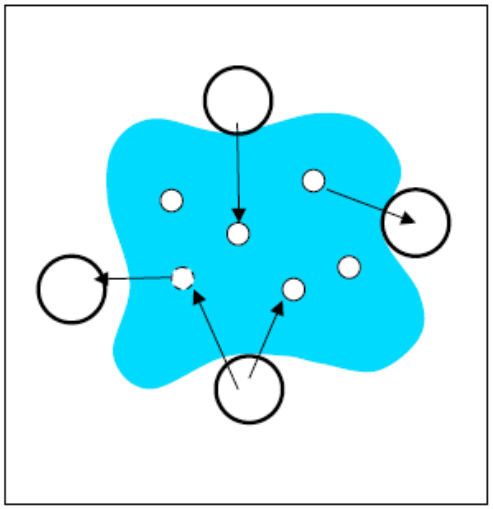
  - Coordination media enabling interaction among
  - coordinables – system components to be coordinated – by means of suitable
  - coordination laws

and on communication languages – syntax used to express and exchange data structures – and coordination languages—linguistic embodiements of the coordination model

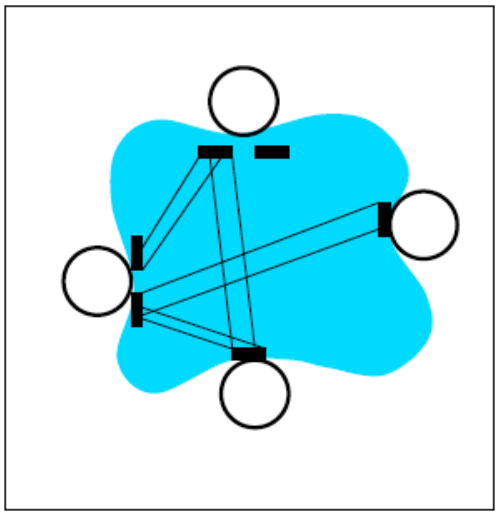


Data vs. Control Driven [Papadopoulos and Arbab, 1998]

- 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 [Papadopoulos and Arbab, 1998]

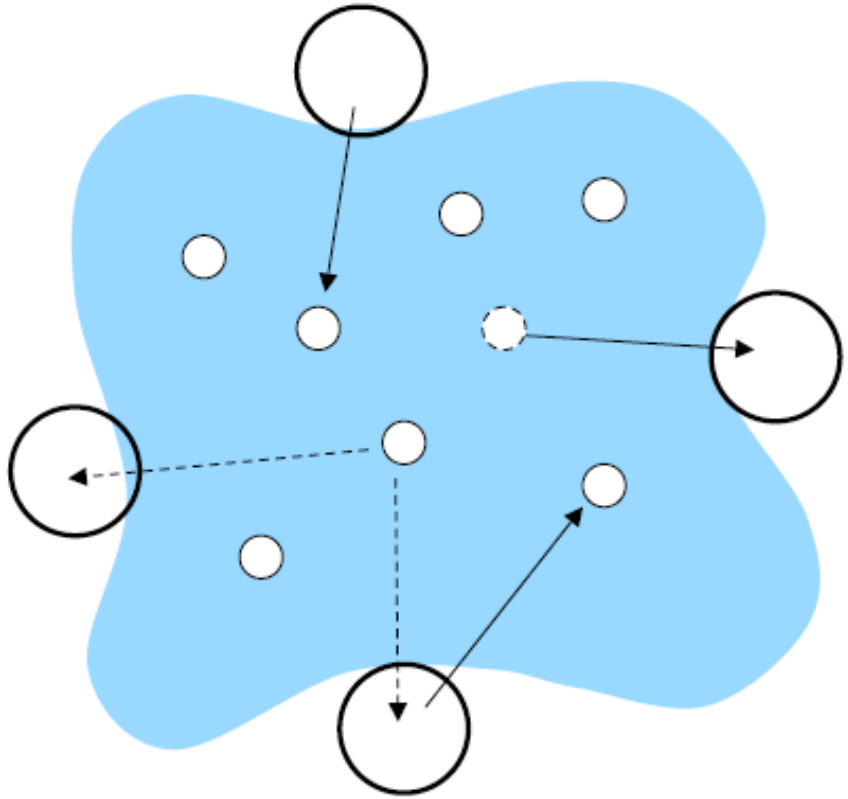
- Almost all coordination models belonging to this category have evolved around the notion of a shared data-space [Roman and Cunningham, 1990]
- A shared data-space is a common, content-addressable data structure. All processes involved in some computation can communicate among themselves only indirectly via this medium

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

$\Rightarrow$ Data-driven models better fit our application scenario than control-driven ones

A Notable Example: The Tuple Space Model

- Coordination medium: Tuple Space
  - Multiset/bag of data objects/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



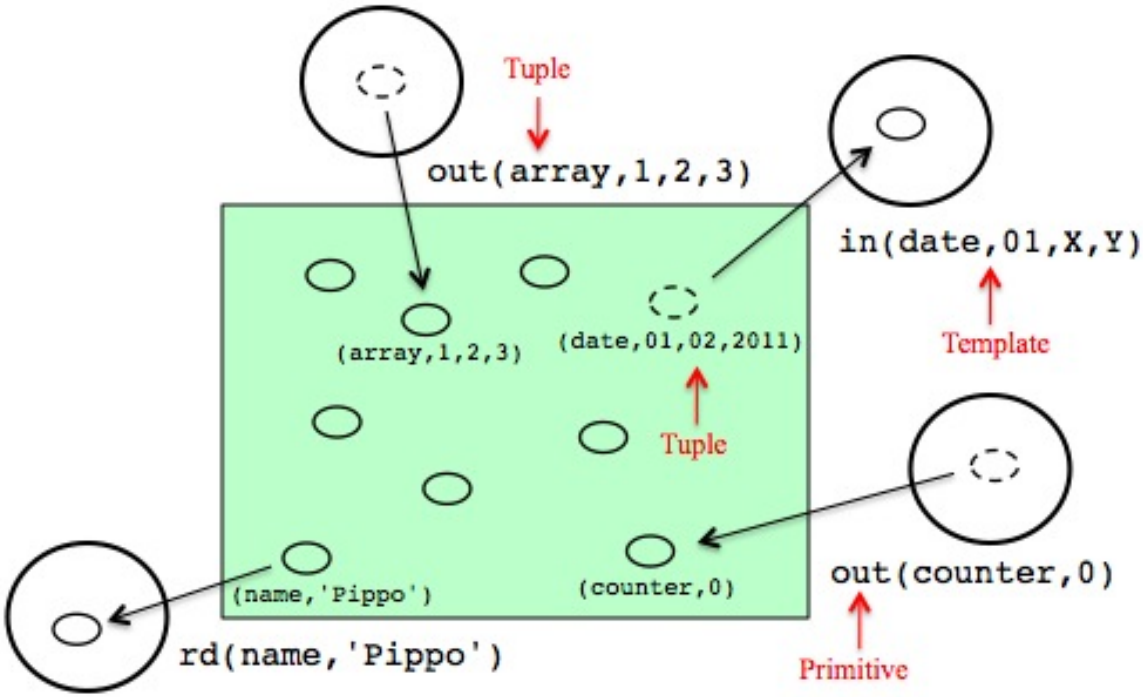
Linda Coordiantion Language [Gelernter, 1985a]

- A coordination language is the materialisation or the linguistic embodiment of a coordination model [Gelernter and Carriero, 1992]
  - ⇒ 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
- Communication Language
  - Tuple, Templates (anti-tuples), and Tuple Matching
- Coordination Primitives
  - `out(T)`
    - ⇒ Puts in the space the tuple `T`
  - `in(TT)`
    - ⇒ Removes from the space a tuple matching the template `TT`
  - `rd(TT)`
    - ⇒ Reads (without removing) from the space a tuple matching the template `TT`

Linda Features

- Generative communication [Gelernter, 1985b]: tuples are permanently written in a tuple space
  - * Promotes communication uncoupling—name, space and time uncoupling
  - → Supports openness in distributed systems
- Associative access: in order to read/consume a tuple, a tuple set description has to be specified
  - * Promotes knowledge-based coordination
  - → Fits well with knowledge-intensive systems

Example


Tuple
out(array,1,2,3)
in(date,01,X,Y)
Template
(array,1,2,3)
(date,01,02,2011)
Tuple
(name,'Pippo')
(counter,0)
out(counter,0)
rd(name,'Pippo')
Primitive

Outline

Tuple Centre Spread over the Network (TuCSoN) [Omicini and Zambonelli, 1999]

- Infrastructure providing services enabling the communication and coordination of distributed/cuncurrent independent software components (agents)
- It is based on programmable tuple spaces called tuples centres [Omicini and Denti, 2001]
- Each tuple centre provides the following primitives:

out to put the specified tuple in the specified tuple centre

in to remove a tuple matching with the specified template from the specified tuple centre

rd to read a tuple matching with the specified template from the specified tuple centre

inp to remove, if present, a tuple matching with the specified template from the specified tuple centre (if not present, the operation fails)

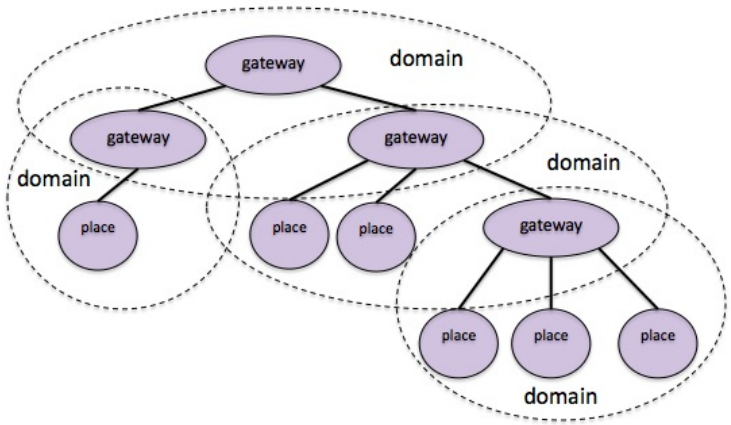
rdp to read, if present, a tuple matching with the specified template from the specified tuple centre (if not present, the operation fails)

Tuple Centres Features

- Programmable
  - Tuple centre behaviour can be programmed to enact the desired coordination policies
  - $\Rightarrow$ Tuple centres as a general purpose coordination media customisabled
- Locality/encapsulation
  - Tuple centres embed coordination laws
  - $\Rightarrow$ A tuple centre can be a full-fledged coordination abstraction
- Inspectable at runtime
  - Tuple centre behaviour can be inspected dinamically, at runtime
- Adatable at runtime
  - Tuple centre behaviour can be changed/adapted dynamically, at runtime, by reprogramming the coordination media
- Uniformity of languages
  - Same structure/primitives for communication and coordination

The TuCSoN Topology [Cremonini et al., 1999]

- Tuple centres are distributed over the network hosted in nodes and organised into articulated domains
- A domain is characterised by
  - a gateway node that is meant to host tuple centres used for domain administration, keeping information on the places ($ORG tuple centre)
  - a set of nodes called places, where a place is meant to host tuple centres for the specific applications/systems



- TuCSoN is an infrastructure for Internet environment
  - It is possible to access to remote tuple centres identifying them by means of their complete identifier:

    <Tuple Centre Name> @ *<Tuple Centre IP Address>*

    Examples:

    temperature@'137.204.192.175'

    temperature@'deis.unibo.it'

    temperature@localhost

The TuCSoN Organisations

- Tuple centres are structured and ruled in organisations mapped in domains
- Agents can be thought as a sort of society (permanent or temporal) with a specific objectives
- An organisation can be conceived as a static as well as dynamic set of societies
- Such societies are composed by agents playing some roles
- Organisations modelled through the RBAC model [Omicini et al., 2005]

The TuCSoN Context

- We call context *the physical/virtual and social situation in which an agent is situated* [Moran and Dourish, 2001]

  ⇒ In open world agents need some form of context awareness in order to interact with a system (other agents and resources) [Mamei and Zambonelli, 2006]

- When an agent enters in a new context, the system should provide it with a sort of control room that provides agents with context awareness [Omicini, 2002]
  - Is the only way in which the agent can perceive the system as well as ...
  - ... the only way in which the agent can interact

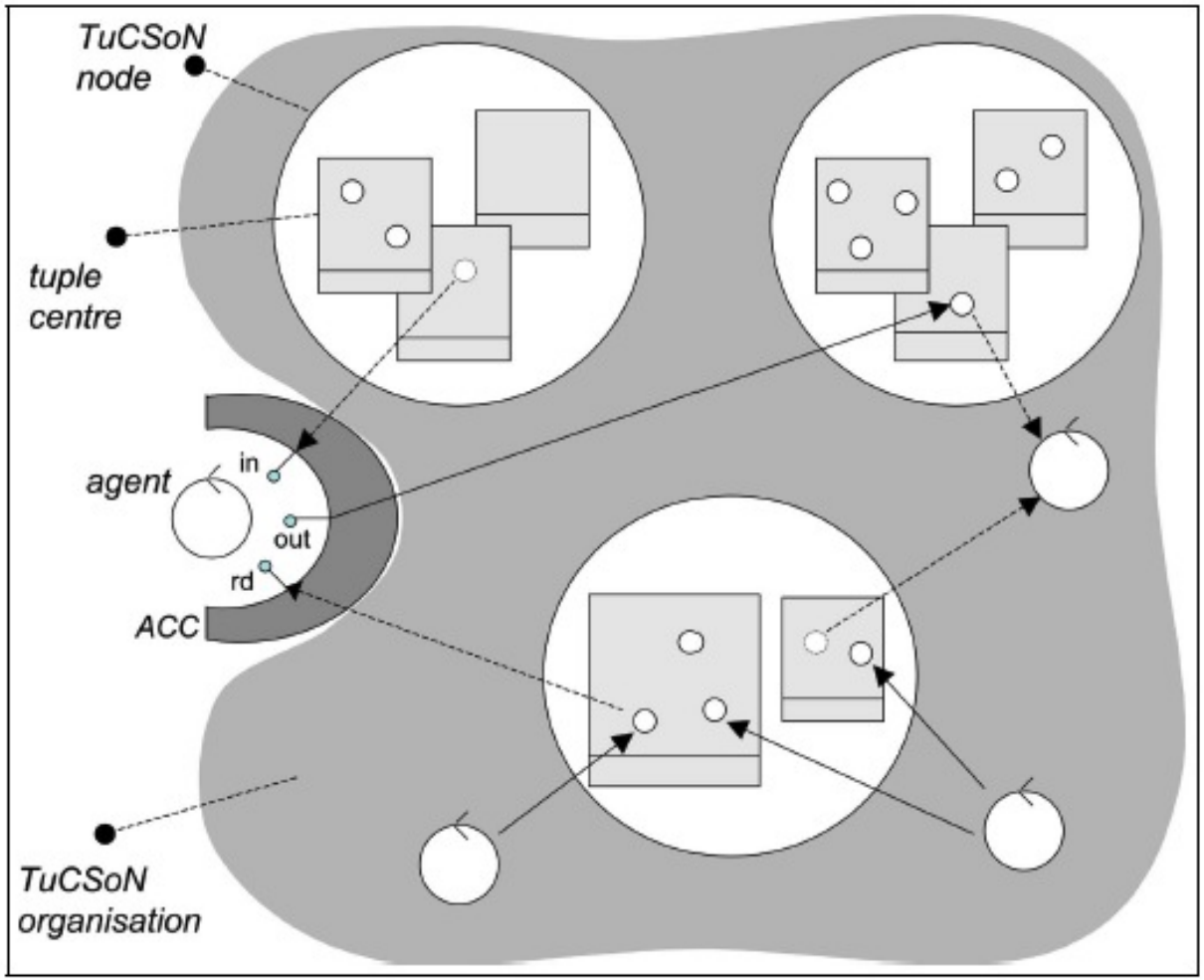
⇒ While the system manages social coordination, the control room manage coordination of the particular agent

⇒ It is possible to scale with openness of modern software systems

The Agent Coordination Context (ACC) [Omicini, 2001]

- Each TuCSoN organisation defines a coordination context, providing an open/dynamic set of tuple centres as coordination media
- Each agent has its own view of the context that is called Agent Coordination Context (ACC)
  - Works as a model for the agent system, by describing the system where an agent can interact
  - Enables and rules the interactions between the agent and the system by defining the space of the admissible agent interactions
- In order to access and use the tuple centres of a domain, an agent must
  - negotiate the ACC in order to start a working session inside an organisation, specifying which roles to activate
  - use the ACC to interact with the system by exploiting the actions enabled by the ACC

The TuCSoN Coordination Model



Outline

TuCSoN Technology

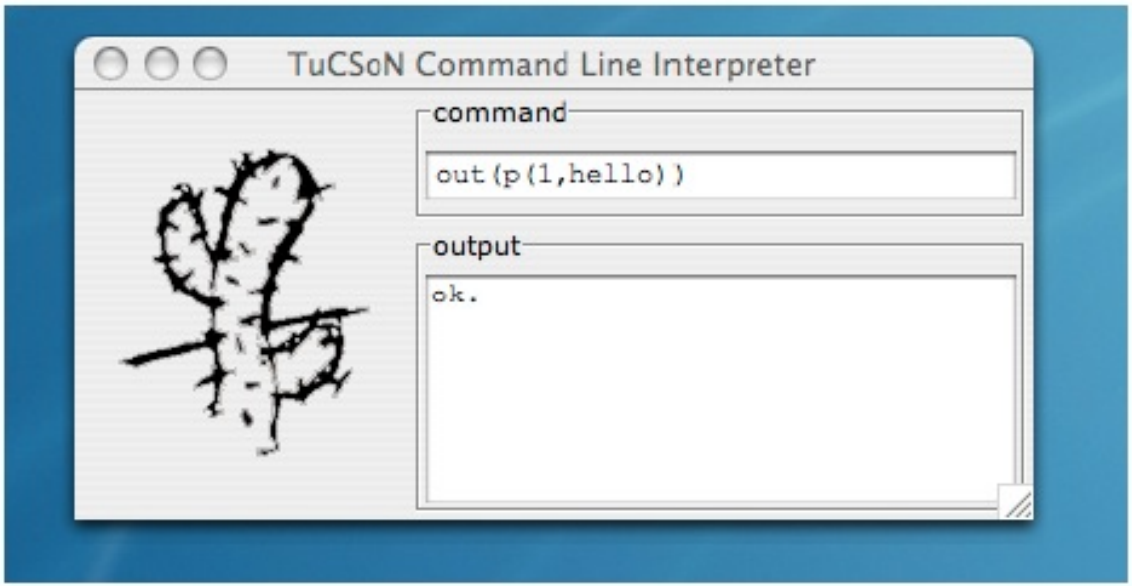
- TuCSoN API
  - Virtually any hosting language, currently Java and Prolog
    - $\Rightarrow$ Support for Java and Prolog agents
  - Heterogeneous hardware support
- TuCSoN Service
  - Booting the TuCSoN Service daemon
    - ★ The host becomes a TuCSoN node
    - ★ With current version (1.9.1) run: `alice.tucson.service.Node`

TuCSoN Tools

- TuCSoN Tools
  - Inspector
    - ★ Fundamental tool to monitor tuple centre communication and coordination state, and to debug tuple centre behaviour
    - ★ With current version (1.9.1) run: `alice.tucson.tools.Inspector`
  - TuCSoN Shell
    - ★ Shell interface for human agents
    - ★ With current version (1.9.1): `alice.tucson.tools.CLIAgent`
- TuCSoN technology is freely available in `http://alice.unibo.it/xwiki/bin/view/TuCSoN/`

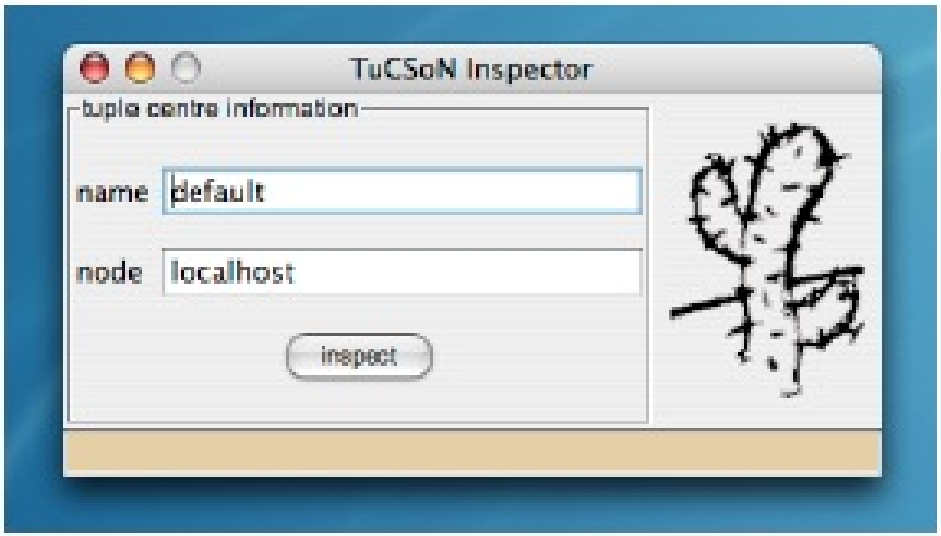
TuCSoN on the Fly: CLI Agent

- Booting a TuCSoN node
- Using the CLIAgent (as a human agent) by exploiting TuCSoN tuple centre default



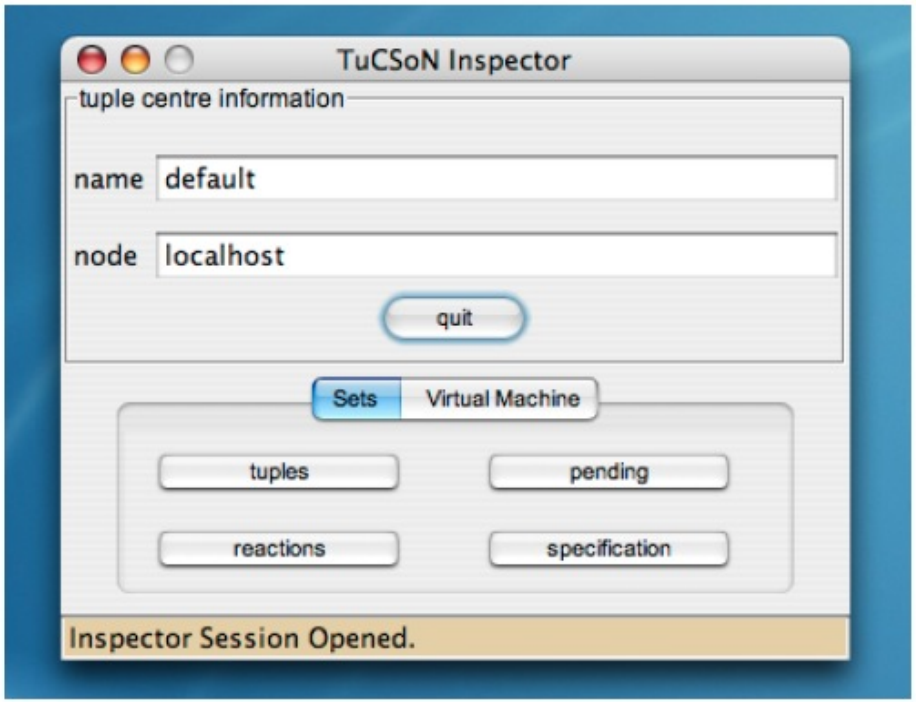
TuCSoN on the Fly: Inspector (1)

- Inspecting and debugging tuple centres by exploiting TuCSoN Inspector



TuCSoN on the Fly: Inspector (2)

- Inspecting and debugging tuple centres by exploiting TuCSoN Inspector



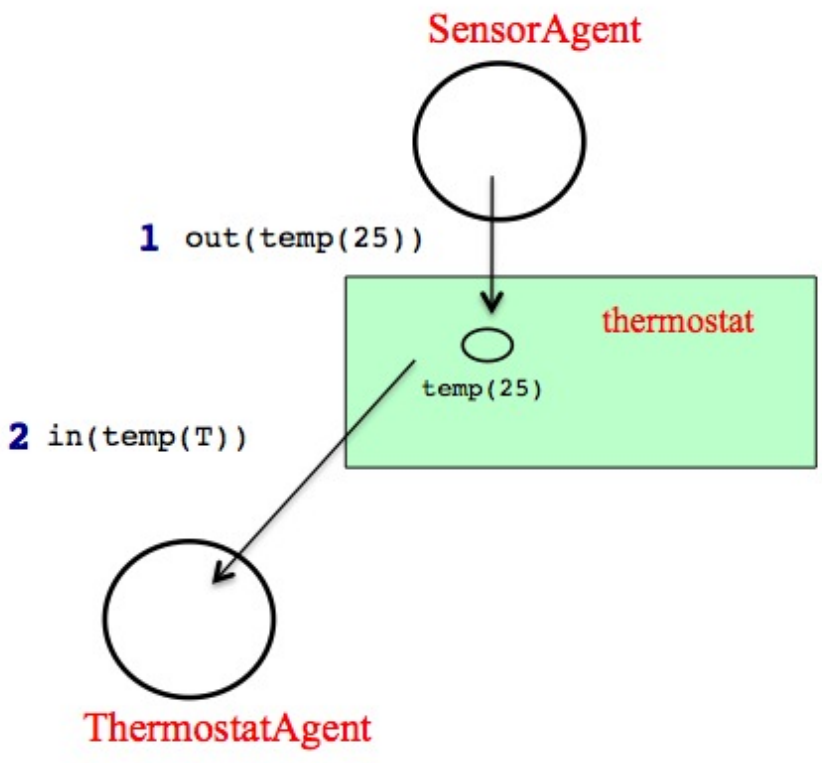
Outline

Exercise 1: 'Hello World' Example

```
package alice.tucson.examples.basic;

import alice.logictuple.LogicTuple;

public class HelloWorld
{
    public static void main(String[] args) throws Exception
    {
        TucsonAgentLanguageManager.useLanguage(TucsonLanguageManager.DEFLANGUAGE);

        TucsonTupleCentreId tid = null;

        if (args.length == 0)
            tid = new TucsonTupleCentreId("default");
            tid = new TucsonTupleCentreId(args[0]);

        TucsonContext cnt = Tucson.getContext(new TucsonAgentId("agent"),
                TucsonAgentLanguageManager.getCurLanguage());

        long now = System.currentTimeMillis();
        LogicTuple tuple = new LogicTuple("msg", new Value("Hello world!"),
                new Value("time", new Value(now)));

        cnt.out(tid, tuple, (Long) null);
        System.out.println("Tuple inserted: " + tuple);

        LogicTuple template = new LogicTuple("msg", new Var("Msg"), new Var("Time"));
        LogicTuple msg = cnt.in(tid, template, (Long) null);

        System.out.println("Tuple retrieved name: " + msg.getName());
        System.out.println("Msg argument: " + msg.getArg(0));
        System.out.println("Time argument: " + msg.getArg(1).getArg(0));
    }
}
```

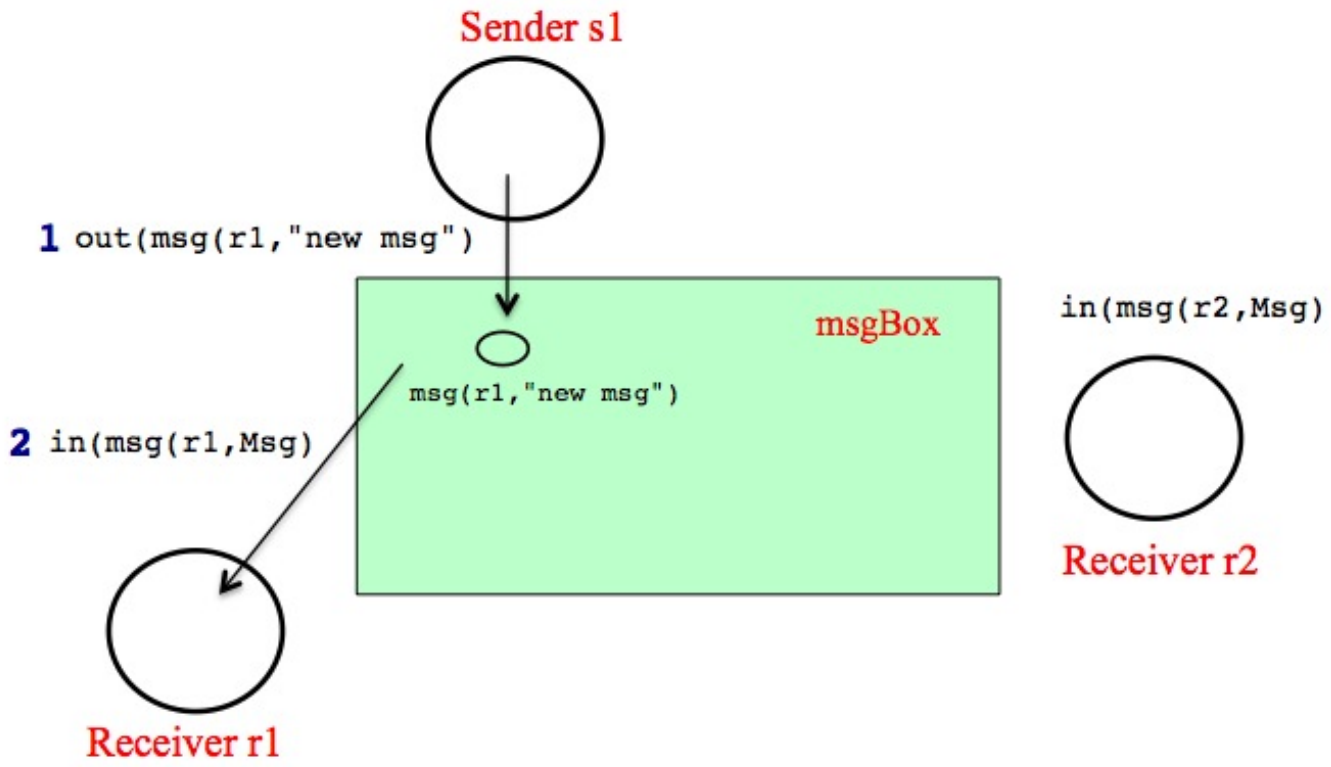
Exercise 2: The Thermostat

- Create the tuple centre thermostat
- Create the agent sensorAgent writing in the tuple centre the sensed external-temperature
- Create the agent thermostatAgent reading the sensed temperature.
  The temperature has to be ≥ 23 and ≤ 26.
  The agent's task is to adapt the external temperature in order to respect the constraint



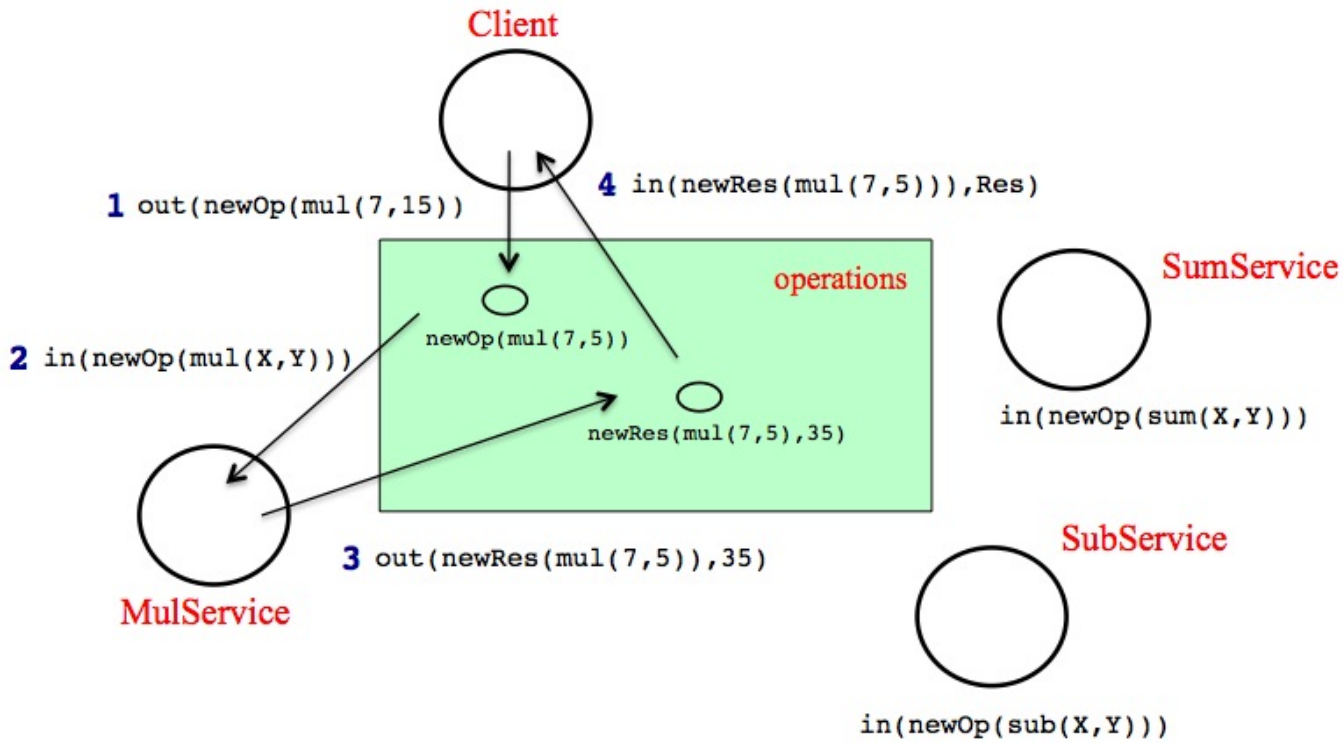
Exercise 3: Message-oriented Communication

- Create the tuple centre msgBox
- Create sender agents
- Create receiver agents



Exercise 4: Remote Procedure Calls and Services

- Create the tuple centre operations
- Create a client agent
- Create a service agent for each kind of operation (sum, mul and sub)



Exercise 5: Exploiting TucsonAgent

- Try to use the alice.tucson.apiTucsonAgent class

Outline

Bibliography I


Ciancarini, P. (1996).
Coordination models and languages as software integrators.
ACM Comput. Surv., 28(2):300–302.


Cremonini, M., Omicini, A., and Zambonelli, F. (1999).
Multi-agent systems on the Internet: Extending the scope of coordination towards security and topology.
In Garijo, F. J. and Boman, M., editors, *Multi-Agent Systems Engineering*, volume 1647 of *LNAI*, pages 77–88. Springer-Verlag. 9th European Workshop on Modelling Autonomous Agents in a Multi-Agent World (MAAMAW'99), Valencia, Spain, 30 June–2 July 1999. Proceedings.


Gelernter, D. (1985a).
Generative communication in linda.
ACM Trans. Program. Lang. Syst., 7(1):80–112.

Bibliography II


Gelernter, D. (1985b).
Generative communication in Linda.
ACM Transactions on Programming Languages and Systems, 7(1):80–112.


Gelernter, D. and Carriero, N. (1992).
Coordination languages and their significance.
Commun. ACM, 35(2):97–107.


Mamei, M. and Zambonelli, F. (2006).
Field-Based Coordination for Pervasive Multiagent Systems. Models, Technologies, and Applications.
Springer Series in Agent Technology. Springer.


Moran, T. and Dourish, P. (2001).
Introduction to this special issue on context-aware computing.
Human-Computer Interaction, 20(2–4):87–95.

Bibliography III


Omicini, A. (2001).
On the notion of agent coordination context: Preliminary notes.
*AI*IA Notizie*, XIV(4):44–46.


Omicini, A. (2002).
Towards a notion of agent coordination context.
In Marinescu, D. C. and Lee, C., editors, *Process Coordination and Ubiquitous Computing*, chapter 12, pages 187–200. CRC Press, Boca Raton, FL, USA.


Omicini, A. and Denti, E. (2001).
From tuple spaces to tuple centres.
Science of Computer Programming, 41(3):277–294.

Bibliography IV


Omicini, A., Ricci, A., and Viroli, M. (2005).
RBAC for organisation and security in an agent coordination infrastructure.
Electronic Notes in Theoretical Computer Science, 128(5):65–85.
2nd International Workshop on Security Issues in Coordination Models, Languages and Systems (SecCo'04), 30 August 2004. Proceedings.


Omicini, A. and Zambonelli, F. (1999).
Coordination for Internet application development.
Autonomous Agents and Multi-Agent Systems, 2(3):251–269.
Special Issue: Coordination Mechanisms for Web Agents.


Papadopoulos, G. A. and Arbab, F. (1998).
Coordination models and languages.
Technical report, Amsterdam, The Netherlands, The Netherlands.

Bibliography V


Roman, G.-C. and Cunningham, H. C. (1990).
Mixed programming metaphors in a shared dataspace model of concurrency.
IEEE Trans. Softw. Eng., 16(12):1361–1373.

Tuple Centre Spread over the Network (TuCSoN)

Laboratory of Distributed Systems

Elena Nardini

Alma Mater Studiorum – Università di Bologna
elena.nardini@unibo.it

May 16, 2011