Tuple Centres Spread over the Network (TuCSoN)

Laboratory of Multiagent Systems LM
Laboratorio di Sistemi Multiagente LM

Elena Nardini
elena.nardini@unibo.it

Ingegneria Due
Alma Mater Studiorum—Università di Bologna a Cesena

Academic Year 2010/2011
1 The TuCSoN Infrastructure

2 Examples

3 Exercises
   - Exercise 1
   - Exercise 2

4 Bibliography
Main Features

- Infrastructure providing services enabling the coordination of distributed/concurrent independent agents.
- Supports agent coordination providing **tuple centres** – shared & reactive information spaces – distributed over the infrastructure nodes.
- Agents insert, consume and read information in the form of **tuple**—ordered collections of heterogeneous information chunks.
Main Features

- Programmable tuple spaces
  - tuple spaces with a reactive behaviour which can be programmed dynamically
- Software components access tuple centres associatively by writing, reading, and consuming tuples via simple communication operations: `out`, `rd`, `in`, `inp`, and `rdp`

Generative communication

- Agents communicate by creating and retrieving associatively tuples whose existence – once created – is independent with respect to agents’ one.
  - This makes it possible to obtain uncoupling properties – temporal and spatial – leading to the openness requirement of software systems
The TuCSoN Infrastructure

Tuple Centres [Omicini and Denti, 2001]

Runtime First-class Abstractions [Ricci et al., 2005]

- **Malleability.** The coordination behaviour can be adapted and changed dynamically.
- **Inspectability.** The communication and the coordination state can be inspected at runtime.
- **Controllability.** The execution can be controlled by means of proper infrastructure tools.
Tuples are logic tuples: first order logic terms (like Prolog terms)
Differently from tuple spaces, the behaviour of tuple centres in response to communication events can be tailored to the application needs by defining a set of specification tuples expressed in the ReSpecT language [Omicini, 2007].
TuCSoN Topology

- Tuple centres are collected in infrastructure nodes, distributed over the network, organised into articulated domains.
- A domain characterised by a **gateway** node and a set of nodes called **places**.
  - A place node is meant to host tuple centres for specific applications/systems.
  - A gateway node is meant to host tuple centres used for domain administration, keeping information on the places (**$ORG$** tuple centre).
  - A place can belong to different domains, and can be itself a gateway for a sub-domain.
TuCSoN Organisations

- Tuple centres are structured and ruled in organisations.
- 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.
- Role-based access control (RBAC) to integrate organisation and security.

Elena Nardini (Università di Bologna)
Agent Coordination Context (ACC)

A runtime and stateful interface released to an agent to execute operations on the tuple centres of a specific organisation

A sort of interface provided to an agent by the infrastructure to make it interact within a certain organisation environment

An organisation abstraction to model RBAC in MAS
TuCSoN Node
TuCSoN Technology

TuCSoN API
- Java and Prolog
- Heterogeneous hardware support

TuCSoN Service
- The host becomes a TuCSoN Node
TuCSoN Tools

CLI-Agent Tool

- Shell interface for human agents
TuCSoN Tools

InspectorTool
- Fundamental tool to monitor tuple centre communication and coordination state, and to debug tuple centre behaviour
Inspector Tool
Inspector Tool

```plaintext
p(1,hello)
temperature('Cesena',17)
author(name('Charles'),surname('Bukowski'))
graph_node(id(0),value(0.5),links([1,4,6]))
```
Inspector Tool

Pending Query Set of default@localhost

vrn time: 109931937701
local time: 109931937748
items: 1

\( \text{in}(q(X)) \) from agent user('1099318461934', '127.0.0.1')

- Observation
- View
- Log
- Action

- get any new observation
- get only when update requested

update
Inspector Tool
Inspector Tool

```
reaction | out (factorial(N,_)),{
  in_r (factorial(N,_)),
  out_r (factorial_loop(1,N,1))}.

reaction | out_r (factorial_loop(N,N,F)),{
  in_r (factorial_loop(N,N,F)),
  out_r (factorial(N,F))}.

reaction | out_r (factorial_loop(I,N,F)),{
  in_r (factorial_loop(I,N,F)),
  N > I,
  I1 is I + 1,
  F1 is F * I1,
  out_r (factorial_loop(I1,N,F1))}.
```
Example 1: Hello World

```java
package alice.tucson.examples.basic;
import alice.tucson.api.*;

public class HelloWorld
{
    public static void main(String[] args) throws Exception
    {
        TupleCentreId tid = null;

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

        TucsonContext cnt = Tucson.enterDefaultContext();

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

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

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

        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));
    }
}
```
Example 2: Java Agent

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

public class MyAgent extends Agent {
    protected MyAgent(String name) throws TucsonException {
        super(name);
    }

    protected void body() {
        try {
            TupleCentreId tid = new TupleCentreId("test_tc");
            out(tid, new LogicTuple("p", new Value("hello world")));

            LogicTuple t = new LogicTuple("p", new Var("X"));
            System.out.println(t);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```
Example 3: Java Agent Test

```java
public class Test {
    public static void main(String[] args) throws Exception {
        new MyAgent("alice").spawn();
    }
}
```
Example 4: Prolog Agent

```prolog
:- load_library('alice.tucson.api.Tucson2PLibrary').
:- solve(go).

go:-
    enter_context([[agent_id(hank)]],
    test_tc ? out(p('hello world')),
    test_tc ? in(p(X)),
    exit_context, write(X), nl.
```
Outline

1. The TuCSoN Infrastructure
2. Examples
3. Exercises
   - Exercise 1
   - Exercise 2
4. Bibliography
Thermostat Agent in Java

Requirements
- Check the environment temperature \( T \).
- Until \( T \) is not: \( > 18 \) and \( < 22 \):
  - Decrease \( T \) of one unit if the temperature is 22
  - Increase \( T \) of one unit if the temperature is 18

Constraint
- **thermostat** is a tuple centre between the environment and the ThermostatAgent
- **ThermostatAgent** interacts with the tuple centre to communicate with the thermostat in order to sense and change the temperature
- The real thermostat is simulated by an agent
Outline

1 The TuCSoN Infrastructure

2 Examples

3 Exercises
   - Exercise 1
   - Exercise 2

4 Bibliography
Thermostat Agent in Prolog

New Constraint

- ThermostatAgent becomes a Prolog agent


Tuple Centres Spread over the Network (TuCSoN)
Laboratory of Multiagent Systems LM
Laboratorio di Sistemi Multiagente LM

Elena Nardini
elena.nardini@unibo.it

Ingegneria Due
Alma Mater Studiorum—Università di Bologna a Cesena

Academic Year 2010/2011