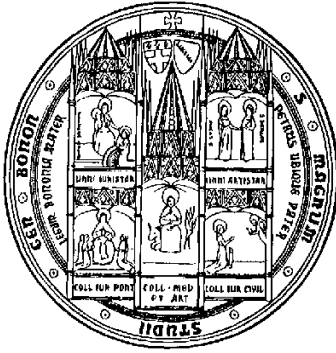




Università degli Studi di
Bologna

DEIS - Dipartimento di Elettronica,
Informatica e Sistemistica

A Coordination Infrastructure for Agent-based Internet Applications

Enrico Denti, Andrea Omicini

Agent-based Internet Applications

- Collections of distributed (possibly heterogeneous) *interactive agents* having to *coordinate* to achieve the application goal(s)
- Some Key Issues:
 - **Which methodologies to design such systems?**
 - **Which technologies for their development?**
 - Support for standard technologies (Java, DB connectivity,...)
 - but also new models / languages / technologies / tools...
specifically tailored to agent-based systems
(agent communication languages, knowledge representation frameworks, coordination models & technologies, ...)

Infrastructures for Agent-based Internet Applications

- *Infrastructures* are aimed at *factorizing properties* of classes of agent-based systems
- Need of coherent **abstractions, technologies** and **methodologies**
- Our viewpoint: **coordination** as a *basic service* to be supplied by infrastructures for agent-based systems.

The LuCe Infrastructure for Agent-based Internet Applications

- **The LuCe infrastructure's major ingredients:**
 - **Java** as the main standard Internet "glue technology"
 - **Logic-based Languages** to support intelligence and supply logic inference capabilities
 - **Tuple-based Coordination Models** to support agent coordination and the actual construction of Multi-Agent Systems (MAS)

The LuCe Infrastructure for Agent-based Internet Applications

- **The LuCe infrastructure's major ingredients:**

From the *methodology*
viewpoint
(not the focus of this paper!)

From the *model & technology*
viewpoints
(here's the focus of this paper!)

- **Tuple-based Coordination Models** to support agent coordination and the actual construction of Multi-Agent Systems (MAS)

LuCe base abstraction: the *(Logic) Tuple Centre*

- **Works like a Tuple Space..**
→ supplies all the advantages of tuple-based coordination models
- **..but its *behavior can be programmed***, achieving the control capability that is typical of control-driven coordination models
- **Adopts *logic tuples*** as
 - the agent communication language
 - the base for the behavior specification language, ReSpecT

LuCe base abstraction: the *(Logic) Tuple Centre* (cont'd)

- **Behavior Programming: the Basics**
 - Each coordination primitive performed on the Tuple Centre (**in**, **out**, **rd**, ...) causes *Communication Events*
 - Communication events can trigger *reactions*, expressed as ReSpecT logic tuples: `reaction (out (p (X) ) , ( ... ) ) .`
- **Behavior Programming: the Execution Model**
 - **Agents perceive only *the final effect* of**
 - the expected effect of the coordination primitive
 - all the reactions it triggered (directly & indirectly)
 - **the semantics of coordination acts, as perceived by agents, can be *user-defined*.**

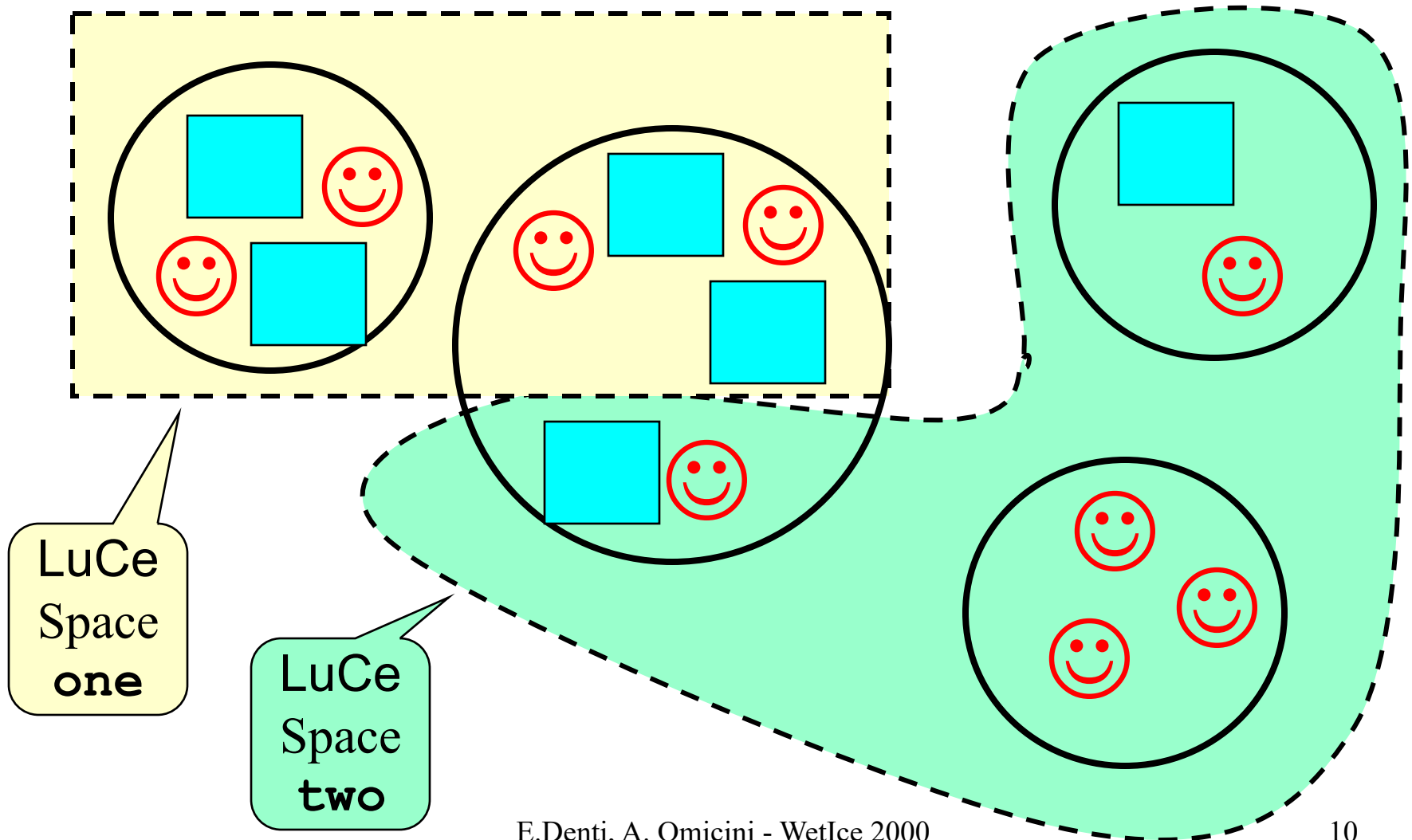
LuCe: impact on Design

- The **tuple centre** as a basic abstraction promotes *uncoupling dimensions* in the system design & development process...
 - *agents* designed *independently from each other*
 - *agent interaction protocols* designed *separately from agents and independently from each other*
 - *coordination media* "*gluing things together*"
- But *abstractions alone are not enough*
 - need an *IDE*
 - now emphasis on *development & support technologies*

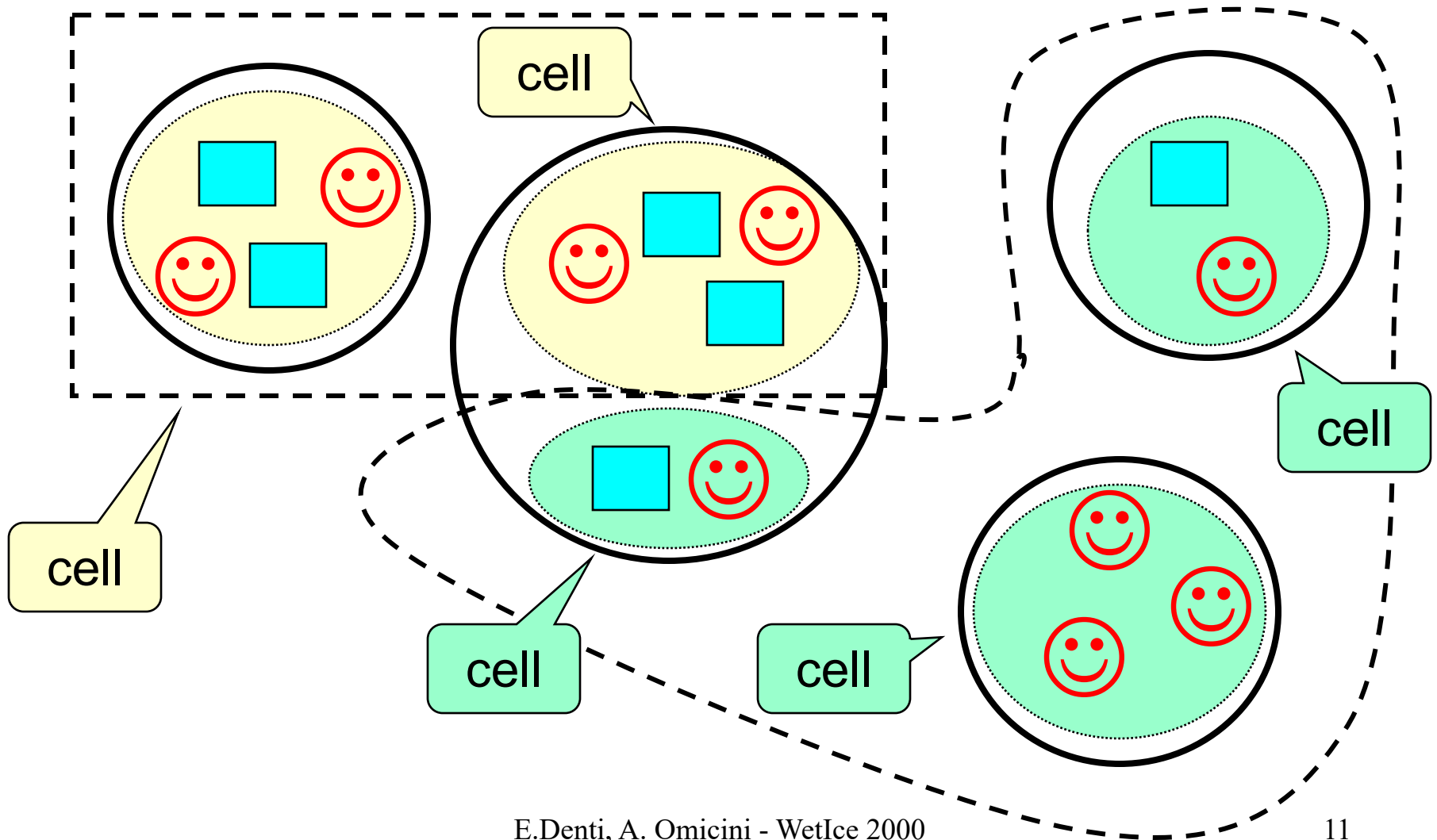
LuCe Architectural Model

- System = a collection of *Cells*
- *Cell* = a collection of *interacting entities* (agents, tuple centres) hosted in a given *physical node*
 - Each cell belongs to a single node
 - But a node can host more than one cell
- *LuCe Space* = a collection of *cells*, possibly spread over the Internet
 - Each LuCe Space is denoted by a global, unique identifier
 - Multiple LuCe Spaces may coexist

LuCe Architectural Model



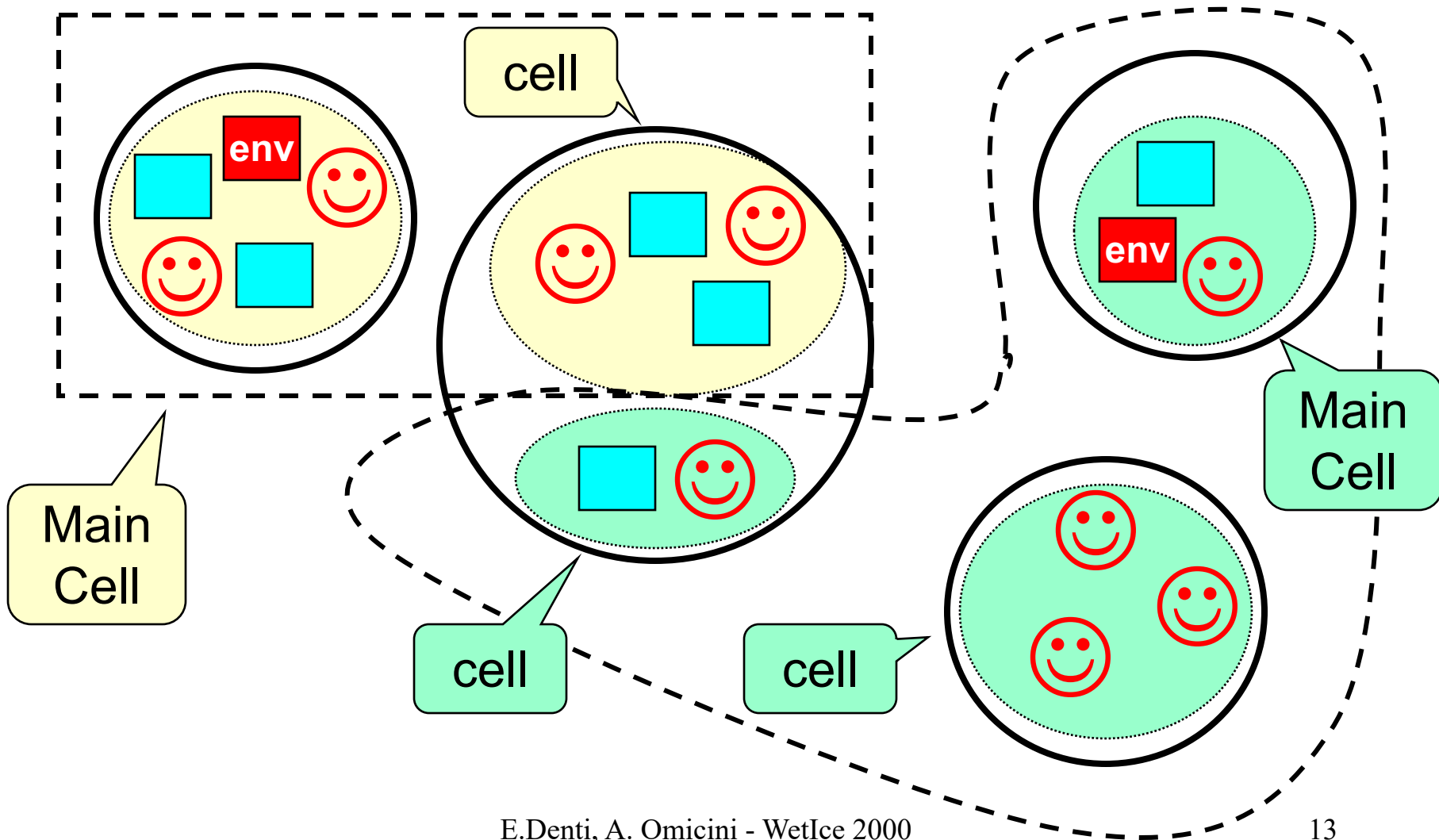
LuCe Architectural Model



LuCe Architectural Model

- **LuCe is Coordination-transparent**
 - Agents do not have to worry about the *physical location* of a tuple centre (they use it *by name*)...
 - *..nor of its existence*: tuple centres are conceptually *ever-existing entities* (practically created when & where needed)
- Such coordination-related information defines the *Environment* of a LuCe space
 - Implemented as a *system tuple centre* (the *environment tuple centre*) handled by an ad-hoc *environment process*
 - Hosted by the **first cell of each space** → *Main Cell*

LuCe Architectural Model



LuCe Technology Overview

- A coordination model *alone* is not enough
- **The LuCe Infrastructure is founded on an effective *Support Technology***
 - Java-based → portability
 - *specifically tailored* to agent-based systems
→ easy (Prolog & Java) agent creation
 - *rooted in the model's abstractions*
 - *exploiting the model's abstractions* to effectively impact the *system development cycle*

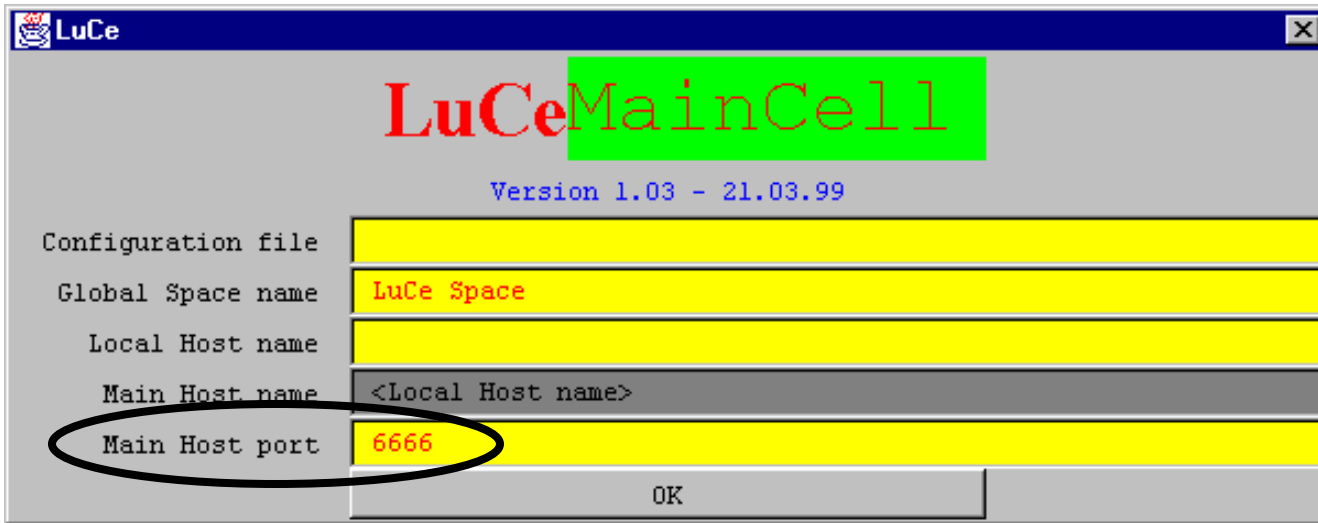
Tuple Centres's Technology

- Behavior Specification → *ReSpecT logic tuples*
- Base Technology → **Java**
 - *Prolog engine (MicroProlog) made in Java*
 - Mapping *Prolog terms* onto *Java objects*
 - chance to work at any time at the most proper level
 - Support for *Agent Heterogeneity*
 - **Java agents, Prolog agents**

LuCe Technology (1/3)

LuCe's **portable IDE** (Java-based) enables users to:

- *Create a new **LuCe Space** along with its **Main Cell** ...*
(the host port identifies the Environment process to be created)



The screenshot shows a window titled "LuCe" with a close button. The main title is "LuCeMainCell" in red text, with "LuCe" highlighted in a green box. Below the title is the version "Version 1.03 - 21.03.99". The window contains a configuration form with the following fields:

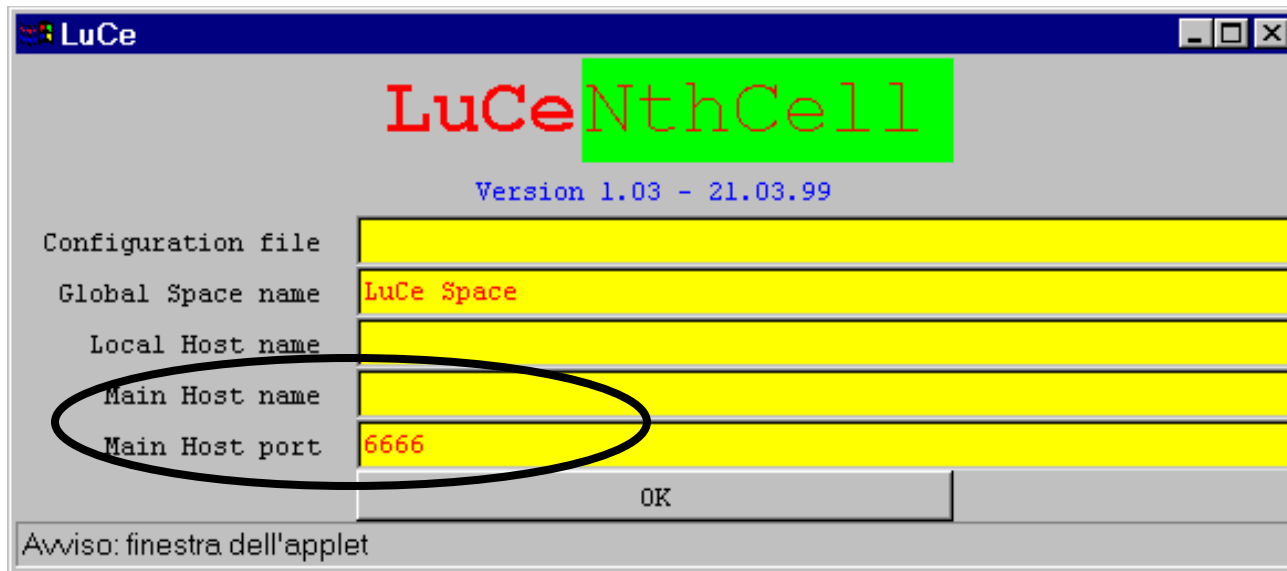
Configuration file	
Global Space name	LuCe Space
Local Host name	
Main Host name	<Local Host name>
Main Host port	6666

The "Main Host port" field is circled in black. At the bottom right is an "OK" button.

LuCe Technology (2/3)

LuCe's **portable IDE** (Java-based) enables users to:

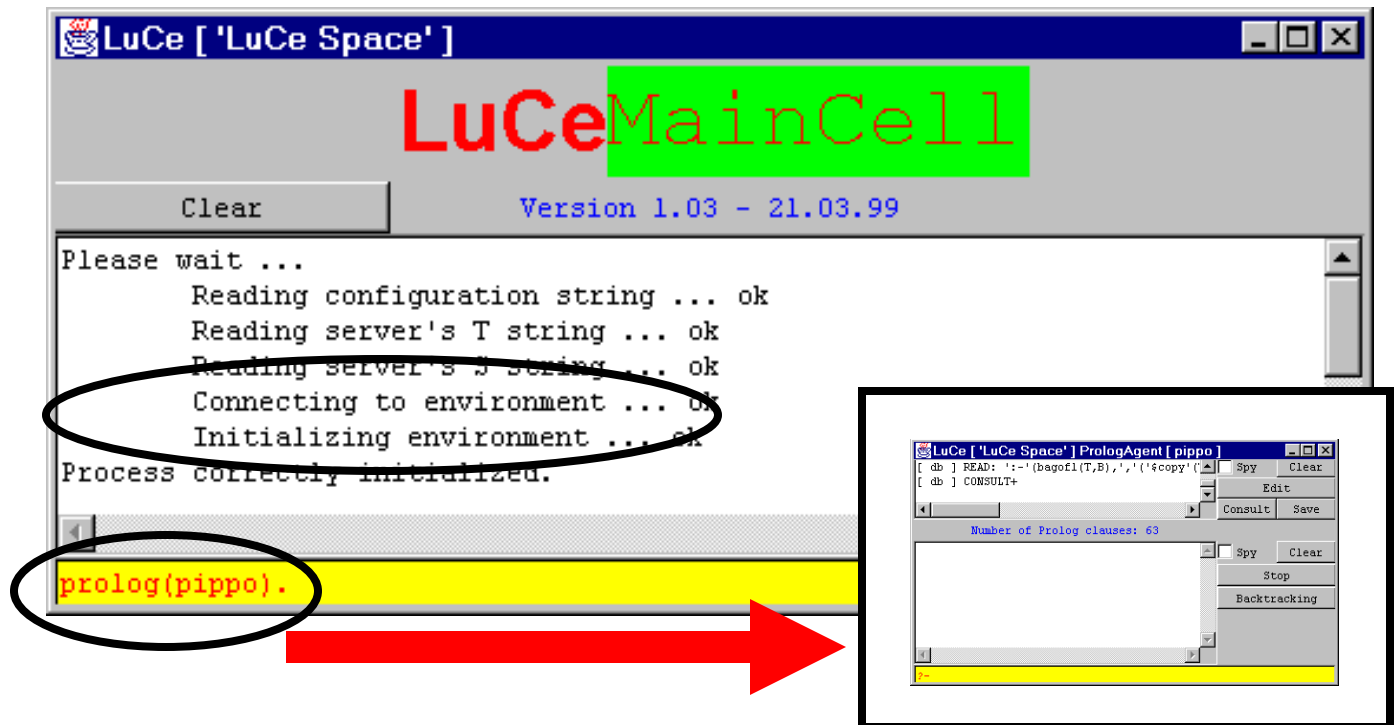
- *...or join an existing **LuCe Space** by creating a new **Nth Cell** (the Main Cell Host & port must be specified).*



LuCe Technology (3/3)

Then, *commands can be issued at the cell console in order to activate new (Prolog & Java) agents & tools*
(new tuple centres are created dynamically when first used)

For example,
this activates
the Prolog
agent `pippo`



LuCe Agents (1/3)

Prolog Agents

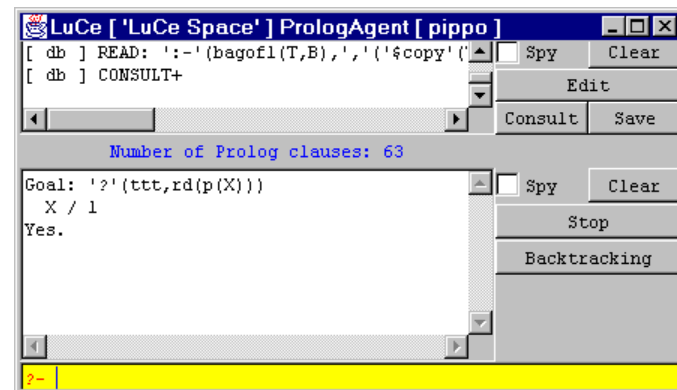
- Built-in MicroProlog interpreter (written in Java)
- Prolog Agents = **Prolog programs** *enabled to perform LuCe coordination operations*

tuplecentre ? in(...)

...

- GUI to control and debug the agent

Prolog agent `pippo` showing its program (top window), step-by-step evolution (center), its control buttons (right), and command line (bottom)



LuCe Agents (2/3)

Java Agents

- **Java Agent** = **Java class** using the **LuCe .Agent** package
- It is a subclass of the **LuCeAgent** class → *enabled to perform LuCe coordination operations*

```
public void Out(mpStruct tuple, mpStruct tcName)  
...
```
- **No predefined GUI** (it's Java!)
- **Simple (applet-like) design pattern:**
 - one **Init** method, one **Begin** method, one **End** method
 - one **mainLoop** method containing the agent's body (*repeatedly executed until mainLoop returns false or Stop is called*)

LuCe Agents (3/3)

A sample Java Agent

```
package Examples.Luce;
import Luce.Agent.*;
import microProlog.Term.*;
public class MyAgent extends LuceAgent {
    private mpStruct tc, tuple;
    public MyAgent() {super(); }
    public void Init() { tc = new mpStruct("tc0");
        tuple = new mpStruct("p", new mpInt(0)); }
    public boolean Begin() { return Connect(); }
    public void End() { Disconnect(); }
    public boolean MainLoop() {
        try { Out(tuple,tc); } catch (Exception e) {}
        return false; }
}
```

Java objects mapping
Prolog structures,
integers, etc

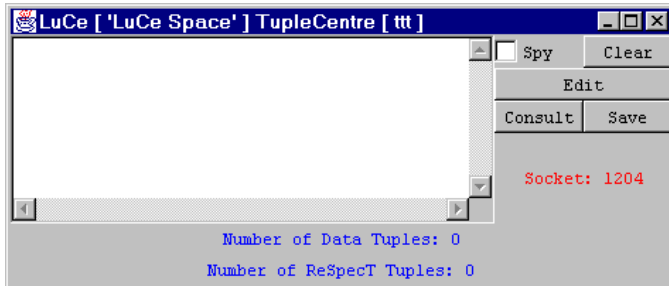
standard agent
pattern

coordination operation

LuCe Tools

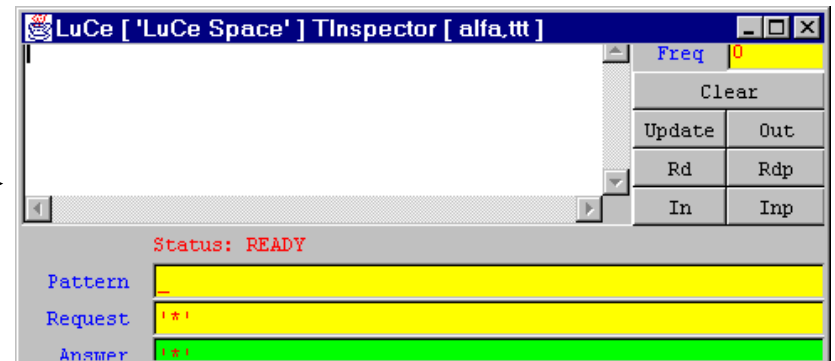
- **LuCe Infrastructure Technology** provides users with *significant views* of Tuple Centres
 - the set *T* of *Tuples* in the tuple space
 - the set *S* of *Specification tuples* in the tuple centre
 - the set *W* of *Waiting Queries* still to be served
- by means of *specialized agents*, the *Inspectors*
 - the *T Inspector* shows (&edits) the (data) tuples
 - the *S Inspector* shows (&edits) the specification tuples
 - the *W Inspector* shows the pending queries

LuCe Tools

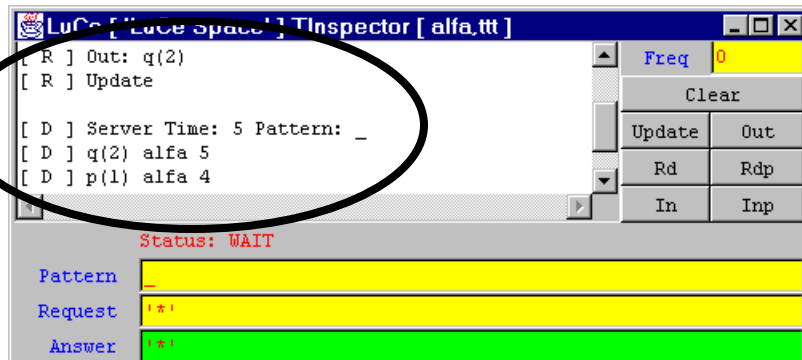


Tuple Centre **ttt** (initially empty)

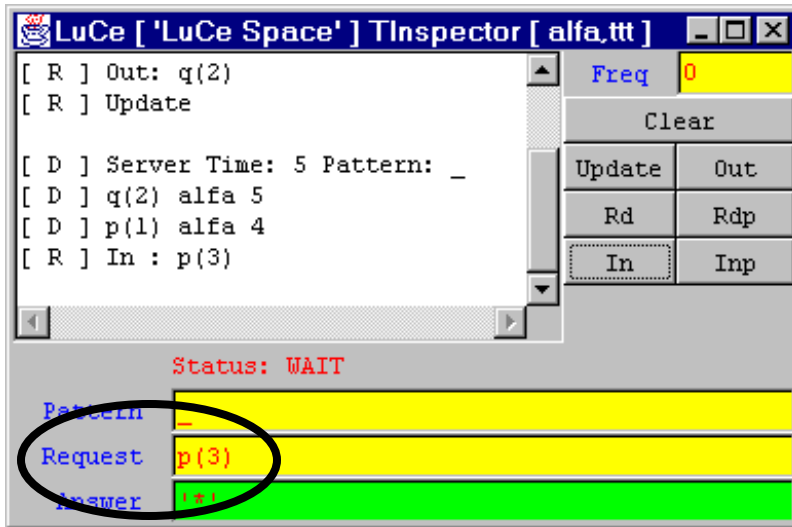
T Inspector **alfa** showing the content of Tuple Centre **ttt**



T Inspector **alfa** showing the content of **ttt** after adding tuples **p(1)** and **q(2)**

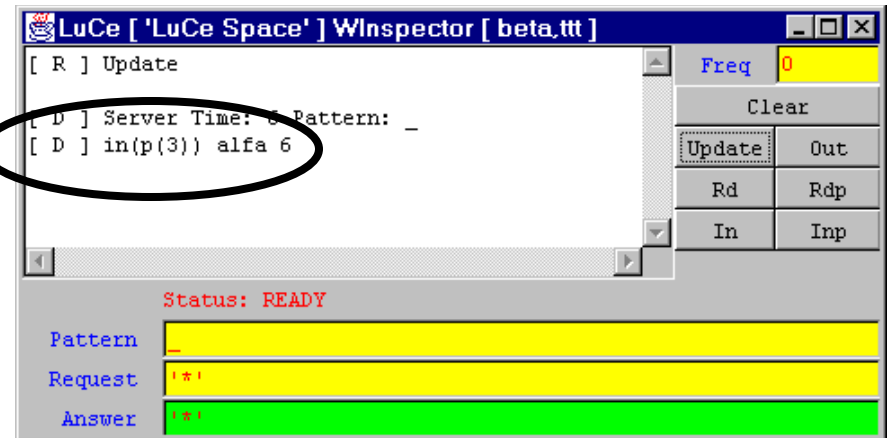


LuCe Tools



T Inspector **alfa** waiting for a tuple matching **p (3)**

W Inspector **beta** showing the requests pending on Tuple Centre **ttt**



LuCe Console Commands

- The console provides users with the ability to:
 - activate new *Prolog agents* → `prolog(name, ...)`
(possibly with a file name containing a Prolog program)
 - activate new *Java agents* → `agent(name, ...)`
(possibly with a file name specifying a Java class, possibly getting command line arguments)
 - activate *LuCe inspecting tools*
→ `tinspect(name, tc)`, `winspect(name, tc)`,
`sinspect(name, tc)`

LuCe: support to Design

- The **LuCe infrastructure** introduces some *uncoupling dimensions* in the system **design & development process**.
 - *agents* designed independently from each other
 - *agent interaction protocols* designed separately from agents and independently from each other
 - *coordination media* "gluing things together"
- ... and ***supports them by means of its tools***
 - **inspectors** can simulate other agents' behaviour
 - **tuple centres can be monitored** (dynamic views)

Conclusions...

- *Infrastructures* to *factorize properties* of classes of agent-based systems need **coherent abstractions, technologies** and **methodologies**
- The **LuCe infrastructure** considers *coordination* as *a basic service* for agent societies
- **LuCe tuple centres** integrate *coordination abstractions* with *technologies*:
 - **Java, Logic** and **Tuples** *in a coherent framework*

...and Open Issues

- Support for *mobility* (join work on TuCSoN)
- *Security* model
- *Methodologies*

Related Works

- **IBM T Spaces** (*programmable Java object space*)
- **Sun Microsystem's JavaSpaces**
integrating coordination into a general-purpose infrastructure (Jini) - but no support tools
- **MARS, TuCSoN**
(TuCSoN = coordination as unifying paradigm accounting for authentication, intelligent agent exploration, etc)
- **PageSpace** (*reference architecture - kernel agents instead of progr. coord. media accounting for global system props*)
- **LogiMOO, Linda Interactor, Jinni**
(Jinni = logic-based scripting language to glue Java objects and kn. processing components together + blackboard)