Programming Distributed Multi-Agent Systems in simpAL

Andrea Santi[1], Alessandro Ricci[1]
a.santi@unibo.it, a.ricci@unibo.it

Dipartimento di Informatica Scienza e Ingegneria (DISI),
Alma Mater Studiorum—Università di Bologna[1]

Tredicesimo Workshop Nazionale Dagli Oggetti agli Agenti

18/09/2012

Outline

Context of the Contribution 1/2

- MASs are a main paradigm for developing complex software systems [Jennings, 2001], featuring
  - Concurrency
  - *Physical distribution*
  - Decentralization of control
  - ...
- Accordingly, Agent-Oriented Programming Languages (APLs) must provide adequate
  - First-class programming abstractions
  - Runtime infrastructures for MASs execution
- Here we focus in particular on the programming of *distributed* MASs

Context of the Contribution 2/2

- Distribution is one of the essential features of MASs...
  - .. giving developers the opportunity to engineer a physically distributed application as a MAS spread among different network nodes

Support of state-of-the-art APLs for developing distributed MASs?

- A distributed MAS is (generally) given by a set of *separate* MASs
- Interactions among the different parts is enabled by:
  - Agent communication via FIPA-ACL [fip, 2011] or KQML [Finin et al., 1994]
  - FIPA-compliant platforms – e.g. JADE [Bellifemine et al., 2007] – for agent discovery, message routing, etc. at runtime

Current Support for Distribution in APLs

- Quite effective to handle openness and dynamism
  - E.g., dynamic addition and lookup of new agents
- However, the programming/deployment/execution of physically distributed MASs is quite a troublesome job [Braubach et al., 2005]
  - *Significantly harder* than the centralized case

Main Issues

- Programming errors related to message-based interactions can be detected only at runtime
- Programs need to be deployed, run and managed manually for each node in which the MAS is meant to run
  - Both tiresome and error-prone

Our Perspective

The Rationale

The programming, deployment and execution management of distributed MASs can be as simple as the case of centralized, not distributed ones

- Specific abstractions and mechanisms must be introduced
  - Directly at the language level
  - And then supported by the runtime
- Easing the programming of physically distributed MASs
  - Detecting programming errors statically
  - Automatizing MASs deployment, run and termination
- Accordingly, here we discuss how to program physically distributed MASs in simpAL [Ricci and Santi, 2011b]

Outline

Outline

simpAL Project Overview 1/2

Background/motivations: the free lunch is over [Sutter and Larus, 2005]

- How to model & organize concurrent, reactive programs scaling with their complexity?
- Mainstream paradigms (e.g., OOP) are not enough
  - Looking for a proper level of abstractions

General Research Objective

- Exploring paradigms based on agent-oriented abstractions
  - From agent programming from the perspective of AI
  - To general-purpose software development viewpoint

Method

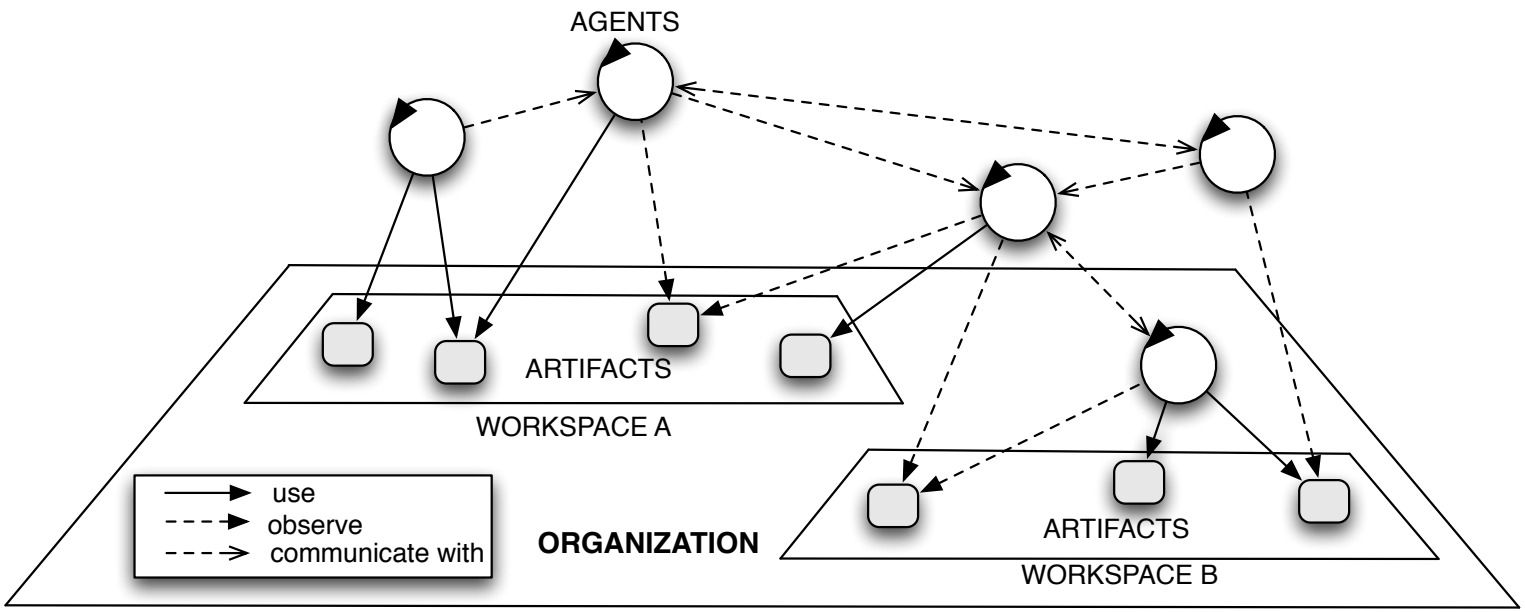
- simpAL programming language & related tools/ecosystem
- Strongly inspired from existing work in MAS
  - JaCa [Ricci and Santi, 2011a] & JaCaMo [Boissier et al., 2011]
  - simpA [Ricci et al., 2011]

simpAL Project Overview 2/2

- simpAL programming language
  - First-class agent-oriented programming abstractions for defining the organization and control of MASs
  - General-purpose for concurrent & distributed programming
  - Compiled & statically typed language
- simpAL platform
  - Distributed runtime
  - IDE &tools
- simpAL core language & formalization
  - Agent-oriented core language & calculus for studying properties

Outline

simpAL Concepts Overview



simpAL program: *organization* of (dynamic) set of *agents*

- Working in a environment structured into *workspaces*
- Sharing resources/tools modeled as *artifacts*

Background Metaphor: an Abstract Representation

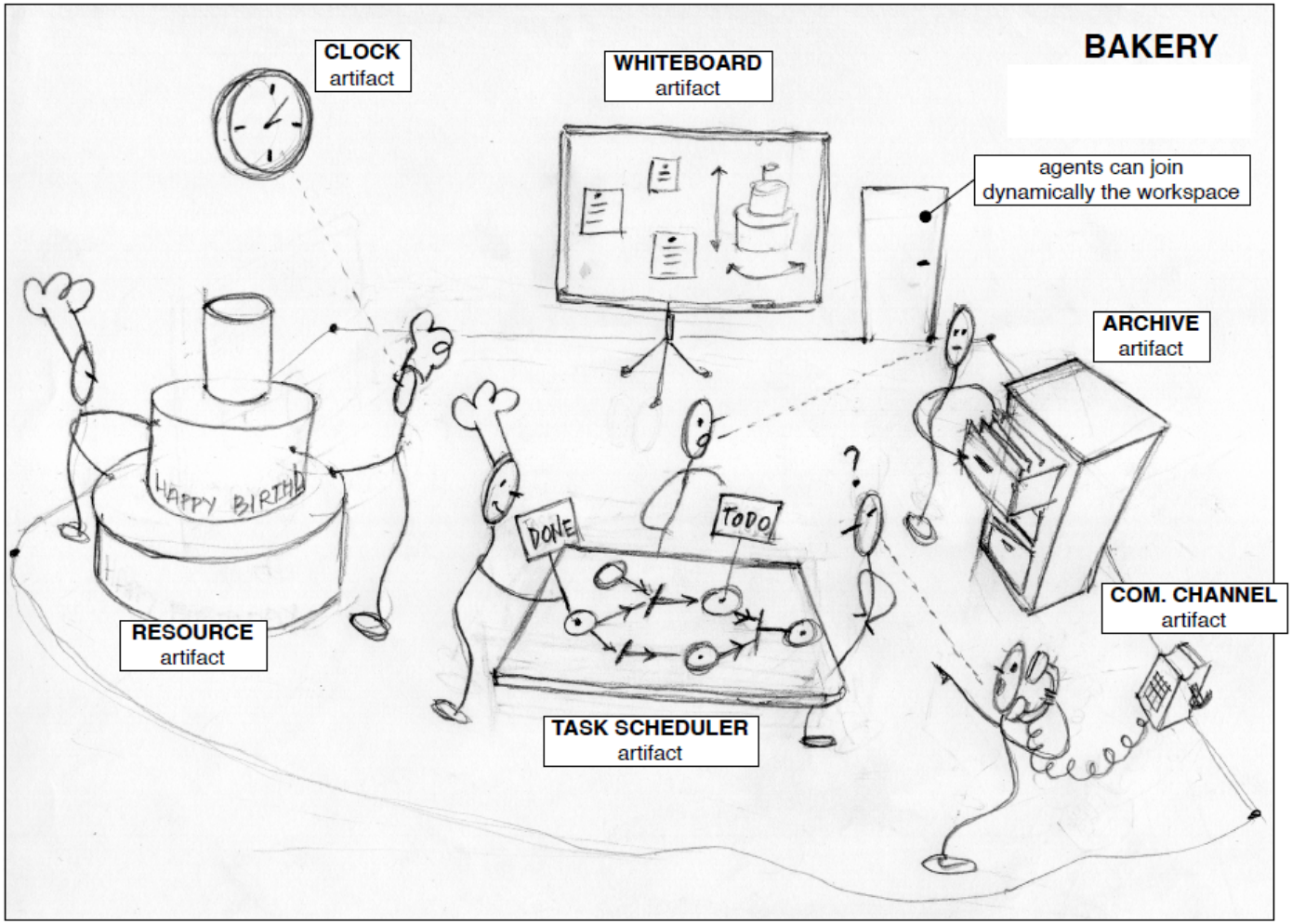
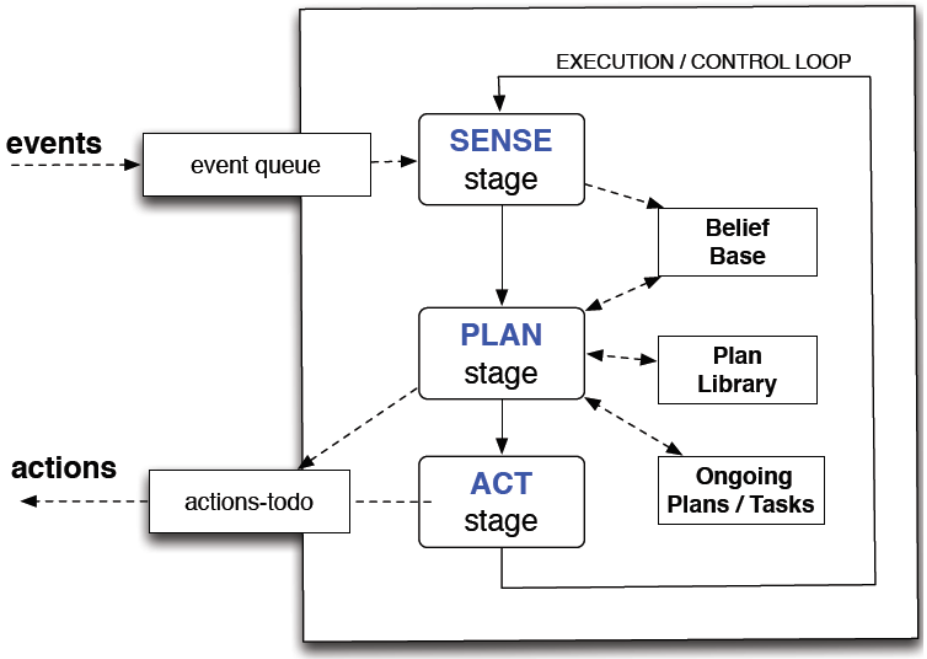


Figure : [Omicini et al., 2008]

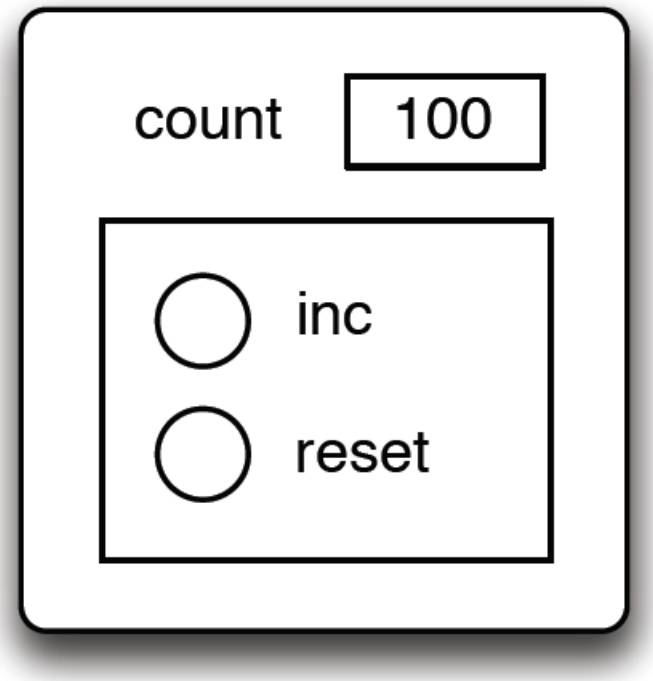
Agent Model At a Glance

- BDI-Inspired
  - Tasks
  - Beliefs
  - Plans to do tasks
  - Intention as plans in execution
  - Actions/percepts
- Agent exec cycle
  - simple SPA-like
  - Extension of actor event-loop

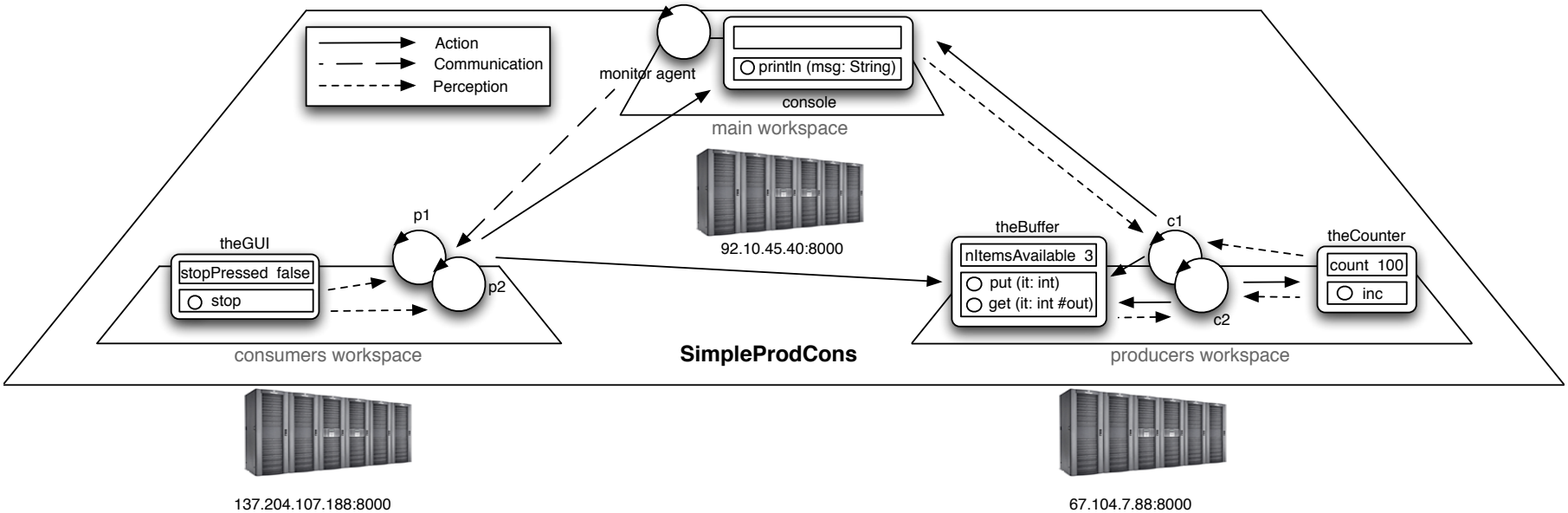


Artifact Model At a Glance

- Based on A&A
  - Observable properties
  - Operations (actions)



Guiding Example: Distributed Producer-Consumer



Programming the Organization

- Specify & type-check the logical structure of the distributed MAS
- Deployment managed through a dedicated deployment config file

```
/* ProdConsOrg organization model */
org-model ProdConsOrg {

  roles: Producer,Consume,ProdMon;
  interfaces: Console,GUI,Buffer,Counter;

  workspace producers {
    p1, p2: Producer theGUI: GUI
  }

  workspace consumers {
    c1, c2: Consumer
    theCounter: Counter theBuffer: Buffer
  }

  workspace main {
    console: Console monitor: ProdMon
  }}
```

```
/* Deployment file */
workspace-addresses {
 main = 92.10.45.40:8000
 producers = 67.104.7.88:8000
 consumers = 137.204.107.188:8000
}
```

```
/* SimpleProdCons organization */
org SimpleProdCons implements ProdConsOrg {

  workspace producers {
    theGui = SimpleGUI ()
    p1 = SimpleProducer(bufferToUse:
      theBuffer@consumers)
      init-task: Producing(numItems: 20)
    p2 = ...
  }

  workspace consumers {
    theCounter = Counter(startValue: 0)
    theBuffer = SimpleBuffer(maxElems: 10)
    c1 = SimpleConsumer()
      init-task:
       Consuming(maxItemsToProcess: 40)
    c2 = ...
  }

  workspace main {
    console = Console()
    monitor = ...
  }
}
```

Programming the Agents: Tasks & Roles

- Tasks
  - Abstract description of a job to be done
  - Fist-class abstraction and data-type
- Roles
  - Defining the type of an agent
  - What kind of tasks can be assigned to agents of that type

```
1   role Producer {
2
3     task Producing {
4
5       input-params {
6         numItems: int;
7       }
8
9       understands {
10        newItemsToProduce: int;
11      }
12    }
13  }
```

Typing Agents: Outcome

- Error checking
- Principle of substitutability

Programming Agents Behavior: Scripts and Plans

- Script = how to fulfill some role(s)
  - Set of plans + beliefs as typed (agent) state variable
- Plan = set of action rules + local beliefs declaration
  - Action rules = ECA-style rules defining *when* executing *which* action
    - `(every-time|when) Ev: Cond => Act`

```
1  agent-script SimpleProducer implements Producer  in ProdConsOrg {
2    plan-for Producing {
3      noMoreItemsToProduce: boolean = false;
4      item: int = 0;
5      nItemsTodo: int = maxItems;
6      completed-when: noMoreItemsToProduce using:  theBuffer, theGui {
7
8        /* purely active part */
9        repeat-until items > nItemsTodo {
10         item = item + 1
11         put(item: item) on buffer
12       }
13       noMoreItemsToProduce = true
14
15       /* reactive part */
16       when changed stopPressed in theGui@producers =>
17                                using: console@main {
18         println(msg: "stopped!")
19         noMoreItemsToProduce = true
20 }}}}
```

Programming the Environment

Separating specification & implementation

- Usage interface (spec)
  - Obs properties & operation (action) signatures
- Artifact template (concrete behavior)

```
1   /* Buffer usage interface */
2   interface Buffer {
3     obs-prop nAvailItems: int;
4     operation put (item: int);
5     operation get (?item: int);
6   }
```

```
1   artifact SimpleBuffer implements Buffer {
2
3     int maxNumElems;
4     java.util.LinkedList<Integer> elems;
5
6     init (maxElems: int) {
7       count = startValue;
8       nAvailItems = 0;
9       maxNumElems = maxElems;
10      elems = new java.util.LinkedList<Integer>();
11    }
12
13    operation put (item: int) {
14      await nAvailItems < maxNumElems;
15      elems.add(item);
16      nAvailItems = nAvailableItems + 1;
17    }
18
19    operation get (?item: int) {
20      await nAvailItems > 0;
21      ...
22  }}
```

Outline

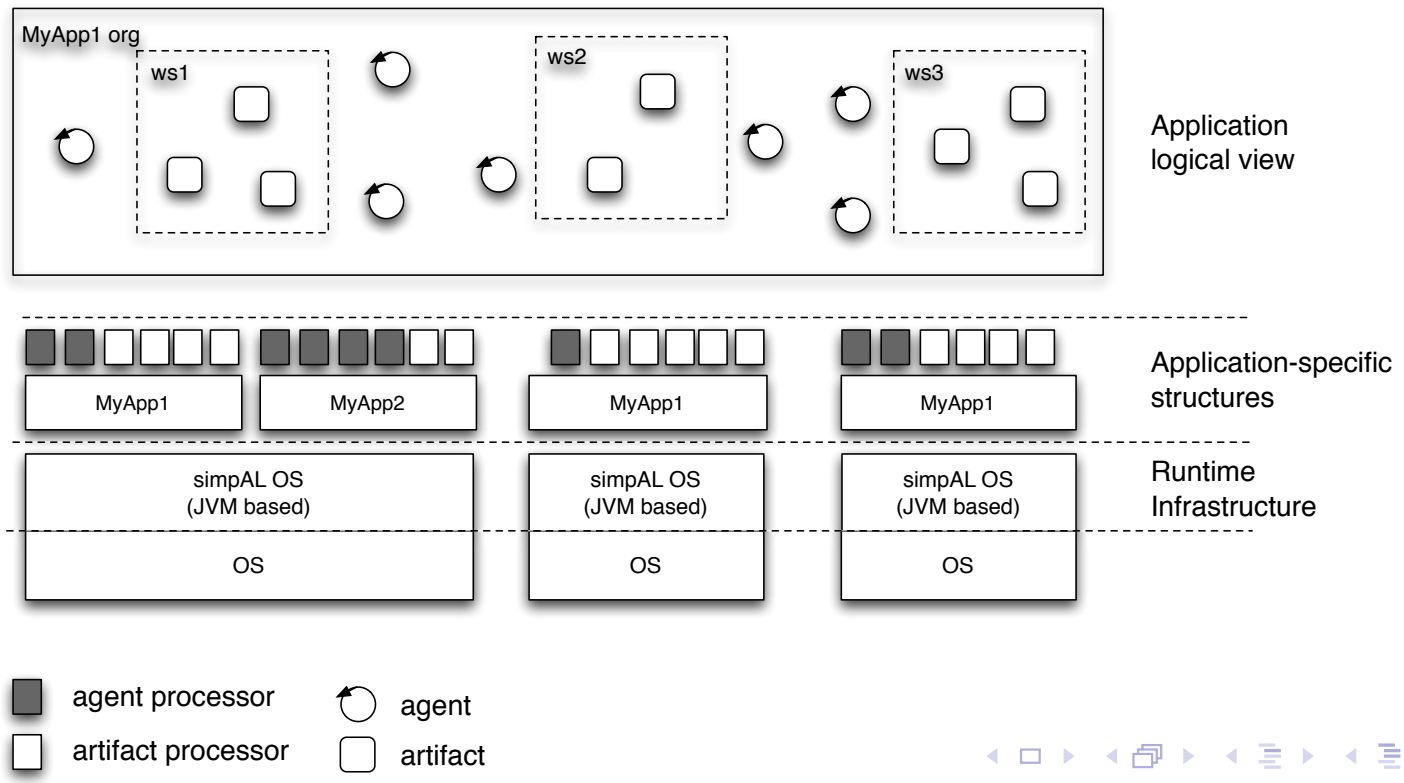
simpAL Distributed Runtime Infrastructure

Building Blocks

Processors for supporting concurrency at the logic level

simpAL kernel manage execution of simpAL programs

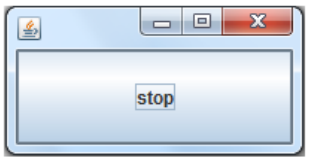
simpAL node network node in which the kernel has been installed

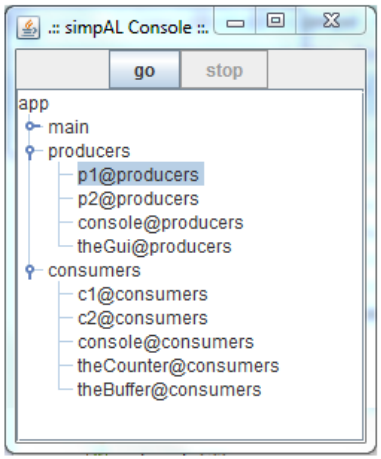


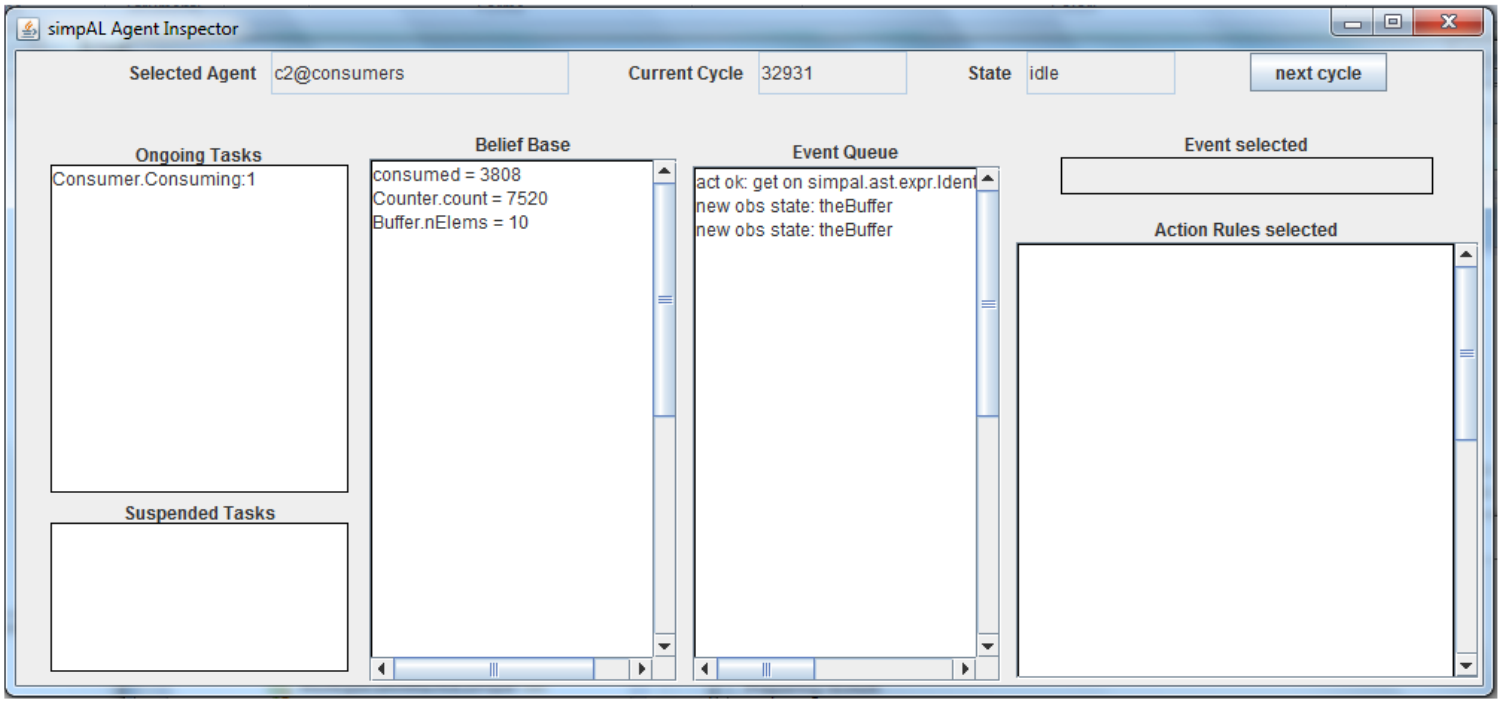
Execution Management of Distributed MAS in simpAL

1. MAS execution can start from any *active* simpAL node
   - The selected node assumes the role of *launch manager*
2. Inspection of the deployment configuration file (if any)
3. (Concurrent) distribution of the zipped compiled sources among the interested simpAL nodes
4. Booting process starts (*asynchronously*) in each node
5. *Coordinated* initialization of the different program parts
   - Through a simple, ad-hoc handshake protocol
   - Execution is suspended until the completion of the *whole* boot process
6. Booting complete on each node, MAS can start executing
7. Shutdown request (from a generic node)
   - Coordinated shutdown of the application
   - Again, through a simple ad-hoc handshake protocol

The simpAL Debugger







Outline

Summing Up: Focus on Main Aspects

- Native support for distribution, part of the programming model
  - First-class abstractions to define the logical structure of the MAS
  - Uncoupled deployment model
- Compile-time error detection concerning both
  - (Remote) agents interactions
  - (Remote) artifacts operation execution
- Easy management of deployment & execution of distributed MAS
  - Entirely automatized
    - Once the distributed runtime infrastructure is up & running
  - No specific code required for MAS initialization

Future Work

- Improve the management of runtime faults/errors
  - E.g. node crashes, network problems
  - For now simply modeled as action failures
  - Final objective: a fault-tolerant runtime infrastructure able to manage
    - Temporary node unreachability
    - Dynamic addition/shutdown of nodes
    - Workspace(s) migration
- Better support for openness and dynamism in programs
  - Enabling interactions between different MAS
  - Maintaining the simpAL application model
- Debugger improvements

Bibliography I


(2011).
The Foundation of Intelligent Physical Agents organization (FIPA) – `http://www.fipa.org`, last retrieved: July 5th 2011.


Bellifemine, F. L., Caire, G., and Greenwood, D. (2007).
Developing Multi-Agent Systems with JADE.
Wiley.


Boissier, O., Bordini, R. H., HŸbner, J. F., Ricci, A., and Santi, A. (2011).
Multi-agent oriented programming with jacamo.
Science of Computer Programming, (0):–.

Bibliography II

Braubach, L., Pokahr, A., Bade, D., Krempels, K., and Lamersdorf, W. (2005).
Deployment of distributed multi-agent systems.
Engineering Societies in the Agents World V, pages 898–898.

Finin, T., Fritzson, R., McKay, D., and McEntire, R. (1994).
Kqml as an agent communication language.
In *Proceedings of the third international conference on Information and knowledge management*, pages 456–463. ACM.

Jennings, N. R. (2001).
An agent-based approach for building complex software systems.
Communication of ACM, 44(4):35–41.

Bibliography III


Omicini, A., Ricci, A., and Viroli, M. (2008).
Artifacts in the A&A meta-model for multi-agent systems.
Autonomous Agents and Multi-Agent Systems, 17 (3):432–456.


Rao, A. S. and Georgeff, M. P. (1995).
BDI Agents: From Theory to Practice.
In *First International Conference on Multi Agent Systems (ICMAS95)*.


Ricci, A. and Santi, A. (2011a).
Agent-oriented computing: Agents as a paradigm for computer programming and software development.
In *Proc. of the 3rd Int. Conf. on Future Computational Technologies and Applications (Future Computing '11)*, Rome, Italy. IARIA.

Bibliography IV

Ricci, A. and Santi, A. (2011b).
Designing a general-purpose programming language based on agent-oriented abstractions: the simpAL project.
In *Proc. of the compilation of the co-located workshops on DSM'11, TMC'11, AGERE!'11, AOOPES'11, NEAT'11, VMIL'11*, SPLASH '11 Workshops, pages 159–170, New York, NY, USA. ACM.

Ricci, A., Viroli, M., and Piancastelli, G. (2011).
simpA: An agent-oriented approach for programming concurrent applications on top of java.
Sci. Comput. Program., 76:37–62.

Sutter, H. and Larus, J. (2005).
Software and the concurrency revolution.
ACM Queue: Tomorrow's Computing Today, 3(7):54–62.