Models, methodologies and infrastructures for agent and component-based systems: interoperability, adaptability and coordination

Enrico Oliva

University of Bologna

Seminar - 2° year Phd, 2006

Outline

1. Motivation
   - Coordination in Complex Software Systems
2. Different Application Domain / Results
   - Component-based System
   - Agent-based System
3. Future Works

Outline

1. Motivation
   - Coordination in Complex Software Systems
2. Different Application Domain / Results
   - Component-based System
   - Agent-based System
3. Future Works

Software Systems I

There is an increasing complexity in new software systems, it is necessary novel models and architectures. Interoperability and adaptability are two new software requirements

- interoperability -> components could be different but they should work together
- adaptability -> components should adapt to the new conditions of the environment

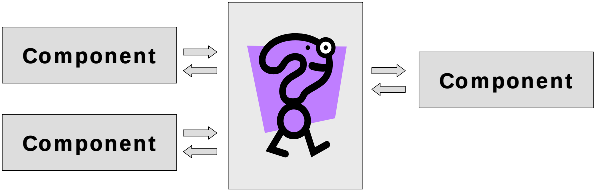
Coordination

Coordination techniques are a strategic point, little considered for building software adaptable and inter-operable

Software Systems II

Software Systems are generally composed of multi-entities,that exchange information in order to work together

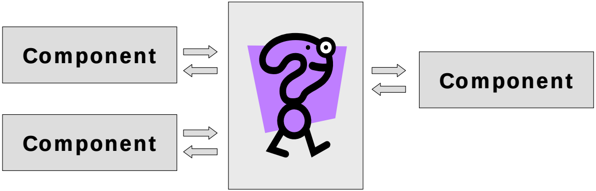
- The main problems in a multi-entities system are: heterogeneity, communication and trust
- The common approaches to reduce the complexity of the system is to build a infrastructure among the parts



Software Systems III

The main properties required from the infrastructure are

- controllability → to control and guide easily the interaction among the parts
- malleability → to change run-time the coordination law
- declarative representation of the coordination rules



Outline

1. Motivation
   - Coordination in Complex Software Systems
2. Different Application Domain / Results
   - Component-based System
   - Agent-based System
3. Future Works

Component-based System

- Component-based approaches are the mainstream for designing complex software architectures
  - The component as a basic brick to structure a system
- The composition is made possible by explicitly declaring the interfaces that a component provides and requires
  - The software engineers reason in terms of structural composition of entities

Coordination and Components

Coordination techniques are not really applied in mainstream approaches

- the composition simply amounts to the adaptation of interfaces
- the interaction laws or logic hidden inside components

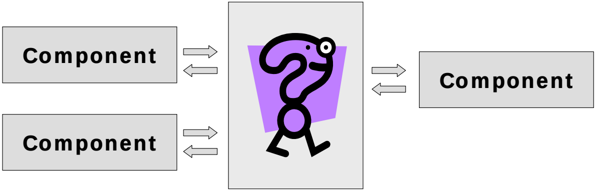
Key Point

specifying management of component-interactions independently of the components

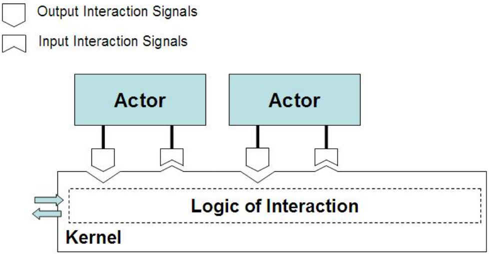
The Framework I

We provide a framework for glueing Java software components; currently it supports

- some basic interaction primitives (inform, notify, ..)
- a logic-based approach to specify other interaction laws



The Framework II



- The Actor represents the component
  - Logic entity capable to do a service
  - It is immersed in an environment

- The Kernel represents the environment
  - It provides the interaction support
  - It encapsulates the logic of interaction

Interaction Primitives

- The current implementation of the Kernel supports some basic *interaction primitivies*
  - modalities for connecting signal emission and reception of different components
- The primitives generate a signal without specifying the target actor
  - The components will receive the signal depending on the specific logic of interaction

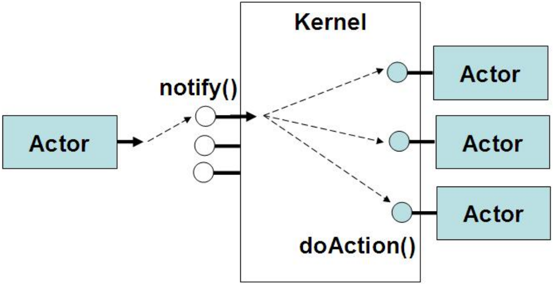
Three primitive

- notify, inform and invoke
- the kernel has the burden of supporting their different semantics

Interaction Primitive: *notify*

Notify used to let interested

- *Actors* observe some change in the state or behavior of the emitter
- *Emitter* does not receive any reply, the signal might be even lost



Results

1. The work has been published in an international workshop

   [A Framework for Engineering Interactions in Java-based Component Systems, FOCLASA, 2005]

2. The framework has been used in the STIL project

   [Strumenti Telematici per l'Interoperabilità nelle reti di imprese: Logistica digitale integrata per l'Emilia Romagna]

Outline

1. Motivation
   - Coordination in Complex Software Systems

2. Different Application Domain / Results
   - Component-based System
   - Agent-based System

3. Future Works

Agent-based System

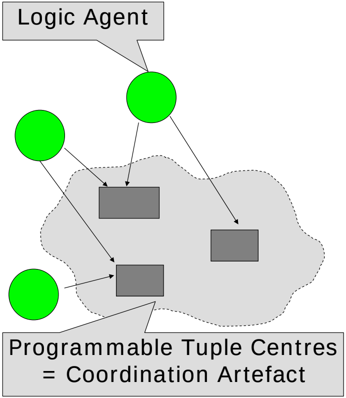
Definition of Agent

An *Agent* is an autonomous and proactive entity situated in a environment with social ability that brings about his goal

Definition of Multi-Agent System (MAS)

A *MAS* is a system composed with a set of agents that cooperate and compete to achieve a common result.

TuCSoN Infrastructure



Definition

TuCSoN is a coordination infrastructure for MAS that provides logic tuple centres where the agents can put and get tuples

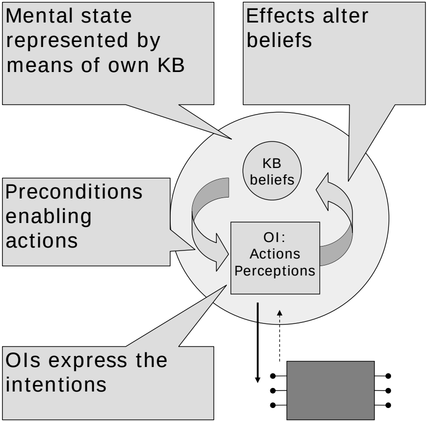
- coordination language is made by the operators: in, rd, inp, rdp
- tuple centres are programmable tuple space → coordination artifacts

The Artifact Abstraction for MAS

- Artifact[1] is a computational device designed to provide some kind of function
  - agents: goal / task oriented, proactive
  - artifacts: function-oriented, passive / reactive
- Basic brick to design / engineer agent computational environments
  - artifacts as targets of agent activities
  - used as mediator for activities (social, individual, resource)
- Reframing agent behaviour
  - computing (deliberating)
  - communicating with other agents
  - working with shared artifacts
  - constructing, manipulating, using/accessing them

[1] *The Artifact Abstraction is a result of the research work team in Cesena*

Logic-based Agent


Mental state represented by means of own KB
Effects alter beliefs
KB beliefs
Preconditions enabling actions
OI: Actions Perceptions
OIs express the intentions

- Operating Instructions (OI) are connected with the agent mental state
  - precondition → action
  - perception → effect
- Knowledge Base compound of facts and rules
  - inference for verifying preconditions and asserting effects
- Agents modifying their behaviours during the time
- Implementation based on tuProlog technology http://tuprolog.alice.unibo.it

TuCSoN Simulation Framework

Coordination artifacts

- all agents share the same coordination artifacts
- coordination as a service to let agents participate to social activities

Different agent roles

- Players → a player agent elaborates and executes a plan
- Observers → a observer agent observes interaction
    - passive: monitor agent observes interaction and visualises results
    - active: tuning agent observes interaction and possibly changes coordination rules
        - tuning coordination parameters

Minority Game (MG) Simulation

MG is a human society abstraction, it is used

- in economics such as a coarse-grained model for financial markets to study their fluctuation phenomena and statistical properties
- in social simulation such as a simplified human scenario to understand mass behaviour

The Game

- minority side wins over an odd population
- at each time-step, each player chooses side A or side B
- player uses past history to take next decision
  - inductive reasoning
  - bounded rationality

Results

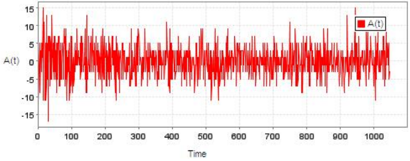


Figure: Typical time evolution of the game

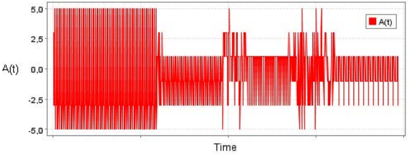


Figure: System evolution after tuning section

The work has been published

"Simulating Minority Game with TuCSoN" Industrial Simulation Conference (ISC), Palermo, 2006.

Future Works

The Argumentation Theory is a logic framework to use inside our coordination infrastructure in order

1. to manage partial and incomplete information
2. to build more realistic agent society
3. to enable the exchanged of arguments among multiple entities

collaboration

This work is started in collaboration with Computer Science Department of Liverpool

Thank you!

Questions?