UNIVERSITÉ DE TECHNOLOGIE DE BELFORT-MONTBÉLIARD

JANUS: Another Yet General-Purpose Multiagent Platform

www.janus-project.org

Stéphane Galland
on behalf of Janus Development Team

ICAP Team,
Laboratoire Systèmes et Transports (SeT),
Université de Technologie de Belfort-Montbéliard (UTBM),
F-90 000 Belfort, France
stephane.galland@utbm.fr

http://www.multiagent.fr

SET Laboratoire Systèmes et Transports

utbm

Outlines

Outlines

Introduction

Context

- Massive Multi-Agent Based Simulation
- Design and Simulation of Complex Systems
- AOSE: From Analysis to Deployment Methodology and Tools
- Main Application Domain: Massive crowd simulation in virtual environment

Janus and ASPECS Consortium

SeT — Belfort, France

Voxelia S.A.S — Belfort, France

CITAT — Tucuman, Argentina

ICAR-CNR — Palermo, Italy

Introduction – Why Another Platform?

To Help Agent-Oriented Software Engineering

1. To support organizational concepts as first-class abstractions: Role and Organization.
2. Direct integration with the ASPECS[1] methodology and CRIO metamodel.
3. Contribute to fill the gap between design and implementation phases in AOSE.
4. To strictly respect agent fundamental characteristics: autonomy.

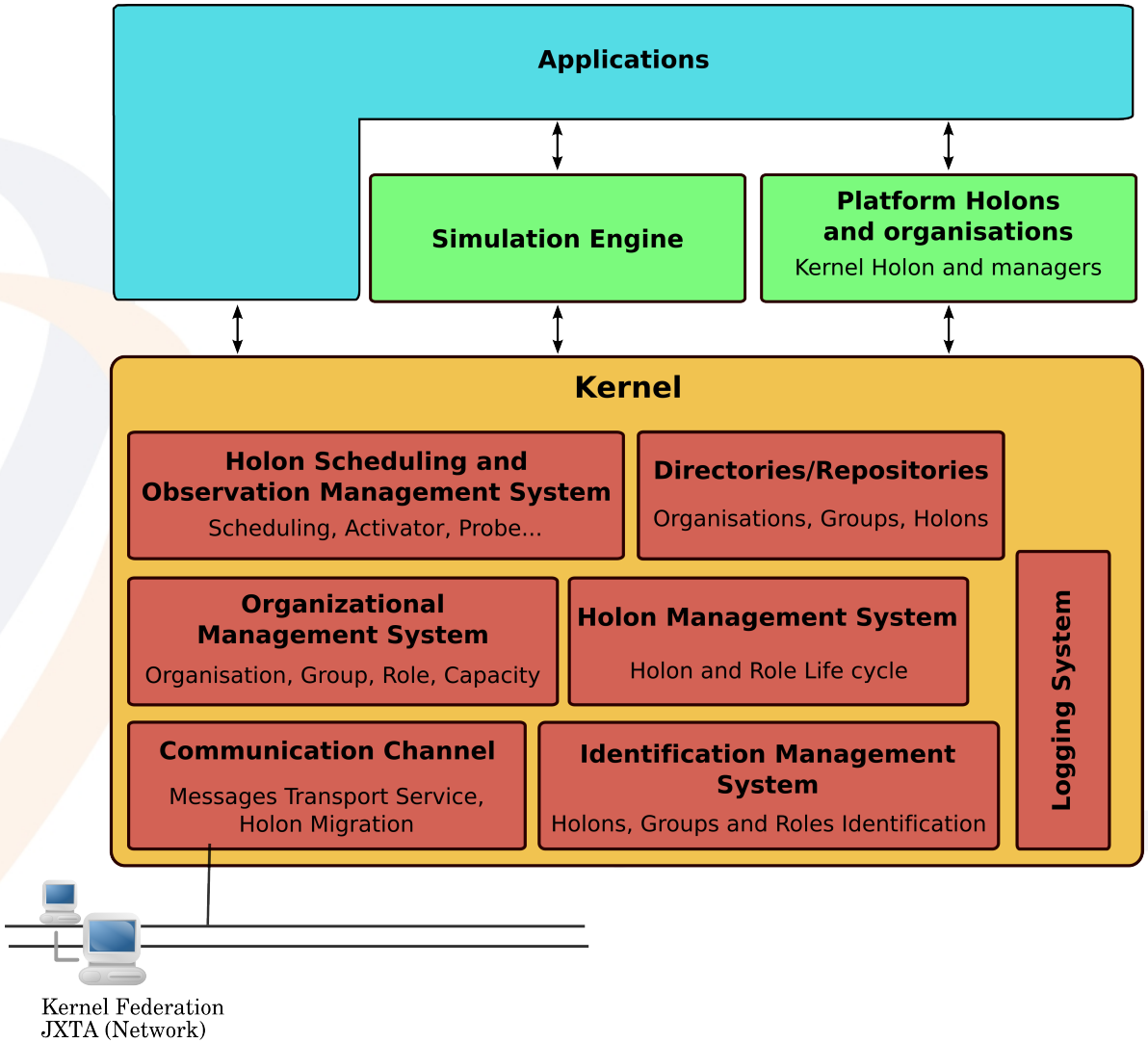
[1] see: www.aspecs.org

Introduction – Why Another Platform?

To Help Developers

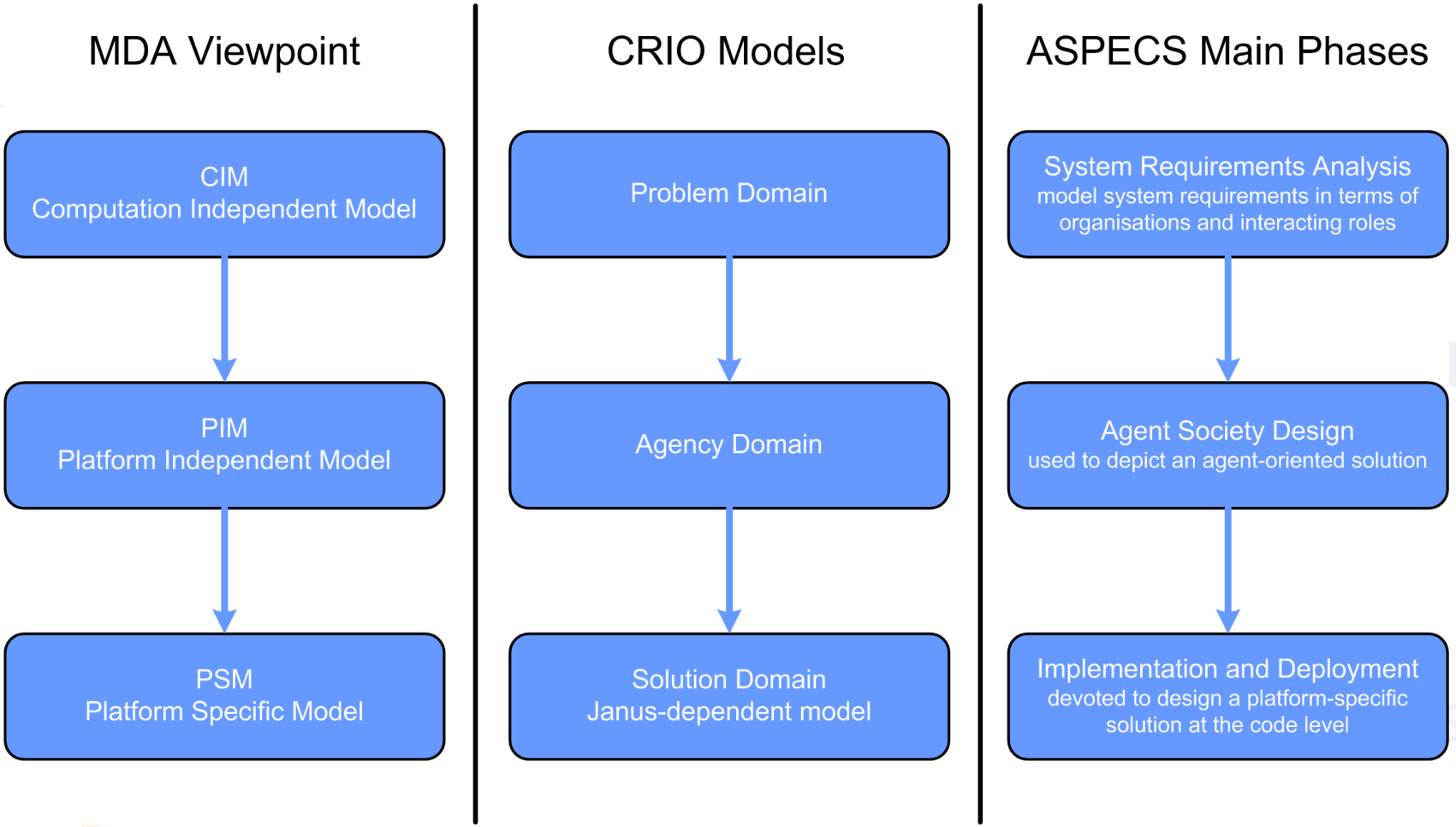
1. To implement with the same general-purpose platform: Agent-oriented, Organisational, Holonic, Eco-Agent, BDI, Simulation.
2. A developer-oriented platform designed to develop applications dealing with real case studies (large, complex, etc.)
3. To preserve computational performance (realtime, mobile devices, embedded systems)

Janus Platform Architecture



Outlines

ASPECS and CRIO Introduction: 3 levels of abstraction



ASPECS Fundementals

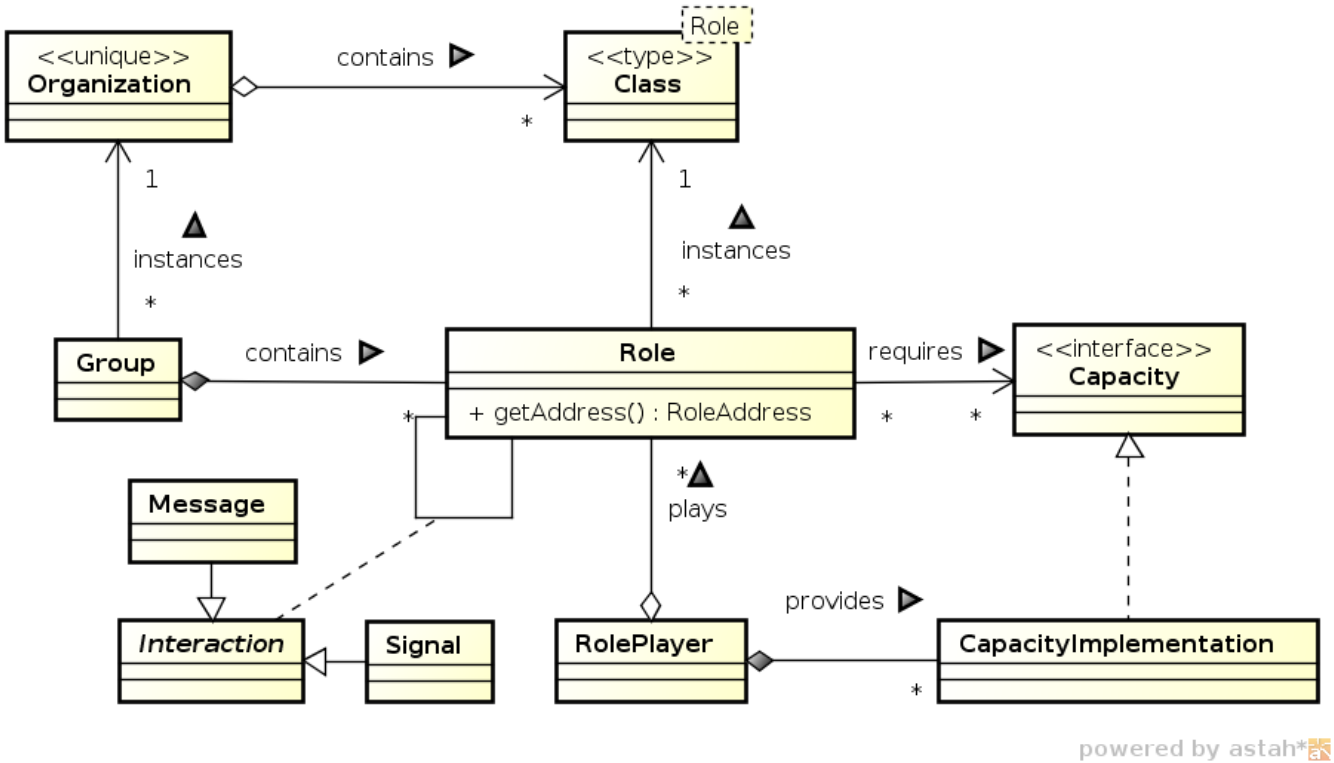
- Explicitly deals with the design of open, dynamic and complex systems.
  Assumption: system is hierarchically decomposable in sub-systems.
- Adoption of organisational approach: intra and inter-level
  - **Vertical** delegate the responsibility of an organisation at level n to sub-organisations at level n-1
  - **Horizontal** collaboration of several entities at the same level to fulfil the required functionalities.
- Domain related ontological knowledge is used as a tool for enhancing the quality of design.
- Joint use of holonic and agency concepts:
  - **Holonic** modelling of collective and compositional aspects of the system.
  - **Agent** modelling of individual aspects and personal goals.

Outlines

Outlines

Janus Metamodel – Organizational Concepts

- Direct implementation of CRIO[1] organizational concepts.

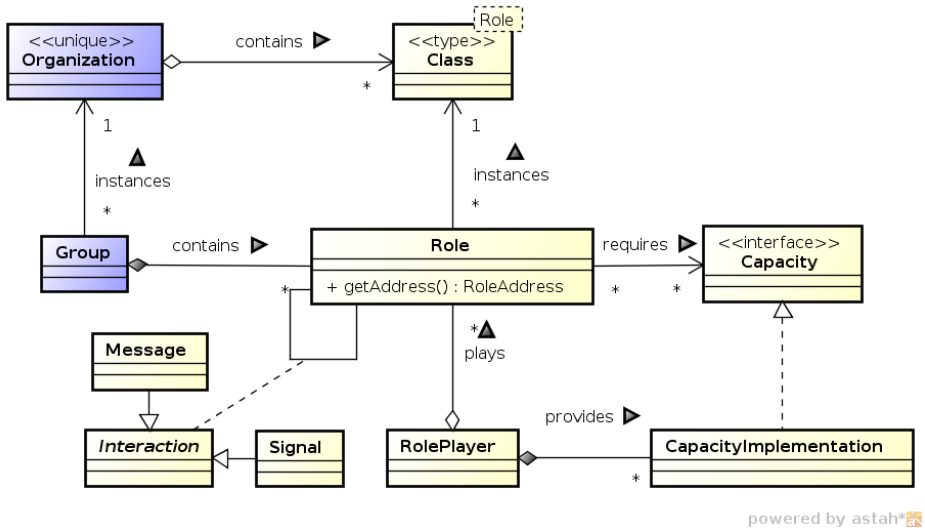


[1] Capacity, Role, Interaction, Organization

Janus Metamodel – Organizational Concepts

Organization

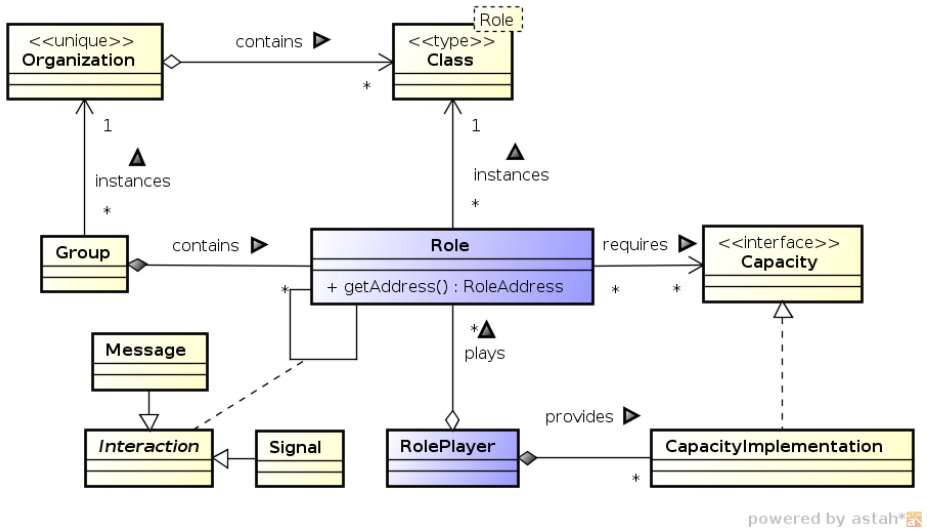
An organisation is defined by a collection of roles that take part in systematic institutionalised patterns of interactions with other roles in a common context. This context consists in shared knowledge and social rules/norms, social feelings, etc and is defined according to an ontology.



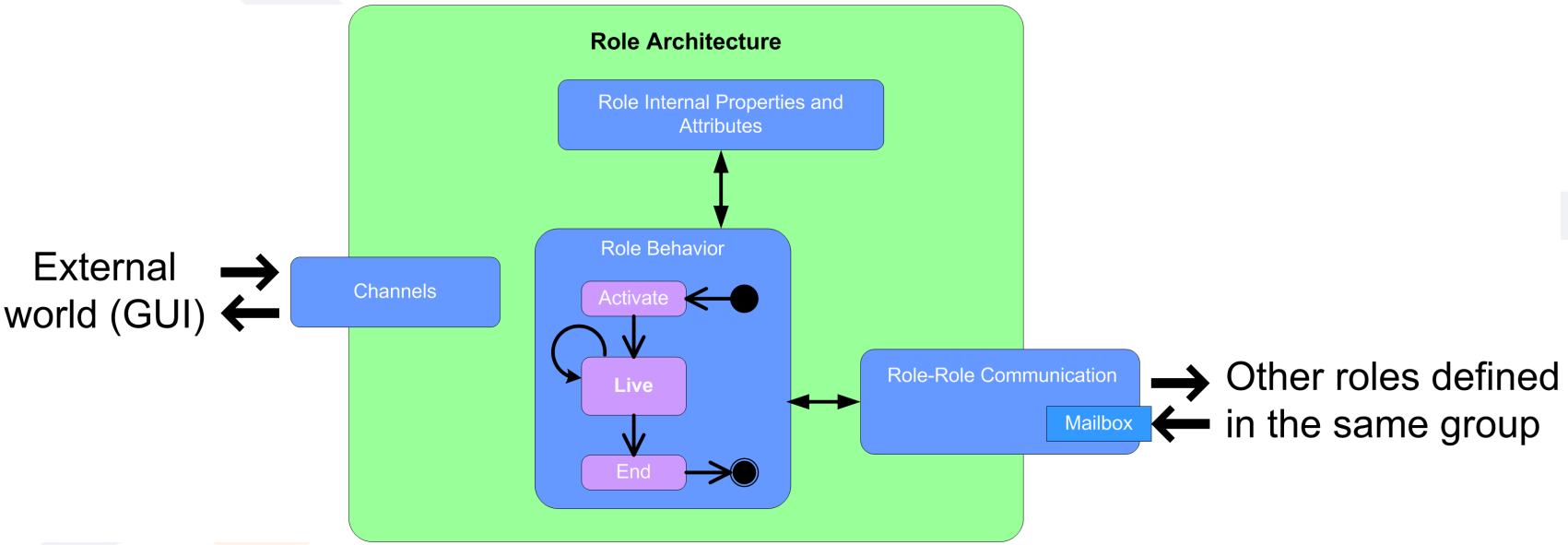
Janus Metamodel – Organizational Concepts

Role

An expected behaviour (a set of role tasks ordered by a plan) and a set of rights and obligations in the organization context.



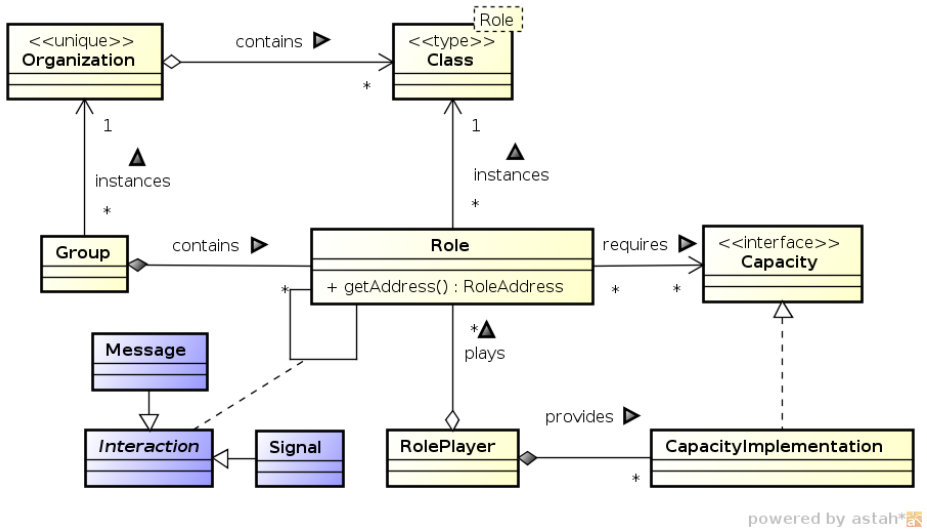
Janus Metamodel – Role Architecture



Janus Metamodel – Organizational Concepts

Interaction

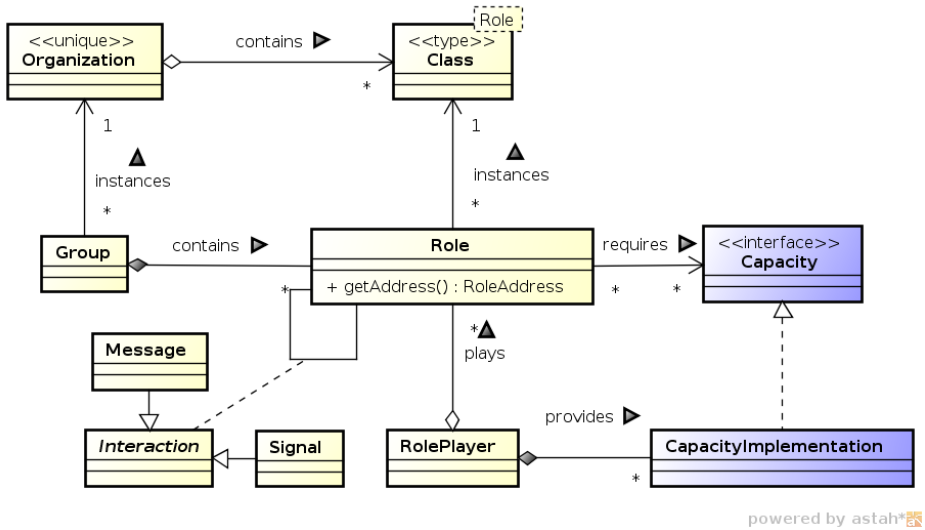
A dynamic, not a priori known sequence of events (a specification of some occurrence that may potentially trigger effects on the system) exchanged among roles, or between roles and entities outside the agent system to be designed. Roles may react to the events according to theirs behaviors.



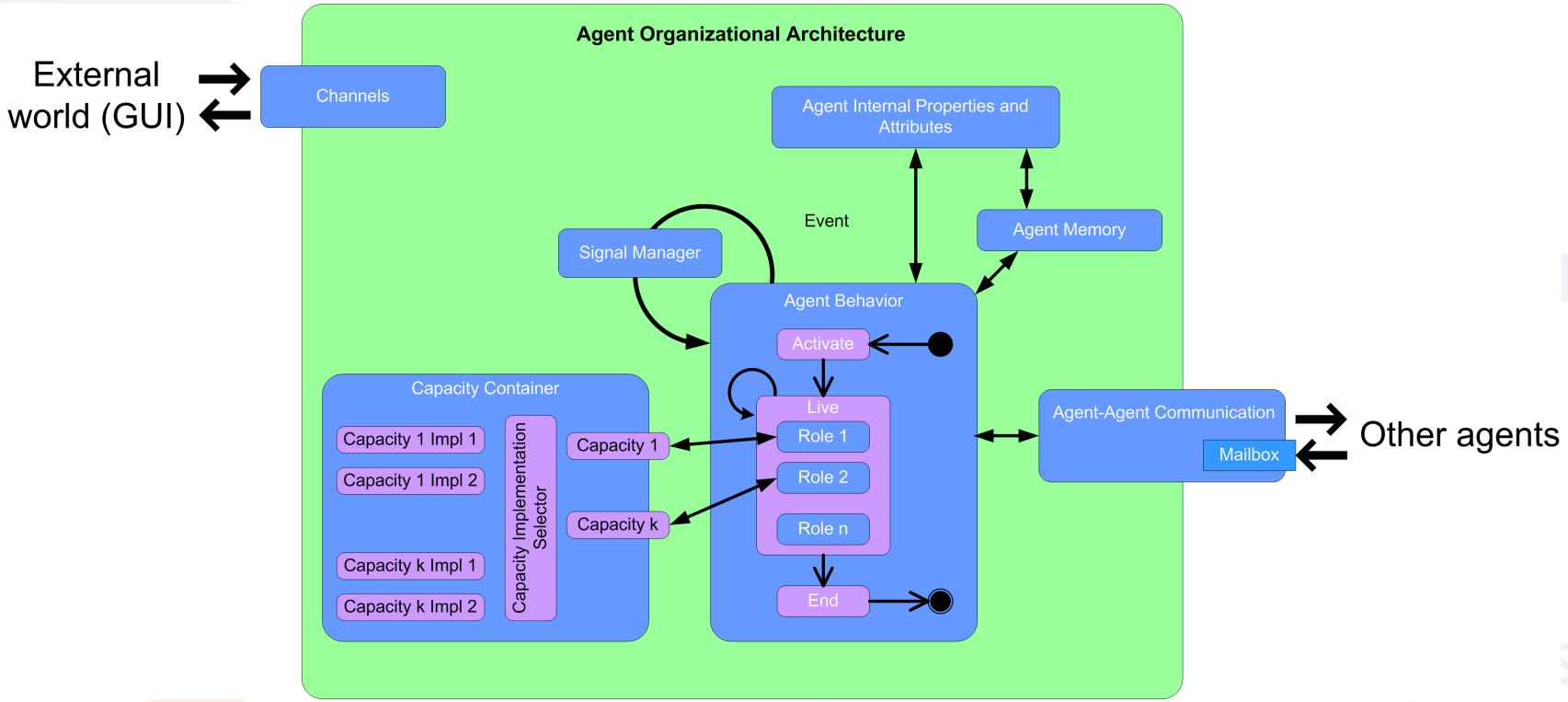
Janus Metamodel – Organizational Concepts

Capacity

A specification of a transformation of a part of the designed system. This transformation guarantees resulting properties if the system before the transformation satisfies a set of constraints. It may be considered as a specification of the pre- and post-conditions of a goal achievement.



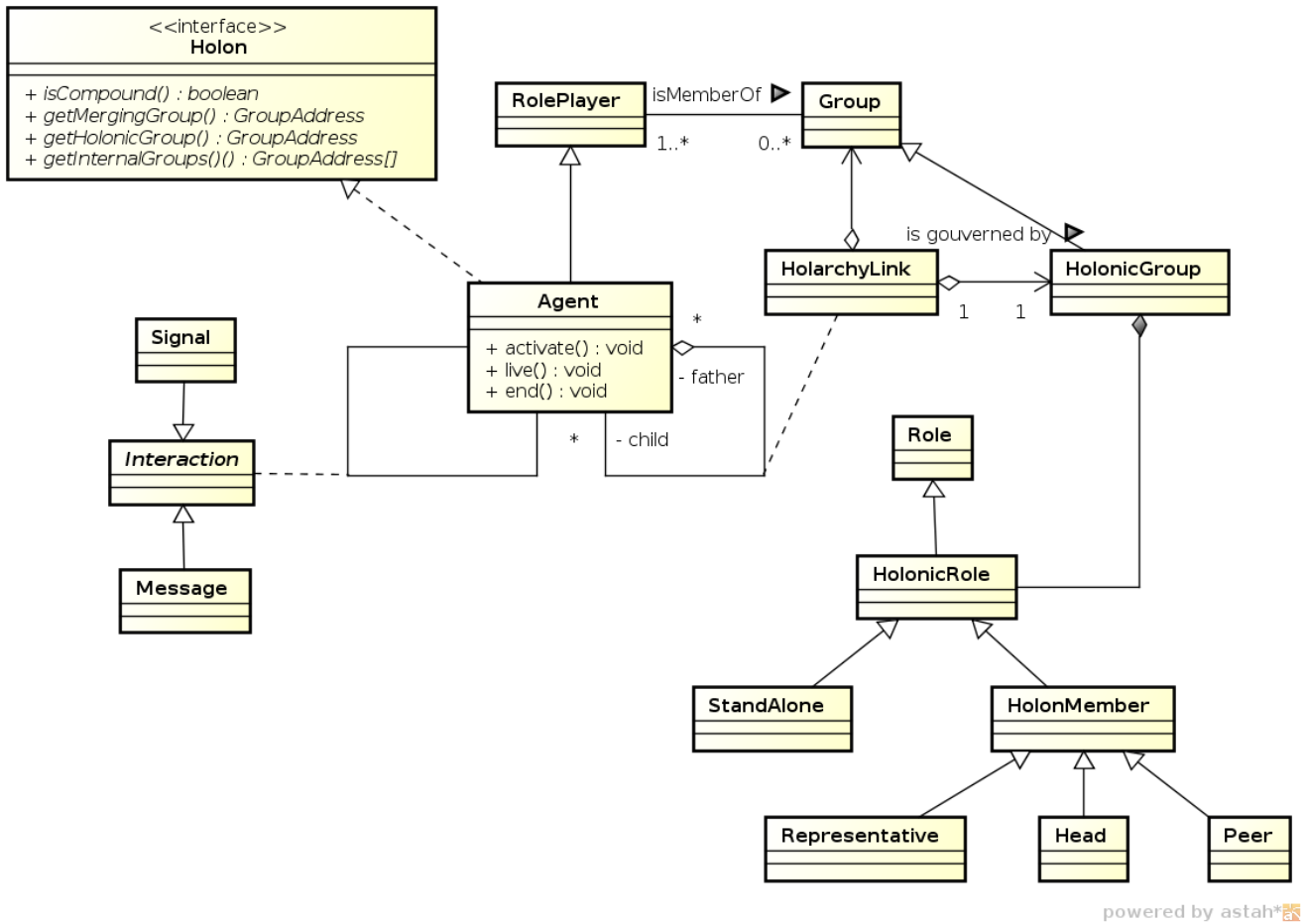
Janus Metamodel – Organizational Architecture of an agent



Outlines

Janus Metamodel – Agent-related Concepts

- Direct implementation of CRIO Agent-related concepts.



Janus Metamodel – Agent

Agent

An abstract rational entity that adopts a decision in order to obtain the satisfaction of one or more of its own goals. An autonomous entity may play a set of Agent Roles within various groups. These roles interact each other in the specific context provided by the entity itself. The entity context is given by the knowledge, the capacities owned by the entity itself. Roles share this context by the simple fact of being part of the same entity.

Janus Metamodel – Agent

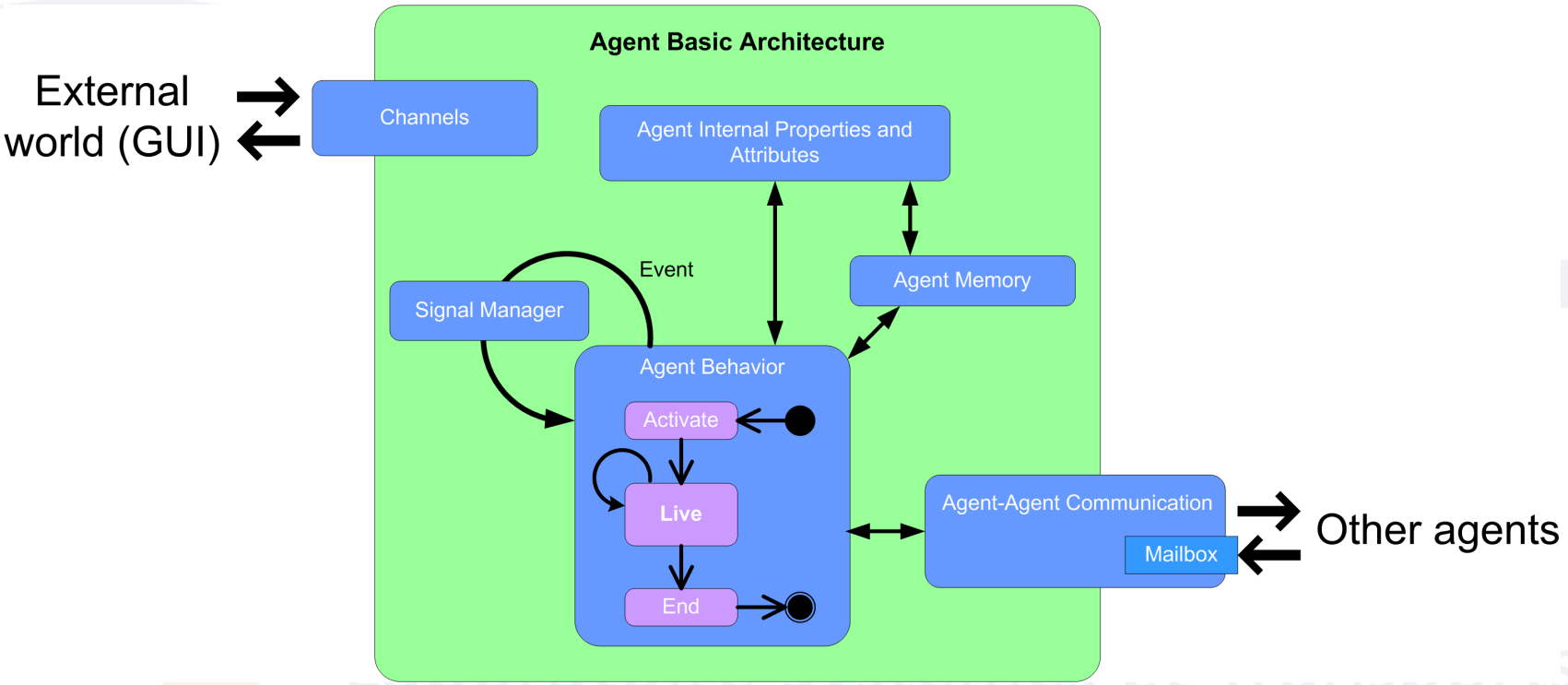
Agent

An abstract rational entity that adopts a decision in order to obtain the satisfaction of one or more of its own goals. An autonomous entity may play a set of Agent Roles within various groups. These roles interact each other in the specific context provided by the entity itself. The entity context is given by the knowledge, the capacities owned by the entity itself. Roles share this context by the simple fact of being part of the same entity.

Holon

A holon is a self-similar structure composed of holons as sub-structures. This hierarchical structure is called a holarchy. A holon may be seen, depending on the level of observation, either as an autonomous "atomic" entity or as a group of interacting holons.

Agent Architecture in Janus



Janus Metamodel – Agent Interaction

Message (Inter-agent communication)

An object sent by an agent — or one of its roles — to another agents – or these roles. — In addition to data, a Message contains the emitter and the receivers. In organizational point of view, a message is restricted to the role's organization. In agent point of view, messages are exchanged by agents without scope restriction.

Janus Metamodel – Agent Interaction

Message (Inter-agent communication)

An object sent by an agent — or one of its roles — to another agents – or these roles. — In addition to data, a Message contains the emitter and the receivers. In organizational point of view, a message is restricted to the role's organization. In agent point of view, messages are exchanged by agents without scope restriction.

Signal (Intra-agent communication)

An event inside the scope of an agent. This signal may be triggered by the agent itself or one of its roles. Listener on signals could be any object created and registered as signal listener by the agent itself or by one of its roles. A signal contains information and the address of the emitter, but not the signal receivers.

Janus Metamodel – Agent Interaction

Channel (Agent-Outside communication)

A bi-directional communication mean between an agent and the Environment. The Environment is composed of all objects that are not agents.
Channel could be restricted to one communication direction.
Channel is providing getter and setter functions to obtain informations from and influence the agent.
Two main types of channels are basically used: GUI binding, Probe on agents.

Janus Metamodel – Agent Mind State

Agent Attributes and Properties

Hard-coded values in the agent source code. They are fields in the agent class.

Janus Metamodel – Agent Mind State

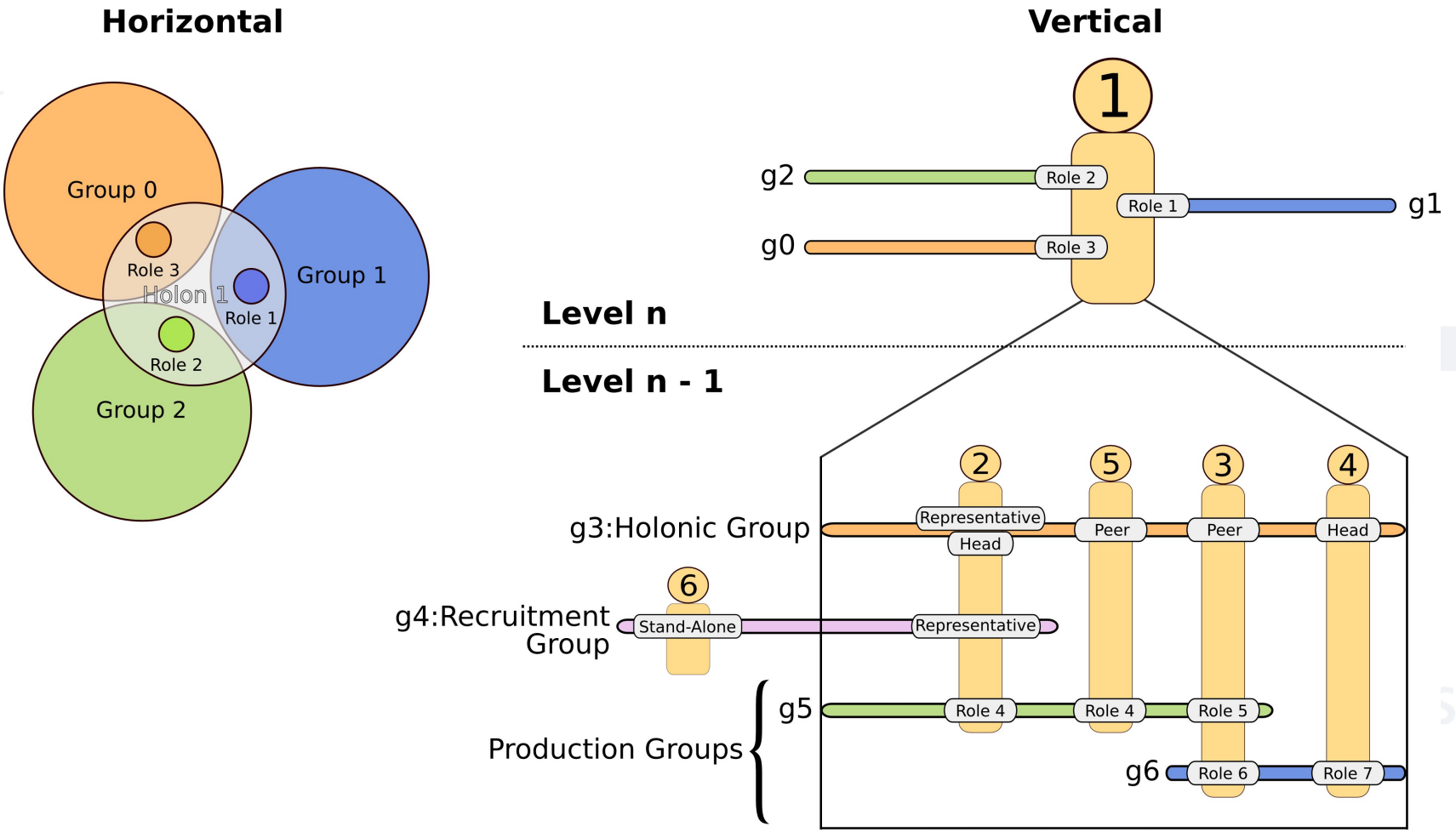
Agent Attributes and Properties

Hard-coded values in the agent source code. They are fields in the agent class.

Agent Memory

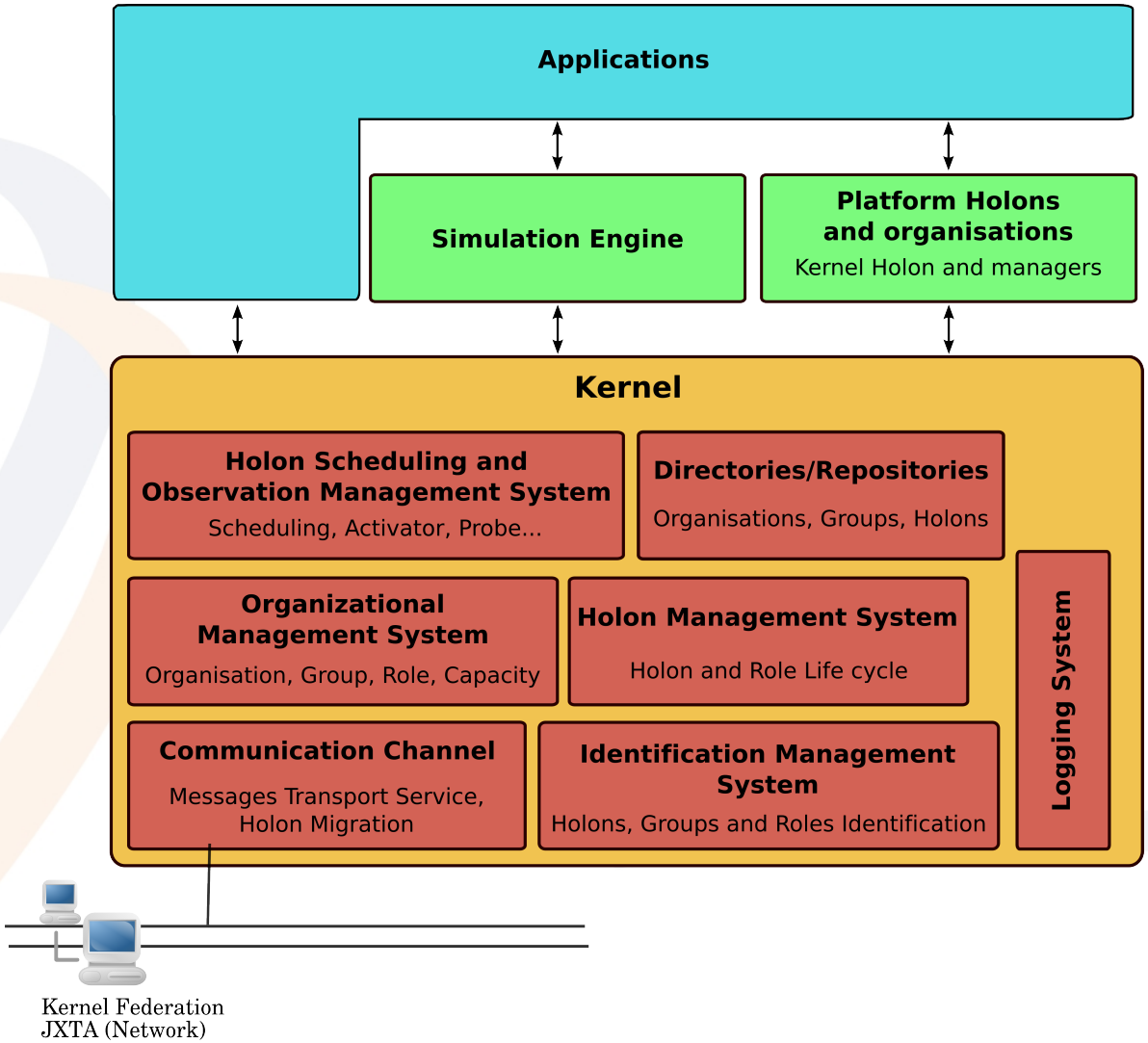
A memory space owned by the agent in which several data may be stored. In addition to previous attributes and properties, agent is able to exhibit a specific type of memory architecture: blackboard, etc.

Holon in Janus

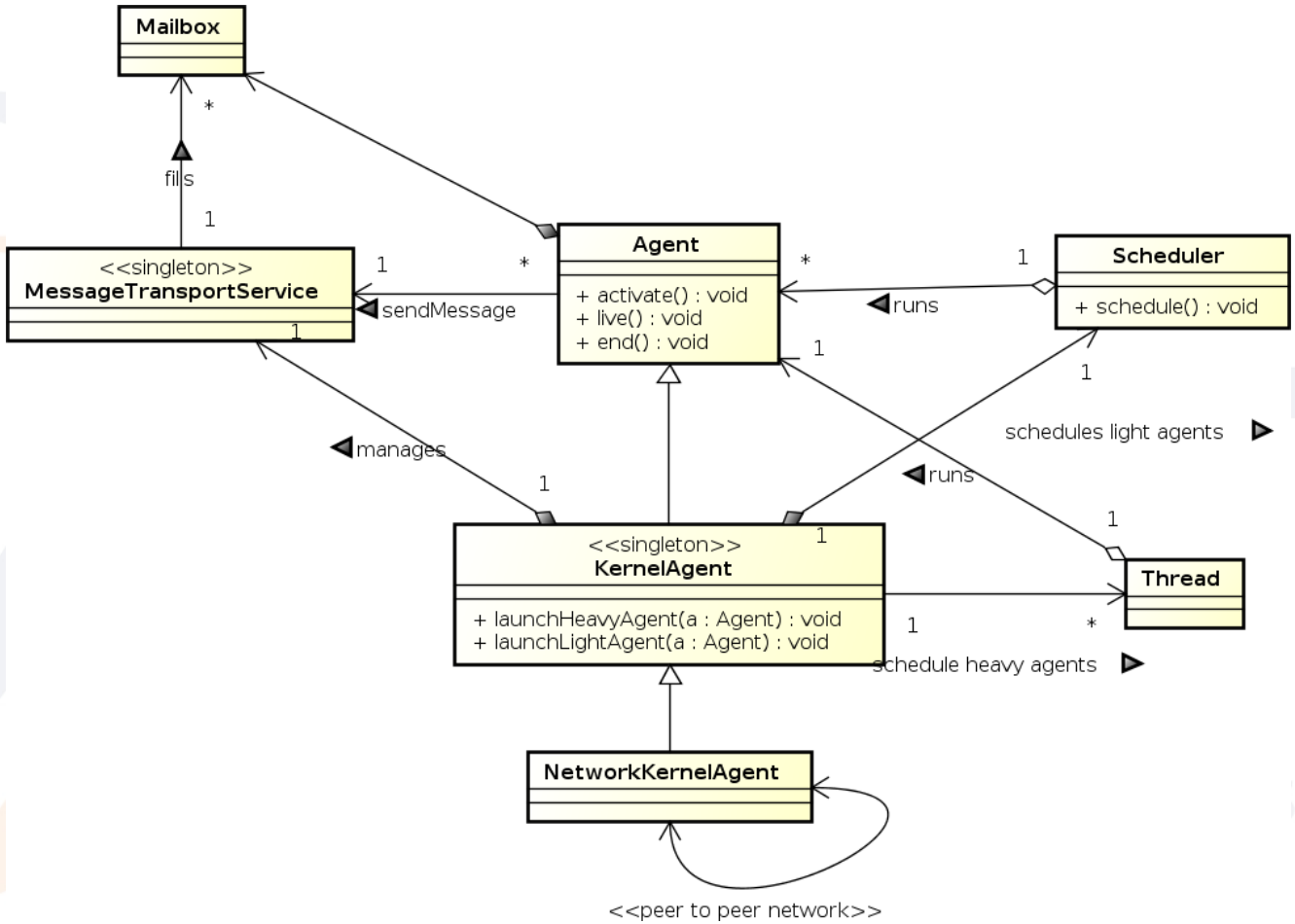


Outlines

Janus Platform Architecture



Janus Metamodel – Execution/life-cycle related concepts



Implementation Metamodel – Kernel

Organizational Management System

- manages roles and organisations and their instantiations in form of groups.
- dynamic acquisition, instantiation and liberation of roles,
- dynamic acquisition and execution of capacities.

Implementation Metamodel – Kernel

Organizational Management System

- manages roles and organisations and their instantiations in form of groups.
- dynamic acquisition, instantiation and liberation of roles,
- dynamic acquisition and execution of capacities.

Life Cycle Management and Execution Policies

- management of the holon life cycle management
- various role execution policies
- a concurrent execution model.
- a synchronous engine (MABS).

Implementation Metamodel – Kernel

Intra- and extra-Kernel Message Transport Service

- various policies for the messages management (i.e. MABS)
- role-oriented communication
- control the exchange of messages within the platform.
- event-based communication
- network support

Implementation Metamodel – Kernel

Intra- and extra-Kernel Message Transport Service

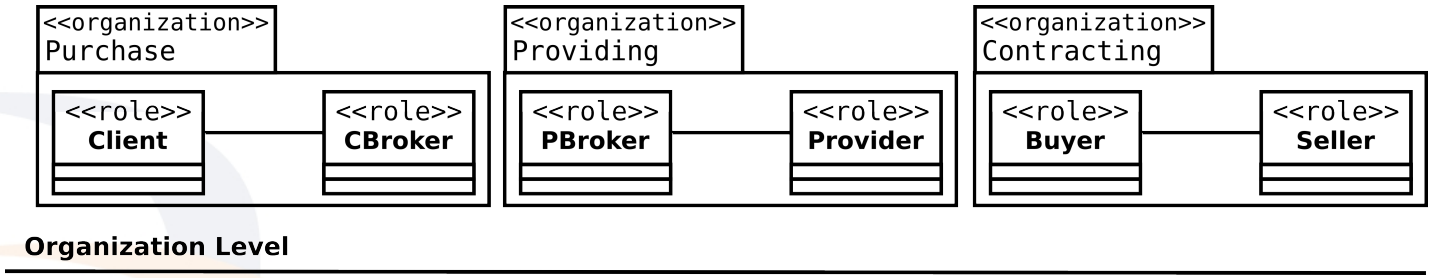
- various policies for the messages management (i.e. MABS)
- role-oriented communication
- control the exchange of messages within the platform.
- event-based communication
- network support

Observation Management System

- provides instrumentation based on probes allowing a role to observe another role.
- channels to GUI

Outlines

Market-like Community Example – How to define an organization



1. Client is requesting to Broker the best offer for a service.
2. Broker is contacting Providers for offer submission and select the best one.
3. Broker is giving back to Client the best offer.
4. Client and Provider are contracting.

Market-like Community Example – How to define an organization

ProvidingOrganization.java

```
public class ProvidingOrganization extends Organization {
    public ProvidingOrganization(CRIOContext context){
        super(context);
        addRole(Provider.class);
        addRole(PBroker.class);
    }
}
```

ContractingOrganization.java

```
public class ContractingOrganization extends Organization {
    public ContractingOrganization(CRIOContext context){
        super(context);
        addRole(Buyer.class);
        addRole(Seller.class);
    }
}
```

Market-like Community Example – How to define a role

PBroker.java

```
public class PBroker extends Role {
    private State state;
    public PBroker() {
        addObtainCondition(
        new HasAllRequiredCapacitiesCondition(
            FindLowestCostProposalCapacity.class,
            FindShortestTimeProposalCapacity.class)));
    }

    public Status activate(Object... parameters) {
        this.state = State.WAITING_FOR_CLIENT_BROKER;
        getSignalEmitter().addSignalListener(this.signalListener);
        return StatusFactory.ok(this);
    }
```

Market-like Community Example – How to define an organization

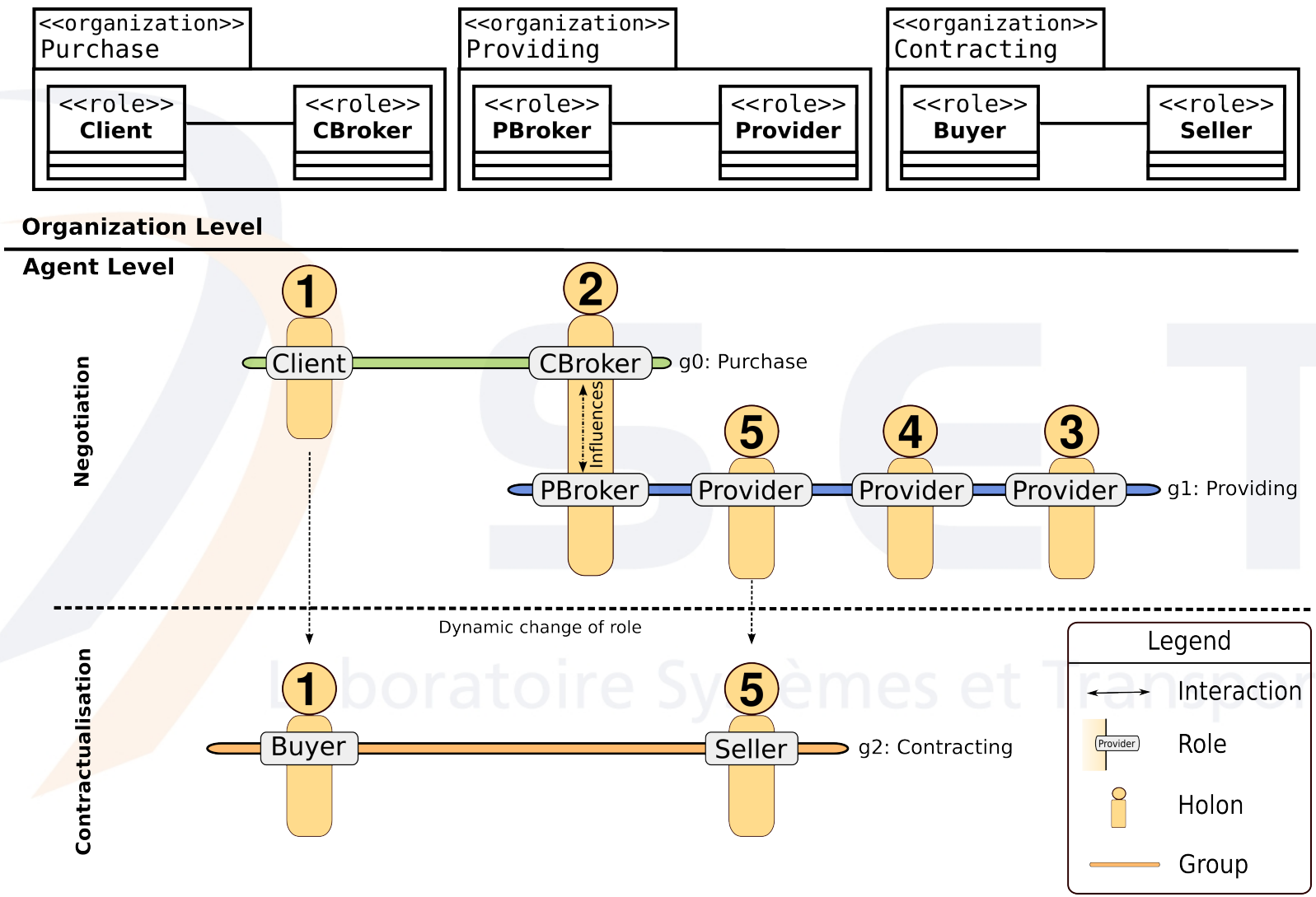
PBroker.java

```
        public enum State {
                WAITING_FOR_CLIENT_BROKER,
                WAITING_FOR_PROVIDER_READY,
                CONTACT_PROVIDER,
                WAIT_PROVIDER_PROPOSAL,
                SELECT_PROPOSAL,
                NOTIFY_PROVIDERS,
                WAIT_CONTRACT_GROUP;
        }

        public Status live() {
                switch (this.state) {
                case WAITING_FOR_CLIENT_BROKER: break;
                case WAITING_FOR_PROVIDER_READY: break;
                case CONTACT_PROVIDER: break;
                case WAIT_PROVIDER_PROPOSAL: break;
                case SELECT_PROPOSAL: break;
                case WAIT_CONTRACT_GROUP: break;
                }
                return StatusFactory.ok(this);
        }

}
```

Market-like Community Example – How to define an agent



Market-like Community Example – How to define an agent

BrokerAgent.java

```
public class BrokerAgent extends Agent {
    public Status activate(Object... params) {
        CapacityContainer cc = getCapacityContainer();
        cc.addCapacity(new FindLowestCostProposalCapacityImpl());
        cc.addCapacity(new FindShortestTimeProposalCapacityImpl());

        GroupAddress clientGA = getOrCreateGroup(PurchaseOrganization.class);
        if (!requestRole(CBroker.class, clientGA)) {
            return StatusFactory.cancel(this);
        }

        GroupAddress providerGA = getOrCreateGroup(ProvidingOrganization.class);
        if (!requestRole(PBroker.class, providerGA, this.providerCount)) {
            return StatusFactory.cancel(this);
        }
        return StatusFactory.ok(this);
    }
}
```

Outlines

Conclusion

Developer Features — An Integrated Approach

- Direct integration with the ASPECS[2] methodology and CRIO metamodel.
- Contribute to fill the gap between design and implementation phases in AOSE.
- A full set of facilities for launching, displaying, developing and monitoring agents, holons, organisations, groups, roles...
- A developer-oriented platform designed to develop applications dealing with real case studies (large, complex, etc).

[2] see: www.aspecs.org

Conclusion

Janus Kernel Development Constraints

- Complete support of organisational concepts as first-class abstractions: Role and Organisation.
- Sake of preserving computational performance (realtime, mobile devices, embedded)
- Sake of strictly respecting agent fundamental characteristics: autonomy.

Successfully Used In

- Crowd simulation in virtual reality
- Energy management of buildings
- Chat rooms

Perspectives

Agent Architecture

- Complete ACL and BDI modules - February 2011
- Complete FIPA compliance.

Perspectives

Agent Architecture

- Complete ACL and BDI modules - February 2011
- Complete FIPA compliance.

Development Tools

- Java Micro-Edition fully compliance, deployment on Google Android - August 2011
- A CASE tools to enable automatic code generation and assist the deployment.

Perspectives

Agent Architecture

- Complete ACL and BDI modules - February 2011
- Complete FIPA compliance.

Development Tools

- Java Micro-Edition fully compliance, deployment on Google Android - August 2011
- A CASE tools to enable automatic code generation and assist the deployment.

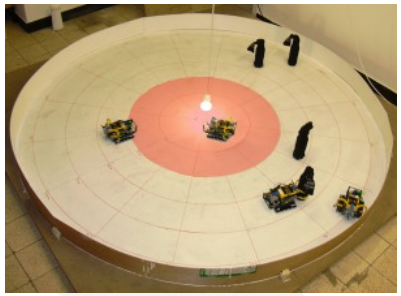
Deployment Features

- Java Micro-Edition fully compliance, deployment on Google Android - August 2011

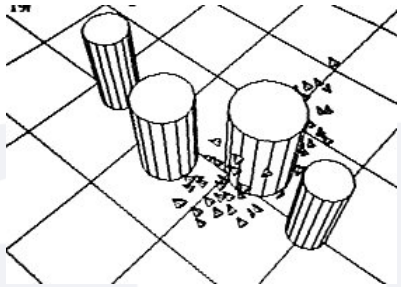
Outlines

Simple Demos on Janus

Ant colony



Forager Bots



Boids

Available on `http://www.janus-project.org`

Movie: Pedestrian Simulation in Metro Station from Voxelia

voxelia

Realtime pedestrian simulation in urban environment.
available on Janus-Jasim (Continuous Environment)

UNIVERSITÉ DE TECHNOLOGIE DE BELFORT-MONTBÉLIARD

JANUS: Another Yet General-Purpose Multiagent Platform

Seventh AOSE Technical Forum – December 15th, 2010 – Paris

SET
Laboratoire Systèmes et Transports

utbm