Multi-Agent Activity Modeling with the Brahms Environment

An Introductory Tutorial

at RuleML 2013

ejenta

Maarten Sierhuis
sierhuis@ejenta.com

Ejenta

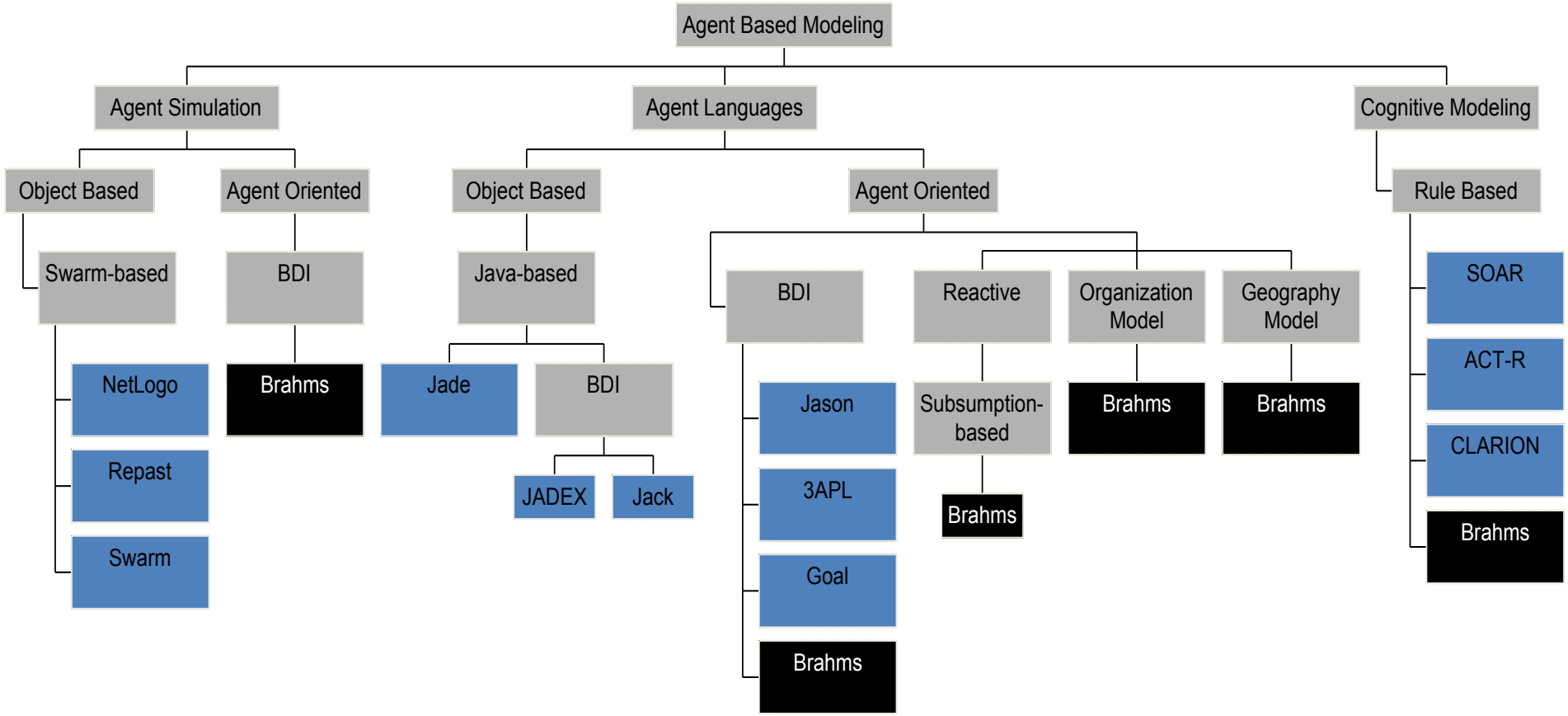
- We provide
  - Brahms Agent Environment
  - Modeling and simulation of people, interactions, systems and operations
  - Agent-based Mission Operations Procedure Execution & Workflow Environment (currently running in NASA's Mission Control for the ISS)
  - Intelligent Personalized Agents for Enterprises and Consumers

- We provide an unique capability with our Simulation to Implementation methodology
  - Building agent-based models and simulations of people, teams and interactions and develop real-world multi-agent systems based on these

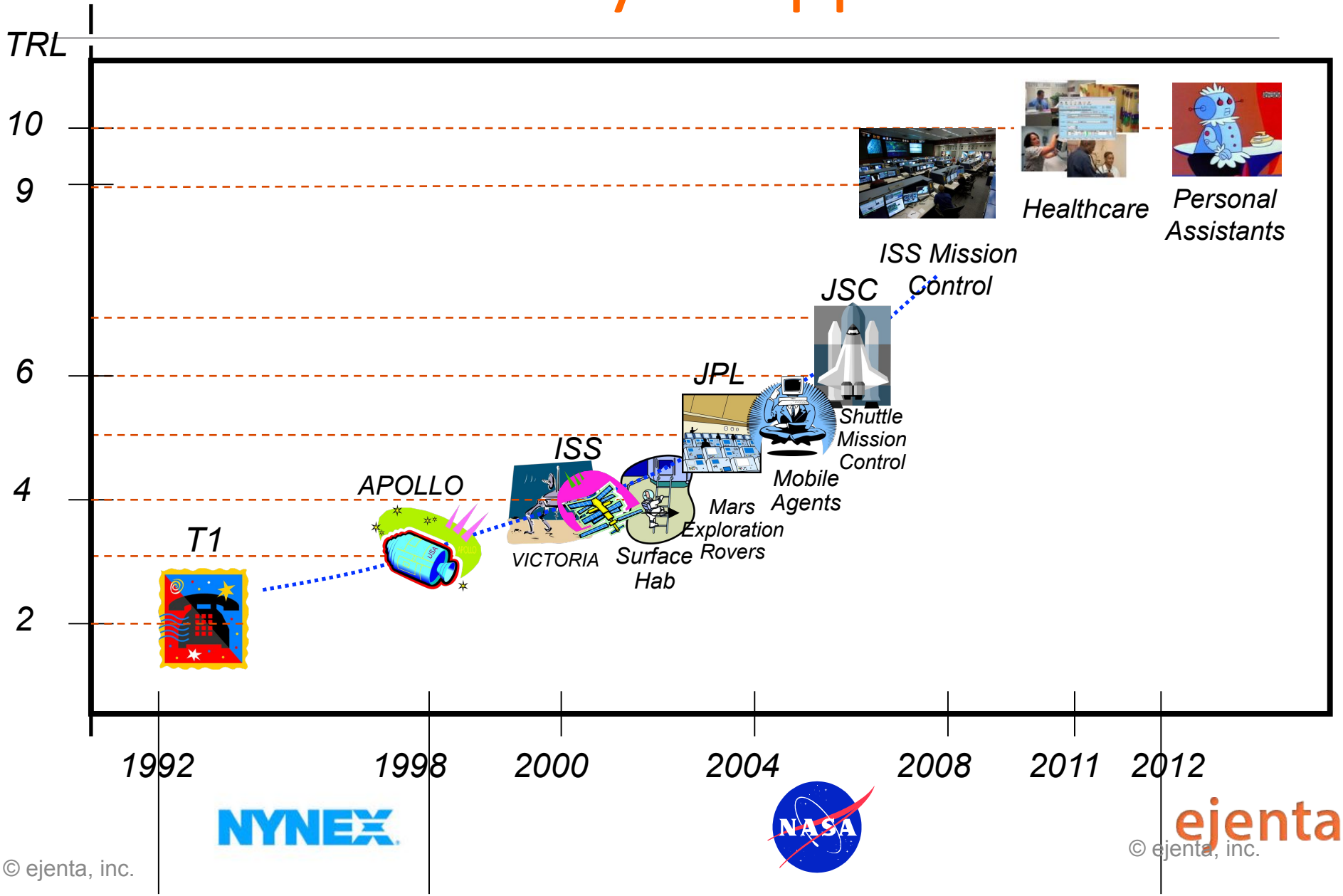
- We integrate with existing real-world systems, sensor networks and interfaces
  - Voice interfaces
  - Mobile devices
  - Internet TV devices
  - In-home and in-car sensor networks
  - Wearable sensors & body networks

- Customers include
  - NASA
  - DARPA
  - United Space Research Association
  - Kaiser Permanente

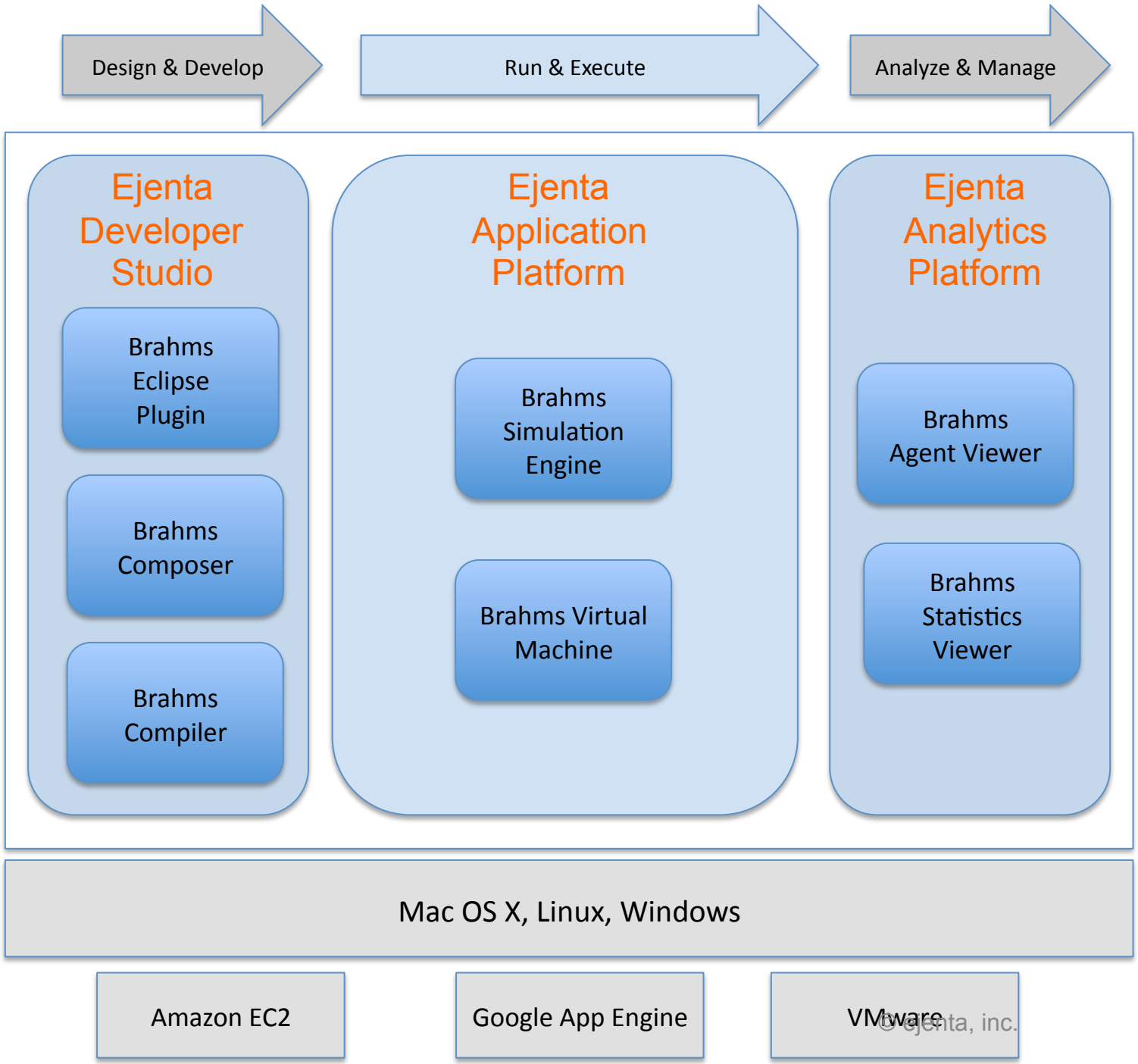
ABM Languages



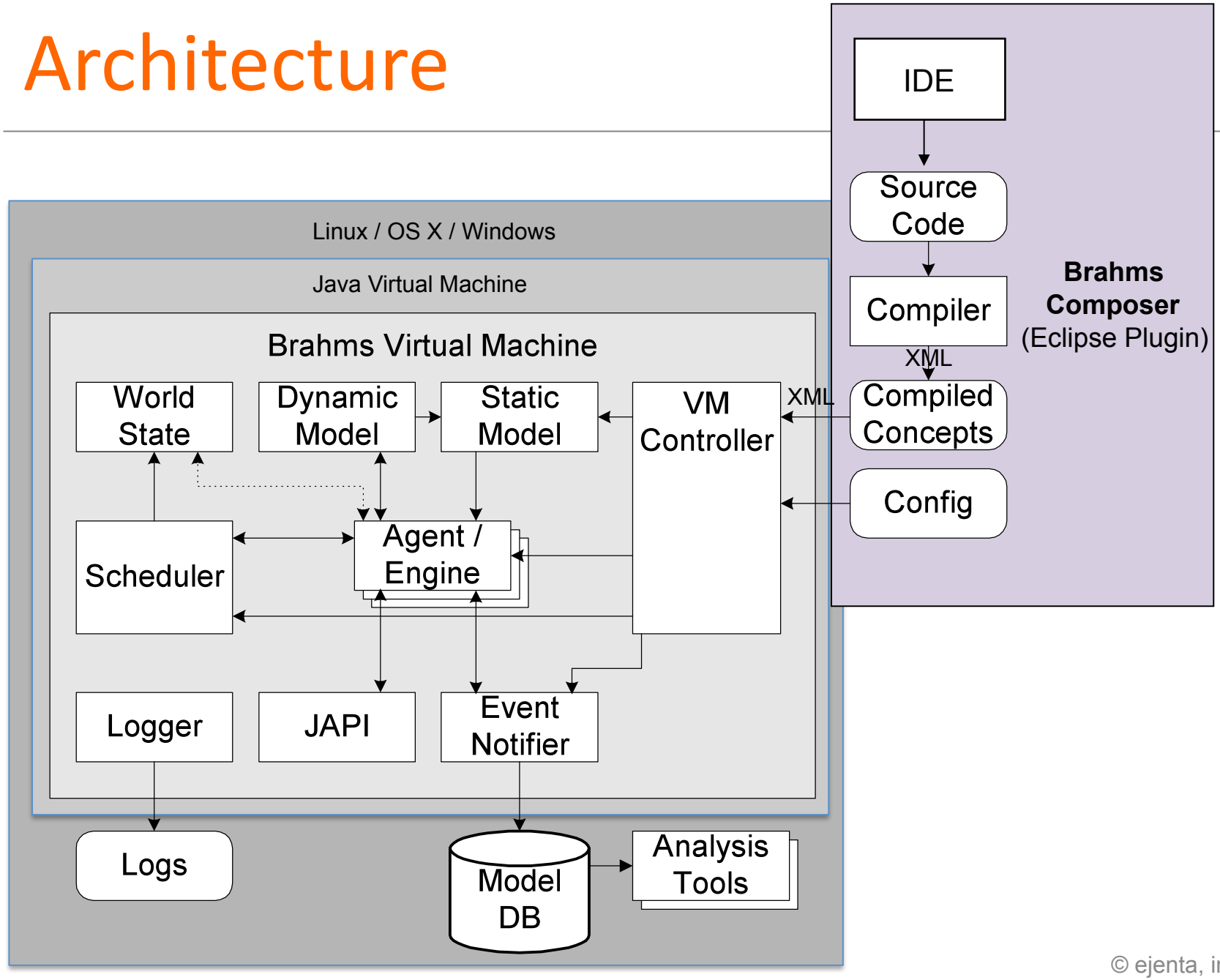
Brahms – History of applications



© ejenta, inc.


Design & Develop
Run & Execute
Analyze & Manage
Ejenta Developer Studio
Brahms Eclipse Plugin
Brahms Composer
Brahms Compiler
Ejenta Application Platform
Brahms Simulation Engine
Brahms Virtual Machine
Ejenta Analytics Platform
Brahms Agent Viewer
Brahms Statistics Viewer
Mac OS X, Linux, Windows
Amazon EC2
Google App Engine
VMware
© ejenta, inc.

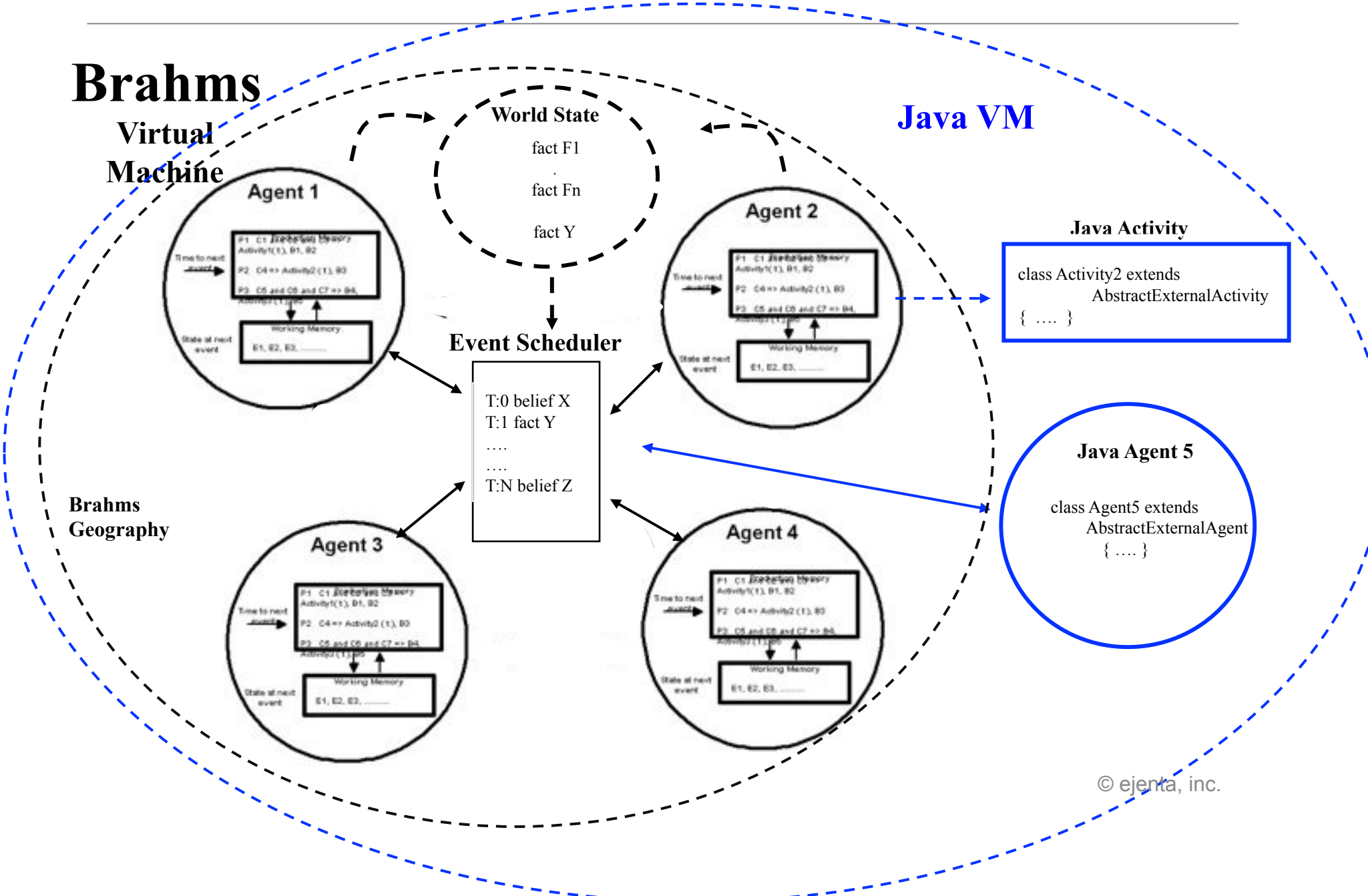
Architecture



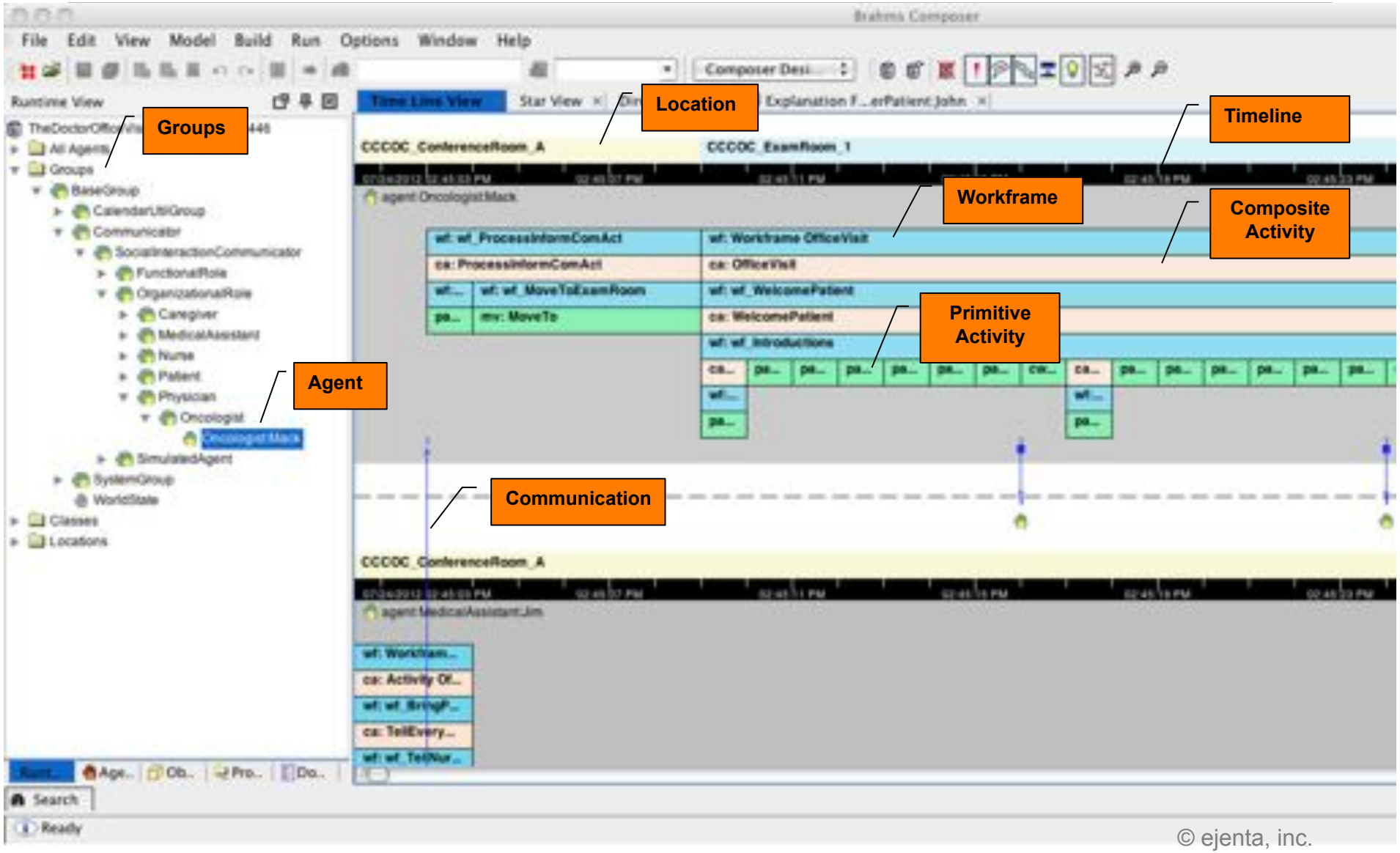
Brahms Language

- **Agent Oriented / BDI**
  - Agents are first-class citizens
  - Agents are belief based
  - Agents are intention based:
    - Beliefs become intentions that trigger reasoning- and/or situation-action rules
  - Agents can communicate
- **Organizational Modeling**
  - Agents can be modeled within a hierarchical member-of inheritance structure
- **Object-based**
  - Objects can represent physical artifacts, data and concepts to reason with
  - Integration of Java objects as data objects, Java activities and Java agents
- **Geography-based**
  - Areas can be conceptual representations of locations
  - Areas can be located within other areas, creating a hierarchical environment model
  - Agents and objects can be located within an area

Multiagent Language



Agent Timeline Viewer



Agents, Groups, Beliefs, Facts, Activities and Workframes

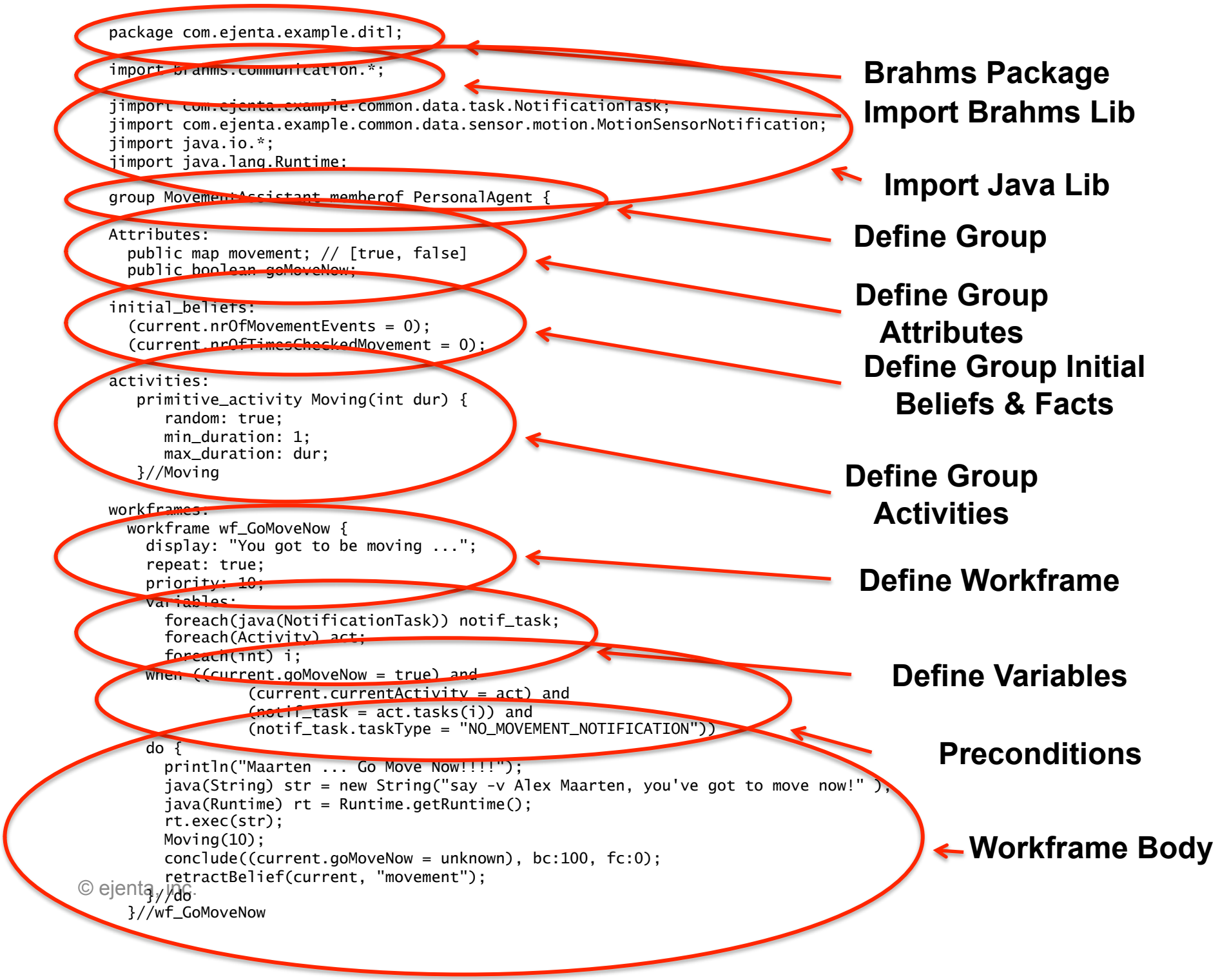
What is a Brahms agent?

- Agents model *human behavior*.
- Agents could be *autonomous intelligent systems*
- Attributes of an agent:
  - autonomy,
  - social ability,
  - reactivity,
  - pro-activeness,
  - mobility
  - bounded rationality.

© ejenta, inc.

What is a Brahms group?

- A Brahms group describes the abstract properties and behaviors of a group of agents
- Types of groups:
  - Functional
  - Organizational
  - Social
  - Community of Practice
- Groups can be members of multiple groups
- Agents can be members of multiple groups

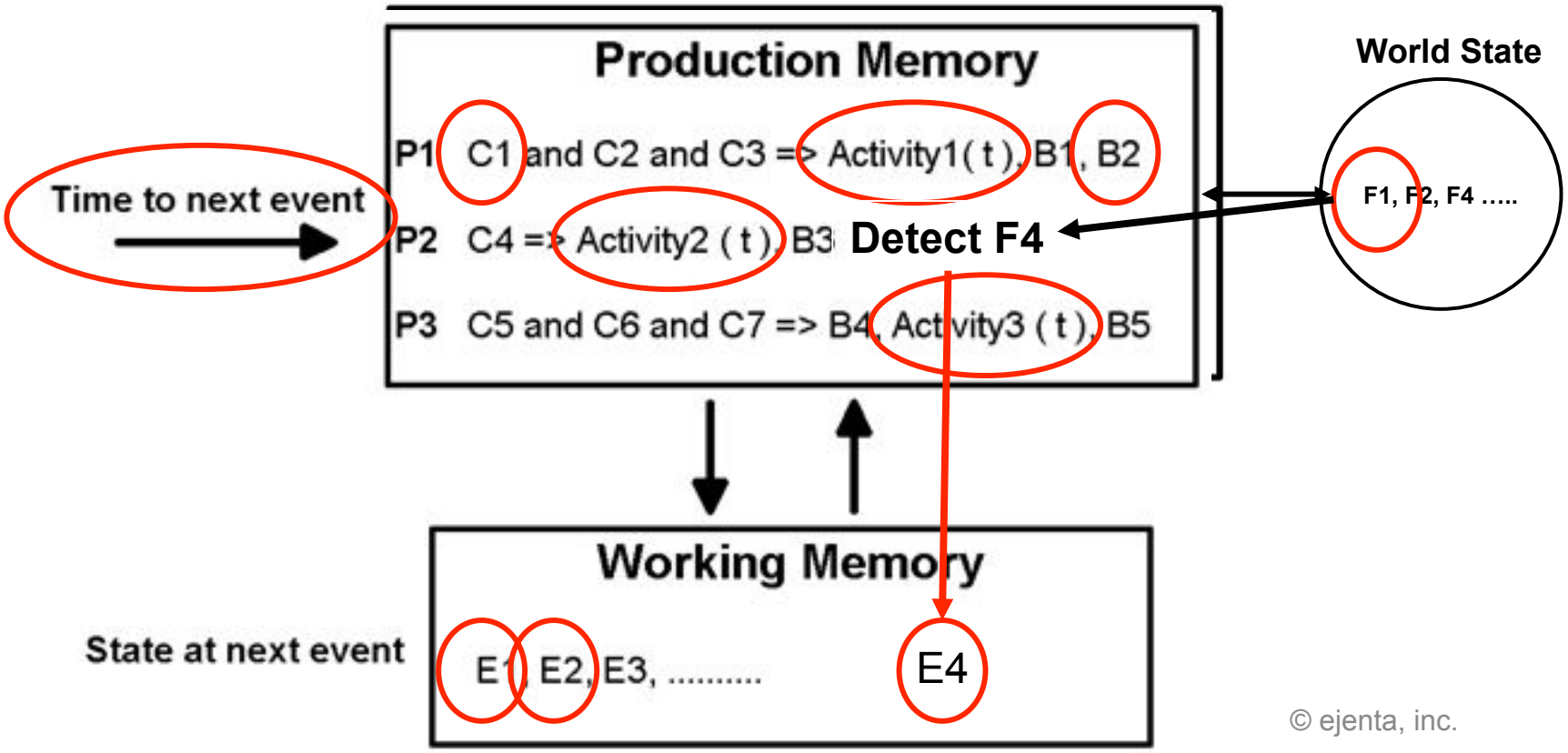

package com.ejenta.example.ditl;
import brahms.communication.*;
jimport com.ejenta.example.common.data.task.NotificationTask;
jimport com.ejenta.example.common.data.sensor.motion.MotionSensorNotification;
jimport java.io.*;
jimport java.lang.Runtime;
group MovementAssistant memberof PersonalAgent {
Attributes:
public map movement; // [true, false]
public boolean goMoveNow;
initial_beliefs:
(current.nrOfMovementEvents = 0);
(current.nrOfTimesCheckedMovement = 0);
activities:
primitive_activity Moving(int dur) {
random: true;
min_duration: 1;
max_duration: dur;
}//Moving
workframes:
workframe wf_GoMoveNow {
display: "You got to be moving ...";
repeat: true;
priority: 10;
variables:
foreach(java(NotificationTask)) notif_task;
foreach(Activity) act;
foreach(int) i;
when ((current.goMoveNow = true) and
(current.currentActivity = act) and
(notif_task = act.tasks(i)) and
(notif_task.taskType = "NO_MOVEMENT_NOTIFICATION"))
do {
println("Maarten ... Go Move Now!!!!");
java(String) str = new String("say -v Alex Maarten, you've got to move now!");
java(Runtime) rt = Runtime.getRuntime();
rt.exec(str);
Moving(10);
conclude((current.goMoveNow = unknown), bc:100, fc:0);
retractBelief(current, "movement");
}//do
}//wf_GoMoveNow
Brahms Package
Import Brahms Lib
Import Java Lib
Define Group
Define Group Attributes
Define Group Initial Beliefs & Facts
Define Group Activities
Define Workframe
Define Variables
Preconditions
Workframe Body

Time-based Situation-Action

Brahms Workframes

Situation-action Rules represent Behavioral Relationships!!

Production Rules represent *Qualitative Relationships*!!



Workframe Syntax

```
activities:
    primitive_activity eat( )  {
        priority: 0;
        max_duration: 400;
    }

workframe wf_eat {
 repeat: true;
 variables:
  forone(Cash) cs;
  forone(Diner) dn;
 when(knownval(current hasCash cs) and
    knownval(current.location = dn.location))
 do {
  eat();
  conclude((current.howHungry = current.howHungry - 3.00), bc:100, fc:0);
  conclude((cs.amount = cs.amount - dn.foodcost), bc:100, fc:100);
  conclude((current.readyToLeaveRestaurant = true), bc:100, fc:0);
 }
}
```

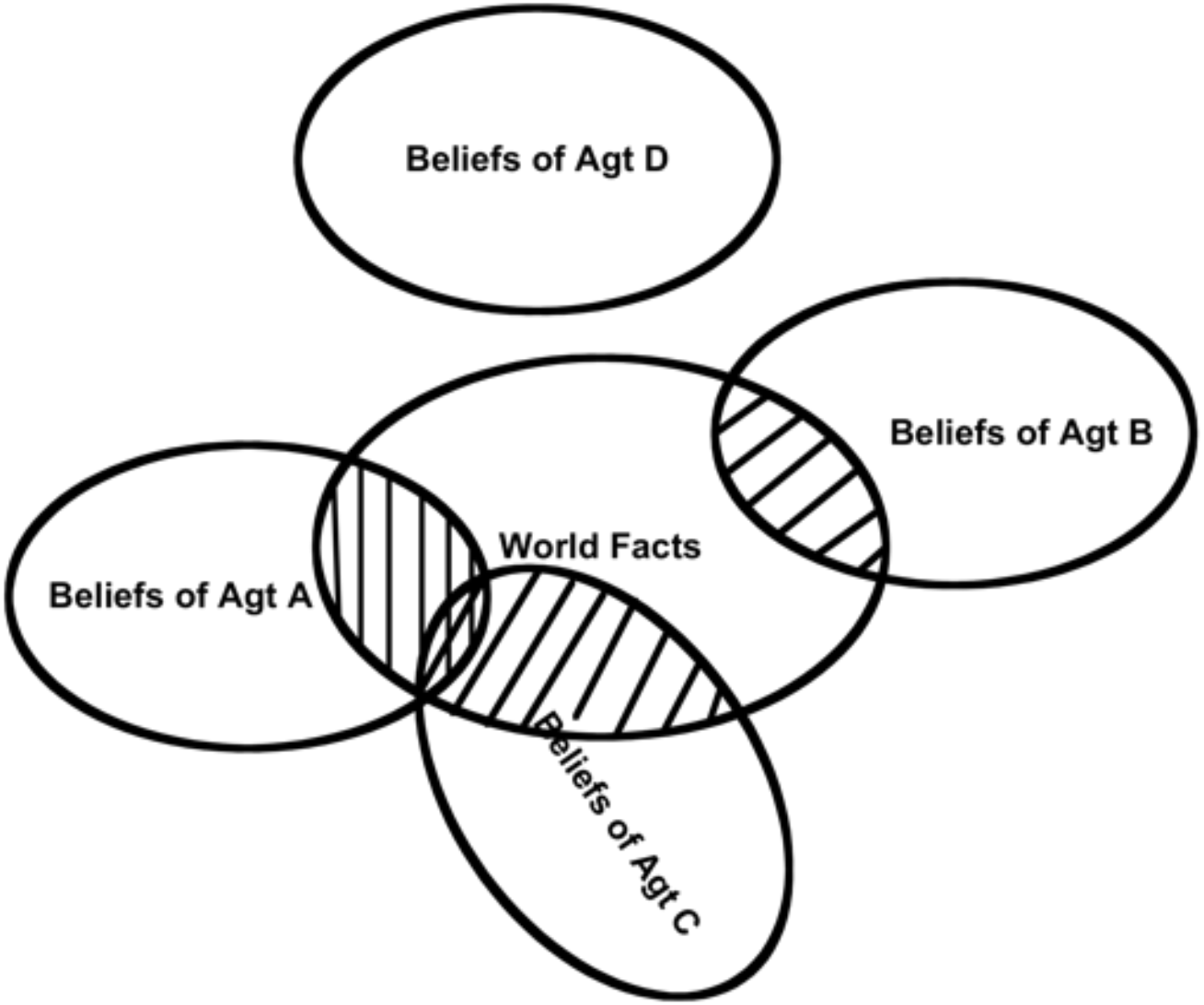
```
workframe workframe-name
{
   { display : ID.literal-string ; }
   { type : factframe | dataframe ; }
   { repeat : ID.truth-value ; }
   { priority : ID.unsigned ; }
   { variables : [ VAR.variable ]* }
   { detectables : [ DET.detectable ]*}
   { when ( { [ PRE.precondition ] [ and PRE.precondition ]
* } ) |
     do { [ PAC.activity-ref | CON.consequence ]* } }
}
```

Brahms Activities

- Primitive activities
  - Lowest level, user-defined, but not further specified.
  - Parameters are time, and resources
- Predefined activities
  - Primitive activities with predefined semantics (communicate, move, etc.)
- Composite activities
  - User-defined detailed activities
  - Decomposed in sub-activities
  - Describes what an agent does while “in” the activity
- Java activities
  - User-defined primitive activities that are implemented in a Java class
  - Uses the Brahms API.

Facts, Beliefs and Detectables

Facts and Beliefs



Brahms Detectables (for reactive behavior)

- Associated with workframes and activities
- Active while a workframe/activity is active
- Used for:
  - Agents noticing states of the world, and being able to act upon those
    - 3-steps: (i) detect fact, (ii) notice (fact becomes belief), (iii) conditionally act on belief
  - Control the execution of workframes and activities
    - Example: do act A **until** you notice fact F
- Type: **continue** | **impasse** | **abort** | **complete** | **end_activity**

Wait for Reply Detectable

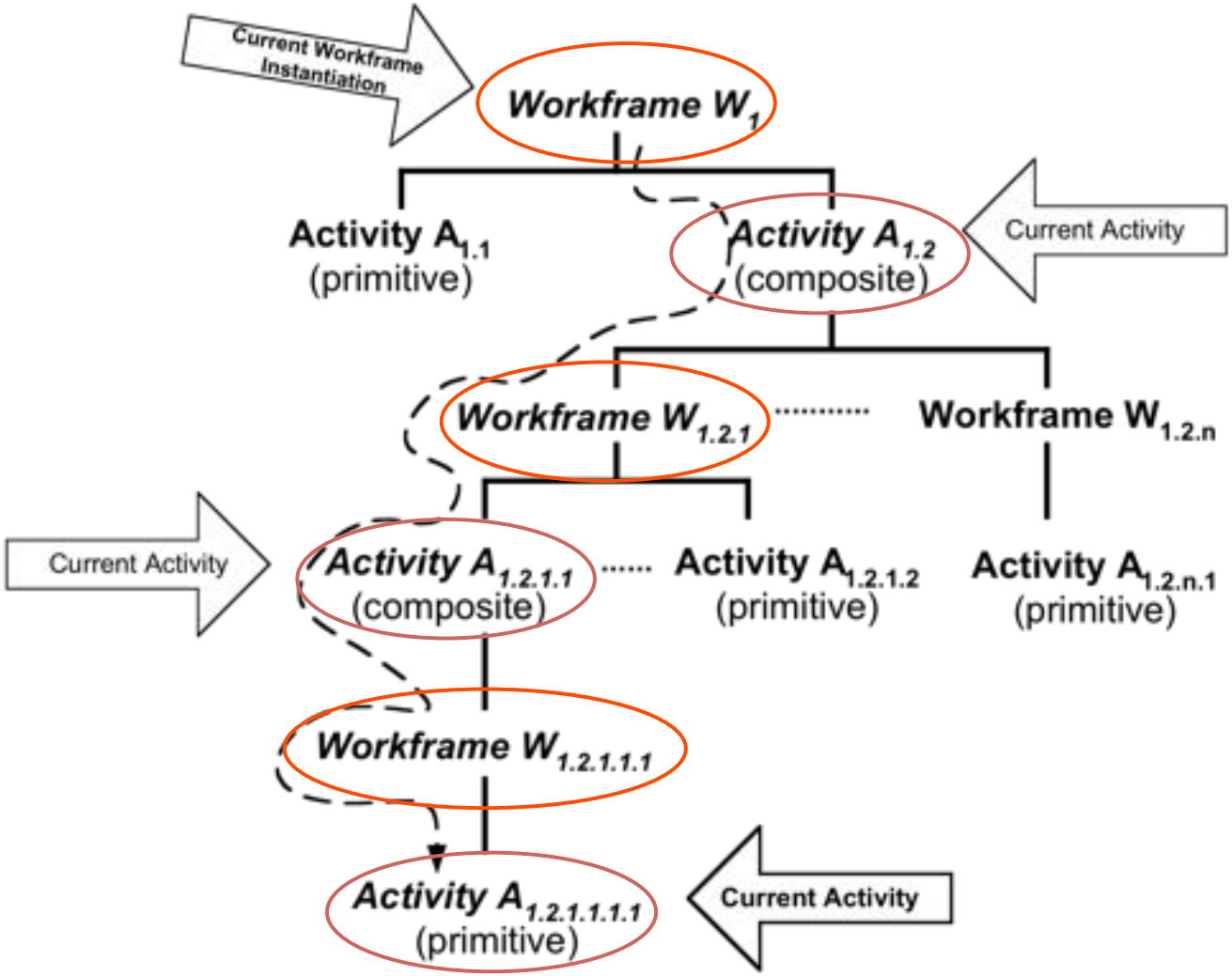
```
composite_activity WaitAndProcessReply (ReplyAgent agt ) {
    end_condition: detectable;
    detectables:
        detectable dt_wait_for_reply {
            when (whenever)
                detect((agt.replied = yes))
            then end_activity;
        }
    activities:
        composite_activity ProcessReply(ReplyAgent agt) { … }

    workframes:
        workframe wf_Replied_n {
            when (knownval(agt.answer = some_answer))
            do {
                ProcessReply(agt);
                conclude((agt.replied = yes), bc:100, fc:0);
            }
        }
}
```

Composite Activities, Interruption and Subsumption

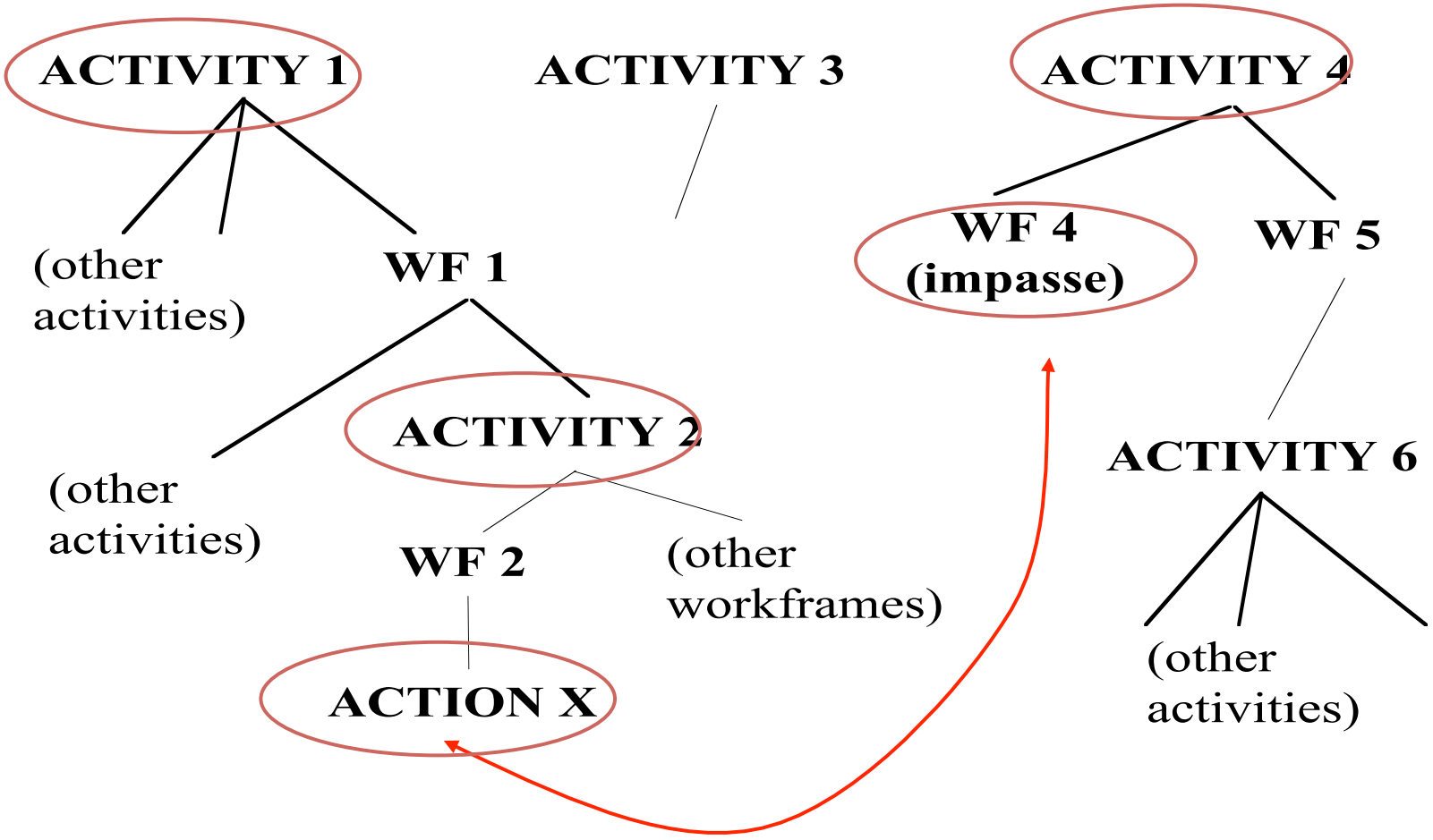
Composite Activities

- Decompose activities into sub-activities and the workframes that can execute them.
- Defines a workframe-activity hierarchy
- Execution is different than traditional rule hierarchies:
  - Subsumption hierarchy
  - While “in” an activity the higher-level activity is still active.

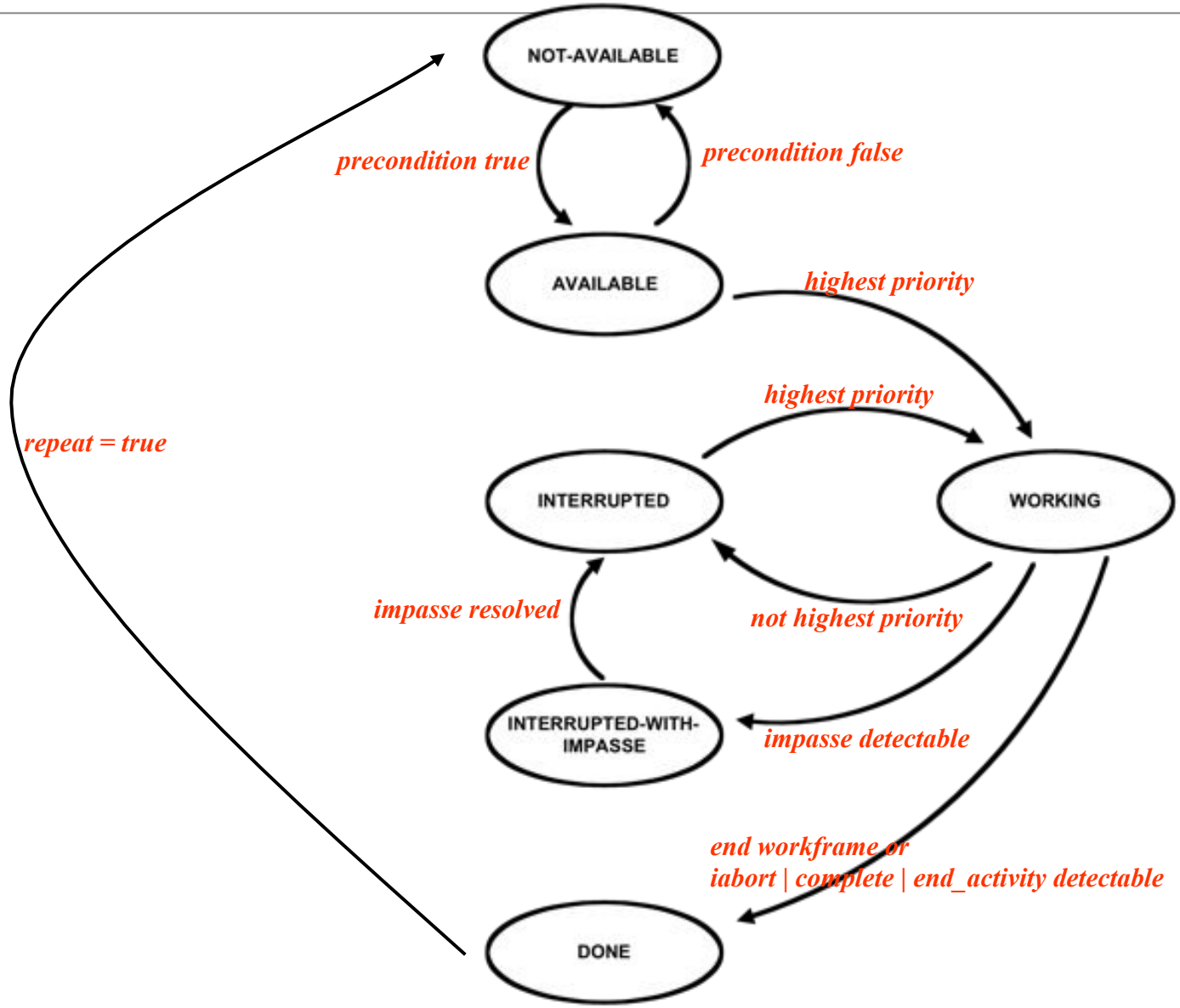
Workframe-Activity Hierarchy

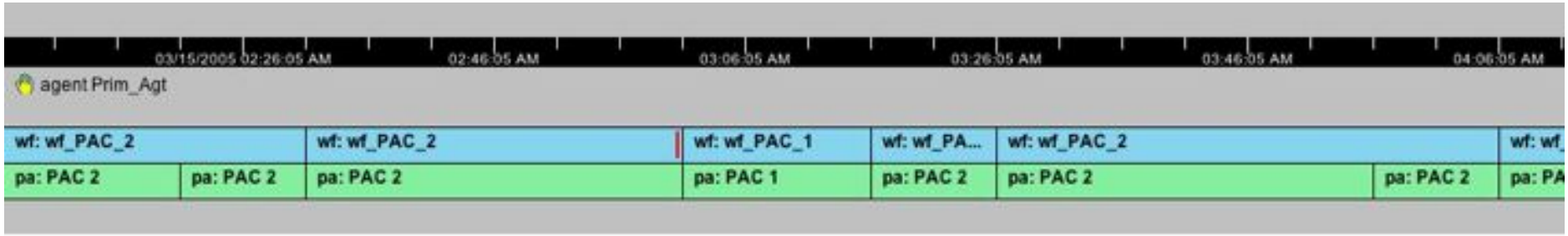


Activity Subsumption



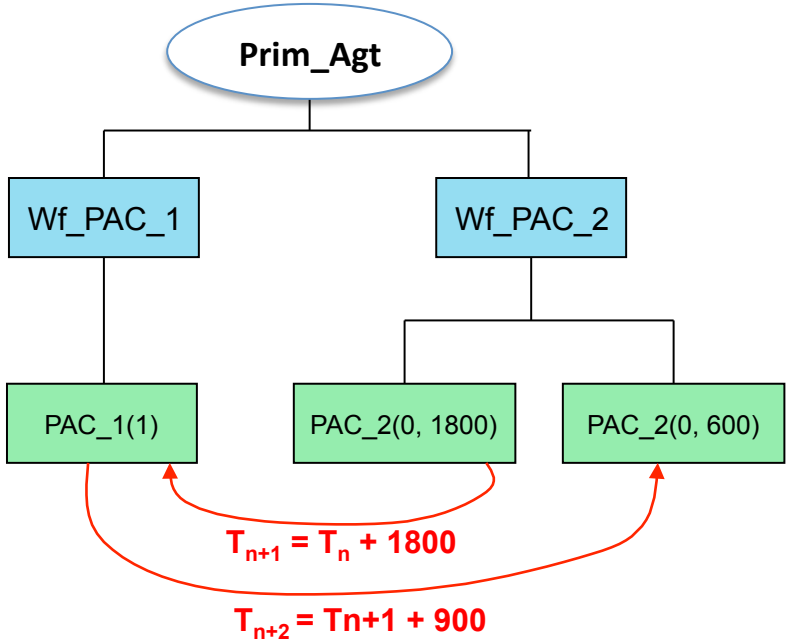
Workframe/Activity States



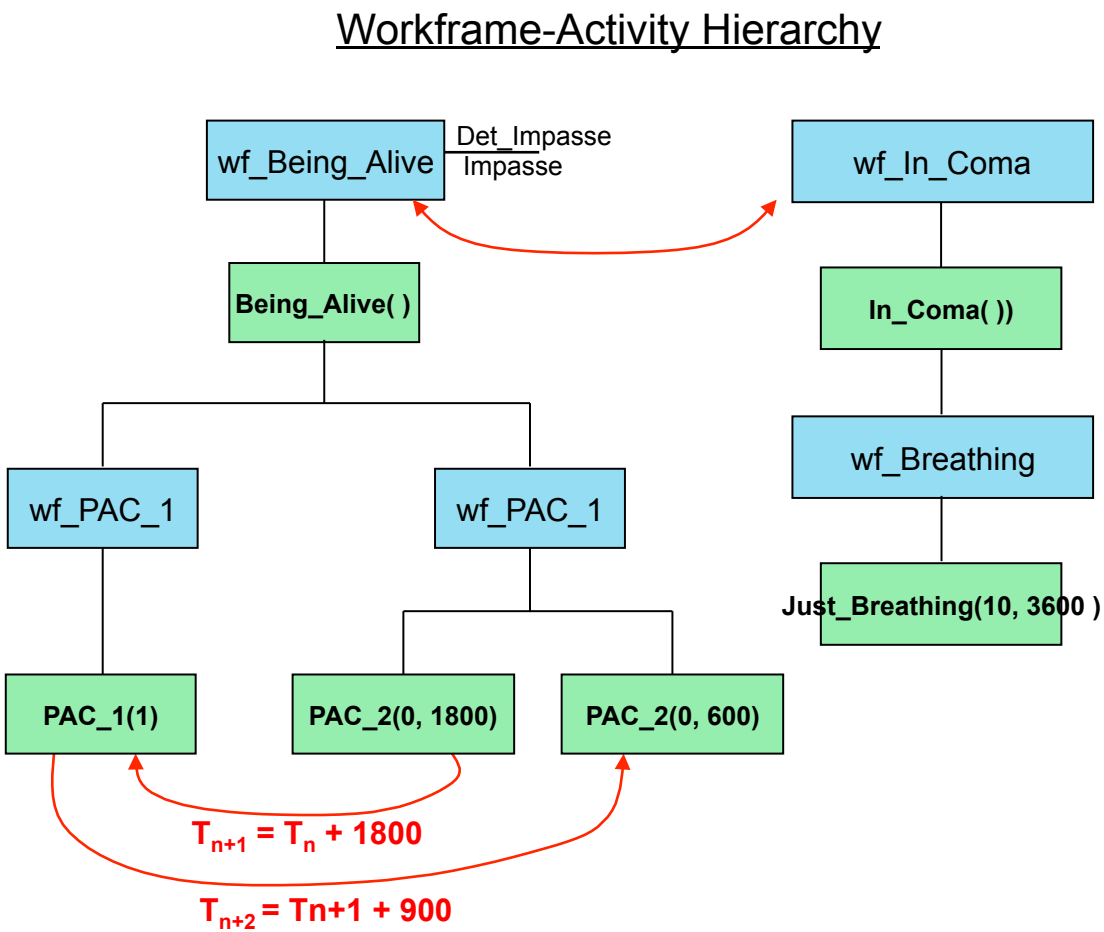



```
        primitive_activity PAC_1(int pri) {
            display: "PAC 1";
            priority: pri;
            max_duration: 900;
        }

        primitive_activity PAC_2(int pri, int dur) {
            display: "PAC 2";
            priority: pri;
            max_duration: dur;
        }

    workframes:
        workframe wf_PAC_1 {
            repeat: true;

            when (knownval(current.execute_PAC_1 = true))
            do {
                PAC_1(1);
                conclude((current.execute_PAC_1 = false));
            }
        }

        workframe wf_PAC_2 {
            repeat: true;

            do {
                PAC_2(0, 1800);
                conclude((current.execute_PAC_1 = true), bc:25);
                PAC_2(0, 600);
            }
        }
}
```

Workframe-Activity Hierarchy

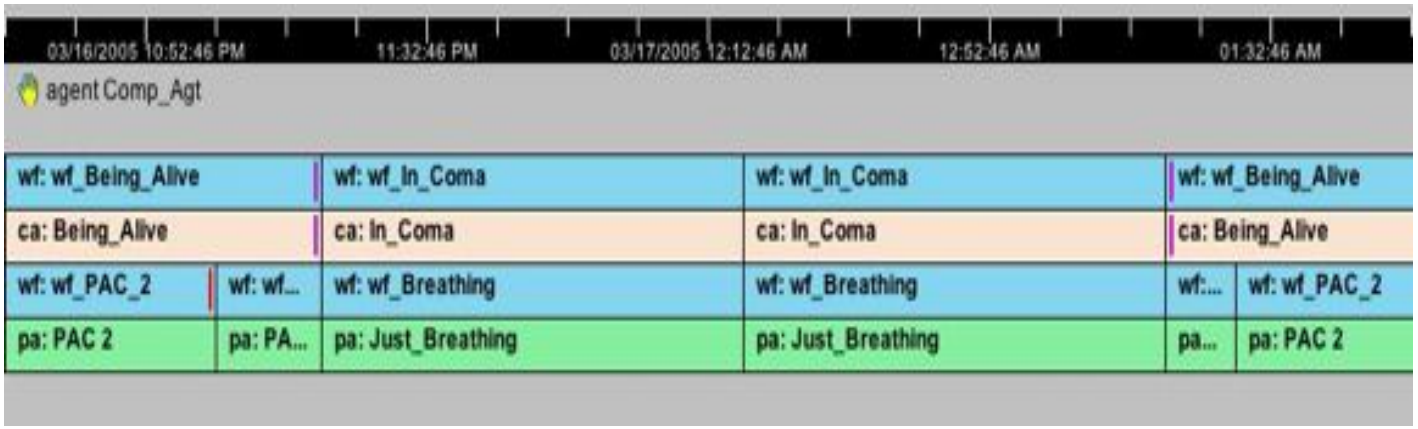


Composite Activities



Composite Activities

```
workframe wf_Being_Alive {
    repeat: true;
    detectables:
        detectable det_Impasse {
            detect((current.headTrauma = true))
            then impasse;
        }

    do {
        Being_Alive();
    }
}

workframe wf_In_Coma {
    repeat: true;

    when(knownval(current.headTrauma = true))
    do {
        In_Coma();
        conclude((current.headTrauma = false), fc:50, bc:50);
        printBelief(current, headTrauma, attribute);
    }
}
```



Agent Communication

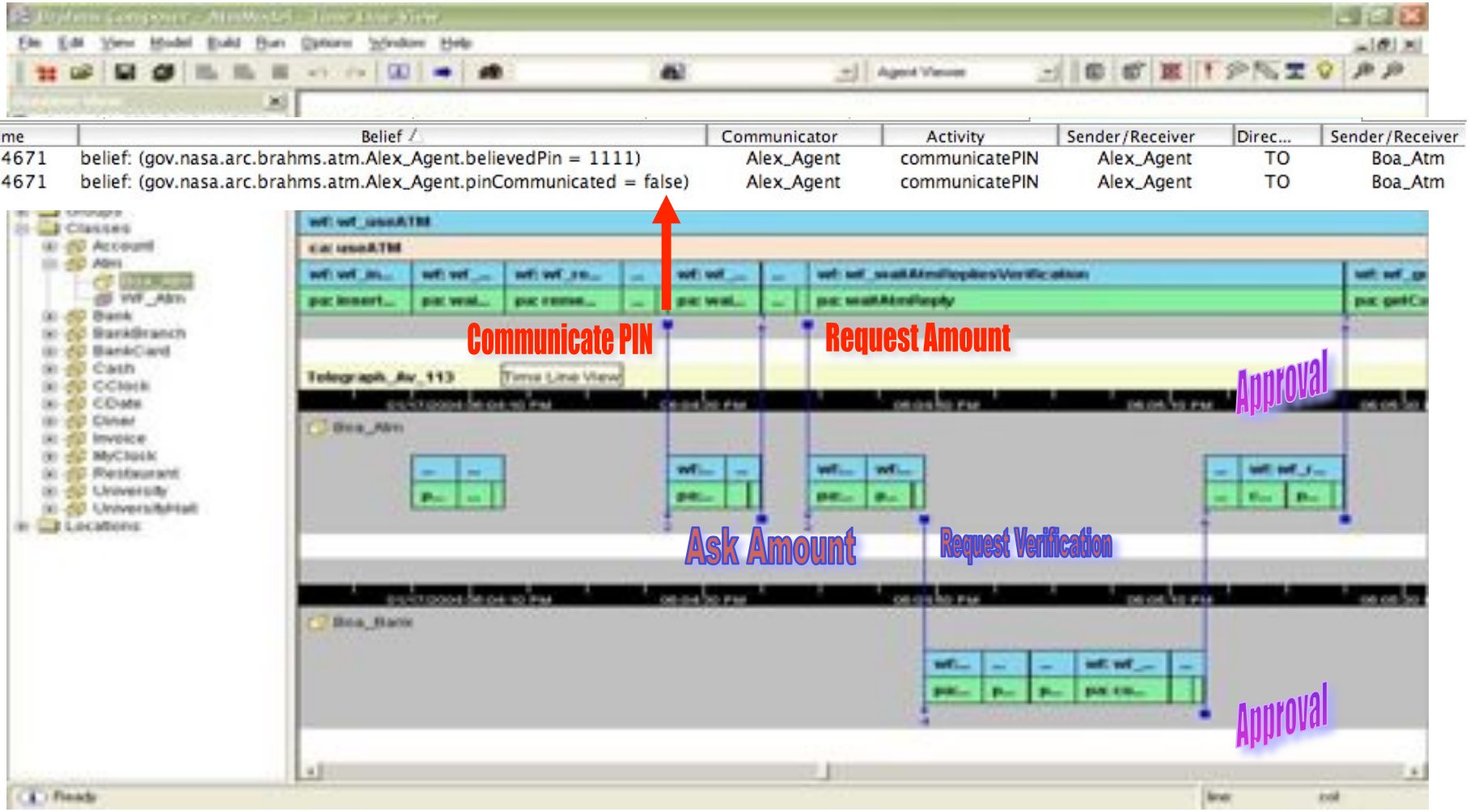
Two Ways of Agent Communication

- Communicating *individual beliefs*
  - simple, but unstructured
  - mostly used in simulation
- Communicating via *speech acts*
  - more complicated, but more structured
  - good for defining standard conversation protocols
  - used in MAS using FIPA standard

Brahms Communications

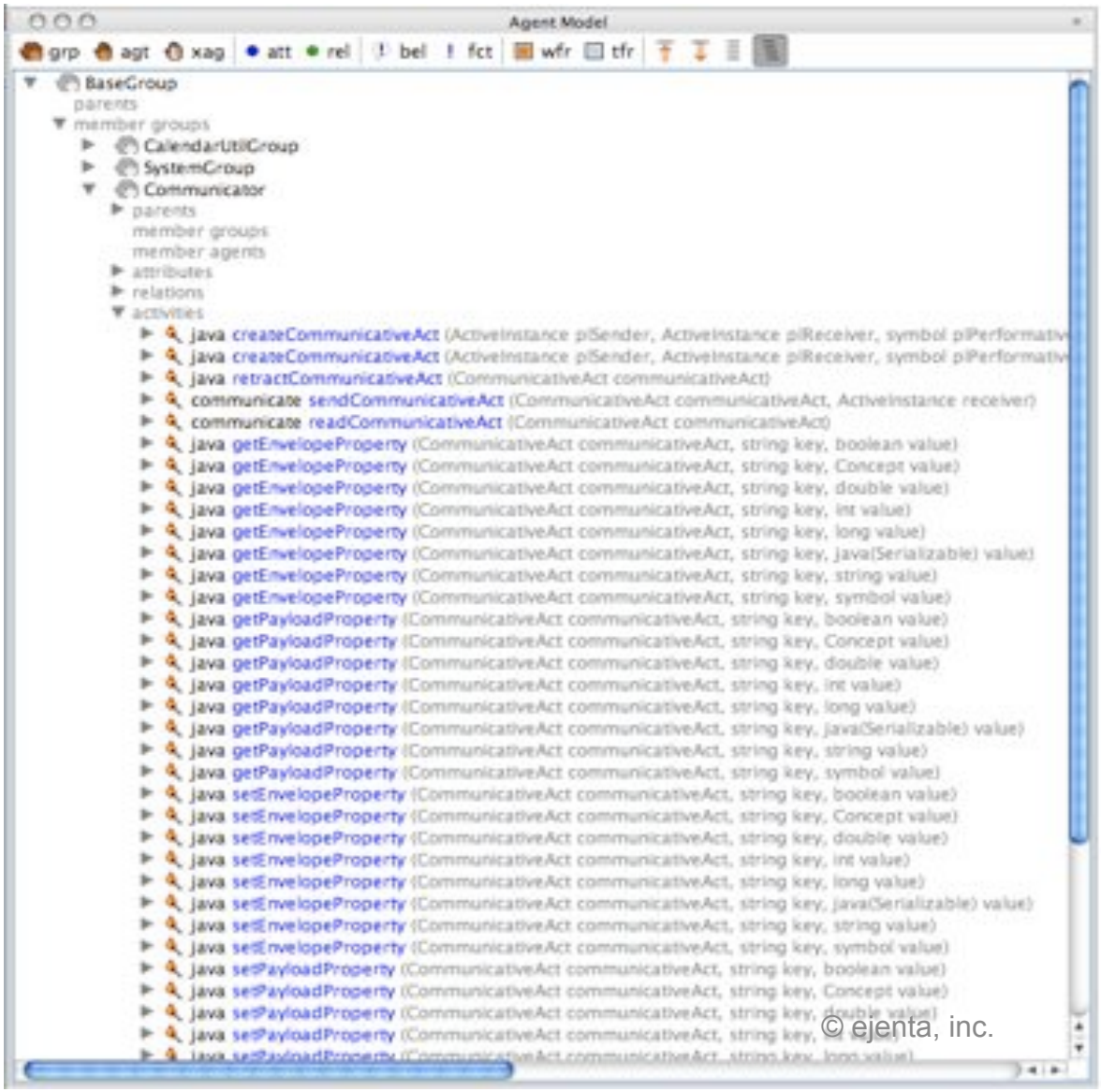
- Activities that transfer beliefs to/from one agent to one or several other agents, or to/from an (information carrier) object. Examples are:
  - Face to face conversations.
  - Reading or writing a document.
  - Data entered into computers.
- An agent/object has to have the belief before it can communicate (i.e. tell) the belief to another agent/object.
- Recipient agent/object will overwrite original beliefs with communicated beliefs.

Alex Communicates with ATM



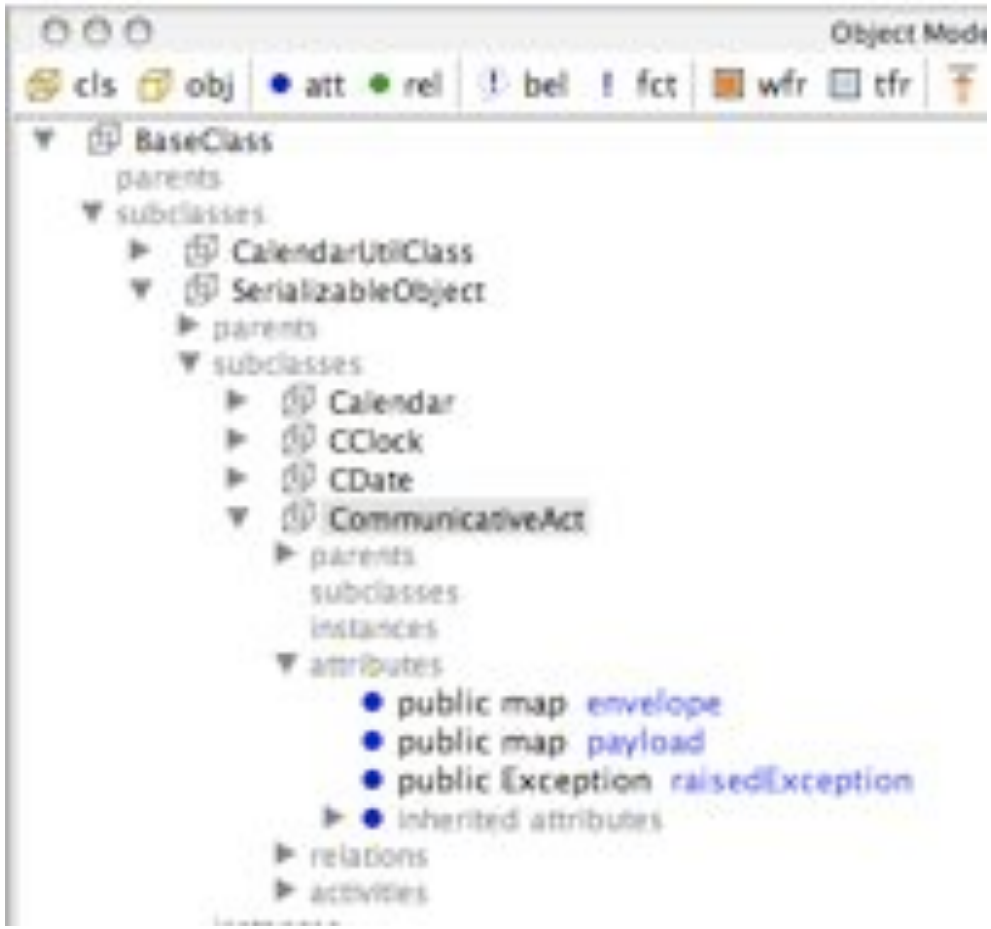
Communication Library

- A **Communicator** is able to communicate with other agents through **communicative acts**
- The Communicator specifies a set of activities that can be used by communicators to **create, read, manipulate, retract**, and **send** communicative acts
- Defines class **CommunicativeAct**



CommunicativeAct

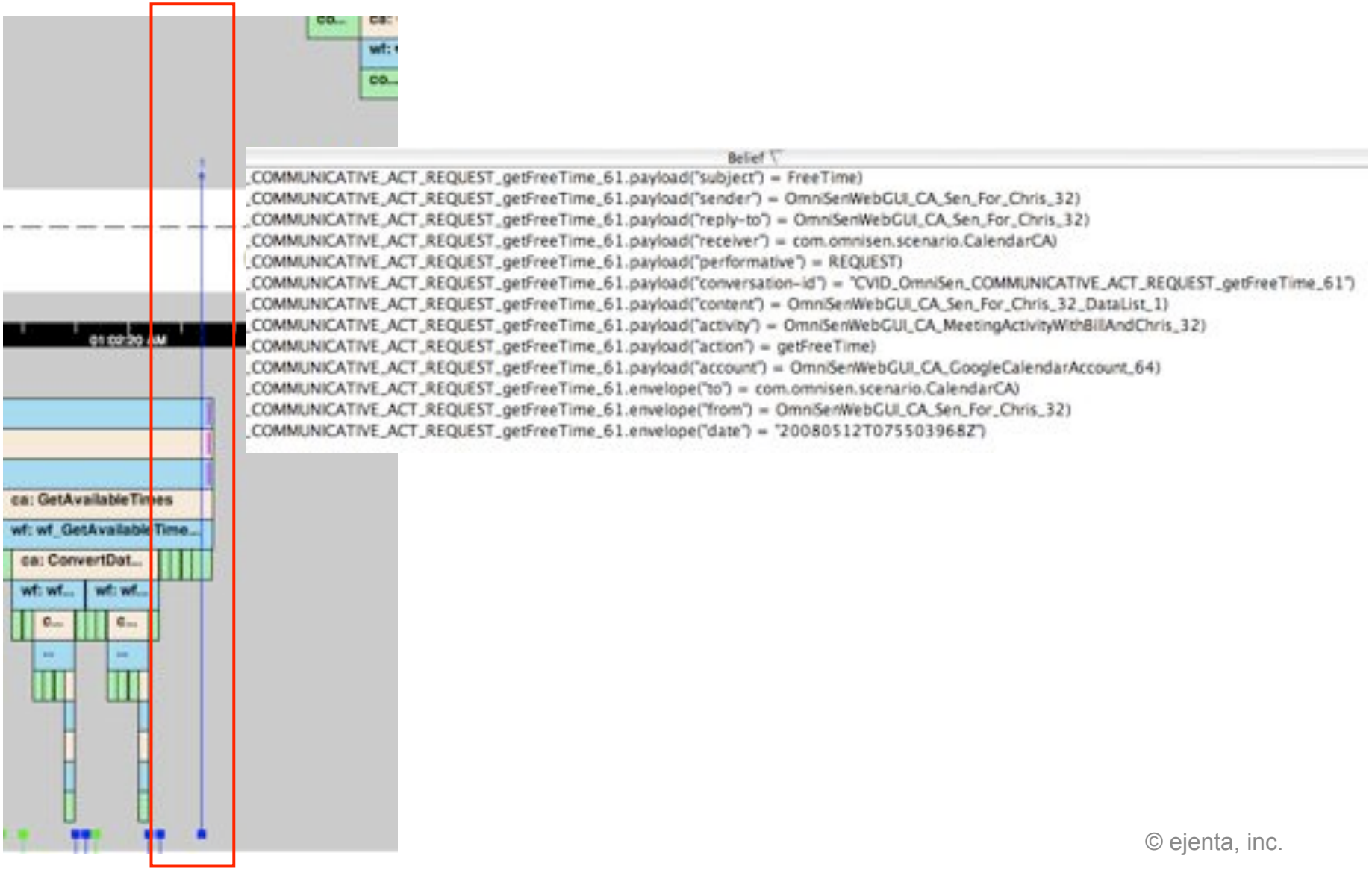
- The **CommunicativeAct** models a communication event between two actors
- Defines a message that is based on the Communicative Act standard defined by **FIPA** (Foundation of Intelligent Physical Agents)
- Specifies an **Envelope** with the **address information** (from, to, date, ...) and transport hints
- Specifies a **Payload** for the **message content** and content properties
- Envelope and payload are maps
- http://www.fipa.org/specs/fipa00037/index.html



Example Sending ComAct

```
workframe wf_ConfirmGetAvailableTime {
    variables:
        forone(ActiveInstance) sender;
        forone(string) convid;
        forone(CommunicativeAct) reply;
        forone(Activity) act;
    when ( (comact.payload("performative") = REQUEST) and
            (comact.payload("action") = getAvailableTime) and
            (comact.payload("sender") = sender) and
            (comact.payload("purpose") = act) and
            (comact.payload("conversation-id") = convid))
    do {
        createCommunicativeAct(current, sender, INFORM, convid, getAvailableTime, classtype, current, reply);
        setPayloadProperty(reply, "content", payloadobj1);
        setPayloadProperty(reply, "purpose", act);
        sendCommunicativeAct(reply, sender);
    }//do
}//wf_ConfirmGetAvailableTime
```

Receiving ComAct



Brahms Geography

Brahms Objects

- Objects are *data* and *real world artifacts*.
- Objects could be *inanimate objects or computational objects*.
- Why objects and agents?
  - Most agent languages only represent agents.
  - Brahms incorporates our theory of work practice, and from a social and practice perspective people do differentiate between *intentional agents* (i.e. humans) and *artifacts*
- Examples:
  - Fax machines
  - Database
  - Instruments
  - Rock samples
  - Photo Cameras
  - Space Suits
  - ATM

Brahms Geography

- Agents and objects can be located (***initial location***).
- Agents and objects can move to/from locations (***move activity)***
- Agents know where they are and notice others:
  - When agents come into a location, the Brahms engine automatically gives the agent a belief about its new location (same as fact), and
  - … gives the agent a location belief for all other agents and objects currently in that location.
  - When an agent/object leaves a location, the location fact and beliefs are retracted (from all agents that are in that location the moment the agent/object leaves.)
- Agents and objects can carry (containment relation) other agent/objects
  - Contained objects are NOT noticed until they are *put* into the area.

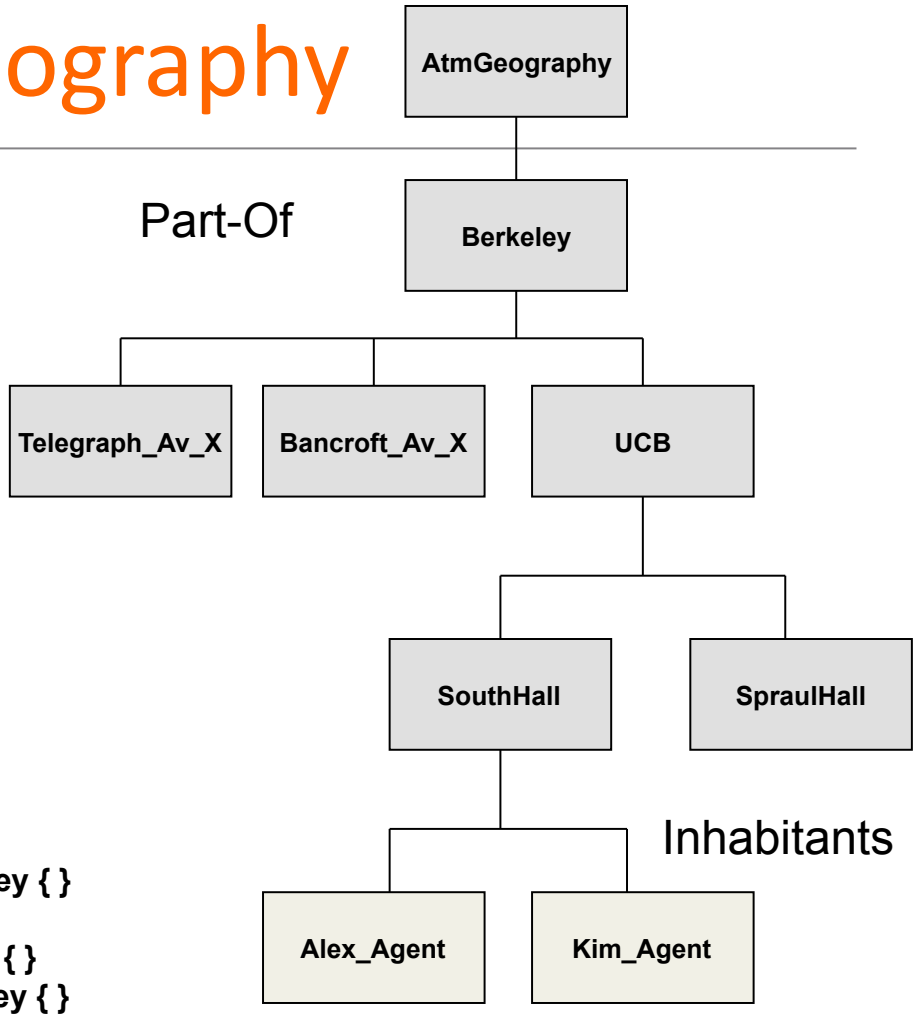
Geography Model

- Geography Model is separate from Agent and Object Model
- Conceptual Geography Model
  - Areas are a special type of geography object
  - Areas have attributes and relations
  - Areas can define initial facts
  - Areas are instances of an Area Definition
  - Area Definition is a special geography class type
- Facts about areas represent state of a location
  - E.g. temperature
- Agent location attribute is inherited from Brahms BaseGroup. For objects from BaseClass

Topological Maps

- Areas do **not** have scale or dimensions
- Areas are **not** necessarily a grid
- Areas can have sub-areas,
- Sub-areas can have sub-sub-areas, etc

Tutorial Scenario Geography




```
package projects.atm;

areadef University extends BaseAreaDef { }
areadef UniversityHall extends Building { }
areadef BankBranch extends Building { }
areadef Restaurant extends Building { }

// ATM World
area AtmGeography instanceof World { }

// Berkeley
area Berkeley instanceof City partof AtmGeography { }

// inside Berkeley
area UCB instanceof University partof Berkeley { }
area SouthHall instanceof UniversityHall partof UCB { }
area Telegraph_Av_113 instanceof BankBranch partof Berkeley { }
area SpraulHall instanceof UniversityHall partof UCB { }
area Bancroft_Av_77 instanceof BankBranch partof Berkeley { }
area Telegraph_Av_2405 instanceof Restaurant partof Berkeley { }
area Telegraph_Av_2134 instanceof Restaurant partof Berkeley { }
```

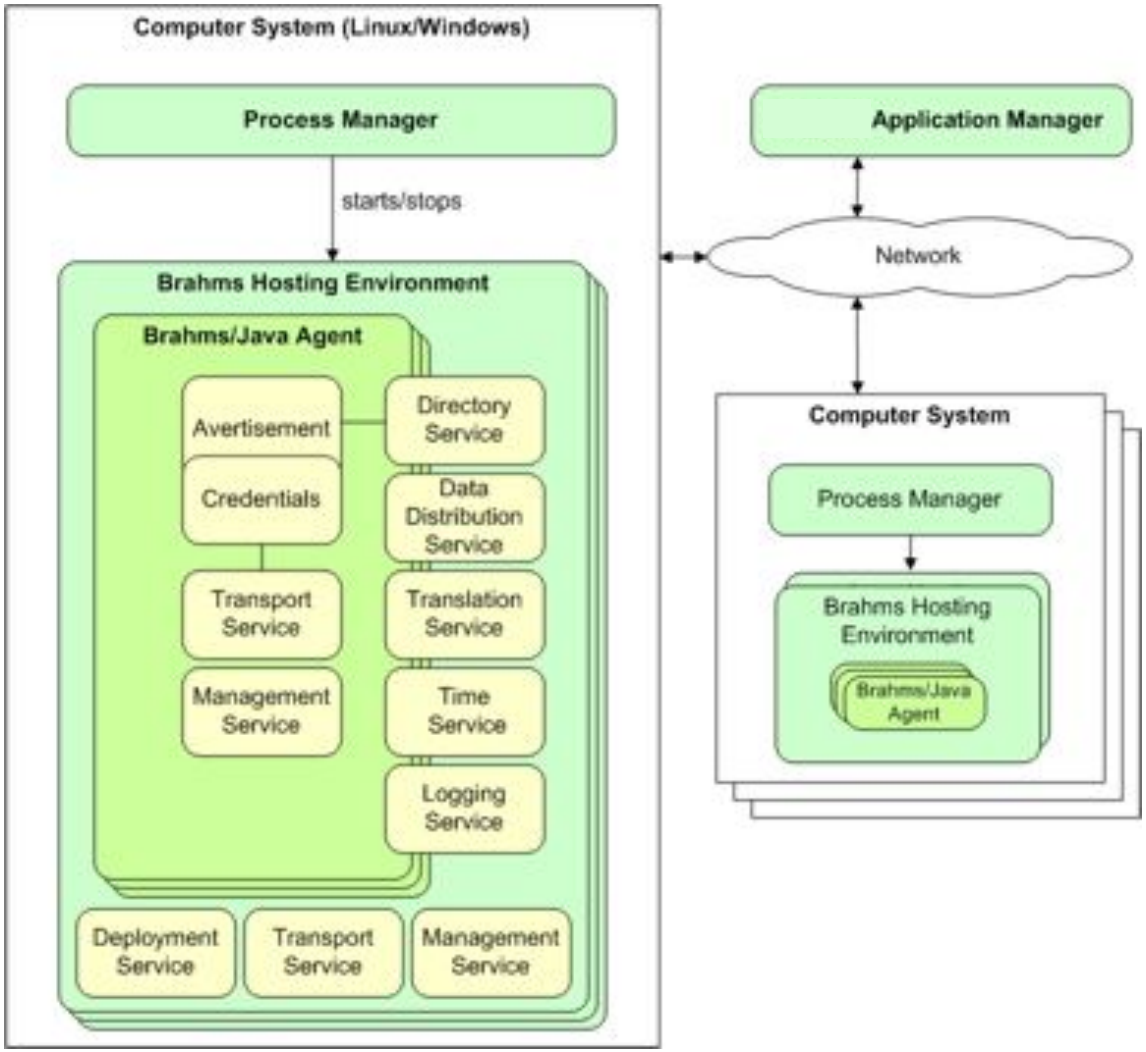
```
agent Kim_Agent memberof Student {
  location: SouthHall;
```

```
agent Alex_Agent memberof Student {
  location: SouthHall;
```

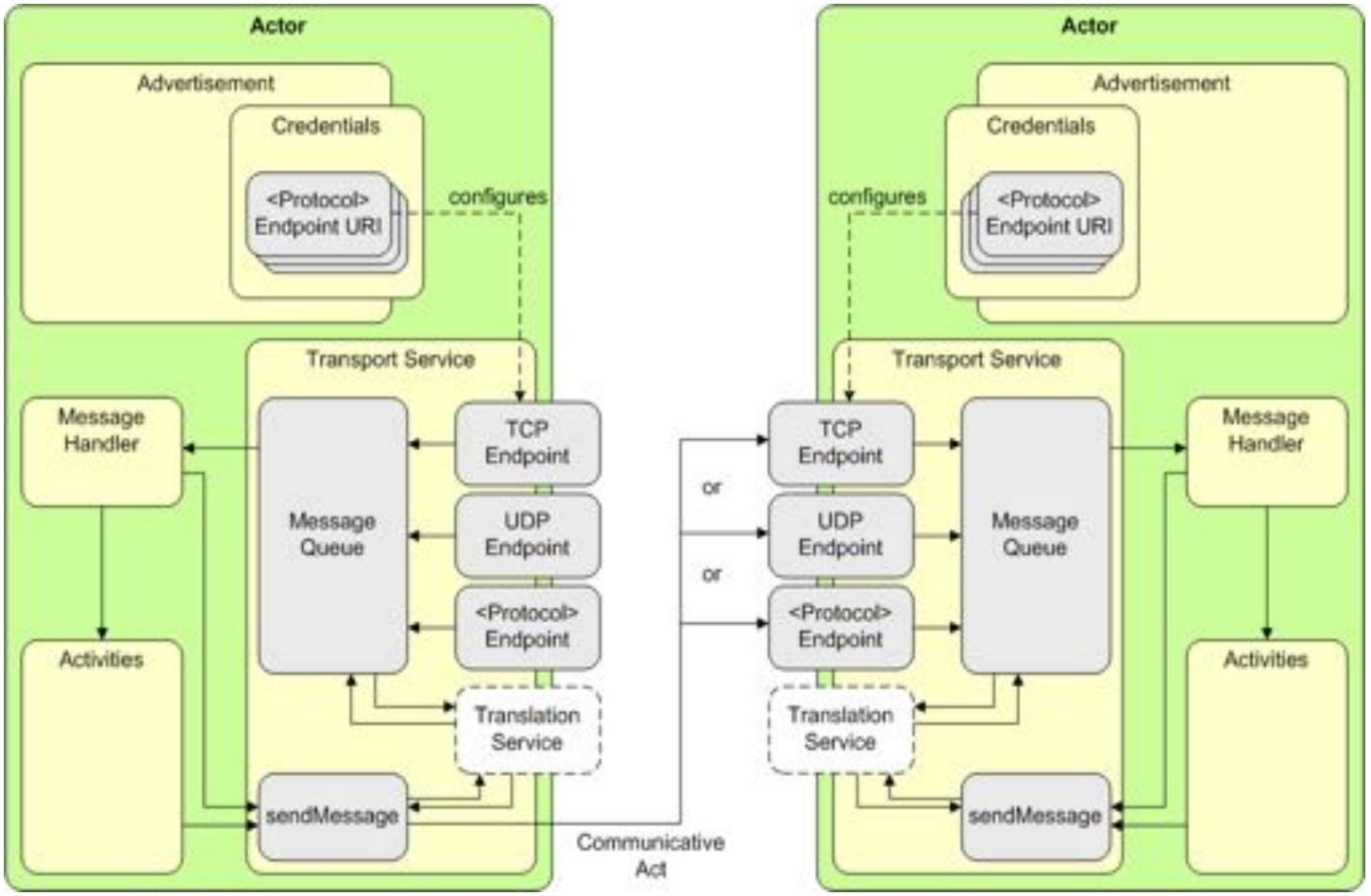
Moving

- Agents and objects can move
  - Use move(to_location) activity in a workframe
  - Can specify duration in clock-ticks
  - Default zero duration, unless
- Define a Path object between two areas
  - Defines duration to move from area1 to area2
  - Bi-directional path
- Engine retracts and creates location facts and beliefs
  - Can specify (sub-)area arrival and departure detection
- Engine calculates shortest path between areas
- Contained objects and agents move with the agent

Brahms Hosting Environment



Distributed MAS Communication Framework



Brahms License

- Brahms Academic License
  - Free use for academic/educational purposes
- Brahms Commercial Trial License
  - Used for any commercial application of Brahms
  - Free trial license

Contact Ejenta

- Ejenta is in San Francisco, CA
- www.ejenta.com
- Getting Brahms:
  - Soon via website
  - Now: send email to brahms@ejenta.com
  - Provide:
    - Name
    - Institution
    - Use of Brahms: academic or commercial