NOWHERE

a Knowledge-Level API
Agent Programming Infrastructure

- Targeted for Personal (Web) agents
- Supports Knowledge Level (KL) Agents

06/09/06 Davide Guidi - dguidi@cs.unibo.it

KL Agent Properties

- The programmer does not have to manage physical addresses of agents explicitly
- The programmer does not have to handle communication faults explicitly
- Communication is Starvation-proof
- Communication is Deadlock-proof

06/09/06 Davide Guidi - dguidi@cs.unibo.it

Communication Language

NOWHERE implements FT-ACL:

- a *Fault-Tolerant Agent Communication Language* for programming *Knowledge Level Agents*
- Support the *Core Language* (used for reactive agents) and an *Extended Language* (used for proactive agents)

Message datatype

In NOWHERE agents exchange messages, that store data about:

- the message performative (inform, askOne, tell..)
- the sender id and the receiver id
- the requested service
- the content of the message
- other data needed by the platform

In a language-independent way

Core ACL Primitives

Standard conversation primitives:

- inform(message)
- informACK(message, onAnswer, onFail)
- askOne(message, onAnswer, onFail)
- tell(message)

Core ACL Primitives /2

One-to-many conversation primitives (simple form):

- askEverybody(message, onAnswer, onFail)

Core ACL Primitives /3

Support for anonymous interaction:

- handler(serviceURI, function)
- register(serviceURI)
- all-answer()

Core ACL Example

AgentB: provides temperature to other agents (Java)

```
handler("get_temperature", "provideTemp");
register("get_temperature");

public void provideTemp(Message m) {
        Message msg = m.getReplyMessage();
        msg.add("aboutThisLocation”);
        msg.add(temperature);
        tell(msg);
    }
```

AgentA asks AgentB for temperature and prints it on the screen (Python)

```
msg = message('get_temperature')
msg.setReceiver(agentB)
self.askOne(msg, printTemp, fail)

def printTemp(Message m):
    print 'Temperature in ', m.getElement(0),
    ' is: ', m.getElement(1)

def fail():
    print 'No agents found!'
```

Extended ACL (Work In Progress)

- Introduce blocking primitives for performatives like informACK, askOne, askEverybody
- Core ACL properties doesn't hold here (for blocking primitives issues). Must use countdown timer to ensure that the Core ACL properties holds.

Extended ACL /2

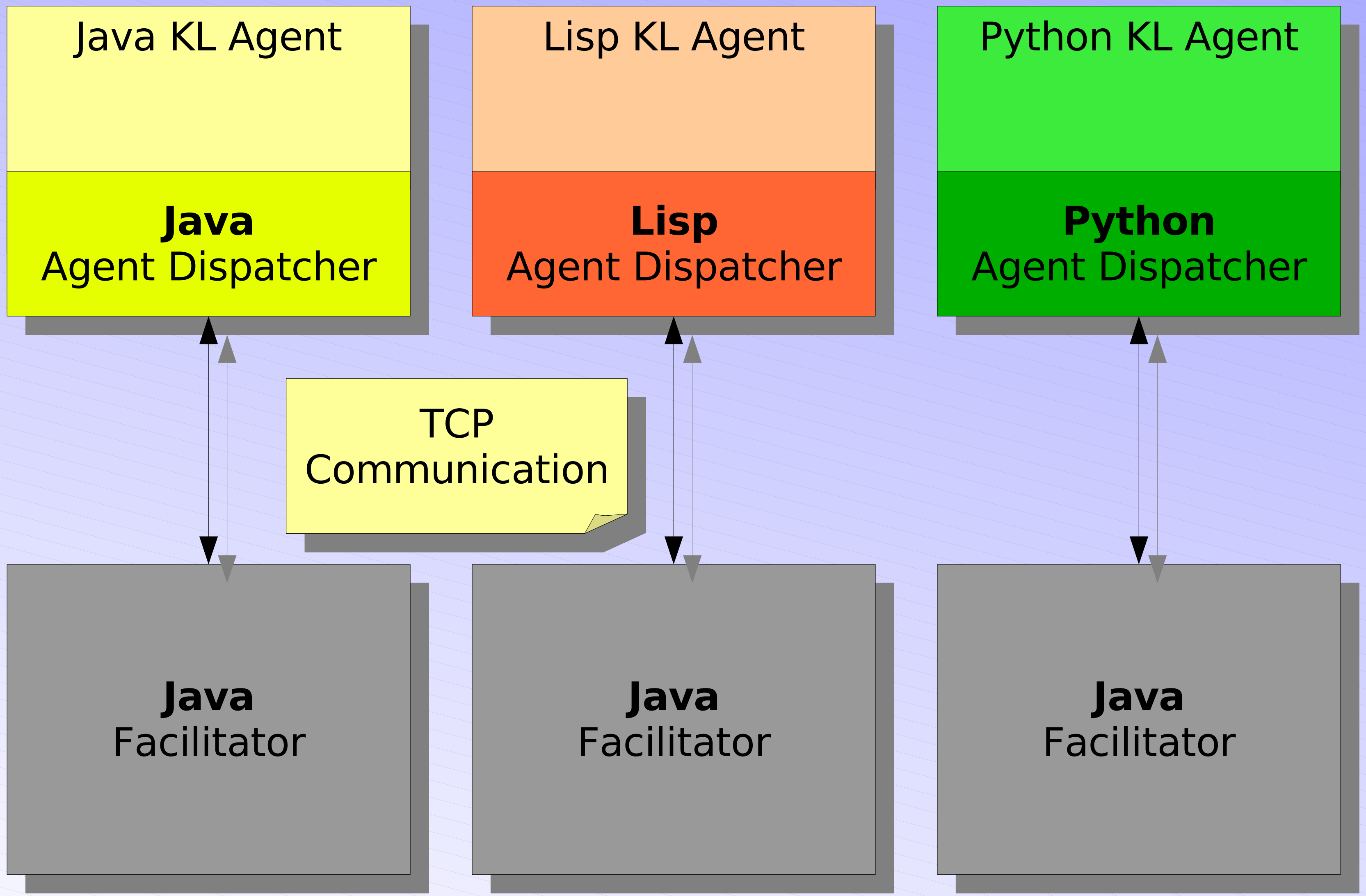
- The key idea is to have something like Java's try & catch

```
msg = message('get_temperature')
m.setReceiver(agentB)
try {
    m = self.askOne(msg, printTemp, fail)
    print 'Temperature in ', m.getElement(0),
    ' is: ', m.Element(1)
}
onError {
    print 'No agents found!'
}
```

NOWHERE Components

- Agent Dispatcher – a language dependent stub that can be extended to create *Knowledge Level* (KL) agents
- Facilitator – a Java object that provides network abstraction to Agent Dispatcher

NOWHERE Components



Agent Dispatcher

Applied AI Lab

Related to a specific programming language, the Dispatcher:

- Provides ACL primitives (not the behaviour!)
- Provides a Message (class) object
- Provides a “dispatcher” function to manage incoming messages

Anonymous Interaction Mechanism

```
# Python Source code: anonymous interaction mechanism
# Behaviour: ask every agent on the network for the
  service "service_1"
class Server(AgentCL1):

  def dispatcher(self, m):
    # This function is called whenever this agent
    # receives an incoming message m

  # Agent's code
  m = message('service_1')
  m.addElement('Data about the service goes here')
  self.askEverybody(m, self.onAnswer, self.onFail)

  def onAnswer(self, m):
    # Agent received the answer

  def onFail(self, m):
    # Error occurred
```

Anonymous Interaction Mechanism/2

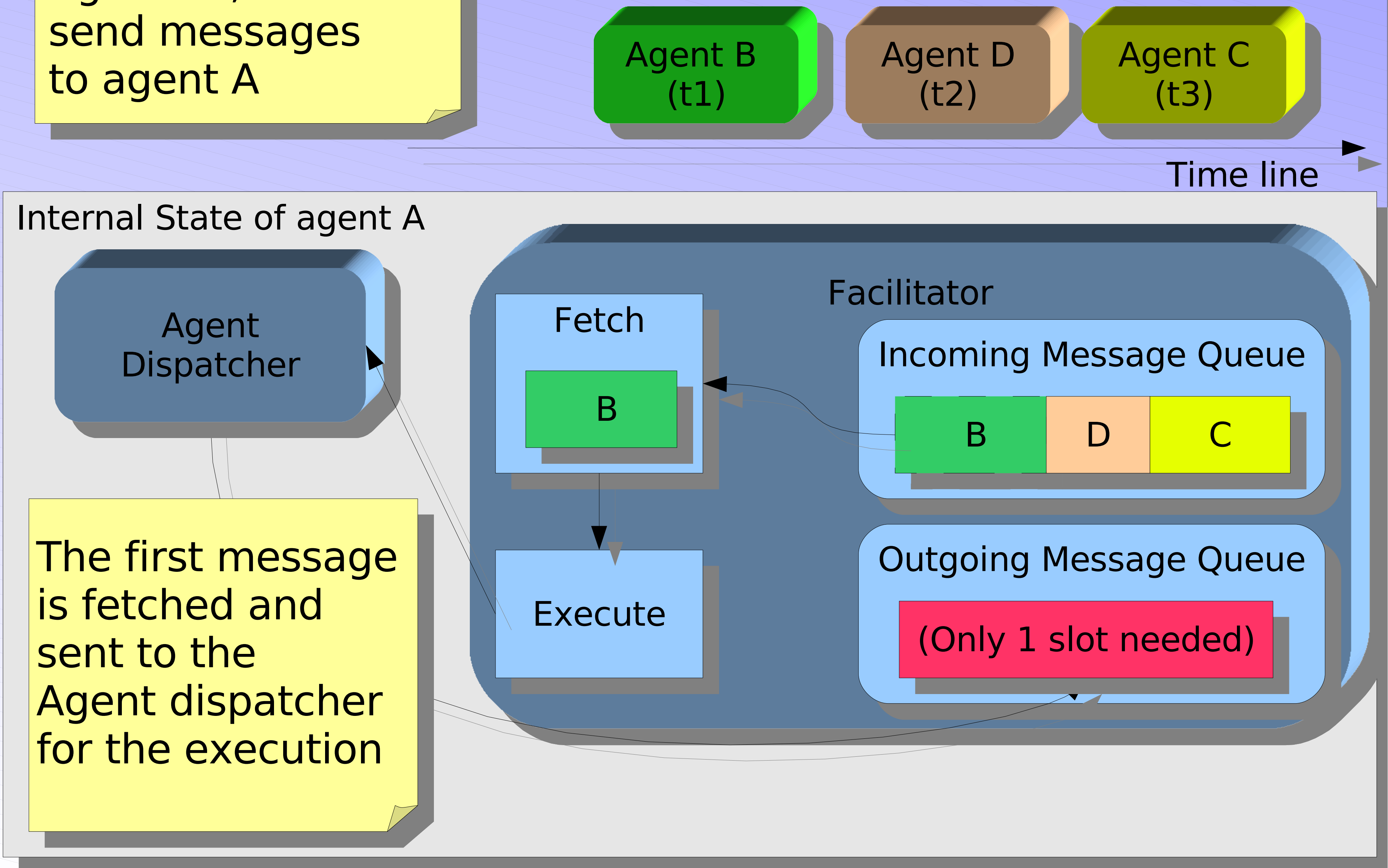
```
# Python Source code: anonymous interaction mechanism
# Behaviour: provide the service 'service_1' to agents
class Client(AgentCL1):

  def dispatcher(self, m):
    # This function is called whenever this agent
    # receives an incoming message m

  # Agent's code

  self.handler('service_1', self.service1)
  self.register('service_1')

  def service1(self, m):
    # This routine implements service1
    replyMsg = m.getReplyMessage()
    m.add('Solution goes here')
    self.tell(m)
```

Notes

- Agents publish services to the local Facilitator and provide a function execute them
- The Facilitator will then execute the proper service, based upon received messages from other agents

Message Flow

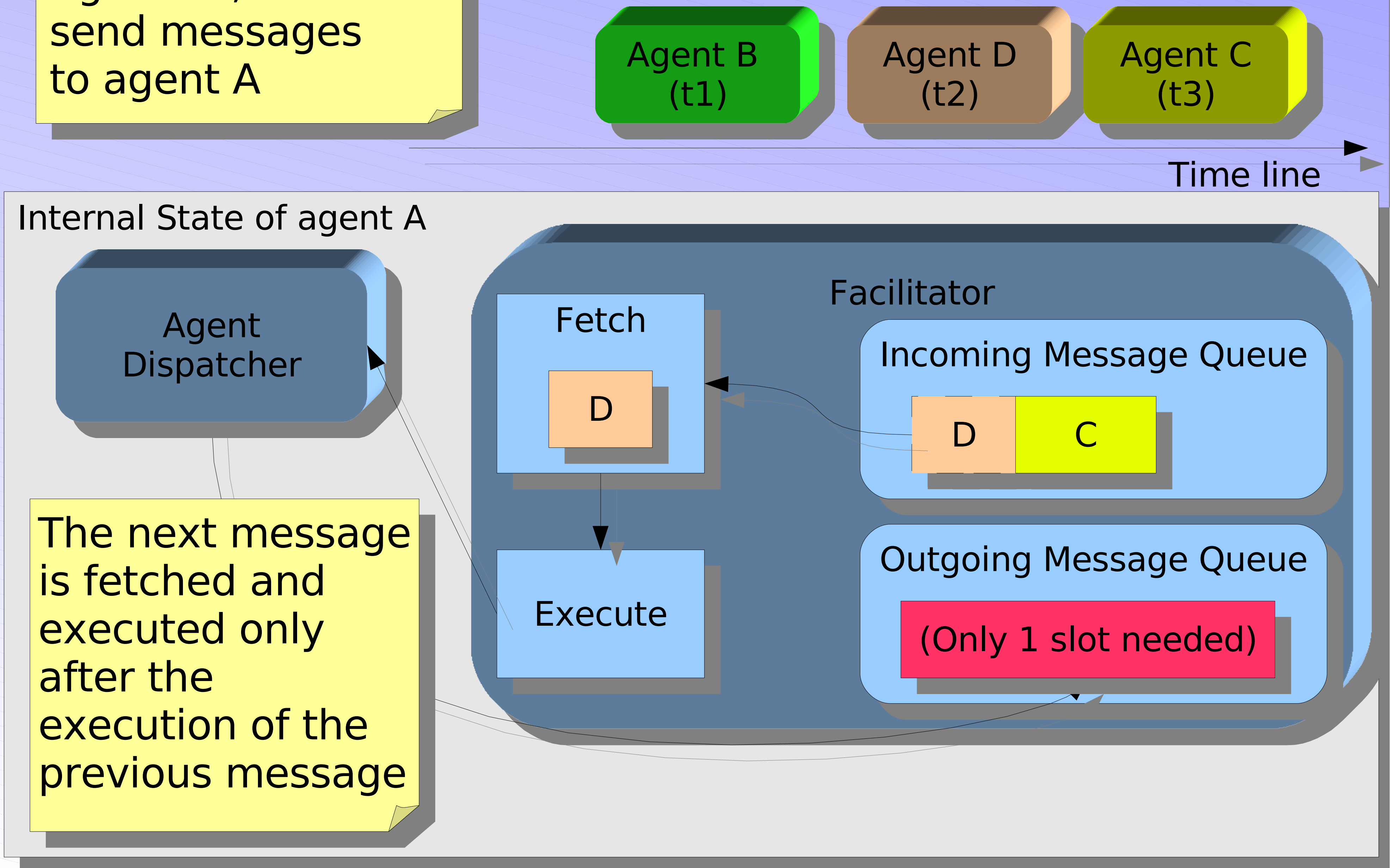
Agents B, C and D send messages to agent A



The first message is fetched and sent to the Agent dispatcher for the execution

Message Flow/2

Agents B, C and D send messages to agent A



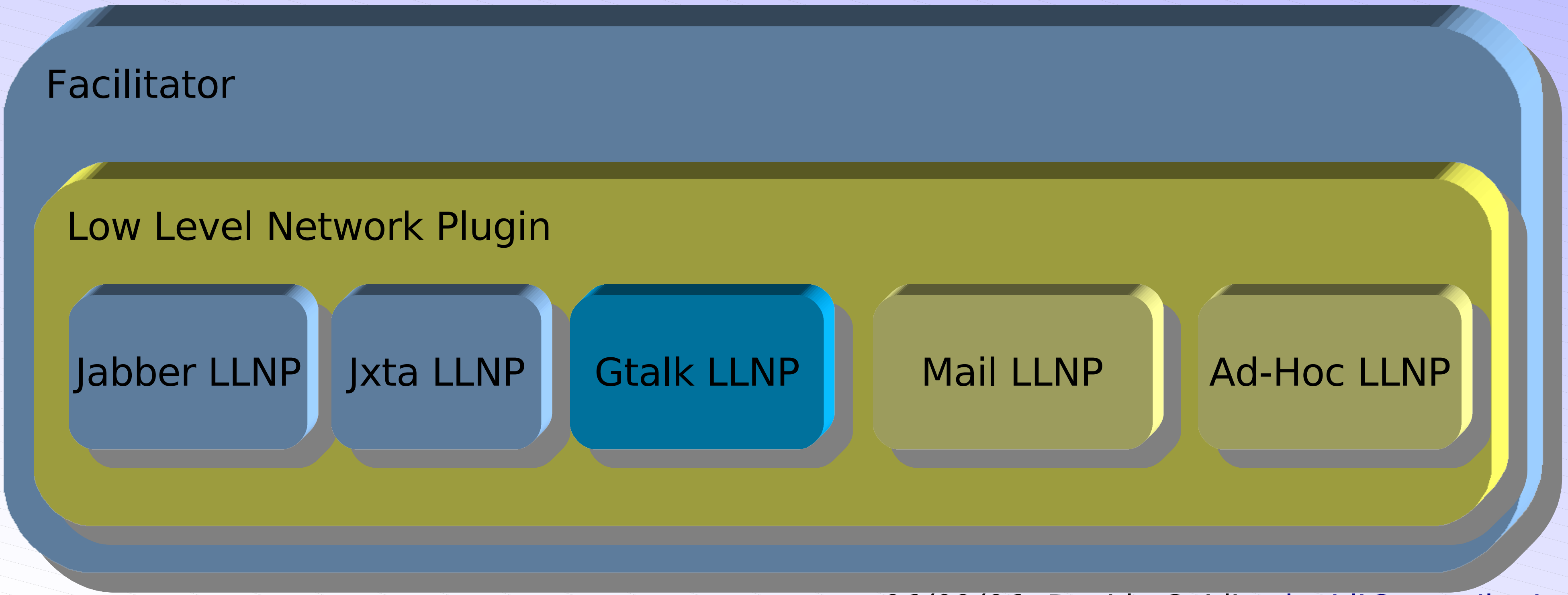
The next message is fetched and executed only after the execution of the previous message

Agent Facilitator

- Implements the logic that controls messages' flow, with an integrated failure detector component
- Supports communication both sending dispatcher messages and by managing dispatcher's local services
- Uses a pluggable low level network to send messages (Jxta, Jabber...)

Low Level Network Protocol

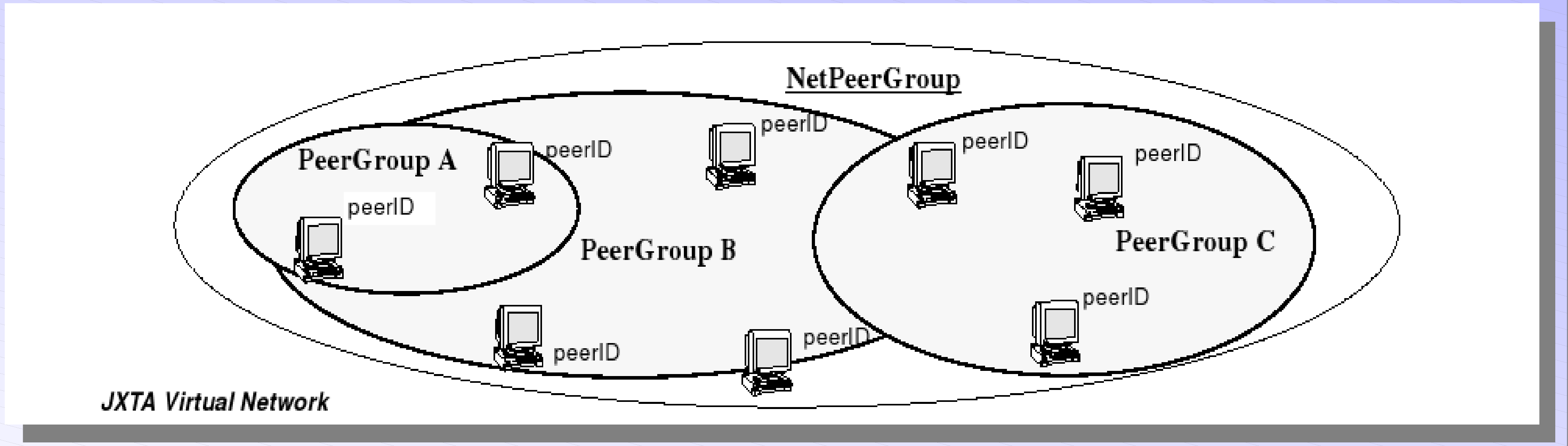
- **sendMessage**(Message m);
- **broadcastMessage**(Message m);
- **join**(String group);
- **leave**(String group);



Developing scalable networks

Applied AI Lab

Using the PeerGroup concept: Communities of agents interested in a common topic.



06/09/06 Davide Guidi - dguidi@cs.unibo.it

Managing Groups

- Groups are identified by URIs
- If agent a register a capability C, then it will be part of the group C

```
# Python Source code: anonymous interaction mechanism
...
  self.register('service_1')

...
```

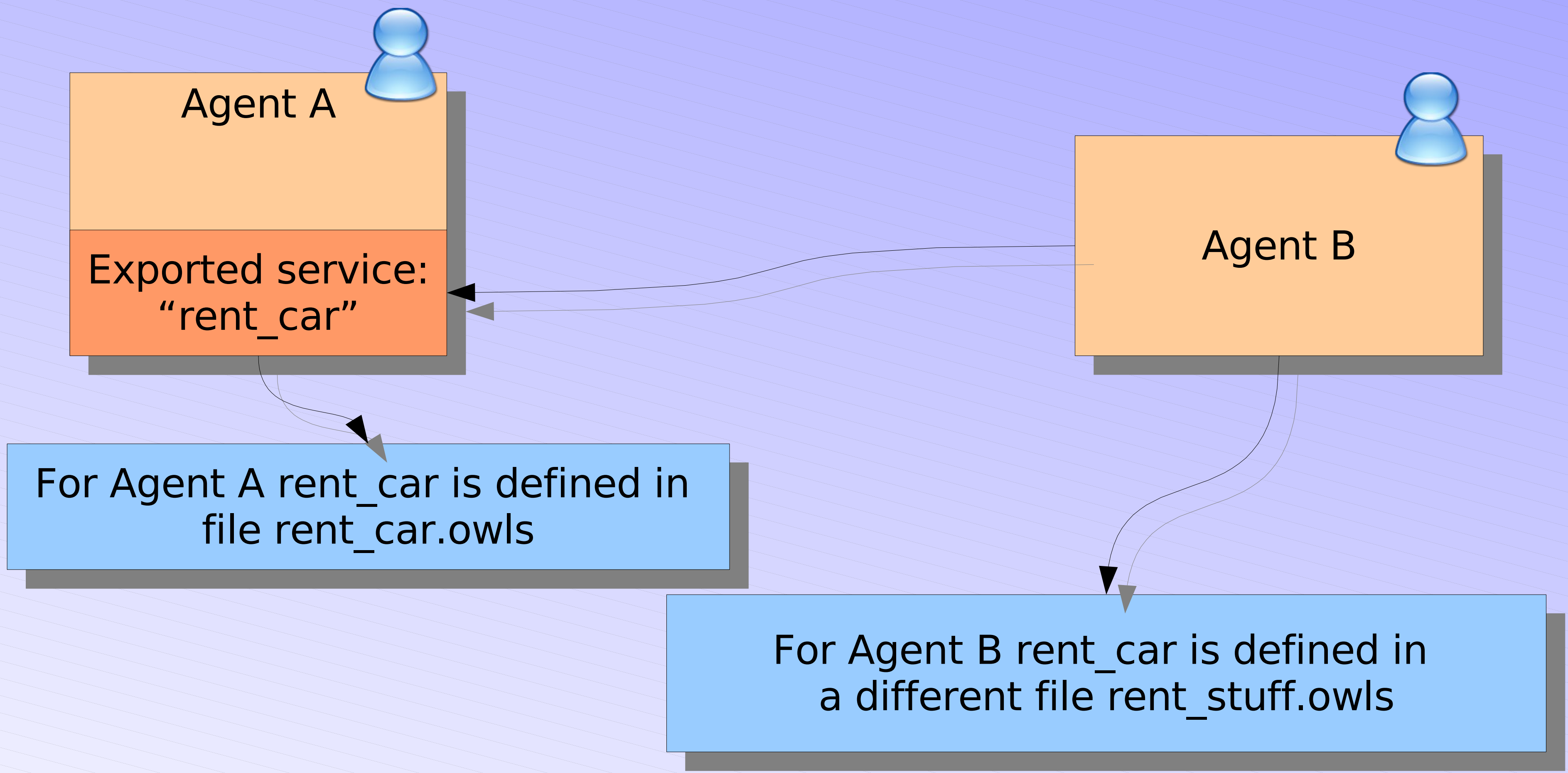
Become member
of Group 'service_1'

Problem: agreement on terms

- Agent B asks Agent A for service "ping" and there is NO common definition for it.
- Both agents must have the same hard-coded internal representation of the service

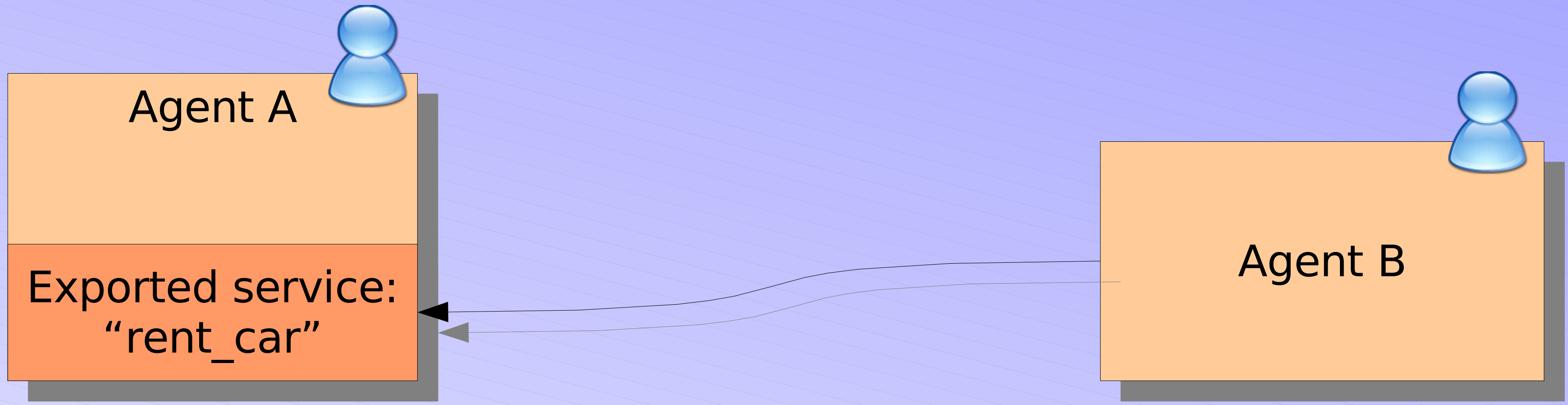
06/09/06 Davide Guidi - dguidi@cs.unibo.it

Solution: ontology-driven communication



- Agent B asks Agent A for service “rent_car”. If the two ontology match at a certain level, than then there is an agreement on this term

Ontology-driven example



- Agent B asks agent A for its semantic description about the service "rent_car" (or for all its services)
- Agent B obtains the description, for example OWL-S description
- Agent B is then able to match the service with its description and eventually use the service provided by Agent A

Web and Semantic Web compatibility

Web compatibility is achieved in 2 ways (Work in progress):

- Agent's capabilities can be exported as Web Services
- Web services can be invoked with ACL primitives askOne or askEverybody

Semantic Web capability is achieved using ontology-driven communication

Summary of key features

- Provides language, operating system and network abstraction
- Fault-Tolerant ACL
- Anonymous interaction mechanism
- (Semantic) Web compatibility
- Provides support for scalable networks

Future

- New language bindings
- Control Panel to control agents running on a single platform
- Uniform, cross-platform GUI for agents
- General improvements

Building applications for user communities