Multi-agent systems

Andrej Lúčny

KAI FMFI UK

lucny@fmph.uniba.sk

http://www.agentspace.org/mas

teaser:

platform: **Java**

Lecturer

RNDr. Andrej Lúčny, PhD.

- Dealing with real-time AI
- Co-founder of world-wide operating company dealing with hardware & software of monitoring systems
- Developer in its daughter company (computer vision)
- Co-founder of civil society robotika.sk
- Former judge on ACM Scholastic Programming Contest
- Judge on mobile robot contest ISTROBOT

www.agentspace.org/andy

Mission

- Learn technologies of artificial intelligence methods integration via multi-agent systems (object serialization, process sychronization, inter-thread, inter-process and network communication, operation in real time, knowledge manipulation)
- Learn to implement multi-agent framework
- Learn to use existing multi-agent frameworks
- Learn more about development of the complex artificial intelligent systems
- Try to use particular artificial intelligence methods (virtual reality, phase correlation, HOG detector, Hough transform, cascade regressor, deep neural network) - as a black-box

Inquiry

Have you written a program that successfully used:

- Java language
- anonymous class (e.g. TimeredTask)
- Object derived from Thread
- Serializable object
- Socket (network communication)
- Any middle-ware
- A multi-agent framework

?

Ranking

App. max. 44 point per term
(app. 12 weeks)
every week:
1 point for presence
1 point for the basic task
1 point for the best test
1 bod for homework

+ 1 possible point for the advanced task or any outstanding result

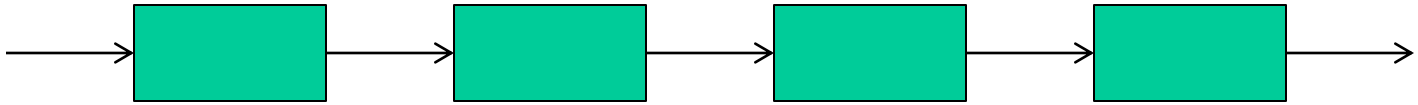
Exam: 40 points per test

61-80 A
56-60 B
51-55 C
46-50 D
41-45 E
0-40 Fx

project is also possible in case of a long illness or foreign study

- We know many interesting particular methods of AI, e.g. we can recognize face on image.
- How to compound the methods to a system for building of complex systems solving complex tasks?

- Integration is easy if it is sufficient to put methods into a pipeline

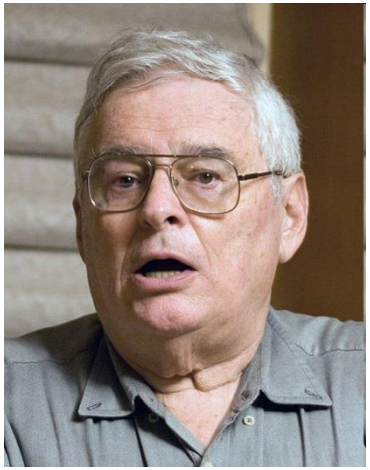


- However, pipelines are not always sufficient

- Solution: we define how methods interacts and the overall system behavior emerges from the interactions

An example of Multi-Agent System (MAS)

- A typical example of MAS is team playing robosoccer: What program have we put to each robot to win the match ?



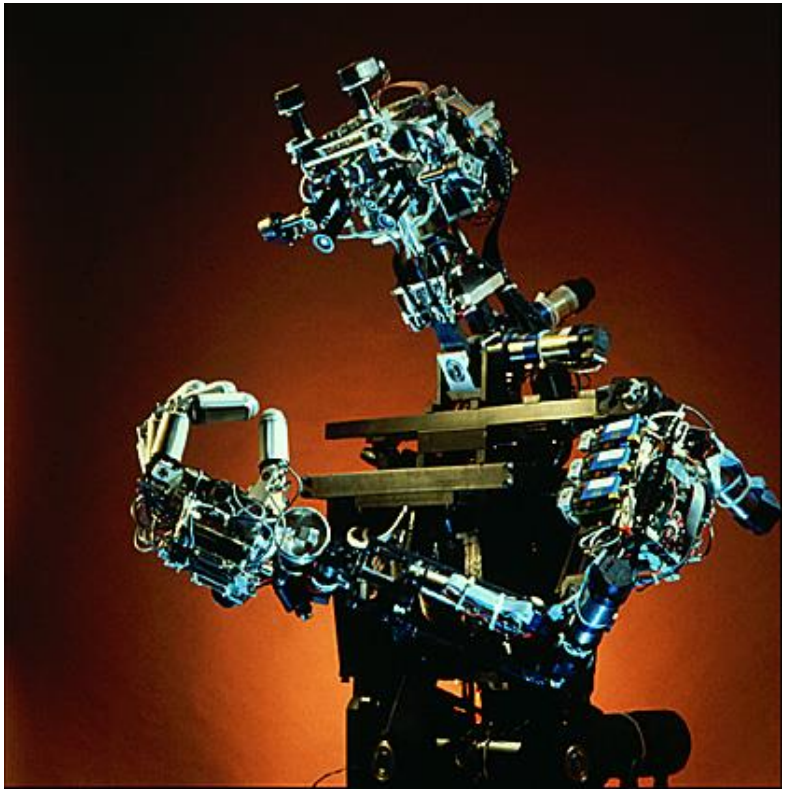
Motivation of AOP

Fodor's mind model:
• modularity of mind

Minsky's society of mind model:
• mind is compounded from agents
• mind works since a accurate group of agents is activated at the accurate moment
• agent is any part or process of the mind that by itself is simple enough to understand – even though the interactions among groups of such agents may produce phenomena that are much harder to understand

An example of AOP

- A good example is control system of humanoid robot: what program we have to put to individual modules which simulate its mind to get a reasonable overall behavior ?

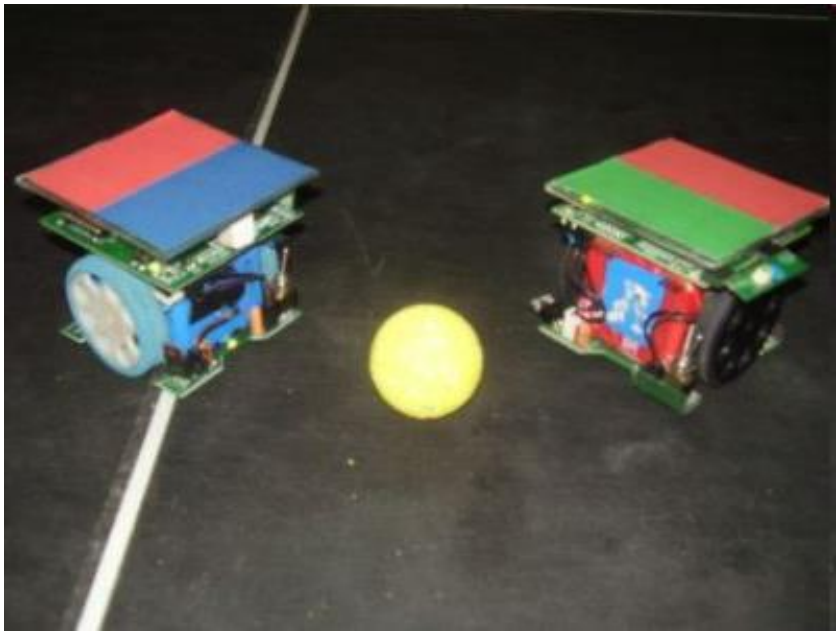


MAS/AOP are domains dealing with systems which exhibit a specific kind of modularity

The modularity is based on distribution and decentralization. The system consists of modules which we call agents. They are able to act in an autonomous way without any necessary stimulus (we say they are proactive)

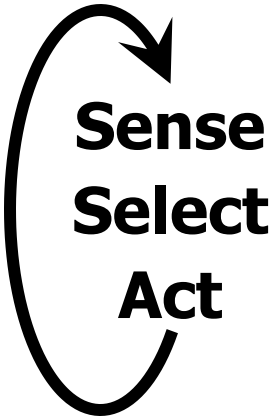
The nature of agents

- Let’s go back to robosoccer
- What schema the program put to the robots has ?

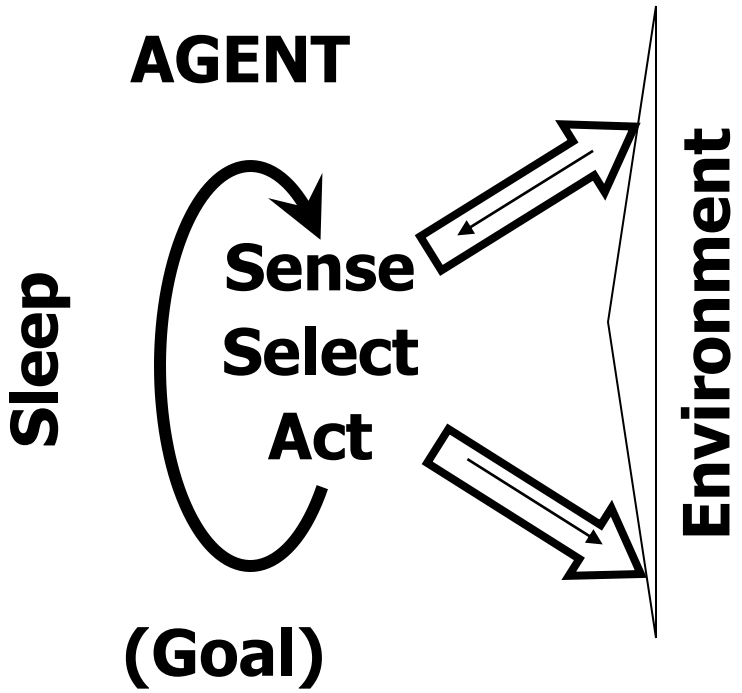


Nature of agent

- We put to robots the following program:



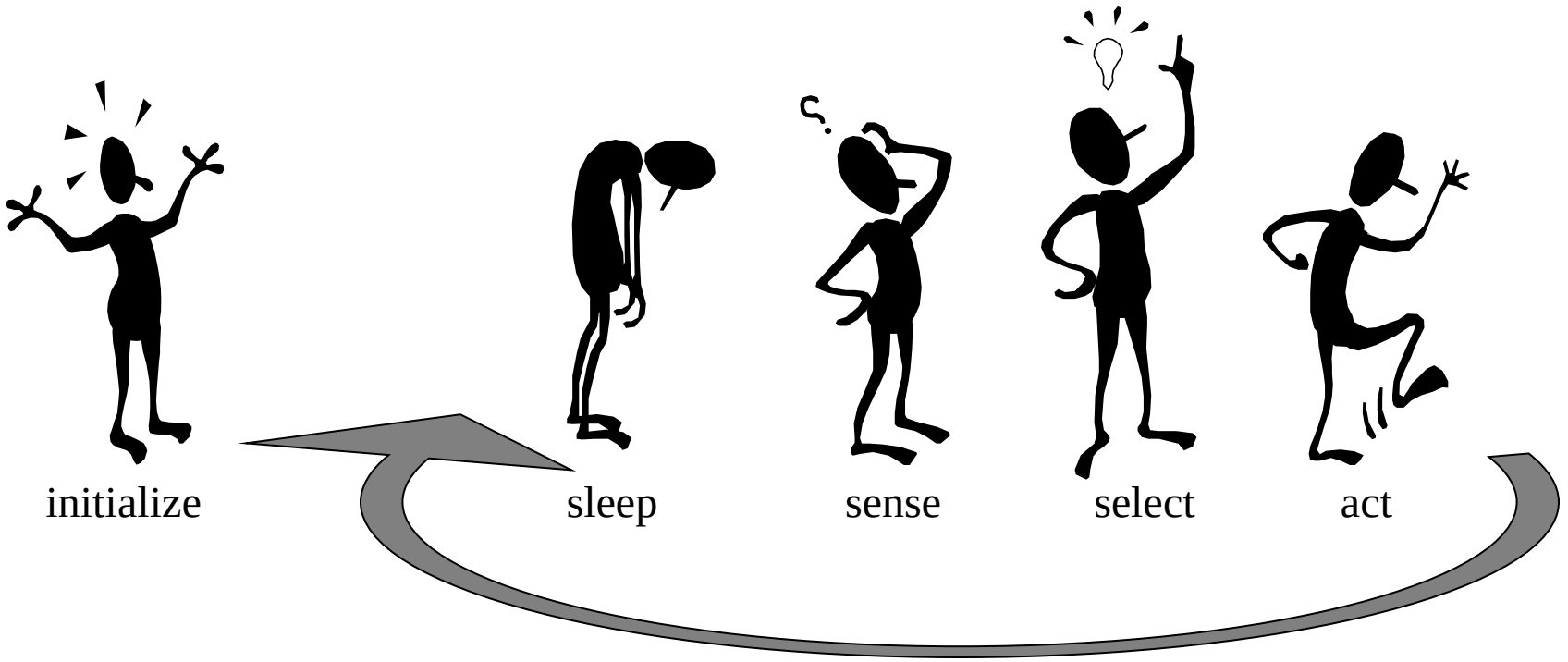
Agent as a process



Agent is a process which continuously senses its environment and selects and perform actions in the environment upon the sensing, following a particular goal

Warning: agent is frequently used metaphor, it is widely used for various purposes, it can be not only an acting entity but also deputy for anything, a mobile entity, …etc. Avoid use of non-sense abstractions.

Agent Life-cycle

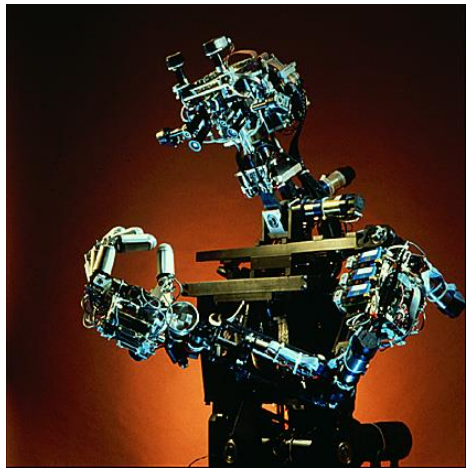


MAS

The network is the computer

AOP

The computer is the network

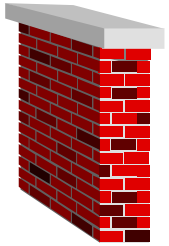
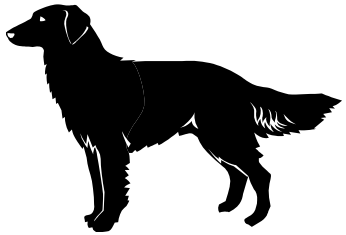


The transfer of real to virtual

- AOP can be considered as a new way how to transfer objects from the real world to the virtual world in computer

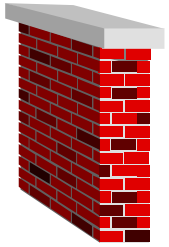
- From historical point of view there are three ways of the transfer. We can classify them following type of activity which can be tied with the transferred object in computer

Types of activity

- **passive entity**  **(record)**
- **reactive entity**  **(object)**
- **proactive entity**  **(agent)**

Types of activity

- **all activity has to be provided from outside**

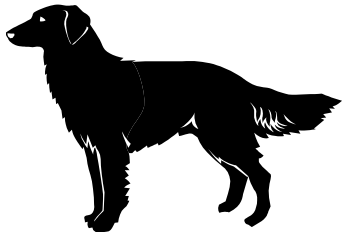


- **structured programming**

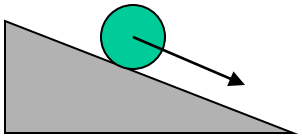
- **entity can provide own activity but only on a stimulus from outside**

- **object-oriented programming**

- **all activity is provided by the entity, it is not needed and even it is not possible to invoke activities from outsides**

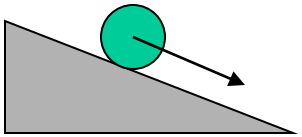


- **agent-oriented programming**



Programming without structures

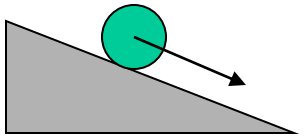
```
#include <math.h>

int main() {
   double x=0.0, y=5.0, fi=0.56;
   int t;
   for (t=0; t<10; t++) {
       x += cos(fi);
       y -= sin(fi);
       sleep(1);
   }
}
```



Structured programming

```
#include <math.h>
typedef struct ball {
   double x;
   double y;
} BALL;

void move(
   BALL *bl,
   double fi
){
   bl->x += cos(fi);
   bl->y -= sin(fi);
}
```

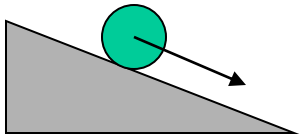
```
int main() {
   BALL b;
   double alpha = 0.56;
   int t;
   b.x = 0.0; b.y = 5.0;
   for (t=0; t<10; t++) {
       move(&b,alpha);
       sleep(1);
   }
}
```



OOP

```
class Ball {
  private:
    double x;
    double y;
  public:
    Ball(double _x, double_y);
    void move(double fi);
}

void Ball::move(
  double fi
){
  x += cos(fi);
  y -= sin(fi);
}
```

```
Ball::Ball (double _x,
  double_y) {
  x = _x;
  y = _y;
}

int main() {
  Ball *b =
    new Ball(1.0, 5.0);
  double alpha = 0.56;
  int t;
  for (t=0; t<10; t++) {
      b->move(alpha);
      sleep(1);
  }
  delete b;
}
```

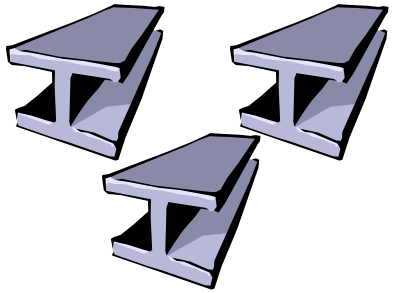


AOP

```
class Ball : Agent {
  private:
    double x;
    double y;
    void move (double fi);
    void handleEvent();
  public:
    Ball(double _x,double_y);
}
Ball::Ball (double _x,
  double_y) {
  x = _x;
  y = _y;
  attachTimer(1);
}
```

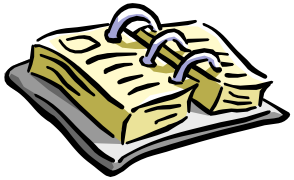
```
Ball::handleEvent() {
  double alpha = (double)
    getSpace().get(˝fi˝);
  this->move(alpha);
}

void Ball::move(…){…}

int main() {
  Space space =
    Space::getInstance();
  space->put(˝fi˝,0.56);
  Agent b =
    new Ball(1.0,5.0);
  sleep(10);
  delete b;
}
```

Multi-agent system



Agents

Communication among agents



Development method

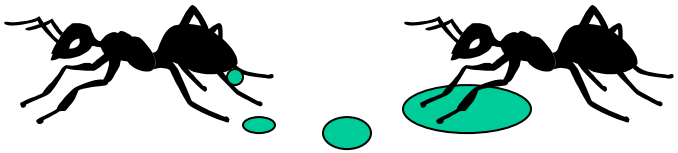
ARCHITECTURE

Communication among agents

Communication

direct – to address

indirect – through environment

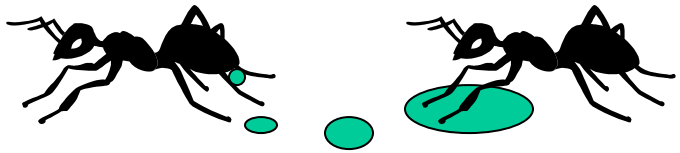


Communication among agents

Counter side reference

static address

named data in environment



Communication among agents

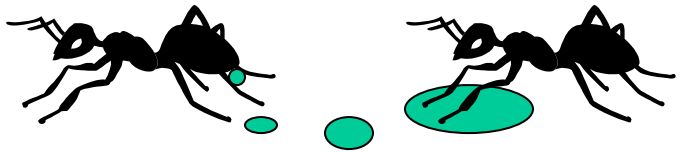
```
send(g,INFROM,
˝(fi 0.56)˝);
```

Communication

```
Space space =
  Space::getInstance();
space->put(˝fi˝,0.56);
```

direct – to an address

indirect – through environment



Communication among agents is complicated bacause:

- Agents can be distributed on more computers or platfoms; e.g., one is written in C++ for 32 bit Windows and other is Java for 64 bit Linux
- They developed by different people on different time and place

Communication among agents

Communicated data:

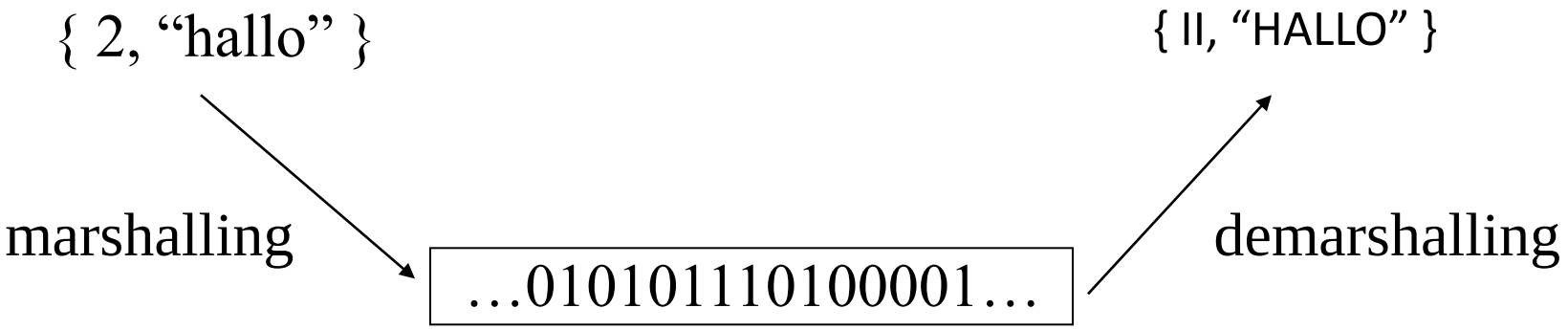
- **raw data** — 00001010 0000000
- **typed data** — Integer 10
- **representation language** — "(age 10 years)" "<age> ten </age>"

  (structured, semi-structured)

Communication envelope:

- **communication language** — "(ask-if (...))"

Marshalling / Demarshalling



- binary form
- text form – based on representation language

Serializable object (Java)

```
import java.io.*;

public class Ball implements Serializable {

  public static final long serialVersionUID = 112132444L;

  float radius;
  public float x;
  public float y;

  public Ball (float radius) {
    this.radius = radius;
  }

  public String toString () {
    return radius+"["+x+","+y+"]";
  }

}
```

Marshalling

```
import java.io.*;

public class Marshall {

  public static void main (String[] args) throws Exception {
    Ball b = new Ball(1.0f);
    b.x = 0.0f; b.y = 0.0f;
    ByteArrayOutputStream byteStream = new ByteArrayOutputStream();
    ObjectOutputStream marshaller = new ObjectOutputStream(byteStream);
    marshaller.writeObject(b);
    byte[] bytes = byteStream.toByteArray();
    System.out.println(bytes.length);
    for (int i=0; i<bytes.length; i++) System.out.print(bytes[i]+" ");
    System.out.println();
  }

}
```

Demarshalling

```
import java.io.*;

public class Demarshall {

  public static void main (String[] args) throws Exception {
    byte[] marshalledObject = {
      -84, -19, 0, 5, 115, 114, 0, 5, 71, 117, 108, 107, 97,
      0, 0, 0, 0, 6, -81, 1, 92, 2, 0, 3, 70, 0, 6, 114, 97,
      100, 105, 117, 115, 70, 0, 1, 120, 70, 0, 1, 121, 120,
      112, 63, -128, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
    };
    ByteArrayInputStream byteStream = new
   ByteArrayInputStream(marshalledObject);
    ObjectInputStream demarshaller = new
   ObjectInputStream(byteStream);
    Ball b = (Ball) demarshaller.readObject();
    System.out.println(b); // vypise 1.0[0.0,0.0]
  }

}
```

Languages for data communication among agents

- Representation language – expression of communication content (what is communicated)

- Communication language – expression of communication envelope (how it is communicated)

Representation language

- KIF – common syntax and semantics; ontology based on vocabulary
- XML – common syntax, one ontology can be translated to others

KIF (Knowledge Interchange Format)

- syntax – LISP
- semantics – one big vocabulary

When we want to express point [1, 1] in 2D plane, we need to check vocabulary and we find that we can use:

(point (euclid-coordinates 1 1))

or

(point (polar-coordinates 1.1412136 0.7853981))

KIF - Example

```
(instance simultaneousProcess BinaryPredicate)

(documentation simultaneousProcess
   "(simultaneousProcess ?Proc1 ?Proc2) means that processes
   ?Proc1 ?Proc2 cooccur at the same object, but not
   necessarily at the same region. "
)

(<=>
   (simultaneousProcess ?Proc1 ?Proc2)
   (and (cooccur ?Proc1 ?Proc2)
           (exists (?object)
                      (and (patient ?Proc1 ?Object)
                              (patient ?Proc2 ?Object)))))
```

SUMO (Suggested Upper Merged Ontology)

- http://www.ontologyportal.org/
- KIF is ontologic language
- SUMO is ontologic vocabulary written in KIF

SUMO - Examples

```
(=>
    (and
        (instance ?SUBSTANCE PlantSubstance)
        (instance ?PLANT Organism)
        (part ?SUBSTANCE ?PLANT))
    (instance ?PLANT Plant))
```

```
(=>
    (instance ?DRIVE Driving)
    (exists (?VEHICLE)
        (and
            (instance ?VEHICLE Vehicle)
            (patient ?DRIVE ?VEHICLE))))
```

XML

<tag attribute=value … > data </tag>	**element with data**
<tag attribute=value ... />	**empty element**
<? … ?>	**directive**
<!-- remark -->	**remark**
<![CDATA[…]]>	**raw data**

`& &quot; &apos; < >`

XML

```
<?xml version=“1.0” ?>
<announcement>
        <going-to who=“007”>
                10 10 <stop/>
        </going-to>
</announcement>
```

```
<?xml version=“1.0” ?>
<announce-going-to>
        007 10 10 <stop/>
</announce-going-to>
```

DTD, DSD, RELAX NG, XML Schema

XML Transformer XSLT

Elementy:
<xsl:attribute-set>
<xsl:decimal-format>
<xsl:import>
<xsl:include>
<xsl:key>
<xsl:namespace-alias>
<xsl:output>
<xsl:param>
<xsl:preserve-space>
<xsl:strip-space>
<xsl:template>
<xsl:variable>
<xsl:script>

match= name= priority= mode=

Inštrukcie template:
<xsl:apply-imports>
<xsl:apply-templates>
<xsl:attribute>
<xsl:call-template>
<xsl:choose>
<xsl:comment>
<xsl:copy>
<xsl:copy-of>
<xsl:element>
<xsl:fallback>
<xsl:for-each>
<xsl:if>
<xsl:message>
<xsl:number>
<xsl:processing-instruction>
<xsl:text>
<xsl:value-of>
<xsl:variable>

select= disable-output-escaping=

XSLT

```
<?xml version=“1.0” ?>
<xsl:stylesheet version=“1.0” xmlns:xsl= http://www. w3.org/1999/XSL/Transform>
<xsl:template match=“/annoucement”>
        <oznam-idem-na>
        <xsl:apply-templates select=“going-to”/>
         <xsl:value-of select=“going-to”/>
        </oznam-idem-na>
</xsl:template>
<xsl:template match=“going-to”>
        <xsl:value-of select=“@who”/>
</xsl:template>
```

XSLT

```
<?xml version="1.0" ?>
  <announcement>
   <going-to who="007">
        10 10 <stop/>
    </going-to>
  </announcement>
```

```
<?xml version="1.0" ?>
  <oznam-idem-na>
      007 10 10 <stop/>
  </oznam-idem-na>
```

How to use XML

```
<ball>
   <radius>1</radius>
</ball>
```

(semi-structured data)

```
<ball radius="1"/>
```

(structured data)

Communication language

- speech acts – KQML
- syntax – LISP-like frames

```
(performative
        :parameter      value
        :parameter      value
)
```

KQML (Knowledge Query Manipulation Language)

Performatives:

Achieve	sender asks receiver to perform the content
Advertise	sender advertises a service and its request form
Ask-all	sender asks for all responses to the question in the content
Ask-one	sender asks for one response to the question in the content
Broker-one	sender asks what agent can answer the question
Delete-one	sender announces that the content is not valid anymore
Insert	sender announces that the content is valid and should be concerned
Ready	sender announces that it can provide content
Register	sender registers itself, announces its existence
Recommend-one	sender asks to recommend agent which knows something
Recruit-one	sender asks to find agent which knows something
Sorry	sender has not the required information
Subscribe	sender asks to be informed when something changes
Tell	sender send something to receiver

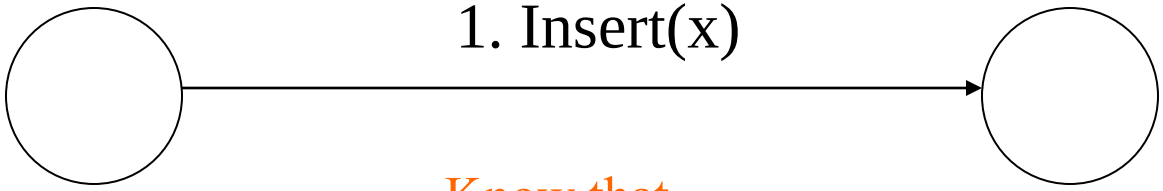
...

KQML

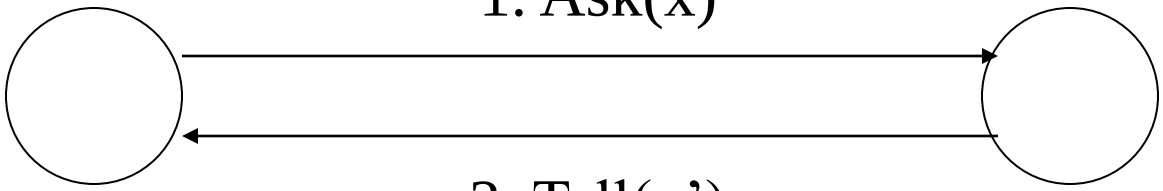
Parameters	
:sender	who sends the message
:receiver	who receives the message
:from	from whom the messages comes
:to	to whom the message is sent
:in-reply-to	request for marking the answer
:reply-with	marking the answer
:language	syntax
:ontology	semantics
:content	content
...	

KQML- scenarios

Direct communication



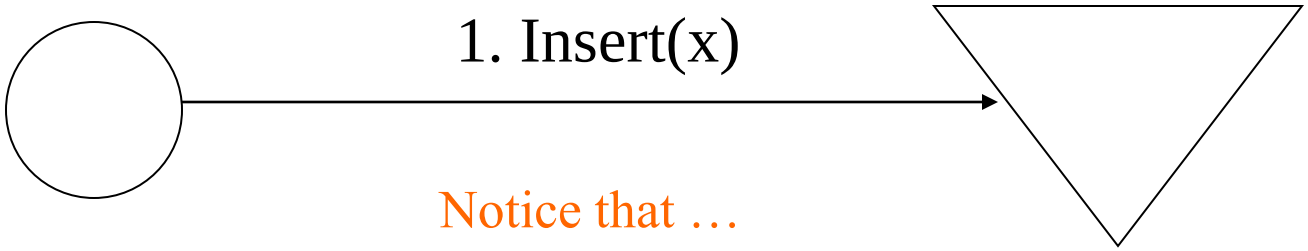
Know that …



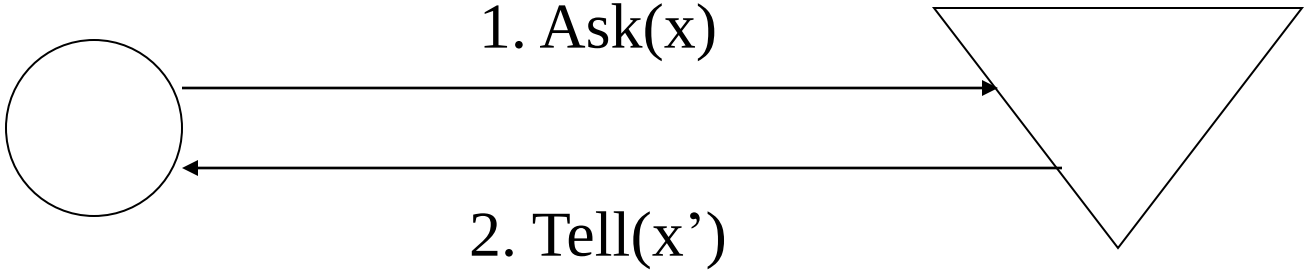
Tell me …

KQML- scenarios

Indirect communication



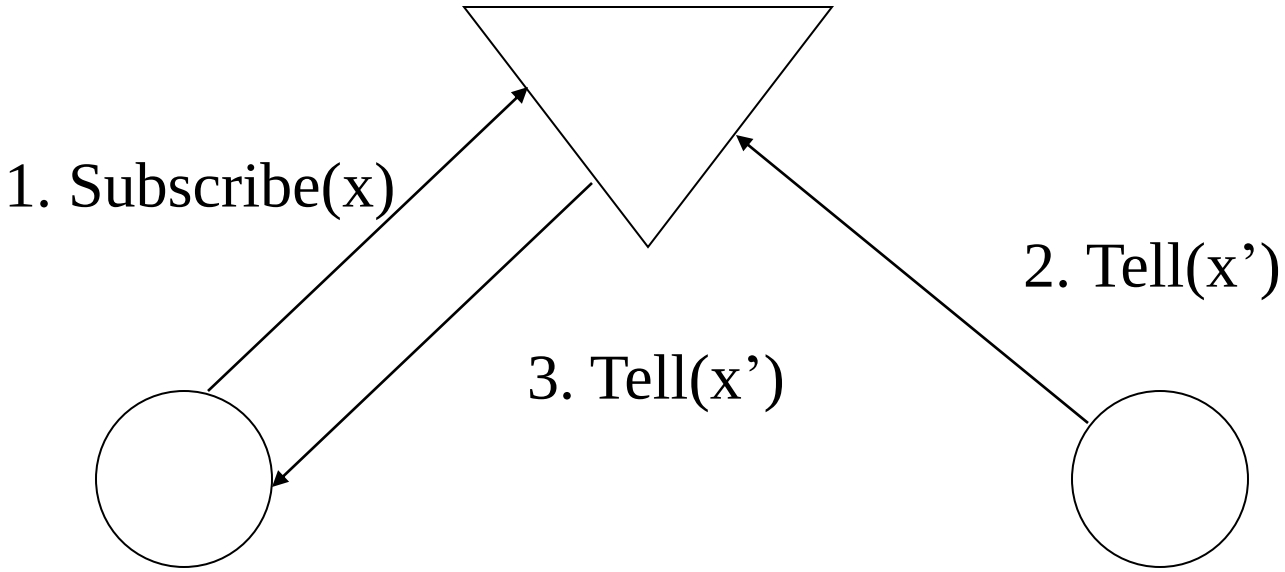
Notice that …



What do you notice about x ?

KQML- scenarios

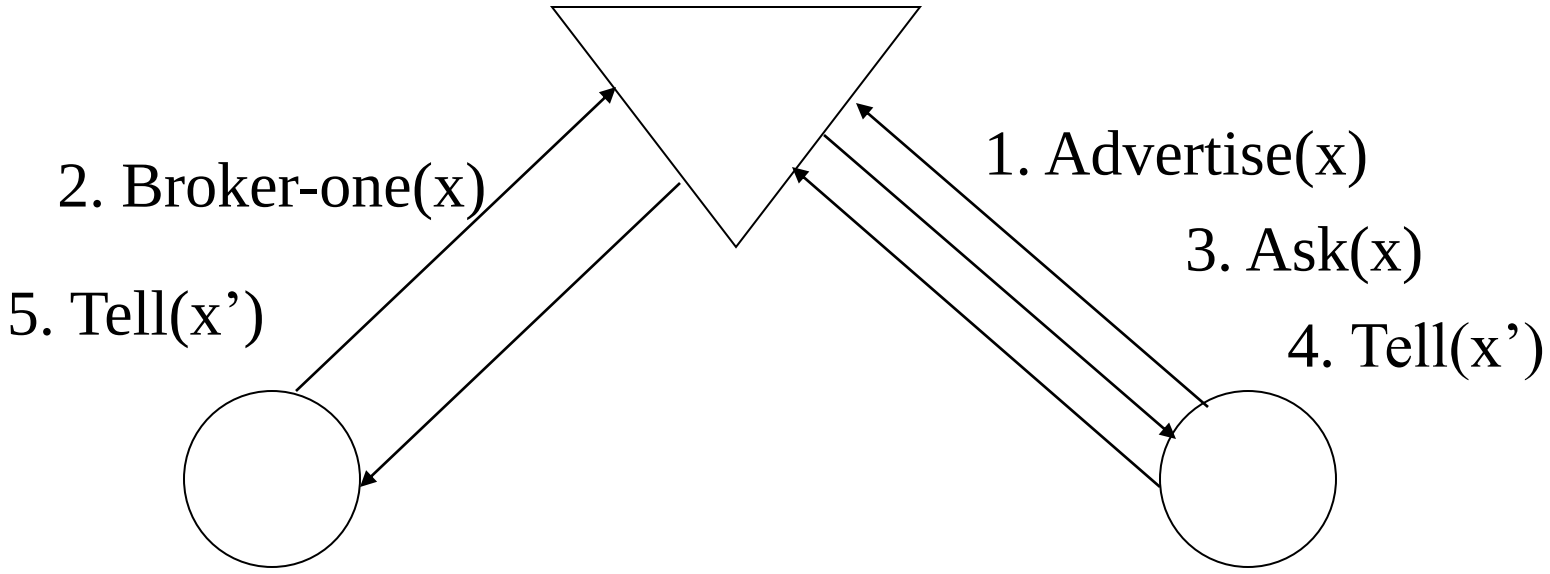
Indirect communication– request on trigger



If somebody tells, I want to listen

KQML- scenarios

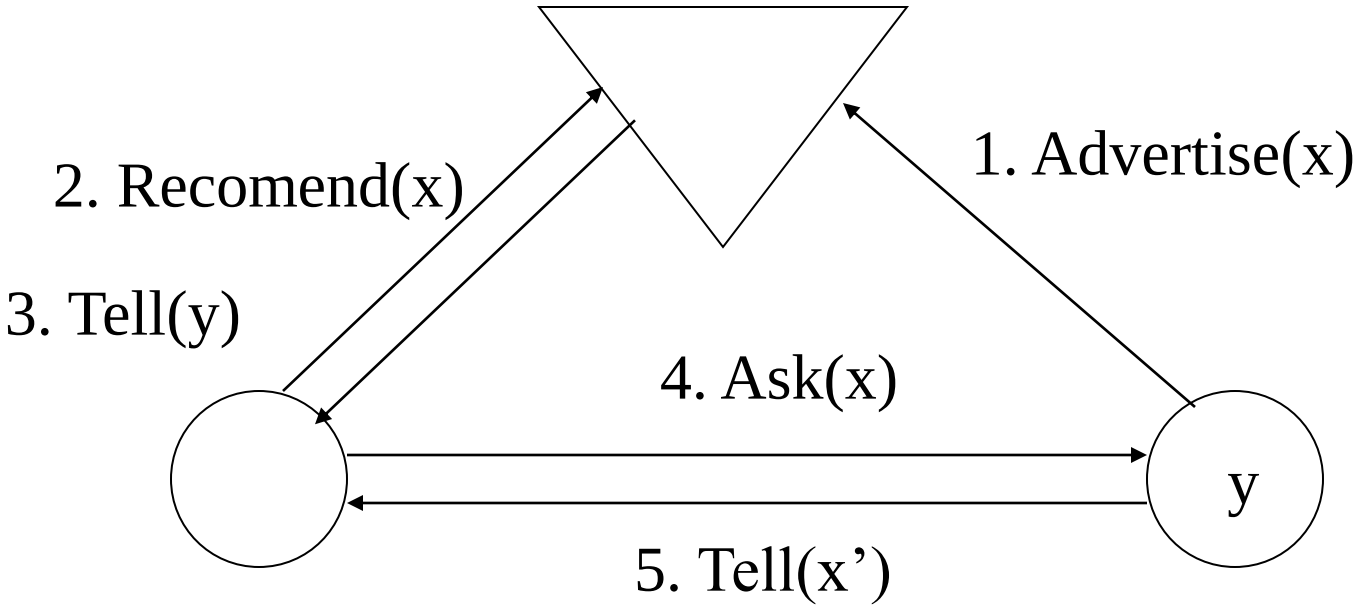
Indirect communication – request bidding



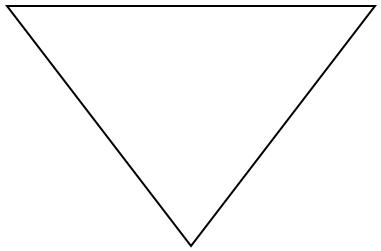
Find answer to my request

KQML- scenarios

Direct communication – request bidding



Recommend me who can answer my question...



What is this?

- meta-agent
- facilitator
- container - mediator
- space
- (environment)