Multiagentové systémy

Andrej Lúčny

KAI FMFI UK

lucny@fmph.uniba.sk

http://www.agentspace.org/mas

čo nás čaká:

primárna platforma: **Java**

Prednášajúci

RNDr. Andrej Lúčny, PhD.

- zaoberám sa umelou inteligenciou
- som spoluzakladateľ po celom svete operujúcej firmy dodávajúcej hardvér a softvér monitorovacích systémov
- aktuálne vyvíjam aplikácie počítačového videnia v jej dcérskej spoločnosti
- som spoluzakladateľ občianskeho združenia robotika.sk
- som bývalý porotca ACM Scholastic Programming Contest
- som porotca súťaže mobilných robotov ISTROBOT

www.agentspace.org/andy

Cieľ predmetu

- zvládnuť technológie integrácie metód umelej inteligencie na báze multiagentových systémov (serializácia objektov, sychronizácia procesov, medzivláknová, medziprocesová a sieťová komunikácia, práca v reálnom čase, manipulácia s poznatkami)
- naučiť sa implementovať multiagentový framework
- naučiť sa použiť existujúce multiagentové frameworky
- dozvedieť sa viac o problematike tvorby komplexnejších systémov umelej inteligencie
- vedieť použiť sprievodné technológie (virtuálna realita, fázová korelácia, HOG detektor, Houghova transformácia, kaskádny regresor, hlboká neurónová sieť) ako black-box

Anketa

Napísali ste niekedy program, ktorý úspešne použil:

- Jazyk Java
- anonymnú triedu (napr. TimeredTask)
- objekt odvodený od Thread (vlákno)
- objekt Serializable (serializovateľný objekt)
- objekt Socket (sieťová komunikácia)
- nejaký middle ware
- nejakú implementáciu multi-agentového systému

Hodnotenie

cca max. 44 bodov za semester
(cca 12 týždňov)
každý týždeň:
1 bod za účasť
1 bod za splnenie základu cvičenia
1 bod za najlepší test
1 bod za domácu úlohu

+ 1 nenárokovateľný bod za úspechy alebo hviezdičkovú úlohu

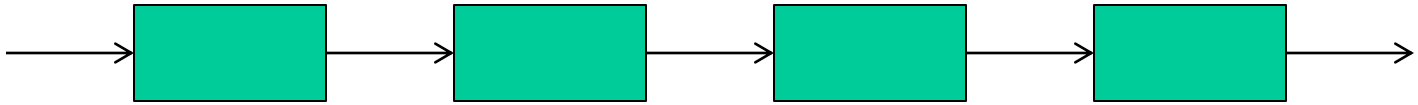
Skúška:
40 bodov za test

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

v prípade dlhodobej choroby, pobytu v zahraničí a podobne – je možný projekt

- V oblasti UI je už známych množstvo zaujímavých metód na riešenie čiastkových úloh, napríklad vieme rozpoznať na obrázku tvár.
- Ako tieto metódy skladať dohromady, aby sme budovali komplexné systémy a riešili komplexné úlohy?

- Dobre nám to ide, keď nám tieto metódy stačí zoradiť za sebou do „pipeline“

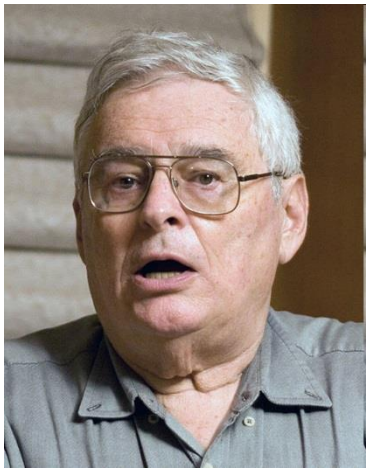


- Horšie je, keď pipeline nestačí

- Riešenie: zadefinujeme ako spolu tieto metódy interagujú a správanie systému necháme z toho vyplynúť

Ukážka Multi Agentového Systému (MAS)

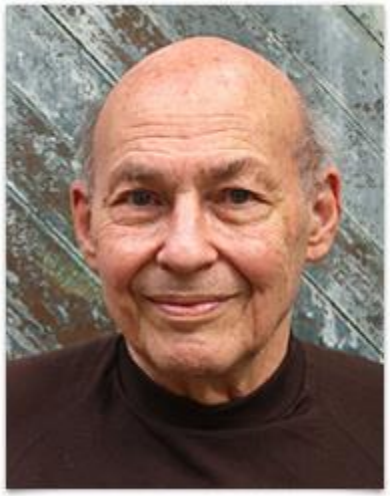
- Typickou ukážkou MAS je napr. robosoccer: Aký program vložiť do jednotlivých robotov aby vyhrali zápas ?



Motivácia AOP

Fodorov model mysle:

• modulárnosť mysle



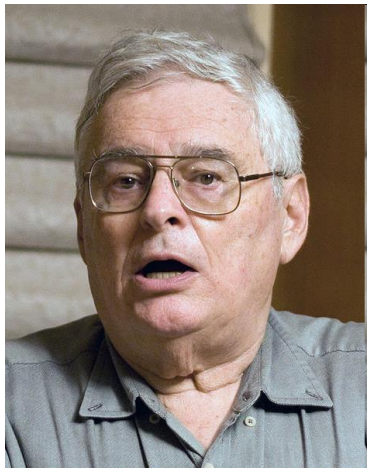
Minského societný model mysle:

• myseľ je zložená z agentov

• jej funkcia spočíva v správnej aktivácii časti agentov v správny okamih

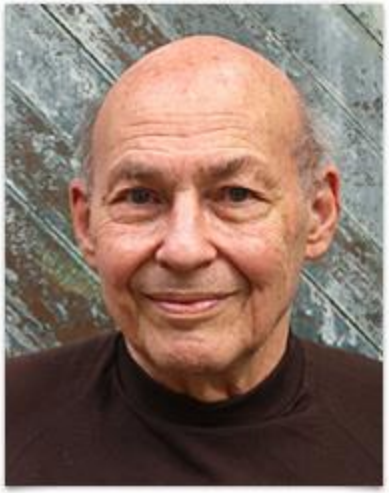
• agent je časť mysle alebo proces v nej, ktorým samým o sebe je pomerne ľahké porozumieť – zatiaľ čo interakcie medzi skupinami takých agentov môžu vyprodukovať javy, ktorým porozumieť je oveľa ťažšie

Motivácia Agentovo Orientovaného Programovania (AOP)



Fodorov model mysle:

• modulárnosť mysle



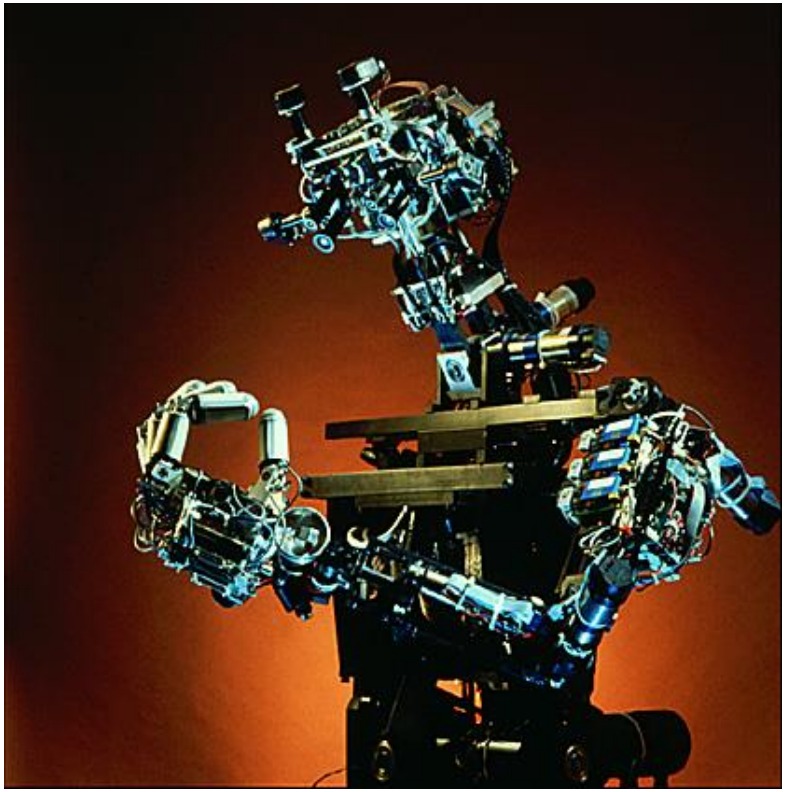
Minského societný model mysle:

• myseľ je zložená z agentov

• jej funkcia spočíva v správnej aktivácii časti agentov v správny okamih

Ukážka AOP

- Typickou ukážkou AOP je napr. riadiaci systém humanoidného robota: Aký program vložiť do jednotlivých modulov realizujúcich jeho myseľ aby konal rozumne ?

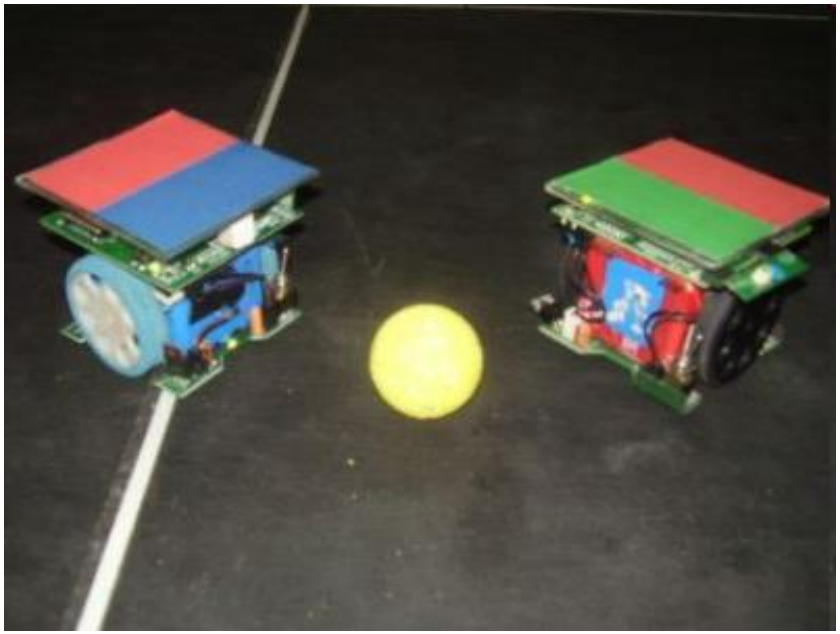


MAS / AOP sú oblasti zaoberajúce sa systémami, ktoré sa vyznačujú špecifickým druhom modularity

Táto modularita je postavená na **distribúcii** a **decentralizácii**. Systém sa tu skladá z modulov, ktoré nazývame agentmi. Tieto sa vyznačujú schopnosťou konať nezávisle na ostatných moduloch (sú **autonómne**) a bez toho, že by boli k činnosti niekým volané (sú **proaktívne**)

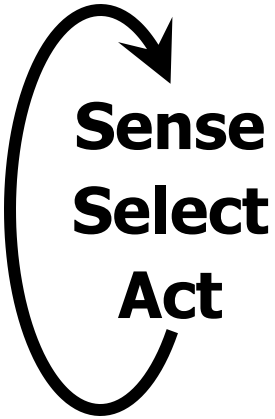
Povaha agenta

- Vráťme sa k robosocceru
- Aký tvar bude mať program, ktorý do robotov budeme vkladať ?

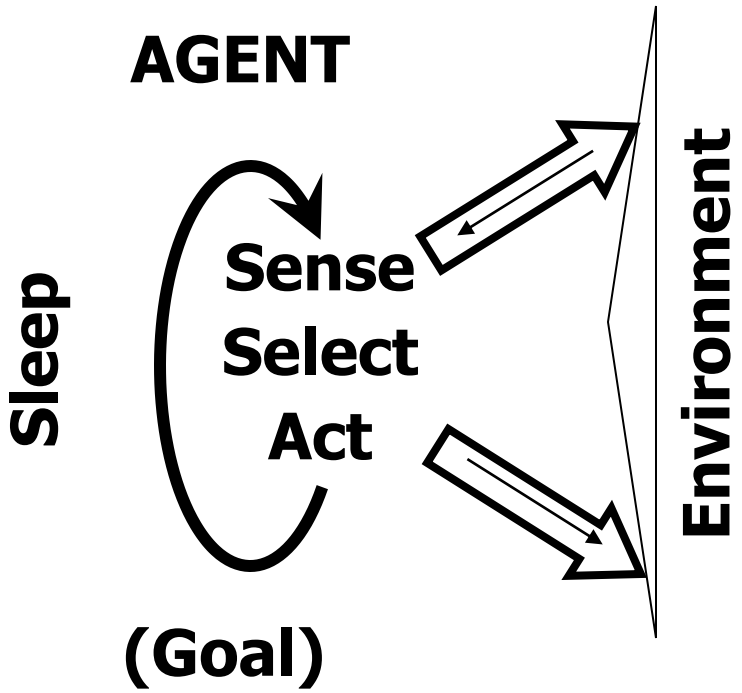


Povaha agenta

- Do robotov budeme vkladať program tvaru:



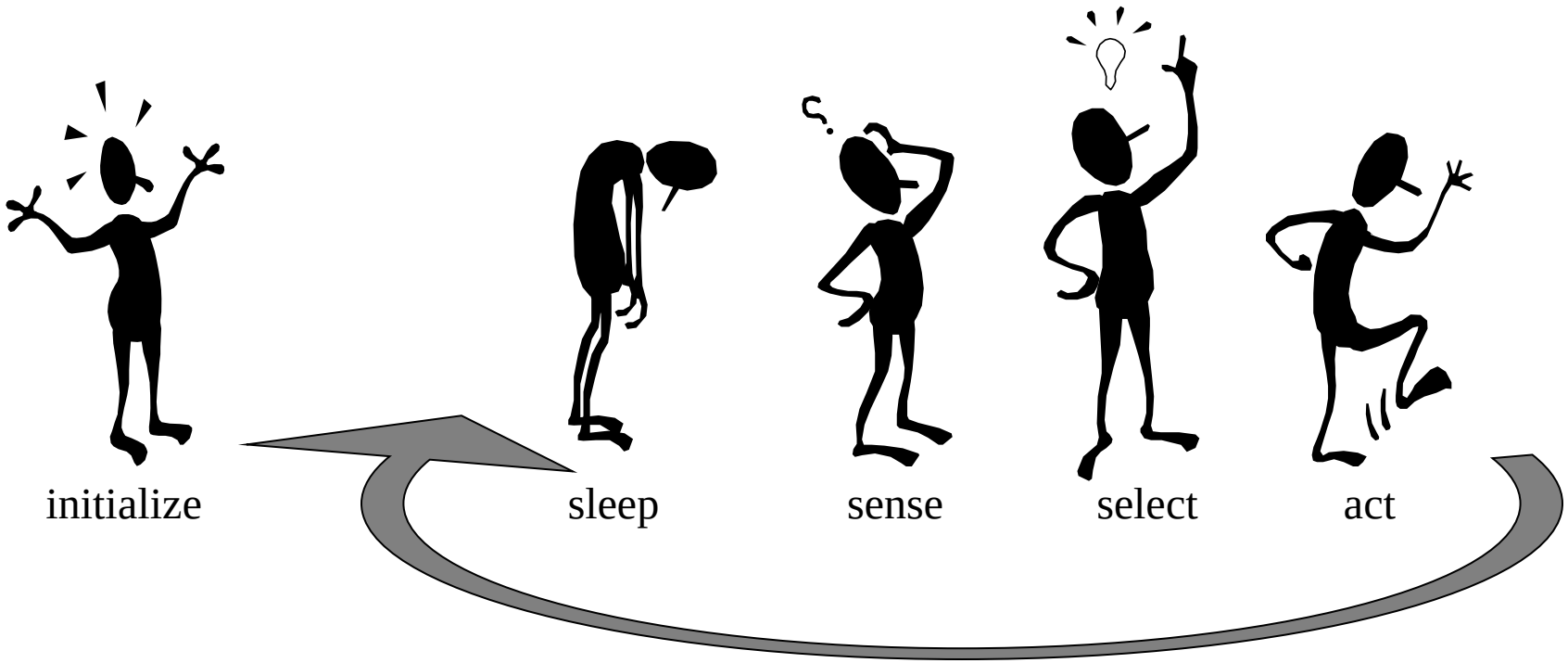
Agent - proces



Agent je proces, ktorý neustále (opakovane) vníma svoje prostredie a na základe toho volí a vykonáva v ňom akcie, pričom sa jeho počínaniu dá pripísať určitý cieľ

Varovanie: agent je veľmi rozšírená metafora, používa sa kdekoľvek, kde niečo zastupuje niekoho a jej jednotlivé použitia nie je vhodné spájať do nezmyselných abstrakcií.

Životný cyklus agenta

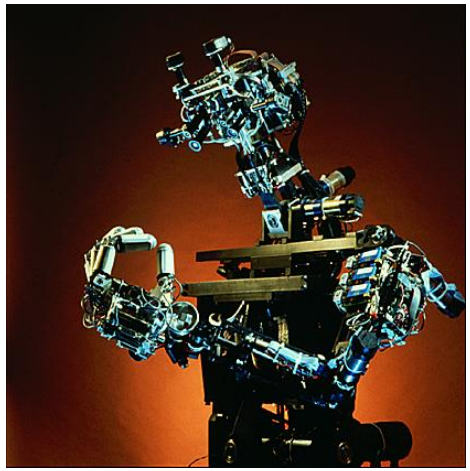


MAS

The network is the computer

AOP

The computer is the network

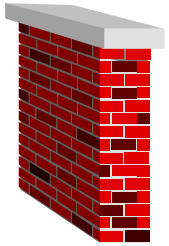
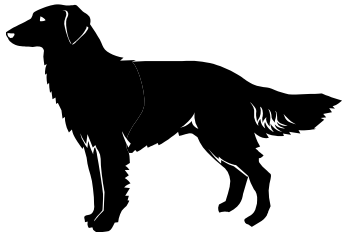


Prenos reálneho do virtuálneho

- Na AOP sa dá nazerať ako na nový spôsob sťahovania objektov reálneho sveta do počítača.

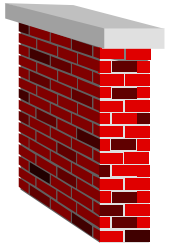
- Historicky existujú tri takéto spôsoby a môžeme ich klasifikovať na základe typu aktivity jednotky, ktorá v počítači reálny objekt predstavuje

Typy aktivity

- **pasívna entita**  **(record)**
- **reaktívna entita**  **(object)**
- **proaktívna entita**  **(agent)**

Typy aktivity

- **každú aktivitu musíme vyvolať my**

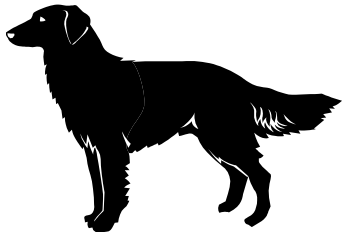


- **programovanie so štruktúrami**

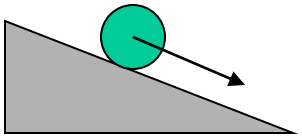
- **entita dokáže vyvolať rôzne aktivity, ale iba ak ju k nejakej prvotnej aktivite vyvoláme my**

- **objektovo-orientované programovanie**

- **aktivitu nemusíme ani nemôžeme vyvolať**

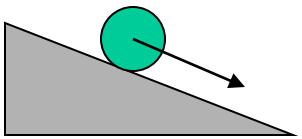


- **agentovo-orientované programovanie**



Programovanie bez štruktúr

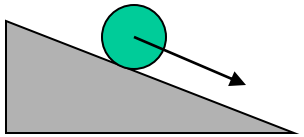
```
#include <math.h>

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);
   }
}
```



Programovanie so štruktúrami

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

void posun(
   GULKA *gul,
   double fi
){
   gul->x += cos(fi);
   gul->y -= sin(fi);
}
```

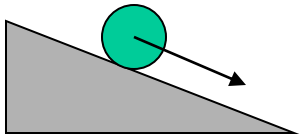
```
int main() {
   GULKA g;
   double alpha = 0.56;
   int t;
   g.x = 0.0; g.y = 5.0;
   for (t=0; t<10; t++) {
      posun(&g,alpha);
      sleep(1);
   }
}
```



OOP

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

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

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

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

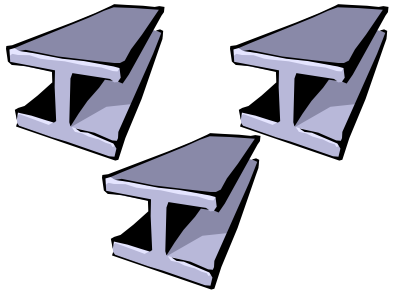


AOP

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

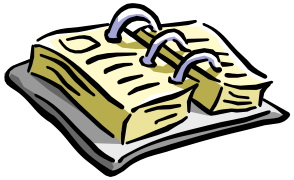
```
Gulka::handleEvent() {
  double alpha = (double)
    getSpace().get(˝fi˝);
  this->posun(alpha);
}

void Gulka::posun(…){…}

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

Multiagentový systém



Agenty

Komunikácia medzi agentami



Metóda tvorby

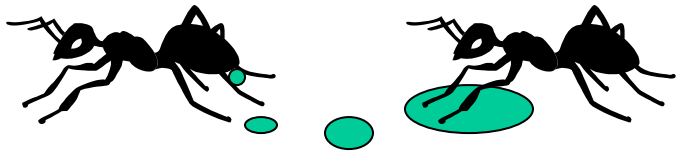
ARCHITEKTÚRA

Komunikácia medzi agentami

Komunikácia

priama - adresná

nepriama -cez prostredie

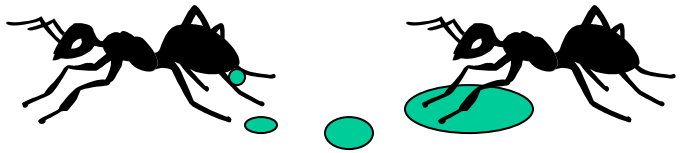


Komunikácia medzi agentami

Referencia protistrany

pevná adresa

pomenované odkazy v pevne adresovanom prostredí



Komunikácia medzi agentami

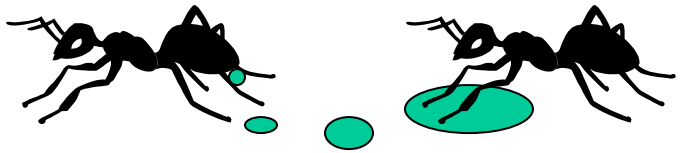
```
send(g,INFROM,
˝(fi 0.56)˝);
```

Komunikácia

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

priama - adresná

nepriama -cez prostredie



Komunikáciu medzi agentmi komplikuje to, že môžu byť:

- distribuované na viacerých počítačoch a platformách napríklad jeden agent je v C++ na 32 bit Windows-och a druhý je v Java na 64 bit Linux-e
- vyvíjané rôznymi vývojármi, ktorí sa trebárs nikdy v živote nevideli

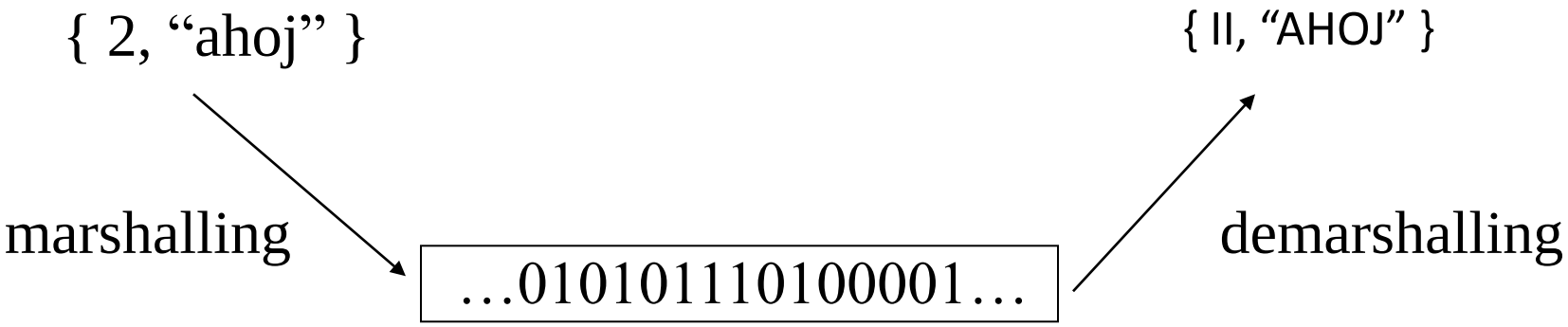
Komunikované dáta:

- **holé dáta** — 00001010 0000000
- **dáta s definovaným typom** — Integer 10
- **reprezentačný jazyk** (**štruktúrované, semištrurované**) — “(vek 10 rokov)” “<vek> ten </vek>”

Komunikačná obálka:

- **komunikačný jazyk** — “(ask-if (...))”

Marshalling / Demarshalling



- Marshallované dáta môžu mať **binárnu** alebo **znakovú** podobu.
- Ide o vyjadrenie dát v tzv. reprezentačnom jazyku

Java Serialization

```
import java.io.*;

public class Gulka implements Serializable {

  public static final long serialVersionUID = 112132444L;

  float radius;
  public float x;
  public float y;

  public Gulka (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 {
    Gulka g = new Gulka(1.0f);
    g.x = 0.0f; g.y = 0.0f;
    ByteArrayOutputStream byteStream = new ByteArrayOutputStream();
    ObjectOutputStream marshaller = new ObjectOutputStream(byteStream);
    marshaller.writeObject(g);
    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);
    Gulka g = (Gulka) demarshaller.readObject();
    System.out.println(g); // vypise 1.0[0.0,0.0]
  }

}
```

Jazyky komunikácie medzi agentami

- Reprezentačný jazyk – vyjadruje komunikovaný obsah (čo sa komunikuje)

- Komunikačný jazyk – vyjadruje komunikačnú obálku (ako sa to komunikuje)

Reprezentačný jazyk

- KIF - jednoznačne určuje ako sa niečo povie (definuje jedinú sémantiku slovníka)
- XML – veľa rôznych výpovedí môže znamenať to isté, ale syntax umožňuje preložiť jednu formu na druhú

KIF (Knowledge Interchange Format)

- dohodnutá syntax – LISP
- dohodnutá sémantika – obrovský slovník

Keď chce programátor zapísať bod [1, 1] v rovine, pozrie do slovníka a zistí že je to

(point (euclid-coordinates 1 1))

alebo

(point (polar-coordinates 1.1412136 0.7853981))

- v princípe funguje, ale pramálo sa používa

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 je ontologický jazyk
- SUMO je ontologický slovník v jazyku 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>

tag s dátami

<tag attribute=value ... />

prázdny tag

<? … ?>

direktíva

<!-- remark -->

poznámka

<![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” ?>
  <oznam-idem-na>
       007 10 10 <stop/>
  </oznam-idem-na>
```

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>
```

Práca s XML

```
<gulka>
   <radius>1</radius>
</gulka>
```

(semištruktúrovaná forma)

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

(plneštruktúrovaná forma)

Komunikačný jazyk

- rečové akty – KQML
- syntax – LISPovské rámce

```
(performatív
        :parameter      hodnota
        :parameter      hodnota
)
```

KQML (Knowledge Query Manipulation Language)

Performatívy	
Achieve	sender žiada receivera o vykonanie contentu
Advertise	sender ponuka službu a tvar žiadosti v contente
Ask-all	sender žiada o všetky odpovede na otazku v contente
Ask-one	sender žiada o jedinú odpoveď na otázku v contente
Broker-one	sender sa pýta kto vie zodpovedať otázku v contente
Delete-one	sender oznamuje že content už neplatí – zmažte si ho
Insert	sender oznamuje že content platí – zapamätajte si
Ready	sender oznamuje že už vie urobiť content
Register	sender oznamuje že existuje
Recommend-one	sender žiada aby mu odporučili agenta ktorý pozná content
Recruit-one	sender žiada aby mu našli agenta ktorý pozná content
Sorry	sender nemá k dispozícii požadovanú informáciu
Standby	sender oznamuje že vie urobiť content
Subscribe	sender žiada aby mu dali vedieť keď sa zmení content
Tell	sender posiela receivrovi čo sa ho pýtal

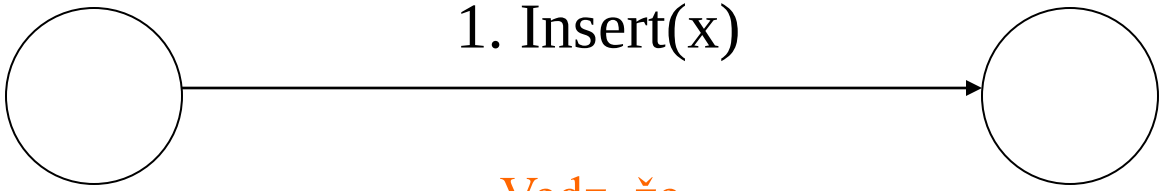
...

KQML

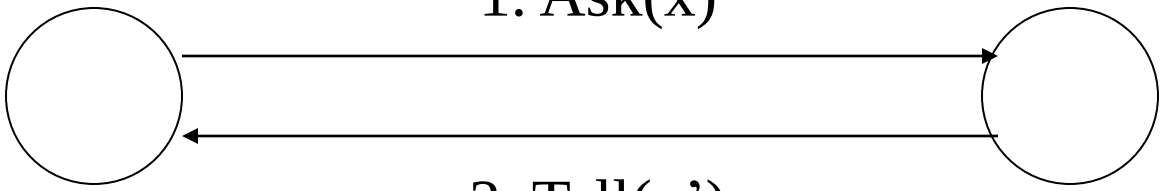
Parametre	
:sender	kto posiela správu
:receiver	kto má prijať správu
:from	od koho správa pochádza
:to	komu je správa určená
:in-reply-to	žiadosť o označenie odpovede
:reply-with	označenie odpovede
:language	názov syntaxe contentu
:ontology	názov sémantiky contentu
:content	obsah správy
...	

KQML- scenáre

Priama komunikácia



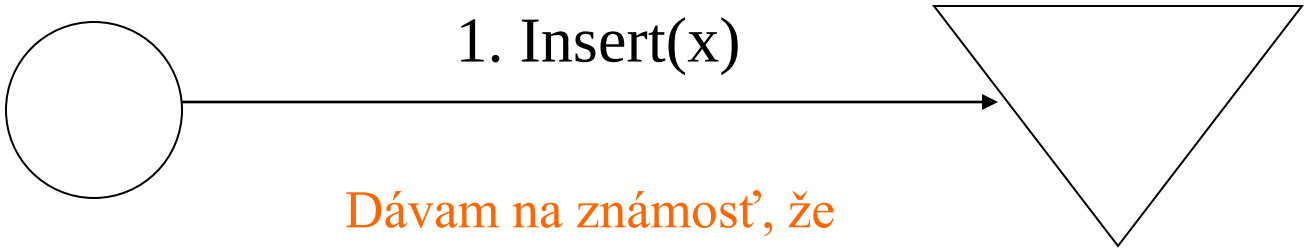
Vedz, že



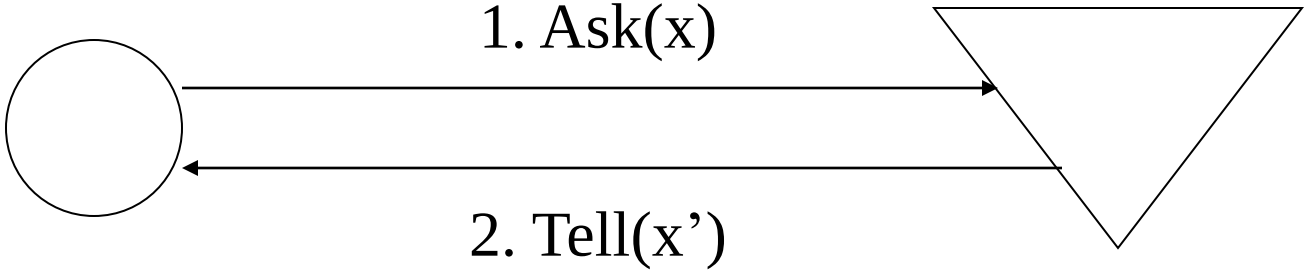
Povedz mi

KQML- scenáre

Nepriama komunikácia



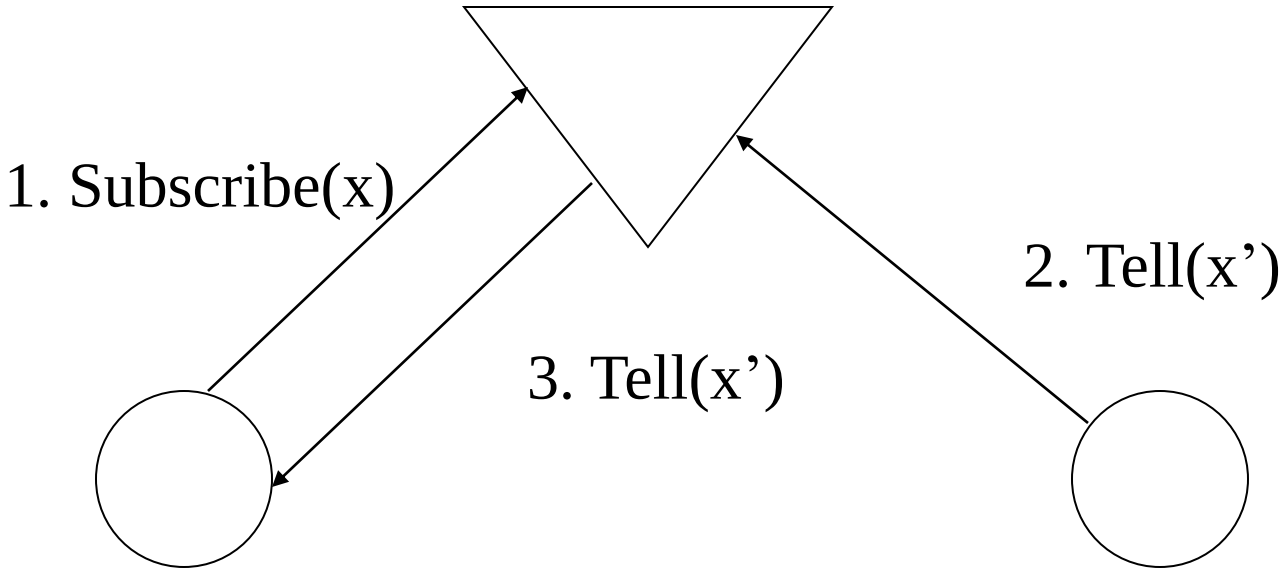
Dávam na známosť, že



Čo sa dalo naposledy na známosť o tomto ?

KQML- scenáre

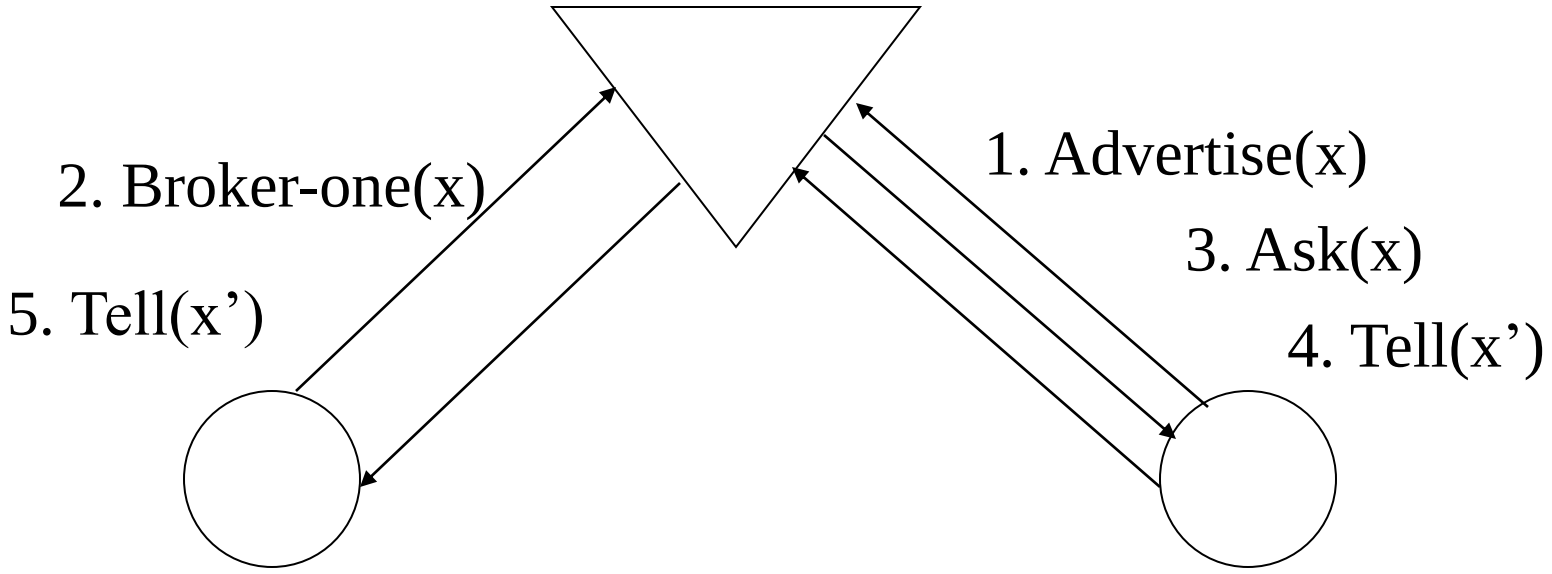
nepriama komunikácia – request on trigger



Keď to niekto povie chcem to tiež vedieť

KQML- scenáre

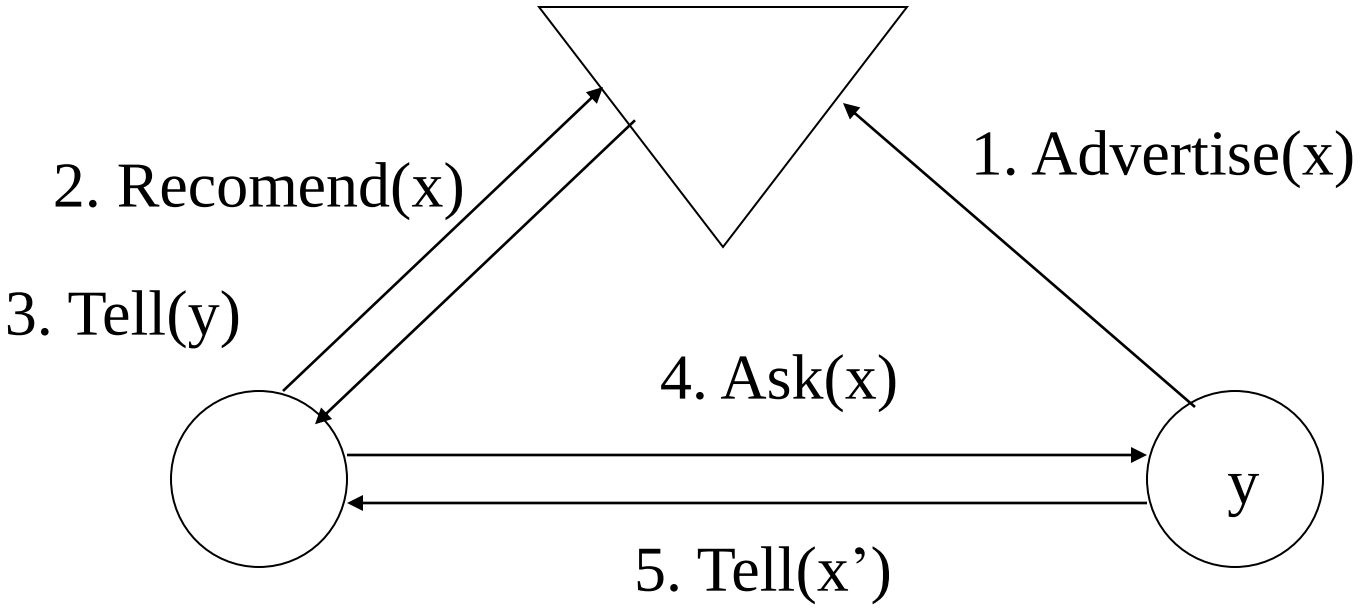
nepriama komunikácia – request bidding



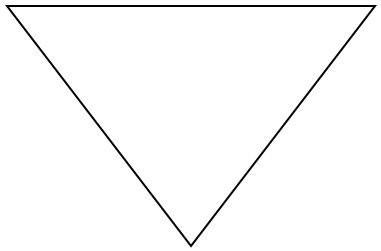
Zistite mi odpoveď

KQML- scenáre

priama komunikácia – request bidding



Povedzte mi kto vie odpovedať



čo je to ?

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