

Multiagentové systémy

Dr. Andrej Lúčny

KAI FMFI UK

andy@microstep-mis.com

<http://www.microstep-mis.sk/~andy>

Motivácia MAS

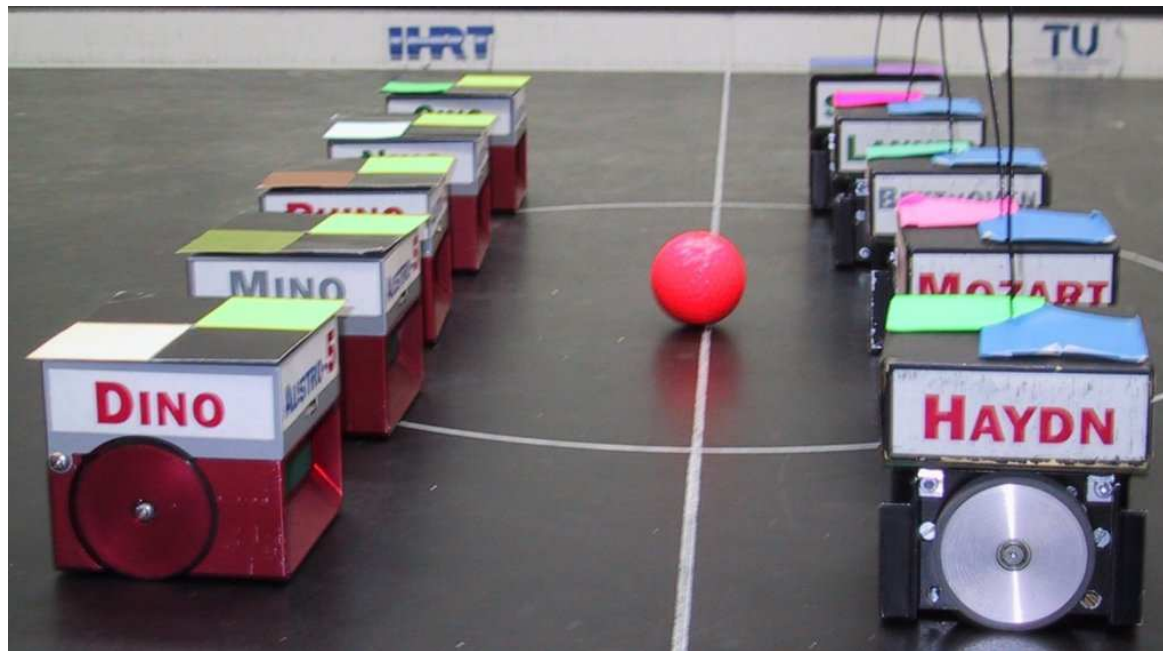
- Na akých princípoch fungujú obligátne spoločenstvá ?



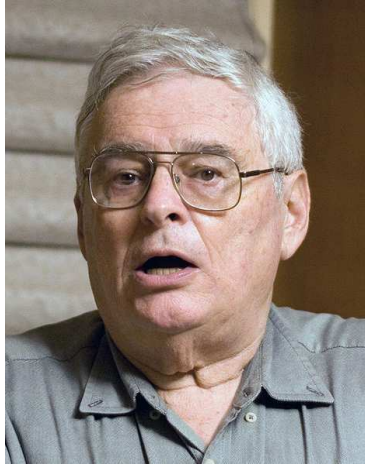
- vyše 30000 druhov mravcov
- spolupráca vyplývajúca z ich haplodiploidity
- jedinec nekoná rozumne, spoločenstvo áno
- mravce poznajú poľnohospodárstvo, chov dobytky, vojny i otroctvo a to už desiatky miliónov rokov

Ukážka MAS

- Typickou ukázkou MAS je napr. robosoccer: Aký program vložit' do jednotlivých robotov aby vyhrali zápas ?



Motivácia AOP



Fodorov model mysle:

- modulárnosť mysle

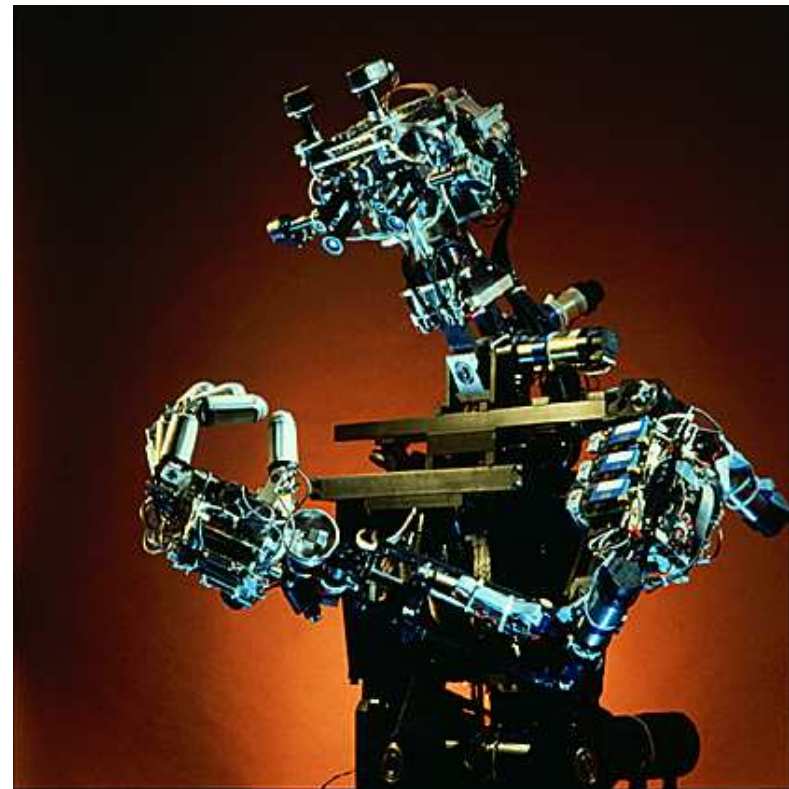


Minského societný model mysle:

- myseľ je zložená z agentov
- jej funkcia spočíva v správnej aktivácii časti agentov v správny okamih

Ukážka AOP

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

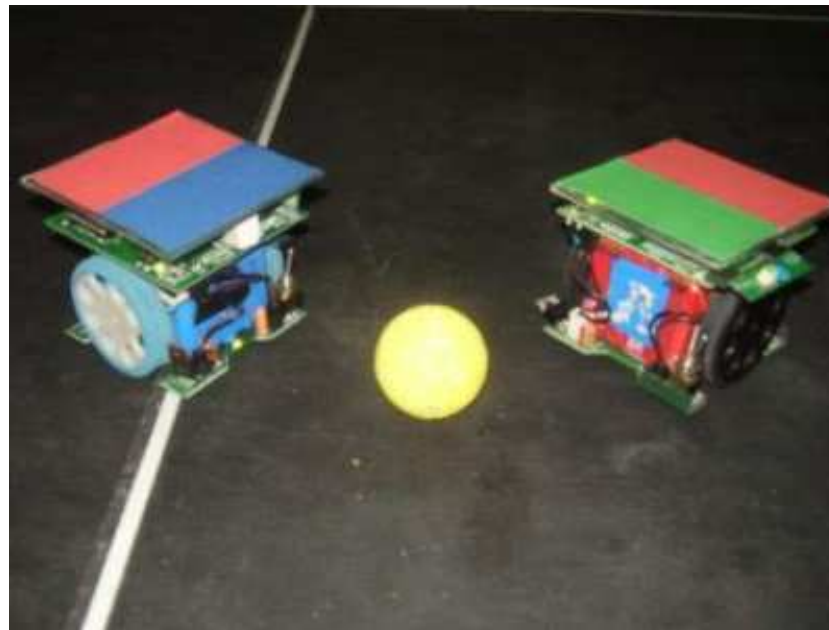


MAS/AOP sú oblasti zaoberajúce sa systémami, ktoré sa vyznačujú špecifickým druhom modularity

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)

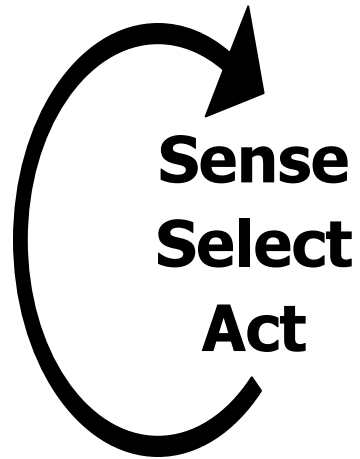
Povaha agenta

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

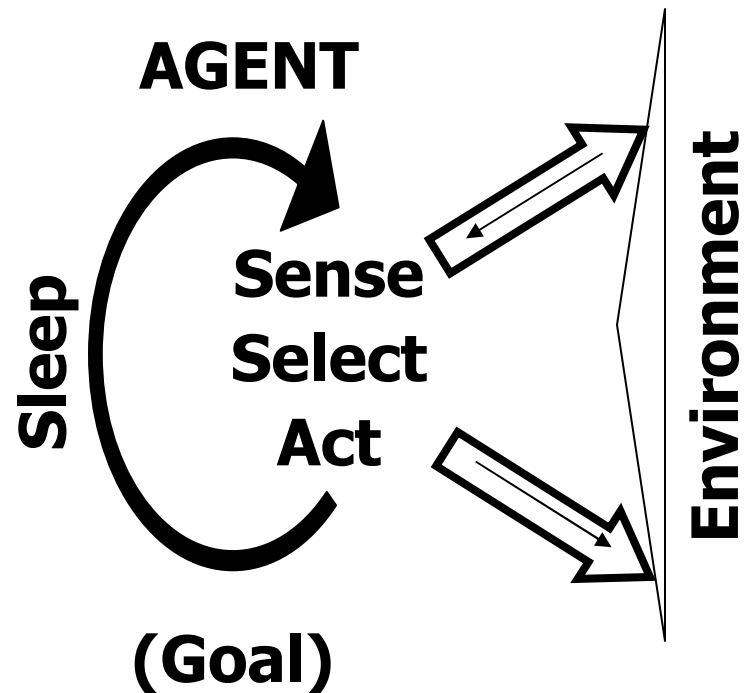


Povaha agenta

- Do robotov budeme vkladat' program tvaru:



Agent - proces



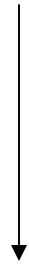
Agent je proces, ktorý neustále (opakovane) vníma svoje prostredie a na základe toho volí a vykonáva v ňom akcie, pričom sa jeho počínaniu dá pripísať určitý cieľ

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

Homogenita a heterogenita

- Nie vždy musia byť moduly v MAS rovnocenné (homogénne)
- Napríklad do robotického futbalového tímu by sme dali útočníkov, obrancov a brankára alebo dokonca mirosotov a kephery a aibo im by robil rozhodcu

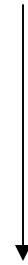
MAS



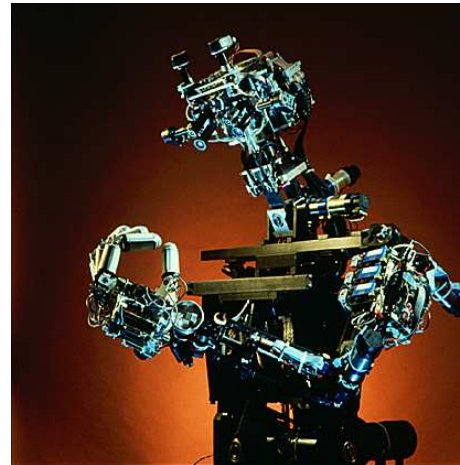
Network is
a computer



AOP



Computer is
a network

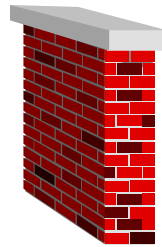


Prenos reálneho do virtuálneho

- Na AOP sa dá nazerat' ako na nový spôsob sťahovania objektov reálneho sveta do počítača.
- Historicky existujú tri takéto spôsoby a môžeme ich klasifikovat' na základe typu aktivity jednotky, ktorá v počítači reálny objekt predstavuje

Typy aktivity

- **pasívna entita**



(record)

- **reaktívna entita**



(object)

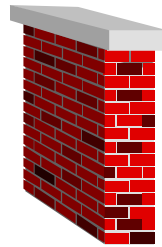
- **proaktívna entita**



(agent)

Typy aktivity

- každú aktivitu musíme iniciovať my



- entita dokáže iniciovať inú, ak ju iniciujeme



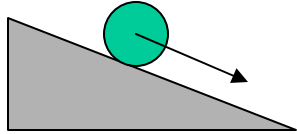
- entity nemusíme ani nemôžeme iniciovať



- štruktúrované programovanie

- objektovo-orientované programovanie

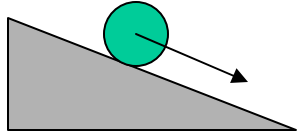
- agentovo-orientované programovanie



Neštruktúrované programovanie

```
#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);  
    }  
}
```

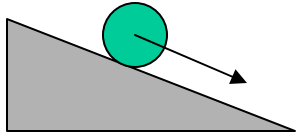


Štruktúrované programovanie

```
#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);
    }
}
```



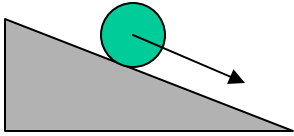
OOP

```
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 gulka =
        new Gulka(1.0,5.0);
    sleep(10);
    delete gulka;
}
```

AOP a UI

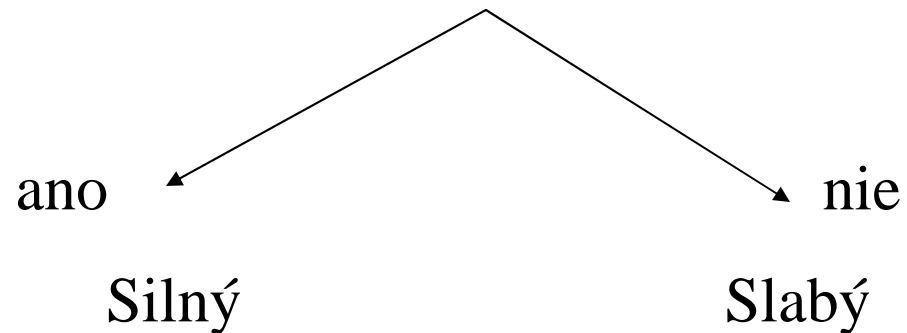
- AOP je nástrojom, ktorý prekonáva možnosti použitia MAS pre DUI a vo všeobecnosti umožňuje zvýšiť komplexnosť systémov a obohatiť ich o prvky neurčitosti
- Tým pádom sa AOP stáva zaujímavým technickým prostriedkom pre celú UI, pokrývajúc úlohy ako modelovanie živých systémov.

AOP a OOP

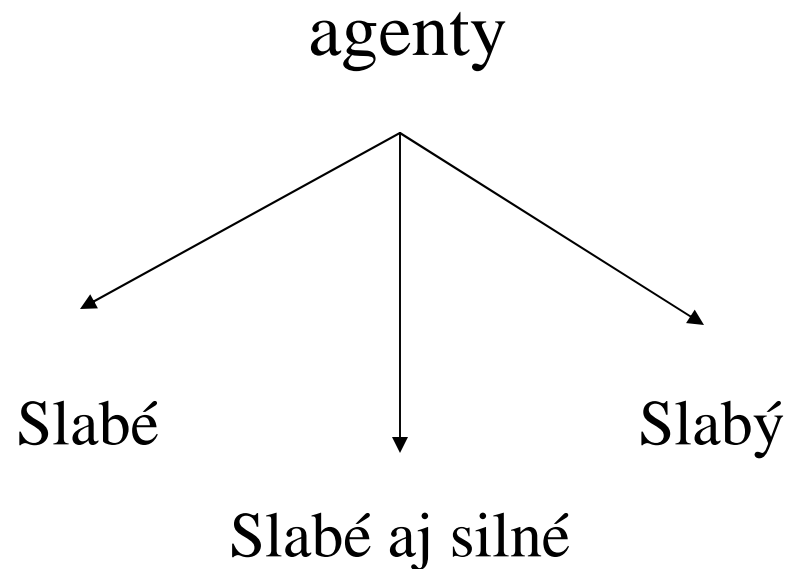
- Na rozdiel od iných prostriedkov UI, AOP vychádza z tradičného programovania
- Oproti tradičnému programovaniu sa však vyznačuje neurčitost'ami, obrazne povedané systémy ním spravené si žijú vlastným životom

„Slabý“ a „silný“ agent

Inteligencia je
zabezpečená špeciálnou
súčiastkou v agentovi

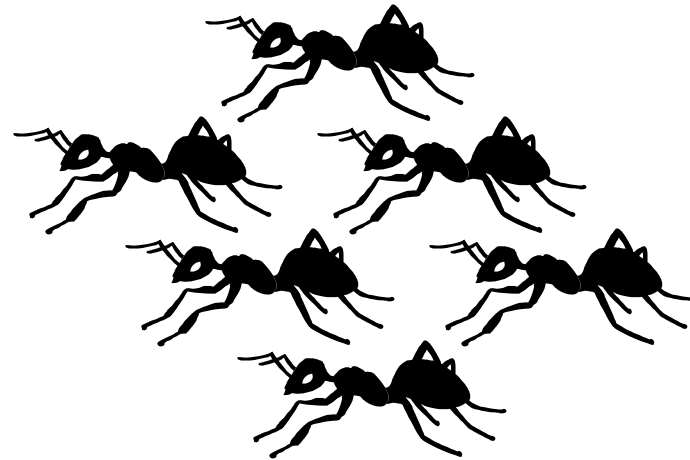
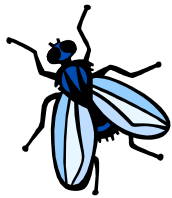


„Slabé“, „silné“ a „hybridné“ MAS (AOP)



Emergencia

Dá sa urobiť inteligentný systém zo slabých agentov ?



Dosahovať novú kvalitu kvantitou alebo šikovným
poskladaním starého