

Od medzimodulových spojení k nepriamej komunikácii medzi agentami

Andrej Lúčny

Katedra aplikovanej informatiky FMFI UK

andy@microstep-mis.com

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

Od medzimodulových spojení k nepriamej komunikácii medzi agentami

- Multiagentové technológie
- Ich aplikovateľnosť pre nedistribúované systémy
- Tradičná architektúra riadiacich systémov
- Problémy tradičnej architektúry
- Alternatívna architektúra založená na multiagentovej technológii
- Porovnanie tradičnej a alternatívnej architektúry
- Ďalšie možnosti

Multiagentové technológie

- Od polovice 90-tych rokov prebieha výskum typu „technology push“ uplatňujúci rôzne technologické varianty metafory, ktorá vychádza z distribuovaných systémov
- Jej kľúčovým pojmom je agent – entita, ktorá opakovane vníma svoje prostredie a na základe toho volí akcie, ktoré v ňom vykonáva, pričom sa tejto činnosti dá v rámci systému pripísať určitý cieľ.

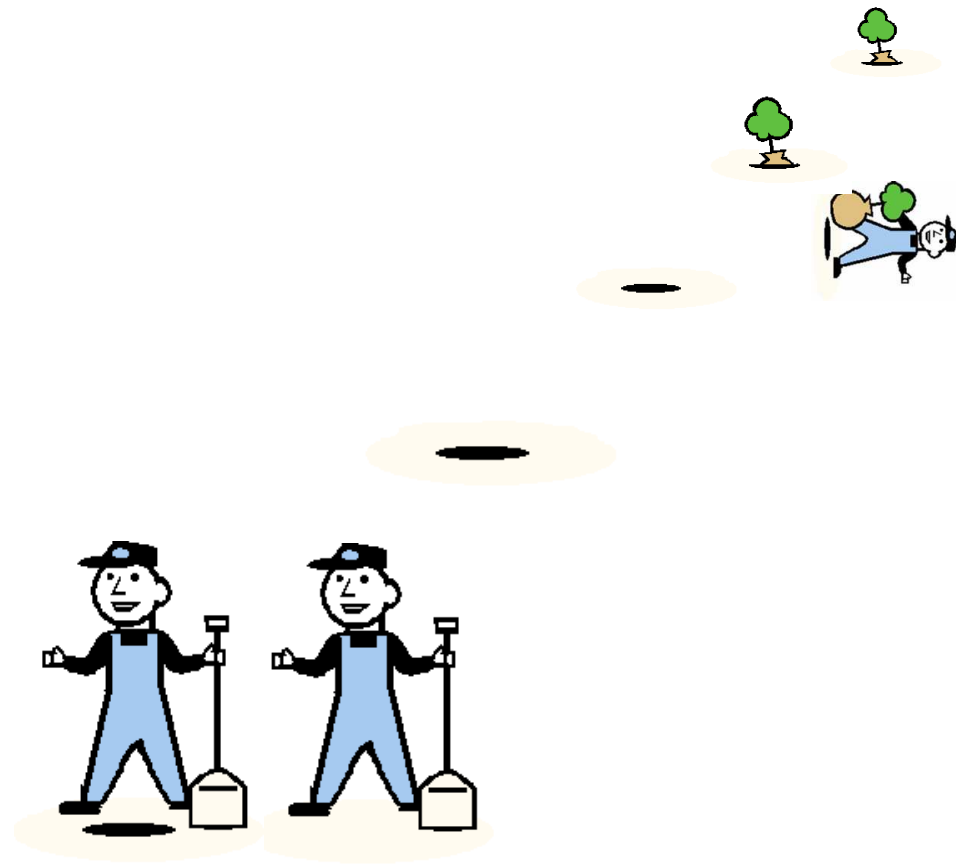
- Klasickou ukázkou multiagentového systému je robotický fotbal



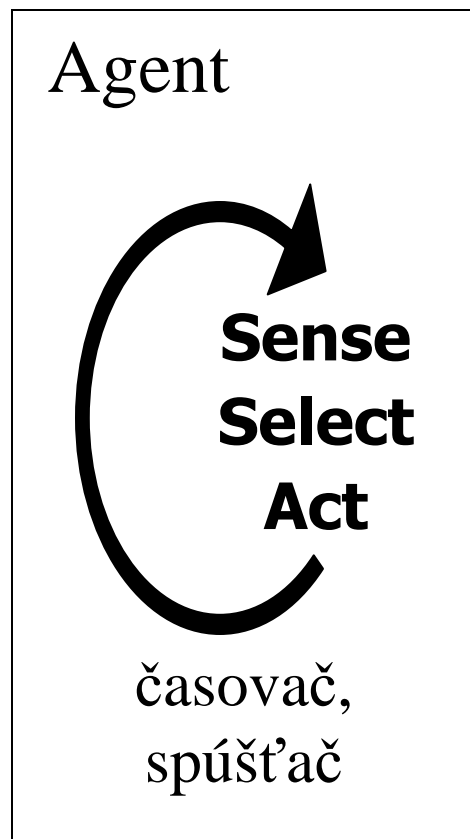
- Je obtiažne napísať program, ktorý manipuluje celým tímom. Oveľa ľahšie je nakódovať riadenie každého hráča zvlášť a hru tímu nechať povstať z interakcie medzi hráčmi, z ktorých každý hrá „sám o sebe“

Špecifiká agentovej modularity

- autonómnosť modulov
- proaktivita modulov
- decentralizácia systému



Povaha agentov

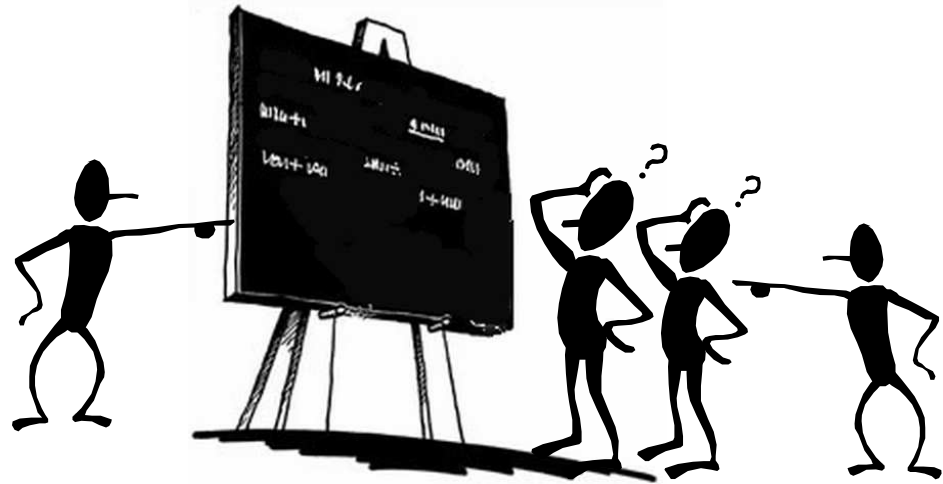
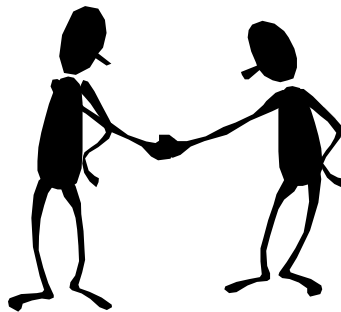


voľba akcie je zabezpečená
„inteligentnou“ súčiastkou v
agentovi
(tzv. **silné**, inteligentné či
deliberatívne agenty)

voľba akcie je zabezpečená
bežným kódom
(tzv. **slabé**, reaktívne agenty)

Komunikácia medzi agentami

- agenti môžu vzájomne komunikovať a to buď priamo, alebo nepriamo cez nejakú „čiernu tabuľu“ (blackboard, space)



Aplikačné oblasti

Uplatnenie rozbitia určitého celku na agentov možno nájsť na rôznych úrovniach:

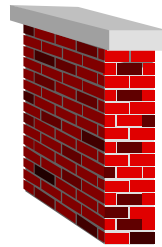
- WAN, napr. agentami nahradzujeme ľudí poskytujúcich a odberajúcich určitú službu v Internete
- LAN, napr. agentami nahradzujeme ľudskú obsluhu strojov v továrni
- VM, napr. agentami nahradzujeme moduly mysle, ktorými implementujeme riadenie stroja

„Počítač je sieť“

- Metódy vyvinuté primárne pre distribuované systémy (WAN a LAN programovanie) je možné aplikovať i pre modulárne systémy, ktoré distribuované nie sú (VM)
- Podstatný je typ modularity a nie konkrétna inštancia tohto typu v konkrétnej aplikácii

Vývoj programátorských štruktúr

- pasívna entita



záznam

- reaktívna entita



objekt

- proaktívna entita



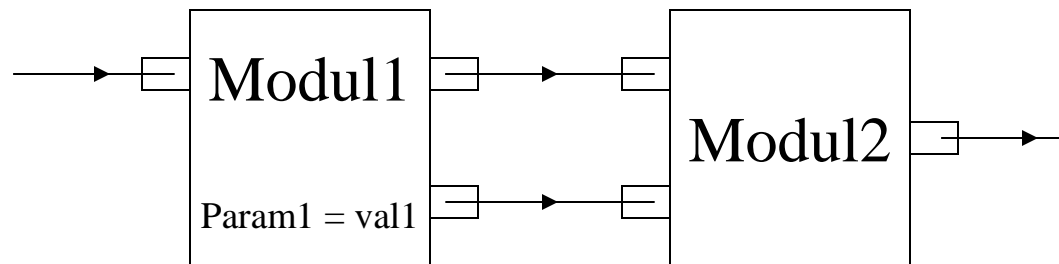
agent

Príklad použitia

- Uvedieme tu práve jednu z aplikácií MAS na úrovni VM, konkrétne pre tvorbu riadiacich systémov (napríklad mobilných robotov)
- Je založená na prechode od tradičnej architektúry, ktorej modularita (softwaru) kopíruje usporiadanie hardware, k alternatívnej architektúre, ktorej modularita je multiagentová
- (Mohli by sme softwarové moduly organizovať lepšie než ako keď napodobňujeme hardware?)

Tradičné riešenie

- riadenie sa skladá z modulov s pevne definovanými vstupmi a výstupmi a nastaviteľnými parametrami
- skladanie systému sa realizuje vhodným prepojením vstupov a výstupov



- vykonávanie kódu, ktorý v module prepočíta vstupy na výstupy má na starosti plánovač

Motivačný príklad

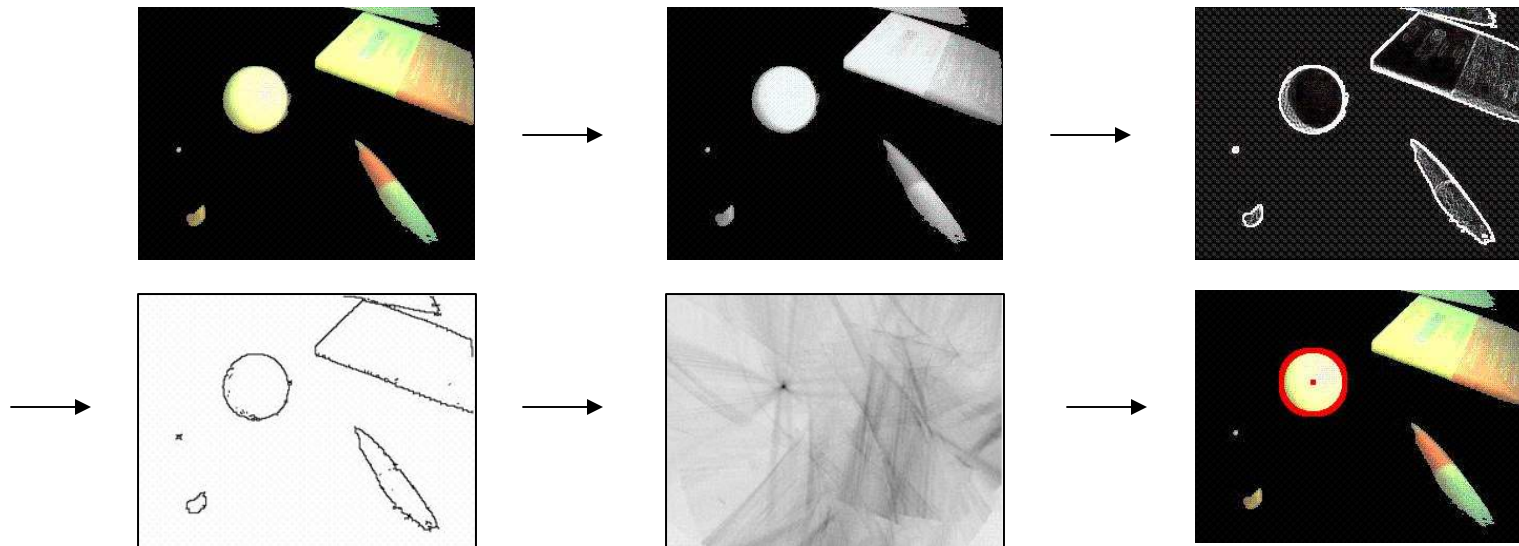
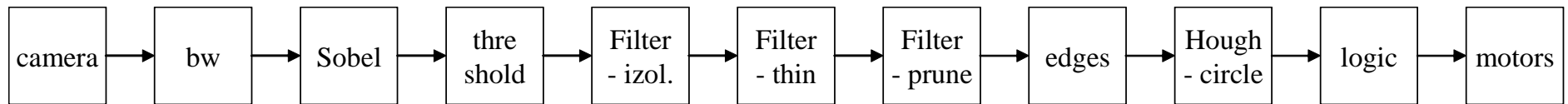
- zobrali sme jednoduchý príklad:

mobilného robota,
ktorý sleduje
pingpongovú
loptičku



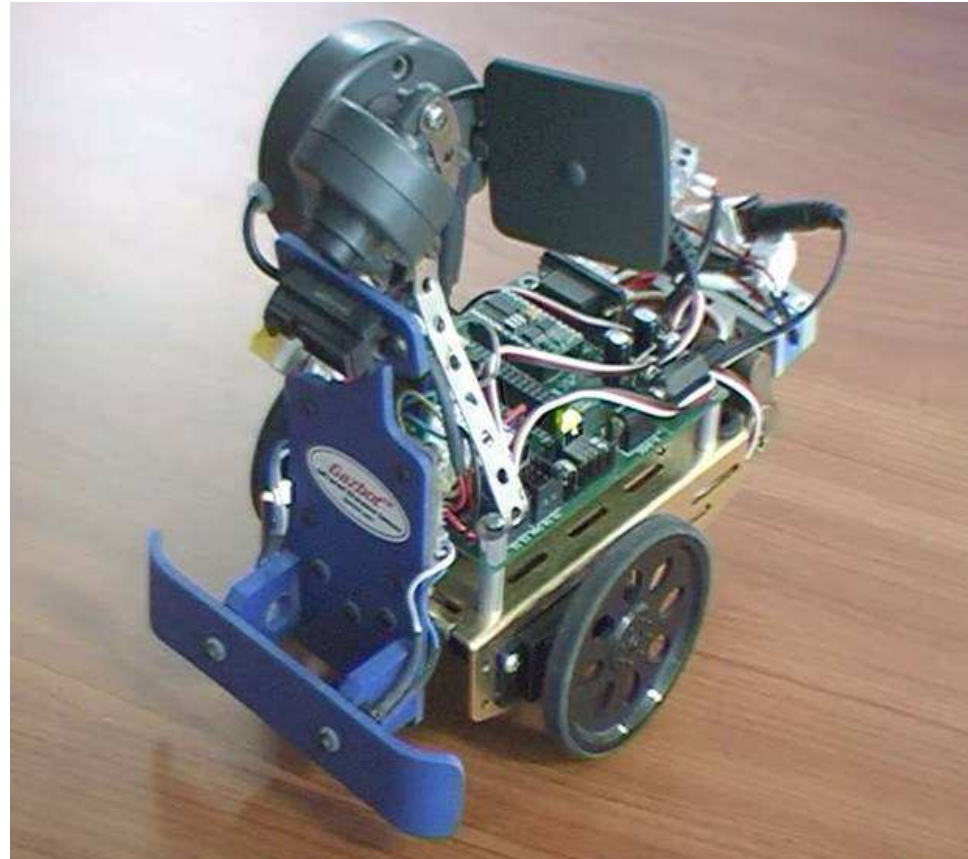
Motivačný príklad

- Ako-tak fungujúce riešenie sa tu dá založiť na jednoduchom pipeline



Motivačný príklad

- postupne sme kládli vyššie a vyššie nároky na kvalitu
- a pozorovali kde sa tradičná architektúra dostáva do problémov



Motivačný príklad

Konkrétne sme riešili:

- zvládnutie rôznej úrovne osvetlenia scény
- zvládnutie viacerých loptičiek v scéne
- zvládnutie sledovania loptičky, ktorá podlieha zákrytu
- zvládnutie aktívneho vyhľadávania loptičky, ktorú nevidno na obraze kamery (náhodný pohyb vyhýbajúci sa prekážkam)

Problémy tradičnej architektúry

- keď boli kombinované pomalé moduly s rýchlymi
- keď mali vstupy do modulu rôznu frekvenciu
- keď bolo treba dynamicky meniť počet vstupov a výstupov
- keď bolo treba časovo ohraničiť platnosť nejakého údaju
- keď bolo treba zabezpečiť netriviálnu koordináciu medzi paralelnými subsystémami

Alternatívna architektúra

- spojenia medzi modulmi nahradíme za pomenované dáta na „čiernej tabuli“ (space), tzv. bloky, ktoré realizujú nepriamu komunikáciu medzi agentami
 - agenty ich môžu čítať, zapisovať a mazať na základe ich pomenovania
 - pri zápise môže agent pre blok definovať jeho časovú platnosť a prioritu

Alternatívna architektúra

- moduly nahradíme agentami, t.j.
 - odstránime plánovač prepočítavania vstupov na výstupy
 - tento prepočet vložíme do cyklu, ktorý blokujú vhodné časovače a spúšť'ače (citlivé na zmenu určitých blokov)
 - tomuto cyklu priradíme vlastné vlákno
 - vstupy a výstupy nahradíme čítaním a zapisovaním blokov

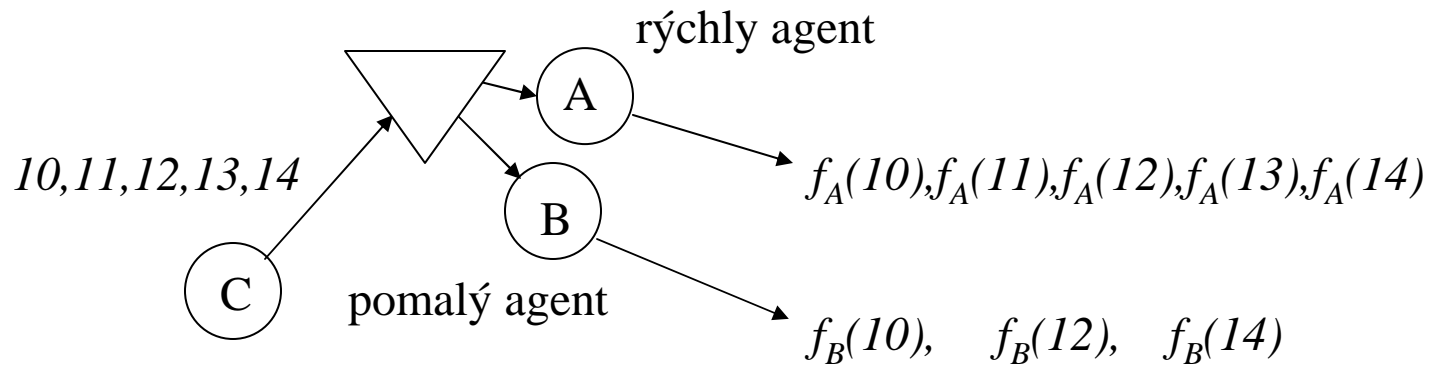
Alternatívna architektúra

Od jemných detailov manipulácie s blokmi záleží, či prekonáme nevýhody tradičnej architektúry. Je dôležité, aby:

- nebolo treba na vytvorenie bloku volať špeciálnu operáciu, ale aby vznikali zápisom
- aby mohli byť bloky prázdne
- agenti do nich radšej nič nezapísali, než zapísali prázdnu hodnotu
- aby bolo možné vykonávať hromadné čítanie blokov na základe masky (*, regex)

Implicitné vzorkovanie

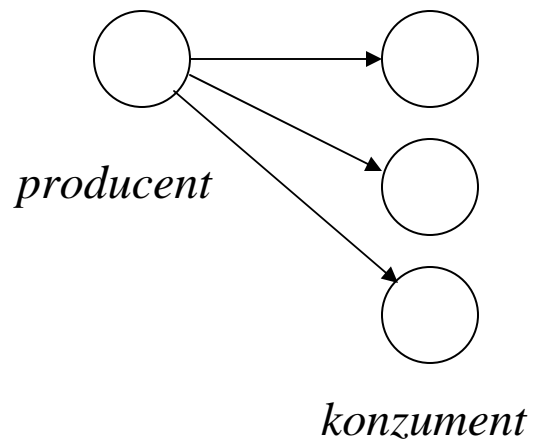
Keďže zápis bloku prepíše jeho hodnotu bez ohľadu na to, či ho jeho čitatelia stihli prečítať, každý dátový tok je automaticky navzorkovaný tak, aby sa vždy pracovalo s údajmi v reálnom čase



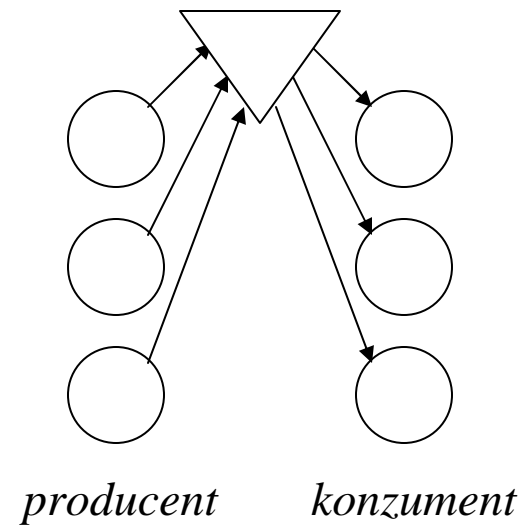
Dátový tok

“od mnohých k mnohým”

tradičný tok



alternatívny



Ukážka

```
package com.microstepmis.agentspace.demo;  
import com.microstepmis.agentspace.*;
```

```
public class Agent1 extends Agent {
```

```
    int i = 0;
```

```
    public void init(String[] args) {  
        attachTimer(1000);  
    }
```

```
    public void senseSelectAct() {  
        System.out.println("write: "+i);  
        write("a",i++);  
    }
```

```
}
```

```
public class Agent2 extends Agent {
```

```
    int i;
```

```
    public void init(String args[]) {  
        attachTrigger("a");  
    }
```

```
    public void senseSelectAct() {  
        i = (Integer) read("a",-1);  
        System.out.println("read "+i);  
    }
```

```
}
```

```
public class Starter {
```

```
    public static void main(String[] args) {
```

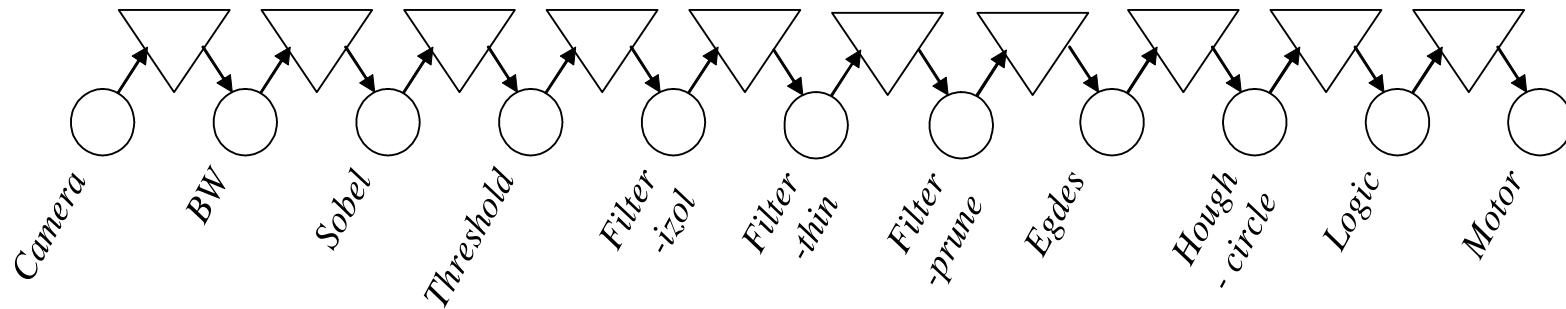
```
        new SchdProcess("space","com.microstepmis.agentspace.SpaceFactory",new String[]{"DATA"});  
        new SchdProcess("agent1","com.microstepmis.agentspace.demo.Agent1",new String[]{});  
        new SchdProcess("agent2","com.microstepmis.agentspace.demo.Agent2", new String[]{});
```

```
    }
```

```
}
```

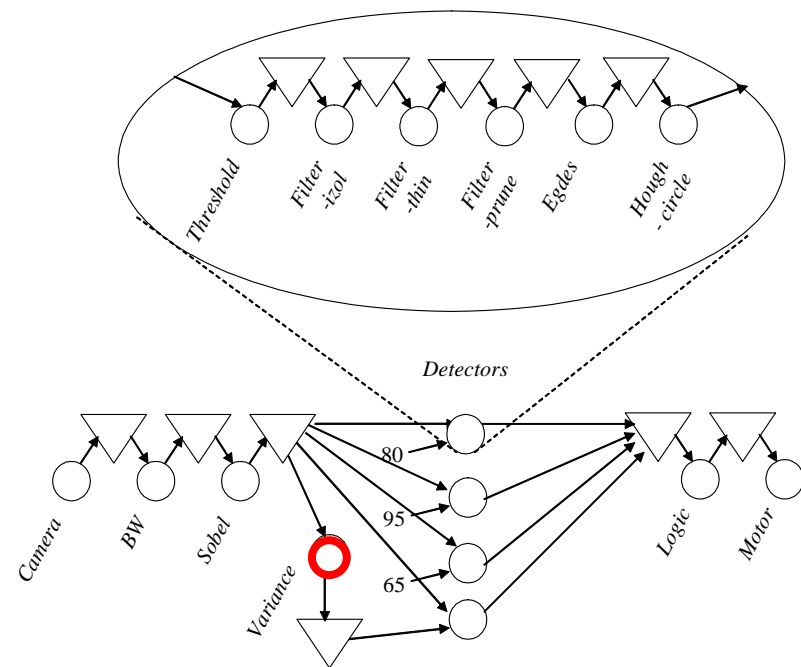
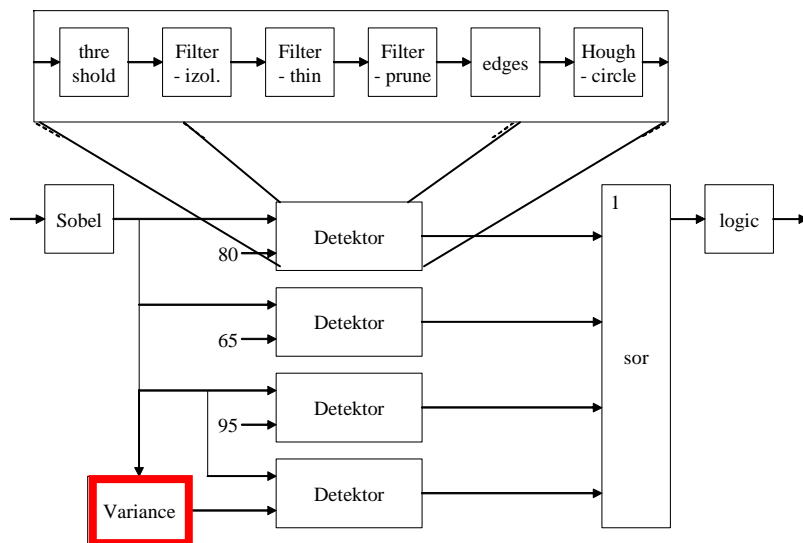
Porovnanie

Rôzne frekvencie vstupov



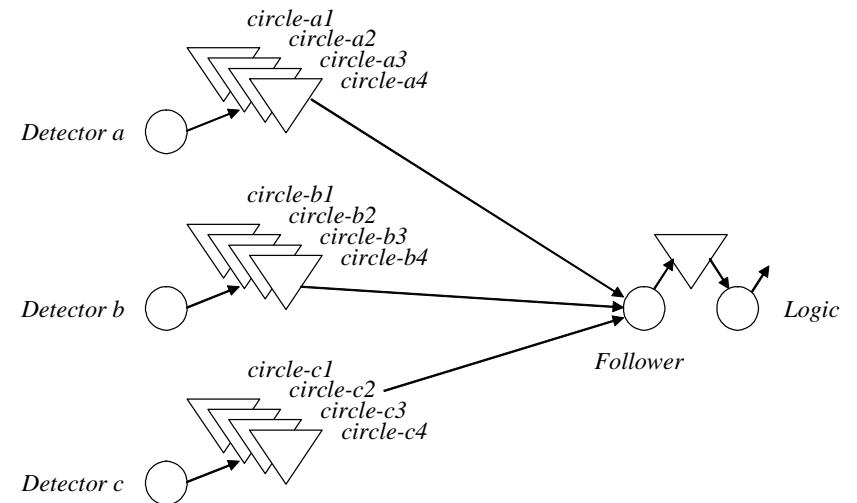
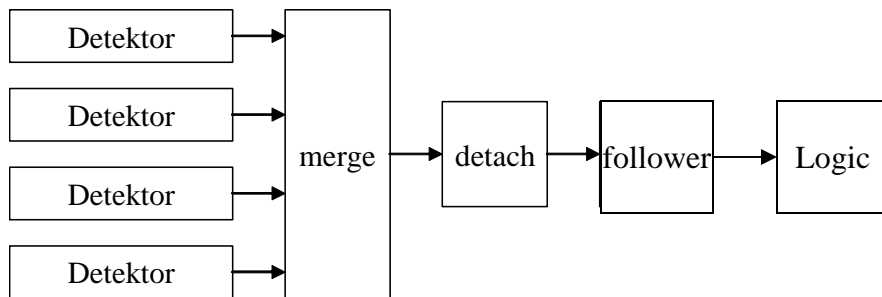
Porovnanie

Kombinácia pomalých a rýchlych modulov



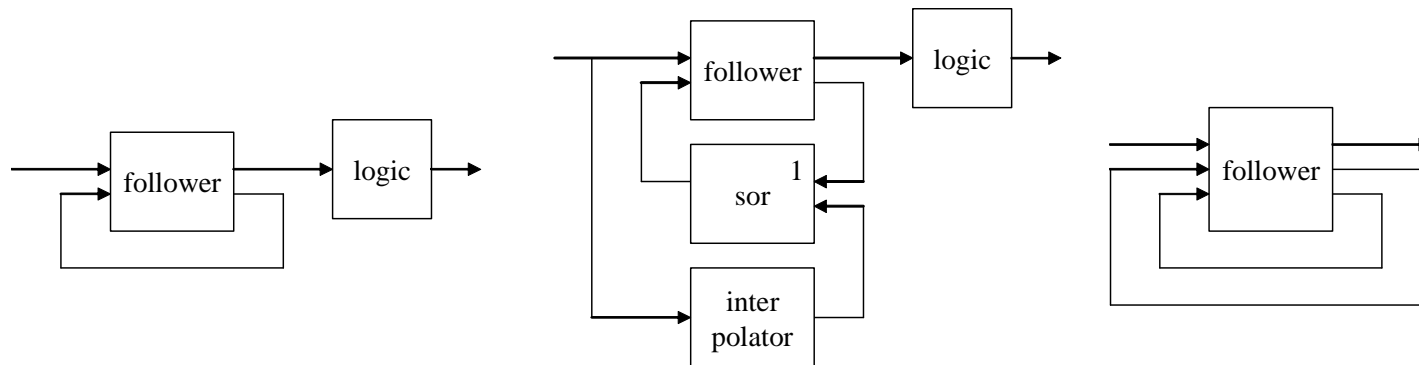
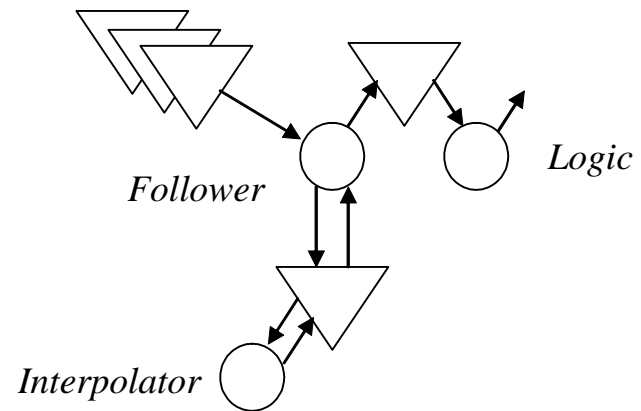
Porovnanie

Dynamická zmena počtu vstupov a výstupov



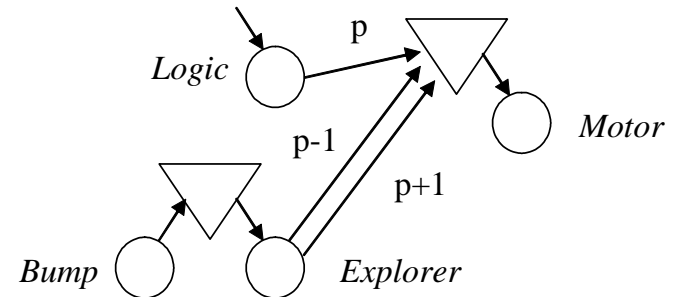
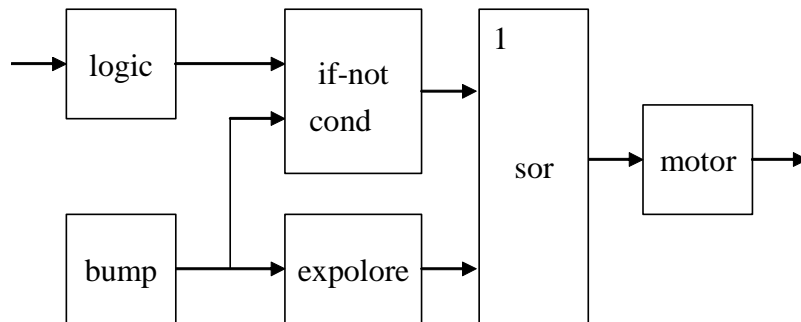
Porovnanie

Neprístupnosť vnútorného stavu a obmedzenie časovej platnosti údajov



Porovnanie

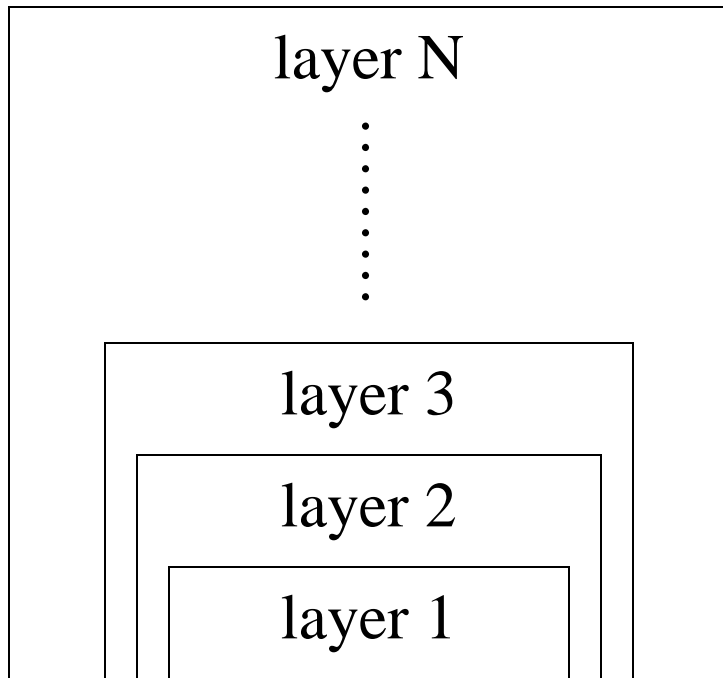
netriviálna koordinácia medzi paralelnými subsystémami



Výhody

- Vybraté problémy tradičnej architektúry vieme riešiť v alternatívnej architektúre šikovnejšie
- Poskytuje nám táto architektúra aj nejaké ďalšie výhody?
- Dáva nám nejaké ďalšie podnety pre zdokonaľovanie programovania?

Subsumpcia



nepriama
komunikácia
podporuje aby
„evolučne novšie“
vrstvy systému
ovplyňovali
„evolučne staršie“
vrstvy

Kombinovanie silných a slabých agentov

- „agentová“ povaha systému umožňuje prirodzené zakomponovanie tradičných techník umelej inteligencie (na báze matematickej logiky) do systému
- (od určitých hierarchicky vyšších vrstiev začneme používať „silné“ agenty)
- (tým pádom používame „silné“ agenty len tam, kde je to vhodné)

Vplyv MAS na programovacie jazyky

- Vrastanie komunikácie medzi agentami do jazyka
- Čo je to premenná, čo to znamená priradiť jej hodnotu ?

`var = val;` `var = val for 1s at priority 0.5;`

- Ako jedna časť kódu volá druhú ?

`obj.met(arg);` `obj.attr = val;` `delay(1s);`

Záver

- Ukázali sme jedno z použití multiagentovej „paradigmy“
- Netradičné v tom, že nešlo o distribuovaný systém
- Zaujímavé tým, že poskytuje určité výhody a dáva d'alekosiahle podnety pre programovanie ako také
- Dúfame, že aplikácie tohto typu modularity budú stále populárnejšie

Ďakujem za pozornosť

Od medzimodulových spojení k
nepriamej komunikácii medzi
agentami

Andrej Lúčny

Katedra aplikovanej informatiky FMFI UK

andy@microstep-mis.com

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