Programovacie jazyky pre vývoj inteligentných agentov

(BDI architektúra)

Peter Novák

Computational Intelligence Group
Clausthal University of Technology
Nemecko

3. Október 2006

Agenda

1. Predstavenie
   - Kto? Kde? Prečo?
2. Motivácia
   - Inteligentné agenty a BDI
   - AOP
   - AOSE
   - Programovacie jazyky
3. Jazyky - súčasný stav
   - Pragmatický prístup
   - Teoreticky orientované jazyky
   - Porovnanie
   - Modulárna BDI architektúra
4. Zhrnutie a záver


Kiel
Schleswig-Holstein
Mecklenburg-Vorpommern
Schwerin
Hamburg
Bremen
Brandenburg
Niedersachsen
Berlin
Hannover
Potsdam
Magdeburg
Clausthal-Zellerfeld 51°48' N
Sachsen-Anhalt
Nordrhein-Westfalen 600 m 10°20' E
Düsseldorf
Dresden
Erfurt
Sachsen
Thüringen
Hessen
Rheinland-Pfalz
Wiesbaden
Mainz
Saarland
Saarbrücken
Bayern
Stuttgart
Baden-Württemberg
München

Prečo to všetko?

Erasmus/Sokrates kooperácia medzi KAI FMFI UK a IfI TUC

- Jún 2005 - návšteva skupiny J. Šefránka v CLZ
- leto/jeseň 2005 - zmluva o Erasmus/Sokrates spolupráci
- September/Október 2006 - ja na FMFI UK
- zimný semester 2006/2007 - 4 študenti z FMFI UK v CLZ
- budúcnosť - žiarivá !!! ☺

Kto sme a čo robíme

Skupina výpočtovej inteligencie: **Prof. Jürgen Dix**
+ medzinárodná zostava (2xDE, 1xSK, 1xPL, 1xMX)

Čo robíme?

- Logické programovanie
  - Nemotónne usudzovanie
  - Answer Set Programming
  - deduktívne databázy
- Multi-agentové systémy
  - logiky stratégii pre MAS
  - programovanie (multi-)agentových systémov, verifikácia
  - výpočtová logika pre (multi-)agentové systémy
  - CLIMA Contest

Programovacie jazyky pre vývoj inteligentných agentov

(Inteligentný?) agent

Definition

Agent $\rightsquigarrow$ autonómna výpočtová entita, ktorá koná v prostredí a interaguje s ním, prípadne s inými agentami.
Multiagentový systém $\rightsquigarrow$ systém pozostávajúci z množstva interagujúcich agentov.

- alternatíva: agent je akýkoľvek systém, ktorému pripisujeme antropomorfné vlastnosti ako *autonómnosť*, či *proaktívnosť*
- metafora na modelovanie reality
  - modelovanie inteligentného správania
- abstrakcia použiteľná na vývoj umelých systémov

Belief Desire Intention

Bratman (1987) - teória intencií
Rao & Georgeff (1991) - výpočtový model, AgentSpeak(L)

architektúra agenta = (znalosti + ciele + zámery) + interakcie

- **znalosti**: model sveta
- **ciele:** popisy situácii, ktoré by agent rád dosiahol
- **zámery:** situácie a plány, ktoré agent práve sleduje
- **interakcie** (model racionality): formálny popis "toku informácie" v agentovom systéme

... veľa voľnosti

I-Systém

Systém axióm regulujúci interakcie BDI komponentov (integritné obmedzenia):

1. agent prijíma len splniteľné ciele
2. agent prijíma len zámery/plány korešpondujúce s cieľmi
3. každá naplánovaná akcia bude aj niekedy vykonaná
4. agent vie o svojich zámeroch a cieľoch
5. ak má agent istý zámer, má aj cieľ mať tento zámer
6. ak agent vykonal akciu, vie o tom - história
7. žiadny zámer nemá agent nekonečne dlho

Agentovo Orientované Programovanie

Yoav Shoham, 1991

- programovacia paradigma: *spoločenský* pohľad na výpočtové systémy
- v centre je *mentálny stav* agenta = znalosti, názory, schopnosti, ciele, zámery, roly
- navzájom *komunikujúci* agenti
- *inžiniersky pohľad*: ako programovať (multi-)agentové systémy?

- podobnosti AOP vs. OOP
  - objekt - agent
  - dáta - mentálny stav
  - metódy - správy
  - rečové akty, normatívne obmedzenia

Agentovo Orientované Softwérové Inžinierstvo

Ako *vyvíjať* multiagentové systémy?

- ucelené metodológie na vývoj MAS
  - od požiadaviek na systém, cez analýzu k dizajnu
- organizácia, interakcie
- ciele, roly, správania
- priemyselné aplikácie!

BDI orientované metodológie:

- Gaia
- Tropos
- Prometheus

Programovacie jazyky pre agenty

Krok od výstupov analýzy a dizajnu k realizácii systému.

Analýza MAS

- typy agentov
- roly
- ciele
- správania
- protokoly, špecifikácia interakcií

Implementácia (softwér)

- podpora AOP na úrovni syntaxe jazyka
- modularita
- interoperabilita
- integrácia s existujúcimi systémami

Programovacie jazyky pre BDI agenty

- pragmatické, prakticky orientované jazyky
  - JACK
  - Jadex
- teoreticky orientované
  - 3APL
  - AgentSpeak(L)
  - GOAL
  - ...

Pekné teoretické vlastnosti vs. IT pragmatizmus

JACK, Jadex

JACK: *Agent Oriented Software Group, Austrália*
Syntaktické rozšírenie Javy o koncepty umožňujúce implementáciu agentov, akcií, udalostí, plánov, báz znalostí

kompilácia JACK programu do Java kódu $\rightsquigarrow$ štandardná Java aplikácia

Jadex: *Univerzita Hamburg, Nemecko*
Sada knižníc a agentový cyklus implementovaný v priamo v Jave

- BDI systém nad Jade platformou
- znalosti, ciele a plány agenta v špecializovanom formáte ADF (XML).
- vybudované nad Jade middleware platformou $\rightsquigarrow$ FIPA kompatibilné

parametrizácia Jadex agentového cyklu

JACK - syntax

```
#private data BeliefType belief_name(arg_list)


plan PlanName extends Plan {
#handles event EventType event_ref
body() { }
}


@action(parameters) <body>


@maintain(logical_condition, event)
```

AgentSpeak(L), 3APL a spol.

AgentSpeak(L): *Rao & Georgeff 1991*
platforma: *Jason (Bordini et. al., Uni Durham UK ,Uni Blumenau, BR)*

- báza znalostí, udalostí, zámerov a knižnica plánov (pravidiel)
- `<udalost' >:<podmienka>⟵(<plán> | <udalost'>)*`

3APL: *Hindriks et. al. 1998*
platforma: *3APL (Dastani et. al., Uni Utrecht, NL)*

- báza znalostí, cieľov, zámerov a knižnica plánov (pravidiel)
- `(<ciel'>|<plán>)⟵<podmienka> "|" (<plán> | <ciel'>)*`

- jednoduchý agentový cyklus, parametrizácia ⇝ Java kód
- operačná sémantika ⇝ prechodový systém nad mentálnymi stavmi agenta

3APL - príklad

```
PROGRAM "agent"
BELIEFBASE { location(r1,2,4).  }
CAPABILITIES { {dirty(R)} Clean(R) {NOT dirty(R)} }
GOALBASE { cleanRoom(r1).  }
PLANBASE { SetStatus(started); }
PG-RULES {
cleanRoom(R) <- dirty(R) |
     { SetStatus(cleaning(R)); goTo(R); clean(R);
       SetStatus(standby); }
}
PR-RULES {
clean(R) <- location(R,X,Y) AND at(X,Y) |
     { Clean(R); }
}
```

Porovnanie tried BDI jazykov

BDI programovacie systémy

Inžinierske prístupy (JACK, Jadex)

- ⊞ vrstva špecializovaných konštruktov nad robustným programovacím jazykom (Java) ⇝ znovupoužiteľnosť kódu a knižníc
- ⊟ sémantika hostiteľského jazyka
- ⊟ reprezentácia znalostí v imperatívnom (OO) jazyku
- ⊞ ľahká integrácia s externými systémami a vonkajším prostredím

Teoreticky orientované (AgentSpeak(L), 3APL)

- ⊟ deklaratívne jazyky vybudované od základu ⇝ nová syntax
- ⊞ jasné teoretické vlastnosti ⇝ verifikácia
- ⊞ deklaratívne techniky reprezentácie znalostí (relatívne slabé usudzovanie)
- ⊟ žiadna integrácia s externými a existujúcimi systémami

Modulárna BDI architektúra

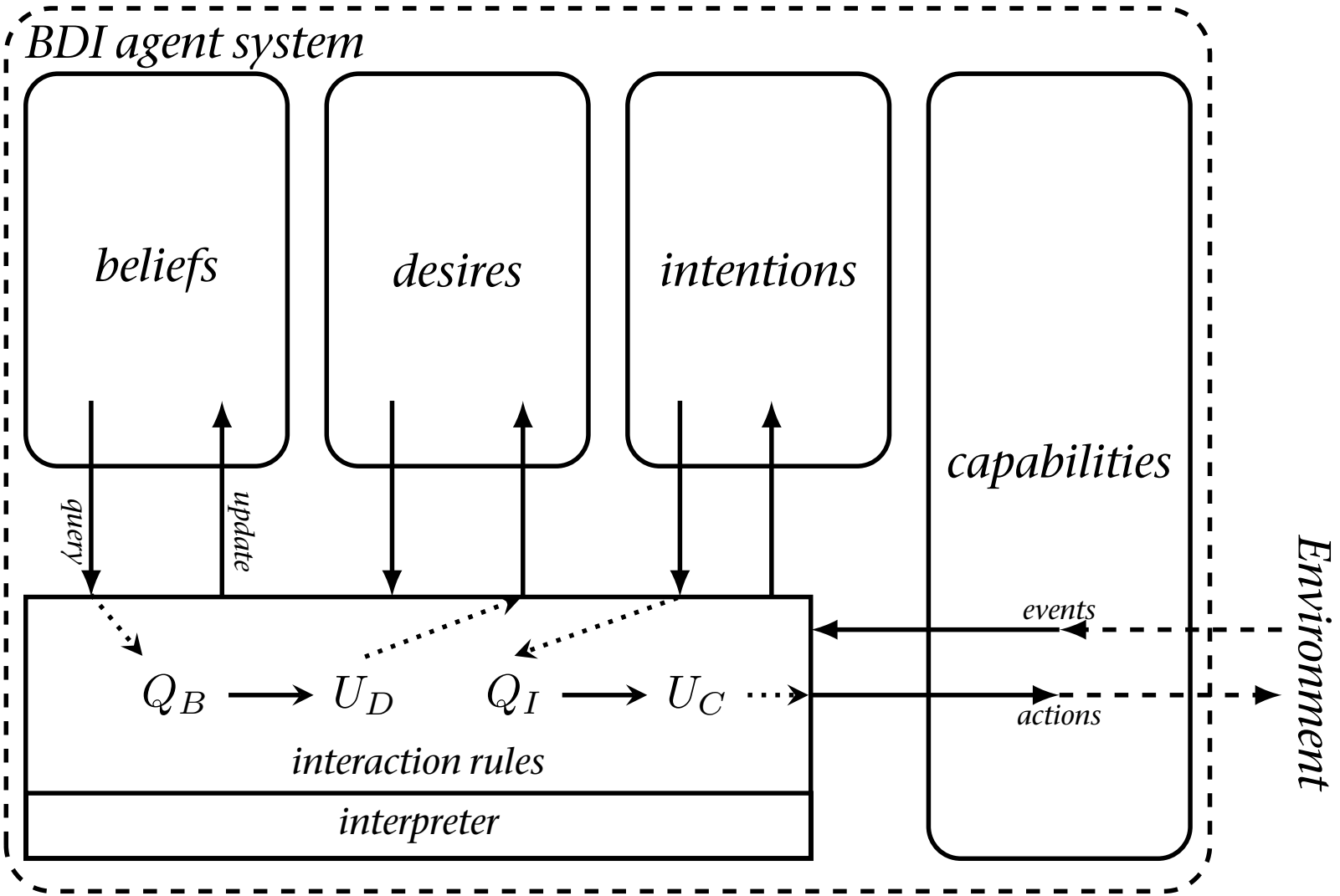
Reprezentácia znalostí:

- zapuzdrime BDI moduly a povolíme iba query/update interfejs
- techniky reprezentácie znalostí a použité programovacie jazyky $\rightsquigarrow$ rozhodnutie programátora
- schopnosti agenta (akcie) $\rightsquigarrow$ ďalší BDI komponent

Dynamika agenta:

- interakcie medzi BDI modulmi $\rightsquigarrow$ interakčné pravidlá
- aplikácia interakčného pravidla $\rightsquigarrow$ atomický prechod

Architektúra



Príklad

Beliefs (Prolog)

```
ready :- cup_present,
         cup_empty,
         not error.
```

Desires (set of Prolog atoms)

```
make_espresso.
```

Intentions (stack - Lisp)

```
(define push ...)
(define pop ...)
(define top? ...)
```

Capabilities (C)

```
void mill_start();
void mill_stop();
int stand_empty();
int cup_empty();
```

$$Q_C(\text{!stand\_empty() \&\& cup\_empty()}) \longrightarrow U_B(\text{assert(cup\_present)})$$

$$Q_B(\text{ready}) \wedge Q_D(\text{make\_espresso}) \longrightarrow U_I(\text{(push (grind boil pour clean))})$$

$$Q_I(\text{(top? grind)}) \longrightarrow U_C(\text{mill\_start()}) \circ U_I(\text{(pop)})$$

Teoreticky praktický prístup

Pekné vlastnosti:

- jasná sémantika
- *modularita* - jednoduché znovu-použitie kódu
- jednoduchšia *integrácia* s externými a existujúcimi systémami ⇝ *agentizácia* existujúcich systémov

Nepekné vlastnosti:

- znovu nová syntax ⇝ hľadanie *minimálneho* syntaktického jadra (nutné zlo?)
- interpreter a konkrétna syntax ⇝ súčasná práca
- žiadny model racionality ⇝ štýl programovania, podpora v jazyku
- praktickosť (???)

Záver

Rôzne techniky reprezentácie znalostí sú vhodné na rôzne aplikácie!

Ďakujem za pozornosť.

Otázky?