Programming framework for cognitive agents

(motivation & overview)

Peter Novák

Computational Intelligence Group
IfI @ TUC

February 22, 2007

Outline

1. Motivation
   - Scenarios
   - Problem
2. State of the art
3. Jazyk project
   - Modular BDI architecture
   - Jazyk: The language
   - Higher level programming constructs
   - Discussion
4. Conclusion

Motivation scenarios (Robot contests)

RoboCup Rescue League

- team of agents navigating in a fairly complex map
- several types of agents
- limited communication resources

RoboCup Four-Legged League

- 2 teams of 4 robots playing soccer

AAAI Robot competition: Integration challenge

Integrate existing components to produce a working robot that is:

- robust, fault-tolerant, flexible, easily adaptable to new tasks

Motivation scenarios (Robot contests)

RoboCup Rescue League

- team of agents navigating in a fairly complex map
- several types of agents
- limited communication resources

RoboCup Four-Legged League

- 2 teams of 4 robots playing soccer

AAAI Robot competition: Integration challenge

Integrate existing components to produce a working robot that is:

- robust, fault-tolerant, flexible, easily adaptable to new tasks

Motivation scenarios (Robot contests)

RoboCup Rescue League

- team of agents navigating in a fairly complex map
- several types of agents
- limited communication resources

RoboCup Four-Legged League

- 2 teams of 4 robots playing soccer

AAAI Robot competition: Integration challenge

Integrate existing components to produce a working robot that is:

- robust, fault-tolerant, flexible, easily adaptable to new tasks

Knowledge manipulating autonomous agents

Agent (working definition)

Software entitity *embodied* in an environment, which acts *autonomously* and proactively in order to reach its *goals*.

Agent with mental states

- builds a ***model*** of its environment
- explicitely uses ***mental attitudes*** ⇝ keeps track of goals, its decisions and contexts it is currently in

⇝ Hybrid cognitive robotic architectures: e.g. **BDI.**

Knowledge manipulating autonomous agents

Agent (working definition)

Software entitity *embodied* in an environment, which acts *autonomously* and proactively in order to reach its *goals*.

Agent with mental states

- builds a ***model*** of its environment
- explicitely uses ***mental attitudes*** $\leadsto$ keeps track of goals, its decisions and contexts it is currently in

$\leadsto$ Hybrid cognitive robotic architectures: e.g. **BDI.**

Challenges

1. reactiveness vs. mental states (deliberation)
2. knowledge representation modularity

Problem

Develop a BDI based programming system for development of agents with mental states:

- architecture
- programming language
- methodology

State of the art

BDI based programming systems

Engineering approaches (JACK, Jadex)

- + layer of specialized constructs over Java ⇝ easy code re-use, vast number of 3rd party libraries
- + easy integration with external systems/environment
- − semantics of the underlying programming language
- − knowledge representation in terms of an imperative/object language

Theoretically driven (AgentSpeak(L), 3APL)

- − declarative programming language built from scratch ⇝ new syntax
- − no direct integration with 3rd party/legacy systems
- + clear theoretical properties ⇝ easier verification(?)
- + declarative KR techniques (currently rather weak reasoning capabilities)

Modular BDI architecture

Knowledge Representation:

- encapsulate BDI modules allowing only *query/update* interface
- KR techniques and programming languages ⇝ programmer's decision
- treat agent's capabilities as just another BDI component

Agent System Dynamics:

- interaction between BDI modules ⇝ interaction rules
- application of a interaction rule ⇝ atomic system transition

Interpreter:

- select and execute arbitrary applicable interaction rule

Modular BDI architecture

Knowledge Representation:

- encapsulate BDI modules allowing only *query/update* interface
- KR techniques and programming languages $\rightsquigarrow$ programmer's decision
- treat agent's capabilities as just another BDI component

Agent System Dynamics:

- interaction between BDI modules $\rightsquigarrow$ interaction rules
- application of a interaction rule $\rightsquigarrow$ atomic system transition

Interpreter:

- select and execute arbitrary applicable interaction rule

Modular BDI architecture

Knowledge Representation:

- encapsulate BDI modules allowing only *query/update* interface
- KR techniques and programming languages $\rightsquigarrow$ programmer's decision
- treat agent's capabilities as just another BDI component

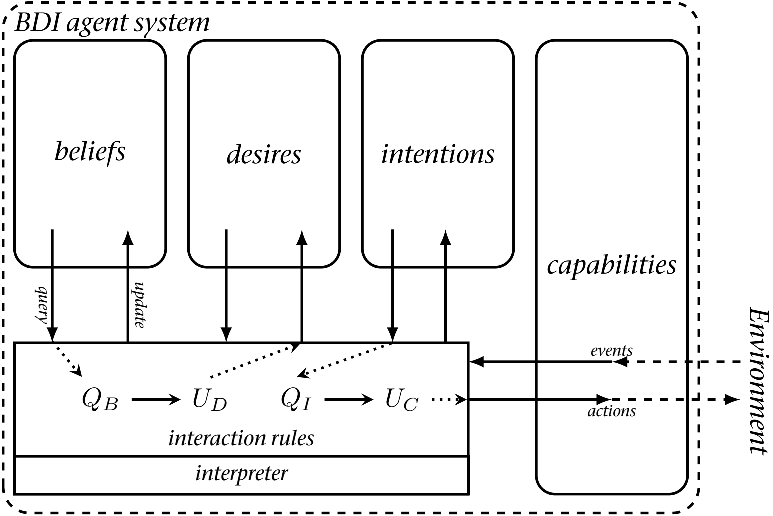
Agent System Dynamics:

- interaction between BDI modules $\rightsquigarrow$ interaction rules
- application of a interaction rule $\rightsquigarrow$ atomic system transition

Interpreter:

- select and execute arbitrary applicable interaction rule

Architecture



Example: Espresso machine

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();
```

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

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

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

Example: Espresso machine

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();
```

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

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

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

Programming language: syntax

declare beliefs **as** Prolog *[{ . . . }]*
declare desires **as** Prolog *[{ . . . }]*
declare intentions **as** Lisp *[{ . . . }]*
declare capabilities **as** C *[{ . . . }]*

when query capabilities *[{!stand_empty}]*
  then update beliefs *[{assert(cup_present)}]*;

when query beliefs *[{ready}]* **and query** desires *[{make_espresso}]*
  then update intentions *[{(push (. . .))}]*;

when query intentions *[{(top? grind)}]*
  then update capabilities *[{mill_start()}]*, **update** intentions *[{(pop)}]*;

when query desires(Type) *[{make(Type)}]* **and query** beliefs *[{ready}]***and**
  query beliefs(Type,Time,Temp,Vol) *[{recipe(Type,Time,Temp,Vol)}]*
then update intentions(Type,Time,Temp,Vol)
  [{(push ((grind Time)(boil Temp)(pour Vol)(done Type)))}];

Programming language: syntax

declare beliefs **as** Prolog *[{ . . . }]*
declare desires **as** Prolog *[{ . . . }]*
declare intentions **as** Lisp *[{ . . . }]*
declare capabilities **as** C *[{ . . . }]*

when query capabilities *[{!stand_empty}]*
then update beliefs *[{assert(cup_present)}]*;

when query beliefs *[{ready}]* **and query** desires *[{make_espresso}]*
then update intentions *[{(push (. . .))}]*;

when query intentions *[{(top? grind)}]*
then update capabilities *[{mill_start()}]*, **update** intentions *[{(pop)}]*;

when query desires(Type) *[{make(Type)}]* **and query** beliefs *[{ready}]***and**
query beliefs(Type,Time,Temp,Vol) *[{recipe(Type,Time,Temp,Vol)}]*
then update intentions(Type,Time,Temp,Vol)
[{(push ((grind Time)(boil Temp)(pour Vol)(done Type)))}];

Mental state transformers

Observations

- rule (a set of rules) $\rightsquigarrow$ partial function on the set of mental states
- unification of two sets of rules $\rightsquigarrow$ partial function! - generalization
- nested rules $\rightsquigarrow$ partial function again! - specialization

... named compound code structures? $\rightsquigarrow$ add macro expansion facility!

Mental state transformers

Observations

- rule (a set of rules) $\rightsquigarrow$ partial function on the set of mental states
- unification of two sets of rules $\rightsquigarrow$ partial function! - generalization
- nested rules $\rightsquigarrow$ partial function again! - specialization

... named compound code structures? $\rightsquigarrow$ add macro expansion facility!

Example: Stock exchange trading agent

```
define careful_strategy(TITLE) {
    when [{ wants(TITLE) }] then [{ drop_goal(wants(TITLE)) }] ;
}
define opportunistic_strategy(TITLE) {
    when [{ wants(TITLE) }] and [{ price(TITLE)<avg(TITLE,12h) }]
    then [{ act(issue_order(buy(TTILE,10))) }] ;

    when [{ price(TITLE)<max(TITLE,180d) }] and [{ price(TITLE)<avg(TITLE,7d) }]
    then [{ introduce_goal(wants(TITLE)) }] ;
}
defineq market_turmoil {
    [{ news('overtake')>2 }] and [{ avg(DOW,5h)<0.70*avg(DOW,2d) }]
}/*****************************************************************/
when market_turmoil then {
    careful_strategy(APPL);
    careful_strategy(MSFT);
} else {
    opportunistic_strategy(APPL);
    opportunistic_strategy(MSFT);
}
```

Pros and cons

Pro's:

- translational semantics $\rightsquigarrow$ plain program
- source code modularity $\rightsquigarrow$ behaviors(?)
- integration of heterogenous components under a BDI umbrella

That's all nice, but:

- how to use it?
- mst's vs. behaviors, roles, etc.
- mst's vs. BDI concepts (goal directed decomposition)
- methodology:
  - how to decompose a problem into mst's?

Pros and cons

Pro's:

- translational semantics $\rightsquigarrow$ plain program
- source code modularity $\rightsquigarrow$ behaviors(?)
- integration of heterogenous components under a BDI umbrella

That's all nice, but:

- how to use it?
- mst's vs. behaviors, roles, etc.
- mst's vs. BDI concepts (goal directed decomposition)
- methodology:
    - how to decompose a problem into mst's?

Ongoing work and outlooks

Modular BDI architecture

paper published, AAMAS 2006.

Programming language

- code modularity $\rightsquigarrow$ higher level programming constructs (mental state transformers), TR IfI-06-12
- *Jazyk* language interpreter under construction (summer 2007?)

Methodology

experiments, experiments, experiments! $\rightsquigarrow$ bottom-up approach

Ongoing work and outlooks

Modular BDI architecture

paper published, AAMAS 2006.

Programming language

- code modularity $\rightsquigarrow$ higher level programming constructs (mental state transformers), TR IfI-06-12
- *Jazyk* language interpreter under construction (summer 2007?)

Methodology

experiments, experiments, experiments! $\rightsquigarrow$ bottom-up approach

Conclusion

Project

Programming framework for development of BDI agents with mental states:

- architecture
- programming language
- methodology

Modularity & integration

Different programming languages are suitable for different knowledge representation tasks.

Question?

THANK YOU FOR YOUR ATTENTION.