TU Clausthal
Clausthal University of Technology

Peter Novák and Jürgen Dix
Department of Informatics, Clausthal University of Technology
Julius-Albert-Str. 4, D-38678 Clausthal-Zellerfeld, Germany
peter.novak@in.tu-clausthal.de, dix@tu-clausthal.de

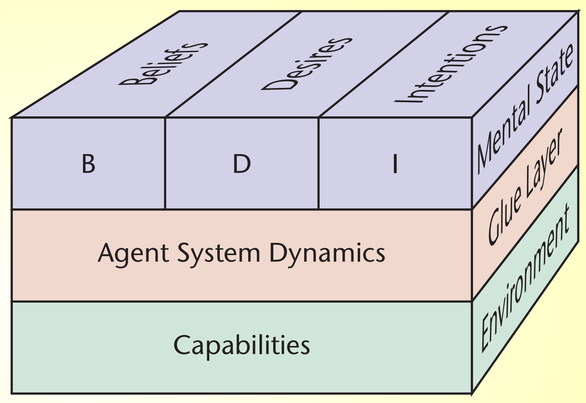
Modular BDI Architecture

(single agent programming system)

1. Motivation

agent

... can be seen as a 3-layer system ...



Programming a BDI agent means:

- providing structures representing agent's mental state (beliefs, desires, intentions),
- programming the system dynamics so that agent moves from one mental state to another in a controlled manner,
- providing means to manipulate agent's environment (capabilities).

2. State of the art

BDI Programming Platforms

Theoretically driven systems (AgentSpeak(L)/Jason, 3APL)

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

Engineering approaches (JadeX, JACK)

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

Different programming languages are suitable for different knowledge representation tasks.

Beliefs (*Prolog*)

```
%%% Query Interface %%%
% Failures of machine
error(1) :- grinder_error.
error(2) :- boiler_error.
error :- error(_).

warning(1) :- noBeans.
warning(2) :- noWater.
warning :- warning(_).

% Recipes for coffee
% ( type,
%   grind time/s,
%   temperature/deg,
%   water amount/ml)
recipe(espresso,3,92,30).
recipe(coffee,4,98,150).

% Is machine ready
ready :- cup_present,
         cup_empty,
         not warning,
         not error.

%%% Update Interface %%%
assert(Term) :- ...
retract(Term) :- ...
```

Desires (*set of terms - Prolog*)

```
%%% Query Interface %%%
make(espresso).
no_error.
no_warning.

%%% Update Interface %%%
assert(Term) :- ...
retract(Term) :- ...
```

Intentions (*stack - Lisp*)

```
;; Update Interface
(define push ...)
(define pop ...)
(define flush ...)

;; Query Interface
(define top? ...)
```

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

$$Q_D(\text{make(Type)}) \wedge Q_B(\text{ready}) \wedge Q_B(\text{recipe(Type,Time,Temp,Vol)}) \longrightarrow U_I(\text{(push ((grind Time) (boil Temp) (pour Vol) (done Type)))})$$

$$Q_I(\text{(top? (done Type))}) \wedge Q_D(\text{make(Type)}) \longrightarrow U_D(\text{retract(make(Type))}) \circ U_I(\text{(pop)})$$

$$Q_I(\text{(top? (grind(Time)))}) \longrightarrow U_C(\text{mill\_grind(Time)})$$

$$Q_C(\text{mill\_error()}) \longrightarrow U_B(\text{assert(grinder\_error)}) \circ U_I(\text{(flush)})$$

$$Q_D(\text{no\_error}) \wedge Q_B(\text{error(Err)}) \longrightarrow U_C(\text{display\_error(Err)})$$

Capabilities (C)

```
enum {NoBeans=1, NoWater=2} EWarning;
enum {MillErr=1, BoilerErr=2} EError;

// actions
void boiler_heat(int temperature);
void mill_grind(int time);
void water_pour(int amount);
void display_error(EError);
void display_warning(EWarning);

// events/queries
int stand_empty();
int cup_empty();
int detect_cup_size();
int mill_error();
int boiler_error();
```

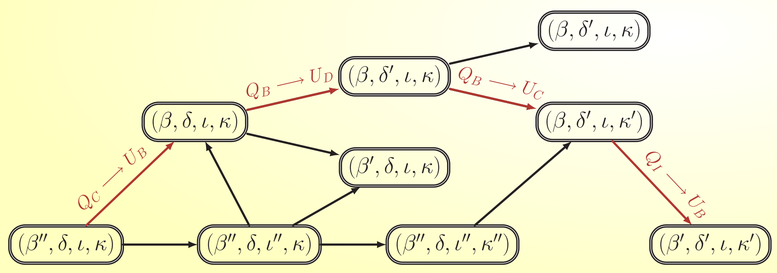
3. Modular BDI Architecture

We propose to clearly separate knowledge representation issues and agent system dynamics.

- **Knowledge representation layer:**
  1. encapsulate and separate each BDI module allowing only *query*/*update* interfaces
  2. leave decision about KR techniques and programming languages to a programmer
- **Agent system dynamics layer:**
  1. interaction between BDI components - *exclusively* via interaction rules
  2. application of an interaction rule ⇝ atomic agent system transition
- **Agent vs. Environment**
  - treat agent's capabilities (sensors/effectors) as just another BDI component

4. Agent system semantics

A BDI agent is a tuple $(\beta_0, \delta_0, \iota_0, \kappa_0, \mathcal{IR})$, where $(\beta_0, \delta_0, \iota_0, \kappa_0)$ is an initial configuration and $\mathcal{IR}$ is a set of interaction rules.



- interaction rule: $\phi \longrightarrow \psi$ = *if query, then update*
  - chaining of queries and updates ($\wedge$/$\circ$)
  - variables across query and update formulas
- set of interaction rules $\mathcal{IR}$ induces an agent transition system
- *interpreter*'s cycle:
  1. select an interaction rule
  2. evaluate query part (lhs)
     - if true ⇝ execute the update (rhs) and restart the selection cycle
     - if false ⇝ continue selection

5. Contributions & Shortcomings

- + highly abstract framework allowing to accommodate a range of cognitive agent models ⇝ various models of rationality
- + modular agent programming system
- + easy integration with 3rd party/legacy systems
- + easy code and 3rd party library re-use
- + standard SW development techniques for BDI components
- + clear semantics of system dynamics
- − semantics of components relies on the underlying programming language
- − weak interpreting mechanism ⇝ *ongoing and future work*
- − new syntax ⇝ yet another agent language

6. Ongoing & Future Work

lifting to a more general case:

- defining custom BDI components

programming language:

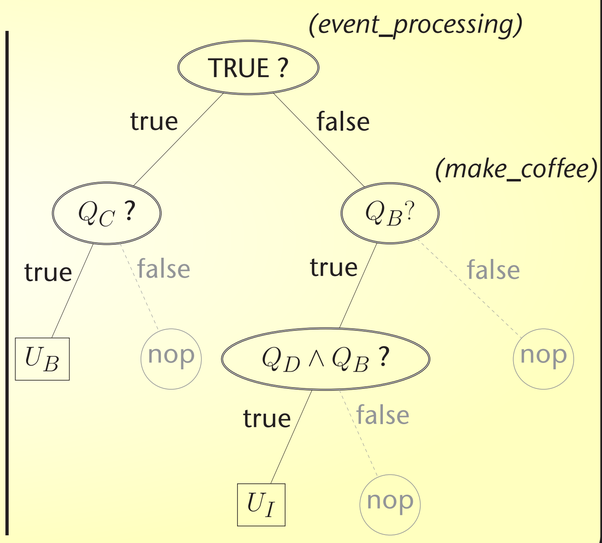
- tree program structure, descriptive syntax, rules nesting (roles), name scopes, inheritance-like rule extension, dynamic/reflexive program structure, powerful interpreter control

interpreter:

- we currently work on an implementation of the interpreter with plug-in architecture (resembling Apache httpd) in C++

```
event_processing: when TRUE then {
  when query capabilities
       [{!stand_empty()}]
  then update beliefs
       [{assert(cup_present)}]
}

make_coffee: when query beliefs
       [{ready}]
then {
  when query desires [{make(...)}]
   and query beliefs [{recipe(...)}]
  then update intentions
    [{(push ((grind Time) ...))}]
}
```



AAMAS'06, Future University Hakodate, Hokkaido, Japan