Behavioural State Machines

Framework for programming cognitive agents

Peter Novák

Clausthal University of Technology, Germany

Thursday, February 19, 2009
ATG@ČVUT

Motivation & domain frame

Embodied cognitive/knowledge intensive agent

generates and maintains a model of its environment and uses it as a basis for deciding about its future actions.

environment: rich, unstructured, dynamic, noisy
mental attitudes: beliefs, desires, plans, obligations, norms, etc.
knowledge: different KR techniques are suitable for different tasks

Problem statement

1. How to program cognitive agents?
   - reactiveness vs. deliberation & heterogeneous KRR
   - ⇝ practical framework
2. How to use such a framework? Pragmatics?
   - set of sound methodological guidelines, design phase support

Agenda

1 Motivation

2 Behavioural State Machines/Jazzyk

3 Code patterns for agent programming

4 Case studies

5 Summary & conclusion

Behavioural State Machines

Different programming languages are suitable for different knowledge representation tasks.

↓

Heterogeneous knowledge bases!

Focus on encoding agent's behaviours.

Behavioural State Machines

A programming framework with clear separation between *knowledge representation* and agent's *behaviours*.

BSM framework provides:

- clear *semantics:* Gurevich's Abstract State Machines
- *modularity:* KR, source code
- easy *integration* with external/legacy systems

Jazzyk BSM agent: $\mathcal{A} = (\mathcal{M}_1, \ldots, \mathcal{M}_n, \mathcal{P})$

KR module $\mathcal{M} = (\mathcal{S}, \mathcal{L}, \mathcal{Q}, \mathcal{U})$

- $\mathcal{S}$ - a set of states
- $\mathcal{L}$ - a KR language,
- $\mathcal{Q}$ - a set of query operators $\models: \mathcal{S} \times \mathcal{L} \to \{\top, \bot\}$,
- $\mathcal{U}$ - set of update operators $\oplus : \mathcal{S} \times \mathcal{L} \to \mathcal{S}$.

mental state transformer τ: $\models_i \varphi \longrightarrow \oplus_j \psi$

when query$_i$ **module**$_i$ [{ φ }] **then** update$_j$ **module**$_j$ [{ ψ }]

```
when query_i module_i [{...}] and query_j module_j [{...}] then {
    when query_k module_k [{...}] then {
        ...
    } |
    update_l module_l [{...}]
}
```

Jazzyk - the BSM language
$\rightsquigarrow$ appeared @ ProMAS'08

Jazzyk BSM agent: $\mathcal{A} = (\mathcal{M}_1, \ldots, \mathcal{M}_n, \mathcal{P})$

KR module $\mathcal{M} = (\mathcal{S}, \mathcal{L}, \mathcal{Q}, \mathcal{U})$

- $\mathcal{S}$ - a set of states
- $\mathcal{L}$ - a KR language,
- $\mathcal{Q}$ - a set of query operators $\models: \mathcal{S} \times \mathcal{L} \rightarrow \{\top, \bot\}$,
- $\mathcal{U}$ - set of update operators $\oplus : \mathcal{S} \times \mathcal{L} \rightarrow \mathcal{S}$.

mental state transformer τ: $\models_i \varphi \longrightarrow \oplus_j \psi$

when query$_i$ **module**$_i$ [{ φ }] **then** update$_j$ **module**$_j$ [{ ψ }]

```
when query_i module_i [{...}] and query_j module_j [{...}] then {
    when query_k module_k [{...}] then {
        ...
    } |
    update_l module_l [{...}]
}
```

Jazzyk **- the BSM language**
⇝ appeared @ ProMAS'08

Jazzyk BSM agent: $\mathcal{A} = (\mathcal{M}_1, \ldots, \mathcal{M}_n, \mathcal{P})$

KR module $\mathcal{M} = (\mathcal{S}, \mathcal{L}, \mathcal{Q}, \mathcal{U})$

- $\mathcal{S}$ - a set of states
- $\mathcal{L}$ - a KR language,
- $\mathcal{Q}$ - a set of query operators $\models: \mathcal{S} \times \mathcal{L} \rightarrow \{\top, \bot\}$,
- $\mathcal{U}$ - set of update operators $\oplus : \mathcal{S} \times \mathcal{L} \rightarrow \mathcal{S}$.

mental state transformer τ: $\models_i \varphi \longrightarrow \oplus_j \psi$

when query$_i$ **module**$_i$ [{ φ }] **then** update$_j$ **module**$_j$ [{ ψ }]

when query$_i$ **module**$_i$ [{...}] **and** query$_j$ **module**$_j$ [{...}] **then** {
    when query$_k$ **module**$_k$ [{...}] **then** {
        ...
    } |
    update$_l$ **module**$_l$ [{...}]
}

***Jazzyk* - the BSM language**

$\rightsquigarrow$ appeared @ ProMAS'08

Semantics: $\mathcal{A} = (\mathcal{M}_1, \ldots, \mathcal{M}_n, \mathcal{P})$

labeled transition system over states $\sigma = \langle \sigma_1, \ldots, \sigma_n \rangle$

induced by updates $\oplus\psi$

$yields(\tau, \sigma, \rho)$

$$\frac{\top}{yields(\textbf{skip}, \sigma, \textbf{skip})} \qquad \frac{\top}{yields(\textbf{update}_{\oplus_i}\ \textbf{module}_i\ \psi, \sigma, \oplus_i\psi)} \qquad \frac{yields(\tau, \sigma, \rho)}{yields(\{\tau\}, \sigma, \rho)}$$

$$\frac{yields(\tau, \sigma, \rho),\ \sigma \models_i \phi}{yields(\textbf{when query}_{\models_i}\ \textbf{module}_i\ \phi\ \textbf{then}\ \tau, \sigma, \rho)} \qquad \frac{yields(\tau, \sigma, \rho),\ \sigma \not\models_i \phi}{yields(\textbf{...}, \sigma, \textbf{skip})}$$

$$\frac{yields(\tau_1, \sigma, \rho_1),\ yields(\tau_2, \sigma, \rho_2)}{yields(\tau_1 ; \tau_2, \sigma, \rho_1) \quad yields(\tau_1 ; \tau_2, \sigma, \rho_2)}$$

$$\frac{yields(\tau_1, \sigma, \rho_1),\ yields(\tau_2, \sigma \bigoplus \rho_1, \rho_2)}{yields(\tau_1, \tau_2, \sigma, \rho_1 \bullet \rho_2)}$$

Semantics: $\mathcal{A} = (\mathcal{M}_1, \ldots, \mathcal{M}_n, \mathcal{P})$

labeled transition system over states $\sigma = \langle \sigma_1, \ldots, \sigma_n \rangle$
induced by updates $\oplus\psi$
$yields(\tau, \sigma, \rho)$

$$\frac{\top}{yields(\textbf{skip},\sigma,\textbf{skip})} \qquad \frac{\top}{yields(\textbf{update}_{\oplus_i}\ \textbf{module}_i\ \psi,\sigma,\oplus_i\psi)} \qquad \frac{yields(\tau,\sigma,\rho)}{yields(\{\tau\},\sigma,\rho)}$$

$$\frac{yields(\tau,\sigma,\rho),\ \sigma\models_i\phi}{yields(\textbf{when query}_{\models_i}\ \textbf{module}_i\ \phi\ \textbf{then}\ \tau,\sigma,\rho)} \qquad \frac{yields(\tau,\sigma,\rho),\ \sigma\not\models_i\phi}{yields(\textbf{...},\sigma,\textbf{skip})}$$

$$\frac{yields(\tau_1,\sigma,\rho_1),\ yields(\tau_2,\sigma,\rho_2)}{yields(\tau_1;\tau_2,\sigma,\rho_1)\quad yields(\tau_1;\tau_2,\sigma,\rho_2)}$$

$$\frac{yields(\tau_1,\sigma,\rho_1),\ yields(\tau_2,\sigma\bigoplus\rho_1,\rho_2)}{yields(\tau_1,\tau_2,\sigma,\rho_1\bullet\rho_2)}$$

Semantics cont.

computation step

$\mathcal{A} = (\mathcal{M}_1, \ldots, \mathcal{M}_n, \mathcal{P})$ induces a transition $\sigma \rightarrow \sigma'$ iff $\sigma' = \sigma \bigoplus \rho$ and the program $\mathcal{P}$ yields ρ in σ, i.e. $\exists\theta : yields(\mathcal{P}, \sigma, \rho)$.

Jazzyk BSM semantics (operational view)

A sequence $\sigma_1, \ldots, \sigma_i, \ldots$, s.t. $\sigma_i \rightarrow \sigma_{i+1}$, is a trace of BSM.
An agent system (BSM), is characterized by a set of all traces.

Jazzyk BSM semantics (denotational view)

$\tau \rightsquigarrow \mathfrak{f}_\tau : \sigma \mapsto \{\rho | yields(\tau, \sigma, \rho)\}$: a specification of *enabled* updates

policies, code modularity

$\rightsquigarrow$ appeared @ AAMAS'06, ProMAS'07, AITA'08, ProMAS'08

Semantics cont.

computation step

$\mathcal{A} = (\mathcal{M}_1, \ldots, \mathcal{M}_n, \mathcal{P})$ induces a transition $\sigma \rightarrow \sigma'$ iff $\sigma' = \sigma \bigoplus \rho$ and the program $\mathcal{P}$ yields ρ in σ, i.e. $\exists \theta : yields(\mathcal{P}, \sigma, \rho)$.

Jazzyk BSM semantics (operational view)

A sequence $\sigma_1, \ldots, \sigma_i, \ldots$, s.t. $\sigma_i \rightarrow \sigma_{i+1}$, is a trace of BSM.
An agent system (BSM), is characterized by a set of all traces.

Jazzyk BSM semantics (denotational view)

$\tau \rightsquigarrow \mathfrak{f}_\tau : \sigma \mapsto \{\rho | yields(\tau, \sigma, \rho)\}$: a specification of *enabled* updates

policies, code modularity

$\rightsquigarrow$ appeared @ AAMAS'06, ProMAS'07, AITA'08, ProMAS'08

```
/* When the searched item is found, pick it */
when desires_G [{ task(pick(X)) }] then {
    /* PICK */
    when believes_B [{ see(X) }] then {
        when believes_B [{ dir(X,'ahead'), dist(X,Dist) }] then act_E [{ move forward Dist }] ;
        when believes_B [{ dir(X, Angle) }] then act_E [{ turn Angle }]
    } . . . ;
} ;
/* Goal adoption */
when believes_B [{needs(X)}] then add_G [{task(pick(X))}] ;
/* Drop the goal */
when desires_G [{ task(pick(X)) }] and believes_B [{holds(X)}] then remove_G [{task(pick(X))}] ;

/* When endangered, run away */
when desires_G [{ maintain(safety) }] and believes_B [{ threatened }] then {
    /* RUN_AWAY */
    when believes_B [{ random(Angle) }] then {
        act_E [{ turn Angle }] ,
        act_E [{ move forward 10 }]
    } ;
    . . .
}
```

```
/* When the searched item is found, pick it */
ACHIEVE('task(pick(X))', 'needs(X)', 'holds(X)', PICK)
 ;
/* When endangered, run away */
MAINTAIN('maintain(safety)', 'threatened', RUN_AWAY)
```

⇝ code adapted from (Novák, Köster @ CogRob'08)

Problem: *pragmatics of programming with BSM*

*BSM $\rightsquigarrow$ **generic** programming language for cognitive agents*

specification $\phi \rightsquigarrow$ program $\mathcal{P}$

Support of design process by
code templates/idioms/design patterns...

Logic for BSM

⇝ (Novák, Jamroga @ AAMAS'09)

DCTL*=DL+CTL* LTL⊂DCTL*

A hybrid of *Dynamic Logic* and *Temporal Logic CTL**:

$$([\tau_1]\Diamond\varphi_1)\ \mathcal{C}\ [\tau_2]\Box(\varphi_2 \vee \varphi_3)$$

Program annotations ⇝ aggregation!

Annotation function $\mathfrak{A} : (\mathcal{Q}(\mathcal{A}) \cup \mathcal{U}(\mathcal{A})) \rightarrow LTL$

$$\mathfrak{A}(\oplus_{\mathcal{B}} see(friend)) = \bigcirc\, happy \quad \rightsquigarrow \quad [\oplus_{\mathcal{B}} see(friend)] \bigcirc\, happy$$

Logic for BSM

$\rightsquigarrow$ (Novák, Jamroga @ AAMAS'09)

DCTL*=DL+CTL* — LTL⊂DCTL*

A hybrid of *Dynamic Logic* and *Temporal Logic CTL**:

$$([\tau_1]\Diamond\varphi_1)\ \mathcal{C}\ [\tau_2]\Box(\varphi_2 \vee \varphi_3)$$

Program annotations $\rightsquigarrow$ aggregation!

Annotation function $\mathfrak{A} : (\mathcal{Q}(\mathcal{A}) \cup \mathcal{U}(\mathcal{A})) \rightarrow LTL$

$$\mathfrak{A}(\oplus_{\mathcal{B}} see(friend)) = \bigcirc\, happy \quad \rightsquigarrow \quad [\oplus_{\mathcal{B}} see(friend)] \bigcirc\, happy$$

Logic for BSM

⇝ (Novák, Jamroga @ AAMAS'09)

DCTL*=DL+CTL* LTL⊂DCTL*

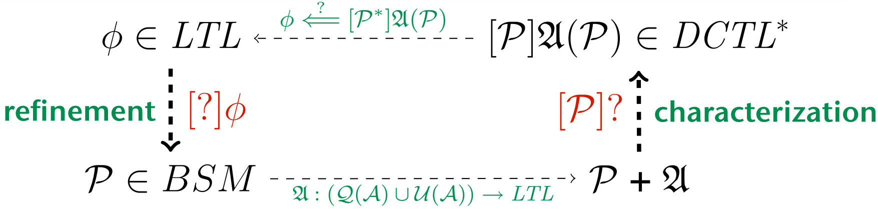
A hybrid of *Dynamic Logic* and *Temporal Logic CTL**:

$$([\tau_1]\Diamond\varphi_1)\ \mathcal{C}\ [\tau_2]\Box(\varphi_2 \vee \varphi_3)$$

Program annotations ⇝ aggregation!

Annotation function $\mathfrak{A} : (\mathcal{Q}(\mathcal{A}) \cup \mathcal{U}(\mathcal{A})) \rightarrow LTL$

$$\mathfrak{A}(\oplus_{\mathcal{B}} see(friend)) = \bigcirc\, happy \quad \rightsquigarrow \quad [\oplus_{\mathcal{B}} see(friend)] \bigcirc\, happy$$



Behavioural design patterns: decomposition

Example (gradual refinement)

1. $S_1 \equiv \varphi$
2. $S_2 \equiv \phi_1 \wedge \phi_2 \vee \phi_3$ and $\phi_1 \wedge \phi_2 \vee \phi_3 \Rightarrow \varphi$
3. $S_3 \equiv [ACHIEVE(\tau_1)]\phi_1 \wedge \ldots \vee [MAINTAIN(\tau_3)]\phi_3$
4. $\ldots$
5. $S_5 \equiv [\mathcal{P}](\phi_1 \wedge \ldots \vee \phi_3)$ and $\mathcal{P} \leftrightarrow \pi_1, \ldots, \pi_3$

$$S_5 \Rightarrow S_4 \Rightarrow S_3 \Rightarrow S_2 \Rightarrow S_1 \equiv \varphi$$

Agent system architecture (BDI instance)

We assume BDI-like agent architecture: $\mathcal{B}$, $\mathcal{G}$, $\mathcal{E}$ with $\models_i, \oplus_i, \ominus_i$.

robot in 3D environment: search/pick/deliver/escape

Structure:

$\mathcal{B}$: belief base in Prolog-like language ($\models_\mathcal{B}, \oplus_\mathcal{B}, \ominus_\mathcal{B}$)

$\mathcal{G}$: goal base in Prolog as well ($\models_\mathcal{G}, \oplus_\mathcal{G}, \ominus_\mathcal{G}$)

$\mathcal{E}$: interface to environment - body ($\models_\mathcal{E}, \oplus_\mathcal{E}, \ominus_\mathcal{E}$)

Behaviours:

FIND: $[\mathsf{FIND}]\mathfrak{A}(\mathsf{FIND}) \Rightarrow [\mathsf{FIND}^*]\Diamond holds(item42)$

RUN_AWAY: $[\mathsf{RUN\_AWAY}]\mathfrak{A}(\mathsf{RUN\_AWAY}) \Rightarrow [\mathsf{RUN\_AWAY}^*]\Diamond safe$

BSM design patterns: TRIGGER

define TRIGGER($\varphi_{\mathbf{G}}$, τ)
  when $\models_{\mathcal{G}} \varphi_{\mathbf{G}}$ **then** τ
end

$$[\tau]\mathfrak{A}(\tau) \Rightarrow (\mathfrak{A}(\models_{\mathcal{G}} \varphi_{\mathbf{G}}) \rightarrow [\text{TRIGGER}(\varphi_{\mathbf{G}}, \tau)^*]\Diamond\mathfrak{A}(\tau))$$

running example (cont.)

TRIGGER($has(item42)$, FIND)

TRIGGER($keep\_safe$, RUN_AWAY)

BSM design patterns: ADOPT/DROP

define ADOPT($\varphi_{\mathbf{G}}$, $\psi_{\oplus}$)
  when $\models_{\mathcal{B}} \psi_{\oplus}$ **and not** $\models_{\mathcal{G}} \varphi_{\mathbf{G}}$ **then** $\oplus_{\mathcal{G}} \varphi_{\mathbf{G}}$
end

define DROP($\varphi_{\mathbf{G}}$, $\psi_{\ominus}$)
  when $\models_{\mathcal{B}} \psi_{\ominus}$ **and** $\models_{\mathcal{G}} \varphi_{\mathbf{G}}$ **then** $\ominus_{\mathcal{G}} \varphi_{\mathbf{G}}$
end

$$\mathfrak{A}(\models_{\mathcal{B}} \psi_{\oplus}) \rightarrow [\mathrm{ADOPT}(\varphi_{\mathbf{G}}, \psi_{\oplus})^*]\Diamond\mathfrak{A}(\models_{\mathcal{G}} \psi_{\mathbf{G}})$$
$$\mathfrak{A}(\models_{\mathcal{B}} \psi_{\ominus}) \rightarrow [\mathrm{DROP}(\varphi_{\mathbf{G}}, \psi_{\ominus})^*]\Diamond\neg\mathfrak{A}(\models_{\mathcal{G}} \psi_{\mathbf{G}})$$

running example cont.

ADOPT($has(item42)$, $needs(item42)$)
DROP($has(item42)$, $\neg needs(item42) \vee \neg exists(item42)$)

BSM design patterns: ACHIEVE

define ACHIEVE($\varphi_\mathbf{G}, \varphi_\mathbf{B}, \psi_\oplus, \psi_\ominus, \tau$)
TRIGGER($\varphi_\mathbf{G}, \tau$) ;
ADOPT($\varphi_\mathbf{G}, \psi_\oplus$) ;
DROP($\varphi_\mathbf{G}, \varphi_\mathbf{B}$) ;
DROP($\varphi_\mathbf{G}, \psi_\ominus$)
end

$$([\tau]\mathfrak{A}(\tau) \wedge [\tau^*]\Diamond\varphi_\mathbf{B}) \Rightarrow$$
$$[\text{ACHIEVE}(\varphi_\mathbf{G}, \varphi_\mathbf{B}, \psi_\oplus, \psi_\ominus, \tau)^*]\mathfrak{A}(\models_\mathcal{G} \varphi_\mathbf{G})\,\mathcal{U}\,\mathfrak{A}(\models_\mathcal{B} \varphi_\mathbf{B} \vee \models_\mathcal{B} \varphi_\ominus)$$

running example cont.

ACHIEVE(
$has(item42)$,
$holds(item42)$,
$needs(item42)$,
$\neg needs(item42) \vee \neg exists(item42)$,
FIND)

BSM design patterns: MAINTAIN

define MAINTAIN($\varphi_{\mathbf{G}}, \varphi_{\mathbf{B}}, \tau$)
  when not $\models_{\mathcal{B}} \varphi_{\mathbf{B}}$ **then** TRIGGER($\varphi_{\mathbf{G}}, \tau$) ;
  ADOPT($\varphi_{\mathbf{G}}, \top$)
end

$$([\tau]\mathfrak{A}(\tau) \wedge [\tau^*]\Diamond\mathfrak{A}(\models_{\mathcal{B}} \varphi_{\mathbf{B}})) \Rightarrow$$
$$(\mathfrak{A}(\models_{\mathcal{G}} \varphi_{\mathbf{G}}) \rightarrow [\mathsf{MAINTAIN}(\varphi_{\mathbf{G}}, \varphi_{\mathbf{B}}\tau)^*]\Box(\neg\mathfrak{A}(\models_{\mathcal{B}} \varphi_{\mathbf{B}}) \rightarrow \Diamond\mathfrak{A}(\models_{\mathcal{B}} \varphi_{\mathbf{B}})))$$

running example cont.

MAINTAIN($keep\_safe$, $safe$, RUN_AWAY)

Running example finish

Robot program

PERCEIVE ,
{
MAINTAIN(
$keep\_safe$,
$threatened$,
RUN_AWAY) ;

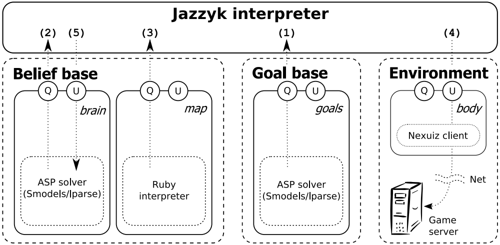
ACHIEVE(
$has(item42)$,
$holds(item42)$,
$needs(item42)$,
$\neg needs(item42) \vee \neg exists(item42)$,
FIND)
}

Jazzbot: bot in a simulated 3D environment

- testbed for heterogeneous KRR in agent domain
- testbed for NMR/ASP/EKB apps in dynamic environment
- challenging, rich, dynamic environment

Agent program:

```
when believes goals(Obj) [{find(Obj)}] and                    (1)
     believes brain(Obj) [{see(Obj)}] and                     (2)
     query map(Object, Dist) [{Dist=get_distance_of(Obj)}]    (3)
then {
     act body(Dist) [{move forward Dist}] ,                   (4)
     update brain(Obj) [{keeps(Obj)}]                         (5)
}
```



⇝ Novák @ ProMAS'08, detailed report under submission

URBI-Bot: simulated e-Puck mobile robot

⇝ towards embodied cognitive robotics

URBI

Simple event-based programming language

⇝ OS independent, easy integration, modular - components for robotic HW modules

URBI-Bot (e-Puck robot)

- similar architecture as Jazzbot, i.e. uses NMR/ASP
- code for the *Webots* simulator is directly transferable to the real robot

⇝ detailed report under submission

Summary

Original problem statement

1. How to program cognitive agents?
   - reactiveness vs. deliberation & heterogeneous KRR
2. How to use such a framework? Pragmatics?
   - set of sound methodological guidelines, design phase support

Proposed solutions:

1. Behavioural State Machines/Jazzyk framework
2. BSM design patterns + informal BDI directed methodology

Not only theory, but also demonstrated functionality!

- *robust & efficient action selection* (BSM semantics)
- *elaboration tolerant programming style* (macros, patterns)
- *horizontal & vertical modularity*
  (source code modules, KRR technologies/3rd party apps)

On-going & future research agenda

1. Efficient & robust action selection
   - model-checking of DCTL* annotations
   - *Probabilistic Behavioural State Machines* (submitted)
   - extensive library of *BSM* design patterns
2. Towards open multi-agent systems
   - platform for open heterogeneous MASs (position paper)
   - using SRI's OAA as a MAS communication middleware (Agent Contest 2009)
3. Exploiting limits of *BSM*
   - non-player characters for computer games
   - entertainment robotics: Rovio, Nao (URBI)

Thank you for your attention...

http://jazzyk.sourceforge.net/