Behavioural State Machines

(programming modular agents)

Peter Novák

Clausthal University of Technology, Germany

March 27th, 2008
AITA'08/AAAI SS 2008, Stanford University, USA

Outline

Agent programming

Knowledge intensive/cognitive agents

- *knowledge* - attitudes, mental state, state of environment
- *body* - sensors/effectors $\rightsquigarrow$ environment
- *system dynamics* - reasoning, behaviours and performing actions

Challenges for programming frameworks:

- *theoretical properties* - insight into system properties $\rightsquigarrow$ verification, analysis, design
- *practical applicability* - support of traditional SW development techniques, modularity, integration with external systems

(Belief-Desire-Intention metaphor in mind)

Agent programming

Knowledge intensive/cognitive agents

- *knowledge* - attitudes, mental state, state of environment
- *body* - sensors/effectors $\rightsquigarrow$ environment
- *system dynamics* - reasoning, behaviours and performing actions

Challenges for programming frameworks:

- *theoretical properties* - insight into system properties $\rightsquigarrow$ verification, analysis, design
- *practical applicability* - support of traditional SW development techniques, modularity, integration with external systems

(Belief-Desire-Intention metaphor in mind)

My way to go...

Different programming languages are suitable for different knowledge representation tasks.

Focus on encoding agent's behaviours.

Behavioural State Machines

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

Desiderata:

- clear *semantics*
- *modularity* (KR, source code)
- easy *integration* with external/legacy systems

My way to go...

Different programming languages are suitable for different knowledge representation tasks.

↓

Focus on encoding agent's behaviours.

Behavioural State Machines

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

Desiderata:

- clear *semantics*
- *modularity* (KR, source code)
- easy *integration* with external/legacy systems

My way to go...

Different programming languages are suitable for different knowledge representation tasks.

⬇

Focus on encoding agent's behaviours.

Behavioural State Machines

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

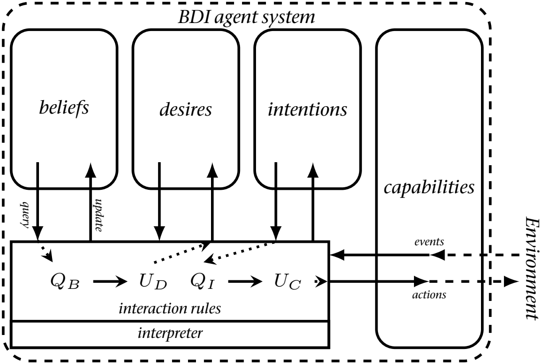
Desiderata:

- clear *semantics*
- *modularity* (KR, source code)
- easy *integration* with external/legacy systems

BSM overview

core concept: KR module $\mathcal{M} = (\mathcal{L}, \mathcal{Q}, \mathcal{U})$

- $\mathcal{L}$ - 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 $\oplus : \mathcal{S} \times \mathcal{L} \rightarrow \mathcal{S}$.



BSM Syntax: $Query \longrightarrow Update$

query formulae

- $\models \varphi$ is a query ($\varphi \in \mathcal{L}_i$, and $\models \in \mathcal{U}_i$ of a KR module $\mathcal{M}_i$)
- $\phi_1 \wedge \phi_2$, $\phi_1 \vee \phi_2$ and $\neg\phi_1$ are queries

mental state transformer (mst)

primitive	$\mathbf{skip}$ is a mst
primitive	$\oplus\psi$ is a mst ($\oplus \in \mathcal{U}_i$, $\psi \in \mathcal{L}_i$ of a module $\mathcal{M}_i$)
conditional	$\phi \longrightarrow \tau$ is a mst (ϕ is a query, and τ is a mst)
choice	$\tau \vert \tau'$
sequence	$\tau \circ \tau'$

BSM Syntax: $Query \longrightarrow Update$

query formulae

- $\models \varphi$ is a query ($\varphi \in \mathcal{L}_i$, and $\models \in \mathcal{U}_i$ of a KR module $\mathcal{M}_i$)
- $\phi_1 \wedge \phi_2$, $\phi_1 \vee \phi_2$ and $\neg\phi_1$ are queries

mental state transformer (mst)

primitive	**skip** is a mst
primitive	$\oplus\psi$ is a mst ($\oplus \in \mathcal{U}_i$, $\psi \in \mathcal{L}_i$ of a module $\mathcal{M}_i$)
conditional	$\phi \longrightarrow \tau$ is a mst (ϕ is a query, and τ is a mst)
choice	$\tau \vert \tau'$
sequence	$\tau \circ \tau'$

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

transition system over states $\sigma = \langle \sigma_1, \ldots, \sigma_n \rangle$ induced by updates $\oplus\psi$

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

$$\frac{\top}{yields(\mathbf{skip},\sigma,\emptyset)} \qquad \frac{\top}{yields(\oplus\psi,\sigma,\{(\oplus,\psi)\})}$$

update **module** *[{...}]*

$$\frac{yields(\tau,\sigma,\nu),\ \sigma\models\phi}{yields(\phi\longrightarrow\tau,\sigma,\nu)} \qquad \frac{yields(\tau,\sigma,\nu),\ \sigma\not\models\phi}{yields(\phi\longrightarrow\tau,\sigma,\emptyset)}$$

when query **module** *[{...}]*
then {...}

$$\frac{yields(\tau_1,\sigma,\nu_1),\ yields(\tau_2,\sigma,\nu_2)}{yields(\tau_1|\tau_2,\sigma,\nu_1\cup\nu_2)}$$

{...} ;
{...}

$$\frac{yields(\tau_1,\sigma,\nu_1\neq\emptyset),\ \forall\rho\in\nu_1 : yields(\tau_2,\sigma \bigoplus \rho,\nu_\rho)}{yields(\tau_1\circ\tau_2,\sigma,\bigcup_{\forall\rho\in\nu_1}\{\rho\}\bullet\nu_\rho)}$$

{...} , // $\nu_1 = \{\rho_1^{\nu_1}, \ldots \rho_n^{\nu_1}\} \bullet$
{...} // $\nu_2 = \{\rho_1^{\nu_2}, \ldots \rho_m^{\nu_2}\}$

$$\frac{yields(\tau_1,\sigma,\emptyset),\ yields(\tau_2,\sigma,\nu_2)}{yields(\tau_1\circ\tau_2,\sigma,\nu_2)}$$

$$\nu_1 \bullet \nu_2 = \{\rho_1 \bullet \rho_2 | (\rho_1, \rho_2) \in \nu_1 \times \nu_2\}$$
$$\sigma \bigoplus \rho_1 \bullet \rho_2 \rightsquigarrow (\sigma \bigoplus \rho_1) \bigoplus \rho_2$$

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

transition system over states $\sigma = \langle \sigma_1, \ldots, \sigma_n \rangle$ induced by updates $\oplus\psi$

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

$$\frac{\top}{yields(\mathbf{skip},\sigma,\emptyset)} \qquad \frac{\top}{yields(\oplus\psi,\sigma,\{(\oplus,\psi)\})}$$

update **module** *[{...}]*

$$\frac{yields(\tau,\sigma,\nu),\ \sigma\models\phi}{yields(\phi\longrightarrow\tau,\sigma,\nu)} \qquad \frac{yields(\tau,\sigma,\nu),\ \sigma\not\models\phi}{yields(\phi\longrightarrow\tau,\sigma,\emptyset)}$$

when query **module** *[{...}]*
then {...}

$$\frac{yields(\tau_1,\sigma,\nu_1),\ yields(\tau_2,\sigma,\nu_2)}{yields(\tau_1|\tau_2,\sigma,\nu_1\cup\nu_2)}$$

{...} ;
{...}

$$\frac{yields(\tau_1,\sigma,\nu_1\neq\emptyset),\ \forall\rho\in\nu_1 : yields(\tau_2,\sigma\bigoplus\rho,\nu_\rho)}{yields(\tau_1\circ\tau_2,\sigma,\bigcup_{\forall\rho\in\nu_1}\{\rho\}\bullet\nu_\rho)}$$

{...} , // $\nu_1 = \{\rho_1^{\nu_1}, \ldots \rho_n^{\nu_1}\} \bullet$
{...} // $\nu_2 = \{\rho_1^{\nu_2}, \ldots \rho_m^{\nu_2}\}$

$$\frac{yields(\tau_1,\sigma,\emptyset),\ yields(\tau_2,\sigma,\nu_2)}{yields(\tau_1\circ\tau_2,\sigma,\nu_2)}$$

$\nu_1 \bullet \nu_2 = \{\rho_1 \bullet \rho_2 | (\rho_1, \rho_2) \in \nu_1 \times \nu_2\}$
$\sigma \bigoplus \rho_1 \bullet \rho_2 \rightsquigarrow (\sigma \bigoplus \rho_1) \bigoplus \rho_2$

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

transition system over states $\sigma = \langle \sigma_1, \ldots, \sigma_n \rangle$ induced by updates $\oplus\psi$

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

$$\frac{\top}{yields(\mathbf{skip}, \sigma, \emptyset)} \qquad \frac{\top}{yields(\oplus\psi, \sigma, \{(\oplus, \psi)\})}$$

update **module** *[{...}]*

$$\frac{yields(\tau, \sigma, \nu), \sigma \models \phi}{yields(\phi \longrightarrow \tau, \sigma, \nu)} \qquad \frac{yields(\tau, \sigma, \nu), \sigma \not\models \phi}{yields(\phi \longrightarrow \tau, \sigma, \emptyset)}$$

when query **module** *[{...}]*
then {...}

$$\frac{yields(\tau_1, \sigma, \nu_1), yields(\tau_2, \sigma, \nu_2)}{yields(\tau_1 | \tau_2, \sigma, \nu_1 \cup \nu_2)}$$

{...} ;
{...}

$$\frac{yields(\tau_1, \sigma, \nu_1 \neq \emptyset), \forall \rho \in \nu_1 : yields(\tau_2, \sigma \bigoplus \rho, \nu_\rho)}{yields(\tau_1 \circ \tau_2, \sigma, \bigcup_{\forall \rho \in \nu_1} \{\rho\} \bullet \nu_\rho)}$$

{...} , // $\nu_1 = \{\rho_1^{\nu_1}, \ldots \rho_n^{\nu_1}\} \bullet$
{...} // $\nu_2 = \{\rho_1^{\nu_2}, \ldots \rho_m^{\nu_2}\}$

$$\frac{yields(\tau_1, \sigma, \emptyset), yields(\tau_2, \sigma, \nu_2)}{yields(\tau_1 \circ \tau_2, \sigma, \nu_2)}$$

$$\nu_1 \bullet \nu_2 = \{\rho_1 \bullet \rho_2 | (\rho_1, \rho_2) \in \nu_1 \times \nu_2\}$$

$$\sigma \bigoplus \rho_1 \bullet \rho_2 \rightsquigarrow (\sigma \bigoplus \rho_1) \bigoplus \rho_2$$

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

transition system over states $\sigma = \langle \sigma_1, \ldots, \sigma_n \rangle$ induced by updates $\oplus\psi$

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

$$\frac{\top}{yields(\mathbf{skip},\sigma,\emptyset)} \qquad \frac{\top}{yields(\oplus\psi,\sigma,\{(\oplus,\psi)\})}$$

update **module** *[{...}]*

$$\frac{yields(\tau,\sigma,\nu),\ \sigma\models\phi}{yields(\phi\longrightarrow\tau,\sigma,\nu)} \qquad \frac{yields(\tau,\sigma,\nu),\ \sigma\not\models\phi}{yields(\phi\longrightarrow\tau,\sigma,\emptyset)}$$

when query **module** *[{...}]*
then {...}

$$\frac{yields(\tau_1,\sigma,\nu_1),\ yields(\tau_2,\sigma,\nu_2)}{yields(\tau_1|\tau_2,\sigma,\nu_1\cup\nu_2)}$$

{...} ;
{...}

$$\frac{yields(\tau_1,\sigma,\nu_1\neq\emptyset),\ \forall\rho\in\nu_1: yields(\tau_2,\sigma\bigoplus\rho,\nu_\rho)}{yields(\tau_1\circ\tau_2,\sigma,\bigcup_{\forall\rho\in\nu_1}\{\rho\}\bullet\nu_\rho)}$$

{...} , // $\nu_1 = \{\rho_1^{\nu_1}, \ldots \rho_n^{\nu_1}\}$ $\bullet$
{...} // $\nu_2 = \{\rho_1^{\nu_2}, \ldots \rho_m^{\nu_2}\}$

$$\frac{yields(\tau_1,\sigma,\emptyset),\ yields(\tau_2,\sigma,\nu_2)}{yields(\tau_1\circ\tau_2,\sigma,\nu_2)}$$

$$\nu_1 \bullet \nu_2 = \{\rho_1 \bullet \rho_2 | (\rho_1, \rho_2) \in \nu_1 \times \nu_2\}$$
$$\sigma \bigoplus \rho_1 \bullet \rho_2 \rightsquigarrow (\sigma \bigoplus \rho_1) \bigoplus \rho_2$$

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

transition system over states $\sigma = \langle \sigma_1, \ldots, \sigma_n \rangle$ induced by updates $\oplus\psi$

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

$$\frac{\top}{yields(\mathbf{skip},\sigma,\emptyset)} \qquad \frac{\top}{yields(\oplus\psi,\sigma,\{(\oplus,\psi)\})}$$

update **module** *[{...}]*

$$\frac{yields(\tau,\sigma,\nu),\ \sigma\models\phi}{yields(\phi\longrightarrow\tau,\sigma,\nu)} \qquad \frac{yields(\tau,\sigma,\nu),\ \sigma\not\models\phi}{yields(\phi\longrightarrow\tau,\sigma,\emptyset)}$$

when query **module** *[{...}]*
then {...}

$$\frac{yields(\tau_1,\sigma,\nu_1),\ yields(\tau_2,\sigma,\nu_2)}{yields(\tau_1|\tau_2,\sigma,\nu_1\cup\nu_2)}$$

{...} ;
{...}

$$\frac{yields(\tau_1,\sigma,\nu_1\neq\emptyset),\ \forall\rho\in\nu_1 : yields(\tau_2,\sigma\bigoplus\rho,\nu_\rho)}{yields(\tau_1\circ\tau_2,\sigma,\bigcup_{\forall\rho\in\nu_1}\{\rho\}\bullet\nu_\rho)}$$

{...} , // $\nu_1 = \{\rho_1^{\nu_1}, \ldots \rho_n^{\nu_1}\} \bullet$
{...} // $\nu_2 = \{\rho_1^{\nu_2}, \ldots \rho_m^{\nu_2}\}$

$$\frac{yields(\tau_1,\sigma,\emptyset),\ yields(\tau_2,\sigma,\nu_2)}{yields(\tau_1\circ\tau_2,\sigma,\nu_2)}$$

$$\nu_1 \bullet \nu_2 = \{\rho_1 \bullet \rho_2 | (\rho_1, \rho_2) \in \nu_1 \times \nu_2\}$$
$$\sigma \bigoplus \rho_1 \bullet \rho_2 \rightsquigarrow (\sigma \bigoplus \rho_1) \bigoplus \rho_2$$

Semantics cont.

BSM computation step

A BSM $\mathcal{A} = (\mathcal{M}_1, \ldots, \mathcal{M}_n, \mathcal{P})$ induces a transition $\sigma \rightarrow \sigma'$ iff the program $\mathcal{P}$ yields an update set $\nu \neq \emptyset$ in σ, $\oslash\psi \in \nu$ is an update and $\sigma' = \sigma \bigoplus \oslash\psi$.

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.

BSM semantics (functional view)

$yields(\tau, \sigma, \nu) \rightsquigarrow$ a set of *enabled* updates in states

code modularity

Semantics cont.

BSM computation step

A BSM $\mathcal{A} = (\mathcal{M}_1, \ldots, \mathcal{M}_n, \mathcal{P})$ induces a transition $\sigma \rightarrow \sigma'$ iff the program $\mathcal{P}$ yields an update set $\nu \neq \emptyset$ in σ, $\oslash\psi \in \nu$ is an update and $\sigma' = \sigma \bigoplus \oslash\psi$.

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.

BSM semantics (functional view)

$yields(\tau, \sigma, \nu) \rightsquigarrow$ a set of *enabled* updates in states

code modularity

Semantics cont.

BSM computation step

A BSM $\mathcal{A} = (\mathcal{M}_1, \ldots, \mathcal{M}_n, \mathcal{P})$ induces a transition $\sigma \to \sigma'$ iff the program $\mathcal{P}$ yields an update set $\nu \neq \emptyset$ in σ, $\oslash\psi \in \nu$ is an update and $\sigma' = \sigma \bigoplus \oslash\psi$.

BSM semantics (operational view)

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

BSM semantics (functional view)

$yields(\tau, \sigma, \nu) \rightsquigarrow$ a set of *enabled* updates in states

code modularity

Abstract interpreter

input: agent program $\mathcal{P}$, initial mental state state σ_0

$\sigma = \sigma_0$
loop
  compute ν, s.t. $yields(\mathcal{P}, \sigma, \nu)$
  if $\nu \neq \emptyset$ **then**
    non-deterministically choose $\rho \in \nu$
    $\sigma = \sigma \oplus \rho$
  end if
end loop

Example: office space robot

```
/* Initialization */
declare module beliefs as ASP
declare module goals as Prolog
declare module body as Java
/* initializations omitted */

/* Default operation */
when sense body [{ (Battery.status() == OK) }] then {
  /* Roam around */
  perform body [{ Motors.turn(Rnd.get(), Rnd.get()) }] ;
  perform body [{ Motors.stepForward() }]
} else
{
  /* Safe emergency mode — degrade gracefully */
  perform body [{ Face.smile(off) }] ;
  perform body [{ InfraEye.switch(off)}] ;
  update goals [{ assert(dock) }]
} ;
```

```
/* Interruption handling */
when sense body (X) [{ Visual.see(X) }] and
     query beliefs (X) [{ friend(X), not met(X) }]
then {
  perform body [{ Face.smile(on) }] ,
  perform body [{ Audio.say("Hello!") }] ,
  adopt beliefs (X) [{ met(X) }]
}
```

Jazzbot

Bot in a simulated 3D environment.

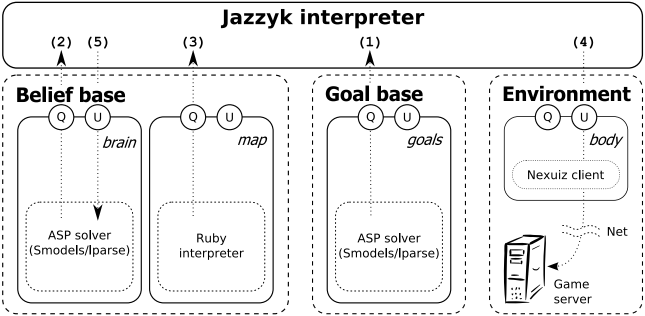
A case study application/proof-of-concept for *BSM/Jazzyk*.

- test-bed for various KR technologies in agent setting
- test-bed for applications of *evolving KBs/LP updates* in a dynamic, single agent scenario (DLP)
- challenging, dynamic, unstructured and rich environment

Jazzbot overview

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)
}
```



Summary, open issues

Behavioural State Machines & Jazzyk

An implemented hybrid programming framework with clear separation between *knowledge representation* and *agent's behaviours*.

- *heterogeneous KR* technologies
- *clear semantics*
- *source-code modularity*: code blocks, macro pre-processor
- easy *integration* with external systems/environments

Hot topic:

- *methodology* $\rightsquigarrow$ How to build such agents? What about BDI? Goal-oriented behaviours?

Summary, open issues

Behavioural State Machines & Jazzyk

An implemented hybrid programming framework with clear separation between *knowledge representation* and *agent's behaviours*.

- *heterogeneous KR* technologies
- *clear semantics*
- *source-code modularity*: code blocks, macro pre-processor
- easy *integration* with external systems/environments

Hot topic:

- *methodology* $\rightsquigarrow$ How to build such agents? What about BDI? Goal-oriented behaviours?

Related work

AgentSpeak(L) family - Jason, 3APL, 2APL, GOAL, etc.

- rule based syntax
- operational semantics: transition system

IMPACT - framework for heterogeneous agent systems

- strong KR modularity
- differences in semantics, code modularity, scripting

On-going and future work

methodology

- higher level programming constructs:
  - *goals*
  - *commitment strategies*
  - modal logics(?)

Jazzbot

- towards a public system demonstration
- *goal-oriented* behaviours
- *multi-agent setting* $\rightsquigarrow$ communication, richer LP updates, agent platform integration

Thank you for your attention.

http://jazzyk.sourceforge.net/