Code patterns for agent-oriented programming

Peter Novák[1] and Wojciech Jamroga[1,2]

[1]Clausthal University of Technology
[2]University of Luxembourg

Wednesday, May 13, 2009
AAMAS 2009, Budapest,Hungary

Motivation

- reactivity vs. deliberation $\rightsquigarrow$ hybrid architectures $\rightsquigarrow$ BDI
- programming with mental attitudes: beliefs, goals, etc.

agent oriented programming languages

1. choose a set of agent-oriented features
2. implement the set in the language interpreter

- fixed set of language constructs
- fixed architecture of created agent systems

extensions require changes of the language semantics
$\Rightarrow$ adaptation of the interpreter

Motivation

- reactivity vs. deliberation $\rightsquigarrow$ hybrid architectures $\rightsquigarrow$ BDI
- programming with mental attitudes: beliefs, goals, etc.

agent oriented programming languages

1. choose a set of agent-oriented features
2. implement the set in the language interpreter

- fixed set of language constructs
- fixed architecture of created agent systems

extensions require changes of the language semantics
$\Rightarrow$ adaptation of the interpreter

Motivation

- reactivity vs. deliberation $\rightsquigarrow$ hybrid architectures $\rightsquigarrow$ BDI
- programming with mental attitudes: beliefs, goals, etc.

agent oriented programming languages

1. choose a set of agent-oriented features
2. implement the set in the language interpreter

- fixed set of language constructs
- fixed architecture of created agent systems

extensions require changes of the language semantics
$\Rightarrow$ adaptation of the interpreter

Problem & the way to go...

How to design
extensible programming languages
for cognitive agents.

?

How to develop domain independent high level language constructs for programming with mental attitudes?

generic language for reactive systems + dynamic temporal logic ⇝ domain independent code patterns

Problem & the way to go...

How to design
extensible programming languages
for cognitive agents.

?

How to develop domain independent high level language constructs for programming with mental attitudes?

generic language for reactive systems + **dynamic temporal logic** $\rightsquigarrow$ **domain independent code patterns**

Behavioural State Machines

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

- heterogeneous knowledge bases
- structured source code, macros

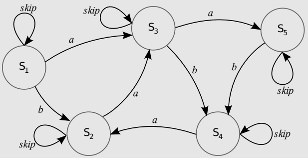
the core $\rightsquigarrow$ KR module $\mathcal{M}$
BSM agent system $\rightsquigarrow$ $\mathcal{A} = (\mathcal{M}_1, \ldots, \mathcal{M}_n, \mathcal{P})$

```
/* PICK an item behaviour */
when ⊨_G [{ task(pick(X)) }] and ⊨_B [{ see(X) }] then {
        when ⊨_B [{ dir(X, Angle) }] then ⊘_E [{ turn Angle }] |        /* either turn to the item, or */
        when ⊨_B [{ dir(X,'ahead'), dist(X,Dist) }] then {              /* pick up the item */
                ⊘_E [{ move forward Dist }] ∘
                ⊕_B [{ holds(X) }]
        }
}
```

BSM semantics

BSM $\rightsquigarrow$ labelled transition system

- *operational* $\rightsquigarrow$ computation runs



run of $\mathcal{P}$ $\quad \lambda = \underbrace{s_1 \xrightarrow{a} s_3 \xrightarrow{b} s_4}_{\mathcal{P}_1} \underbrace{\xrightarrow{\text{skip}} s_4 \xrightarrow{a} s_2}_{\mathcal{P}_2} \longrightarrow \cdots$

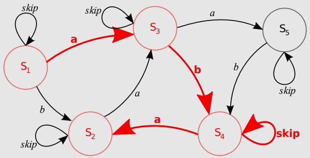
reasoning about computation runs:

$\rightsquigarrow$ *a logic interpreted over the same structure!*

BSM semantics

BSM $\rightsquigarrow$ labelled transition system

- *operational* $\rightsquigarrow$ computation runs



run of $\mathcal{P}$ $\quad \lambda = \underbrace{s_1 \xrightarrow{a} s_3 \xrightarrow{b} s_4}_{\mathcal{P}_1} \underbrace{\xrightarrow{\textbf{skip}} s_4 \xrightarrow{a} s_2}_{\mathcal{P}_2} \rightarrow \cdots$

reasoning about computation runs:

$\rightsquigarrow$ *a logic interpreted over the same structure!*

DCTL*=Dynamic Logic + CTL*

$$\theta ::= \mathsf{p} \mid \neg\theta \mid \theta \wedge \theta \mid [\tau]\varphi$$

$$\varphi ::= \theta \mid \neg\varphi \mid \varphi \wedge \varphi \mid \bigcirc \varphi \mid \varphi \,\mathcal{U}\, \varphi \mid \varphi \,\mathcal{C}\, \varphi$$

$[\tau]\varphi \leadsto$ *during execution of τ, φ holds*

From BSM to DCTL*: annotations $\mathfrak{A}$

Annotated BSM $\mathcal{A}^{\mathfrak{A}} = (\mathcal{M}_1, \ldots, \mathcal{M}_n, \mathcal{P}, \mathfrak{A})$

$$\mathfrak{A} : \mathcal{Q}(\mathcal{A}) \cup \tau(\mathcal{A}) \rightarrow DCTL^*$$

from subprograms to complex programs $\leadsto$ aggregation

semantic characterization $\leadsto$ the key to code re-usability

DCTL*=Dynamic Logic + CTL*

$$\theta ::= \mathsf{p} \mid \neg\theta \mid \theta \wedge \theta \mid [\tau]\varphi$$

$$\varphi ::= \theta \mid \neg\varphi \mid \varphi \wedge \varphi \mid \bigcirc\varphi \mid \varphi\,\mathcal{U}\,\varphi \mid \varphi\,\mathcal{C}\,\varphi$$

$[\tau]\varphi \rightsquigarrow$ *during execution of* τ, φ *holds*

From BSM to DCTL*: annotations $\mathfrak{A}$

Annotated BSM $\mathcal{A}^{\mathfrak{A}} = (\mathcal{M}_1, \ldots, \mathcal{M}_n, \mathcal{P}, \mathfrak{A})$

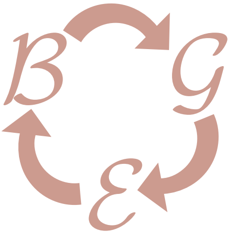
$$\mathfrak{A} : \mathcal{Q}(\mathcal{A}) \cup \tau(\mathcal{A}) \rightarrow DCTL^*$$

from subprograms to complex programs $\rightsquigarrow$ aggregation

semantic characterization $\rightsquigarrow$ the key to code re-usability

Agent system architecture

$$\mathcal{A} = (\mathcal{B}, \mathcal{G}, \mathcal{E}, \mathcal{P})$$



robot in a 3D environment: search & deliver

Structure:

$\mathcal{B}$: belief base $(\models_{\mathcal{B}}, \oplus_{\mathcal{B}}, \ominus_{\mathcal{B}})$

$\mathcal{G}$: goal base $(\models_{\mathcal{G}}, \oplus_{\mathcal{G}}, \ominus_{\mathcal{G}})$

$\mathcal{E}$: interface to the environment $\rightsquigarrow$ body $(\models_{\mathcal{E}}, \oslash_{\mathcal{E}})$

Basic capabilities:

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

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

BSM design patterns: TRIGGER

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

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

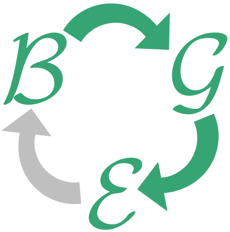
BSM design patterns: ADOPT/DROP

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

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

BSM design patterns: ACHIEVE



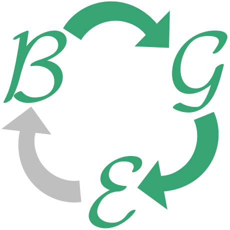
define ACHIEVE($\varphi_{\mathbf{G}}, \varphi_{\mathbf{B}}, \psi_\oplus, \psi_\ominus, \tau$)
  TRIGGER($\varphi_{\mathbf{G}}, \tau$) |
  ADOPT($\varphi_{\mathbf{G}}, \psi_\oplus$) |
  DROP($\varphi_{\mathbf{G}}, \varphi_{\mathbf{B}}$) |
  DROP($\varphi_{\mathbf{G}}, \psi_\ominus$)
end

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

running example cont.

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

BSM design patterns: MAINTAIN



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

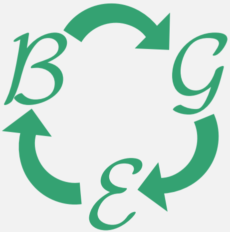
$$\mathfrak{A}(\models_{\mathcal{G}} \varphi_{\mathbf{G}}) \rightarrow [\mathsf{MAINTAIN}(\varphi_{\mathbf{G}}, \varphi_{\mathbf{B}}\tau)^*]\Box(\neg\mathfrak{A}(\models_{\mathcal{B}} \varphi_{\mathbf{B}}) \rightarrow \Diamond\mathfrak{A}(\models_{\mathcal{B}} \varphi_{\mathbf{B}}))$$

running example cont.

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

Putting it altogether

Robot program



PERCEIVE $\circ$
{
  MAINTAIN(
    $maintain(keep\_safe)$,
    $threatened$,
    RUN_AWAY) |

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

Summary

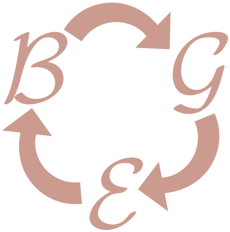
different applications require different programming constructs

extensible agent oriented programming languages

purely syntactic approach to development of arbitrary high level programming constructs

Thank you for your attention.

http://jazzyk.sourceforge.net/



see you at the poster session...