Code patterns for agent-oriented programming

Peter Novák¹ and Wojciech Jamroga¹,²

¹Clausthal University of Technology
²University of Luxembourg

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

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

\[ \downarrow \]

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 $\rightarrow$ hybrid architectures $\rightarrow$ 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 $\leadsto$ hybrid architectures $\leadsto$ 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 ⇝ 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** $\leadsto$ KR module $\mathcal{M}$

**BSM agent system** $\leadsto$ $\mathcal{A} = (\mathcal{M}_1, \ldots, \mathcal{M}_n, \mathcal{P})$

```c
/*@ PICK an item behaviour */
when $\models_\mathcal{G} \{\text{task(pick(X))}\}$ and $\models_\mathcal{B} \{\text{see(X)}\}$ then {
  when $\models_\mathcal{B} \{\text{dir(X, Angle)}\}$ then $\mathcal{E}$ \{\text{turn Angle}\} |
  when $\models_\mathcal{B} \{\text{dir(X,’ahead’), dist(X,Dist)}\}$ then {
    $\mathcal{E}$ \{\text{move forward Dist}\} $\circ$
    $\mathcal{B}$ \{\text{holds(X)}\}
  }
}

/*@ either turn to the item, or */
/*@ pick up the item */
```

P. Novák, W. Jamroga · Clausthal University of Technology, Germany

May 13th, 2009, Budapest, Hungary

4/14
BSM semantics

BSM $\leadsto$ labelled transition system

- *operational* $\leadsto$ computation runs

run of $\mathcal{P}$

$\lambda = s_1 \xrightarrow{a} s_3 \xrightarrow{b} s_4 \xrightarrow{\text{skip}} s_4 \xrightarrow{a} s_2 \rightarrow \cdots$

reasoning about computation runs:

$\leadsto$ a logic interpreted over the same structure!
BSM semantics

**BSM \(\rightsimeq\) labelled transition system**

- **operational** \(\rightsimeq\) computation runs

![Diagram of a labelled transition system with states and transitions labeled with 'skip', 'a', and 'b'.]

**run of** \(\mathcal{P}\)

\[
\lambda = s_1 \xrightarrow{a} s_3 \xrightarrow{b} s_4 \xrightarrow{\text{skip}} s_4 \xrightarrow{a} s_2 \xrightarrow{\cdot\cdot\cdot}
\]

**reasoning about computation runs:**

\(\rightsimeq\) *a logic interpreted over the same structure!*
DCTL* = Dynamic Logic + CTL*

\[ \theta ::= p \mid \neg \theta \mid \theta \land \theta \mid [\tau] \varphi \]

\[ \varphi ::= \theta \mid \neg \varphi \mid \varphi \land \varphi \mid \bigcirc \varphi \mid \varphi U \varphi \mid \varphi C \varphi \]

\[ [\tau] \varphi \leadsto during \ execution \ of \ \tau, \ \varphi \ holds \]

From BSM to DCTL*: annotations \( \mathcal{A} \)

Annotated BSM \( A^\mathcal{A} = (M_1, \ldots, M_n, P, \mathcal{A}) \)

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

from subprograms to complex programs \( \leadsto \) aggregation

semantic characterization \( \leadsto \) the key to code re-usability
DCTL* = Dynamic Logic + CTL*

\[ \theta ::= p \mid \neg \theta \mid \theta \land \theta \mid [\tau] \varphi \]

\[ \varphi ::= \theta \mid \neg \varphi \mid \varphi \land \varphi \mid \bigcirc \varphi \mid \varphi U \varphi \mid \varphi C \varphi \]

\[ [\tau] \varphi \Rightarrow \text{during execution of } \tau, \varphi \text{ holds} \]

From BSM to DCTL*: annotations \( \mathcal{A} \)

Annotated BSM \( \mathcal{A}^{\mathcal{A}} = (M_1, \ldots, M_n, \mathcal{P}, \mathcal{A}) \)

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

from subprograms to complex programs \( \Rightarrow \) aggregation

semantic characterization \( \Rightarrow \) the key to code re-usability
Agent system architecture

\[ A = (B, G, E, P) \]

**Structure:**

- **B**: belief base \( (\models B, \oplus B, \ominus B) \)
- **G**: goal base \( (\models g, \oplus g, \ominus g) \)
- **E**: interface to the environment \( \sim \) body \( (\models E, \ominus E) \)

**Basic capabilities:**

- **FIND**: \([\text{FIND}] A(\text{FIND}) \Rightarrow [\text{FIND}^*] \diamond \text{holds(item42)}\)
- **RUN_AWAY**: \([\text{RUN}_A\text{WAY}] A(\text{RUN}_A\text{WAY}) \Rightarrow [\text{RUN}_A\text{WAY}^*] \diamond \text{safe}\)
BSM design patterns: TRIGGER

define TRIGGER(\( \varphi_G, \tau \))
  when \| G \varphi_G \| then \tau
end

\( A(\| G \varphi_G \|) \rightarrow [\text{TRIGGER}(\varphi_G, \tau)^*] \diamond A(\tau) \)
BSM design patterns: ADOPT/DROP

**define** ADOPT(\(\varphi_G, \psi_\oplus\))

\[\text{when } |_B \psi_\oplus \text{ and not } |_G \varphi_G \text{ then } \oplus_G \varphi_G\]

**end**

**define** DROP(\(\varphi_G, \psi_\ominus\))

\[\text{when } |_B \psi_\ominus \text{ and } |_G \varphi_G \text{ then } \ominus_G \varphi_G\]

**end**

\[\mathcal{A}(|_B \psi_\oplus) \rightarrow [\text{ADOPT}(\varphi_G, \psi_\oplus)^*] \diamond \mathcal{A}(|_G \varphi_G)\]

\[\mathcal{A}(|_B \psi_\ominus) \rightarrow [\text{DROP}(\varphi_G, \psi_\ominus)^*] \diamond \neg \mathcal{A}(|_G \varphi_G)\]
BSM design patterns: ACHIEVE

define ACHIEVE(\(\varphi_G, \varphi_B, \psi\oplus, \psi\ominus, \tau\))

TRIGGER(\(\varphi_G, \tau\)) \mid
ADOPT(\(\varphi_G, \psi\oplus\)) \mid
DROP(\(\varphi_G, \varphi_B\)) \mid
DROP(\(\varphi_G, \psi\ominus\))

end

[ACHIEVE(\(\varphi_G, \varphi_B, \psi\oplus, \psi\ominus, \tau\))^*]A(\(\models_G \varphi_G\)) U A(\(\models_B \varphi_B \lor \models_B \psi\ominus\))

running example cont.

ACHIEVE(
   achieve(has(item42)),
   holds(item42),
   needs(item42),
   \neg needs(item42) \lor \neg exists(item42),
   FIND)
BSM design patterns: MAINTAIN

define MAINTAIN(ϕ_G, ϕ_B, τ)
when not \( \models_B ϕ_B \) then TRIGGER(ϕ_G, τ) | ADOPT(ϕ_G, ⊤)
end

\[ \mathcal{A}(\models_G ϕ_G) \rightarrow [\text{MAINTAIN(ϕ_G, ϕ_B τ)^*}] \square(\neg \mathcal{A}(\models_B ϕ_B) \rightarrow \Diamond \mathcal{A}(\models_B ϕ_B)) \]

running example cont.

MAINTAIN(maintain(keep_safe), safe, RUN_AWAY)
Putting it altogether

Robot program

PERCEIVE ○
{
  MAINTAIN(
    maintain(keep_safe),
    threatened,
    RUN_AWAY) |

  ACHIEVE(
    achieve(has(item42)),
    holds(item42),
    needs(item42),
    ¬needs(item42) ∨ ¬exists(item42),
    FIND)
}

P. Novák, W. Jamroga · Clausthal University of Technology, Germany
May 13th, 2009, Budapest, Hungary
Summary

different applications require different programming constructs

extensible agent oriented programming languages

purely syntactic approach to development of arbitrary high level programming constructs
Conclusion

Thank you for your attention.

http://jazzyk.sourceforge.net/

see you at the poster session...