TU Clausthal
Clausthal University of Technology

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

Code Patterns for Agent-Oriented Programming

Designing an agent programming language

state of the art design principle

choose a set of agent-oriented features with a particular semantics ⇝ implement that features set in a language interpreter

advantages & shortcomings

- ⊞ clear semantics (logic-based)
- ⊟ fixed internal architecture of the agent system
  - fixed set of knowledge bases
  - fixed KR technology
- ⊟ fixed set of language constructs for implementation of agent's mental attitudes (beliefs, goals, etc.)
- ⊟ every language extension requires change to the language semantics
  - ⇝ adaptation of the interpreter
- ⊟ and yet, usually a problematic connection to BDI style logics, such as [Rao & Georgeff], or [Cohen & Levesque]

Engineering extensible agent-oriented programming languages...

Our way to go...

generic language for reactive systems

Behavioural State Machines

- horizontal modularity
- vertical modularity
- simple & clear semantics
- integrated macro preprocessor
  - ⇝ extensible syntax

+

dynamic temporal logic for the language

DCTL*: Dynamic Logic + CTL*

- DCTL formula: $[\tau]\varphi$
  - ⇝ during execution of τ, φ holds
- temporal anotations of subprograms
- program verification

=

BDI code patterns

- ⊞ re-usable design patterns
  - ⇝ clear semantic description
  - ⇝ pattern library
- ⊞ purely syntactic approach
- ⊞ application domain independent
- ⊞ implements core BDI constructs
  - *achievement goal*
  - *maintenance goal*

Behavioural State Machines

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

KR module: abstract knowledge base

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

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

Syntax: mental state transformers (mst)

- **skip** is a *primitive* mst,
- $\oslash_i\psi$ is a *primitive* mst
- $\models_i\varphi \longrightarrow \tau$ is a *conditional* mst
- $\tau_1|\tau_2$ and $\tau_1 \circ \tau_2$ are *choice* and *sequence* mst's
- $\{\tau\}$ is a *block* mst

Semantics: *yields* calculus

$$\frac{}{yields(\mathsf{skip},\sigma,\mathsf{skip})}\top \qquad \frac{}{yields(\oslash_i\psi,\sigma,\oslash_i\psi)}\top \qquad \frac{yields(\tau,\sigma,\rho)}{yields(\{\tau\},\sigma,\rho)}$$

$$\frac{yields(\tau,\sigma,\rho),\ \sigma\models_i\phi}{yields(\models_i\phi\longrightarrow\tau,\sigma,\rho)} \qquad \frac{yields(\tau,\sigma,\rho),\ \sigma\not\models_i\phi}{yields(\models_i\phi\longrightarrow\tau,\sigma,\mathsf{skip})}$$

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

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

Denotational view: mst τ as a function $\tau \rightsquigarrow \mathfrak{f}_\tau$

$\mathfrak{f}_\tau : \sigma \mapsto \{\rho | yields(\tau,\sigma,\rho)\}$

$\sigma \xrightarrow{\rho} \sigma'$ iff $\sigma' = \sigma \oplus \rho$ $\rho \in \mathfrak{f}_\tau$ in σ, i.e. $yields(\mathcal{P},\sigma,\rho)$

Operational view: set of computation runs

Agent system $\mathcal{A}$ (*BSM*) ⇝ the set of all runs

$\sigma_0 \xrightarrow{\rho_1} \sigma_1 \xrightarrow{\rho_2} \ldots \xrightarrow{\rho_k} \sigma_k$

$\mathcal{T}(\overline{\mathcal{A}}, \tau^*)$

Jazzyk — *BSM* programming language

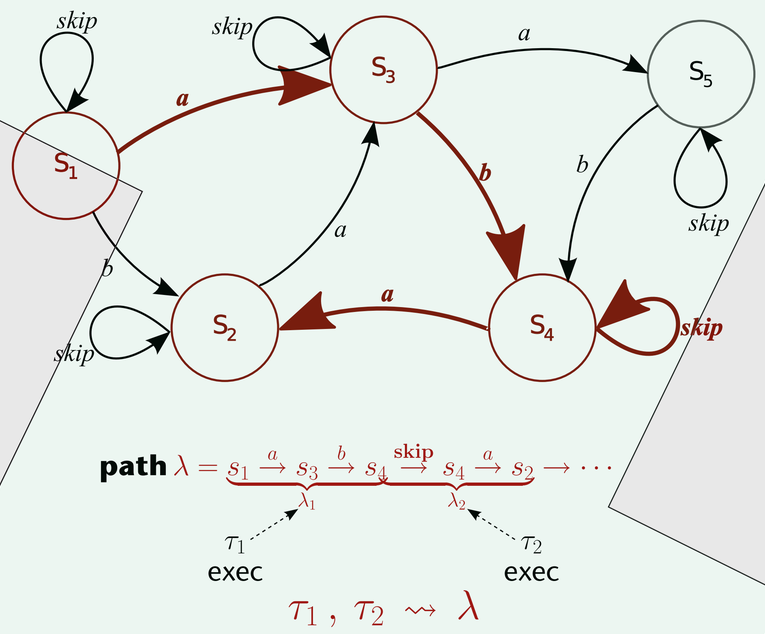
- vertical modularity ⇝ KR plug-ins
- horizontal modularity ⇝ hierarchical structure
- integrated macro preprocessor

Common semantic structure

Labeled Transition System $LTS(\mathcal{A})$

states: $\sigma = \langle\sigma_1, \ldots, \sigma_n\rangle$

transitions: $\oslash\psi \in \mathfrak{f}_\tau = \{\rho | yields(\tau,\sigma,\rho)\}$



path $\lambda = s_1 \xrightarrow{a} s_3 \xrightarrow{b} s_4 \xrightarrow{\mathsf{skip}} s_4 \xrightarrow{a} s_2 \to \cdots$ (λ_1, λ_2)

exec τ_1 exec τ_2

$\tau_1\ ,\ \tau_2 \rightsquigarrow \lambda$

DCTL*: Logics for BSM

$[\tau]\Diamond\varphi$ ⇝ *on every run generated by execution of τ, eventually φ holds*

Syntax — DL + CTL*

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

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

Semantics

$\mathcal{A},\lambda \models \mathsf{p}$	**iff** $\lambda[0] \in \pi(\mathsf{p})$ in $\overline{LTS}(\mathcal{A})$	LTL
$\mathcal{A},\lambda \models \neg\varphi$	**iff** $\mathcal{A},\lambda \not\models \varphi$	
$\mathcal{A},\lambda \models \varphi_1 \wedge \varphi_2$	**iff** $\mathcal{A},\lambda \models \varphi_1$ and $\mathcal{A},\lambda \models \varphi_2$	
$\mathcal{A},\lambda \models \bigcirc\varphi$	**iff** $\mathcal{A},\lambda[1..\infty] \models \varphi$	
$\mathcal{A},\lambda \models \varphi_1\,\mathcal{U}\,\varphi_2$	**iff** there exists $i \geq 0$, s.t. $\mathcal{A},\lambda[i..\infty] \models \varphi_2$, and $\mathcal{A},\lambda[j..\infty] \models \varphi_1$ for every $0 \leq j < i$	
$\mathcal{A},\lambda \models \varphi_1\,\mathcal{C}\,\varphi_2$	**iff** there exists $i \geq 0$, s.t. $\mathcal{A},\lambda[0..i] \models \varphi_1$ and $\mathcal{A},\lambda[i..\infty] \models \varphi_2$	
$\mathcal{A},\lambda \models \theta$	**iff** $\mathcal{A},\lambda[0] \models \theta$;	CTL*
$\mathcal{A},\sigma \models \mathsf{p}$	**iff** $\sigma \in \pi(\mathsf{p})$	
$\mathcal{A},\sigma \models \neg\theta$	**iff** $\mathcal{A},\sigma \not\models \theta$	
$\mathcal{A},\sigma \models \theta_1 \wedge \theta_2$	**iff** $\mathcal{A},\sigma \models \theta_1$ and $\mathcal{A},\sigma \models \theta_2$	
$(\mathcal{A},\mathcal{P}),\sigma \models [\tau]\varphi$	**iff** for every trace $\lambda \in \mathcal{T}(\mathcal{A},\tau)$, s.t. $\lambda[0] = \sigma$, we have $(\mathcal{A},\tau),\lambda \models \varphi$	DL

derived modalities: $\langle\tau\rangle, \Diamond, \Box, \ldots$ — **LTL⊂DCTL***

Program annotations: intended functionality

from heterogeneous KR languages to a single one

Annotation function $\mathfrak{A}$

- queries: $\mathfrak{A} : \mathcal{Q}(\mathcal{A}) \to LTL$
- updates: $\mathfrak{A} : \mathcal{U}(\mathcal{A}) \to LTL$
- mst's: $\mathfrak{A} : \tau \mapsto LTL$

Annotation aggregation: bottom-up

extract program characterization from component mst's annotations

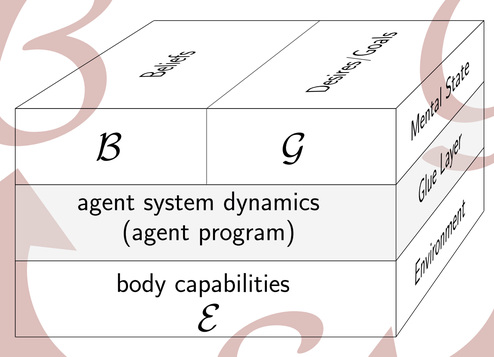
- query: $\mathfrak{A}(\neg\phi) = \neg\mathfrak{A}(\phi)$, $\mathfrak{A}(\phi_1 \wedge \phi_2) = \mathfrak{A}(\phi_1) \wedge \mathfrak{A}(\phi_2)$, and $\mathfrak{A}(\phi_1 \vee \phi_2) = \mathfrak{A}(\phi_1) \vee \mathfrak{A}(\phi_2)$
- conditional: $\mathfrak{A}(\phi \longrightarrow \tau) = \mathfrak{A}(\phi) \to \mathfrak{A}(\tau)$
- compound: $\mathfrak{A}(\tau_1|\tau_2) = \mathfrak{A}(\tau_1) \vee \mathfrak{A}(\tau)$ and $\mathfrak{A}(\tau_1 \circ \tau_2) = \mathfrak{A}(\tau_1)\,\mathcal{C}\,\mathfrak{A}(\tau)$

Reasoning about annotations ⇝ verification

DCTL specification* — *LTL annotations* — *iterated agent program execution*

Modular BDI architecture

BDI instance of the BSM behavioural template



belief base $\mathcal{B}$
- $\mathcal{Q}_\mathcal{B} = \{\models_\mathcal{B}\}$
- $\mathcal{U}_\mathcal{B} = \{\oplus_\mathcal{B}, \ominus_\mathcal{B}\}$

goal base $\mathcal{G}$
- $\mathcal{Q}_\mathcal{G} = \{\models_\mathcal{G}\}$
- $\mathcal{U}_\mathcal{G} = \{\oplus_\mathcal{G}, \ominus_\mathcal{G}\}$

body $\mathcal{E}$
- $\mathcal{Q}_\mathcal{E} = \{\models_\mathcal{E}\}$
- $\mathcal{U}_\mathcal{E} = \{\oslash_\mathcal{E}, \ldots\}$

BDI code patterns

Example: *Jazzbot*

Tasks

1. find $item42$
2. deliver it to the base
3. run away from dangers & enemies

Capabilities

- $\mathcal{B}$, $\mathcal{G}$ implemented as *Prolog* KR modules
- two basic capabilities: FIND, RUN_AWAY

$[\mathrm{FIND}]\mathfrak{A}(\mathrm{FIND}) \Rightarrow [\mathrm{FIND}^*]\Diamond holds(item42)$
$[\mathrm{RUN\_AWAY}]\mathfrak{A}(\mathrm{RUN\_AWAY}) \Rightarrow [\mathrm{RUN\_AWAY}^*]\Diamond safe$

The bot wants to achieve a situation in which it posesses $item42$ and maintains its own safety.

TRIGGER($has(item42)$, FIND)
TRIGGER($keep\_safe$, RUN_AWAY)

Basic capabilities $\curvearrowright\mathcal{E}$

- possibly complex reactive behaviours
- flexible level of annotation abstraction

```
PERCEIVE ∘
{
MAINTAIN_GOAL(
        goal(keep_safe),
        safe,
        RUN_AWAY) |

 ACHIEVE_GOAL(
        goal(has(item42)),
        holds(item42),
        needs(item42),
        ¬needs(item42) ∨ ¬exists(item42),
        FIND)
}
```

Goal oriented behaviours $\mathcal{G}\curvearrowright\mathcal{E}$

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

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

Perceptions $\mathcal{E}\curvearrowright\mathcal{B}$

define PERCEIVE(φ_E, φ_B)
  when $\models_\mathcal{E}\varphi_\mathrm{E}$ **then** $\oplus_\mathcal{B}\varphi_\mathrm{B}$
end

$\mathfrak{A}(\models_\mathcal{E}\varphi_\mathrm{E}) \to [\mathrm{PERCEIVE}(\varphi_\mathrm{E},\psi_\mathrm{B})]$
$\bigcirc\mathfrak{A}(\oplus_\mathcal{B}\psi_\mathrm{B})$

Commitment strategies $\mathcal{B}\curvearrowright\mathcal{G}$

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

$\mathfrak{A}(\models_\mathcal{B}\psi_\oplus) \to [\mathrm{ADOPT}(\varphi_\mathrm{G},\psi_\oplus)^*]\Diamond\mathfrak{A}(\models_\mathcal{G}\varphi_\mathrm{G}))$

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

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

Achievement goal $\mathcal{B}\curvearrowright\mathcal{G}\curvearrowright\mathcal{E}$

define ACHIEVE(φ_G, φ_B, $\psi_\oplus$, $\psi_\ominus$, τ)
  TRIGGER(φ_G, τ) |
  ADOPT(φ_G, $\psi_\oplus$) |
  DROP(φ_G, φ_B) |
  DROP(φ_G, $\psi_\ominus$)
end

Assuming $[\tau]\mathfrak{A}(\tau) \Rightarrow [\tau^*]\Diamond\mathfrak{A}(\models_\mathcal{B}\varphi_\mathrm{B})$:

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

Maintenance goal $\mathcal{B}\curvearrowright\mathcal{G}\curvearrowright\mathcal{E}$

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

Assuming $[\tau]\mathfrak{A}(\tau) \Rightarrow [\tau^*]\Diamond\mathfrak{A}(\models_\mathcal{B}\varphi_\mathrm{B})$:

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