Adding structure to agent programming languages

(programming agents with mental states)

Peter Novák, Jürgen Dix

Clausthal University of Technology, Germany

May 15th, 2007
ProMAS'07, Honolulu, Hawai'i

Agenda

Problem

Ideal agent oriented programming language:

- **modularity**
  - knowledge representation
  - source code
- code encapsulation
- **decomposition** and **combination** $\rightsquigarrow$ compound structures
- code reuse!
- readability

The way to go...

1. basis for the minimalistic core language
   - *Modular BDI Architecture* [Novák and Dix, 2006]
   - IMPACT [Subrahmanian et. al., 2000]
2. structural extension:
   - *alternative semantical* view on the language
   - notion of *mental state transformer*
3. further extensions: *macros*

Modular core language: syntax

Definitions

Let $\mathcal{L}$ be a language:

- *mental state* is a theory $\sigma \subseteq \mathcal{L}$,
- *query language* $\mathcal{L}_Q$: $\varphi \in \mathcal{L}$, then $Q(\varphi) \in \mathcal{L}_Q$, $\top, \bot \in \mathcal{L}_Q$ and $\vee, \wedge, \neg$ are allowed in $\mathcal{L}_Q$,
- *update language* $\mathcal{L}_U$: $\varphi \in \mathcal{L}$, then $U(\varphi) \in \mathcal{L}_U$,
- let $\phi \in \mathcal{L}_Q$, $\psi \in \mathcal{L}_U$, then $\phi \longrightarrow \psi$ is a *transition rule*.
- *program* $\rightsquigarrow$ a set of transition rules

Core language: standard operational semantics

Definitions

- *abstract operators:*
  - $Query_{\mathcal{L}} : 2^{\mathcal{L}} \times \mathcal{L} \rightarrow \{true, false\}$, $(\sigma \models \varphi)$
  - $Update_{\mathcal{L}} : 2^{\mathcal{L}} \times \mathcal{L} \rightarrow 2^{\mathcal{L}}$, $(\sigma \oplus \psi = \sigma')$

Definition

Application of a rule $\phi \longrightarrow \psi$ in a state σ:

$$\frac{\sigma \models \phi, \sigma \oplus \psi = \sigma'}{\sigma \longrightarrow \sigma'}$$

Definition

Agent system semantics $\rightsquigarrow$ set of all possible *computation runs* $\sigma_0, \sigma_1 \ldots$ induced by the program $\mathcal{P}$.

Core language: standard operational semantics

Definitions

- *abstract operators:*
  - $Query_{\mathcal{L}} : 2^{\mathcal{L}} \times \mathcal{L} \rightarrow \{true, false\}$, $(\sigma \models \varphi)$
  - $Update_{\mathcal{L}} : 2^{\mathcal{L}} \times \mathcal{L} \rightarrow 2^{\mathcal{L}}$, $(\sigma \oplus \psi = \sigma')$

Definition

Application of a rule $\phi \longrightarrow \psi$ in a state σ:

$$\frac{\sigma \models \phi, \sigma \oplus \psi = \sigma'}{\sigma \longrightarrow \sigma'}$$

Definition

Agent system semantics $\rightsquigarrow$ set of all possible *computation runs* $\sigma_0, \sigma_1 \ldots$ induced by the program $\mathcal{P}$.

Core language: denotational semantics

Definition

Program $\mathcal{P}$ can be characterized by a *partial function*

$$\mathcal{F}_{\mathcal{P}} : 2^{\mathcal{L}} \times \mathcal{L}_U \longrightarrow 2^{\mathcal{L}}; \mathcal{F}_{\mathcal{P}}(\sigma, \psi) = \sigma'$$

- σ is such, that $\exists$ a rule $r = (\phi \longrightarrow \psi) \in \mathcal{P}$ and r is *applicable* in σ
- a corresponding set of states $\Sigma_{\mathcal{F}_{\mathcal{P}}} \subseteq 2^{2^{\mathcal{L}}}$ is an *application domain* of $\mathcal{F}_{\mathcal{P}}$

Program $\mathcal{P}$:

operational semantics: set of all the specified paths in the space of *all possible mental states*

denotational semantics: set of all the specified transitions between *classes of mental states*

Core language: denotational semantics

Definition

Program $\mathcal{P}$ can be characterized by a *partial function*

$$\mathcal{F}_{\mathcal{P}} : 2^{\mathcal{L}} \times \mathcal{L}_U \longrightarrow 2^{\mathcal{L}}; \mathcal{F}_{\mathcal{P}}(\sigma, \psi) = \sigma'$$

- σ is such, that $\exists$ a rule $r = (\phi \longrightarrow \psi) \in \mathcal{P}$ and r is *applicable* in σ
- a corresponding set of states $\Sigma_{\mathcal{F}_{\mathcal{P}}} \subseteq 2^{2^{\mathcal{L}}}$ is an *application domain* of $\mathcal{F}_{\mathcal{P}}$

Program $\mathcal{P}$:

operational semantics: set of all the specified paths in the space of *all possible mental states*

denotational semantics: set of all the specified transitions between *classes of mental states*

Mental state transformer

Definition

A mental state transformer τ characterized by a function $\mathcal{F}$:

1. **primitive mst:** $\tau = \{\phi \longrightarrow \psi\} \leadsto \mathcal{F}(\sigma, \psi) = Update_{\mathcal{L}}(\psi, \sigma)$ and $\Sigma_{\mathcal{F}} = \{\sigma | \sigma \models \phi\}$
2. **specialization:** $\tau = \{\phi \longrightarrow \tau'\} \leadsto \mathcal{F}(\sigma, \psi) = \mathcal{F}'(\sigma, \psi)$ and $\Sigma_{\mathcal{F}} = \{\sigma | \sigma \in \Sigma_{\mathcal{F}'} \wedge \sigma \models \phi\}$
3. **generalization:** $\tau = \tau' \cup \tau'' \leadsto$ $\mathcal{F}(\sigma, \psi) = \begin{cases} \mathcal{F}'(\sigma, \psi) & \textit{if } \sigma \in \Sigma_{\mathcal{F}'} \\ \mathcal{F}''(\sigma, \psi) & \textit{if } \sigma \in \Sigma_{\mathcal{F}''} \end{cases}$ and $\Sigma_{\mathcal{F}} = \Sigma_{\mathcal{F}'} \cup \Sigma_{\mathcal{F}''}$

Example

when *[{at(X,Y)}]* **and** not *[{desiredPosition(X,Y)}]* **and**
[{towards((X1,Y1),(X,Y))}] **and** not *[{obstacle(X1,Y1)}]*
then *[{stepTo(X1,Y1)}]*

when *[{at(X,Y)}]* **and** *[{goldAt(X,Y)}]* **and** *[{loaded}]*
then *[{broadcast(goldAt(X,Y))}]*

when not *[{batteryLow}]* **then** {
<moving to the desired position>
<communication>
}
when *[{batteryLow}]* **then** {
<moving to the desired position>
when $\top$ **then** *[{adopt(desiredPosition(X_R, Y_R))}]*
}

Example

when *[{at(X,Y)}]* **and** not *[{desiredPosition(X,Y)}]* **and**
[{towards((X1,Y1),(X,Y))}] **and** not *[{obstacle(X1,Y1)}]*
then *[{stepTo(X1,Y1)}]*

when *[{at(X,Y)}]* **and** *[{goldAt(X,Y)}]* **and** *[{loaded}]*
then *[{broadcast(goldAt(X,Y))}]*

when not *[{batteryLow}]* **then** {
<moving to the desired position>
<communication>
}
when *[{batteryLow}]* **then** {
<moving to the desired position>
when $\top$ **then** *[{adopt(desiredPosition(X_R, Y_R))}]*
}

Example

when *[{at(X,Y)}]* **and** not *[{desiredPosition(X,Y)}]* **and**
  [{towards((X1,Y1),(X,Y))}] **and** not *[{obstacle(X1,Y1)}]*
then *[{stepTo(X1,Y1)}]*

when *[{at(X,Y)}]* **and** *[{goldAt(X,Y)}]* **and** *[{loaded}]*
then *[{broadcast(goldAt(X,Y))}]*

when not *[{batteryLow}]* **then** {
  <moving to the desired position>
  <communication>
}
when *[{batteryLow}]* **then** {
  <moving to the desired position>
  when $\top$ **then** *[{adopt(desiredPosition(X_R, Y_R))}]*
}

```
define movementToPosition {
        when [{at(X,Y)}] and not [{desiredPosition(X,Y)}] and
             [{towards((X1,Y1),(X,Y))}] and not [{obstacle(X1,Y1)}]
        then [{stepTo(X1,Y1)}]
}

define communicateGold {
        when [{at(X,Y)}] and not [{goldAt(X,Y)}] and [{loaded}]
        then [{broadcast(goldAt(X,Y))}]
}
...
when not [{batteryLow}] then {
        movementToPosition /* 1 */
        communicateGold /* 2 */
} else {
        when not [{desiredPosition(X_R,Y_R)}] then {
                [{adopt(desiredPosition(X_R,Y_R))}] /* 3 */
        } else {
                movementToPosition /* 4 */
        }
}
```

Example: translation to the core language

/* 1 */

$$\neg Q(\textit{'batteryLow'}) \wedge Q(\textit{'at(X,Y)'}) \wedge \neg Q(\textit{'desiredPosition(X,Y)'}) \wedge$$
$$Q(\textit{'towards((X1,Y1),(X,Y))'}) \wedge \neg Q(\textit{'obstacle(X1,Y1)'})$$
$$\longrightarrow U(\textit{'stepTo(X1,Y1)'})$$

/* 2 */

$$\neg Q(\textit{'batteryLow'}) \wedge Q(\textit{'at(X,Y)'}) \wedge \neg Q(\textit{'goldAt(X,Y)'}) \wedge Q(\textit{'loaded'})$$
$$\longrightarrow U(\textit{'broadcast(goldAt(X,Y))'})$$

/* 3 */

$$\neg\neg Q(\textit{'batteryLow'}) \wedge \neg Q(\textit{'desiredPosition}(X_R, Y_R)\textit{'})$$
$$\longrightarrow U(\textit{'adopt(desiredPosition}(X_R, Y_R))\textit{'})$$

/* 4 */

$$\neg\neg Q(\textit{'batteryLow'}) \wedge \neg\neg Q(\textit{'desiredPosition}(X_R, Y_R)\textit{'}) \wedge Q(\textit{'at(X,Y)'}) \wedge$$
$$\neg Q(\textit{'desiredPosition(X,Y)'}) \wedge Q(\textit{'towards((X1,Y1),(X,Y))'}) \wedge \neg Q(\textit{'obstacle(X1,Y1)'})$$
$$\longrightarrow U(\textit{'stepTo(X1,Y1)'})$$

Related work

Modularity of rule-based agent programming languages:

3APL

- Dastani et. al.: *Enacting and deacting roles in agent programming*, 2004
- Riemsdijk et. al.: *Goal oriented modularity in agent programming*, 2006

AgentSpeak(L)

- Hübner et. al.: *Programming declarative goals using plan patterns*, 2006

GOAL

- Hindriks: *Modules as policies*, ProMAS 2007

Ongoing & future-work

- integration with BDI framework
  - modular BDI architecture
  - BDI rather a methodological guideline, than an explicit programming language?
- testing in real-world
  - Jazzyk language interpreter $\rightsquigarrow$ working prototype(!)
  - integration with Python, Answer Set Programming solver, later Prolog and LISP
  - demo application(!)

Summary

Structure of rule-based programming language:

- concept of mental state transformer
- powerful macros

Purely syntactical approach $\rightsquigarrow$ No change of the core language semantics!

Thank you for your attention.

Please direct your questions to

peter.novak@in.tu-clausthal.de

Summary

Structure of rule-based programming language:

- concept of mental state transformer
- powerful macros

Purely syntactical approach $\rightsquigarrow$ No change of the core language semantics!

Thank you for your attention.

Please direct your questions to

peter.novak@in.tu-clausthal.de