Using Pi-calculus to formalize UML activity diagram for business process modeling

Yang dong, Zhang ShenSheng

IEEE International Conference and Workshop on the Engineering of Computer-Based Systems 2003

2007. 12.12
Chanhee Yi
Contents

- Introduction
- Background
- Related work
- Syntax of UML ADs (Activity Diagrams)
- Semantics of UML ADs
- Model analysis and verification
- Conclusion
- Discussion
Introduction

- **Motivation**
  - Well-known usage of UML activity diagrams
    - It usually models software and business processes
    - It represents control flows among activities
  - **Lack of rigorous semantics** in UML activity diagrams
    - Many suggested trials about formalization were insufficient

- **Research goal**
  - **Suggestion to rigorous process semantics** of UML ADs
    - With Pi-calculus
  - **Utilization of advantages of Pi-calculus**
    - To check equivalence of two business models
    - To verify correctness of properties of business process models
Background : Pi – calculus (1/2)

- General description
  - Computing model for representing **concurrent** systems
  - Expression to **interactions** between evolving processes

- Advantages
  - Provision to the way of bottom-up construction of systems
  - Existence of Pi-calculus analytical tools

- Two basic concepts
  - Name : channels(ports), variables, data
  - Processes : entity in systems
Syntax of process expression

<table>
<thead>
<tr>
<th>Type</th>
<th>Representation</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input prefix</td>
<td>a(x).P</td>
<td>Receive x through a, then behave as P</td>
</tr>
<tr>
<td>Output prefix</td>
<td>(\overline{a}&lt;x&gt;.P)</td>
<td>Output x on a, then behave as P</td>
</tr>
<tr>
<td>Summation</td>
<td>(P_1 + P_2)</td>
<td>Behave like either (P_1) or (P_2)</td>
</tr>
<tr>
<td>Composition</td>
<td>(P_1 \mid P_2)</td>
<td>Do all that (P_1) or (P_2) can do</td>
</tr>
<tr>
<td>Restriction</td>
<td>new y (\langle P\rangle)</td>
<td>y can’t be used for communicating</td>
</tr>
<tr>
<td>Matching</td>
<td>(\langle x=y\rangle.P)</td>
<td>Do P when x=y</td>
</tr>
</tbody>
</table>

Example

\[S = \text{def } \text{new } c \langle A \mid C \rangle \mid B\]

\[A = \text{def } a.\text{new } d \langle A \mid A' \rangle + c.A''\]
Related work

- About trials to formalize UML activity diagrams

<table>
<thead>
<tr>
<th>Author(s)</th>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>E.Borger et al., [AMAST’00]</td>
<td>Uses ASM(Abstract State Machine) for representing semantics of activity diagrams</td>
</tr>
<tr>
<td>Jorgenson et al., [UML’00 workshop]</td>
<td>Adapts FSP(Finite State Processes) to formalize activity diagrams</td>
</tr>
</tbody>
</table>

- But still not has accepted as an formal semantics

<table>
<thead>
<tr>
<th>Author(s)</th>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>Martin Goglla et al., [LNCS 2263]</td>
<td>Uses OCL(Object Constraint Languages) to describe constraints</td>
</tr>
</tbody>
</table>

- Fails to express dynamic behavior of activity diagrams

*AMAST = Algebraic Methodology And Software Technology conference*
Syntax of ADs (1/3)

- Subset of structural elements of ADs*

*According to UML 1.4 standards
Syntax of ADs (2/3)

- Definitions

  1. UML ADs

- $\text{Nodes} = \text{AS} \cup \text{SS} \cup \text{PS} \cup \{ \text{initial}, \text{final} \}$
  is the set of state nodes

  where
  - $\text{AS}$: set of action nodes
  - $\text{SS}$: set of subactivity nodes
  - $\text{PS}$: \{node $|$ type(node) = DECISION $\lor$ type(node) = Merge $\lor$ type(node) = JOIN $\lor$ type(node) = FORK $\lor$ type(node)\}

- $\text{Edges} \subseteq (\text{Nodes} \times \text{Guards} \times \text{Nodes})$
  is the transition relation between the state nodes

- $\text{Guard}$
  is the set of guard expressions

$$\text{AD} := (\text{Nodes}, \text{Edges}, \text{Guards})$$
Syntax of ADs (3/3)

- Definitions (cont’d)
  - 2. Binary relation \( R \)
    \[
    R = \{ \langle node_i, node_j \rangle \mid node_i \text{ and } node_j \text{ are in the same swimlane} \}
    \]
    (1) \( R \) is an equivalence relation
    (2) \( R \) is a partition of the set \( Nodes^{AD} \)

  - 3. UML ADs with swimlanes

\[
AD\text{-swimlanes} := \{ AD, Lanes, R, h \}
\]
Where
- \( Lanes \) : the set of names of swimlanes
- \( h : Lanes \rightarrow \{ [x]_R \mid x \in Nodes^{AD} \} \)

If \( edge \in Edges^{AD} \), then the following holds:
\[
\text{source}(edge) \in h(lane) \quad \text{end}(edge) \not\in h(lane) \quad \Rightarrow \quad j \neq i, \text{end}(edge) \in h(lane)
\]
Semantics of ADs (1/4)

- Definitions
  - 4. Function $\alpha$
    \[
    \alpha : \text{Nodes}^{AD} \cup \{AD\} \rightarrow \{P\}
    \]
    - $P$ : process expression in the pi-calculus
  - 5. $\in$ set of action nodes
    \[
    \alpha_{as} = \text{new ack (inevent. (ACTION | ack.outevent ))}, \text{ACTION} = \text{action ack. 0}
    \]
  - 6. edge
    \[
    P \cap Q(m/r,m/l) = \new m(\{m/r\}P|\{m/l\}Q)
    \]
    \[
    \alpha_{source(edge)} \cap \alpha_{end(edge)} (m/outevent, m/inevent)
    \]
Semantics of ADs (2/4)

- Definitions (cont’d)
  - 7. \( \text{ps} \in \text{set of pseudo-state nodes} \)
    - (1) Type(\text{ps}) = \text{FORK}
      \[
      \alpha_{\text{FOR}K} \overset{\text{def}}{=} \text{inevent}. \prod_{i \in I} \text{outevent}_i
      \]
    - (2) Type(\text{ps}) = \text{JOIN}
      \[
      \alpha_{\text{JOIN}} \overset{\text{def}}{=} \text{new } x \left( \prod_{i \in I} (\text{invent}_1. x) \mid \prod_{i \in I} \text{outevent}_i \right)
      \]
    - (3) Type(\text{ps}) = \text{DECISION}
      \[
      \alpha_{\text{DECISION}} \overset{\text{def}}{=} \text{inevent. guard}(c). (\text{[c = c1] outevent}_1 + \text{[c = c2] outevent}_2)
      \]
    - (4) Type(\text{ps}) = \text{MERGE}
      \[
      \alpha_{\text{MERGE}} \overset{\text{def}}{=} \text{inevent}_1. \text{outevent} + \text{invent}_2. \text{outevent}
      \]
Semantics of ADs (3/4)

- Definitions (cont’d)
  - 8. Initial, final node
  - 9. \( ss \in \text{set of sub-activity nodes} \)
  - 10. Flow graph \( G_{AD} \)

\[
G_{AD} = \{P, R\} \\
R = \{(\alpha_i, \alpha_j) | \alpha_i \cap \alpha_j (m/\text{outevent, m/inevent}), (i, j) \in \text{Edges}^{AD}\} \\
P = \{\alpha_{as}\} \cup \{\alpha_{ss}\} \cup \{\alpha_{ps} | \text{type}(ps) = \text{FORK} \lor \text{type}(ps) = \text{JOIN} \lor \text{type}(ps) = \text{DESIGN} \lor \text{type}(ps) = \text{MERGE}\} \cup \{\alpha_{initial}, \alpha_{final}\}
\]

- 11. \( \alpha_{AD} \)

\[
\alpha_{AD} \overset{\text{def}}{=} \text{new} \lambda (\prod_{\alpha \in AS} \alpha_{as} | \prod_{ps \in PS} \alpha_{ps} | \alpha_{initial} | \alpha_{final})
\]
Semantics of ADs (4/4)

Example

\[ \alpha_{\text{request}} = \text{def new ack (invent. (REQUEST | ack. outevent))} \]

\[ \text{REQUEST} = \text{def requestservice. ack. 0} \]

\[ \alpha_{\text{deliver}} = \text{def new ack (invent. (DELIVER | ack. outevent))} \]

\[ \text{DELIVER} = \text{def deliver. ack. 0} \]

\[ \alpha_{\text{customer}} = \text{def new } e_1 e_2 (\{ e_0 / \text{outevent} \} \alpha_{\text{initial}} \]

\[ \quad | \{ e_0, e_1 / \text{invent, outevent} \} \alpha_{\text{request}} \]

\[ \quad | \{ e_1, e_2, e_3 / \text{invent, outevent}_1, \text{outevent}_2 \} \alpha_{\text{fork}} \]

\[ \quad | \{ e_2, e_4 / \text{invent, outevent} \} \alpha_{\text{pay}} \quad | \{ e_9 / \text{invent} \} \alpha_{\text{final}} )} \]
Model analysis & verification (1/4)

- Equivalence of two process models
  
  ✓ 12. Weak equivalence “≈”
  - Output the same result, but possible be through different procedures

\[ P \approx Q \text{ iff, for all } \alpha \in ACT, \]

(i) whenever \( P \xrightarrow{\alpha} P' \), then \( \exists Q', Q \xrightarrow{\hat{\alpha}} Q', \) and \( P' \approx Q' \)

(ii) whenever \( Q \xrightarrow{\alpha} Q' \), then \( \exists P', P \xrightarrow{\alpha} P' \) and \( P' \approx Q' \)

✓ 13. Equivalence of two processes \( AD, AD' \)

\[ \alpha_{AD} \approx \alpha_{AD'} \]

✓ 14. Deadlock

\[ \exists t, \alpha_{AD} \xrightarrow{t} \alpha_{AD}' \text{ and } \alpha_{AD}' \sim 0. \]

\[ \begin{align*}
ACT & : \text{Action} \\
P, P', Q, Q' & : \text{Processes in pi-calculus} \\
\alpha_{AD}, \alpha_{AD}' & : \text{corresponding process expression of } AD \text{ and } AD' \text{s} \\
t & : \text{Sequence of actions} \\
\sim & : \text{Strong equivalence}
\end{align*} \]

Software Engineering Lab, KAIST
Model analysis & verification (2/4)

- Equivalence of two process models (cont’d)

- Can be checked with the aid of Mobility WorkBench (MWB) Pi-calculus analytical tool

[Diagram A]

[Diagram B]


Model analysis & verification (3/4)

- Requirements correctness of business process

  - Two requirements
    - Safety - Normal termination of process, No dangling activities
    - Liveness - Any activities can be executed

  - Mu-calculus to check lasting properties

\[
\Phi ::= Z | \Phi_1 \land \Phi_2 | \Phi_1 \lor \Phi_2 | [K] \Phi | <K> \Phi | \nu Z. \Phi | \mu Z. \Phi
\]

[ Syntax of mu-calculus ]

- \( Z \) : Variable
- \( \nu Z \) : Largest fixed-pointed operator
- \( uZ \) : Smallest fixed-pointed operator
- \( \land, \lor \) : AND, OR
- \( [K] \) : Globally K presents
- \( <K> \) : Finally K presents
Requirements correctness of business process (cont’d)

Expressions to check lasting properties with mu-calculus

- **Requirement 1 - Safety**
  - Processes always can normally terminate
  
  \[ \alpha_{AD} \models \mu Z. (<final>) tt \lor (\langle\rangle tt \land [-]Z) \]

- **Requirement 2 - Liveness**
  - There are no deadlocks
  
  \[ \alpha_{AD} \models \nu Z. (<\rangle tt \land [-]Z) \]

- **Example**
  
  \[ AD \models \nu Z. [requestService]( \ uY \langle\rangle tt \land [fillOrder]Y ) \land [-]Z \]

[K] : Globally K presents
<K> : Finally K presents
tt : Constant calculus
Conclusion

- Contribution
  - Formalize the subset of UML ADs with Pi – calculus
  - By doing so, Show the way to
    - Check the equivalence of two process models
    - Verify requirements correctness of business process models
Discussion

- **Characteristic**
  - Shows that
    - Calculus is not easy to understand and use but,
      - Syntax is not familiar
      - Semantics are too difficult
    - It can become good source to write a paper
      - There are many things to be formalized in SE
      - Finding appropriate sub-mapping between SE and calculus can be enough to write

- **Limitation**
  - Insufficient in contents to point up strong points of Pi-calculus
    - Contrast to the other approaches
$\text{Nodes}^{AD}$ : set of state nodes

partition : denoted as $\{ [x]_R \mid x \in \text{Nodes} \}$

$\text{ACT}$ : Action

$P, P', Q, Q'$ : Processes in pi–calculus

$\alpha_{AD}, \alpha_{AD}$ : corresponding process

expression of AD and AD’s

$t$ : Sequence of actions

$\sim$ : Strong equivalence