

Formal Methods for Mobile Agent Applications

Kenji Taguchi

Dept. of Computing, University of Bradford

A Joint Work with

Jin Song Dong

Computer Science Department, National University of Singapore

Today's Talk

1. Mobile Agents
2. Development Method
3. Method Integration
4. Example
5. Semantics
6. Related Work
7. Future Work
8. Conclusion

What is Mobile Agent?

- A computing entity which can move around different hosts carrying its state and procedures
- Applications on the Web
 - E-Commerce
 - Information Gathering On the Internet
 - Information Dissemination

Mobile Agent Systems/Languages

- Telescript (General Magic)
- Aglets (IBM) (JAVA API for Mobile Agents)
- D'Agents (formerly Agent Tcl) (Darmouth College)
- A huge list will follow...

Aglets (Program Example)

```
try {  
  
    dispatch('atp:\\ida.dis.uu.se:9900');  
  
}  
catch (Exception x) {  
  
    do_something_for_exception_handling();  
  
}
```

Theories on Mobile Agents

- Mobile Ambient (L. Cardelli and A. Gordon)
- $D\pi$ (J. Riely and M. Hennessy)
- π (R. Milner)
- MobileUnity (G-C. Roman and P.J. McCann)
- LLinda (R. De Nicola, G. Ferrari and R. Pugliese)

Development Methods (1)

- There are many layers which we need to consider.
 - Application Layer (E.g., E-Commerce)
 - System Layer (Mobile Agent Systems)
 - Communication Layer (Protocol)
- Many issues (e.g., security) related to each layer

Development Methods (2)

- Layers and Modelling Methods
 - Application Layer (Mobile Object-Z)
 - System Layer (Mobile Calculi, Process Algebra)
 - Communication Layer (Mobile Calculi, Process Algebra)

Method Integration

- To Combine Compensating (Complementary) Formalisms (Views)
 - Data Oriented View
 - * State-based Formalisms, e.g., Z
 - Process Oriented View
 - * ViewProcess Algebras, e.g., CSP

Examples of Method Integration

- Timed-CSP and Object-Z (J.S. Dong and B. Mahony)
- CCS and Z (K. Taguchi and K. Araki)
- Duration Calculus and Object-Z (G. Smith)
- Integration based on Unifying Theories of Programming, Circus (J. Woodcock)

Our Method Integration

- Data Oriented View
 - Object-Z
- Locality Oriented View
 - Mobile Primitives
- Comparison between TCOZ and MobiOZ
 - Real-time primitives (wait, timeout, deadline)
 - Mobile primitives (go, here)

Object-Z Class

ClassName[*generic parameters*]

inherited classes

type definitions

constant definitions

state schema

initial state schema

operation schemas

Extensions of Object-Z

MoZClassAgent

OZ_Definitions

Process_Definition

Mobile_Op ::= go(l) |
send(⟨l, ..., l⟩) |
kill() |
here(x)

$$\begin{aligned} Proc & ::= 0 \mid \\ & P(x_1, \dots, x_n) \mid \\ & Guard_Expr \mid \\ & Mobile_Op \mid \\ & Proc ; Proc \mid \\ & Proc \square Proc \end{aligned}$$

Design Principle

- Based On Agent-Place Model
 - No mismatch between existing mobile agent systems/languages
- Communication Model
 - Local communication is synchronous
 - Remote communication is asynchronous

Communication

- Channels as state variables in a class

com : **chan**[*Address*]

- Remote communication

addr :: \overline{com}

- *Address* for the unique address in the network

[*Address*]

- Mobile Primitives
 - *go*(*l*)
 - *here*(*x*)
 - *send*($\langle l_1, \dots, l_n \rangle$)

Example 1

[*Address, Data, Name*]

BasicAgent

dest : Address

home : Address

*com : **chan**[Address]*

FAgent _____

BasicAgent

_____ *SetDest(next_dest : Address)* _____

$\Delta(dest)$

_____ *dest' = next_dest* _____

Beh $\hat{=}$ *go(dest)* ;

com(next_dest) \rightarrow *SetDest(next_dest)* ;

Beh

Forwarder

next_dest : *Address*

home : *Address*

com : **chan**[*Address*]

$Beh \hat{=} \overline{com}\langle next_dest \rangle \rightarrow Beh$

Network Configuration

$$addr_1 \llbracket Beh \rrbracket_{\sigma_{basic}} \parallel addr_2 \llbracket Beh \rrbracket_{\sigma_{forward}}$$

where,

$addr_1$ and $addr_2$ are addresses of places where the *FAgent* and the *Forwarder* reside respectively. \parallel is the parallel composition of those places.

SimpleItinerary _____

itinerary : seq *Address*

Del _____

$\Delta(\textit{itinerary})$

itinerary' = tail *itinerary*

SimpleTraveller _____

BasicAgent

SimpleItinerary

New_Dest _____

$\Delta(dest)$

Del

dest' = head itinerary

$Beh \hat{=} here(home) ; Beh_1$

$Beh_1 \hat{=} go(dest) ; Do_Task ;$

$([itinerary \neq \langle \rangle] \bullet New_Dest ; Beh_1$

$\square [itinerary = \langle \rangle] \bullet go(home))$

$Do_Task \hat{=} 0$

Traveller

SimpleTraveller[**redef** *Beh₁*]

report : **chan**[*Name*, *Address*]

unreg : **chan**[*Name*]

db_addr : *Address*

name : *Name*

Beh₁ $\hat{=}$

db_addr :: $\overline{\text{report}}$ \langle *name*, *dest* \rangle ;

go(*dest*) ; *Do_Task* ;

([*itinerary* \neq $\langle \rangle$] • *New_Dest* ; *Beh₁*

□

[*itinerary* = $\langle \rangle$] •

db_addr :: $\overline{\text{unreg}}$ \langle *name* \rangle ; *go*(*home*)

Finder

BasicAgent

traveller_addr, db_addr : Address

traveller_name : Name

query : chan[Name, Address]

answer : chan[Address]

Get_Address(addr : Address)

$\Delta(traveller_addr)$

traveller_addr' = addr

$\widehat{Beh} = db_addr :: \overline{query}\langle traveller_name, home \rangle ;$

answer(addr) → Get_Address(addr)

DBAgent

$db : Name \rightarrow Address$

$report, query : \mathbf{chan} [Name, Address]$

$unreg : \mathbf{chan} [Name]; answer : \mathbf{chan} [Address]$

$Register(n : Name, current_addr : Address)$

$\Delta(db)$

$db' = db \oplus \{n \mapsto current_addr\}$

$Unregister(n : Name) \hat{=} [\Delta(db) \mid db' = \{n\} \triangleleft db]$

$Beh \hat{=} (report(n, current_addr) \rightarrow Register(n, current_addr)) \square$

$query(n, home) \rightarrow [n \in \text{dom } db] \bullet home :: \overline{answer} \langle db(n) \rangle \square$

$unreg(n) \rightarrow Unregister(n); Beh$

Semantics (1)

- Semantic function which associates a process with an address, a state and an ID

$$\llbracket \cdot \rrbracket : Proc \longrightarrow Proc \times Address \times State \times ID$$

- Different Agents running in parallel at the same location (at different locations)

$$l\llbracket P \rrbracket_{\sigma_a} \parallel l\llbracket Q \rrbracket_{\sigma_b} \quad (l\llbracket P \rrbracket_{\sigma_a} \parallel m\llbracket Q \rrbracket_{\sigma_b})$$

- Unique Entry Point

$$l\llbracket Beh \rrbracket_{\sigma_a}$$

Semantics (2)

- State Transition Semantics of Object-Z

$$\sigma, \sigma' \models Op(V_1, \dots, V_n)$$

- Validity of a guard under a single state

$$\sigma \models G$$

Derivation Rules (1)

$$\frac{\sigma_a(addr) = m}{l[[go(addr); P]]_{\sigma_a} \xrightarrow{go(m)} m[[P]]_{\sigma_a}}$$

$$\frac{\sigma'_a = \sigma_a \oplus \{x \mapsto l\}}{l[[here(x); P]]_{\sigma_a} \xrightarrow{here(l)} l[[P]]_{\sigma'_a}}$$

Derivation Rules (2)

$$\frac{}{l[[kill(); P]]_{\sigma_a} \xrightarrow{kill()} l[[0]]_{\emptyset}}$$

$$\sigma_a \models G, \quad \sigma_a, \sigma'_a \models Op(V_1, \dots, V_n)$$

$$\frac{}{l[[G \bullet Op(x_1, \dots, x_n); P]]_{\sigma_a} \xrightarrow{Op(V_1, \dots, V_n)} l[[P\{x_1/V_1, \dots, x_n/V_n\}]]_{\sigma'_a}}$$

Derivation Rules (3)

$$\frac{\sigma'_a = \sigma_a \oplus \{x \mapsto V\}, \sigma'_a \models G, \sigma'_a, \sigma''_a \models Op(V)}{l[[G \bullet com(x) \rightarrow Op(x) ; P]]_{\sigma_a} \xrightarrow{com(V)} l[[P\{x/V\}]]_{\sigma''_a}}$$

$$\frac{\sigma_a \models G, \sigma_a(expr) = V, \sigma_a, \sigma'_a \models Op}{l[[G \bullet \overline{com}\langle expr \rangle \rightarrow Op ; P]]_{\sigma_a} \xrightarrow{\overline{com}\langle V \rangle} l[[P]]_{\sigma'_a}}$$

$$\frac{}{l[[\overline{com}\langle V \rangle]]_{\emptyset} \xrightarrow{\overline{com}\langle V \rangle} l[[0]]_{\emptyset}}$$

Derivation Rules (4)

$$\frac{\sigma_a \models G, \sigma_a(\text{expr}) = V, \sigma_a, \sigma'_a \models Op}{l[[G \bullet k :: \overline{\text{com}}\langle \text{expr} \rangle \rightarrow Op ; P]]_{\sigma_a} \xrightarrow{k :: \overline{\text{com}}\langle V \rangle} k[[\overline{\text{com}}\langle V \rangle]]_{\emptyset} \parallel l[[P]]_{\sigma'_a}}$$

$$\frac{l[[P]]_{\sigma_a} \xrightarrow{\text{com}(V)} l[[P']]_{\sigma'_a} \quad l[[Q]]_{\sigma_b} \xrightarrow{\overline{\text{com}}\langle V \rangle} l[[Q']]_{\sigma_b}}{l[[P]]_{\sigma_a} \parallel l[[Q]]_{\sigma_b} \xrightarrow{\tau} l[[P']]_{\sigma'_a} \parallel l[[Q']]_{\sigma_b}}$$

Derivation Rules (5)

$$\frac{\mathcal{P}_1 \xrightarrow{\alpha} \mathcal{P}'_1}{\mathcal{P}_1 \parallel \mathcal{P}_2 \xrightarrow{\alpha} \mathcal{P}'_1 \parallel \mathcal{P}_2}$$

$$\frac{\mathcal{P} \equiv \mathcal{P}' \quad \mathcal{P} \xrightarrow{\alpha} \mathcal{Q} \quad \mathcal{Q} \equiv \mathcal{Q}'}{\mathcal{P}' \xrightarrow{\alpha} \mathcal{Q}'}$$

Congruence

$$l[[0]]_{\emptyset} \equiv 0$$

$$l[[P \square Q]]_{\sigma_a} \equiv l[[Q \square P]]_{\sigma_a}$$

$$l[[(P \square Q) \square R]]_{\sigma_a} \equiv l[[P \square (Q \square R)]]_{\sigma_a}$$

$$l[[(0; P)]]_{\sigma_a} \equiv l[[P]]_{\sigma_a}$$

$$l[[(P; 0)]]_{\sigma_a} \equiv l[[P]]_{\sigma_a}$$

$$\mathcal{P} \parallel 0 \equiv \mathcal{P}$$

$$\mathcal{P}_1 \parallel \mathcal{P}_2 \equiv \mathcal{P}_2 \parallel \mathcal{P}_1$$

$$(\mathcal{P}_1 \parallel \mathcal{P}_2) \parallel \mathcal{P}_3 \equiv \mathcal{P}_1 \parallel (\mathcal{P}_2 \parallel \mathcal{P}_3)$$

Related Work

- Stateless Formalisms
 - Mobile Calculi
 - * Hard to model Stateful Agents

- Statefull Formalisms
 - MobileUnity
 - LLinda
 - * Not readily applicable for the development
 - Lack of corresponding mobile primitives
 - Mismatch of underlying model

Future Work

- Enhancement of language features
 - Communication
- Verification
 - Hennessy-Milner Logic based on transition systems presented

Conclusion

- An integrated Notation of Object-Z and mobile primitives for mobile agent applications
- A Operational Semantics for that Integrated Notation