Transformation from Seal Calculus to Mobile Ambient Calculus

Zhang Jing[†], *Zhang Li-Cui*[††], *Guo De-Gui*[†††]

[†,†††]College of Computer Science and technology, Key Laboratory of Symbolic Computation and Knowledge Engineering of Ministry of Education of P.R.China Jilin University, ChangChun, 130012;
[††]College of Communication Engineering, Jilin University, ChangChun, 130012

Summary
Based on analyzing syntax structrure and semantics reduction system of Seal calculus and Mobile Ambient calculus, we investigate three equivalence relations of the two calculus : communication's equivalence, communication primitives's equivalence and code movement equivalence. Then we show the structural transformation technology from Seal to Mobile Ambient. Our work proposes a systemitical method for analysising and comparing logical structure and expressive power of different formal systems, proves expressive power of Seal calculus. The results presented in this paper summarize our work on formal foundations of mobile languages.

Key words:
Seal calculus; Mobile ambient calculus; process communication; mobile computation

Introduction

We relate two models of distributed mobile programming, the Seal calculus[1] and the Mobile Ambient calculus(MA)[2]. Technically, these two models have a lot of common. They consist of name-passing process calculi in the spirit of the π-calculus[3]. They make explicit the spatial structure of the computation by distributing processes over a tree of nested locations(seals or ambients) that stand for both machines and agents. They provide some mechanism to rearrange the location tree as part of the computation, thereby describing agent migration. Still, these two models address different aspects of wide-area distributed computations, and thus yield different interpretations of locality[4].

In the Seal calculus, *the association between names and locations is weak*. Names are merely tags used by parent seals to tell their children apart. They can be changed at the parent's whim. Physical and logical resources are modeled by *channels*, which are named computational structures used to synchronize processes. Channels can be interpreted as located or shared. The seal calculus differentiates between local and remote process interaction. To retain a realistic programming model, interaction between locations is restricted to the asynchronous sending of messages or sub-locations. Overall, locations in the Seal calculus are adequate for high-level programming with asynchronous messages and agents[4].

In the MA calculus, *locality and control are tightly connected*; each ambient acts as an opaque box, and interactions can occur only between processes that are in adjacent ambients. The routing of a process from one ambient to another is kept explicit; to accomplish the migration, the moving ambient must be aware of the path in the ambient tree; if an intermediate ambient decides to block the migration, or if the path evolves during the migration, ambients may get stuck or lost. Interaction is local, in the sense that any reduction involves processes separated by at most one ambient boundary, but the synchronization between these processes is rather complex. Overall, ambients are good at expressing administrative domains, highly dynamic environments, and controlled migration[5].

To investigate the relation between Seal and MA and establish the theory foundation of a distributed programming language, we describe a transformation technique from Seal to MA. This paper is structured as follows. Section 2 and Section 3 present a review of MA calculus and Seal respectively, includes the syntax and semantics. Section 4 discusses the communication, communication primitives, code mobility in both calculus firstly; then presents three equivalence relations: communication equivalence, communication primitives' equivalence, code mobility equivalence; finally, gives a structural transformation function from Seal's communication processes into Ambient's communication processes. Section 5 states some conclusions and outlines future directions of investigation.

2. Mobile Ambient Calculus

2.1 Syntax

We briefly describe the Mobile Ambient calculus, from[2].

Manuscript received May 5, 2006.
Manuscript revised May 25, 2006.

Processes P, Q ::= (νn)P restriction
| 0 inactivity
| P | Q composition
| !P replication
| M[P] ambient
| M.P action
| $(\vec{x})$ .P input
| $<\vec{M}>$ output

Expressions M ::= n name
| in M can enter M
| out M can exit M
| open M can open M
| ε empty
| M.M′ path

2.2 Semantics

We now give an operational semantics of the calculus of section 2.1, based on a structural congruence between processes, ≡, and a reduction relation →. This is a semantics in the style of Milner's reaction relation[6] for the π-calculus, which was itself inspired by the Chemical Abstract Machine of Berry and Boudol[7].

Structural Congruence

P ≡ P (Struct Refl)
P ≡ Q ⇒ Q ≡ P (Struct Symm)
P ≡ Q , Q ≡ R ⇒ P ≡ R (Struct Trans)
P ≡ Q ⇒ (νn)P ≡ (νn)Q (Struct Res)
P ≡ Q ⇒ P|R ≡ Q|R (Struct Par)
P ≡ Q ⇒ ! P ≡ ! Q (Struct Repl)
P ≡ Q ⇒ n[P] ≡ n[Q] (Struct Amb)
P ≡ Q ⇒ M.P ≡ M.Q (Struct Action)
P ≡ Q ⇒(n).P ≡ (n).Q (Struct Input)
P | Q ≡ Q | P (Struct Par Comm)
(P | Q)| R ≡ P | (Q | R) (Struct Par Assoc)
!P ≡ P | !P (Struct Repl Par)
(νn)(νm)P ≡ (νm) (νn)P if n≠m (Struct Res Res)
(νn)(P | Q) ≡ P | (νn)Q if n∉ fn(P) (Struct Res Par)
(νn)m[P] ≡ m[(νn)P] if n≠m (Struct Res Amb)
P | 0 ≡ P (Struct Zero Par)
(νn) 0 ≡ 0 (Struct Zero Res)
!0 ≡ 0 (Struct Zero Repl)
ε.P ≡ P (Struct ε)
(M.M′).P ≡ M.M′.P (Struct .)

Reduction

n[in m.P | Q]|m[R] → m[n[P | Q] | R]
m[n [out m.P | Q] | R] → n[P | Q] | m[R]
open n.P | n[Q] → P | Q
(x).P | <M> → P {x←M}
P → Q ⇒ (νn) P→ (νn) Q
P → Q ⇒ n[P] n[Q]
P → Q ⇒ P | R → Q | R
P′ ≡ P, P→ Q, Q ≡Q′ ⇒ P′→ Q′
→* reflexive and transitive closure of →

3. Seal Calculus:Syntax and Semantics

3.1 Syntax

In the Seal calculus[1], the main concepts of distributed computing are distilled down to three abstractions: processes, locations, and resources. Process are sequential threads of control modeled on the π-calculus with terms to denote the inert process, sequential and parallel composition, and replication. Locations with an internal process are denoted by terms called seals. Resources are modeled by channels, which are named computational structures used to synchronize processes, they can be either located or shared. In this paper, we use shared channel. We use $m, n, \ldots, x, y, z$ to range over variables, P,Q,R,S to range over processes, $\vec{x}_n$ to denote the tuple $x_1, \ldots, x_n$, and just $\vec{x}$ when arity is not important or clear from the context. The syntax of Seal is formally defined by the following grammar:

Processes P, Q::=0 inactivity
| P ‖ Q composition
| ! α.P replication
| (νx)P restriction
| α .P action
| χ [P] seal
Actions α ::= $\overline{x}^{\eta}(\vec{y})$ output
| $x^{\eta}(\lambda\vec{y})$ input
| $\overline{x}^{\eta}\{y\}$ send
| $x^{\eta}\{\vec{y}\}$ receive
Locations η ::= *n* down
| ↑ up
| * local

3.2 Semantics

$x^*(\vec{u}).P \| \overline{x}^*(\vec{v}).Q \rightarrow P\{\vec{v}/\vec{u}\} \| Q$ (write local)

$$\overline{x}^{y} < \vec{v} > .P \parallel y[(\nu \vec{z})x^{\uparrow} < \vec{u} > .Q] \rightarrow$$

$$P \parallel y[(\nu \vec{z})Q\{\vec{v}/\vec{u}\}] \quad \vec{z} \cap \vec{v} = \varnothing \qquad \text{(write in)}$$

$$x^{y}(\vec{u}).P \parallel y[(\nu \vec{z})x^{\uparrow}(\vec{v})Q1 \parallel Q2] \rightarrow$$

$$(\nu \vec{v} \cap \vec{z})P\{\vec{v}/\vec{u}\} \parallel y[(\nu \vec{z} \setminus \vec{v})Q1 \parallel Q2] \qquad \text{(write out)}$$

$$x*\{\vec{u}\}.P \parallel \overline{x}*\{v\}.Q \parallel v[B] \rightarrow$$

$$P \parallel Q \parallel u_1[B] \parallel ... \parallel u_n[B] \qquad \text{(move local)}$$

$$x^{y} < v > .P \parallel v[R] \parallel y[(\nu \vec{z})x^{\uparrow} < \vec{u} > .Q \parallel S] \rightarrow$$

$$P \parallel y[(\nu \vec{z})(Q \parallel S \parallel u1[R] \parallel ...un[R])] \qquad \text{(move in)}$$

$$x^{y} < u > .P \parallel y[(\nu \vec{z})x^{\uparrow} < v > .Q \parallel v[R] \parallel S] \rightarrow$$

$$P \parallel u[R] \parallel y[(\nu \vec{z})Q \parallel S] \qquad \text{(move out)}$$

4. Equivalence Relation

In this section, equivalence relation between Seal Calculus and Mobile Ambient Calculus is given. This research helps to compare logical relation and function of the two calculus, aids to optimize process expression and transform between different process . The main problem faced when transform from Seal to MA is channel and code mobility, because there are channels in Seal and no channel in MA; Code mobility expressions in the two calculus have different syntax structure; in MA, primitive *open* can dissolve ambient' s boundary, in Seal, boundary can not be dissolved.

4.1 Communication Based on Channels

Communication in the basic ambient calculus happens in the local ether of an ambient. Messages are simply dropped into the ether, without specifying a recipient other than any process that does or will exist in the current ambient. However, most process calculi use communication based on named channels, like Seal. In Seal, there are many named channels to be used by a seal to communication[8,9]. While in MA, we should think of a channel as a new entity that may reside within an ambient. In particular, communications executed on the same channel name but in separate ambients will not interact, at least until those ambients are somehow merged. So, we need to generate an ambient to act as a channel, called as ***channel ambient***.

The basic idea for representing channels is as follows; see[10] for details. If *c* is the name of a channel we want to represent, then we use a name *c* to name an ambient that acts as channel ambient. Due to each ambient can participate in communication actions, we need create a subambient for each non-channel ambient to act as channel ambient and name it as *c*. Channel ambients open all the incoming packet and activate communication interaction. So, an output on channel *c* is represented as a communication packet that enters *c*(where it is opened up) and that contains an input operation; after the input is preformed, the rest of the process exits *c* to continue execution. The creation of a channel name c is represented as the creation of *c*[! (x)<x> | <x>]. Similarly, the communication of a channel name c is represented as local communication of ambient c.

First, we review the structure of communication expression of the two calculus. According the syntax, communication in Seal includes three kinds: local communication, upwords communication and dowards communication; there is only local communication in Ambient. So in order to describe the upwards communication and downwards communication, we need utilize Ambient's mobility primitives.

The standard Ambient calculus's mobility is called "subjective mobility", because processes can control their own mobility, however, process mobility in Seal is called "objective mobility" for processes can not decide when or where to move and they are controlled by their parent. Objective mobility can be simulated by subject mobility, but the program is very tedious. So we induce an objective mobile primitive"*go*" and expression go N.M[P] denotes move process M[P] to ambient N along the rout N, then activate it. We call "*go*" as an objective primitive because when process M[P] moves, the ambient enclosing it doesn't move, which is different from in, out primitive. In fact, *go* N.M[P] equivals to (vk)k[N.M[out k.P]] in terms of process mobility.

The following is the syntax and semantics of the extended Ambient calculus with a special mobility primitive go.

Processes *P*, *Q* ::= ……same to 2.1
| *go N.M*[*P*]

Structural Congruence

P ≡ Q ⇒ go N.M[P] ≡ go N.M[Q] (Struct Go)

go ε .*M*[P] ≡ *M*[*P*] (Struct Go ε)

Reduction :

go (in m.N).n[P] | m[Q] → m[go N.n[P] | Q] (Red Go In)

m[go (out m.N).n[P] | Q]→ go N.n[P] | m[Q] (Red Go Out)

Then, we discuss the equivalence relation between the communication process based on channel in Seal and communication process without channel. What in the left

of ⇔ is Seal's communication expression, what in the right of ⇔ is corresponding MA communication expression, symbol ⇔ denotes semantics equivalence.

Communication Equivalence (⇔)

$\overline{c}$ *<y> | c*<λz> ⇔
go in c.<y> | go in c.(z) (local communication)
s[$\overline{c}^{\uparrow}$<y>] | c*<λz> ⇔
s[go out s.in c.<y>] | go in c.(z) (up-communication)
c^{s}<λy> | s[$\overline{c}$ *<y>] ⇔
go in s.in c.(y) | s[go in c.<y>] (down-communication)
s[$c^{\uparrow}$<λy>] | $\overline{c}$ *<y> ⇔
s[go out s.in c.(y)] | go in c.<y> (up-mobility)
$\overline{c}^{s}$<y> | s[c*<λy>] ⇔
go in s.in c.<y> | s[go in c.(y)] (down-mobility)

The above is the basic principle of channel and communication handling. We discuss the equivalent theorem of Seal's communication primitives and MA's communication expression, this theorem can be used to implement transformation from Seal to MA.

Communication primitives equivalence (=)

m[$\overline{c}$ *<y>] = m[go in c.<y>]
m[c*<λy>] = m[go in c.(y)]
m[$\overline{c}^{\uparrow}$<y>] = m[go out m.in c.<y>]
m[$c^{\uparrow}$<λy>] = m[go out m.in c.(y)]
m[$\overline{c}^{s}$<y>] = m[go in s.in c.<y>]
m[c^{s}<λy>] = m[go in s.in c.(y)]

Term in the left of = is a seal, term in the right of = is an ambient, they equivalent in terms of communication effect.

4.2 Process Mobility

The most difficult problem of transformation from Seal to MA is process's mobility, because the process expression's syntax structure of Seal is different from that of MA's.

Difference between Seal and Mobile Ambient in code mobility are listed as follows:
a) mobility in Seal is objective, that is seal's mobility is controlled by its enclosing seal. However, mobility in MA is subjective, that is ambients' mobility is controlled by itself;
b) mobility in Seal is implemented by communication between processes. However, mobility in Ambients is implemented by mobile primitives.
c)seals in Seal doesn't move, what moves is the body of seal, however, ambient and its body move together.
In Seal, since processes' mobility is implemented by communication, so the mobility in Seal is also devides into local mobility, in which process moves in the same seal; upwords mobility, in which process moves from subseal into parent seal; downwords mobility, in which process moves from parent into subseal. These three mobility are described in Fig.1.

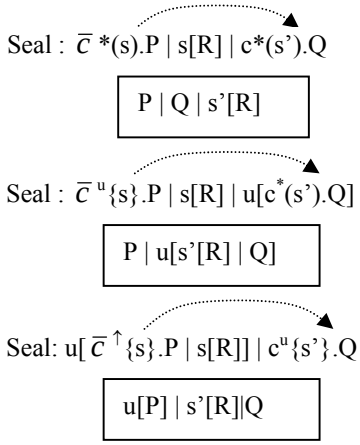

Fig.1.Seal calculus's process mobility

In Fig.1, the text tagged by arcs are Seal's process expression, the arcs illustrate code's movement track. Seal body is transitted on channel, when the receiver received the code, then give it a new name. The text circled by frames are the result process expression of code movement. In the following , we discuss the corresponding MA processes.

In syntax, the prominence difference between MA and Seal are: (1) In Seal, there are explicit location symbol ↑, * and u to denotes location, in MA, ambients' name denotes location directly; (2) Seal implements mobility using sending and receiving action, MA implements mobility using mobile primitives: in or out, without the synchronous between sending and receiving actions.

Suppose we want to move an ambient n[$\boldsymbol{B_n}$] into ambient m[$\boldsymbol{B_m}$], then the notation of Seal and MA are as follows:

Seal : n[$\overline{c}^{\uparrow}${n} | $\boldsymbol{B_n}$]|c*{n}. $\overline{c}^{m}${n} | m[c*{n} | $\boldsymbol{B_m}$]
Mobile Ambient : n[in m . $\boldsymbol{B_n}$] | m[$\boldsymbol{B_m}$]

for under their reduction rules respectively, they get the similar result m[n[$\boldsymbol{B_n}$] | $\boldsymbol{B_m}$].

Suppose we want to enhance an ambient m[Bm] out of n from n[P | m[Bm]], then the notation of Seal and MA are as follows:

Seal : n[$\overline{c}^{\uparrow}${m} | P | m[$\boldsymbol{B_m}$]]| c*{m}
Mobile Ambient : n[P | m[out n . $\boldsymbol{B_m}$]]

for under their reduction rules respectively, they get the similar result m[$\boldsymbol{B_m}$] | n[P].

Based on the above analysis, we summary equivalence relation between these two calculus as follows:

1) n[$\overline{c}^{\uparrow}${n} | $\boldsymbol{B_n}$]|c*{n}. $\overline{c}^{m}${n} | m[c*{n} | $\boldsymbol{B_m}$] $\leftrightarrow$ n[in m . $\boldsymbol{B_n}$] | m[$\boldsymbol{B_m}$]

2) n[$\overline{c}^{\uparrow}${m} | P | m[$\boldsymbol{B_m}$]]| c*{m} $\leftrightarrow$ n[P | m[out n . $\boldsymbol{B_m}$]]

where $\leftrightarrow$ represents equivalence function.

3.3 Tranformation from Seal to Mobile Ambient

A complete transformation system from Seal processes to MA processes is just like a huge, complicate compiler system. Due to the intrinsic difference between Seal and MA, we can not give a complete transformation system in text level. So we just give a structural transformation from Seal's communication processes into MA's communication processes. We use 【E】m to denotes transformation functions of processes, that is 【E】m represents the equival MA process of E in the environment m, where E represents Seal's process, m represents the current Seal's name. The transformation function is defined as follows:

【0】m = 0

【P | Q】m = 【P】m | 【Q】m

【(νx)P】m = (νx)x[! (y)<y> | <y>] | 【P】m

【χ [P]】m = x[【P】m | c [! (y)<y> | <y>]]

【α . P】m = 【α】m | 【P】m

【$\overline{c}$ *<y>】m = go in c.<y>

【c*<λy>】m = go in c.(y)

【$\overline{c}^{\uparrow}$<y>】m = go out m.in c.<y>

【$c^{\uparrow}$<λy>】m = go out m.in c.(y)

【$\overline{c}^{s}$<y>】m = go in s.in c.<y>

【c^{s}<λy>】m = go in s.in c.(y)

4 Conclusions and Future Work

Mobile Ambients calculus (MA) is the first abstract calculus for describing global computation and mobile c omputation successfully, there is a mature theory research for MA. While Seal calculus is a new calculus based on MA, so exploring the relation between Seal and MA bears an important theory meaning. This paper presents an equivalence relation between Seal calculus and MA calculus and proposes a transformation technique from Seal calculus to MA calculus. This work helps to deeply understand mobility and communication of Seal calculus and MA calculus, which establishs the theory foundation of a unified mobile computation framework.

Finally let us point out some directions in which further work could be done. First, the MA can be extended to include communication based on names, and increase the transformation functions correspondingly, then get the equal Seal representation; Also we only give the mobility equivalence between the two calculus, further transformation functions need to be defined, maybe intermediate language can be used to implement transformation from MA to Seal.

References

[1] J.Vitek and G.Castagna. *Seal: A Framework for secure Mobile Computations*. In Internet Programming Languages, number 1686 in Lectures Notes in Computer Science, pages 47-77. Springer-Verlag, 1999.

[2] L.Cardelli and A.D.Gordon. *Mobile Ambients*. In M.Nivat, editor,Foundations of Software Science and Computational Structures, number 1378 in LNCE, Springer-Verlag, 1998, 140-155.

[3] Milner R., and Walker D. A Calculus of Mobile Processes, Part I/II. Information and Computation, 100:1-77,1992.

[4] C.Fournet and A.Schmitt. An implementation of Ambients in JoCAML. http://join.inria.fr/ambients.html, 1999.

[5] Luca Cardelli, Andrew D.Gordon, Mobile Ambients, Foundations of software Science and Computation Structures, LNCS 1578(1999).

[6] Milner, R., Functions as processes. Mathematical Structures in Computer Science 92. Springer Verlag. 1980.

[7] Berry, G. and G.Boudol, The chemical abstract machine. Theoretical Computer Science 96(1), 217-248, 1992.

[8] Cardelli L. and Ghelli G., and Gordon A.D. Mobility types for mobile ambients[R]. Technical Reports. Microsoft Research Center. MSR-TR-99-32. 1999.

[9] Hennessy M, Riely Z. Type-safe Execution of Mobile Agents in Anoynymous Networks[A]. In Proceedings of the Workshop on Internet Programming Languages, (WIPL)[C]. Chicago, 1998.

[10] Abadi M, Fournet C, Gonthier G. Secure Implementation of Channel Abstractions[DB/OL]. Long version(Draft). http :// citeseer. ist.psu.edu/abadi00secure.html, 1998-09.

Zhang Jing received the B.S. and M.S. degrees in College of Computer S cience and Technology of Jilin University in 1998 and 2001, respectively. Since 2001, she has been a Ph.D student in College of Computer Science and Technology of Jilin University. The main research objects include formal method and compiling technique of programming languages..