OPERATIONAL SEMANTICS OF THE SEAL CALCULUS

Zhang Jing,[1] Zhang Li-Cui [2,*] Guo De-Gui[1]

[1] *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,P.R.China
zhangjing99@jlu.edu.cn,guodg@jlu.edu.cn

[2] *College of Communication Engineering, Jilin University, Changchun, 130012, P.R.China*
zlc6796@sohu.com

Abstract As a distributed process calculus with localities and mobility of computational entities, Seal calculus is playing an important role in expressing key features such as security and mobility of Internet programming directly. However, little implementation technique proposed for the calculus, partly due to the complication of mobile computation, which fusions three important techniques: concurrency, distribution and mobility at the same time. The abstract machine PSN for a distributed implementation of the Seal calculus is presented. In PSN the logical structure of a seal system and its physical distribution are separated which induces a more simple and clear implementation. Moreover, an operational semantics description of the Seal calculus based on PSN is given.

Keywords: Mobile computation, Seal Calculus, abstract machine, operational semantics

1. Introduction

The Seal calculus[1,2] is a mobile process calculus aims to model programming large scale distributed systems over open networks, with the goal of being able to express the essential properties of Internet programs. Seal can be seen as a framework for exploring the design space of security and mobility features[3,4]. However, at present the research on Seal calculus is still at the stage of perfecting its theory, less work has been done on its application and implementation, only one implementation is mentioned in [5]. Moreover, the existed formal semantics of the Seal all base on the reduction semantics, which is easy to understand but difficult to implement. The problems of implementa-

*Corresponding author: zlc6796@sohu.com

tion have been a restraint to the development of programming languages based on Seal and to experimentation of Seals on concrete examples. In our opinion, implementation is one of the aspects of Seal that most need investigations.

One of the difficulties of a distributed implementation of a hierachical language such as Mobile Ambient[6] and Seal is that each movement operation involves ambients(or seals) on different hierachical levels. In [7,8] locks are used to achieve a synchronization among all ambients(or seals) affected by a movement. In a distributed setting, however, this lock-based policy can be expensive. Many programming languages and abstract calculi use abstract machine to describe their semantics. Abstract machine as an intermediate stage increases portability and maintainability of compilers[9,10]. [11] present a distributed abstract machine of SA(Safe Ambients Calculus), the main idea is to mode each seal as a network node, communication between these nodes base on asychcronous message transimission, which simulates the communication and mobility of seals. The contribution in [11] motivates our work. However, The Seal calculus differs from Ambients in two important ways. First, Ambients use subjective mobility(an agent moves itself) in Seal mobility is objective(an agent is moved by its context). Second, in Seal both communication and mobility between seals base on channels, which are named computational structures used to synchronize processes, in Ambients communication is local and mobility bases on capabilities. So neither the definition nor the implementation of the abstract machine of Seal will be defferent from the Ambients'. In [12], we give a simple distributed implementation of the Seal Calculus, which is the basis of this paper.

The paper is organized as follows. Section 2 introduces the formal syntax and relevant properties of the abstract machine PSN. Section 3 presents the operational semantics of Seal based on the states transition, finally, a transition example is proposed to verify the correctness of the semantics. The last section, concludes the paper with a discussion of future work.

2. Abstract Machine

We call the abstract machine defined in this paper PSN(Pervasive Seal Network), which separates between the logical and the physical distribution of the seals. The logical distribution is given by the tree structure of the seal syntax(a seal can contain other seals). The physical distribution is given by the association of a location with each seal.

In PSN, a seal named s is represented as a located seal $(h : s[P], f, ss)$, where h is the location, or site, at which the seal runs, f is the location of the parent of the seal, and P is the proceses local to the seal and ss is the location set of the subseals. While the same name may be assigned to several seals, a location univocally identifies a seal; it can be thought of as its physical address.

A tree of seals is rendered, in PSN, by the parallel composition of the seals in the tree. In this sense, the physical and the logical topology are separated: the space of the physical locations is flat, and each location hosts at most one seal, each seal resides at a distinct physical location(this gives us the physical distribution), but each seal knows the location at which its parent and its sons reside(this gives us the logical topology). For instance, an Seal term $s_1[P_1 \parallel P_2 \parallel s_2[Q_1] \parallel s_3[Q_2]]$, where P_1 and P_2 are the local processes of s_1, and $Q_i (i = 1, 2)$ is a local process of m_i(i.e., m_i has no subseals), becomes in PSN:

$h : s_1[P_1|P_2], root, \{k_1, k_2\}) \parallel (k_1 : s_2[Q_1], h, \{\}) \parallel (k_2 : s_3[Q_2], h, \{\})$
where h, k_1, k_2 are different location names, root is a special name indicating the outermost location, and $\parallel$ is the parallel composition of located seals. Since seals may run at different physical sites, they communicate with each other by means of asynchronous messages.

We use $m, n, \ldots$ to range over names, $h, k, \ldots$ to range over locations, $p, q, \ldots$ to range over both names and locations. The syntax of PSN is shown as follows: A term of PSN, a net, is the parallel composition of agents and

Net $A ::= 0 \mid A_1 \mid\mid\mid A_2 (\nu p) A \mid Agent \mid h(MsgBody)$
Agent $Agent ::= (h : n[P], k, SS)$
Process $P ::= 0 \mid P_1 \mid P_2 \mid (\nu n)P \mid M.p \mid M[P] \parallel wait.P \mid \{MsgBody\}$
Action $M ::= x \mid n \mid \bar{x}^{\eta}(\vec{y}) \mid x^{\eta}(\lambda \vec{y}) \mid \bar{x}^{\eta}\langle \vec{y} \rangle \mid x^{\eta}\langle \vec{y} \rangle$
Message $MsgBody ::= write(c, x) \mid Okwrite \mid send(c, \eta, n) \mid$
$moveAck \mid move(h) \mid Okmove(h)$

messages, with some names possibly restricted. An agent is a located seal. Located seal is the basic unit of PSN, and represent seals of Seal with their local processes. Messages include two kinds. One kind is the messages that the requestor sends to the receiver, to ask for services; Another kind is the acknowledagement messages that the services provider sends back to the requestor to notify the completion of services and to execute the next operation. The syntax of the processes inside located seals is similar to that of processes in Seal. The only additions are: the prefix wait.P, which appears in a seal when this has sent a request but has not received an answer yet; and the requests, which represent messages received from the requestors and not yet served. We use A to range over nets.

For example, for the following Seal program:
$s_1[c^{s_3}(y).y^{s_2}(x).0 \parallel s_3[\bar{c}^{\uparrow}(c_1).0] \parallel s_2[\bar{c_1}^{\uparrow}(z).0]]$
the PSN is:
$\{\langle r.r_n[0], rp, \{\ell_1\}\rangle,$
$\langle \ell_1 : s_1[rcomm(c, y, son(s_3)).rcomm(y, x, son(s_2)).0], r, \{\ell_2, \ell_3\}\rangle,$
$\langle \ell_2 : s_2[scomm(c_1, z, fath).0], \ell_1, \{\}\rangle,$
$\langle \ell_3 : s_3[scomm(c, c_1, fath).0], \ell_1, \{\}\rangle\}$

3. Operational Semantics

Operational Semantics Based On Transitions

Our operational semantics is based on a transition system. In this kind of formalism the semantics of a program is given in terms of the transitions which can make from one configuration to another. The execution of a program is then modeled by a sequence of configurations with transitions, starting from a suitable initial configuration. The transitions are given by a transition relation $\mapsto \subset Conf \times Conf$ (where $Conf$ is the set of configurations).

In transition systems, a configuration usually consists of something like the statement that is to be executed, plus some extra state information. The configuration that we shall use will have a rather complex structure. Formally we define the set of $Conf$ by:

$Conf\{\langle l_i : s_i[cl_i], f_i, SS_i\rangle\}, i \in 1 \dots n$

Where l_i denotes the location of the seal, s_i denotes the name of the seal, cl_i denotes the actionlist of seal s_i, f_i denotes the location of the parent seal and SS_i denotes the location set of the subseals.

According to the above definition of configurationčňthe initial configuration is the final transformation result of Seal source program P to PSN, i.e,

$conf^0 = PSN(P) = \{\langle l_1 : s_1[cl_1], f_1, SS_1\rangle, \dots, \langle l_n : s_n[cl_n], f_n, SS_n\rangle\}$

the terminal configuration is the state that all the parallel process' action queues become empty, i.e.,

$conf^\sharp = \{\langle l_1 : s_1[0], f_1, SS_1\rangle, \dots, \langle l_n : s_n[0], f_n, SS_n\rangle\}$

Now, having an intuitive understanding of the meaning of the programming constructs, it is rather easy to give the corresponding transition rules.

Throughout this section, whenever we write $\langle\alpha\rangle \triangleright \rho$, we require that $\langle\alpha\rangle \notin \rho$.

(1) local communication

(T1) $(h : n[scomm(c, y, loc).cl'_1 \triangleright rcomm(c, x, loc).cl'_2 \triangleright cl], f, SS) \triangleright \rho'$
$\mapsto (h : n[cl'_1 \triangleright cl'_2\{y/x\} \triangleright cl], f, SS) \triangleright \rho'$

(2) local son to parent communication

(T2) $(l_1 : n_1[scomm(c, x, fath).cl'_1 \triangleright cl_1, f_1, SS_1) \triangleright \rho'$
$\mapsto (l_1 : n_1(wait.cl'_1 \triangleright cl_1 \triangleright fath\{write(c, x)\}], f_1, SS_1) \triangleright \rho'$
(if $fath = f_1$)

(T3) $(l_2 : n_2[rcomm(c, y, son(n_1)).cl'_2 \triangleright \{write(c, x), l_1\}], f_2, SS_2) \triangleright \rho$
$\mapsto (l_2 : n_2[cl'_2\{x/y\} \triangleright cl_2 \triangleright l_1\{Okwrite\}], f_2, SS_2) \triangleright \rho'$
(if $l_1 \in SS_2$)

(T4) $(l_1 : n_1[wait.cl'_1 \triangleright cl_1 \triangleright \{Okwrite, fath\}], f_1, SS_1) \triangleright \rho'$
$\mapsto (l_1 : n_1[cl'_1 \triangleright cl_1], f_1, SS_1) \triangleright \rho'$ (if $fath = f_1$)

(3) parent to son communication

(T5) $(l_1 : n_1[scomm(c, x, son(n_2)).cl'_1 \triangleright cl_1], f_1, SS_1) \triangleright \rho'$
$\mapsto (l_1 : n_1[wait.cl'_1 \triangleright cl_1 \triangleright l_2\{write(c, x)\}], f_1, SS_1) \triangleright \rho'$
(if $l_2 = Loc(n_2)$ and $n_2 \in SS_1$)

(T6) $(l_2 : n_2[rcomm(c, y, fath).cl'_2 \triangleright cl_2 \triangleright \{write(c, y), l_1\}], f_2, SS_2) \triangleright \rho'$
$\mapsto (l_2 : n_2[cl'_2\{x/y\} \triangleright cl_2 \triangleright l_1\{Okwrite\}], f_2, SS_2) \triangleright \rho' \quad$ (if $f_2 = l1$)

For space limitation, we omit the transition rules of process movement, which include local movement, parent to son movement and son to parent movement. These rules are similar to the above communication transition rules.

An Execution Sample of PSN

For the sample presented in section 3, we apply the transition rules in subsection 4.1, we get the following transition steps:

S1: Suppose $\rho = \rho_1 \triangleright \rho_2 \triangleright \rho_3$, there

$\rho_1 = \{\langle l_1 : s_1[rcomm(c, y, son(s_3)).rcomm(y, x, son(s_2)).0], r, \{l_2, l_3\}\rangle\}$,

$\rho_2 = \{\langle l_2 : s_2[scomm(c_1, z, fath).0], l_1, \{\}\rangle\}$,

$\rho_3 = \{\langle l_3 : s_3[scomm(c, c_1, fath).0], l_1, \{\}\rangle\}, then$

S2: After transition T2, ρ becomes

$\rho_1 \triangleright \{\langle l_2 : s_2[wait.0 \triangleright l_1\{write(c, z)\}], l_1, \{\}\rangle\} \triangleright \{\langle l_3 : s_3[wait.0 \triangleright l_1\{write(c, c_1)\}], l_1, \{\}\rangle\}$

S3: Using T3 transition, becomes

$\{\langle l_1 : s_1[rcomm(c_1, x, son(s_2)).0 \triangleright l_3\{Okwrite\} \triangleright \{write(c_1, z), l_2\}], r, \{l_2, l_3\}\rangle\} \triangleright \{\langle l_2 : s_2[wait.0\}], l_1, \{\}\rangle\} \triangleright \{\langle l_3 : s_3[wait.0\}], l_1, \{\}\rangle\}$

S4: Using T4 transition, becomes

$\{\langle l_1 : s_1[rcomm(c_1, x, son(s_2)).0 \triangleright \{write(c_1, z), l_2\}], r, \{l_2, l_3\}\rangle\} \triangleright \{\langle l_2 : s_2[wait.0\}], l_1, \{\}\rangle\} \triangleright \{\langle l_3 : s_3[0\}], l_1, \{\}\rangle\}$

S5: Using T3, T4 transition continously, we get

$\{\langle l_1 : s_1[0], r, \{l_2, l_3\}\rangle\} \triangleright \{\langle l_2 : s_2[0], l_1, \{\}\rangle\} \triangleright \{\langle l_3 : s_3[0], l_1, \{\}\rangle\}$

4. Conclusion and future work

We have presented an abstract machine for the Seal calculus, and discussed briefly its operational semantics based on transition system. The main originality of our abstract machine lies in the fact that an operational semantics based on transition system is given not a reduction one, this work facilitates the implementation and constitutes the first step in a potential series of more and more refined abstract machies, getting us closer to a provably correct implementation of the Seal calculus.

Finally let us point out some directions in which further work could be done. First, it would certainly be worthwhile to see whether for this kind of language also a denotational semantics can be developed, and possibly proved equivalent to the current operational semantics. Maybe the representation used here for parallel processes could be adapted to denotational semantics in such a way that a clear description is possible. Also this kind of operational semantics could be a good basis to explore the possibility of automatic implementation of mobile computation languages by means of a interpreter.

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] G.Castagna and F.Zappa. The Seal Calculus Revisited. *In Proceedings 22th FST-TCS*, number 2556 in LNCS. Springer, 2002.

[3] M.Bugliesi and G.Castagna. Secure safe ambients. *In Proc. of POPL'01*, pages 222-235. ACM Press, 2001.

[4] F.Nielson, H.Riis Nielson, R.R.Hansen, and J.G.Jensen. Validating firewall in mobile ambients. *In Proc.CONCUR'99*, number 1664 in Lecture Notes in Computer Science, pages 463-477. Springer-Verlag, 1999.

[5] J.Vitek and G.Castagna. Towords a calculus of secure mobile computations. *Proceedings Workshop on Internet Programming Languages*. Chicago, Illinois, USA, Lectures Notes in Computer Science 1686, Springer, 1998.

[6] 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.

[7] L.Cardelli, Ambit.http://www.luca.demon.co.uk/Ambit.html.1997.

[8] L.Cardelli. Mobile ambient synchronization, *Technical Report* 1997-013, Digital SRC, 1997.

[9] Stephan Diehl. A generative methodology for the design of abstract machines. *Science of Computer Programming*. 2000. 38. 125-142.

[10] G.Berry and G.Boudol. The chemical abstract machine. *Theoretical Computer Science*, vol.96,1992.

[11] D.Sangiorgi and A.Valente. A Distributed Abstract Machine for Safe Ambients. *In Proceedings of the 28th ICALP*, volume 2076 of LNCS. Springer-Verlag, 2001.

[12] Zhang Jing, Zhang Li-Cui and Jin Cheng-Zhi. A Distributed Implementation for the Seal Calculus. *To Appear in Proceedings of the First International Symposium on Pervasive Computations and Applications(SPCA06)*. Urumchi, Xin Jiang, P.R.China.