类型化移动资源[*]

傅 城[+], 尤晋元

(上海交通大学 计算机科学与工程系,上海 200030)

Typing Mobile Resources

FU Cheng[+], YOU Jin-Yuan

(Department of Computer Science and Engineering, Shanghai Jiaotong University, Shanghai 200031, China)

+ Corresponding author: Phn: +86-0-13818722138, E-mail: frankfch@sjtu.edu.cn, http://dctc.sjtu.edu.cn/~Fu.C

Received 2003-12-12; Accepted 2004-11-03

Fu C, You JY. Typing mobile resources. *Journal of Software*, 2005,16(5):979−990. DOI: 10.1360/jos160979

Abstract: A kind of interference, called direct access interference, is found in the calculus of Mobile Resources (MR), which will cause more damage than the grave interference one finds in the calculus of Mobile Ambients, because in MR malicious environments or contexts can freely access the sensitive resources inside a process. This kind of interference should be regarded as a program error. To control the direct access interference, we devise a variant of MR, the calculus of Safe Mobile Resources (SR). The authors use a type system to avoid the occurrence of all direct access interferences. Due to the study, the grave interference is a special form of the direct access interference, which is also controlled in SR. At the end of the paper, several examples are provided to illustrate how to use the new calculus and how robust it is.

Key words: concurrency; mobile computing; process calculi; type system; mobile resources

摘 要: 在移动资源演算(MR)中发现了一种干扰现象,称为直接访问干扰,该现象比移动灰箱演算(MA)中的墙干扰现象更具破坏力,因为在 MR 中恶意的环境或上下文可以不受限制地访问进程内部的敏感资源.因而该干扰问题当被视为一种程序运行错误.为了控制直接干扰现象,提出了一种 MR 的变体:安全移动资源演算(SR).它使用了一种类型系统来避免所有的直接访问干扰的发生.基于该研究,MA 中的强干扰现象实际上是直接访问干扰的一种特殊形式,自然地,在 SR 中也得到了相应的控制.最后给出一些用例,说明如何使用新设计的演算系统,以及它的健壮性.

关键词: 并发;移动计算;进程演算;类型系统;移动计算

中图法分类号: TP393 **文献标识码**: A

1 Introduction

The calculus of Mobile Ambients[1] (MA) is proposed for better modeling and describing the properties of the

* Supported by the National Natural Science Foundation of China under Grant No.60173033 (国家自然科学基金)

FU Cheng was born in 1978. He is a Ph.D. candidate at the Department of Computer Science and Engineering, Shanghai Jiaotong University. His current research areas are process calculi, bioinformatics, embedded computing, and mobile computing. **YOU Jin-Yuan** was born in 1939. He is a professor and doctoral supervisor at the Department of Computer Science and Engineering, Shanghai Jiaotong University. His research areas are operating systems and distributed systems.

mobile processes than other calculi (such as π-calculus). In MA, each process, who wants to move, should initiate the movement activity by itself. For instance, $n[\mathbf{in}\ m.P]|m[Q] \rightarrow m[n[P] \mid Q]$. The mobile process $n[\mathbf{in}\ m.P]$ initiates the **in** m primitive. In some cases, many objects are not able to initiate the movement, however, they may be regarded as mobile because someone can take them from one place to another. Here arises a question: how do we model mobile resources of this kind? Suppose we have the situation that process P will fetch the resource r from place n to place m. In MA, the environment is modeled like this:

$$P = tmp[\mathbf{in}\ n.\mathbf{in}\ r.\mathbf{out}\ n.\mathbf{in}\ m] \mid P'$$
$$Places = n[r[\mathbf{open}\ tmp \mid Q_2] \mid Q_1] \mid m[R]$$
$$Env = P \mid Places$$
$$Env \rightarrow^* P' \mid n[Q_1] \mid m[R \mid \mathrm{r}[Q_2]]$$

We have to add a worker process $tmp[\mathbf{in}\ n.\mathbf{in}\ r.\mathbf{out}\ n.\mathbf{in}\ m]$ to take the mobile primitive into r, and to put an **open** primitive into ambient r to be ready for unleashing certain activities. This model has two drawbacks: (1) we have to construct many assistant processes when designing the mobile resources of this kind. This makes the application designers contribute more commitment to the mobility controlling than to the application business logic; (2) The course of the movement of r is not atomic, which may cause much more serious concurrency problems. In the above example, it totally produces five reduction steps. Although, in Ref.[1], Cardelli has suggested several ways to obtain the atomicity, much more complex constructs will be used when implementing the atomic function.

We find that the calculus of Mobile Resources[2] (MR), which is proposed to describe and analyze the systems containing mobile and nested computing devices that may have resource and access constraints, is more appropriate to model the application of the mobile resources which are not able to initiate the movement activity. Moreover the course of the process movement is syntactically atomic when it is taken from one place to another. The three primitives in MR allow: a process to be taken from one place to another, $m\ \mathbf{to}\ \mathbf{co}(n).p \mid m[q] \mid n[r] \rightarrow p \mid m[\text{-}] \mid n[q \mid r]$; a name to be consumed in nested places, $nm\ \mathbf{co}(a).p \mid n[m[a]] \rightarrow p \mid n[m[\text{-}]]$; a process to be removed, $\mathbf{del}\ n.p \mid n[q] \rightarrow p$. In the second primitive, nm is a name path, it allows the access of a inside the ambient m which in turn is inside n. This is called *direct access*. It can be also applied in the first primitive. The MR computational model is based on the notion of taken and given (a variant of movement).

We begin our work with a study of *interference* in MR. From Ref.[3], we know that the coexistence of plain and grave interferences* can cause a poor algebraic theory, difficulties in writing correct programs and difficulties in proving behavioral properties of processes. We find similar grave interferences in MR, for example:

$$n\ \mathbf{to}\ \mathbf{co}(m).p \mid n[r] \mid \mathbf{del}\ m.q \mid m[\text{-}]$$

In this process, $n\ \mathbf{to}\ \mathbf{co}(m)$ wants to take r from n to m, and **del** m wants to remove the ambient m. Then we have two nondeterministic consequences while the process reduces: one is $p|n[\text{-}]|\mathbf{del}\ m.q|m[r]$ and the other is $n\ \mathbf{to}\ \mathbf{co}(m).p\ |n[r]|q$. The first result can continue to reduce to $p|n[\text{-}]|q$. The two results are totally logically different from each other. Such kind of interference occurs among the parallel processes that want to operate the same ambient.

However in MR, there is another kind of interference which will cause more dangerous consequences that can affect much more processes. This interference may occur among the processes located in different nested ambients. Let's examine the following two processes:

$$r_1 = nm\ \mathbf{co}(a).p \mid n[m[a]]$$
$$r_2 = n'nm\ \mathbf{to}\ \mathbf{co}(m').q \mid n'[r_1] \mid m'\ [\text{-}]$$

* *plain interference* is initially found in CCS[4] and π-calculus[5] which is caused by two or more redexes sharing the same interacting partners; *grave interference* is found in the calculus of Mobile Ambients which will cause the shared ambient to perform logically different interactions.

nm **co**(a) in process r_1 will consume the name a inside ambient m which is nested in n. It is clear that r_1 can only reduce to $p \mid n[m[\text{-}]]$. But when r_1 is put in an environment as shown in process r_2, $n'nm$ **to co**(m') in process r_2 wants to take name a from m to m'. If r_2 makes the reduction step by the movement of a, then nm **co**(a) will fail to consume a in m. The resource is lost when the process is put in improper environment. In contrast with the previous example of grave interference, here the same ambient can be shared through the nested environment. The root reason to cause this phenomenon is that MR allows *direct access* as mentioned above. So this kind of interference is called *direct access interference*. To illustrate how dangerous the direct access interference is, let's examine another example:

$$r_A = !out\ \mathbf{to}\ \mathbf{co}(network) \mid out[key[Msg]]$$
$$r_B = !network\ \mathbf{to}\ \mathbf{co}(in) \mid in[\text{-}]$$
$$r_{comm} = (key)(\ r_A \mid r_B) \mid network[\text{-}]$$
$$Env = \mathcal{C}(r_{comm}) \text{ where } \mathcal{C}(\text{-}) = trap\ network\ \mathbf{to}\ \mathbf{co}(treasure) \mid treasure[\text{-}] \mid trap[\text{-}]$$

r_A wants to send a secure message to r_B through a public network. The message Msg is encrypted by a key key that is shared between r_A and r_B. But when r_{comm} reduces to

$$(key)(\ r_A \mid r_B \mid network[key[Msg]])$$

the process $trap\ network$ **to co**($treasure$) in Env then gets opportunity to intercept the secure message $(key)key[Msg]$ though key is unknown. According to cryptology, if enough cipher is gained, then we have more chances to decipher it. It is very dangerous to model secure communication like this example. Even the example, Digital Signature Card in Ref.[2] can be interfered by a malicious environment. This example will be discussed in Section 4.

We compare the direct access interference with grave interference, and find that grave interference is indeed a special form of the direct access interference. If the name path $trap\ network$ in the capability $trap\ network$ **to co**($treasure$) in the above example becomes an empty string, then this is just one form of grave interferences. Thus in SR, we only consider the problems caused by direct access interferences because grave interferences problems are already included in the direct access interferences.

As a result of our study in interferences in MR, we regard the occurrence of direct access interference as a proramming error. The existence of direct access interference has following severe results:

1. free names can be easily accessed by the nested environment which results in data loss;
2. secure resources are hard to be modeled;
3. programs cannot behave in the expected way in all contexts.

Before we design the new calculus, we try to construct some normal forms or patterns to avoid drawbacks, but we failed at last. The root reason is that, as long as the activity occurs through a free ambient, the environment must have the chance to access it. The current theory of MR is unable to solve the problems mentioned above. Moreover, the lack of a proper type system is another reason why MR gains so many problems.

To eliminate these drawbacks, we enhance the MR calculus with full coactions semantically. We call it *Safe Mobile Resources* (SR). And in this paper, we build a type system for the calculus to control the direct access interferences. According to MA paper series, the MR processes are also designed to have mobility and threadness types. In addition, we use *resource type set* to represent what types the processes or ambients can provide the resources. Then we prove the soundness of the type system. Moreover we prove that all well typed processes in SR do not contain any direct access interferences. The type system is partly inspired by SA[3]. In SR, resources can be consumed and are uncopiable. We also allow a certain kind of subtyping on the process types. At the end of this paper, we give some examples to illustrate how to use it and to show its robustness.

2 The Calculus of Safe Mobile Resources

The calculus of Mobile Resources (MR) allows an ambient to be taken, given and removed, and allows a name to be consumed. But it has no control on the access to the certain ambient when it is taken or given. In Safe Mobile Resources (SR), we rectify the syntax and semantics so that any activities relevant to taking, giving, deletion and consumption can occur only if all participants agree. We achieve the goal by add the coactions: **cotof** (allow being moved from), **cotod** (allow sth. move in) and **codel** (allow remove) where the first and second ones are the coactions of **to** (take and give) and the third one is the coaction of **del** (delete). Its three essential behaviors are listed below:

$$n\ \mathbf{co}(a).p \mid n[a.q] \rightarrow p \mid n[q] \tag{1}$$

$$n\ \mathbf{to}\ m\ .p \mid n[res[\mathbf{cotof}\ n.q_1 \mid q_2] \mid q_3] \mid m[\mathbf{cotod}\ m.r] \rightarrow p \mid n[q_3] \mid m[res[q_1 \mid q_2] \mid r] \tag{2}$$

$$\mathbf{del}\ m.p \mid \check{n}[\mathbf{codel}\ m.q] \rightarrow p,\ m \in \check{n} \tag{3}$$

For the first formula, we use capability n $\mathbf{co}(a)$ to denote that it wants to consume the resource a located in ambient n. p and q are processes. There is a name (resource) a in ambient n. n $\mathbf{co}(a)$ and a in n can make a contract and lead to a reduction step. Moreover in n $\mathbf{co}(a)$ we call the part before (left to) $\mathbf{co}(a)$ is a *path*. We can use an ambient name string to denote a path such as n_1n_2m. For simplicity, we use γ to represent a nullable name string, and δ for a non-empty name string. So if a capability wants to consume a resource a in the process $n_1[n_2[m[a]]]$, we use n_1n_2m $\mathbf{co}(a)$ to denote it, or let $\gamma = n_1n_2m$, then use γ $\mathbf{co}(a)$ instead. The process $n_1[n_2[m[a]]]$ can also be simplified as $\mathcal{C}_\gamma(a)$ by defining $\mathcal{C}_\gamma(\text{-}) = n_1[n_2[m[\text{-}]]]$ where we call $\mathcal{C}_\gamma$ a *path context*.

For Eq.(2), capability n **to** m represents that the qualified process in ambient n will be taken into ambient m. The process $res[\mathbf{cotof}\ n.q_1 \mid q_2]$ in ambient n is the qualified process because there is the process $\mathbf{cotof}\ n.q_1 \mid q_2$. The capability **cotof** n allows the ambient enclosing it to be ready for being taken out of the ambient n. For this example, ambient res cannot be taken out of any ambient other than ambient n. Capability **cotod** m allows a process to be given into ambient m only if the capability is inside ambient m. As for (2), the reduction contract is made among three participants, and all of them must hold **to**, **cotof** and **cotod** respectively at a proper location. Furthermore, SR allows *direct access* by means of name path for movement primitive. The part before **to** is the source path while the part after it is the destination path. If the process r will be taken out of the nested ambients $n_1[n_2[n_3[\text{-}]]]$ and given into $m_1[m_2[m_3[\text{-}]]]$, we use $n_1n_2n_3$ to $m_1m_2m_3$ to denote it. Here, for simplicity, we use path context instead, for instance, let $\gamma_1 = n_1n_2n_3$, $\gamma_2 = m_1m_2m_3$, $\mathcal{C}_{\gamma_1}(\text{-}) = n_1[n_2[n_3[\text{-}]]]$ and $\mathcal{D}_{\gamma 2}(\text{-}) = m_1[m_2[m_3[\text{-}]]]$, then we have γ_1 **to** $\gamma_2.p \mid \mathcal{C}_{\gamma_1}(res[\mathbf{cotof}\ n_3.q_1 \mid q_2] \mid q_3) \mid \mathcal{D}_{\gamma_2}(\mathbf{cotod}\ m_3.r) \rightarrow p \mid \mathcal{C}_{\gamma_1}(q) \mid \mathcal{D}_{\gamma_2}(res[q_1 \mid q_2] \mid r)$.

For (3), we use **del** m to initiate deletion activity. Parameter m is the target ambient to be deleted. **codel** m is the coaction of **del** m. It allows the ambient m to be ready for the removal when inside it. As mentioned above, deletion capability also supports path name. We use **del** γm where $\gamma = n_1n_2$ to represent that it wants to remove the ambient m in n_2 that is nested in n_1. Moreover, SR supports certain kind of alias, for instance, $\{n_1, n_2, m\}[p]$ is a process where the outer ambient has three names n_1, n_2 and m. All of them denote the same ambient. We use $\check{n}$ to denote a name set, $\check{n}[\text{-}]$ to denote an ambient that has an alias name set, and $\check{n}$ to denote a collection of the bound names.

For mobile calculi, there are two forms to depict the infinite behaviors: $!p$ representing the unbounded number of copies of p in parallel and the recursive construct **rec** $x.p$. MR uses the former, but we prefer the latter. This is because most interferences are caused by parallel process where it is hard to devise a type system for Par rule when adopting the form of $!p$. But if we use recursive construct **rec** $x.p$, many former replication constructs can avoid being typed by Par rule in the type system. In general every replicated form $!p$ can be denoted by **rec** $x.(p \mid x)$ using the recursive construct.

Let $\mathcal{N}$ be a countable set of names ranged over by a, b, ..., n, m. Generally, we use a, b to denote resource names, and m, n to denote ambient names; the names used in the process can be easily distinguished by contexts. We use x, y to range over the set of all recursive variables $\mathcal{V}$. The set of all processes is denoted by $\mathcal{P}$ (ranged over by p, q, ...) and the set of capabilities Λ (ranged over by λ). In typed version, we use a set of restricted names ($\check{n}$:$\check{U}$)

defined as $(n_1{:}U_1)(n_2{:}U_2)...(n_k{:}U_k)$ where $\check{n}=\{n_1,n_2,...,n_k\}$ and $\check{U}=\{U_1,U_2,...,U_k\}$ to represent the typed names in an abbreviated form. And **rec** x is a binder for the free recursive variable x in a process.

Capabilities are the expressions that are not names or recursive variables. We write $n(p)$ for all names of process p, and $fn(p)$ for all free names of the process p, and $n(\lambda)$ and $fn(\lambda)$ for those of the capability λ. For free recursive variables, we use $fv(p)$ to denote the set of all free recursive variables in p. $\rightarrow$ stands for one-step reduction. $\rightarrow^*$ stands for the transitive closure of $\rightarrow$. The definition of structural congruence relation $\equiv$ is a standard which is listed in Table 2. *Context* is defined as standard. *Path context* is defined as follows:

$$\mathcal{C}_\varepsilon(\text{-}) = (\text{-}) \qquad\qquad \mathcal{C}_{n\gamma}(\text{-}) = n[\mathcal{C}_\gamma(\text{-}) \mid p]$$

The SR Grammars and syntax are shown in Table 1. The reduction rules are in Table 3. To simplify our objective of the study, we require every process for reduction has no free recursive variables. Thus the rules in Table 3 are defined on all processes that have no free recursive variables. The rules for structural congruence and typing rules (in Section 3) are defined on all processes.

Table 1 The syntax of safe mobile resources

$a, b, ..., n, m$	*names*	$\check{n} ::= n_1, n_2, ..., n_k$	*set of names*
p, q, r	*processes*	$\gamma ::= \varepsilon \mid n\gamma$	*empty-able name path*
x, y, z	*recursive variables*	$\delta ::= n\gamma$	*non-empty name path*
Capabilities		*Processes*	
$\lambda ::= \delta_1$ **to** δ_2	*move process*	$p, q ::= \mathbf{0}$	*nil*
$\mid$ **cotof** n	*allow being moved*	$\mid\ p \mid q$	*parallel*
$\mid$ **cotod** m	*allow enter*	$\mid\ \lambda.p$	*prefixing*
$\mid\ a$	*resource*	$\mid\ (n : U)\ p$	*restriction*
$\mid\ \gamma$ **co**(a)	*consume resource*	$\mid\ x$	*recursive variable*
$\mid$ **del** γm	*remove ambient*	$\mid$ **rec** $x.p$	*recursive process*
$\mid$ **codel** m	*allow remove*	$\mid\ \check{n}[p]$	*slot*

Table 2 Structural congruence

$p \equiv q \Rightarrow n[p] \equiv n[q]$	(*Struct Nest*)	$(p \mid q) \mid r \equiv p \mid (q \mid r)$	(*Struct ParAss*)
$(n_1 : U_1)(n_2 : U_2)p \equiv (n_2 : U_2)(n_1 : U_1)\ p$	(*Struct ResRes*)	$p \equiv p$	(*Struct Refl*)
$n \notin fn(p) \Rightarrow (n : U)(p \mid q) \equiv p \mid (n : U)\ q$	(*Struct ResPar*)	$p \mid \mathbf{0} \equiv p$	(*Struct ParNil*)
$m \neq n \Rightarrow (n : U)\ m[p] \equiv m[(n : U)p]$	(*Struct ResNest*)	$(n : U)\ \mathbf{0} \equiv \mathbf{0}$	(*Struct ResNil*)
$p \equiv q, q \equiv r \Rightarrow p \equiv \mathrm{r}$	(*Struct Trans*)	$q \equiv p \Rightarrow p \equiv q$	(*Struct Symm*)
$p \equiv q \Rightarrow (n : U)p \equiv (n : U)q$	(*Struct Res*)	$p \equiv q \Rightarrow \lambda.p \equiv \lambda.q$	(*Struct Pre*)
$p \equiv q \Rightarrow \mathbf{rec}\ x.p \equiv \mathbf{rec}\ x.q$	(*Struct Rec*)	$p \mid q \equiv q \mid p$	(*Struct ParCom*)
$p \equiv q \Rightarrow p \mid r \equiv q \mid r$	(*Struct Par*)		

Table 3 Reduction rules of SR

$\gamma_1 n\ \mathbf{to}\ \gamma_2 m.p \mid \mathcal{C}_{\gamma1}(n[res[\mathbf{cotof}\ n.q_1 \mid q_2] \mid r_1]) \mid \mathcal{D}_{\gamma2}(m[\mathbf{cotod}\ m.r_2])$ $\rightarrow p \mid \mathcal{C}_{\gamma1}(n[r_1]) \mid \mathcal{D}_{\gamma2}(m[res[q_1 \mid q_2] \mid r_2])$	(*Red Mov*)			
$\gamma\ \mathbf{co}(a).p \mid \mathcal{C}_\gamma(a.q) \rightarrow p \mid \mathcal{C}_\gamma(q)$	(*Red Act*)	$\mathbf{del}\ \gamma m.p \mid \mathcal{C}_\gamma(\check{n}[\mathbf{codel}\ m.q]) \rightarrow p \mid \mathcal{C}_\gamma(\mathbf{0})$	$(m \in \check{n})$	(*Red Del*)
$p \rightarrow p' \Rightarrow p \mid q \rightarrow p' \mid q$	(*Red Par*)	$p \rightarrow p' \Rightarrow (n : U)p \rightarrow (n : U)p'$ (*Red Res*)		
$p \rightarrow p' \Rightarrow \check{n}[p] \rightarrow \check{n}[p']$	(*Red Nest*)	$p \equiv q, q \rightarrow r, r \equiv s \Rightarrow p \rightarrow s$		(*Red Struct*)
$\mathbf{rec}\ x.p \rightarrow p\{\mathbf{rec}\ x.p/x\}$	(*Red Rec*)			

3 Types

In the type system of SR, we have three main types (with their forms): *capability type* (Cap$[T, \mathcal{R}]$), *ambient type* (Amb$[T, \mathcal{R}]$) and *process type* (Proc$[T, \mathcal{R}]$), where T is called the *mobility and threadness attribute* or *inner type*, and $\mathcal{R}$ is called *resource attribute* or *resource type set*. The capability type indicates what behavior (cause movement, deletion or resource consumption) will take place after it is consumed. Ambient type indicates that which type of the processes it can contain. The process type is used to depict the global behavior of the whole process. The type grammars for SR are shown in Table 4. For threadness, we use 0, 1 and ω to indicate a quiet process, a single-threaded process, and a multi-threaded process respectively. For mobility, we use ♀ to denote immobile attribute and ♂ to denote mobile attribute. For resource types, we use resource type set to indicate the ambient or process to contain the specified kind of resources. An ambient may provide some resources of the specified types (we use capitalized S with subscript to denote different resource types) inside it or allow the inner process to consume other resources. As for programming languages, the abstract symbol S can represent intrinsic types such as strings, integers and arrays etc., or custom types such as abstract data types and class types etc. For example, we may use $\mathcal{R} = \{S_1, S_2\}$ to show the resources type set for a certain ambient, and it describes that the resources of the type S_1 and S_2 are provided in the ambient. If the resources type set $\mathcal{R}$ is typed to a process, it indicates the process provides the resources of the type S_1 and S_2. An empty resource type set $\varnothing$ indicates that there is no activity related to any resources for a process, or no resources contained in a ambient. Furthermore, in SR, we have an auxiliary type: *location type* (denoted as Δn). We use location type to indicate which place a capability will access. For each capability type (as well as ambient type and process type), it has a source location type and a target location type. The source location type indicates that certain capability or process in the source location may disappear after reduction. The target location type indicates that certain process may appear in the target location. Now let's examine some examples to illustrate the SR type system:

- A capability that causes a process to be taken out: Cap[$♂^{1,\varnothing,\varnothing}$, $\varnothing$]
- A capability that provide a resource of type S for consumption: Cap[$♀^{1,\varnothing,\varnothing}$, $\{S\}$]
- A capability that will take a certain process from n to m: Cap[$♀^{1,\{\Delta n\},\{\Delta m\}}$, $\varnothing$]
- A capability that will remove a specified ambient n: Cap[$♀^{1,\{\Delta n\},\varnothing}$, $\varnothing$]
- An empty ambient: Amb[$♀^{0,\varnothing,\varnothing}$, $\varnothing$]
- An immobile ambient that can contain a multi-threaded process, provide the resources of type S_1 and S_2, and remove the ambient m: Amb[$♀^{\omega,\{\Delta m\},\varnothing}$, $\{S_1, S_2\}$]
- An immobile single-threaded process that won't provide any resources for consumption, and that will take a certain process from n to m: Proc[$♀^{1,\{\Delta n\},\{\Delta m\}}$, $\varnothing$]
- A mobile multi-threaded process that contains resources of type S_1 and S_2, and be typed as Proc[$♂^{\omega,\varnothing,\varnothing}$, $\{S_1, S_2\}$]

Table 4 Typing grammars

S	*resource type*	$\mathcal{R}$	*set of resource types*	Δn	*location types*
Ξ	*set of source location types*	Y	*set of target location types*		

X	::=	♀	*immobile*	Z	::=	0	*no thread*
	|	♂	*mobile*		|	1	*single thread*
ξ	::=	Δn	*location type*		|	ω	*multi threads*
Cap	::=	Cap[T, $\mathcal{R}$]	*capability type*	T	::=	$X^{Z,\xi,Y}$	*inner type*
Amb	::=	Amb[T, $\mathcal{R}$]	*ambient type*	W	::=	*Amb*	*ambient type*
Proc	::=	Proc[T, $\mathcal{R}$]	*process type*		|	S	*resource type*

We use X to range over the set of mobility type **X**= {♀, ♂}, Z over the set of threadness type $\mathbf{Z}= \{0, 1, \omega\}$, and ξ over the set of all location types **L** is defined as $\{\Delta n \mid n \in \mathcal{N}\}$. Ξ and Y represent the set of source location type and the set of target location type respectively, and both of them range over the set $2^{\mathbf{L}}$. T ranges over the set of inner type $\mathbf{T} = \{X^{Z,\Xi,Y} \mid X \in \mathbf{X}, Z \in \mathbf{Z}, \Xi \in 2^{\mathbf{L}}, Y \in 2^{\mathbf{L}}\}$. Moreover, **RES** represents the collection of all resources types, and we use S to range over the set **RES**. Furthermore, we use $\mathcal{R}$ to range over the set $2^{\mathbf{RES}}$ to represent the resource attribute for ambients and processes. The following is the detailed description for the three major kinds of types used in SR.

- Cap[T, $\mathcal{R}$] is the capability type where T indicates what mobility and threadness attribute of the capability, and $\mathcal{R}$ indicates what type of the resource the capability provides. For each typed capability, at most one resource type is contained in its resource attribute.

- Amb[T, $\mathcal{R}$] is the ambient type where T indicates the process of what mobility and threadness attribute the ambient can contain, and $\mathcal{R}$ restricts that the ambient can only contain the the resources of the specified type in $\mathcal{R}$.
- Proc[T, $\mathcal{R}$] is the process type where T indicates the mobility and threadness attribute of the process and $\mathcal{R}$ indicates that the process may provide the resources of the type specified in $\mathcal{R}$.

The typing rules are shown in Table 5. For each capability, it can only carry one thread because there is naturally only one activity for each capability. Therefore, each kind of capability is typed as single-threaded, which is shown in (SRT Cap) series rules. As for rule (SRT Amb), intuitively an ambient only allows a process whose inner attribute and resource attribute should equal to those of that ambient. But in fact, by rule (SRT Sub), the process can easily become “larger”, thus ambient can hold the process whose inner attribute and resource attribute may be “less than” the ambient. For instance, if the type of the ambient has ♂ attribute, it can contain the processes that has ♀ attribute as well as ♂. But an immobile ambient cannot contain a mobile process because mobile attribute is not “less than” immobile attribute. The “less than” relation has same meaning for threadness, resource type set and location type set, and it will formally appear to be a kind of subtype defined later. Rule (SRT Sub) is indeed for subtyping.

To simplify the typing result of the computation on the multiple kinds of types in SR type system, we define five commutative operators ○, | on **Z**, * on **X** and ♥, ● on **T** as follows:

*	♀	♂
♀	♀	♂
♂	♂	♂

○	0	1	ω
0	0	1	ω
1	1	1	ω
ω	ω	ω	ω

|	0	1	ω
0	0	1	ω
1	1	ω	ω
ω	ω	ω	ω

$$X_1^{Z_1,\,\Xi_1,\,Y_1} \heartsuit X_2^{Z_2,\,\Xi_2,\,Y_2} = (X_1 * X_2)^{Z_1|Z_2,\,\Xi_1\cup\Xi_2,\,Y_1\cup Y_2}$$
$$X_1^{Z_1,\,\Xi_1,\,Y_1} \bullet X_2^{Z_2,\,\Xi_2,\,Y_2} = (X_1 * X_2)^{Z_1\circ Z_2,\,\Xi_1\cup\Xi_2,\,Y_1\cup Y_2}$$

The reflexive order $\le_{\mathbf{X}}$ on **X**, $\le_{\mathbf{Z}}$ on **Z** and $\le_{\mathbf{T}}$ on **T** are defined as follows, where $\le_{\mathbf{X}}$ and $\le_{\mathbf{Z}}$ are linear orders:

$$♀ \le_{\mathbf{X}} ♂ \qquad\qquad 0 \le_{\mathbf{Z}} 1 \le_{\mathbf{Z}} \omega$$
$$X_1^{Z_1,\,\Xi_1,\,Y_1} \le_{\mathbf{T}} X_2^{Z_2,\,\Xi_2,\,Y_2} \Leftrightarrow X_1 \le_{\mathbf{X}} X_2 \wedge Z_1 \le_{\mathbf{Z}} Z_2 \wedge \Xi_1 \subseteq \Xi_2 \wedge Y_1 \subseteq Y_2$$

Then we define a transitive, reflexive relation $\le$ of subtyping on the types of process, which is shown below:

$$\mathrm{Proc}[T_1, \mathcal{R}_1] \le \mathrm{Proc}[T_2, \mathcal{R}_2] \Leftrightarrow T_1 \le_{\mathbf{T}} T_2 \wedge \mathcal{R}_1 \subseteq \mathcal{R}_2$$

We only allow subtyping on processes. There is no need to build subtype relation on ambient and capability type because it makes no sense for the comparison between ambients or capabilities in the calculus.

Theorem 1. If $\Gamma \vdash p{:}Proc$ and $p \equiv p'$, then $\Gamma \vdash p'{:}Proc$.

Theorem 2. If $\Gamma \vdash p{:}Proc$ and $p \rightarrow p'$, then $\Gamma \vdash p'{:}Proc$ with $Proc' \le Proc$.

The type system in SR is built to protect the processes from serious concurrency problems such as direct access interferences and grave interferences. As we know, plain interferences could be regarded as rational competition. But the coexistence of grave interference(in this paper, grave interference is regarded as a kind of direct access interference as mentioned in Section 1 and plain interference can cause serious results. The following theorem assures that under SR type system all processes run in a good behavior, that is, at most only plain interferences can occur in SR processes.

Theorem 3. If $\Gamma \vdash p{:}Proc$ then no direct access interferences or grave interferences occur in p.

Through Theorems 2 and 3, it is certain that each well typed process in SR won't cause any direct access interferences, and after reduction the process is also well typed without any direct access interferences.

Table 5 Typing rules

$\varnothing \vdash \Diamond$ (*Env Empty*) $\Gamma \vdash \Diamond \wedge n \notin dom(\Gamma) \Rightarrow \Gamma, n{:}W \vdash \Diamond$ (*Env n*) $\Gamma, n{:}W \vdash \Diamond \Rightarrow \Gamma, n{:}W \vdash n{:}W$ (*SRT n*)
$\Gamma \vdash n{:}Amb_n \Rightarrow \Gamma \vdash \mathbf{cotof}\ n{:}\mathrm{Cap}[\text{♂}^{1, \varnothing, \varnothing}, \varnothing]$ (*SRT Cap* **cotof**)
$\Gamma \vdash n{:}Amb_n \Rightarrow \Gamma \vdash \mathbf{cotod}\ n{:}\mathrm{Cap}[\text{♀}^{1, \varnothing, \varnothing}, \varnothing]$ (*SRT Cap* **cotod**)
$\Gamma \vdash m_1{:}\mathrm{Amb}[\text{♀}^{Z1, \Xi1, Y1}, \mathcal{R}_1] \wedge \Gamma \vdash m_2{:}\mathrm{Amb}[\text{♀}^{Z2, \Xi2, Y2}, \mathcal{R}_2] \Rightarrow \Gamma \vdash m_1\ \mathbf{to}\ m_2{:}\mathrm{Cap}[\text{♀}^{1, \{\Delta m\}, \{\Delta n\}}, \varnothing]$ (*SRT Cap* **to** 1)
$\Gamma \vdash m_1\ \mathbf{to}\ m_2{:}\ Cap \wedge \forall n \in fn(\gamma_1) \cup fn(\gamma_2) \Rightarrow \Gamma \vdash \gamma_1 m_1\ \mathbf{to}\ \gamma_2 m_2{:}Cap$ (*SRT Cap* **to** 2)
$\Gamma \vdash a{:}S \wedge \Gamma \vdash n{:}Amb_n \Rightarrow \Gamma \vdash a{:}\mathrm{Cap}[\text{♀}^{1, \varnothing, \varnothing}, \{S\}]$ (*SRT Cap a*)
$\Gamma \vdash a{:}S \wedge (\forall n \in fn(\gamma) \Rightarrow \Gamma \vdash n{:}Amb_n) \Rightarrow \Gamma \vdash \gamma\ \mathbf{co}(a){:}\mathrm{Cap}[\text{♀}^{1, \varnothing, \varnothing}, \varnothing]$ (*SRT Cap* **co**(*a*))
$\Gamma \vdash n{:}Amb_n \Rightarrow \Gamma \vdash \mathbf{codel}\ n{:}\mathrm{Cap}[\text{♀}^{1, \varnothing, \varnothing}, \varnothing]$ (*SRT Cap* **codel**)
$\Gamma \vdash m{:}Amb_m \wedge (\forall n \in fn(\gamma) \Rightarrow \Gamma \vdash n{:}Amb_n) \Rightarrow \Gamma \vdash \mathbf{del}\ \gamma m{:}\mathrm{Cap}[\text{♀}^{1, \{\Delta m\}, \varnothing}, \varnothing]$ (*SRT Cap* **del**)
$\Gamma \vdash \Diamond \Rightarrow \Gamma \vdash \mathbf{0}{:}\mathrm{Proc}[\text{♀}^{1, \varnothing, \varnothing}, \varnothing]$ (*SRT Zero*) $\Gamma, x{:}Proc \vdash \Diamond \Rightarrow \Gamma \vdash x{:}Proc$ (*SRT x*)
$\Gamma, a{:}S \vdash p{:}\mathrm{Proc}[X^{Z, \Xi, Y}, \mathcal{R}] \Rightarrow \Gamma \vdash (a{:}S)p{:}\mathrm{Proc}[X^{Z, \Xi, Y}, \mathcal{R}\text{-}\{S\}]$ (*SRT Res* 1)
$\Gamma, n{:}Amb_n \vdash p{:}\mathrm{Proc}[X^{Z, \Xi, Y}, \mathcal{R}] \Rightarrow \Gamma \vdash (n{:}Amb_n)p{:}\mathrm{Proc}[X^{Z, \Xi\text{-}\{\Delta n\}, Y\text{-}\{\Delta n\}}, \mathcal{R}]$ (*SRT Res* 2)
$\Gamma, x{:}Proc \vdash p{:}Proc \Rightarrow \Gamma \vdash \mathbf{rec}\ x.p{:}Proc$ (*SRT Rec*) $\Gamma \vdash p{:}Proc \wedge Proc \leq Proc' \Rightarrow \Gamma \vdash p{:}Proc'$ (*SRT Sub*)
$\Gamma \vdash \lambda{:}\mathrm{Cap}[T_1, \mathcal{R}_1] \wedge \Gamma \vdash p{:}\mathrm{Proc}[T_2, \mathcal{R}_2] \Rightarrow \Gamma \vdash \lambda.p{:}\mathrm{Proc}[T_1 \bullet T_2, \mathcal{R}_1 \cup \mathcal{R}_2]$ (*SRT Pre*)
$\Gamma \vdash p{:}\mathrm{Proc}[X_1^{Z1, \Xi1, Y1}, \mathcal{R}_1] \wedge \Gamma \vdash q{:}\mathrm{Proc}\ [X_2^{Z2, \Xi2, Y2}, \mathcal{R}_2] \wedge \Xi_1 \cap \Xi_2 = \varnothing$
$\Rightarrow \Gamma \vdash p\vert q{:}\mathrm{Proc}[X_1^{Z1, \Xi1, Y1} \heartsuit X_2^{Z2, \Xi2, Y2}, \mathcal{R}_1 \cup \mathcal{R}_2]$ (*SRT Par*)
$\Gamma \vdash n{:}\mathrm{Amb}[X^{Z, \Xi, Y}, \mathcal{R}] \wedge \Gamma \vdash p{:}\mathrm{Proc}[X^{Z, \Xi, Y}, \mathcal{R}] \Rightarrow \Gamma \vdash n[p]{:}\mathrm{Proc}[\text{♀}^{0, \Xi, Y}, \mathcal{R}]$ (*SRT Nest*)

4 Applications

In this section, we give some examples below to show how to use the SR calculus to model the usual applications. In the third example Digital Signature Card, we will show the robustness of SR by comparing it with MR.

4.1 Immobile server

The Server/Agent model can be constructed as follows:

$Server$ = $\mathbf{rec}\ x_s(n\ \mathbf{to}\ s.\mathbf{del}\ n.x_s) \mid s[\mathbf{rec}\ x_s.(\mathbf{cotod}\ s.Collect \mid x_s)]$

Agent = $a[\mathbf{rec}\ x_a.(\mathbf{cotof}\ n.Data \mid x_n)]$

SA = $Server \mid n[Agent \mid \mathbf{codel}\ n]$

In this model the mobile agent is taken into the server in order that the process *Collect* can access the *Data* in the agent. By assuming $\Gamma \vdash Collect{:}\mathrm{Proc}[X_s^{Z_s, \varnothing, Y_s}, \mathcal{R}_s]$ and $\Gamma \vdash Data{:}\mathrm{Proc}[X_d^{Z_d, \varnothing, Y_d}, \mathcal{R}_d]$, we have:

$\Gamma \vdash Server{:}\mathrm{Proc}[\text{♀}^{1, \{\Delta n\}, Y_s}, \mathcal{R}_s]$ $\Gamma \vdash Agent{:}\mathrm{Proc}[\text{♂}^{0, \{\Delta n\}, Y_d}, \mathcal{R}_d]$

The result shows that the process *Agent* is typed as mobile, and *Server* is typed as immobile.

4.2 Products delivery chain

We can easily construct a process transfer chain in SR, which is shown below:

$Places_n$ = $Place_1[\mathbf{rec}\ x_{pl}.(\mathbf{cotod}\ Place_1.x_{pl})] \mid \ldots \mid Place_n[\mathbf{rec}\ x_{pl}.(\mathbf{cotod}\ Place_n.x_{pl})]$

$MovePermission_n$ = $\mathbf{cotof}\ Place_1. \ldots .\mathbf{cotof}\ Place_n$

$Product_n$ = $Item[\mathbf{cotof}\ Warehouse.MovePermission_n.p]$

$Factory_n$ = $(Warehouse)\ (\mathbf{rec}\ x_{ra}.(Warehouse\ \mathbf{to}\ Place_1.x_{ra}) \mid Warehouse[\mathbf{rec}\ x_f.(Product_n \mid x_f)])$

$RelayAgent_i$ = $\mathbf{rec}\ x_{ra}.(Place_{i-1}\ \mathbf{to}\ Place_i.x_{ra})$ $i \geq 2$

RA_n = $RelayAgent_1 \mid \ldots \mid RelayAgent_n$

$User_n$ = $(Home)(Home[\mathbf{rec}\ x_h.(\mathbf{cotod}\ Home.x_h) \mid q] \mid \mathbf{rec}\ x_u.(Place_n\ \mathbf{to}\ Home.x_u))$

$DeliveryChain_n$ = $Factory_n \mid RA_n \mid Places_n \mid User_n$

This example shows how to model that the products are transferred to the end user from the factory. The process $DeliveryChain_n$ can be well typed by the SR type system with the assumption $\Gamma \vdash p{:}\mathrm{Proc}[X_p^{Z_p, \varnothing, Y_p}, \mathcal{R}_p]$ and $\Gamma \vdash q{:}\mathrm{Proc}[X_q^{Z_q, \varnothing, Y_q}, \mathcal{R}_q]$:

$\Gamma \vdash DeliveryChain_n{:}\mathrm{Proc}[\text{♀}^{\omega, \Xi, Y}, \mathcal{R}]$ where $\Xi = \mathrm{Y} = \{\Delta Place_1, \ldots, \Delta Place_n\}$ and $\mathcal{R} = \mathcal{R}_p \cup \mathcal{R}_q$

If the process $DeliveryChain_n$ is put in a malicious environment, it will be forbidden by the type system if the environment contains the capability that will access any public locations in $DeliveryChain_n$. This will assure that all cargo will be safely shipped to the destination.

4.3 Digital signature card

For this example, we will discuss the problem when modeling the application by using the calculus of Mobile Resources (MR) through comparing two versions of the Digital Signature Card. One is modeled by MR, the other by the calculus of Safe Mobile Resources (MR) proposed in this paper. Both of the two models use *reg*, *in* and *out* to represent a *register*, an *in-buffer* and an *out-buffer*. Enc_k and Dec_k are the two *k*-parameterized (in SR version, we have additional parameter *vpc*) processes respectively in charge of encrypting and decrypting resources where *k* is the key and it is only a shared secret between the two specified processes. Communication occurs between pairs such as Alice and Bob. Messages are transmitted through the public *network* that can be used by anyone.

4.3.1 Problems in the MR version

In Section 1, we have examined the three main reduction rules of MR. Other rules are the same as in SR. Then the application is modeled as follows:

$Enc_k \quad = !(reg)(in\ \mathbf{to}\ \mathbf{co}(reg\ k).reg\ \mathbf{to}\ \mathbf{co}(out) \mid reg[k[\text{-}]]) \mid in[\text{-}] \mid out[\text{-}]$

$Dec_k \quad = !(reg)(in\ \mathbf{to}\ \mathbf{co}(reg).reg\ k\ \mathbf{to}\ \mathbf{co}(out) \mid reg[\text{-}]) \mid in[\text{-}] \mid out[\text{-}]$

$Alice_{k,M} \quad = (m)(a)(a[Enc_k] \mid m[M] \mid m\ \mathbf{to}\ \mathbf{co}(a\ in).a\ out\ \mathbf{to}\ \mathbf{co}(network))$

$Bob_k \quad = (m)(b)(b[Dec_k] \mid m[\text{-}] \mid network\ \mathbf{to}\ \mathbf{co}(b\ in).b\ out\ \mathbf{to}\ \mathbf{co}(\mathrm{m})$

$SecretCom_M \quad = (k)(Alice_{k,M} \mid Bob_k) \mid network[\text{-}]$

Then we have:

$Alice'_k \quad = \quad (m)(a)(a[Enc_k] \mid m[\text{-}])$

$Bob'_{k,M} \quad = \quad (m)(b)(b[Dec_k] \mid m[M])$

$SecretCom_M \quad \to^* \quad (k)(Alice'_k \mid Bob_k \mid network[M]$

$\qquad \to^* \quad (k)(Alice'_k \mid Bob'_{k,M} \mid network[\text{-}])$

This model works well if there is only one communication on the *network*. But what will happen when more than one communication are carrying through the *network*? Assume there is another communication between $Jack_{k'}$ and $Joan_{k',M'}$ on the same *network*. Since *network* is a public name, it should be excluded from the secret communication model. So the *SecretCom* process for *M* and *M'* are redefined as follows:

$SecretCom_M \ = (k)(Alice_{k,M} \mid Bob_k)$

$SecretCom_{M'} \ = (k')(Joan_{k',M'} \mid Jack_{k'})$

Then we define the process *AllComs* to denote all the communications carrying through the *network*, which is modelled by:

$AllComs \ = \quad SecretCom_M \mid SecretCom_{M'} \mid network[\text{-}]$

$$\to^* \ (k)(k')(Alice'_k \mid Bob_k \mid Joan'_{k'} \mid Jack_{k'} \mid network[k[M] \mid k'[M']]) \tag{4}$$

When execution may move to (4), there are two encrypted messages on the *network*. Both *Bob* and *Jack* do not know which message is for them respectively, so *Bob* may fetch $k'[M']$ from the *network* and *Jack* may fetch $k[M]$. Because both of them do not hold the corresponding key, the messages can not be read by the wrong receiver. But the most critical problem is that the messages are lost, moreover, no one knows. The cause of this problem is that both receivers can observe the data on the *network* in this model. Furthermore, if the communication is put in a non-trusted environment, the malicious context could directly access the data inside the communication. This has been discussed in Section 1. The direct access interference in this model has not been controlled and cannot be controlled by the calculus. Moreover if a malicious user has decrypted the shared key, he can fabricate the messages to some receivers. This will cause much more serious security problems which violate the will of the authors who design the MR calculus.

4.3.2 The solution in SR: a virtual private channel

The SR version of Digital Signature Card uses the type system to check the resource type of the different messages on the *network*. We assume that different communications (user to user pair) have different resource types. Then by Theorem 2, the receiver process must be well typed after receiving the message. This means that every message will be delivered correctly in terms of the corresponding resource type. Now we will explain how it works. The $SecretCom_M$ and related processes are modeled as follows:

$$\begin{array}{lcl}
Enc_{k\ vpc} & = & (reg{:}W_r)(\mathbf{rec}\ x_1.(in_e\ \mathbf{to}\ reg\ k.reg\ \mathbf{to}\ out_e\ vpc.x_1)\ | \\
 & & reg[\mathbf{rec}\ x_2.(k[\mathbf{cotof}\ reg.\mathbf{cotof}\ out_e.\mathbf{cotof}\ network.\mathbf{codel}\ k\ |\ \mathbf{cotod}\ k]\ |\ x_2)])\ | \\
 & & in_e[\mathbf{rec}\ x_3.(\mathbf{cotod}\ in_e.x_3)]\ |\ out_e[\mathbf{rec}\ x_4.(vpc[\mathbf{cotod}\ vpc.\mathbf{cotof}\ out_e\ |\ c.\mathbf{codel}\ vpc]\ |\ x_4)] \\
Dec_k & = & (reg{:}W_r)(\mathbf{rec}\ y_1.(in_d\ \mathbf{to}\ reg.reg\ k\ \mathbf{to}\ out_d.\mathbf{del}\ reg\ k.y_1)\ |\ reg[\mathbf{rec}\ y_2.(\mathbf{cotod}\ reg.y_2)])\ | \\
 & & in_d[\mathbf{rec}\ y_3.(\mathbf{cotod}\ in_d.y_3)]\ |\ out_d[\mathbf{rec}\ y_4.(\mathbf{cotod}\ out_d.y_4)] \\
Alice_{k,vpc,M} & = & (n_1{:}W_{n_1})(in_e{:}W_{in_e})(out_e{:}W_{out_e})(Enc_k\ |\ n_1[env_1[\mathbf{cotof}\ n_1.\mathbf{cotof}\ in_e.\mathbf{cotof}\ k.M]]\ | \\
 & & n_1\ \mathbf{to}\ a\ in_e.a\ out_e\ \mathbf{to}\ network) \\
Bob_{k,vpc} & = & (m_1{:}W_{m_1})(in_d{:}W_{in_d})(out_d{:}W_{out_d})(Dec_k\ |\ m_1[\mathbf{rec}\ x.(\mathbf{cotod}\ m_1.x)]\ | \\
 & & network\ vpc\ \mathbf{to}\ b\ in_d.network\ vpc\ \mathbf{co}(\mathrm{c}).\mathbf{del}\ network\ vpc.b\ out_d\ \mathbf{to}\ m_1) \\
SecretCom_M & = & (vpc{:}W_{vpc})(k{:}W_k)(Alice_{k\ vpc,M}\ |\ Bob_{k\ vpc})
\end{array}$$

To compare with the MR version, we also put two communications on the *network* (one is Alice-Bob, and the other Joan-Jack), then *AllCom* and other related and auxiliary processes are modeled as:

$$\begin{array}{lcll}
Alice'_{k,vpc} & = & (n_1{:}W_{n_1})(in_e{:}W_{in_e})(out_e{:}W_{out_e})(Enc_k\ |\ n_1[\text{-}]) & \\
Bob'_{k,vpc,M} & = & (m_1{:}W_{m_1})(in_d{:}W_{in_d})(out_d{:}W_{out_d})(Dec_k\ |\ m_1[\mathbf{rec}\ x.(\mathbf{cotod}\ m_1.x)\ |\ env_1[M]) & \\
Joan_{k',vpc',M'} & = & (n_2{:}W_{n_2})(in_e{:}W_{in_e})(out_e{:}W_{out_e})(\ Enc_{k'}\ |\ n_2[env_2[\mathbf{cotof}\ n_2.\mathbf{cotof}\ in_e.\mathbf{cotof}\ k'.M']]\ | & \\
 & & n_2\ \mathbf{to}\ a'\ in_e.a'\ out_e\ \mathbf{to}\ network) & \\
Jack_{k',vpc'} & = & (m_2{:}W_{m_2})(in_d{:}W_{in_d})(out_d{:}W_{out_d})(Dec_{k'}\ |\ m_2[\mathbf{rec}\ \mathrm{y}.(\mathbf{cotod}\ m_2.y)]\ | & \\
 & & network\ vpc'\ \mathbf{to}\ b'\ in_d.network\ vpc'\ \mathbf{co}(\mathrm{c}).\mathbf{del}\ network\ vpc'.b'\ out_d\ \mathbf{to}\ m_2) & (5) \\
Joan'_{k',vpc'} & = & (n_2{:}W_{n_2})(in_e{:}W_{in_e})(out_e{:}W_{out_e})(Enc_{k'}\ |\ n_2[\text{-}]) & \\
Jack'_{k',vpc',M'} & = & (m_2{:}W_{m_2})(in_d{:}W_{in_d})(out_d{:}W_{out_d})(Dec_{k'}\ |\ m_2[\mathbf{rec}\ y.(\mathbf{cotod}\ m_2.y)\ |\ env_2[M']]) & \\
NetData & = & vpc[k[\mathbf{cotof}\ network\ |\ env_1[M]]\ |\ \mathbf{codel}\ vpc]\ |\ vpc'[k'[\mathbf{cotof}\ network\ |\ env_2[M']]\ |\ \mathbf{codel}\ vpc'] & \\
AllCom & = & SecretCom_M\ |\ SecretCom_{M'}\ |\ network[\mathbf{rec}\ z.(\mathbf{cotod}\ network.z)] & \\
 & \rightarrow^* & (vpc{:}W_{vpc})(vpc'{:}W_{vpc'})(k{:}W_k)(k'{:}W_{k'})(Alice'_k\ |\ Bob_k\ |\ Joan'_{k'}\ |\ Jack_{k'}\ | & \\
 & & network[\mathbf{rec}\ z.(\mathbf{cotod}\ network.z)\ |\ NetData]) & (6) \\
 & \rightarrow^* & (vpc{:}W_{vpc})(k{:}W_k)(Alice'_k\ |\ Bob'_{k,M})\ |\ (vpc'{:}W_{vpc'})(k'{:}W_{k'})(Joan'_{k'}\ |\ Jack'_{k',\ M'})\ | & \\
 & & network[\mathbf{rec}\ z.(\mathbf{cotod}\ network.z)] & (7)
\end{array}$$

This model works when it is well typed. Assume $\Gamma \vdash c{:}S_c$ and $\Gamma \vdash \mathrm{M}{:}\mathrm{Proc}[X^{Z,\ \Xi,\ Y},\ \mathcal{R}_M]$, $\Gamma \vdash M'{:}\mathrm{Proc}[X_1{}^{Z_1,\ \Xi_1,\ Y_1},\ \mathcal{R}_{M'}]$ with $\Xi \cap \Xi' = \varnothing$, then we will deduce that process *AllCom* is typed as

$$\mathrm{Proc}[\text{♀}^{\omega,\ \Xi_{AllCom},\ Y_{AllCom}},\ \mathcal{R}_M \cup \mathcal{R}_{M'} \cup \{S_c\}] \quad \text{where}$$

$$\Xi_{AllCom} = \Xi \cup \Xi' \quad \text{and} \quad Y_{AllCom} = \{\Delta network\} \cup Y \cup Y'$$

under Γ with other results listed below:

$$\begin{array}{llll}
W_k = \mathrm{Amb}[\text{♂}^{\omega,\ \varnothing,\ \varnothing},\ \varnothing] & W_{k'} = \mathrm{Amb}[\text{♂}^{\omega,\ \varnothing,\ \varnothing},\ \varnothing] & W_{n_1} = \mathrm{Amb}[\text{♀}^{0,\ \Xi,\ Y},\ \mathcal{R}_M] & W_{m_1} = \mathrm{Amb}[\text{♀}^{\omega,\ \varnothing,\ \varnothing},\ \varnothing] \\
W_{n_2} = \mathrm{Amb}[\text{♀}^{0,\ \Xi',\ Y'},\ \mathcal{R}_{M'}] & W_{m_2} = \mathrm{Amb}[\text{♀}^{\omega,\ \varnothing,\ \varnothing},\ \varnothing] & W_{in_e} = \mathrm{Amb}[\text{♀}^{1,\ \varnothing,\ \varnothing},\ \varnothing] & W_{out_e} = \mathrm{Amb}[\text{♀}^{0,\ \varnothing,\ \varnothing},\ \{S_c\}] \\
W_{in_d} = \mathrm{Amb}[\text{♀}^{1,\ \varnothing,\ \varnothing},\ \varnothing] & W_{out_d} = \mathrm{Amb}[\text{♀}^{1,\ \varnothing,\ \varnothing},\ \varnothing] & W_{vpc} = W_{vpc'} = \mathrm{Amb}[\text{♂}^{\omega,\ \varnothing,\ \varnothing},\ \{S_c\}] & W_r = \mathrm{Amb}[\text{♀}^{0,\ \varnothing,\ \varnothing},\ \varnothing]
\end{array}$$

$$\Gamma \vdash env_1{:}\mathrm{Amb}[\text{♂}^{Z\,|\,1,\ \Xi,\ Y},\ \mathcal{R}_M] \qquad \Gamma \vdash env_2{:}\mathrm{Amb}[\text{♂}^{Z'\,|\,1,\ \Xi',\ Y'},\ \mathcal{R}_{M'}] \qquad \Gamma \vdash network{:}\mathrm{Amb}[\text{♀}^{1,\ \varnothing,\ \varnothing},\ \varnothing]$$

$\Gamma \vdash SecretCom_M$:Proc$[\female^{\omega, \Xi_M, Y_M}, \mathcal{R}_M \cup \{S_c\}]$ where $\Xi_M = \Xi$ and $Y_M = \{\Delta network\} \cup Y$

$\Gamma \vdash SecretCom_{M'}$:Proc$[\female^{\omega, \Xi_{M'}, Y_{M'}}, \mathcal{R}_{M'} \cup \{S_c\}]$ where $\Xi_{M'} = \Xi'$ and $Y_{M'} = \{\Delta network\} \cup Y'$

Since the source location type sets Ξ_M of process $SecretCom_M$ and $\Xi_{M'}$ of process $SecretCom_{M'}$ are disjoint, the process *AllCom* can be well typed by (SRT Par). We can then assure that *AllCom* and all its reduction derivatives have no direct access interferences by Theorems 2 and 3. If we have the third and even the fourth secret communications on the same *network*, we need only care the messages should have their source location type sets disjoint, then *AllCom* can be also well typed with no direct access interferences. If *AllCom* is put in a non-trusted environment, the multiple direct access primitive is forbidden by the type system and the virtual private channel (*vpc*) can never be observed by the environment.

We omit all the deduction steps for the above typing results. For other information provided by the typing results, we'll have some intuitive explanation. Throughout the results, we find out that only ambients n_1, env_1 have their resources attribute typed as $\mathcal{R}_M$ and only ambients n_2, env_2 have their resources attribute typed as $\mathcal{R}_{M'}$. This is because env_i, i=1,2 contains the message (M, M'), and in turn, n_i, i=1,2 contains the corresponding envelops. Therefore from the resource type information we know that certain resources of type $\mathcal{R}_M$ ($\mathcal{R}_{M'}$) may be provided in n_1 (n_2) and env_1 (env_2).

For the mobility and threadness attribute about the typing results, ambient k is a secure mobile place which can hold classified data that can be sent through a network. *reg* is an internal immobile place where encrypting and decrypting operations occur, and it contains an recursive process with no thread. in_e, in_d and out_e, out_d are something like fixed buffers to hold incoming and outgoing data. n_1 and n_2 are the fixed message-sending boxes while m_1 and m_2 are the fixed boxes for receiving. env_1, env_2 are something like an envelope to hold the message data and the signature **cotof** k. *network* is a fixed physical place where the data transmission takes place. The processes Enc_k and Dec_k are modeled as immobile services to provide unlimited encryption and decryption operations, thus they have the typing result of multi-threaded. $Alice_{k,\ M}$ ($Joan_{k',\ M'}$) is a sender process while Bob_k ($Jack_{k'}$) is a receiver processes. Both of them stand immobile and contain only one thread to perform their operations.

5 Conclusions and Related Work

Since the calculus of Mobile Resources (MR)[2] are designed for better modeling the mobile resources which have no initiative to move, we base our study on MR instead of Mobile Ambients (MA)[1]. We then find the direct access interferences caused by the direct access mechanism in MR. By comparing with the grave interference in MA through several examples, we conclude that the direct access interference do more damage and affect much more kinds of processes. Therefore in this paper, a variant of MR, the calculus of Safe Mobile Resources (SR) is proposed to control the direct access interference. Any process in SR has to be checked by a newly devised type system of SR so that only well typed process can be regarded to have good behavior; otherwise they will be regarded as invalid. The soundness of the type system has been proved and a theorem for assuring the absence of the direct access interference for the well typed process is provided in this paper. At the end of the paper, the example Digital Signature Card is examined to illustrate how robust SR is by comparing with MR. Other examples shows how to use the new calculus to model the usual applications.

For the study of interferences of other calculus related to mobile ambients, besides the calculus of Mobile Safe Ambients (SA)[3], Ref.[6] proposes a variant of Boxed Ambients (BA)[7] to control the grave communication interferences in BA; In paper [8], the security breaches introduced by coactions in SA are avoided by modifying the parameters of the corresponding capabilities. In the current calculus SR, we ignore the interference between resources consumption and resources movement which is left for the further work.

For some features of the type system in SR, the interference controlling mechanism between SR and SA is different. The SA calculus uses the single threaded process type with thread right, that is, the every process typed single threaded process type can only hold one thread right. This is controlled by the SA-Par rule series. In the SR, processes can be quiet, single-threaded and multi-threaded. We achieve the goal by preventing two or more mobile capabilities from parallelism. Due to our research, it seems that every untyped ambient-based calculus (such as MA, BA, and ROAM[9] etc.) have grave interference problems which are caused by inborn drawbacks in their mobile semantics. The best way to remove all these problems is to enforce a dedicated type system. We also allow subtyping on every process types which builds a certain relationship between processes. According to Ref.[10], an element of a type can be considered also as an element of a super type. In SR, this is applied in the rule (SRT Sub) and Theorem 2. In Ref.[11], there is another subtyping relation for the mobile ambients which is devised to return the minimal type relative to the mobile system. As for other studies of type system upon the ambients, early papers are Ref.[12] where the exchange types are introduced in MA, Ref.[13] where mobility types are proposed to indicate mobility of ambients and processes, and Ref.[14] where SA with its type system is introduced to control the grave interferences. Recently, ROAM[9] is proposed to type evolving processes on the pure mobile ambients. Moreover, in Ref.[15], evolving communication is typed upon the full SA. In Ref.[16], mobility types on a reduced ambient calculus without open capability are studied.

At present, the SR calculus does not support name passing. The behavioral equivalence remains unchanged from MR. We expect to devise applicable process equivalence on the typed SR by means of typing the contexts. The interference between movement primitive and resources consumption will be further studied. For other aspects, we feel that the type system of SR is sufficient but a little far from necessary to control the direct access interference. The present one also forbids some forms of the plain interferences. A more accurate type system will be studied in the future.

References:

[1] Cardelli L, Gordon AD. Mobile ambients. In: Nivat M, ed. Foundations of Software Science and Computation Structure. Heidelberg: Springer Verlag, LNCS 1378, 1998. 140−155.

[2] Godskesen JC, Hildebrandt T, Sassone V. A calculus of mobile resources. In: Brim L, ed. Concurrency Theory. Heidelberg: Springer Verlag, 2002. 272−287.

[3] Levi F, Sangiorgi D. Controlling interference in ambients. In: Reps T, ed. Symp. on Principles of Programming Languages. New York: ACM Press, 2000. 352−364.

[4] Milner R. Communication and Concurrency. New York: Prentice Hall, 1989.

[5] Milner R, Parrow J, Walker D. A calculus of mobile process, (Part I and II). Information and Computation, 1992,100:1−77.

[6] Bugliesi M, Crafa S, Merro M, Sassone V. Communication interferences in mobile boxed ambients. In: Agrawal M, ed. Foundations of Software Technology and Theoretical Computer Science. Heidelberg: Springer Verlag, LNCS 2556, 2002. 71−84.

[7] Bugliesi M, Castagna G, Crafa S. Boxed ambients. In: Kobayashi N, ed. Theoretical Aspects of Computer Software. Heidelberg: Springer Verlag, LNCS 2215, 2001. 38−63.

[8] Guan XD, Yang YL, You JY. Making ambients more robust. In: Kurki-Suonio N, ed. Int'l Conf. on Software: Theory and Practice. Beijing: PHEI Press, 2000. 377−384.

[9] Guan XD, Yang YL, You JY. Typing evolving ambients. Information Processing Letters, 2001,80(5):265−270.

[10] Cardelli L. Type system. In: Tucker AB, ed. The Computer Science and Engineering Handbook. Boca Raton: CRC Press, 1997. 2208−2236.

[11] Zimmer P. Subtyping and typing algorithms for mobile ambients. In: Tiuryn J, ed. Foundations of Software Science and Computation Structure. Heidelberg: Springer Verlag, LNCS 1784, 2000. 375−390.

[12] Cardelli L, Gordon AD. Types for mobile ambients. In: Aiken A, ed. Symp. on Principles of Programming Languages. New York: ACM Press, 1999. 79−92.

[13] Cardelli L, Ghelli G, Gordon AD. Mobility types for mobile ambients. In: Wiedermann J, ed. Automata, Languages and Programming. Heidelberg: Springer Verlag, LNCS 1644, 1999. 230−239.

[14] Fu C, You JY. Application modeling based on typed resources. In: Li ML, ed. Grid and Cooperative Computing. Heidelberg: Springer Verlag, LNCS 3033, 2004. 628−635.

[15] Levi F. Types for evolving communication in safe ambients. In: Zuck LD, ed. Verification, Model Checking, and Abstract Interpretation. Heidelberg: Springer Verlag, LNCS 2575, 2003. 102−115.

[16] Coppo EGM, Dezani-Ciancaglini M, Salvo I. M3: Mobility types for mobile processes in mobile ambients. In: Herland J, ed. Computing: The Australasian Theory Symp. Elsevier, ENTCS 78, 2003. 1−34.