Journal of Zhejiang University SCIENCE
ISSN 1009-3095
http://www.zju.edu.cn/jzus
E-mail: jzus@zju.edu.cn

Using location types to control interferences in mobile resources*

FU Cheng (傅 城), YOU Jin-yuan (尤晋元)

(*Department of Computer Science and Engineering, Shanghai Jiaotong University, Shanghai 200030, China*)

E-mail: fucheng@cs.sjtu.edu.cn; you-jy@cs.sjtu.edu.cn

Received Jan. 20, 2004; revision accepted Mar. 30, 2004

Abstract: This paper presents a type system, called Location System (L-S), to control the interferences in the ambient-like calculi. The L-S allows well-behaved (non-interfering) processes to run in parallel if they do not access shared location during their execution life cycle. This approach is designed for a variant of Mobile Ambient (MA), called Safe Mobile Resources (SR), but it can be also used in other ambient-like calculi which are also discussed in this paper.

Key words: Concurrency, Mobile Ambient, Type System
doi:10.1631/jzus.2005.A0414 **Document code:** A **CLC number:** TP393

MOTIVATION

The calculus of Mobile Ambients (MA) (Cardelli and Gordon, 1998) was proposed to model the mobile computation based upon the notion of ambient, a bounded location where computation takes place. It can also perform movements and carry multiple processes and nested ambients within it: Primitive "in" and "out" allows the ambient to cross the computation boundary, and primitive "open" allows the boundary to dissolve and then to unleash the internal processes. Later on, some variants (Guan *et al.*, 2000; Levi and Sangiorgi, 2000; Merro and Hennessy, 2002) of ambients were developed to semantically enhance the security mechanism, some (Cardelli *et al.*, 1999; 2000; Levi and Sangiorgi, 2000; Guan *et al.*, 2001) to type various ambients, others (Bugliesi *et al.*, 2001; Godskesen *et al.*, 2002) to enrich the use of ambients.

For security mechanism, there are two kinds of methods to avoid illegitimate mobile interferences: One is to use a proper type system to prevent the bad (interfering) process to run further, SA (Levi and Sangiorgi, 2000) uses this approach; the other way is to modify the primitives, such as BA (Bugliesi *et al.*, 2001). The latter approach is calculus-specific and it cannot be easily applied to other calculi. The former one, with a few modifications, can be widely transplanted. But current implementation (Levi and Sangiorgi, 2000) for ambients has a drawback: The type system of SA managed to control the interferences by using the Single-Threaded (ST) type for ambient and process, which means each action in the process will wait for its thread right to run and at most one action can be fired at one time. It is strange that non-interfering (well-typed) processes in SA cannot execute in parallel, because only single threaded process is allowed. Moreover, the parallel composition operator loses its intrinsic signification of "parallel composition".

Our aim is to rectify the drawbacks incurred by ST type system of SA. So in this work, the Location System (L-S) was designed to take the responsibility of controlling the interferences. By using L-S, well-behaved processes can take advantage of parallel execution for better performance. The current L-S is based on Safe Mobile Resources (SR) (Fu and You, 2003) which is also an ambient-like calculus and an immediate variant of Mobile Resources (MR) (Godskesen *et al.*, 2002).

*Project supported by the National Natural Science Foundation of China (No. 60173033) and the Shanghai Science and Technology Development Foundation (No. 03DZ15027), China

The rest of this paper is organized as follows. Section 2 reviews the syntax and reduction semantics of SR. Section 3 sums up all forms of interferences in SR. Section 4 presents the pure L-S which only records the location access information. Section 5 discusses the feasibility for L-S to be applied to some other kinds of ambient-like calculi and gives the conclusion.

SR REVIEW

The SR (Fu and You, 2003) calculus is a variant of MR (Godskesen *et al*., 2002), with strict reduction agreement by all participants.

Let *N* be a countable set of names ranged over by *a*, *b*, ..., *m*, *n*. 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 *V*. The set of all processes is denoted by *P* (ranged over by *p*, *q*, ...) and the set of capabilities Λ (ranged over by *λ*).

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*(*λ*) and *fn*(*λ*) 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*. Context is defined as standard.

The SR BNF-like grammar is shown below:

$$\lambda ::= \delta_1 \triangleright \delta_2 \,|\, \overline{\triangleright}\, n \,|\, \underline{\triangleright}\, n \,|\, a \,|\, \gamma\overline{a} \,|\, \natural\gamma m \,|\, \overline{\natural} m$$
$$p ::= 0 \,|\, p \,|\, p \,|\, \lambda.p \,|\, (n:T)p \,|\, x \,|\, \mathrm{rec}\ x.p \,|\, n[p]$$

There are seven different types of actions in SR: letting a process move from a nested ambient path to another nested ambient path ($\delta_1 \triangleright \delta_2$), allowing taken from or given to an ambient ($\overline{\triangleright}\, n \,/\, \underline{\triangleright}\, n$), a resource (*a*), consuming a resource in a nested ambient path ($\gamma\overline{a}$), deleting an ambient ($\natural n$), or allowing being deleted ($\overline{\natural} n$). In SR, resource *a* is a special capability which does not cause any movement but a synchronization. Especially, it represents a consumable resource. As for precedence, *λ.p*|*q* stands for (*λ.p*)|*q*.

For processes, nil process (0), parallel composition (*p*|*p*), restriction with type (*n*:*T*) and capability prefixing (*λ.p*) are standard. *n*[*p*] represents an ambient *n* with a process *p* inside it. *x* represents a recursive process variable. rec *x* is the only binder for the recursive variable, and we use rec *x.p* to describe infinite behaviors for process.

The reduction rules of SR are shown below:

R-Mov:
$$\gamma_1 n \triangleright \gamma_2 m.p \,|\, C_{\gamma 1n}(n'[\overline{\triangleright}\, n.q_1 \,|\, q_2]) \,|\, D_{\gamma 2m}(\overline{\triangleright}\, m.r)$$
$$\to p \,|\, C_{\gamma 1n}(0) \,|\, D_{\gamma 2m}(r \,|\, n'[q_1 \,|\, q_2])$$

R-Act: $\gamma\overline{a}.p \,|\, C_\gamma(a.q) \to p \,|\, C_\gamma(q)$

R-Del: $\natural\gamma n.p \,|\, C_\gamma(n[\overline{\natural} n.p]) \to p \,|\, C_\gamma(0)$

R-Par: $p \to p' \Rightarrow p \,|\, q \to p' \,|\, q$

R-Res: $p \to p' \Rightarrow (n:T)p \to (n:T)p'$

R-Nst: $p \to p' \Rightarrow \tilde{n}[p] \to \tilde{n}[p']$

R-Str: $p \equiv q, q \to r, r \equiv s \Rightarrow p \to s$

R-Rec: $\mathrm{rec}\ x.p \to p\{\mathrm{rec}\ x.p / x\}$

We require that every reduction process has no free recursive variables. Thus the above rules are defined on all processes that have no free recursive variables.

Rule R-Mov allows a process to take an ambient from a nested location and then send it to another nested location. Because not all resources are required to leave an ambient when doing a move reduction, we use asymmetrical placing of the taken and given capabilities. Additionally, in this rule, path context is defined as

$$C_n(\cdot) \triangleq n[(\cdot) \,|\, p], \quad C_{n\gamma}(\cdot) \triangleq n[C_\gamma(\cdot) \,|\, p]$$

R-Act allows a process to consume a resource inside a nested location and R-Del allows a process to delete an ambient inside a nested location. In the last rule R-Rec, *p*{rec *x.p*/*x*} denotes the substitutions for all free occurrences of *x* in *p* with the process rec *x.p*. Other rules are standard. Additionally, for rule R-Str, → is preserved by the structural congruence ≡ which is defined below:

$p \equiv p$; $p|0 \equiv p$; $(n:T)0 \equiv 0$; $(p|q)|r \equiv p|(q|r)$;
$p|q \equiv q|p$; $p \equiv q \Rightarrow n[p] \equiv n[q]$; $p \equiv q \Rightarrow \lambda.p \equiv \lambda.q$;

$$p\equiv q\wedge q\equiv r\Rightarrow p\equiv r;\ p\equiv q\Rightarrow \text{rec}\ x.p\equiv\text{rec}\ x.q;$$
$$n_1\neq n_2\Rightarrow(n_1{:}T_1)(n_2{:}T_2)p\equiv(n_2{:}T_2)(n_1{:}T_1)p;$$
$$p\equiv q\Rightarrow q\equiv p;\ n\notin fn(p)\Rightarrow(n{:}T)p|q\equiv p|(n{:}T)q;$$
$$p\equiv q\Rightarrow(n{:}T)p\equiv(n{:}T)q;\ p\equiv q\Rightarrow p|r\equiv q|r;$$
$$m\neq n\Rightarrow(n{:}T)m[p]\equiv m[(n{:}T)p]$$

The rules for structural congruences and typing rules are defined on all processes, which is different from the definition of reduction rules.

SR INTERFERENCES

Because processes in SR are allowed to access resources in nested ambients, the calculus brings more complex forms of grave interferences than mobile ambients. The following example shows the traditional form of grave interferences in SR:

$$n\triangleright m.p\mid n[r[\overline{\triangleright}\,n]]\mid\natural m.q\mid m[\overline{\natural}m\mid\underline{\triangleright}\,m]$$

In this process, $n\triangleright m$ wants to take ambient r from n to m, and $\natural\,m$ wants to remove the ambient m. Then we have two nondeterministic consequences while the process reduces (where $n[\ ]$ is an abbreviation for $n[0]$): one is $p|n[]|\natural m.q|m[\overline{\natural}m|r[]]$ and the other is $n\triangleright m.p|n[r[\overline{\triangleright}\,n]]|q$. The first result can continue to reduce to $p|n[]|q$. 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.

The new form of grave interferences comes into being while direct access capability is used to access nested ambients:

$$r_1\triangleq nma.p\mid n[m[\overline{\triangleright}\,n\mid a]]$$
$$r_2\triangleq n'n\triangleright m'.q\mid n'[r_1]\mid m'[\underline{\triangleright}\,m']$$

nma 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 $p|n[m[\overline{\triangleright}\,n]]$. But when r_1 is put in an environment as shown in process r_2, $n'nm\triangleright m'$ 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 nma 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.

Let Δ be a set of capabilities $\lambda_1, \lambda_2, \ldots, \lambda_n$ and $r_1[\lambda_1], r_2[\lambda_2], \ldots, r_n[\lambda_n]$, and we write p^{Δ} if

$$p=\lambda_1.p_1|\ldots|\lambda_n.p_n|r_1[\lambda_1.p_1'|p_1'']|\ldots|\ r_n[\lambda_n.p_n'|p_n'']$$

for some $p_1, p_2, \ldots, p_n, p_1', p_2', \ldots, p_n'$, and $r_1, r_2, \ldots, r_n$ where λ_i is not of the form $\overline{\triangleright}\,n$ while λ_i' is of the form $\overline{\triangleright}\,n$ or a.

Definition 1 An SR interference occurs in a process p if $p\equiv C(q)$ where the form of q is one of the following:

$$p_1^{\{\natural\gamma_1\gamma_2n_1\}}\mid C_{\gamma_1}(p_2^{\{\natural\gamma_2n_1\}}\mid C_{\gamma_2}(n_1[p_3^{\{\overline{\natural}n\}}]))$$

$$p_1^{\{\natural\gamma_1\gamma_2n_1\}}\mid C_{\gamma_1}(p_2^{\{\gamma_2n_1\triangleright\gamma_3n_2\}}\mid C_{\gamma_2}(n_1[p_3^{\{\overline{\natural}n_1,m[\overline{\triangleright}n_1]\}}])\mid$$
$$C_{\gamma_3}(n_2[p_4^{\{\underline{\triangleright}n_2\}}]))$$

$$p_1^{\{\gamma_1\gamma_2n_1\triangleright\gamma_3n_2\}}\mid C_{\gamma_1}(p_2^{\{\natural\gamma_2n_1\}}\mid C_{\gamma_2}(n_1[p_3^{\{\overline{\natural}n_1,m[\overline{\triangleright}n_1]\}}]))\mid$$
$$C_{\gamma_3}(n_2[p_4^{\{\underline{\triangleright}n_2\}}])$$

$$p_1^{\{\gamma_1\gamma_2n_1\triangleright\gamma_4n_3\}}\mid C_{\gamma_1}(p_2^{\{\gamma_2n_1\triangleright\gamma_3n_2\}}\mid C_{\gamma_2}(n_1[p_3^{\{m[\overline{\triangleright}n_1]\}}])\mid$$
$$C_{\gamma_3}(n_2[p_4^{\{\underline{\triangleright}n_2\}}]))\mid C_{\gamma_4}(n_3[p_5^{\{\underline{\triangleright}n_3\}}])$$

In the next section, we will introduce the location system, and Theorem 3 is provided to ensure that all processes in SR do not contain the forms of SR interferences defined above.

LOCATION SYSTEM (L-S)

In this section, we give a formal definition of the L-S. The location type is rather simple, and the type value just indicates which locations the typed process will access. We use Δn to represent a solo location type for a specific name n. And the set $\{\Delta n, \Delta m, \ldots\}$ represents the location type for the qualified expression in SR. The grammar for the location types is formally defined as below:

$S ::= \Delta n$ solo location
$T ::= \Phi \mid \{\Delta n\}\cup T$ empty location/location type

The solo location type S denotes a single location, where S ranges over the set Tp which is defined as the universal location set: $Tp=\{\Delta n|n\in N\}$. And $Ta=2^{Tp}$ denotes the set of all location types (ranged over by T). The union operation $\cup$ in the type grammar is standard.

Under L-S type system, capabilities, ambients, and processes all have the same kind of types. Actually, we can easily know what the type value denotes (e.g. if $\Gamma\vdash n{:}T_n$, then T_n is an ambient type). An environment Γ is composed of a sequence of environment names (such as $\Gamma'=n{:}T_n,\ x{:}T_x$). For capability, location type denotes which location it will access; for ambient, location type denotes which locations the running process inside it will access; for process, location type denotes which locations it will access.

There are three kinds of typing judgments in L-S. $\Gamma\vdash\Diamond$ means that Γ is a good type environment; $\Gamma\vdash\lambda{:}T$ means that λ can be typed as T under Γ; Let J range over the set $N\cup P$, $\Gamma\vdash J{:}T$ means ambient and process can have type T under Γ. The typing rules are divided into the following three parts:

Part 1. Typing rules for good environment and names:

$\Phi\vdash\Diamond \qquad \Gamma\vdash\Diamond \wedge n\notin\mathrm{dom}(\Gamma)\Rightarrow\Gamma, n{:}T\vdash\Diamond$

$\Gamma\vdash\Diamond\wedge x\notin\mathrm{dom}(\Gamma)\Rightarrow, x{:}T\vdash\Diamond$

$\Gamma, n{:}T\vdash\Diamond\Rightarrow\Gamma, n{:}T\vdash n{:}T$

Part 2. Typing rules for capabilities:

$\Gamma\vdash n{:}T_n\Rightarrow\Gamma\vdash \overline{\triangleright}\, n{:}\Phi \qquad \Gamma\vdash n{:}T_n\Rightarrow\Gamma\vdash \overline{\natural}\ n{:}\Phi$

$\Gamma\vdash m_1{:}T_1\wedge\Gamma\vdash m_2{:}T_2\Rightarrow\Gamma\vdash m_1\triangleright m_2{:}\{\Delta m_1\}$

$\Gamma\vdash m_1\triangleright m_2{:}T\wedge(\forall n\in fn(\gamma_1)\cup fn(\gamma_2)\Rightarrow\Gamma\vdash n{:}T_n)$

$\Rightarrow \Gamma\vdash\gamma_1 m_1\triangleright\gamma_2 m_2{:}T$

$\Gamma\vdash a{:}T\wedge(\forall n\in fn(\gamma)\Rightarrow\Gamma\vdash n{:}T_n)\Rightarrow \Gamma\vdash\gamma\bar{a}{:}\Phi$

$\Gamma\vdash m{:}T_m\wedge(\forall n\in fn(\gamma)\Rightarrow\Gamma\vdash n{:}T_n)\Rightarrow\Gamma\vdash\natural\,\gamma m{:}\{\Delta m\}$

Part 3. Typing rules for process:

$\Gamma\vdash\Diamond\Rightarrow\Gamma\vdash 0{:}\Phi \qquad \Gamma, x{:}T\vdash\Diamond\Rightarrow\Gamma\vdash x{:}T$

$\Gamma, n{:}T\vdash p{:}\ T'\Rightarrow\Gamma\vdash(n{:}T)p{:}T'-\{\Delta n\}$

$\Gamma, x{:}T\vdash p{:}T\Rightarrow\Gamma\vdash\mathrm{rec}\ x.p{:}T$

$\Gamma\vdash\lambda{:}T_1\wedge\Gamma\vdash p{:}T_2\Rightarrow\Gamma\vdash\lambda.p{:}T_1\cup T_2$

$\Gamma\vdash p{:}T_1\wedge\Gamma\vdash q{:}T_2\wedge T_1\cap T_2\neq\Phi\Rightarrow\Gamma\vdash p|q{:}T_1\cup T_2$

$\Gamma\vdash n{:}T\wedge\Gamma\vdash p{:}T\Rightarrow\Gamma\vdash n[p]{:}T$

$\Gamma\vdash p{:}T'\wedge T\subseteq T'\Rightarrow\Gamma\vdash p{:}T'$

To control the interferences, we achieve the goal by preventing parallel compositions of any processes that will access the same location. In SR, $\gamma_1 m_1\triangleright\gamma_2 m_2$ and $\natural m$ are considered to have the feature of accessing other locations: in the former, parameter m_1 denotes the target location one wants to access, so does m in the latter. Therefore, in the typing rules of Part 2, all other capabilities are typed as empty. For example, $\natural n$ is typed as Δn irrespective of the type of n; $n_1m_1\triangleright n_2m_2$ is typed as Δm_1 irrespective of the types of n_1, n_2, m_1 and m_2; $\overline{\natural}n$ is typed as Φ.

In typing rules of Part 3, the Par rule (6th) guarantees both of the two processes cannot interfere with each other by the side condition $T_1\cap T_2=\Phi$ which means those parallel processes do not access the same location. And after the operation of parallel composition, the whole process is able to access all the locations which the previous two will access. That is why we type the result as $T_1\cup T_2$. The Res rule (2nd), removes the location types for local names when performing the restriction operation. The 3rd and 4th rules are used to type the recursive processes and variables. It is natural to type the process and the recursive variable with the same type. The ambient operation preserves the type of the result process (second to last rule). The last rule is for process subtyping which complies with the subset relation. The subtyping rule allows the current type to be cast to a larger type for a process.

As an example, let $\Gamma\vdash p{:}T_p,\ q{:}T_q$, we can derive $\Gamma\vdash n\triangleright m.p{:}\{\Delta n\}\cup T_p$ and $\Gamma\vdash n\triangleright m.q{:}\{\Delta n\}\cup T_q$. But $n\triangleright m.p|n\triangleright m.q$ is untypable, because both processes will access at least one shared location n, which is rejected by the par rule.

The soundness of the L-S is ensured by the following subject reduction theorems.

Theorem 2 If $\Gamma\vdash p{:}T$ and $p\rightarrow p'$, then $\Gamma\vdash p'{:}T'$ with $T'\subseteq T$.

Proof The proof is shown by induction on the deri-

derivation of $p \rightarrow p'$.

The following theorem assures that under L-S all processes run in a good behavior.

Theorem 3 If $\Gamma' \vdash p{:}T$ then no SR interferences occur in p.

Proof By reduction to absurdity, we can show that the process is not typable for any of the cases in Definition 1.

For example:

$$n \triangleright m.p|n[r[\overline{\triangleright}\, n.q]]|m[\overline{\triangleright}\, m] \rightarrow p|n[\ ]|m[r[q]]$$

By assuming $\Gamma \vdash p{:}T_p$, $q{:}T_q$ and $T_p \cap T_q = \Phi$, the type of the left process can be deduced as $T_p \cup T_q \cup \{\Delta n\}$ under Γ. By the Theorem 2, we know that the right process is also well typed.

We can also reason that the type of the example in Section 3, the 3rd form of SR interferences, cannot be typed by L-S.

The L-S is an independent type system, it can be regarded as orthogonal with any other different type systems and easily combined with the type system of mobility ($\curvearrowright$, $\veebar$) (Cardelli *et al.*, 1999), threadness (0, 1, ω) (Guan *et al.*, 2001) and evolving ($U[T]$) (Guan *et al.*, 2001) types for related ambient-like calculi. For example, a process p may be typed as ($\curvearrowright^1$, $\{\Delta n, \Delta m\}$) $[T]$ which means the current process is mobile and single-threaded, will access location n and m, and will evolve into a process typed as T.

DISSCUSSION

Typing MA (Cardelli and Gordon, 1998) with locations? Since in n, out n and open n these capabilities respectively carry a parameter which represents the target ambient, it appears each of them can be typed as Δn. But it is not the truth that we can obtain the solution for controlling interference in MA. This is because in/out are both subjective mobile primitives, the parameter represents the target location but not the mobile ambient (itself). Therefore from the type values we cannot determine which process may suffer the interferences caused by the shared mobile ambient. For example, suppose an MA process

$$p \triangleq h[n[\text{in } m.q_1|\text{out } h.q_2]|m[q_3]] \qquad (1)$$

This process can cause interferences because ambient n has different moving direction where n may be typed as $\{\Delta m, \Delta h\}$ as it contains in m and out h actions. However, there is no valuable information to indicate the shared mobile ambient n in this type value. Thus it is hard to control the interferences by using location types in MA.

Typing SA (Levi and Sangiorgi, 2000) with locations? As we know, SA adds co-capability for each capability in MA. But this modification only guarantees that agreement should be made between both participants when attempting reduction. These additional co-capabilities provide no help to indicate any information about the mobile locations. For example, now the process (1) becomes in SA:

$$p \triangleq h[n[\text{in } m.q_1|\text{out } h.q_2]|\text{co-out } h|m[\text{co-in } m.q_3]] \qquad (2)$$

We can easily notice that none of those parameters in capability in/out/co-in/co-out contains the information of mobile ambient n. This is the same result we had in the previous section.

Typing ROAM (Guan *et al.*, 2000) with locations. Contrary to those calculi as mentioned above, ROAM provides us a good stage to use location types because the parameter of each (co-) capability represents another participant. Therefore we can obtain enough type information for mobile ambients to control the interference. For example, process (2) now becomes in ROAM:

$$p \triangleq h[n[\text{in } m.q_1|\text{out } h.q_2]|\text{co-out } n|m[\text{co-in } n.q_3]] \qquad (3)$$

As we can see, in ambient h, co-out n indicates only ambient n can move out, in n in ambient m indicates only ambient n can move in. Then we give the following typing rules for capabilities in ROAM (Other rules are similar to SR):

$$\Gamma \vdash n{:}T \Rightarrow \Gamma \vdash \text{in } n{:}\Phi \quad \Gamma \vdash n{:}T \Rightarrow \Gamma \vdash \text{out } n{:}\Phi$$

$$\Gamma \vdash n{:}T \Rightarrow \Gamma \vdash \text{co-in } n{:}\{\Delta n\}$$

$\Gamma\vdash n{:}T \Rightarrow \Gamma\vdash \text{co-out } n{:}\{\Delta n\}$

$\Gamma\vdash n{:}T \Rightarrow \Gamma\vdash \text{open } n{:}\{\Delta n\}$

$\Gamma\vdash n{:}T \Rightarrow \Gamma\vdash \text{co-open } n{:}\Phi$

For process (3), suppose $\Gamma\vdash$co-out $n{:}\{\Delta n\}$ and $\Gamma\vdash q_3{:}T$. Then we have $\Gamma\vdash m[\text{co-in } n.q_3]{:}T\cup\{\Delta n\}$ which cannot be parallel with co-out n because the side condition of the process par rule requires that the two type values are disjoint. Then process (3) cannot be typed by this type system and will be regarded as a program error

Further views to control interferences. The SA approach and ours have one feature in common. Both of them prohibit two processes from being parallel with each other through the Par rule for process if both of the processes will have the rights to access the same ambient. But this kind of approach cannot coexist with Replication rule. This is the reason why SA uses recursive constructs for infinite behaviors for processes, and hence in SR we use the same constructs. Though ROAM uses replication constructs to describe infinite behaviors for processes, it is easy to make few modifications to apply the L-S.

The distinction of type system between SA and SR is threadness. The SA interference-free process must be typed as single-threaded which means the execution process should strictly follows the execution sequence generated by the type system. However, by using L-S, parallel execution is allowed in well-behaved processes. Thus, process execution will speed up dramatically.

As for open capability in mobile ambients, if an ambient is opened, the internal process will be unleashed. This may cause an "interference-free" process to reduce to an interference-causing process such that it will break the typing theorem of soundness. To prevent this, process Amb rule should preserve the type value when performing the ambient enclosing operation. Therefore, the previous "interference-free" process cannot be typed with location types.

Although we do not provide any theorem and proofs for ROAM and MR to claim the soundness of applying location type system to them, in terms of the above analysis, we strongly believe that our solution can serve as a general typing architecture to control interference in ambient-like calculi if (1) the primitives provide enough information for the mobile target and (2) recursive constructs are used to describe infinite process behavior instead of replication.

As discussed above, all of our works are based on pure ambient-like calculus. For future works, L-S will be introduced to the non-pure ambient-like calculus with message passing. We expect the channel can also be typed by L-S.

References

Bugliesi, M., Castagna, G., Crafa, S., 2001. Boxed Ambients. Proc. TACS 2001, Lecture Notes in Computer Science, Springer, **2215**:38-63.

Cardelli, L., Gordon, A.D., 1998. Mobile Ambients. *In*: Nivat, M. (Ed.), Proc. FoSSaCS'98, Lecture Notes in Computer Science, Springer, **1378**:140-155.

Cardelli, L., Ghelli, G., Gordon, A.D., 1999. Mobility Types for Mobile Ambients. Technical Report MSR-TR-99-32, Microsoft Research.

Cardelli, L., Ghelli, G., Gordon, A.D., 2000. Ambient Groups and Mobility Types. IFIP TCS, p.333-347.

Fu, C., You, J.Y., 2003. Application Modeling Based on Typed Resources. Proc. GCC 2003, Lecture Notes in Computer Science, Springer, **3033**:628-635.

Godskesen, J.C., Hildebrandt, T., Sassone, V., 2002. A Calculus of Mobile Resources. Proc. CONCUR'02, Lecture Notes in Computer Science, Springer, **2412**:272-287.

Guan, X.D., Yang, Y.L., You, J.Y., 2000. Making Ambients more Robust. Proc. ICS 2000, PHEI Press, p.377-384.

Guan, X.D., Yang, Y.L., You, J.Y., 2001. Typing evolving ambients. *Information Processing Letters*, **80**(5):265-270.

Levi, F., Sangiorgi, D., 2000. Controlling Interference in Ambients. Short Version Appeared in Proc. 27th POPL, ACM Press.

Merro, M., Hennessy, M., 2002. Bisimulation Congruences in Safe Ambients. Proc. POPL'02, ACM Press, p.71-80.