Resource Access Control with Dynamic Acquisition of Access Rights

Daniele Gorla and Rosario Pugliese

Dipartimento di Sistemi e Informatica
Università di Firenze

MIKADO – Workshop
Firenze, 6th December 2002

Summary

- Overview of KLAIM
- μKLAIM: Main Features and Syntax
- A Capability Based Type System
- Static and Dynamic Typing
- Variants and Future Work

KLAIM : An Overview

Main Features of past works:

- Asynchronous communication via a shared memory
- Distribution and Mobility
- Remote and Local operations
- Flat net architecture with dynamic evolution
- Access control via types
  - Process type = actions the process intends to perform over the net
  - Node type = security policy of the node
  - Well-typedness = types of processes do agree with the security policy of the nodes hosting them

μKLAIM: A core calculus for KLAIM

We removed:	- distinction between logical and physical addresses (and allocation environments)
	- higher order communication
	- types with global information
We added:	dynamic privileges acquisition
We obtained:	- types with only local information
	- efficient type handling
	- simpler semantics and type systems
The price paid:	more run time checks

μKLAIM Syntax

Nets	N	$::=$	$\mathbf{0} \;\big|\; l ::^{\delta} P \;\big|\; N_1 \parallel N_2$
Processes	P	$::=$	$\mathbf{nil} \;\big|\; a.P \;\big|\; P_1 \mid P_2 \;\big|\; A$
Actions	a	$::=$	$\mathbf{read}(T)@\ell \;\big|\; \mathbf{in}(T)@\ell \;\big|\; \mathbf{out}(t)@\ell$
		$\big|$	$\mathbf{eval}(P)@\ell \;\big|\; \mathbf{newloc}(u : \delta)$
Templates	T	$::=$	$F \;\big|\; F, T$
Tem.Fields	F	$::=$	$f \;\big|\; !\,x \;\big|\; !\,u : \pi$
Tuples	t	$::=$	$f \;\big|\; f, t$
Tuple Fields	f	$::=$	$e \;\big|\; \ell : \mu$
Expressions	e	$::=$	$V \;\big|\; x \;\big|\; \ldots$

Types for Resource Access Control (1)

- We control via types the possible operations, i.e. i, r, o, e, n (**capabilities**). Π is formed by the non–empty subsets of capabilities

- A node is $l ::^{\delta} P$, where δ is the *security policy* of the node (i.e. what P can perform once executed in l)

- Formally, $\delta : Loc \rightharpoonup \Pi$

- For example :

  $l ::^{[l_1 \mapsto \{i,o\},\ldots]}$ **in**$(\ldots)@l_1$ is legal

  $l ::^{[l_1 \mapsto \{i,o\},\ldots]}$ **eval**$(\ldots)@l_1$ is not

- Well–typedness $\Rightarrow$ no illegal operations at run-time.

Types for Resource Access Control (2)

Dynamic Acquisition of Privileges:

We want to model a situation like

$$N \quad \triangleq \quad l_1 ::^{[l_2 \mapsto \{i\}]} \ \mathbf{in}(!u : \{o\})@l_2.\mathbf{out}(100)@u \ \ \| \\ l_2 ::^{[\ldots]} \ \mathbf{out}(l)@l_2$$

$$\overset{\mathbf{out}}{\rightarrowtail}\overset{\mathbf{in}}{\rightarrowtail} \quad l_1 ::^{[l_2 \mapsto \{i\}, l \mapsto \{o\}]} \ \mathbf{out}(100)@l \ \ \| \ \ l_2 ::^{[\ldots]} \ \mathbf{nil}$$

i.e. l_2 grants l_1 the capability of performing an **out** at l.

But what if l_2 does not own capability o over l?

Pre-Types (1)

1. In **out**, each locality is annoted with the capabilities passed.

$$
\begin{aligned}
N_1 \;\triangleq\;\; & l_1 ::^{[l_2 \mapsto \{i\}]} \;\mathbf{in}(!u : \{o\})@l_2.\mathbf{out}(100)@u \;\; \| \\
& l_2 ::^{\delta} \mathbf{out}(l : [l_1 \mapsto \{o, e\}, l_3 \mapsto \{i\}])@l_2
\end{aligned}
$$

2. When the **out** is fired, it is verified that the capabilities passed be effectively owned by the node performing it.

$$
\begin{aligned}
N_1 \;\xrightarrow{\mathbf{out}}\;\; & l_1 ::^{[l_2 \mapsto \{i\}]} \;\mathbf{in}(!u : \{o\})@l_2.\mathbf{out}(100)@u \;\; \| \\
& l_2 ::^{\delta} \mathbf{out}(l : [l_1 \mapsto \{o, e\}, l_3 \mapsto \{i\}]) \;\triangleq\; N_1'
\end{aligned}
$$

if $\{o, e, i\} \subseteq \delta(l)$

Pre-Types (2)

3. When the communication takes place, it is verified that the capabilities required in the template are granted by the tuple to the locality performing the **in**.

$$N_1' \stackrel{\mathbf{in}}{\rightarrowtail} \; l_1 ::^{[l_2 \mapsto \{i\}, l \mapsto \{o\}]} \mathbf{out}(100)@l \;\; \| \;\; l_2 ::^{\delta} \mathbf{nil}$$

since $o \in \{o, e\}$

Pre-Types (3)

It is reasonable to:

- pass all the capabilities owned over a given locality
- pass all the capabilities, except someones

The capabilities really passed can be extablished ONLY at run-time; a pre-type syntactically expresses only the intentions of passing.

A pre-type is a partial function

$$\mu : \mathcal{L} \cup \mathcal{U} \rightharpoonup \Pi \cup \overline{\Pi}_{\emptyset}$$

with finite domain, where $\overline{\Pi}_{\emptyset} \triangleq \{ \ \overline{\pi} \ : \ \pi \in \Pi \cup \{\emptyset\} \ \}$.

Pre-Types (4)

Examples:

- **out**$(l' : [l \mapsto \overline{\pi}])$ passes everything except π
- **out**$(l' : [l \mapsto \overline{\emptyset}])$ passes everything

Pre-types are evaluated before firing the **out** in order to

- evaluate the set of capabilities to be passed
- check them against the security policy

Static Type Inference (1)

$$l ::^{[\ldots, l' \mapsto \{i\}]} \ \mathbf{in}(!u : \{o\})@l'.\mathbf{out}(100)@l'$$

What should we do with it? Two possibilities:

1. statically refuse it
2. delay the decision at run-time

On the contrary, we shall always refuse

$$l ::^{[\ldots, l' \mapsto \{i\}]} \ \mathbf{in}(!u : \{o\})@l'.\mathbf{in}(100)@u$$

Static type inference :

- action using a variable as target: check that the action respects the declaration of the variable
- action using a locality as target: if the action is not legal, mark it and delay decision at run-time

Static Type Inference (2)

Definition 1 *A net is* well-typed *if for each node* $l ::^{\delta} P$ *there exists* P' *s.t.* $\delta \vdash_{l} P \triangleright P'$.

Intuitively, it says that P can be properly marked (P') s.t.

- non marked actions in P' respect policy δ
- variables are used coherently to their declarations

Definition 2 *A net is* executable *if for each node* $l ::^{\delta} P$ *it holds* $\delta \vdash_{l} P \triangleright P$.

Intuitively, it says that no further markings are needed in order to safely execute the net.

μKLAIM Semantics (Dynamic Typing)

- Evaluating pre-types
- Enabling communication
- Authorizing migrations

$$\frac{\delta' \vdash_{l'} Q \ \triangleright \ Q'}{l ::^{\delta} \mathbf{eval}(Q)@l'.P \parallel l' ::^{\delta'} P' \succ\!\!\!\longrightarrow l ::^{\delta} P \parallel l' ::^{\delta'} P'|Q'}$$

- Executing marked actions (*in-lined reference monitor*)

$$\frac{l' = loc(a) \qquad cap(a) \in \delta(l') \qquad l ::^{\delta} a.P \parallel l' ::^{\delta'} Q \succ\!\!\!\longrightarrow N}{l ::^{\delta} \underline{a}.P \ \parallel l' ::^{\delta'} Q \succ\!\!\!\longrightarrow N}$$

Main Results

Theorem 1 (Subject Reduction) *If N is executable and $N \rightarrowtail N'$ then N' is executable.*

Theorem 2 (Type Safety) *If N is executable then $N \uparrow l$ for no $l \in loc(N)$.*

where $N \uparrow l$ is the run time error predicate, whose main case is

$$\frac{cap(a) \notin \delta(loc(a))}{l ::^{\delta} a.P \uparrow l}$$

Corollary 1 *If N is executable and $N \rightarrowtail^* N'$ then $N' \uparrow l$ for no $l \in loc(N')$.*

The same results hold in a *local* version, i.e.

Theorem 3 *If all the nodes in $D \subseteq loc(N)$ are executable and $N \rightarrowtail^* N'$ then $N' \uparrow l$ for no $l \in D$.*

Variations(1)

Acquisition by Nodes and Processes: Mobile processes can acquire rights also for themselves.

- Actions: $a ::= \ldots \mid \mathbf{inp}(T)@\ell \mid \mathbf{readp}(T)@\ell$
- Mobile Processes syntactic category: $\mathcal{AP} ::= \{\!\{\ P\ \}\!\}_\delta$
- New capabilities: i_p, r_p
- The semantics of **inp**/**readp** are similar to **in**/**read** but the acquisition is recorded in the process type.

$$\frac{match_l(\mathcal{T}[\![\ T\ ]\!]_{\delta_1[\delta]}, et) = \langle \delta'', \sigma \rangle}{l ::^{\delta} \{\!\{\ \mathbf{inp}(T)@l'.P\ \}\!\}_{\delta_1} \parallel l' ::^{\delta'} \mathbf{out}(et) \succ\!\!\longrightarrow l ::^{\delta}\ \{\!\{\ P\sigma\ \}\!\}_{\delta_1[\delta'']} \parallel l' ::^{\delta'} \mathbf{nil}}$$

Variations (2)

Consumption of Access Rights: Once used, rights are lost.

- Multisets of capabilities.
- In the reductions, types are decreased accordingly to
  - the action performed
  - the privileges passed
- In migrations, rights are properly split.

$$\frac{\delta_1 = \delta_1'[\delta_1''] \qquad \delta'[\delta_1''] \vdash_{l'} Q \rhd \underline{Q} \qquad \delta[\delta_1'] \vdash_{l} \overline{P} \rhd \underline{P}}{l ::^{\delta} \{\!\{\, \mathbf{eval}(Q)@l'.P \,\}\!\}_{\delta_1} \parallel l' ::^{\delta'} \mathcal{AP} \succ\!\!\longrightarrow l ::^{\delta} \{\!\{\, \underline{P} \,\}\!\}_{\delta_1'} \parallel l' ::^{\delta'} \mathcal{AP} | \{\!\{\, \underline{Q} \,\}\!\}_{\delta_1''}}$$

Variations (3)

Time Expiration of Access Rights: Capabilities can be assigned a duration: a privilege is available until the timeout associated to it is not yet expired.

- Capabilities are indexed with a number representing a timeout.
  E.g. $[l \mapsto \{i_{10}, o_5\}]$
- Static checking of actions is possible only with *persistent* capabilities (i.e. capabilities with an infinite timeout)
- All the other operations have to be marked (it is impossible to exactly know when they will be fired)

Variations (3 cont.)

Representation of time passing:

- Transitions labeled with time: "$\overset{\tau}{\rightarrowtail}$" records the passing of τ time units
- Time passes uniformly for all the processes running at a certain node

$$l ::^{\delta} P \overset{\tau}{\rightarrowtail} l ::^{\delta_{-\tau}} P \qquad \frac{N \overset{\tau}{\rightarrowtail} N' \quad sites(N) \cap sites(N'') = \emptyset}{N \parallel N'' \overset{\tau}{\rightarrowtail} N' \parallel N''}$$

where $\delta_{-\tau}$

1. decreases all the time annotations of capabilities in δ of τ time units
2. deletes the capabilities with expired duration

E.g. $l ::^{[l' \mapsto \{i_{10}, o_5\}]} P \overset{5}{\rightarrowtail} l ::^{[l' \mapsto \{i_5\}]}$

Conclusions

Other works done:

- Finer grained access policies (i.e., different rights over different tuple patterns)
- Authorization (e.g. assigning processes different rights according to the origin of a migration)

Work in progress:

- Confinement (e.g. regions constraint processes mobility and tuple exchange)
- Localities organized in *groups* and *roles*
- Behavioural equivalences to relate nets with the same behaviour

Still to be done: A calculus with distribution and cryptography