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

• $\mu$KLAIM: 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
\textbf{\(\mu\text{KLAIM: A core calculus for KLAIM}\)}

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
### \( \mu \text{Klaim} \) Syntax

#### Nets

\[
N ::= 0 \mid l :: \delta P \mid N_1 \parallel N_2
\]

#### Processes

\[
P ::= \text{nil} \mid a.P \mid P_1 \mid P_2 \mid A
\]

#### Actions

\[
a ::= \text{read}(T)@l \mid \text{in}(T)@l \mid \text{out}(t)@l
\]
\[
\mid \text{eval}(P)@l \mid \text{newloc}(u : \delta)
\]

#### Templates

\[
T ::= F \mid F,T
\]

#### Template Fields

\[
F ::= f \mid !x \mid !u : \pi
\]

#### Tuples

\[
t ::= f \mid f,t
\]

#### Tuple Fields

\[
f ::= e \mid \ell : \mu
\]

#### Expressions

\[
e ::= V \mid x \mid \ldots
\]
Types for Resource Access Control (1)

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

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

- Formally, \( \delta : \text{Loc} \rightarrow \Pi \)

- For example: 
  
  \[ \begin{align*} 
  l &:: [l_1 \mapsto \{i, o\},...] \quad \textbf{in}(\ldots)@l_1 \quad \text{is legal} \hfill \\
  l &:: [l_1 \mapsto \{i, o\},...] \quad \textbf{eval}(\ldots)@l_1 \quad \text{is not} 
  \end{align*} \]

- Well-typedness \( \Rightarrow \) no illegal operations at run-time.
Dynamic Acquisition of Privileges:

We want to model a situation like

\[ N \triangleq l_1 :: [l_2 \mapsto \{i\}] \text{ in}(!u : \{o\})@l_2.\text{out}(100)@u \parallel l_2 :: [\ldots] \text{ out}(l)@l_2 \]

\[ \xrightarrow{\text{out}} \xrightarrow{\text{in}} l_1 :: [l_2 \mapsto \{i\}, l \mapsto \{o\}] \text{ out}(100)@l \parallel l_2 :: [\ldots] \text{ nil} \]

i.e. \( l_2 \) grants \( l_1 \) the capability of performing an \texttt{out} at \( l \).

But what if \( l_2 \) does not own capability \( o \) over \( l \)?
1. In \textbf{out}, each locality is annotated with the capabilities passed.

\[ N_1 \triangleq l_1 :: [l_2 \mapsto \{i\}] \quad \text{in}(!u : \{o\})@l_2.\textbf{out}(100)@u \parallel \]
\[ l_2 :: \delta \textbf{out}(l : [l_1 \mapsto \{o, e\}, l_3 \mapsto \{i\}])@l_2 \]

2. When the \textbf{out} is fired, it is verified that the capabilities passed be effectively owned by the node performing it.

\[ N_1 \xrightarrow{\text{out}} l_1 :: [l_2 \mapsto \{i\}] \quad \text{in}(!u : \{o\})@l_2.\textbf{out}(100)@u \parallel \]
\[ l_2 :: \delta \textbf{out}(l : [l_1 \mapsto \{o, e\}, l_3 \mapsto \{i\}]) \triangleq N'_1 \]
\[ \text{if } \{o, e, i\} \subseteq \delta(l) \]
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**.

\[
N'_1 \xrightarrow{\text{in}} l_1 :: [l_2 \mapsto \{i\}, l \mapsto \{o\}] \quad \text{out}(100)@l \quad \parallel \quad l_2 :: \delta \text{ nil}
\]

since \( o \in \{o, e\} \)
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 established ONLY at run-time; a pre-type syntactically expresses only the intentions of passing.

A pre-type is a partial function

\[ \mu : \mathcal{L} \cup \mathcal{U} \to \Pi \cup \overline{\Pi}_0 \]

with finite domain, where \( \overline{\Pi}_0 \triangleq \{ \overline{\pi} : \pi \in \Pi \cup \{\emptyset\} \} \).
Pre-Types (4)

Examples:

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

\[ l ::[\ldots, l' \rightarrow \{i\}] \quad \text{in}(!u : \{o\})@l' \text{. out}(100)@l' \]

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

\[ l ::[\ldots, l' \rightarrow \{i\}] \quad \text{in}(!u : \{o\})@l' \text{. in}(100)@u \]

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. \( \delta |_{l} P \triangleright P' \).

Intuitively, it says that \( P \) can be properly marked (\( P' \)) s.t.
- non marked actions in \( P' \) respect policy \( \delta \)
- variables are used coherently to their declarations

**Definition 2** A net is executable if for each node \( l :: \delta \) \( P \) it holds \( \delta |_{l} P \triangleright P \).

Intuitively, it says that no further markings are needed in order to safely execute the net.
\[ \muKLAIM \text{ Semantics (Dynamic Typing)} \]

- Evaluating pre-types
- Enabling communication
- Authorizing migrations

\[
\delta' |_{\mu}^\nu Q \triangleright Q'
\]

\[
\frac{l \::^\delta \text{eval}(Q)@l'.P \parallel l' ::^\delta' P' \rightarrow l ::^\delta P \parallel l' ::^\delta' P'|Q'}{l ::^\delta a.P \parallel l' ::^\delta' Q \rightarrow N}
\]

- Executing marked actions (\textit{in-lined reference monitor})

\[
l' = \text{loc}(a) \quad \text{cap}(a) \in \delta(l') \quad l ::^\delta a.P \parallel l' ::^\delta' Q \rightarrow N
\]

\[
l ::^\delta a.P \parallel l' ::^\delta' Q \rightarrow N
\]
Main Results

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

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

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

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

Corollary 1  If $N$ is executable and $N \succ \rightarrow^* N'$ then $N' \uparrow l$ for no $l \in \text{loc}(N')$.

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

Theorem 3  If all the nodes in $D \subseteq \text{loc}(N)$ are executable and $N \succ \rightarrow^* N'$ then $N' \uparrow l$ for no $l \in D.$
Acquisition by Nodes and Processes: Mobile processes can acquire rights also for themselves.

- Actions: \( a ::= \ldots | \text{inp}(T)@\ell | \text{readp}(T)@\ell \)

- Mobile Processes syntactic category: \( \mathcal{AP} ::= \{ \{ P \} \}_\delta \)

- New capabilities: \( i_p, r_p \)

- The semantics of \( \text{inp}/\text{readp} \) are similar to \( \text{in}/\text{read} \) but the acquisition is recorded in the process type.

\[
\text{match}_l(T[T]_{\delta_1[\delta]}, et) = \langle \delta'', \sigma \rangle
\]

\[
l :: \delta \{ \{ \text{inp}(T)@l'.P \} \}_{\delta_1} || l' :: \delta' \text{ out}(et) \rightarrow l :: \delta \{ \{ P\sigma \} \}_{\delta_1[\delta'']} || l' :: \delta' \text{ nil}
\]
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.

\[
\begin{align*}
\delta_1 &= \delta'_1[\delta''_1] \\
\delta'[\delta''_1] &\mid \Gamma - Q \triangleright Q \\
\delta[\delta'_1] &\mid \Gamma - P \triangleright P \\
\langle l :: \delta \{ \text{eval}(Q) \}@l'.P \rangle_{\delta_1} &\parallel \langle l' :: \delta' \text{AP} \Rightarrow l :: \delta \{ P \} \rangle_{\delta'_1} \parallel \langle l' :: \delta' \text{AP} \{ \{ Q \} \rangle_{\delta''_1} \end{align*}
\]
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: “$\geq \tau$” records the passing of $\tau$ time units
- Time passes uniformly for all the processes running at a certain node

\[
l :: \delta P \geq \tau l :: \delta - \tau P \quad \frac{N \geq \tau N' \quad sites(N) \cap sites(N'') = \emptyset}{N \parallel N'' \geq \tau N' \parallel N''}
\]

where $\delta_{-\tau}$

1. decreases all the time annotations of capabilities in $\delta$ of $\tau$ time units
2. deletes the capabilities with expired duration

E.g. $l :: [l'\rightarrow\{i_{10},o_{5}\}] P \geq 5 l :: [l'\rightarrow\{i_{5}\}]$
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