Type-Based Discretionary Access Control

Silvia Crafa

joint work with

M. Bugliesi and **D.Colazzo**

Universita' Ca' Foscari of Venice

MYTHS project

CONCUR, London, September 3, 2004

Access Control in π-calculus

spool ‹ job1 ›

spool ‹ job2 ›

! spool (x) . print ‹ x ›

Access Control in π-calculus

spool < job1 >

spool < job2 >

! spool (x) . print < x >

spool (y). 0

Capability Types for Access Control

PRINTER

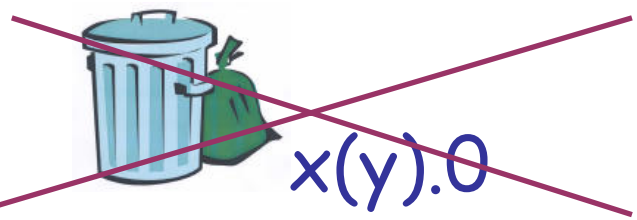
pub

CLIENTS

(ν spool : T ^{rw}) pub ‹ spool ›

! spool (y) . print ‹ y ›

pub (x :T ^{w}) . x ‹job›



The spooler is a
write-only channel

Looking for stronger guarantees

- Client jobs should not be logged or leaked

log <job>

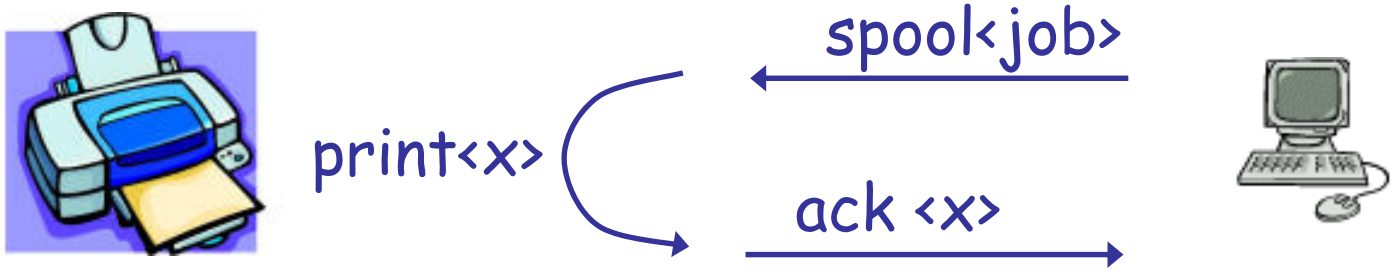
spool<job>

log(y) . SPY

! spool (x) .
log<x>. print<x>

Looking for stronger guarantees

- Client want reliable acknowledgements



! spool (x) . print<x> . ack<x> OK

! spool (x) . ack<x> CHEATING SPOOLER

Need more informative types

- Capability types [PS96] do not help to provide intended guarantees.
- Control the flow of names among system components.
- One need the ability to **express/enforce discretionary policies of access control** governing
  - which authorities may receive values of a given type
  - what (type) capabilities should be passed along with the values

Controlling delivery of names

Associate names with delivery policies:

Capability-based system + new info to describe how values may be exchanged among system components

Generalize **Group Types** [CGG00] adding DAC's ingredients:

$$G\ [\ T\ \|\ \Delta\ ]$$

authority in control of the value

structural info about values

delivery policy, to control how values are passed around (to which authorities, with which capabilities)

Our goal

Use TYPES

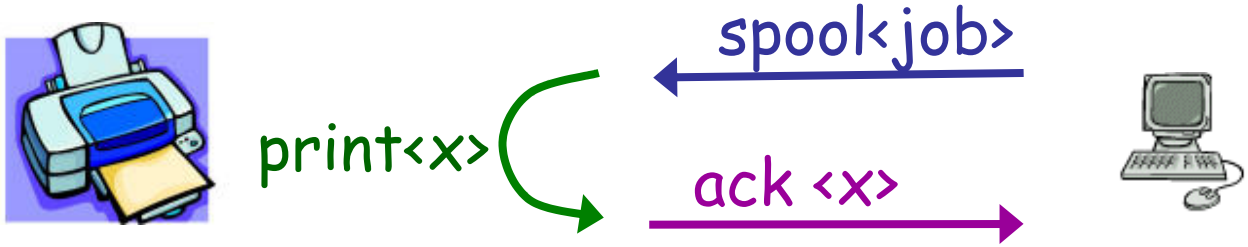
1. **to express** powerful and flexible discretionary access control policies over names (resources)

   (by means of *subtyping*, *capabilities*,

   a special default group, *recursive types*)

2. **to prove** that in well-typed processes
   - all names flow according to the delivery policies specified by their types,
   - and they are received at the intended sites with the intended capabilities.

Type-based control of the spooler

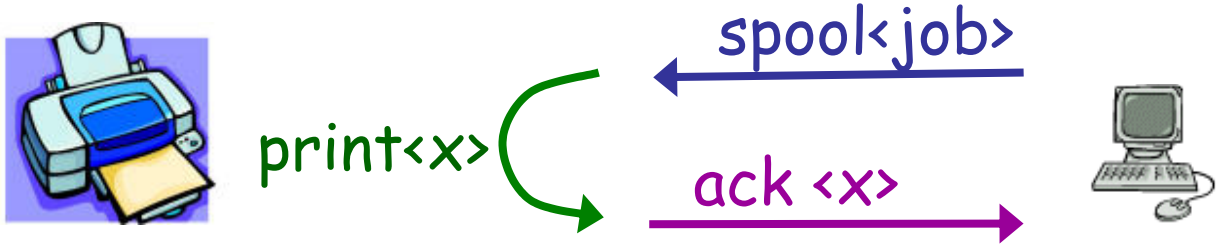


job : Job [fd || Spooler ↷ Printer ↷ Client]

job is a
file descriptor

to be ***first*** delivered to the spooler
then passed on to the printer, and
only then sent back to clients as an ack

Type-based control of the spooler



job : Job [fd || Spooler ↷ Printer ↷ Client]

spool : Spooler [(Job [fd || Printer ↷ Client])^{rw} || Δ]

spool is a (r/w) channel controlled by the Spooler

spool itself should be delivered as dictated by Δ

It carries a file desc which may be passed on to a client only after having been transmitted to the printer

Type based control of the spooler

The client spawns a job with the following type:

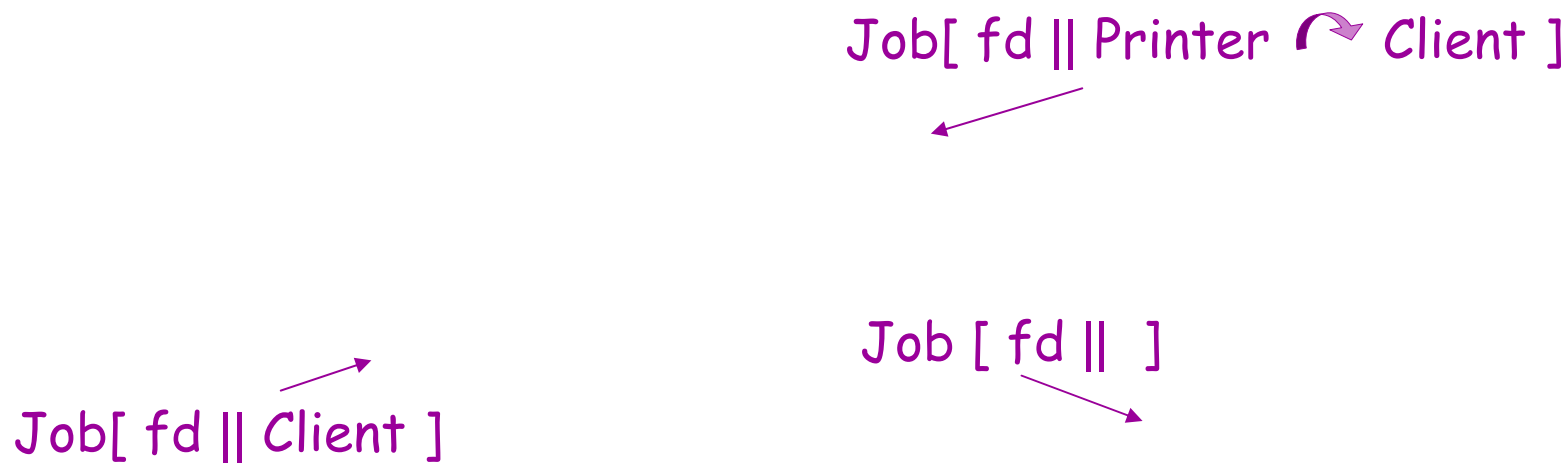
job : Job [fd || Spooler ↷ Printer ↷ Client]

print<job >

spool <job>

ack <job>

Type based control of the spooler

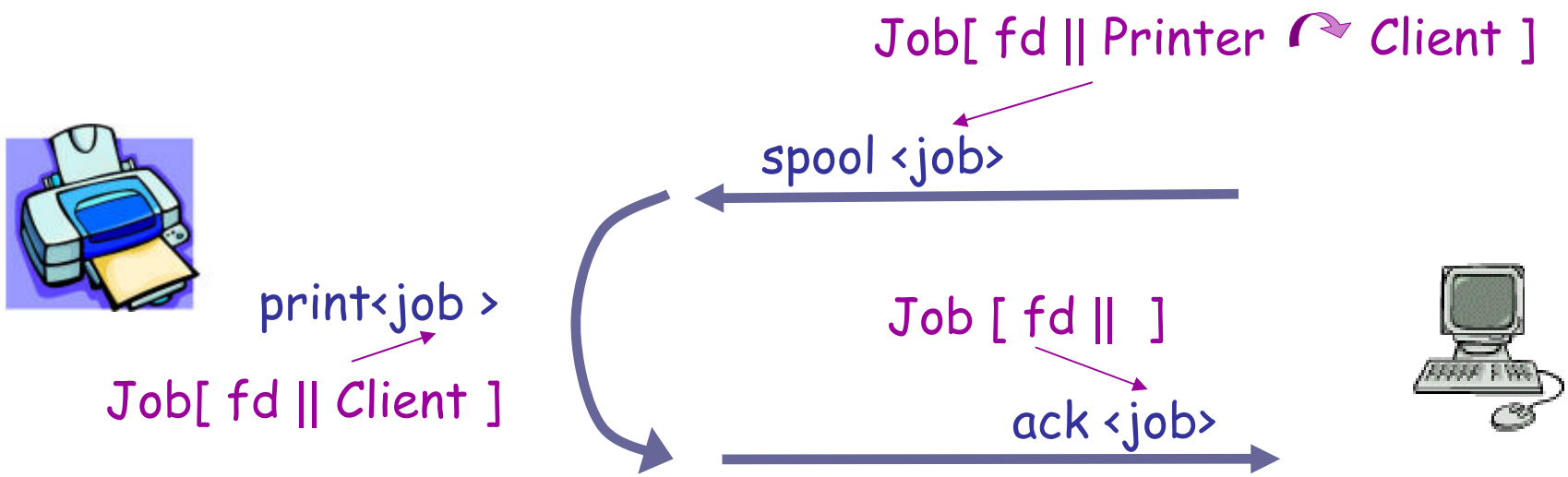


job must be given different types as it is delivered !!

Type based control of the spooler

The client spawns a job with the following type:

job : Job [fd || Spooler ↷ Printer ↷ Client]



job must be given different types as it is delivered !!

A typed π-calculus with groups

Syntax as in [CGG00]:

$$P ::= 0 \mid a(x_1{:}\tau_1,\ldots,x_k{:}\tau_k).P \mid a\langle b_1,\ldots,b_k\rangle.P$$

$$(\nu\, n{:}\tau)\, P \mid (\nu\, G)\, P \mid P|P \mid !P$$

Types generalize those in [CGG00]:

Structural types $\quad T ::= B \mid (\tau_1,\ldots,\tau_k)^{\chi}$

Resource types $\quad \tau ::= G\,[T \,||\, \Delta] \mid X \mid \mu X.G[T \,||\, \Delta]$

Delivery policies $\quad \Delta ::= [\, G_i \rightarrow \tau_i \,]_{i \in I} \qquad G_i{=}G_j \Rightarrow \tau_i{=}\tau_j$

Sample types

- Channels of group G that may be received and retransmitted at any other group as write-only channels:

$$\mu X.\ G[\ (\mathrm{nat})^{rw} \ ||\ \ G \rightarrow X\ ;\ \mathrm{Default} \rightarrow \mu Y.G[\ (\mathrm{nat})^{w} || \ \mathrm{Default} \rightarrow Y]\ ]$$

- Alice and Bob establish a private exchange:

  Alice sends a fresh name c_{AB} to a trusted Server and delegates it to forward it to Bob.

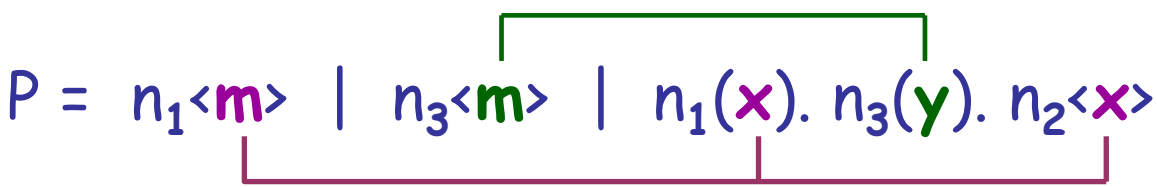
  The Server should only act as a forwarder and not interfere with the exchanges between Alice and Bob.

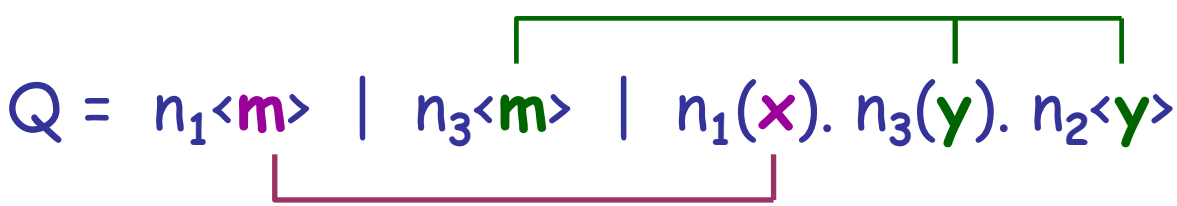
$$c_{AB} : \mathrm{Alice}[(\mathrm{data})^{rw} \ ||\ \mathrm{Server} \rightarrow \mathrm{Alice}[\ (\mathrm{data}) \ ||\ \mathrm{Bob} \rightarrow \mathrm{Alice}[(\mathrm{data})^{rw} \ ||\ ]]\ ]$$

Operational Semantics

Different occurrences of the same name may flow along different paths.

$n_1{:}G_1[\ldots]$, $n_2{:}G_2[\ldots]$, $n_3{:}G_3[\ldots]$ and $m : G[\ B \| G_1 \rightarrow G_2 \ ; \ G_3\ ]$





P is safe, Q is unsafe, but $P \rightarrow\!\!\rightarrow n_2\langle m\rangle \leftarrow\!\!\leftarrow Q$

Operational Semantics

Names are tagged to record their flow history: $m_{[npq]}$

$$n_{[\phi]} \langle m_{[\rho]} \rangle.\ A \mid n_{[\eta]}(x{:}\tau).\ B \quad \rightarrow \quad A \mid B\{ x := m_{[\rho\, n]} \}$$

Now the computation exhibits different flows for P and Q:

$$P = n_1\langle m \rangle \mid n_3\langle m \rangle \mid n_1(x).n_3(y).n_2\langle x \rangle \rightarrow\!\rightarrow n_2 \langle m_{[n1]} \rangle$$

$$Q = n_1\langle m \rangle \mid n_3\langle m \rangle \mid n_1(x).n_3(y).n_2\langle x \rangle \rightarrow\!\rightarrow n_2 \langle m_{[n3]} \rangle$$

Theorem:

1. If $A \rightarrow^* B$ then $|A| \rightarrow^* |B|$.
2. If $|A| \rightarrow^* Q$ then $A \rightarrow^* B$ for some B s.t. $|B| \equiv Q$.

Type formation and Subtyping

Good types

$$G[\ T \,||\, G_1 \rightarrow G[\ T_1 \,||\, G_2 \rightarrow G[T_2 \,||\, D]\ ]\ ]$$

- Delivery preserves the authority in control of values
- T_i are supertypes of T

Subtyping

$$\frac{\Gamma \vdash T <: T' \qquad \Gamma \vdash \Delta <: \Delta'}{\Gamma \vdash G[\ T \,||\, \Delta\ ] <:\ G\ [\ T' \,||\, \Delta'\ ]}$$

Where T' contains less capabilities then T and
Δ' is more restrictive then Δ

Core Typing Rules

(Delivery)

$$\frac{\Gamma \vdash n_{[\phi]} : G[\, T \,\|\, \Delta \,] \qquad \Gamma \vdash m_{[\phi]} : G_1[\, T_1 \,\|\, \Delta_1 \,] \qquad (G_1 \rightarrow \tau \in \Delta) \text{ or } (\text{Default} \rightarrow \tau \in \Delta)}{\Gamma \vdash n_{[\phi m]} : \tau}$$

Core Typing Rules

(Input)

$$\frac{\Gamma \vdash a : G[\,(\tau_1 \ldots \tau_k)^r \,\|\, \Delta\,] \qquad \Gamma, x_1{:}\tau_1, \ldots, x_k{:}\tau_k \vdash P}{\Gamma \vdash a(\,x_1{:}\tau_1, \ldots, x_k{:}\tau_k\,).\ P}$$

(Output)

$$\frac{\Gamma \vdash a : G[\,(\tau_1 \ldots \tau_k)^w \,\|\, \Delta\,] \qquad \Gamma \vdash P \qquad \Gamma \vdash b_i : G_i\,[\,T_i \,\|\, \Delta_i\,] \qquad \Gamma \vdash \Delta_i(G) <: \tau_i}{\Gamma \vdash a\langle b_1, \ldots, b_k \rangle\ .\ P}$$

Soundness: Access Control

If $\Gamma \vdash n_{[\phi]} < a_1,\ldots,a_k >.A \mid n_{[\varepsilon]}(x_1{:}\rho_1,\ldots,x_k{:}\rho_k).B$ then

$\Gamma \vdash n_{[\phi]} : G[\ (\tau_1,\ldots,\tau_k)^{w} \parallel \Delta\ ]$

The write capability
has been grated
to the writer process

$\Gamma \vdash n_{[\varepsilon]} : G[\ (\rho_1,\ldots,\rho_k)^{r} \parallel \Delta'\ ]$

The read capability
has been grated
to the reader process

$\Gamma \vdash a_i : \sigma_i \qquad \sigma_i \downarrow G <: \tau_i \qquad \tau_i <: \rho_i$

The types σ_i of the emitted values
allow a_i to be delivered to G at a subtype
of the type ρ_i at which they are received

Soundness: Flow Control

Let $\Gamma \vdash A$ be a derivable judgment

depending on the judgment $\Gamma' \vdash n_{[\phi]} : \tau$

then

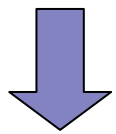
$\Gamma'(n) = \rho$ such that $\rho \downarrow \phi <: \tau$

n flowed as described in ϕ

The type ρ allows n to be delivered at a subtype of τ after flowing according to ϕ

Type soundness

Access control + Flow control
+ Subject Reduction



Safety properties preserved
along the computation

… Secrecy as in [CGG00] observing that

$$\langle\!\langle G[T_1,\ldots,T_n] \rangle\!\rangle = \mu X.\ G[(\langle\!\langle T_1 \rangle\!\rangle,\ldots,\langle\!\langle T_n \rangle\!\rangle)^{rw} \parallel \text{Default} \rightarrow X\ ]$$

Conclusions

What we have done

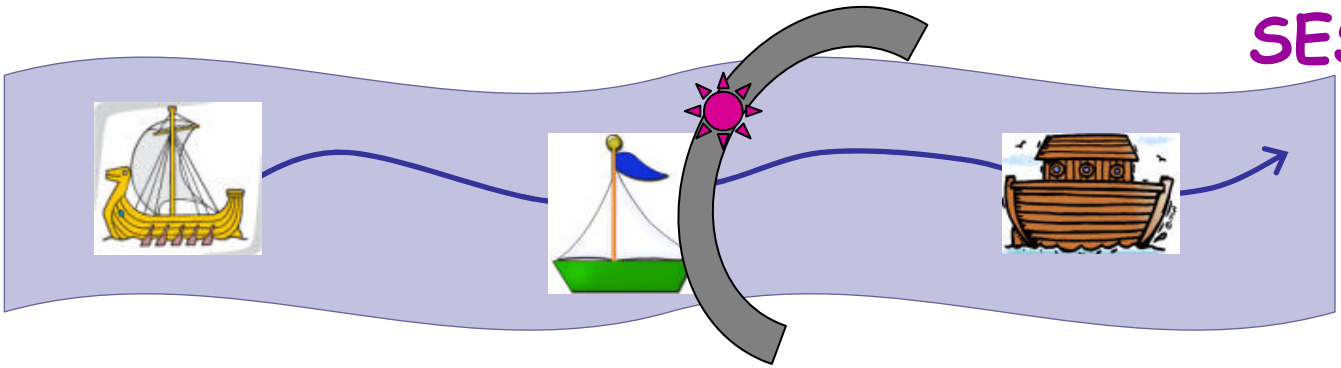
- Developed a new type foundation for discretionary access control policies
- Flexible/powerful and provides strong security guarantees
- A conservative extension of [CGG00]'s type system

A lot to be done

- Allow changes in the ownership of names, account for ordering relationships over authorities,
- Accommodate declassification mechanisms
- Study the import of type-based policies with typed behavioral equivalences

Delivery Types vs Session Types

SESSION TYPES



: Observation point

DELIVERY TYPES