Confining Data and Processes in Global Computing Applications

Daniele Gorla

Joint work with *Rocco De Nicola* and *Rosario Pugliese*

Dipartimento di Sistemi e Informatica – Università di Firenze

Work partially supported by

EU project MIKADO IST-2001-32222

Mikado/MyThS/Dart joint workshop

Venice, June 15th, 2004

Outline

- Motivations
- Klaim
  - Main Features and Syntax
  - Confining Data and Processes
    - Annotating Data and Network Nodes
    - A Static Compilation Phase
    - Operational Semantics with Dynamic Type Checking
    - Main results: subject reduction & safety
  - Implementing Access Control and Ruling out Denial-of-Services
- Confining Data and Processes Statically: Dπ and Ambient
- Conclusions

Motivations

Process mobility is a fundamental aspect of global computing; however it gives rise to a lot of relevant security problems

- Malicious agents can attempt to access private information of the nodes hosting them
- Malicious hosts can try to compromise agent's secrecy

Our Aim:

- enforcing data secrecy at the level of the programming language
- developing a simple (but powerful) alternative to cryptography

KLAIM: Kernel Language for Agent Interaction and Mobility

A LINDA *derived language:*

- Asynchronous Communication via shared repositories (*tuple spaces*)
- *Tuples*: sequences of fields
- Tuples are anonymous and associatively selected via *pattern matching*

GC Features:

- Network Awareness
- Dynamically Evolving Flat Net Architecture (*node creation*)
- Process Distribution and Mobility
- Local and Remote Operations (*withdraw/generate tuples, spawn processes*)

cKLAIM Syntax

Nets	N	$::=$	$l :: C \quad \Big\vert \quad N_1 \parallel N_2$
Components	C	$::=$	$P \quad \Big\vert \quad \langle d \rangle \quad \Big\vert \quad C_1 \vert C_2$
Processes	P	$::=$	$\mathbf{nil} \quad \Big\vert \quad a.P \quad \Big\vert \quad P_1 \mid P_2 \quad \Big\vert \quad * P$
Actions	a	$::=$	$\mathbf{in}(T)@v \quad \Big\vert \quad \mathbf{out}(u)@v \quad \Big\vert \quad \mathbf{eval}(P)@v \quad \Big\vert \quad \mathbf{newloc}(l)$
Templates	T	$::=$	$!\,x \quad \Big\vert \quad u$

Annotating Data and Nodes for Confinement

Main ideas:

- *Regions* are finite sets of node addresses
  (to refer all node addresses we use $\top$)
- each datum is tagged with a region to program the subnet where the datum can appear
- a process can retrieve a datum if its execution does not violate the region tagging the datum

Moreover, to add flexibility and expressiveness

- each node l is tagged with two regions r_d and r_p
  - r_d controls the nodes that can create data in l
  - r_p controls the nodes that spawn processes over l

Preserving Confinement through Computations

Communication Rule:

$$l :: \mathbf{in}(!x)@l'.P \parallel l' :: \langle [d]_r \rangle \quad \rightarrowtail \quad l :: P[{}^{d}\!/\!_{x}] \parallel l' :: \mathbf{nil}$$

Main Check: ensure that $P[{}^{d}\!/\!_{x}]$ does not violate r, i.e.

- $P[{}^{d}\!/\!_{x}]$ writes d only in nodes of r
- $P[{}^{d}\!/\!_{x}]$ spawns processes containing d only to nodes of r

This would require code inspection (too expensive at run-time)

A Static Compilation

Annotating input variables to describe how data retrieved are used

E.g.

$$l :: \mathbf{in}(!x)@l'.\mathbf{out}(x)@h.\mathbf{eval}\big(\mathbf{out}(x)@l''.Q\big)@k$$

should be annotated as

$$l :: \mathbf{in}([!x]^{\{l,h,k,l''\}})@l'.\mathbf{out}(x)@h.\mathbf{eval}\big(\mathbf{out}(x)@l''.Q\big)@k$$

assuming that x does not occur in Q

Variables are annotated by a (simple and efficient) static compilation phase, whose main judgment is $N \succ N'$ (we say that N' is *compiled*)

Dynamic Semantics

Communication Rule:

$$\frac{r \subseteq r'}{l :: \mathbf{in}([!x]^r)@l'.P \parallel l' :: \langle [d]_{r'} \rangle \quad \rightarrowtail \quad l :: P[{}^{d}\!/\!_{x}] \parallel l' :: \mathbf{nil}}$$

Main Results:

Subject Reduction: If N is compiled and $N \rightarrowtail N'$ then N' is compiled

Safety: If N is compiled then, for any $[d]_r$ occurring in N and for all possible evolutions of N, it holds that d only crosses nodes in r

Localized Safety: the results above also hold if only a (properly defined) subnet of N is compiled (see the paper)

Ruling out Denial-of-Service Attacks

A client application like

$$client \;::\; \mathbf{out}([service\_req]_{\{client,server\}})@server.P$$

robustly avoids the denial-of-service attack

$$intruder \;::\; \mathbf{in}(service\_req)@server$$

aiming at cancelling the service request from the server

Indeed, only processes located at *client* and *server* can see the datum $service\_req$

Implementing Access Control Lists

If res is the name of a resource in l readable by nodes in r, then the datum

$$l \;::\; \langle res, [info]_r \rangle$$

implements the access control list for res. Indeed, reading res could be programmed as

$$l' \;::\; \mathbf{in}(res, !x)@l.P$$

that, upon compilation, becomes

$$l' \;::\; \mathbf{in}(res, [!x]^{\{l', \ldots\}})@l.P$$

This process can evolve only if $l' \in r$

Dynamic *vs* Static Type Checking

- Klaim uses a combination of both static and dynamic type checking (the inference of regions for template variables *vs* region inclusions)

- Everything can be done statically, if we assume that each tuple space hosts tuples of the same sort
  - this SHARPLY CONTRASTS the tuple spaces paradigm!
  - it is standard in languages based on channels or derived from Ambient

Dπ Syntax

Nets	N	$::=$	$l[\![\, P \,]\!]$	$N_1 \parallel N_2$	$(\nu e_k)N$		
Processes	P	$::=$	$\mathbf{stop}$	$\alpha.P$	$P_1 \mid P_2$	$(\nu e)P$	$*\, P$
Actions	α	$::=$	$u!\langle W \rangle$	$u?(X)$	$\mathbf{go}\ u$		

Dπ with Regions

- Region annotations: $u!\langle [W]_r \rangle$
- Communiation rule:

$$l[\![\, a!\langle [W]_r \rangle.P \;\mid\; a?(X).Q \,]\!] \quad \rightarrow \quad l[\![\, P \;\mid\; Q[{}^{W}\!/X] \,]\!]$$

  provided that $Q[{}^{W}\!/X]$ carries W only through sites whose addresses are in r

- Typing channels (adapted from [Pierce & Sangiorgi]):
  - a is associated to region r_a
  - outputs on a can be specified only with $r_{out} \supseteq r_a$
  - data retrieved from a can be used only in $r_{in} \subseteq r_a$
  - this enforces the required $r_{in} \subseteq r_{out}$

The Ambient Calculus

$$P \quad ::= \quad \mathbf{0} \quad \Big| \quad a[P] \quad \Big| \quad \alpha.P \quad \Big| \quad P_1 \mid P_2 \quad \Big| \quad (\nu n)P \quad \Big| \quad * \, P$$

$$\alpha \quad ::= \quad \mathbf{in}\_u \quad \Big| \quad \mathbf{out}\_u \quad \Big| \quad \mathbf{open}\_u \quad \Big| \quad (x) \quad \Big| \quad \langle n \rangle$$

Confinement in Ambient

- As usual, we tag data in output actions with regions, $\langle [d]_r \rangle$
- Most problems arises from the **open**. E.g., consider the ambient

$$n[\langle [d]_{\{n\}} \rangle . \cdots]$$

where the secrecy of d is respected. However, the compound system

$$m[\ \ n[\langle [d]_{\{n\}} \rangle . \cdots] \ \mid \ \mathbf{open}\, n \ \ ] \ \rightarrow \ m[\ \langle [d]_{\{n\}} \rangle . \cdots \ ]$$

breaks d's secrecy!

Types for Confinement in Ambient (1)

The type of an ambient takes the form

$$r_1 \triangleright r_2 \triangleright r_3[T]$$

If an ambient u is assigned such a type, then

- r_1 is the set of ambients that can see the name u
- r_2 is the set of ambients that can contain ambients named u
- r_3 is the set where u can asssume its name (this is useful only when u is a variable and avoids dependent types)
- T is the topic of conversation (like in [Cardelli & Gordon])

Types for Confinement in Ambient (2)

Key requirements:

1. whenever n is contained in m (i.e., $m[n[\cdots] \mid \cdots]$), it must hold that $\{m\} \cup \mathit{cont}(m) \subseteq \mathit{cont}(n)$

2. for any datum $\langle [d]_r \rangle$ in n, we must have that $r \cup \mathit{cont}(n) \subseteq r$

This prevents leaks of data security:

$$m[\ n[\langle [d]_r \rangle . \cdots \mid \mathbf{open}\_n\ ] \ \rightarrow\ m[\langle d_r \rangle . \cdots]$$

Well-typedness of $m[\ n[\langle [d]_r \rangle . \cdots \mid \mathbf{open}\_n\ ]$ implies that

$$m \in \mathit{cont}(n) \subseteq r$$

that implies well-typedness of $m[\langle [d]_r \rangle . \cdots]$

Conclusions

- the approach presented is simple and efficient, and can be adapted to different calculi

- it is powerful enough to easily implement access control and rule out denial-of-service attacks

- it is useful also in a cryptographic setting
  (to ensure the secrecy of an encrypted datum we need to ensure the confinement of the decryption key!)

My homepage: `http://www.dsi.uniroma1.it/~gorla/`

Controlling Incoming Data/Processes

Datum Creation (to refuse undesired data):

$$\frac{l \in r'_d}{l \;{}_{r_d}\!::_{r_p}\; \mathbf{out}([d]_r)@l'.P \parallel l' \;{}_{r'_d}\!::_{r'_p}\; C \quad \rightarrowtail \quad l \;{}_{r_d}\!::_{r_p}\; P \parallel l' \;{}_{r'_d}\!::_{r'_p}\; C \mid \langle [d]_r \rangle}$$

Process Spawning (to refuse possibly dangerous processes):

$$\frac{l \in r'_p}{l \;{}_{r_d}\!::_{r_p}\; \mathbf{eval}(Q)@l'.P \parallel l' \;{}_{r'_d}\!::_{r'_p}\; C \quad \rightarrowtail \quad l \;{}_{r_d}\!::_{r_p}\; P \parallel l' \;{}_{r'_d}\!::_{r'_p}\; C \mid Q}$$