Global Computing

WAN programming for building *global* systems. They are hard to be made robust because:

- Absence of centralised control
- Client-Server not enough: P2P
- Administrative domains (*Security*)
- Interoperability
  - different platforms
  - different devices
    (e.g. PDA, laptop, mobile phones...)
- “Mobility” (resources and computation)
- ...

G2G1 L
1 54
2
3
G
a
L open a
a
open a
Intruder
Knowledge
P1 Pn
x
q
θ
!...
– p. 1/
??
Global Computing

WAN programming for building *global* systems. They are hard to be made robust because:

- **Network Awareness**
  - Applications are location dependent
  - Locations have different features
  - and allow multiple (security) policies

- Service Level Agreement
  - Independently programmed in a distributed environment
  - Reasoning on space and time
  - ...

...
Web Services: A programming metaphor

Applications access services that must be
- Published
- Searched
- Binded

Services are
- “Autonomous”
- Independent (local choices, independently built)
- Mobile/stationary
- “Interconnected”

Security issues: hostile environment
$\pi$-calculus [?] (very basic wrt WAN)

- Ambient [?, ?, ?]
- Djoin [?, ?]
- $D_\pi$ [?, ?]
- Klaim [?, ?, ?]
- Seal [?]
A Model for Declarative WAN Programming

In collaboration with G. Ferrari and U. Montanari
Hypergraphs Programming model

- Graphs for distributed systems
- Edge replacement for graph rewritings
- Edge replacement/distributed constraint solving problem
- Graphs grammars for software architecture styles
- Synchronised Hyperedge Replacement (SHR) with mobility for name passing calculi
Hypergraphs Programming model

We aim at tackling new *non-functional* computational phenomena of systems using SHR. The metaphor is

- “WAN systems *as* Hypergraphs”
- “WAN computations *as* SHR”

In other words:

- Components are represented by hyperedges
- Systems are *bunches* of (connected) hyperedges
- Computing means to rewrite hyperedge...
- ...according to a synchronisation policy
Hyperedges and Hypergraphs Syntax

A hyperedge generalises edges: It connects more than two nodes

\[ L : 3, \quad L(y, z, x), \]

\[ \begin{array}{c}
  x & \xrightarrow{3} & L \\
  \downarrow & & \downarrow 1 \\
  y & \xrightarrow{2} & z 
\end{array} \]
Hyperedges and Hypergraphs Syntax

A hyperedge generalises edges: It connects more than two nodes

\[ L : 3, \quad L(y, z, x), \]

\[ G ::= \text{nil} \mid \nu y.G \]
\[ \quad \mid L(\bar{x}) \quad \mid G\mid G \]

\[ xyL \]

\[ 1 \]

\[ 3 \quad 2 \]

\[ z \]
Hyperedges and Hypergraphs Syntax

A hyperedge generalises edges: It connects more than two nodes

\[ L : 3, \quad L(y, z, x), \]

\[ G ::= \text{nil} \mid \nu y.G \mid L(x) \mid G|G \]

Syntactic Judgement

\[ \Gamma \vdash G, \quad fn(G') \subseteq \Gamma \]
Hyperedges and Hypergraphs Syntax

A hyperedge generalises edges: It connects more than two nodes

\[ L : 3, \quad L(y, z, x), \]

\[ G ::= \text{nil} \mid \nu y.G \]

\[ \mid L(\vec{x}) \mid G|G \]

**Syntactic Judgement**

\[ \Gamma \vdash G, \quad \text{fn}(G') \subseteq \Gamma \]

An example:

\[ L : 3, \quad M : 2 \]

\[ x, y \vdash \nu z.(L(y, z, x)|M(y, z)) \]
Hyperedges and Hypergraphs Syntax

A hyperedge generalises edges: It connects more than two nodes

\[ L : 3, \quad L(y, z, x), \]

\[ G ::= \text{nil} \mid \nu \ y . G \]
\[ \mid L(\overline{x}) \mid G | G \]

Syntactic Judgement

\[ \Gamma \vdash G, \quad \text{fn}(G) \subseteq \Gamma \]

An example:

\[ L : 3, \quad M : 2 \]
\[ x, y \vdash \nu \ z . (L(y, z, x)|M(y, z)) \]
Replacement of Hyperedges

\[ L \rightarrow G \]
Replacement of Hyperedges

$L \rightarrow G$
Replacement of Hyperedges

\[ L \rightarrow G \]
Replacement of Hyperedges

$L \rightarrow G$

- Edge replacement: local
- Synchronisation as distributed constraint solving
- Multiple synchronisation
- New node creation
- Node fusion: mobility model

Benefits:
- Uniform framework for \( L \), fusion
- LTS for Ambient ...
- ... for Klaim ...
- ... and path reservation for Qlaim
- Expressive for distributed coordination
**Replacement of Hyperedges**

$L \rightarrow G$

- Edge replacement: local
- Synchronisation as distributed constraint solving
- Multiple synchronisation
- New node creation
- Node fusion: mobility model

**Benefits:**
- Uniform framework for $\pi$, $\pi$-I, fusion
- LTS for Ambient ...
- ... for Klaim ...
Replacement of Hyperedges

$L \rightarrow G$

- Edge replacement: local
- Synchronisation as distributed constraint solving
- Multiple synchronisation
- New node creation
- Node fusion: mobility model

Benefits:
- Uniform framework for $\pi$, $\pi$-I, fusion
- LTS for Ambient ...
- ... for Klaim ...
- ... and path reservation for Qlaim
- wireless networks
- expressive for distributed coordination
Hypergraph Semantics: Productions

\[
x_1, \ldots, x_n \vdash L(x_1, \ldots, x_n) \xrightarrow{\Lambda}{X} \Gamma \vdash G,
\]

- \( \Lambda \subseteq X \times Act \times \mathcal{N}^* \) set of constraints
- \( \pi : X \rightarrow X \) fusion substitution, i.e.
  \[\forall x_i, x_j \in X. \pi(x_i) = x_j \Rightarrow \pi(x_j) = x_j\]
- \( \Gamma = \pi(X) \cup (n(\Lambda) \setminus X) \)
- \( fn(G) \subseteq \Gamma \)
Hypergraph Semantics: Productions

\[ \begin{align*}
x_1, \ldots, x_n &\vdash L(x_1, \ldots, x_n) \xrightarrow{\Lambda}{\pi} \Gamma \vdash G, \\
\end{align*} \]

- \( \Lambda \subseteq X \times Act \times \mathcal{N}^* \) set of constraints
- \( \pi : X \to X \) fusion substitution, i.e.
  \[ \forall x_i, x_j \in X. \pi(x_i) = x_j \Rightarrow \pi(x_j) = x_j \]
- \( \Gamma = \pi(X) \cup (\text{fn}(\Lambda) \setminus X) \)
- \( \text{fn}(G) \subseteq \Gamma \)

Graph Rewritings: \( \Gamma_1 \vdash G_1 \xrightarrow{\Lambda}{\pi} \Gamma_2 \vdash G_2 \)
Applying the Model

Ambient

\[ a[...] \mid \text{open } a \rightarrow \ldots \]
Applying the Model

Ambient

\[ a[...]|\text{open } a \rightarrow \ldots \]

Components

\[ a[\ldots] : \quad x \quad a \quad y , \]

\[ \text{open } a : \quad L_{\text{open } a} \rightarrow z \]
Applying the Model

**Ambient**

\[ a[\ldots]|\text{open } a \rightarrow \ldots \]

**Components**

\[ a[\cdots] : \quad x \quad a \quad y, \]

\[ \text{open } a : \quad \begin{array}{c}
\text{L}_{\text{open } a} \\
\end{array} \quad z \]

**Productions**

\[ x \quad a \quad y \quad \overset{\text{open } a}{\quad [y/x]} \quad y = x \quad \]

\[ \begin{array}{c}
\text{L}_{\text{open } a} \\
\end{array} \quad \overset{\text{open } a}{\quad} \quad z \quad \quad \overset{\text{open } a}{\quad} \quad z \]
Applying the Model: Node Fusion

\[ G \rightarrow a \rightarrow L_{\text{open } a} \]
Applying the Model: Node Fusion

\[ G \]

\[ L_{\text{open } a} \]

Intruder Knowledge

P1 Pn

- p. 11/?
Applying the Model: Node Fusion

\[
G_a \xrightarrow{a} L_{open \ a} \xrightarrow{\text{open } a} \quad y = x
\]

\[
G \xrightarrow{u} y = x \xrightarrow{v} z \quad \text{Intruder Knowledge}
\]
\[
{\text{nil}}_x = x \vdash \text{nil}
\]
\[
{\text{n}[P]}_x = x \vdash \nu y. (G \mid n(y, x)), \quad \text{if } y \neq x \land {\text{P}}_y = y \vdash G
\]
\[
{\text{M}.P}_x = x \vdash L_{\text{M}.P}(x)
\]
\[
{\text{P}_1|P_2}_x = x \vdash G_1 \mid G_2, \quad \text{if } {\text{P}_i}_x = x \vdash G_i \land i = 1, 2
\]
\[
\text{rec X. P}_x = \text{P[rec X. P}/X \text{]}_x
\]

**Theorem** \[ \_ \] \_ is a bijection on ambient graphs
Coordinating Productions for Ambient

\[ x, y \vdash b(x, y) \xrightarrow{\{ (x, \text{in } a, \langle \rangle), (y, \text{input } a, \langle z \rangle) \}} x, y, z \vdash b(x, z) \]

\[(\text{input1})\]

\[x \quad b \quad y \quad \text{in } a \quad \text{input } a, z \quad \Rightarrow \quad x \quad b \quad y \quad \text{in } a \quad \text{input } a, z \]

\[x, y \vdash a(x, y) \xrightarrow{\{ (y, \text{input } a, \langle x \rangle) \}} x, y \vdash a(x, y) \]

\[(\text{input2})\]

\[x \quad a \quad y \quad \text{input } a, x \quad \Rightarrow \quad x \quad a \quad y \quad \text{input } a, x \]
Coordination Productions for Ambient

\[ x, y \vdash b(x, y) \xrightarrow{\{(x, in\, a, \langle \rangle), (y, input\, a, \langle z \rangle)\}} x, y, z \vdash b(x, z) \]

\textit{(input1)}

\[ x \quad in\, a \quad b \quad y \quad \text{input} \, a, z \quad \Rightarrow \quad x \quad b \quad y \quad z \]

\[ x, y \vdash a(x, y) \xrightarrow{\{(y, input\, a, \langle x \rangle)\}} x, y \vdash a(x, y) \]

\textit{(input2)}

\[ x \quad \text{input} \, a, x \quad y \quad \Rightarrow \quad x \quad a \quad y \]
Theorem If $P \rightarrow Q$ then $\llbracket P \rrbracket_x \xrightarrow{\Lambda} \llbracket Q \rrbracket_x$ and

- either $\Lambda = \emptyset$
- or $\Lambda = \{(x, \tau, \langle \rangle)\}$
Semantic Correspondence

**Theorem** If $P \rightarrow Q$ then $\left[ P \right]_x \xrightarrow{\Lambda} \left[ Q \right]_x$ and

- either $\Lambda = \emptyset$
- or $\Lambda = \{(x, \tau, \langle \rangle)\}$

**Theorem** If $\left[ P \right]_x \xrightarrow{\Lambda} G$ is a basic transition, then

- either $\left[ P \right]_x = \Gamma \vdash G$
- or $\exists Q \in Proc : P \rightarrow Q \land \Gamma \vdash G = \left[ Q \right]_x$
Qlaim: Expressing and reasoning on Connection Properties

In collaboration with R. De Nicola, G. Ferrari, U. Montanari, R. Pugliese
Klaim [?]
Multiple TS
Multiple TS

Localities: first class citizens
- Multiple TS
- Localities: first class citizens
- Process migration
- Multiple TS
- Localities: first class citizens
- Process migration

\[ P \vdash \neg p.16/3 \]
Multiple TS

Localities: first class citizens

Process migration
Klaim [?]  

- Multiple TS  
- Localities: first class citizens  
- Process migration  

\[
P ::= \text{nil} \\
\quad | \quad \alpha.P \\
\quad | \quad P_1 \mid P_2 \\
\alpha ::= a@s \\
a ::= \ldots \text{ // Klaim actions} \\
\quad | \quad \text{eval}(P)
\]
Qlaim: Gateways

In [?]
In [?] Coordinators (super processes)
Claim: Gateways

In [?]
- Coordinators (super processes)
- Dynamic creation of sites
Qlaim: Gateways

In [?] 
- Coordinators (super processes)
- Dynamic creation of sites
- Gateway connection management
Claim: Gateways

In [?]

- Coordinators (super processes)
- Dynamic creation of sites
- Gateway connection management
Qlaim: Gateways

In [?]

- Coordinators (super processes)
- Dynamic creation of sites
- Gateway connection management
Claim: Gateways

In [?]

- Coordinators (super processes)
- Dynamic creation of sites
- Gateway connection management
Claim: Gateways

In [?]

- Coordinators (super processes)
- Dynamic creation of sites
- Gateway connection management

\[ P ::= \gamma.P \mid P_1 \mid P_2 \]
\[ \gamma ::= \alpha \]
\[ \mid \text{new}(s, P) \]
\[ \mid \text{login}(s, \kappa) \]
\[ \mid \text{accept}(s, \kappa) \]
\[ \mid \text{logout}(s, \kappa) \]
\[ \mid \text{disconnect}(s, \kappa) \]
Connection costs

Cost $\kappa$ abstracts characteristics of connections (bandwidth, latency, distance, access rights ...)

Algebra on costs: $c$-semiring. For instance $\mathbf{i}^\mathbf{h} c_1 \mathbf{i} \mathbf{h} c_2 \mathbf{i} = \mathbf{i}^\mathbf{h} c_1 + c_2 \mathbf{i}$ if $c_2 < c_1$ and $\mathbf{i}^\mathbf{h} c_1 \mathbf{i} \mathbf{h} c_2 \mathbf{i} = \mathbf{i}^\mathbf{h} c_1$ otherwise.
Connection costs

Cost $\kappa$ abstracts characteristics of connections (bandwidth, latency, distance, access rights ...)

Algebra on costs: $c$-semiring. For instance

$$\langle c_1, \pi_1 \rangle \oplus \langle c_2, \pi_2 \rangle = \langle c_1 + c_2, \pi_1 \cup \pi_2 \rangle$$

$$\langle c_1, \pi_1 \rangle \otimes \langle c_2, \pi_2 \rangle = \begin{cases} 
\langle c_1 + c_2, \pi_1 \cap \pi_2 \rangle & \text{if } c_2 < c_1 \text{ and } \pi_2 \subset \pi_1 \\
\perp & \text{otherwise}
\end{cases}$$
\[
\llbracket s ::^L, P \rrbracket = \Gamma \vdash \nu \vec{x}, p (\llbracket P \rrbracket_p | \mathcal{G}_m,n^s (\vec{u}, \vec{x}, p) | \prod_{j=1}^n G_{t_j}^{K_j} (x_j, v_j))
\]
\[ [ s :: L, P ] = \Gamma \vdash (\nu \bar{x}, p)([ P ]_p \mid \mathcal{G}^s_{m,n}(\bar{u}, \bar{x}, p) \mid \prod_{j=1}^{n} G^{K_j}_{t_j}(x_j, v_j)) \]

\[ [ \text{nil} ]_p = \text{nil} \]
\[ [ \text{outt} ]_p = L_{\text{outt}}(p) \]
\[ [ \gamma . P ]_p = L_{\gamma . P}(p) \]
\[ [ \text{eval}(P)@s ]_p = (\nu u)(\text{eval}^T_s(P)(u, p) \mid S_P(u)) \]
\[ [ P_1 \mid P_2 ]_p = [ P_1 ]_p \mid [ P_2 ]_p \]
\[ [ \text{rec } X . P ]_p = [ P[\text{rec } X . P / X ] ]_p . \]
Qlaim’s Graph semantics: pros & cons

- Many productions (recently reduced)
- Determines the “optimal” path (also Qlaim)
- Path reservation
- Path routing

Theorem
Qlaim remote actions are routed on paths with minimal cost (wrt the $c$-semiring operations)
Qlaim’s Graph semantics: pros & cons

- Many productions (recently reduced :-)

Intruder Knowledge
P₁ Pₙ

$x \sigma$!

\( G \rightarrow G' \)

L open \( a \)

\( G_a \)

\( \theta \)

\( \beta \)
Qlaim’s Graph semantics: pros & cons

- Many productions (recently reduced :-)
+ Determines the “optimal” path (also Qlaim)
Qclaim’s Graph semantics: pros & cons

- Many productions (recently reduced :-)
  + Determines the “optimal” path (also Qclaim)
  + Path reservation
Qlaim’s Graph semantics: pros & cons

- Many productions (recently reduced :-)
  + Determines the “optimal” path (also Qlaim)
  + Path reservation
  + Path routing
Qlaim’s Graph semantics: pros & cons

- Many productions (recently reduced :-)
  + Determines the “optimal” path (also Qlaim)
  + Path reservation
  + Path routing

**Theorem** Qlaim remote actions are routed on paths with minimal cost
  (wrt the c-semiring operations)
In [?] graph transformation is used for modelling dynamic behaviour of UML specifications.

- Formal semantics of computations
  - No local rewritings
  - Distribution is not considered

SHR has been applied as a further refinement step in the software design process.
Security

In collaboration with A. Bracciali, A. Brogi and G. Ferrari
The Dolev-Yao Model

Intruder Knowledge

P1 Pn

Receive and store any transmitted message
Hinder a message
Decompose messages into parts
Forge messages using known data

Perfect Encryption Hypothesis

Names n;m;:::;A;B;S;:::
Keys k;k0:::+A
Messages M::=NKM;MfgM
The Dolev-Yao Model

- Receive and store any transmitted message
- Hinder a message
- Decompose messages into parts
- Forge messages using known data
- Perfect Encryption Hypothesis

Diagram:
- Intruder Knowledge
- P1 to Pn
The Dolev-Yao Model

- Receive and store any transmitted message
- Hinder a message
- Decompose messages into parts
- Forge messages using known data
- Perfect Encryption Hypothesis

Names \( n, m, ..., A, B, S, ... \)

Keys \( k, k', ..., A^+, A^-, ... \)

Messages \( M ::= N \mid K \mid M, M \mid \{M\}_M \)
Intruder capabilities: △

\[
\frac{m \in \kappa}{\kappa \triangleleft m} (\in) \quad \frac{\kappa \triangleleft m, n}{\kappa \triangleleft m, n} (,)
\]

\[
\frac{\kappa \triangleleft m, n}{\kappa \triangleleft m} (+1) \quad \frac{\kappa \triangleleft m, n}{\kappa \triangleleft n} (+2) \quad \frac{\kappa \triangleleft \{m\}_\lambda}{\kappa \triangleleft m} (\{\})
\]

Generalising [?] to asymmetric key cryptography

**Theorem** △ is decidable
Some design choices:

- Extension of IP
- Cryptography & communication (pattern-matching)
- Key-sharing via “name fusion”
- Rôle based calculus
- Multi-session facilities

\[
E, F ::= \text{nil} | \alpha.E | E + E | E \parallel E
\]

\[
\alpha, \beta ::= \text{in}(d) | \text{out}(d)
\]

\[
d ::= N | K | d, d | \{d\}_d | x | ?x
\]
A cIP protocol

1. \( A \rightarrow B : \{na, A\}_{B^+} \)
2. \( B \rightarrow A : \{na, nb\}_{A^+} \)
3. \( A \rightarrow B : \{nb\}_{B^+} \)

\[
\begin{align*}
A & \overset{\Delta}{=} (y) \left[ \begin{array}{l}
\text{out} (\{na, A\}_{y^+}) . \\
\text{in} (\{na, ?u\}_{A^-}) . \\
\text{out} (\{u\}_{y^+})
\end{array} \right] \\
B & \overset{\Delta}{=} () \left[ \begin{array}{l}
\text{in} (\{?x, ?z\}_{B^-}) . \\
\text{out} (\{x, nb\}_{z^+}) . \\
\text{in} (\{nb\}_{B^-})
\end{array} \right]
\end{align*}
\]
cIP Semantics

\[
\begin{align*}
\alpha.E & \xrightarrow{\alpha} E \\
E & \xrightarrow{\alpha} E' \quad E + F & \xrightarrow{\alpha} E' \\
E & \xrightarrow{\alpha} E' \quad F & \xrightarrow{\alpha} E' \\
E \parallel F & \xrightarrow{\alpha} E' \parallel F & \text{bn}(\alpha) \cap \text{fn}(F) = \emptyset
\end{align*}
\]

\[
\begin{align*}
E_i^{\text{in}(d)} & \xrightarrow{\text{d}(\kappa) \triangleright m : \exists \sigma \text{ grounds.t. } d\sigma \sim m} E_i' \\
\langle (\tilde{X}_i)[E_i] \cup C, \chi, \kappa \rangle & \mapsto \langle (\tilde{X}_i)[E_i'] \sigma \cup C, \chi\sigma, \kappa \rangle \\
E_i & \xrightarrow{\text{out}(m)} E_i' \\
\langle (\tilde{X}_i)[E_i] \cup C, \chi, \kappa \rangle & \mapsto \langle (\tilde{X}_i)[E_i'] \cup C, \chi, \kappa \cup m \rangle \\
C' & = \text{join}(A_i, \gamma, C) \\
A \triangleq (\tilde{X})[E] & \text{ i new} \\
\langle C, \chi, \kappa \rangle & \mapsto \langle C', \chi\gamma, \kappa \cup \{A_i\} \rangle
\end{align*}
\]
A symbolic cIP trace

\[
\langle (y_1)[\text{out}\{(na_1, A_1)_{y_1}^+\}].\text{in}\{(na_1, ?u_1)_{A_1}^-\}.*\text{out}\{(u_1)_{y_1}^+\}], \varepsilon, \{A_1\}\rangle
\]

\[
\langle (y_1)[\text{in}\{(na_1, ?u_1)_{A_1}^-\}.\text{out}\{(u_1)_{y_1}^+\}], \varepsilon, \{A_1, \{na_1, A_1\}_{y_1}^+\}\rangle
\]

\[
\langle \text{out}\{(B_2 / y_1) \} \rangle
\]

\[
\kappa = \{A_1, B_2, \{na_1, A_1\}_{B_2}^+\}
\]

\[
\langle ()[\text{in}\{(na_1, ?u_1)_{A_1}^-\}.*\text{out}\{(u_1)_{B_2}^+\}], \rangle
\]

\[
\langle ()[\text{in}\{?x_2, ?z_2\}_{B_2}^-] .\text{out}\{(x_2, nb_2)_{z_2}^+\}.*\text{in}\{(nb_2)_{B_2}^-\}] \rangle
\]

\[
\langle \text{out}\{(x_2(\kappa), nb_2\}_{A_1}^+\}.*\text{in}\{(nb_2)_{B_2}^-\}] \rangle
\]

\[
\langle ()[\text{in}\{(na_1, ?u_1)_{A_1}^-\}.*\text{out}\{(u_1)_{B_2}^+\}], \rangle
\]

\[
\langle ()[\text{out}\{(x_2(\kappa), nb_2\}_{A_1}^+\}.*\text{in}\{(nb_2)_{B_2}^-\}] \rangle
\]

\[
\langle B_2, x_2(\kappa), A_1 / y_1, x_2, z_2, \kappa \rangle
\]
"If B completes a protocol session and thinks that he has been talking to A, then A had started a protocol session thinking that she has been talking to B"
Hypergraphs for security

\[ A^\triangleleft(y)[ \begin{array}{l} out(\{na, A\}_{y^+}). \\ in(\{na, ?u\}_{A^-}). \\ out(\{u\}_{y^+}) \end{array} ] \]

\[ B^\triangleleft()[ \begin{array}{l} in(\{?x, ?z\}_{B^-}). \\ out(\{x, nb\}_{z^+}). \\ in(\{nb\}_{B^-}) \end{array} ] \]
Hypergraphs for security

\[ A = (y) \]

\[ out(\{na, A\}_{y^+}) \]
\[ in(\{na, ?u\}_{A^-}) \]
\[ out(\{u\}_{y^+}) \]

\[ B = () \]
\[ in(\{?x, ?z\}_{B^-}) \]
\[ out(\{x, nb\}_{z^+}) \]
\[ in(\{nb\}_{B^-}) \]
Hypergraphs for security

\[ \Delta A(y) = \begin{cases} \text{out}(\{na, A\}_{y^+}). \\ \text{in}(\{na, ?u\}_{A^-}). \\ \text{out}(\{u\}_{y^+}) \end{cases} \]

\[ \Delta B() = \begin{cases} \text{in}(\{?x, ?z\}_{B^-}). \\ \text{out}(\{x, nb\}_{z^+}). \\ \text{in}(\{nb\}_{B^-}) \end{cases} \]
Hypergraphs for security

\[ A \equiv (y)[ \begin{array}{l}
\text{out}(\{na, A\}y+) . \\
\text{in}(\{na, ?u\}A-).
\end{array} \]

\[ B \equiv ()[ \begin{array}{l}
\text{in}(\{?x, ?z\}B-).
\text{out}(\{x, nb\}z+).
\text{in}(\{nb\}B-).
\end{array} \] \]
Hypergraphs for security

\[
A \triangleq (y)[ \quad \text{out}(\{na, A\}_{y+}). \\
\text{in}(\{na, ?u\}_{A-}). \\
\text{out}(\{u\}_{y+})]
\]

\[
B \triangleq ()[ \quad \text{in}(\{?x, ?z\}_{B-}). \\
\text{out}(\{x, nb\}_{z+}). \\
\text{in}(\{nb\}_{B-})]
\]
Hypergraphs for security

\[
A \equiv (y) \begin{cases} 
\text{out}(\{na, A\}_y^+). \\
\text{in}(\{na, ?u\}_{A^-}). \\
\text{out}(\{u\}_y^+) 
\end{cases}
\]

\[
B \equiv (\cdot) \begin{cases} 
\text{in}(\{?x, ?z\}_B^-). \\
\text{out}(\{x, nb\}_z^+). \\
\text{in}(\{nb\}_B^-). 
\end{cases}
\]
Mihda: Co-Algebraic Minimisation of Automata

In collaboration with G. Ferrari, U. Montanari and R. Raggi
Minimizing History Dependent Automata:
- HD-automata for history dependent calculi
- Co-algebraic specification
- Partition Refinement Algorithm based on co-algebraic specification
- Mihda: Ocaml implementation (refining $\lambda \vdash \Pi, \Sigma$ spec.)

<table>
<thead>
<tr>
<th></th>
<th>Comp. Time</th>
<th>States</th>
<th>Trans.</th>
<th>Min. Time</th>
<th>States</th>
<th>Trans.</th>
</tr>
</thead>
<tbody>
<tr>
<td>GSM small</td>
<td>0m 0.931s</td>
<td>211</td>
<td>398</td>
<td>0m 4.193s</td>
<td>105</td>
<td>197</td>
</tr>
<tr>
<td>GSM full</td>
<td>0m 8.186s</td>
<td>964</td>
<td>1778</td>
<td>0m 54.690s</td>
<td>137</td>
<td>253</td>
</tr>
</tbody>
</table>
Mihda Architecture

- Adherent to specs
- Highly modular
- Easily extendible
The main step
let bundle hd q =
  List.sort compare
  (List.iter (fun h -> (Arrow.source h) = q) (arrows hd))
The main step

List.map \( h_n \) bundle
The main step

\[ h_{n+1} = \text{\textit{norm}}\langle\text{\textit{states}}, \{\langle\ell, \pi, h_n(q'), \sigma' ; \sigma\rangle | q \xrightarrow{\ell \pi \sigma} q' \land \sigma' \in \Sigma_n(q')\}\rangle \]

\[
\text{let red } bl = ...... \\
\text{let bl_in = List.filt_iter covered_in bl} \\
\text{in list_diff bl bl_in }
\]
let an = active_names_bundle (red bundle) in
let remove_in ar = match ar with
  | Arrow(_,_,In(_,_)) → not (List.mem (obj ar) an)
  | _ → false in
  list_diff bundle (List.fold remove_in bundle)
\[ \Sigma_{n+1}(q) = (\text{compute\_group (norm bundle)}) \quad ; \quad \theta_q^{-1} \]
\[ \Sigma_{n+1}(q) = (\text{compute\_group (norm bundle)}) \;; \; \theta_q^{-1} \]

**Theorem** At the end of each iteration, blocks corresponds to \( h_{H_i} \).
Mihda Web Interface

http://jordie.di.unipi.it:8080/pweb
Declarative approach to WAN programming
- Foundational aspects
- QoS at application level
- Modelling wireless communications (ongoing)
- Integrating Milner & Hoare synchronizations
- Web Services
- Secure composition of components
- Coordination mechanism

Tool development
- Distributed infrastructure
- Base on Web Services metaphor
- Proof strategies as programmable coordinators
Ferrari, G., Pugliese, R., Tuosto, E. Calculi for Network Aware Programming. WOA’00

Ferrari, G., Montanari, U., Tuosto, E. LTS Semantics of Ambients via Graph Synchronization with Mobility. ICTCS’01


Bracciali, A., Brogi, A., Ferrari, G., Tuosto, E. Security and Dynamic Compositions of Open Systems. PDPTA’02

Ferrari, G., Montanari, U., Tuosto, E. Graph-based Models of Internetworking Systems. UNU/IIST Colloquium’03

References


[CG00] Luca Cardelli and Andrew D. Gordon. Mobile am-
   bients. *TCS: Theoretical Computer Science*, 240,
   2000.

[CJM98] Edmund M. Clarke, Somesh Jha, and Wilfredo R.
   Marrero. Using state space exploration and a nat-
   ural deduction style message derivation engine to
   verify security protocols. In *IFIP Working Con-
   ference on Programming Concepts and Methods
   (PROCOMET)*, 1998.

[CM83] Ilaria Castellani and Ugo Montanari. Graph Gram-
   mars for Distributed Systems. In Hartmut Ehrig,
   Manfred Nagl, and Grzegorz Rozenberg, editors,
   *Proc. 2nd Int. Workshop on Graph-Grammars and
   Their Application to Computer Science*, volume 153
   of *Lecture Notes in Computer Science*, pages 20–
   38. Springer-Verlag, 1983.

[DFP98] Rocco De Nicola, Gianluigi Ferrari, and Rosario
   Pugliese. KLAIM: A kernel language for agents in-
   teraction and mobility. *IEEE Transactions on Soft-

[DFPV00] Rocco De Nicola, Gianluigi Ferrari, Rosario
   Pugliese, and Betti Venneri. Types for access con-


Robin Milner, Joachim Parrow, and David Walker. A calculus of mobile processes, I and II. *Information"
