# A Formal Model of Provenance in Distributed Systems

Issam Souilah[2] Adrian Francalanza[1] Vladimiro Sassone[2]

[1]Department of Computer Science
ICT, University of Malta

[2]DSSE, ECS
Southampton University

San Francisco, USA. February 2009

# Outline

# Motivation

Trust In a Distributed System

# Motivation

## Trust In a Distributed System

- Distribution $\Rightarrow$ inherent **parallelism**.

# Motivation

## Trust In a Distributed System

- Distribution $\Rightarrow$ inherent **parallelism**.
- Distribution $\Rightarrow$ no shared memory *i.e.*, **message passing**.

# Motivation

## Trust In a Distributed System

- Distribution $\Rightarrow$ inherent **parallelism**.
- Distribution $\Rightarrow$ no shared memory *i.e.,* **message passing**.
- Distribution $\Rightarrow$ lack of centralised coordination *i.e.,* **non-determinism**.

# Motivation

## Trust In a Distributed System

- Distribution ⇒ inherent **parallelism**.
- Distribution ⇒ no shared memory *i.e.,* **message passing**.
- Distribution ⇒ lack of centralised coordination *i.e.,* **non-determinism**.

## Appropriate Calculus?

The **piCalculus** :

# Motivation

## Trust In a Distributed System

- Distribution $\Rightarrow$ inherent **parallelism**.
- Distribution $\Rightarrow$ no shared memory *i.e.,* **message passing**.
- Distribution $\Rightarrow$ lack of centralised coordination *i.e.,* **non-determinism**.

## Appropriate Calculus?

The **piCalculus** :

- Captures the characteristic features of our domain of study.

# Motivation

## Trust In a Distributed System

- Distribution $\Rightarrow$ inherent **parallelism**.
- Distribution $\Rightarrow$ no shared memory *i.e.*, **message passing**.
- Distribution $\Rightarrow$ lack of centralised coordination *i.e.*, **non-determinism**.

## Appropriate Calculus?

The **piCalculus** :

- Captures the characteristic features of our domain of study.
- Well-studied.

# Motivation

## Trust In a Distributed System

- Distribution $\Rightarrow$ inherent **parallelism**.
- Distribution $\Rightarrow$ no shared memory *i.e.*, **message passing**.
- Distribution $\Rightarrow$ lack of centralised coordination *i.e.*, **non-determinism**.

## Appropriate Calculus?

The **piCalculus** :

- Captures the characteristic features of our domain of study.
- Well-studied.
- Close connections to linear logic and resources
- ...

## Names

$$a, b, \ldots \in \textsc{Names}$$

## Names

$$a, b, \ldots \in \textsc{Names}$$

- denote points of interaction (rendez-vous **channels**)...

## Names

$$a, b, \ldots \in \textsc{Names}$$

- denote points of interaction (rendez-vous **channels**)...
- ...and **values** which are transmitted during communication.

## Processes

$$P \quad \| \quad Q \quad \| \quad R$$

## Processes

$$a!b \quad \| \quad Q \quad \| \quad R$$

# piCalculus Primer

## Processes

$$a!b \quad || \quad a?x.c!x \quad || \quad R$$

## Processes

$$a!b \quad \| \quad a?x.c!x \quad \| \quad R$$

## Processes

$$a!b \quad \| \quad a?x.x!c \quad \| \quad R$$

## Processes

$$\underbrace{a!b \quad || \quad a?x.x!c}_{\text{react on } a} \quad || \quad R$$

Processes

$$b!c \quad || \quad R$$

Processes

$$b!c \quad \| \quad b?y.R'$$

# piCalculus Extension: Distribution!

From Processes to Systems

$$P \quad \| \quad Q \quad \| \quad R$$

# piCalculus Extension: Distribution!

## From Processes to Systems

$$p, q, r, \ldots \in \textsc{TrustPrincipals}$$

$$p[P] \quad \| \quad q[Q] \quad \| \quad p[R]$$

# piCalculus Extension: Distribution!

From Processes to Systems

$$p[a?x.P] \quad \| \quad q[a!v_1] \quad \| \quad r[R]$$

# piCalculus Extension: Distribution!

## From Processes to Systems

$$\underbrace{p[a?x.P] \quad \| \quad q[a!v_1]}_{\textbf{across}\text{ units of trust}} \quad \| \quad r[R]$$

# piCalculus Extension: Distribution!

## From Processes to Systems

$$p[a?x.P] \quad \| \quad q[a!v_1] \quad \| \quad r[a!v_2]$$

**Market** of values!

# piCalculus Extension: Distribution!

## From Processes to Systems

$$\overbrace{p[\color{red}{a}?x.P] \quad \| \quad q[\color{red}{a}!v_1]}^{\text{communication}} \quad \| \quad r[a!v_2]$$

$$\underbrace{p[a?x.P] \quad \| \quad q[a!v_1] \quad \| \quad r[a!v_2]}_{\textbf{Market} \text{ of values!}}$$

# piCalculus Extension: Distribution!

## From Processes to Systems

$$\overbrace{p[a?x.P] \quad \| \quad q[a!v_1] \quad \| \quad r[a!v_2]}^{\text{communication}}$$

$$\underbrace{\hphantom{p[a?x.P] \quad \| \quad q[a!v_1] \quad \| \quad r[a!v_2]}}_{\textbf{Market}\text{ of values!}}$$

# Main Problem

What are the control mechanism need to assist consumers of data?

*How do we automate decisions based on trust?*

# Main Problem

What are the control mechanism need to assist consumers of data?

*How do we automate decisions based on trust?*

- Static Analysis *(not scalable)*

# Main Problem

What are the control mechanism need to assist consumers of data?

*How do we automate decisions based on trust?*

- Static Analysis *(not scalable)*
- Dynamic Analysis:
  - decisions need to be computationally lightweight.
  - decision criteria produced in lightweight fashion.

# Main Problem

What are the control mechanism need to assist consumers of data?

*How do we automate decisions based on trust?*

- Static Analysis *(not scalable)*
- Dynamic Analysis:
  - decisions need to be computationally lightweight. *(Full-blown* ***verification methods*** *do not cut it!)*
  - decision criteria produced in lightweight fashion.

# Main Problem

What are the control mechanism need to assist consumers of data?

*How do we automate decisions based on trust?*

- Static Analysis *(not scalable)*
- Dynamic Analysis:
  - decisions need to be computationally lightweight. *(Full-blown **verification methods** do not cut it!)*
  - decision criteria produced in lightweight fashion. *(**Proof-Carrying Code** does not cut it!)*

# Outline

Motivation

Proposed solution

Provenance Correctness

Conclusion

# Provenance

## Annotated Values

$$v :$$

# Provenance

## Annotated Values

$$v : \kappa$$

# Provenance

## Annotated Values

$$v : \kappa$$

| | |
|---|---|
| $\kappa ::= \epsilon$ | empty provenance |
| $\quad \mid \quad \alpha; \kappa$ | sequenced provenace |
| $\alpha ::= \textbf{rcv}(p, \kappa)$ | recieve action |
| $\quad \mid \quad \textbf{snd}(p, \kappa)$ | send action |

# Provenance

## Annotated Values

$$p[a!v]$$

# Provenance

## Annotated Values

$$p[a : \kappa_a \,!\, v : \kappa_v]$$

# Provenance

## Annotated Values

$$p[a : \kappa_a \,!\, v : \kappa_v] \quad \| \quad q[a : \kappa'_a \,!\, v' : \kappa_{v'}.] \quad \| \quad p[a : \kappa''_a \,!\, v'' : \kappa_{v''}.]$$

provenance is linear!

# Provenance Tracking

Automated:

# Provenance Tracking

Automated:

1. orthogonal to programming *(can be abstracted away)*

# Provenance Tracking

Automated:

1. orthogonal to programming *(can be abstracted away)*
2. ensures provenance annotation standardization.

# Provenance Tracking

Automated:

1. orthogonal to programming *(can be abstracted away)*
2. ensures provenance annotation standardization.
3. avoids circular reasoning with respect to trust.

# Provenance Tracking

Automated:

1. orthogonal to programming *(can be abstracted away)*
2. ensures provenance annotation standardization.
3. avoids circular reasoning with respect to trust.

Two tiered architecture:

# Provenance Tracking

Automated:

1. orthogonal to programming *(can be abstracted away)*
2. ensures provenance annotation standardization.
3. avoids circular reasoning with respect to trust.

Two tiered architecture:

- **Computation Layer:** describes computation of values and processes.

# Provenance Tracking

Automated:

1. orthogonal to programming *(can be abstracted away)*
2. ensures provenance annotation standardization.
3. avoids circular reasoning with respect to trust.

Two tiered architecture:

- **Computation Layer:** describes computation of values and processes.
- **Provenance Tracking Layer:** describes the aggregation of provenance information attached to data *(typically assigned to a trusted middleware)*

# Provenance Tracking Semantics

## Operational Semantics

$$p[a!v] \parallel Q \quad \longrightarrow \quad a\langle v\rangle \parallel Q$$

# Provenance Tracking Semantics

## Operational Semantics

$$p[a!v] \parallel Q \quad \longrightarrow \quad \underbrace{a\langle v\rangle}_{\text{loose immediate provenance information!}} \parallel Q$$

# Provenance Tracking Semantics

## *(Provenance Tracking)* Operational Semantics

$$p[a : \kappa_a ! v : \kappa_v] \parallel Q \quad \longrightarrow \quad a\langle v : \underbrace{\mathbf{snd}(p, \kappa_a); \kappa_v}_{\text{provenance aggregation}} \rangle \parallel Q$$

# Provenance Usage

Not-Automated!

# Provenance Usage

Not-Automated!

- program with it to control non-derminism...

# Provenance Usage

Not-Automated!

- program with it to control non-derminism. . .
- . . . using intuitive programming idioms/constructs

# Provenance Usage

Not-Automated!

- program with it to control non-derminism...
- ...using intuitive programming idioms/constructs

## Operational Semantics

$$a\langle v\rangle \parallel q[a?(x).Q] \longrightarrow q[Q\{^{v}/_{x}\}]$$

# Provenance Usage

Not-Automated!

- program with it to control non-derminism. . .
- . . . using intuitive programming idioms/constructs

## Operational Semantics

$$a\langle v : \kappa_v \rangle \parallel q[a : \kappa_a ?(x \ \textbf{from} \ \pi).Q] \longrightarrow q[Q\{^v/_x\}] \quad \text{if } \kappa_v \models \pi$$

$\underbrace{\hphantom{\textbf{from} \ \pi}}_{\text{provenance pattern matching}}$

# Provenance Usage

Not-Automated!

- program with it to control non-derminism...
- ...using intuitive programming idioms/constructs

## Operational Semantics

$$a\langle v{:}\kappa_v\rangle \parallel q[a{:}\kappa_a?(x\,\mathbf{from}\,\pi).Q] \;\longrightarrow\; q[Q\{^{v:\underbrace{\mathbf{rcv}(q,\kappa_a);\kappa_v}_{\text{provenance aggregation}}}/x\}] \quad \text{if } \kappa_v \models \pi$$

# Provenance Usage Example

## Client/Server

| | |
|---|---|
| *srv* | Name of server |
| *ret* | Name of return channel on which server returns answer |

$$\begin{aligned} \mathit{Client} &= p[\,\mathit{srv}!\langle \mathit{ret}\rangle] \quad \| \quad p[\,\mathit{ret}?(x\ \mathbf{from}\ \pi).P] \\ \mathit{Server} &= q[\,\mathit{srv}?(y\ \mathbf{from}\ *).y!\langle v\rangle] \end{aligned}$$

# Provenance Usage Example

## Client/Server

| | |
|---|---|
| *srv* | Name of server |
| *ret* | Name of return channel on which server returns answer |

$$\mathit{Client} = p[\,\mathit{srv}!\langle \mathit{ret}\rangle] \quad \| \quad p[\,\mathit{ret}?(x\ \mathbf{from}\ \pi).P]$$

$$\mathit{Server} = q[\,\mathit{srv}?(y\ \mathbf{from}\ *).y!\langle v\rangle]$$

# Provenance Usage Example

## Client/Server

| | |
|---|---|
| *srv* | Name of server |
| *ret* | Name of return channel on which server returns answer |

$$\begin{aligned} \mathit{Client} &= p[\, \mathit{srv}!\langle \mathit{ret} \rangle] \quad \| \quad p[\, \mathit{ret}?(x \,\textbf{from}\, \pi).P] \\ \mathit{Server} &= q[\, \mathit{srv}?(y \,\textbf{from}\, *).y!\langle v \rangle] \end{aligned}$$

# Provenance Usage Example

## Client/Server

| | |
|---|---|
| *srv* | Name of server |
| *ret* | Name of return channel on which server returns answer |

$$\begin{aligned} \mathit{Client} &= p[\, \mathit{srv}!\langle \mathit{ret} \rangle] \quad \| \quad p[\, \mathit{ret}?(x\ \textbf{from}\ \pi).P] \\ \mathit{Server} &= q[\, \mathit{srv}?(y\ \textbf{from}\ *).y!\langle v \rangle] \end{aligned}$$

# Provenance Usage Example

## Client/Server

| | |
|---|---|
| *srv* | Name of server |
| *ret* | Name of return channel on which server returns answer |

$$Client = p[\, srv : \kappa^1_{srv} \,!\langle\, ret : \epsilon \rangle] \quad \| \quad p[\, ret : \epsilon \,?(x \;\mathbf{from}\; \pi).P]$$

$$Server = q[\, srv : \kappa^2_{srv} \,?(y \;\mathbf{from}\; *).y!\langle v : \kappa_v \rangle]$$

# Provenance Usage Example

## Client/Server

| | |
|---|---|
| *srv* | Name of server |
| *ret* | Name of return channel on which server returns answer |

$$Client = p[\,srv : \kappa^{1}_{srv}\,!\langle\, ret : \epsilon\rangle] \quad \| \quad p[\,ret : \epsilon\,?(x\ \mathbf{from}\ \pi).P]$$

$$Server = q[\,srv : \kappa^{2}_{srv}\,?(y\ \mathbf{from}\ *).y!\langle v : \kappa_v\rangle]$$

# Provenance Usage Example

## Client/Server

| | |
|---|---|
| *srv* | Name of server |
| *ret* | Name of return channel on which server returns answer |

$$\begin{aligned} \mathit{Client} &= p[\,\mathit{srv}\!:\!\kappa^1_{\mathit{srv}}\,!\langle\,\mathit{ret}\!:\!\epsilon\rangle] \quad \| \quad p[\,\mathit{ret}\!:\!\epsilon\,?(x\ \mathbf{from}\ \pi).P] \\ \mathit{Server} &= q[\,\mathit{srv}\!:\!\kappa^2_{\mathit{srv}}\,?(y\ \mathbf{from}\ *).y!\langle v\!:\!\kappa_v\rangle] \end{aligned}$$

$$\pi = \mathbf{snd}(q, \mathbf{rcv}(q, \mathbf{snd}(p, \epsilon))); *$$

# Outline

Motivation

Proposed solution

Provenance Correctness

Conclusion

# Provenance Correctness

## Correctness Intuition

- provenance attached to values records **history related** to that **value**.
- provenance of a value is correct if it describes a **partial history** which corresponds to the **total history** of events.

# Provenance Correctness

## History represented by Logs

| | | |
|---|---|---|
| $\phi ::= \emptyset \mid \rho; \phi$ | | logs |
| $\rho ::= \textbf{rcv}(p) \mid \textbf{snd}(p)$ | | log actions |

# Provenance Correctness

## History represented by Logs

$$\phi ::= \emptyset \quad | \quad \rho;\phi \qquad \text{logs}$$

$$\rho ::= \textbf{rcv}(p) \quad | \quad \textbf{snd}(p) \qquad \text{log actions}$$

## Sub-Log Comparison

$$\text{cmp1}\,\frac{}{\emptyset \le \phi} \qquad \text{cmp2}\,\frac{\phi \le \phi'}{\rho;\phi \le \rho;\phi'} \qquad \text{cmp3}\,\frac{\phi \le \phi'}{\phi \le \rho;\phi'}$$

# Provenance Correctness

## Monitored Systems

$$\phi \triangleright p[a : \kappa_a ! v : \kappa_v] \parallel Q \quad \longrightarrow_{\text{mon}} \quad \textbf{snd}(p); \phi \triangleright a\langle v : \textbf{snd}(p, \kappa_a); \kappa_v \rangle \parallel Q$$

# Provenance Correctness

## Monitored Systems

$$\phi \triangleright p[a:\kappa_a!v:\kappa_v] \parallel Q \quad \longrightarrow_{\text{mon}} \quad \mathbf{snd}(p);\phi \triangleright a\langle v:\mathbf{snd}(p,\kappa_a);\kappa_v\rangle \parallel Q$$

**Erasure Function:**

$$|-| : \textsc{MonitoredSys} \rightarrow \textsc{Sys}$$

# Provenance Correctness

## Monitored Systems

$$\phi \triangleright p[a : \kappa_a ! v : \kappa_v] \parallel Q \quad \longrightarrow_{\text{mon}} \quad \textbf{snd}(p); \phi \triangleright a\langle v : \textbf{snd}(p, \kappa_a); \kappa_v \rangle \parallel Q$$

**Erasure Function:**

$$| - | : \textsc{MonitoredSys} \rightarrow \textsc{Sys}$$

**Lemma**

$$M \longrightarrow_{\text{mon}} M' \quad \text{implies} \quad |M| \longrightarrow |M'|$$

# Provenance Correctness

## Partial Log Extraction

$$\mathbf{pLog} : \kappa \rightarrow \mathbb{P}(\phi)$$

$$\begin{aligned}
\mathbf{pLog}(\epsilon) &= \emptyset \\
\mathbf{pLog}(\mathbf{rcv}(p,\kappa);\kappa') &= \mathbf{rcv}(p);\mathbf{pLogV}(\kappa') \cup \mathbf{pLog}(\kappa) \\
\mathbf{pLog}(\mathbf{snd}(p,\kappa);\kappa') &= \mathbf{snd}(p);\mathbf{pLogV}(\kappa') \cup \mathbf{pLog}(\kappa)
\end{aligned}$$

$$\begin{aligned}
\mathbf{pLogV}(\epsilon) &= \emptyset \\
\mathbf{pLogV}(\mathbf{rcv}(p,\kappa);\kappa') &= \mathbf{rcv}(p);\mathbf{pLogV}(\kappa') \\
\mathbf{pLogV}(\mathbf{snd}(p,\kappa);\kappa') &= \mathbf{snd}(p);\mathbf{pLogV}(\kappa')
\end{aligned}$$

# Provenance Correctness

**Definition**

$M$ has correct provenance iff $\forall \phi \in \mathbf{pLog}(\mathbf{prov}(M))$ we have $\phi \leq \mathbf{log}(M)$.

**Theorem**

$M$ has correct provenance and $M \longrightarrow_{\mathrm{mon}} M'$ implies $M'$ has correct provenance.

# Outline

Motivation

Proposed solution

Provenance Correctness

Conclusion

# Conclusions

- Designed a provenance based calculus for distributed computing.
- Proposed a two-tier system for provenance tracking and usage.
- Defined provenance correctness
- Proved provenance correctness for our provenance tracking semantics.

# Conclusions

Thank You... Questions?