Privacy Enforcement and Analysis for Functional Active Objects

Florian Kammüller

Middlesex University London and TU Berlin

Data Privacy Management

Athens, 23. September 2010

Motivation and goals

- Language analysis with interactive theorem provers (HOL) “Killer-Application” (Java, C)
- We develop new language ASP$_{\text{fun}}$ in Isabelle/HOL: calculus of functional, active objects, distributed, plus typing

$\Longrightarrow$ Explore language based security for distributed active objects;

$\Longrightarrow$ Enforce and analyse privacy by flexible parameterization (currying)

$\Longrightarrow$ Long-term goal: Language based assembly kit for distributed security (LB-MAKS)

Overview

$\mathrm{ASP}_{\mathrm{fun}}$ – Asynchronous Sequential Processes – functional

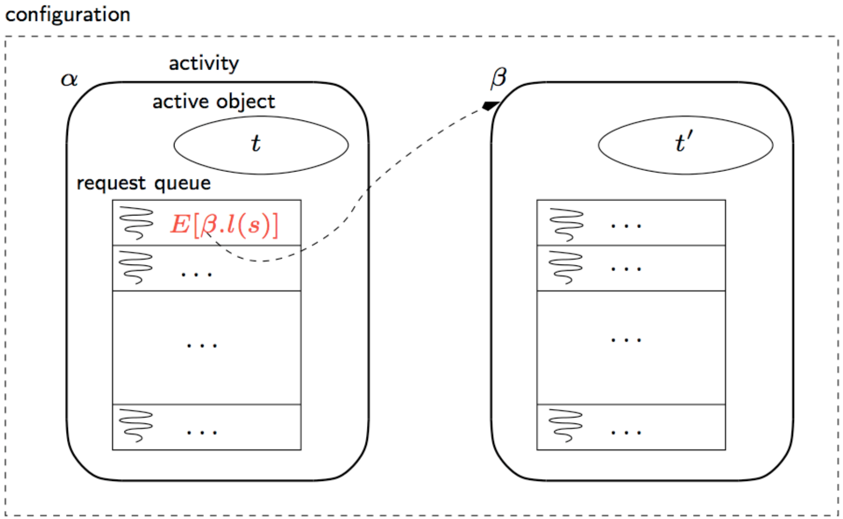
- ProActive (Inria/ActiveEON): Java API for active objects

- New calculus $\mathrm{ASP}_{\mathrm{fun}}$ for ProActive
- Asynchronous communication with *Futures*
  - Futures are *promises* to results of method calls
  - Futures enable asynchronous communication

$\Rightarrow$ $\mathrm{ASP}_{\mathrm{fun}}$ avoids deadlocks when accessing futures

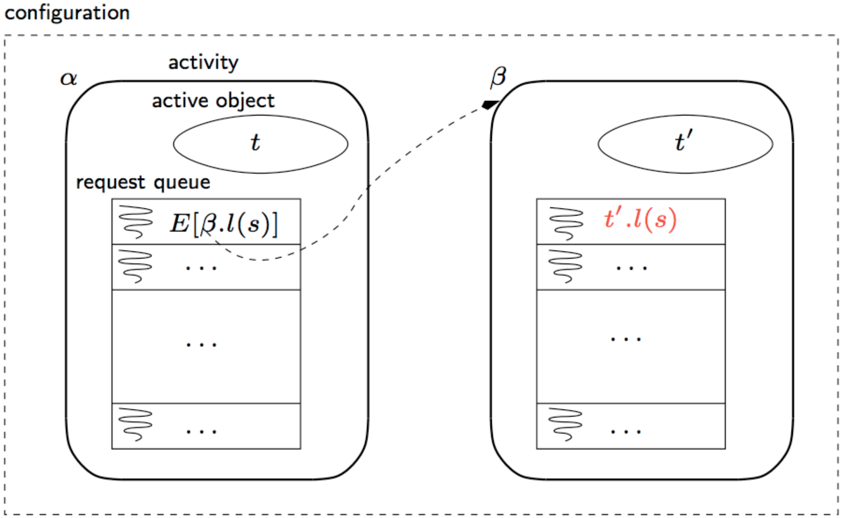
ASP$_{fun}$

ASP$_{fun}$: at a glance



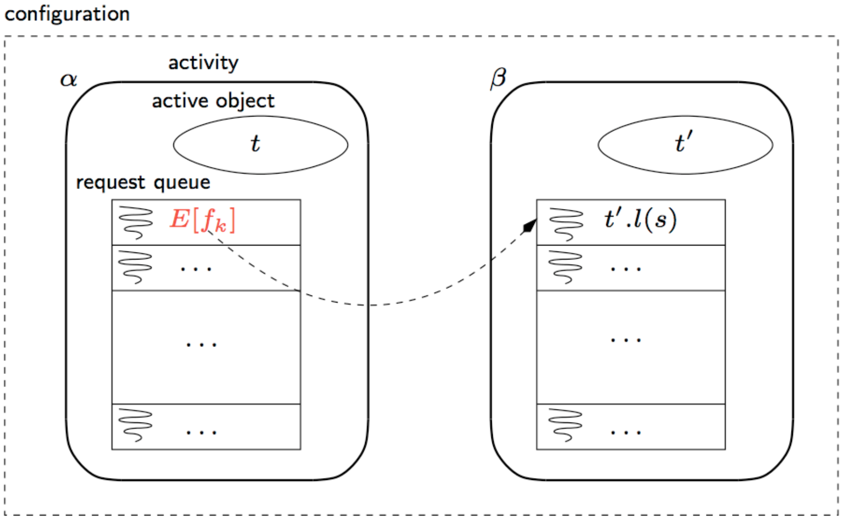
ASP$_{fun}$

ASP$_{fun}$: at a glance



ASP$_{fun}$

ASP$_{fun}$: at a glance



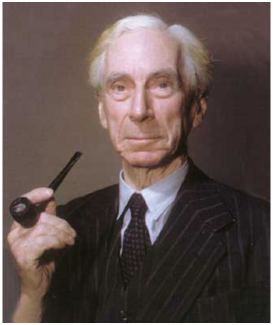
Informal semantics of ASP$_{\text{fun}}$

Local (ς-calculus) and *parallel* (configuration) semantics

- LOCAL: reduction $\rightarrow_\varsigma$ of ς-calculus.
- ACTIVE: *Active*(t) creates a new activity $\alpha[\emptyset, t]$ for new name α, empty request queue, and with t as active object.
- REQUEST: *method call* $\beta.l$ creates new future f_k in future-list of activity β.
- REPLY: *returns result*, i.e. replaces future f_k by referenced result term s (possibly not fully evaluated).
- UPDATE-AO: *activity update* creates a copy of activity and updates active object of copy – original remains the same (*immutable*).

Language development in Isabelle/HOL

- Isabelle/HOL: interactive theorem prover for HOL
- Generic theorem prover
- Formalization of arbitrary object logics
- Interactive proof, tactic support
- Notation close to paper style



- We completely formalized syntax, semantics, and type system of ASP$_{fun}$, and proved language properties.
- Proof of type safety for ASP$_{fun}$: preservation and progress (deadlock freedom)

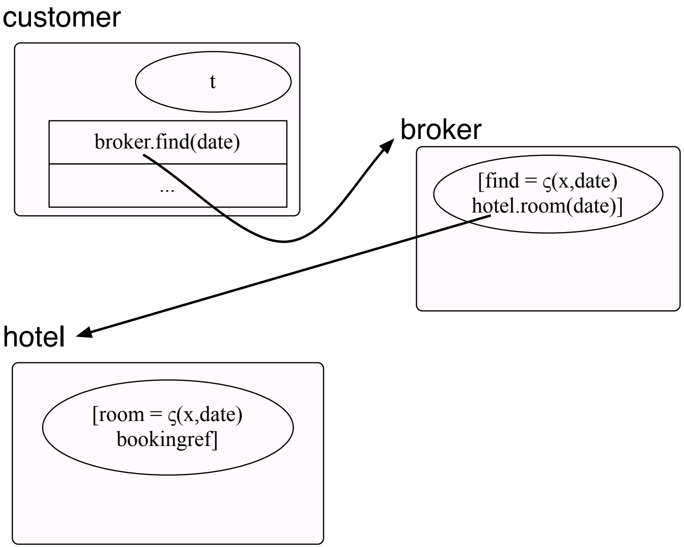
Example: service broker

Customer reserves a hotel using a *broker*

$$\begin{array}{l} \mathsf{customer}[f_0 \mapsto \mathsf{broker.find(date)}, t] \\ \parallel \mathsf{broker}[\varnothing, [\mathsf{find} = \varsigma(x, \mathit{date})\mathsf{hotel.room}(\mathit{date}), \ldots]] \\ \parallel \mathsf{hotel}[\varnothing, [\mathsf{room} = \varsigma(x, \mathit{date})\mathsf{bookingref}, \ldots] \end{array}$$

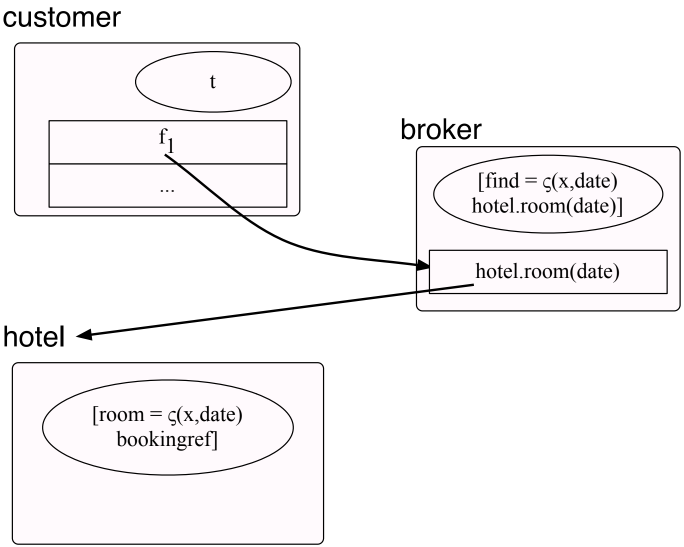
Example: service broker

Customer reserves a hotel using a *broker*



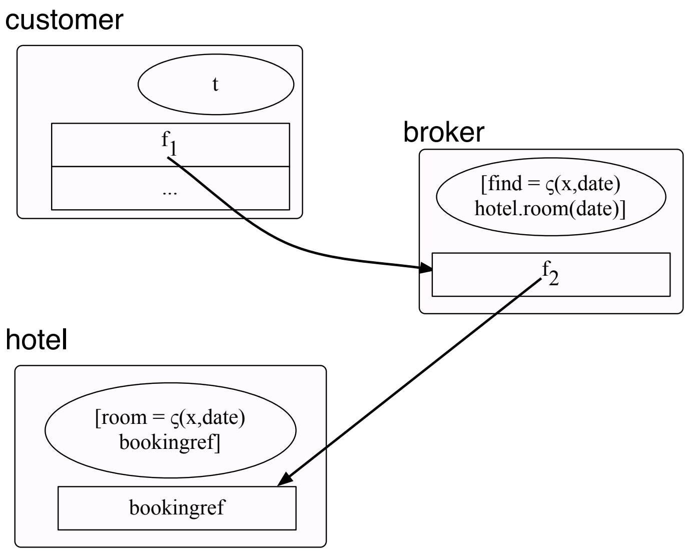
Example: service broker

Customer reserves a hotel using a *broker*


customer
t
f_1
...
broker
[find = ς(x,date) hotel.room(date)]
hotel.room(date)
hotel
[room = ς(x,date) bookingref]

Example: service broker

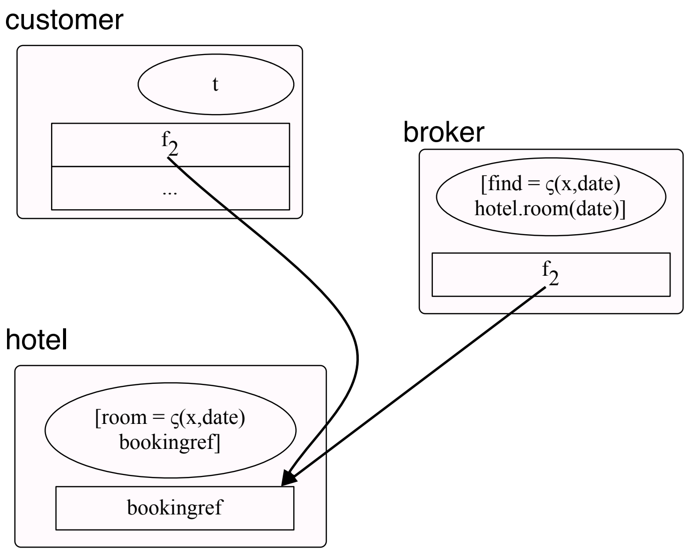
Customer reserves a hotel using a *broker*



customer
t
f1
...
broker
[find = ς(x,date) hotel.room(date)]
f2
hotel
[room = ς(x,date) bookingref]
bookingref

Example: service broker

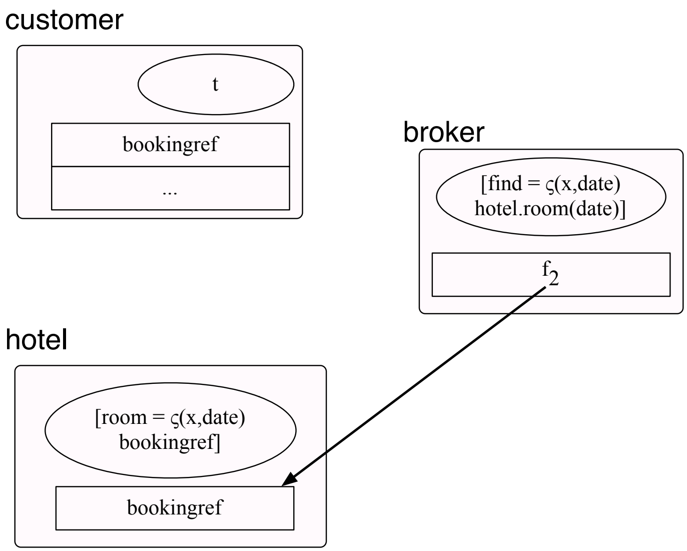
Customer reserves a hotel using a *broker*



customer
t
f2
...
broker
[find = ς(x,date) hotel.room(date)]
f2
hotel
[room = ς(x,date) bookingref]
bookingref

Example: service broker

Customer reserves a hotel using a *broker*



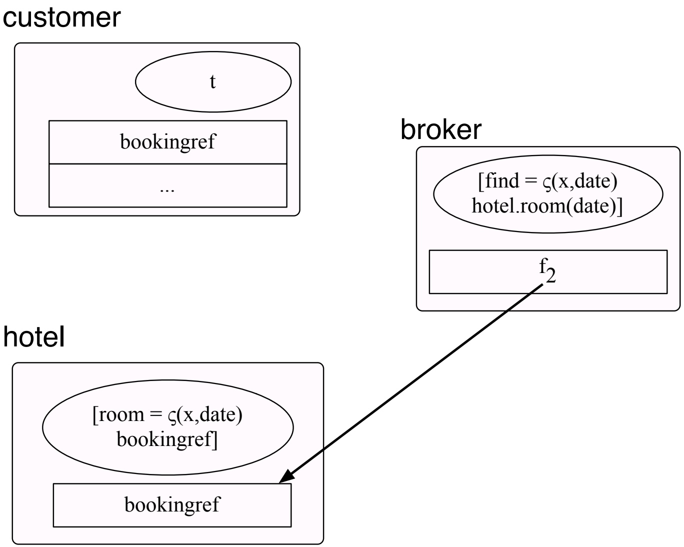
customer
t
bookingref
...
broker
[find = ς(x,date) hotel.room(date)]
f2
hotel
[room = ς(x,date) bookingref]
bookingref

Observations

- Service broker has a private domain of hotel addresses, negotiates and only replies selected hotel or bookingref to customer.
- Client receives bookingref using f_2 without viewing details of the hotel nor others from broker's domain.
- It would be nice if the reply bookingref would also be private to customer, but ...

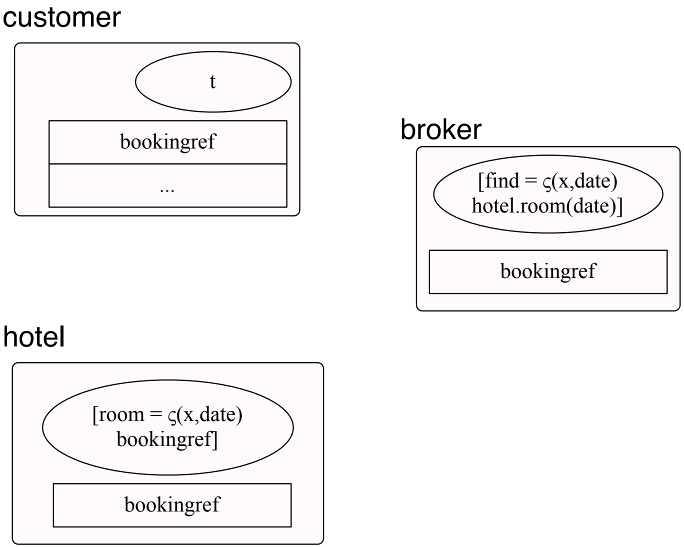
Example: service broker

. . . broker has also f_2 and can thus get customer's bookingref.


customer
t
bookingref
...
broker
[find = ς(x,date) hotel.room(date)]
f2
hotel
[room = ς(x,date) bookingref]
bookingref

Example: service broker

. . . broker has also f_2 and can thus get customer's bookingref.


customer
t
bookingref
...
broker
[find = ς(x,date) hotel.room(date)]
bookingref
hotel
[room = ς(x,date) bookingref]
bookingref

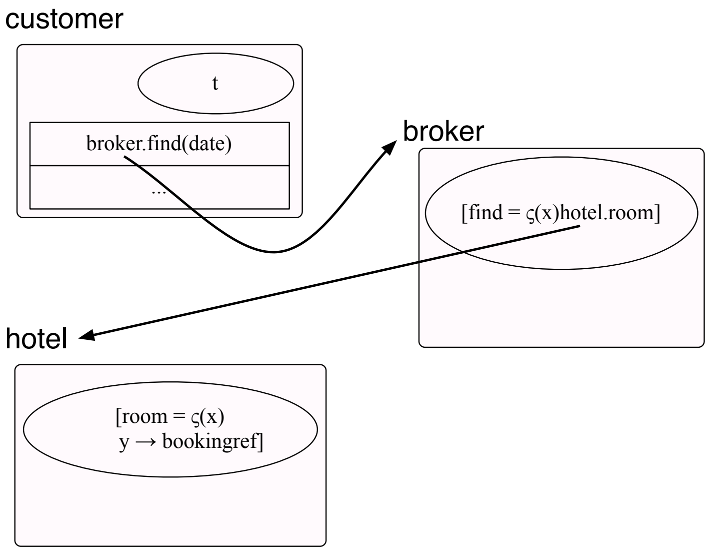
Function Replies for Privacy

- Idea: avoid communication of private data

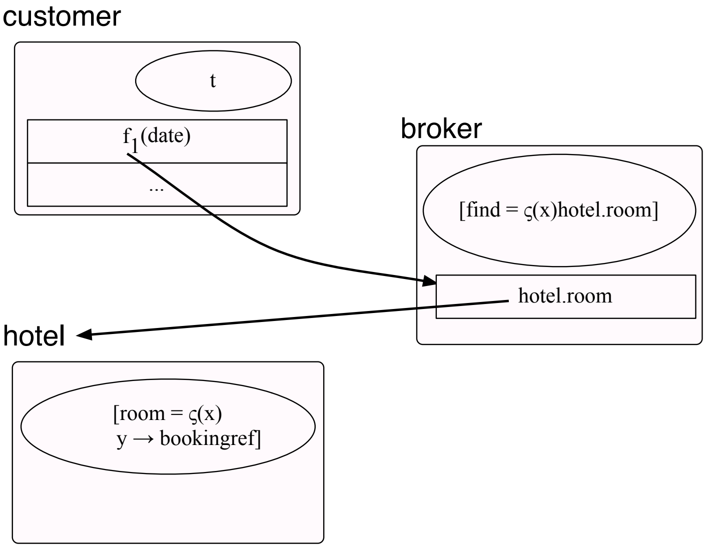
$\Longrightarrow$ Use the reply of functions in ASP$_{\text{fun}}$

- Example broker with private parameter *date*
  - Client requests booking *without disclosing parameter date*
  - Hotel returns function $y \rightarrow$ bookingref to client
  - Client calculates his individual bookingref by supplying parameter *date* afterwards

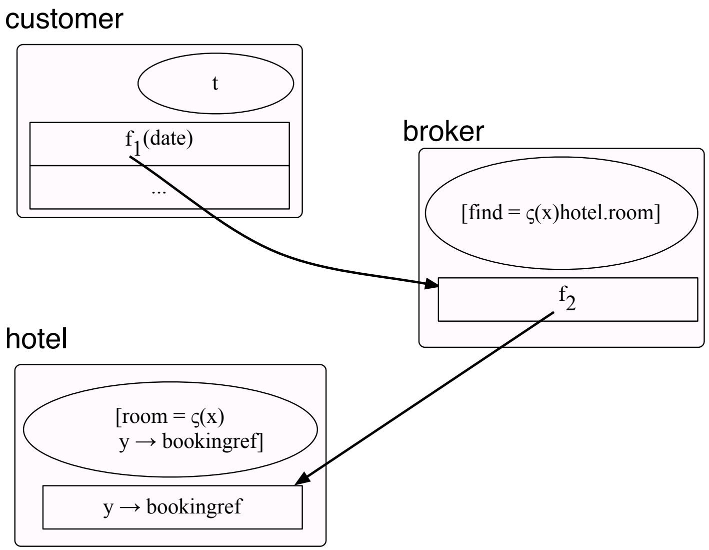
Private Hotel Reservation



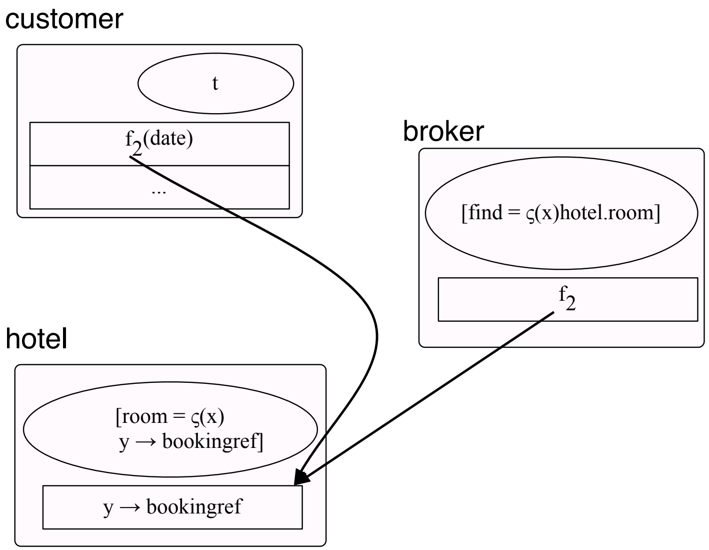
Private Hotel Reservation



Private Hotel Reservation



Private Hotel Reservation



Stock Taking

- Two versions of broker example:
    1. broker preserves his privacy (futures)
    2. customer can keep his data private as well (currying)
- Private booking 2. uses currying, so is data secure?

$\Longrightarrow$ Implementation of ASP$_{\text{fun}}$ in Erlang supports currying

- Can we provide analysis support for privacy?

$\Longrightarrow$ (Language Based) Information Flow Control for ASP$_{\text{fun}}$

Contribution

- Formal definitions for ASP_{fun} of:
  - *Hiding* of object labels Δ in object o: $o \setminus \Delta$
  - *Noninterference* (formal definition of information flow security) based on hiding

$\Longrightarrow$ Currying is a means for privacy enforcement

$\Longrightarrow$ Prove formally “No information flow to public” in curried broker example using formal definitions

but Tedious analysis of all possible program evaluations

$\Longrightarrow$ Define type systems for efficient security verification

Conclusions

- ASPEN$_{DFG}$: Security analysis of distributed active objects
    - Co-development of a new language ASP$_{fun}$ in Isabelle/HOL
    - Isabelle/HOL: type safe and deadlock free
    - Erlang interpreter prototype of ASP$_{fun}$
- Broker example illustrates privacy enforcement
- Information flow control to analyse security: expensive analysis (type systems)
- Outlook: LB-MAKS for ASP$_{fun}$: compositionality of security properties

Current papers

[1] L. Henrio, F. Kammüller. A Mechanized Model of the Theory of Objects. Formal Methods for Open Object-Based Distributed Systems, FMOODS'07. LNCS **4468**, 2007.

[2] F. Kammüller. Formalizing Noninterference for Bytecode in Coq. Formal Aspects of Computing: **20**(3):259–275. Springer, 2008.

[3] L. Henrio and F. Kammüller. Functional Active Objects: Typing and Formalisation. Foundations of Coordination Languages and System Architectures, FOCLASA'09. Satellite to ICALP'09. ENTCS, 2009. Also invited to *Science of Computer Programming*.

[4] F. Kammüller and R. Kammüller. Enhancing Privacy Implementations of Database Enquiries. The Fourth International Conference on Internet Monitoring and Protection. IEEE, 2009. Also *Int. Journal on Advances in Security* **2**(2 + 3), 2009.

[5] F. Kammüller. Using Functional Active Objects to Enforce Privacy. 5th Conf. on Network Architectures and Information Systems Security. Menton, 2010.

[6] A. Fleck and F. Kammüller. Implementing Privacy with Erlang Active Objects Int. Conference on Internet Monitoring and Protection. 2010.

[7] F. Kammüller. Privacy Enforcement and Analysis for Functional Active Objects. 5th International Workshop on Data Privacy Management, DPM2010, co-located with ESORICS 2010.