

# Middlesex University Research Repository

An open access repository of

Middlesex University research

<http://eprints.mdx.ac.uk>

KammueLLer, Florian ORCID logo ORCID: <https://orcid.org/0000-0001-5839-5488> (2011) Privacy enforcement and analysis for functional active objects. Lecture Notes in Computer Science, 6514 . pp. 93-107. ISSN 0302-9743 [Article] (doi:10.1007/978-3-642-19348-4\_8)

UNSPECIFIED

This version is available at: <https://eprints.mdx.ac.uk/6516/>

## Copyright:

Middlesex University Research Repository makes the University's research available electronically.

Copyright and moral rights to this work are retained by the author and/or other copyright owners unless otherwise stated. The work is supplied on the understanding that any use for commercial gain is strictly forbidden. A copy may be downloaded for personal, non-commercial, research or study without prior permission and without charge.

Works, including theses and research projects, may not be reproduced in any format or medium, or extensive quotations taken from them, or their content changed in any way, without first obtaining permission in writing from the copyright holder(s). They may not be sold or exploited commercially in any format or medium without the prior written permission of the copyright holder(s).

Full bibliographic details must be given when referring to, or quoting from full items including the author's name, the title of the work, publication details where relevant (place, publisher, date), pagination, and for theses or dissertations the awarding institution, the degree type awarded, and the date of the award.

If you believe that any material held in the repository infringes copyright law, please contact the Repository Team at Middlesex University via the following email address:

[eprints@mdx.ac.uk](mailto:eprints@mdx.ac.uk)

The item will be removed from the repository while any claim is being investigated.

See also repository copyright: re-use policy: <http://eprints.mdx.ac.uk/policies.html#copy>

# 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
- ⇒ Explore language based security for distributed active  
objects;
- ⇒ Enforce and analyse privacy by flexible parameterization  
(currying)
- ⇒ Long-term goal: Language based assembly kit for  
distributed security (LB-MAKS)

# Overview

- 1  $ASP_{fun}$
- 2 Example for  $ASP_{fun}$ : Service Triangle
- 3 Privacy Enforcement and Analysis

# ASP<sub>fun</sub> – Asynchronous Sequential Processes – functional

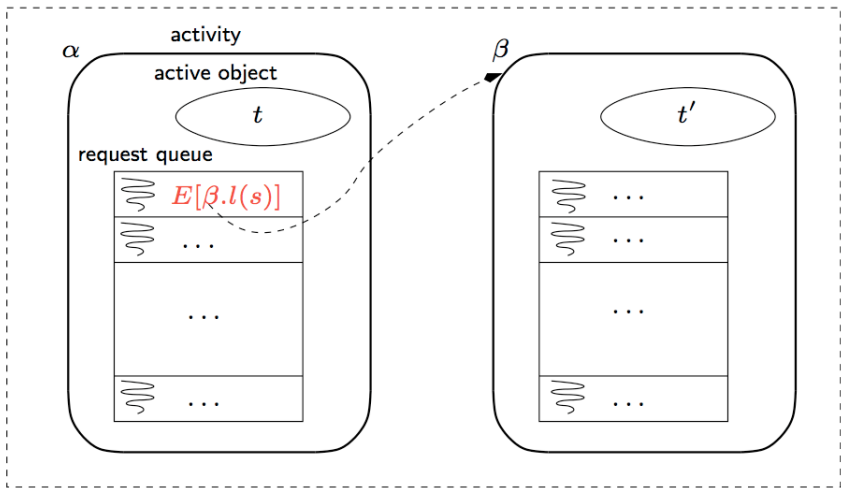
- ProActive (Inria/ActiveEON): Java API for active objects



- New calculus ASP<sub>fun</sub> for ProActive
  - Asynchronous communication with *Futures*
    - Futures are *promises* to results of method calls
    - Futures enable asynchronous communication
- ⇒ ASP<sub>fun</sub> avoids deadlocks when accessing futures

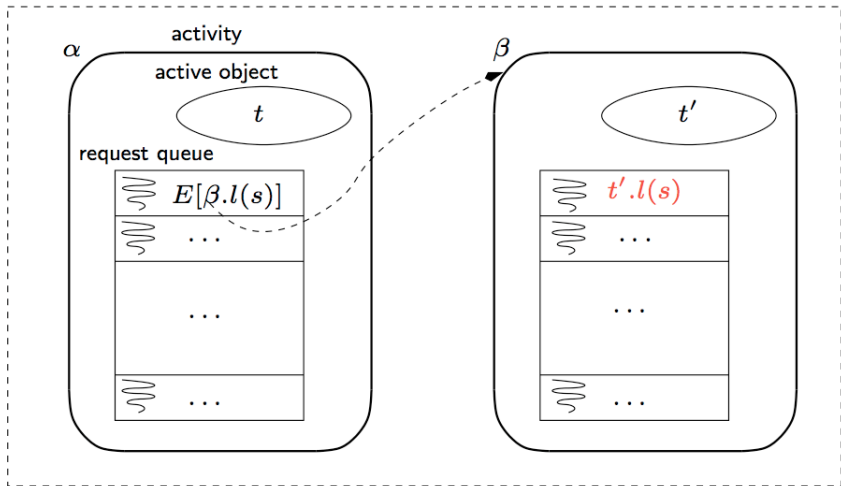
ASP<sub>fun</sub>: at a glance

configuration



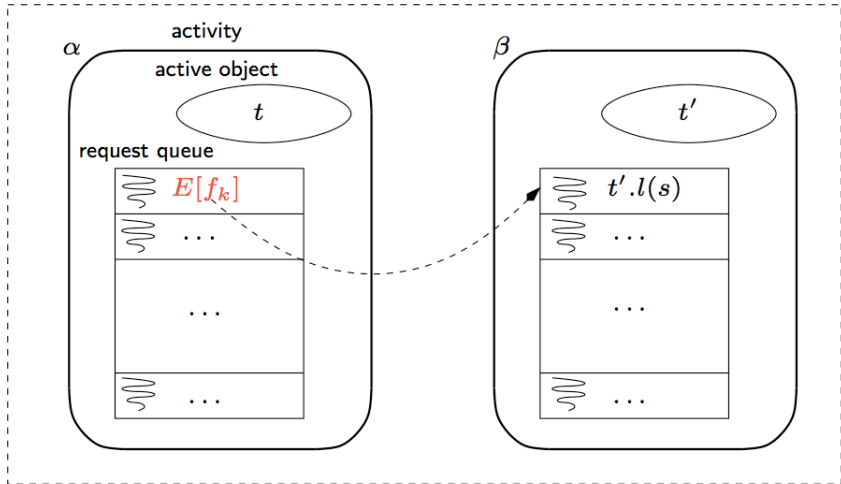
ASP<sub>fun</sub>: at a glance

configuration



ASP<sub>fun</sub>: at a glance

configuration





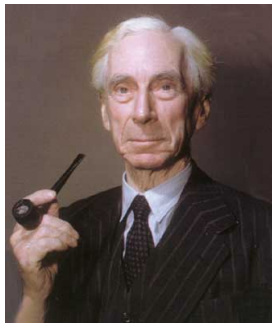
# Informal semantics of ASP<sub>fun</sub>

*Local* ( $\zeta$ -calculus) and *parallel* (configuration) semantics

- **LOCAL**: reduction  $\rightarrow_{\zeta}$  of  $\zeta$ -calculus.
- **ACTIVE**: *Active*( $t$ ) creates a new activity  $\alpha[\emptyset, t]$  for new name  $\alpha$ , empty request queue, and with  $t$  as active object.
- **REQUEST**: *method call*  $\beta.l$  creates new future  $f_k$  in future-list of activity  $\beta$ .
- **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

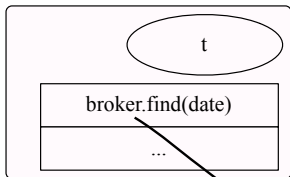
Customer reserves a hotel using a *broker*

```
customer[ $f_0 \mapsto$  broker.find(date),  $t$ ]  
|| broker[ $\emptyset$ , [find =  $\varsigma(x, date)$ hotel.room(date), ...]]  
|| hotel[ $\emptyset$ , [room =  $\varsigma(x, date)$ bookingref, ...]]
```

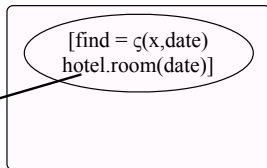
## Example: service broker

Customer reserves a hotel using a *broker*

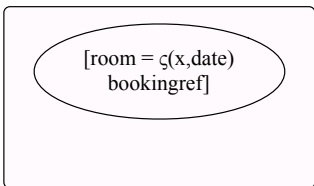
customer



broker



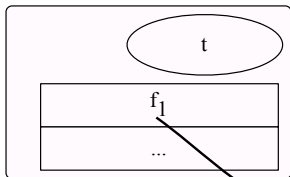
hotel



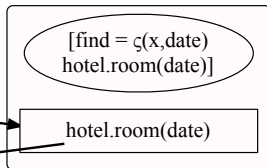
## Example: service broker

Customer reserves a hotel using a *broker*

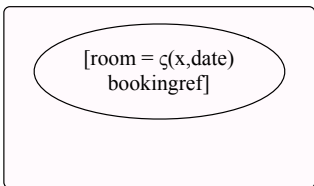
customer



broker



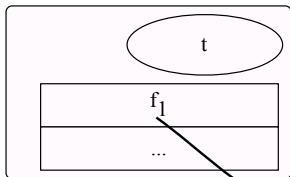
hotel



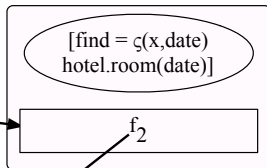
## Example: service broker

Customer reserves a hotel using a *broker*

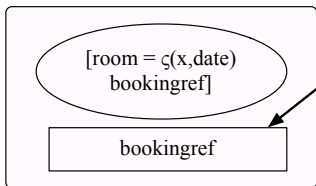
customer



broker



hotel



f<sub>1</sub>

f<sub>2</sub>

[room = ζ(x,date)  
bookingref]

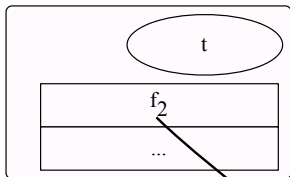
bookingref

[find = ζ(x,date)  
hotel.room(date)]

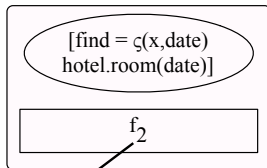
## Example: service broker

Customer reserves a hotel using a *broker*

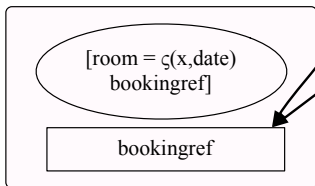
customer



broker



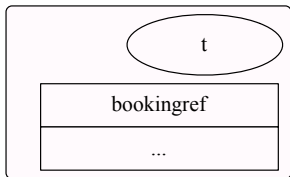
hotel



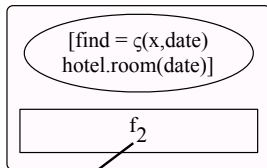
## Example: service broker

Customer reserves a hotel using a *broker*

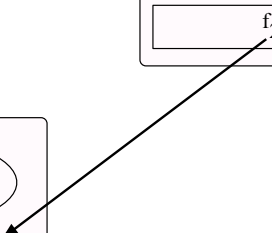
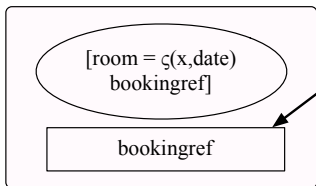
customer



broker



hotel





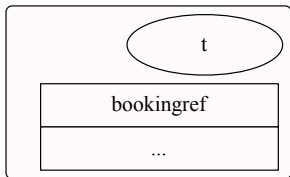
# Observations

- 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 . . .

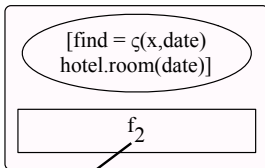
## Example: service broker

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

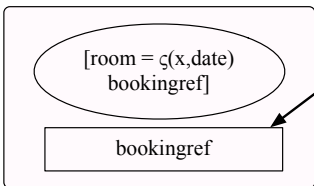
customer



broker



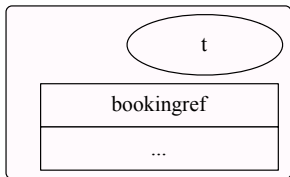
hotel



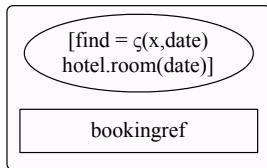
## Example: service broker

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

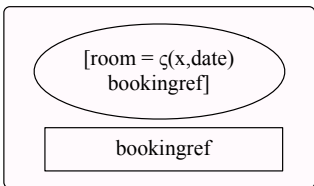
customer



broker



hotel

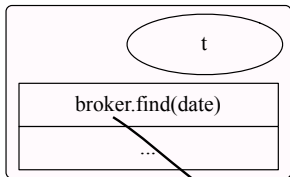


# Function Replies for Privacy

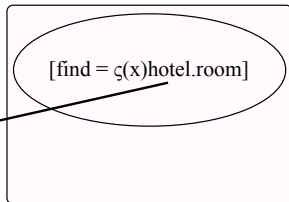
- Idea: avoid communication of private data
- ⇒ Use the **reply of functions** in  $ASP_{\text{fun}}$
- Example broker with **private** parameter *date*
  - Client requests booking *without disclosing parameter date*
  - Hotel returns function  $y \rightarrow \text{bookingref}$  to client
  - Client calculates his individual **bookingref** by supplying parameter *date* afterwards

# Private Hotel Reservation

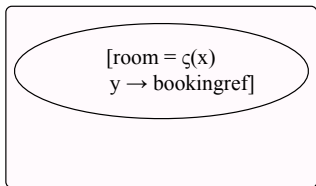
customer



broker

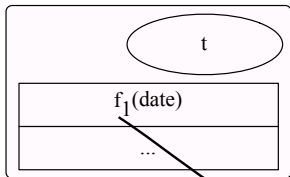


hotel

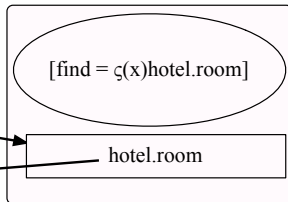


# Private Hotel Reservation

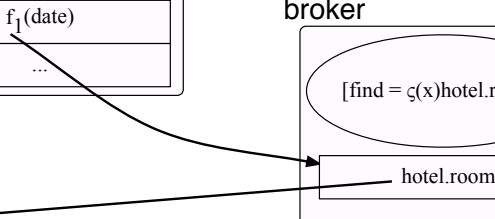
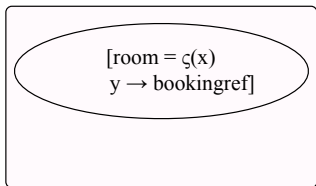
customer



broker

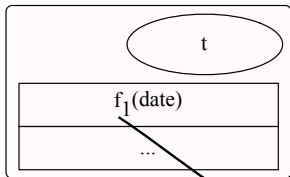


hotel

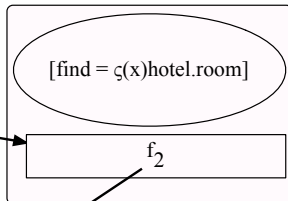


# Private Hotel Reservation

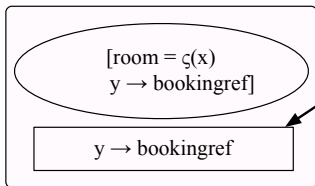
customer



broker

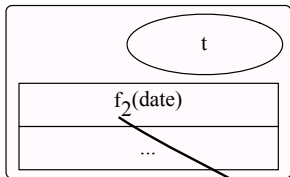


hotel

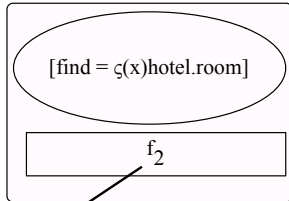


# Private Hotel Reservation

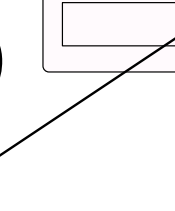
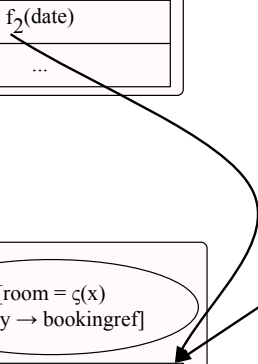
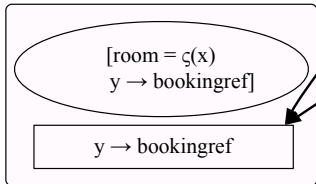
customer



broker



hotel





# 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?
- ⇒ Implementation of  $ASP_{fun}$  in Erlang supports currying
- Can we provide analysis support for privacy?
- ⇒ (Language Based) Information Flow Control for  $ASP_{fun}$

# Contribution

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

⇒ Currying is a means for privacy enforcement

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

**but** Tedious analysis of all possible program evaluations

⇒ Define type systems for efficient security verification

# Conclusions

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