



**Why
Object-Oriented**

(this page intentionally left blank)

YOO

(Why Object-Orientation)

Report of a satellite workshop of Concur 2000
held at the Pennsylvania State University, State College, 26 August 2000

Arend Rensink (organiser)
University of Twente
<http://www.cs.utwente.nl/~rensink>

September 13, 2000

Contents

1	Introduction	2
2	Programme and participants	3
3	Conclusion	6
	U. Montanari, <i>Concurrency, Higher Order and Types. What Else?</i> (slides)	8
	F.S. de Boer, <i>Components in OO</i> (slides)	16
	U. Nestmann, <i>How Much of Objects do Processes Need?</i> (position paper)	26
	Discussion summary sheets	28

1 Introduction

This report contains a summary of a satellite workshop of Concur 2000. Both the main conference and the workshop took place at the Pennsylvania State University at State College, Pennsylvania, in August 2000. Relevant internet links are:

CONCUR: <http://www.cse.psu.edu/concur2000/>

YOO: <http://trese.cs.utwente.nl/Workshops/yoo/>

Rather than the “mini-conference” format, which is traditionally almost exclusively used for Concur satellites, YOO was intended as a forum for discussion—a format in fact well known from other conferences, such as ECOOP. Thus, it represented something of an experiment. To reduce the “risk” inherent in the experiment, it was decided to keep the workshop to half a day, i.e., only Saturday morning was planned. We will comment on this aspect in detail in the conclusions; in summary, however, the experiment was a reasonable success, but half a day is definitely too short.

The following paragraphs contain excerpts from the workshop’s call for position statements (see <http://trese.cs.utwente.nl/Workshops/yoo/call.html>). In Section 2 we report on the actual event, and Section 3 presents some tentative conclusions, not about the subject matter but rather about the organisational aspects of the workshop. These “lessons learnt” are intended to benefit future editions or variations of this (kind of) workshop.

Topic of the workshop. Object-orientation is pervading computer science practice and therefore naturally attracts computer science researchers. For some it may be merely a new buzzword to attach to what they do best; to others the concepts of object-orientation (OO) pose new and relevant research questions. Yet others may be unfamiliar with the principles that lie behind OO -indeed even among those that consider themselves familiar there is no consensus about these principles.

Within the theory of concurrency, OO is the subject of much debate. Many seem to feel that even though OO may have its uses in other fields of computer science, it does not bring new problems to or shed new light upon their favourite subject area. Others find in the concepts of OO precisely the elements that they think of as most important for a successful design or verification method.

Aim of the workshop. This workshop aims to bring together OO sceptics and enthusiasts in identifying (or at least openly discussing) ways in which OO has relevance to the theory of concurrency. Questions that can be addressed are:

- What (if any) are the advantages of OO in the theory of concurrency?
- What difference does OO make to specific fields, such as specification, testing, model checking or compositional system design (to name just a few)?
- What are the research questions that most urgently need to be answered?

Specifically, it is not the aim of the workshop to present theoretical novelties that purport to encapsulate, obviate or improve upon OO; rather, the interest should be directed towards the motivation for investigating such matters: either YOO? or Y, OO!

Form of the workshop. The workshop will consist of two invited 30-minute presentations, by

- Prof. Dr. Ugo Montanari, University of Pisa

- Dr. Frank de Boer, University of Utrecht

and a larger number of 10-minute position statements, followed by a free-for-all discussion. Position statements for the 10-minute presentations are hereby invited (see below); they will be made available both over the internet and in the form of a technical report of the University of Twente, distributed during the workshop. A summary of the discussion will be published at a later date, in a form yet to be decided. No formal proceedings are planned.

The number of participants is envisaged at no more than 25, so as to ensure a lively discussion. All participants should submit a position statements; the 10-minute presentations will be solicited on the basis of the submissions.

2 Programme and participants

Below we summarise the phases of the workshop programme. A list of participants is included in Table 1.

9:00–10:30 Invited presentations

The workshop started with two invited 30-minute presentations:

- Ugo Montanari (University of Pisa): *Concurrency, Higher Order and Types. What Else?*

This presentation painted a general picture of object-orientation as “a remarkable combination of features” which may be investigated from any of a number of different points of view, in particular *programming languages and methodology, semantics and type theory, operational concurrency* and *observation and abstraction*. Combinations of two or more of these viewpoints can be seen to give rise to distinct areas of research.

See Page 8 for copies of the slides.

- Frank de Boer (University of Utrecht): *Components in OO*

This presentation was more focused on a specific technical issue: the appropriate notion of observable behaviour of an object, given that its run-time context consists of a collection of other objects. It was argued that information hiding is insufficient to capture the observable behaviour of an object.

See Page 16 for copies of the slides.

The two presentations nicely complemented each other; both gave rise to a number of questions from the audience, the ensuing discussion lasted approximately 15 minutes.

11:00–11:45 Position statements

All participants (see Table 1 for a complete list) briefly introduced themselves and declared their interest in and experience with object-orientation. In addition, Uwe Nestmann gave a position statement (see Page 26) and Silvia Crafa presented some motivations for her research in object-orientation.

Table 1: List of participants

Paolo Baldan	University of Pisa Dipartimento di Informatica, Corso Italia, 40, 56100 Pisa, Italy	<code>baldan@di.unipi.it</code>
Martin Berger	Dept. of Computing, QMW Mile End Rd., London E1 4NS, United Kingdom	<code>m.berger@doc.ic.ac.uk</code>
Frank de Boer	Universiteit Utrecht Institute of Information and Computing Sciences, Padualaan 14, De Uithof, 3584 CH Utrecht	<code>frankb@cs.uu.nl</code>
Silvia Crafa	Università Di Venezia Ca Foscari Via Torino 155, Venezia Mestre 30172, Italy	<code>silvia@dsi.unive.it</code>
Matthew Davis	Pennsylvania State University 220 Pond Lab, University Park, PA 16802	<code>msd4@psu.edu</code>
Rogier van Eijk	Universiteit Utrecht Institute of Info and Computing Sciences, P.O. Box 80089, 3508 TB Utrecht, The Netherlands	<code>rogier@cs.uu.nl</code>
Oltea Herescu	Pennsylvania State University Dept. of Comp. Sci and Engr, 311 Pond Lab, University Park, PA 16802	<code>herescu@cse.psu.edu</code>
Ugo Montanari	University of Pisa Dipartimento di Info, 40 Corso Italia, Pisa I-56100, Italy	<code>ugo@di.unipi.it</code>
Uwe Nestmann	Aalborg University, Brics Fredrik Bajes Vej 7E, Aalborg 9220, Denmark	<code>uwe@cs.auc.dk</code>
Catuscia Palamidessi	Pennsylvania State University 325 Pond Lab, University Park, PA 16802	<code>catuscia@cse.psu.edu</code>
Elaine Pimentel	Pennsylvania State University 200 Pond Laboratory, University Park, PA 16802	<code>pimentel@cse.psu.edu</code>
Ernesto Posse	McGill University School of Computer Science, 3480 Univ. St., Rm. 318, Montreal – Quebec, Canada	<code>e posse@cs.mcgill.ca</code>
Antonio Ravara	Technical University of Lisbon Dep. Mathematics, IST, Av. Rovisco Pais, Lisbon 1049-001, Portugal	<code>amar@math.ist.utl.pt</code>
Arend Rensink	University of Twente PO Box 217, Enschede 7500 AE, The Netherlands	<code>rensink@cs.utwente.nl</code>
Vladimiro Sassone	Università di Catania Viale Andrea Doria 6, Dip. Mat Informatica, Catania 95125, Italy	<code>vs@dmi.unict.it</code>
Eijiro Sumii	University of Pennsylvania 200 South 33rd St., Philadelphia, PA 19104	<code>sumii@saul.cis.upenn.edu</code>
Frank Valencia	University of Aarhus BRICS, Int'l Ph.D School, Ny Munkegade Bldg. 540, Aarhus DK 8000, Denmark	<code>fvalenci@brics.dk</code>

11:45–12:30 Discussion groups

With the background of the invited presentations and position statements, as well as their own research activities and experiences, the participants were asked to discuss issues of their own choice regarding the relations between object-orientation and concurrency theory, guided by the following questions:

What Object-Orientation? What are the essential features of OO?

- Data abstraction/encapsulation
- Object identity/aliasing
- Classes = types
- Inheritance/subtyping/reuse
- ...

Concurrency theory and object behaviour.

- Is there life after the π -calculus?
(do the existing process calculi cover all relevant behavioural aspects?)
- What are the limits of types?
(type = behaviour? inferable? checkable?)
- Is **self** anything special?
(does self-referencing require a special treatment?)
- Are process calculi the way?
(what about actors/concurrent rewriting/...)

Concurrency theory and the UML: What do we do about the UML?

- Ignore it
- Curse it
- Use it
- Formalise it
- Improve it

Object-Orientation and concurrency theory: where can we make ourselves most useful?

- Semantics for UML/OOPLs
- Temporal logic & model checking
- Verification & testing
- Real-time & stochastic modelling

For the purpose of the discussion, the participants were divided into groups of five, which retreated to surroundings in which they could discuss the issues more easily and informally among themselves. Each group was asked to summarise their findings on a single overhead sheet.

13:00–13:30 Plenary discussion

After the discussion session, the groups reconvened and presented their findings; see Pages 28–30. These were then discussed in the plenary session.

3 Conclusion

From the point of view of the organisation, the workshop was satisfactory: the participants were indeed actively discussing the subject matter, and afterwards the consensus seemed to be that this had been useful and enlightening for them. The following conclusions therefore seem justified:

- The principal format of the workshop functioned as intended;
- The invited presentations did an excellent job of raising the relevant issues;
- The division into small discussion groups, reconvening at a later time, had the (intended) effect of giving everyone the chance to speak freely;
- The total number of participants was in the right interval (15–25).

Nevertheless, the organisation could be improved on the following points:

- In preparation for the workshop, it was intended that all registrees submit a position paper (see above). In reality, only a single position paper had been submitted before the workshop (see Page 26). This fact may be attributed to one or more of the following reasons:
 1. It was not clear to potential contributors what kind of submission was asked for;
 2. The subject matter was too unfamiliar for most participants to venture an opinion about;
 3. The submission date was too far ahead of the workshop;
 4. No one wants to spend time writing anything that will not actually be published.

These arguments can be countered in some degree:

1. With experience and examples, it will become clearer what is being asked
2. The informal nature of position papers should be stressed
3. Submissions could be solicited from those who have already registered, instead of requiring them to be finished at the time of registration
4. A formal publication contrasts with the setup of the workshop; the only possibility is inclusion in a technical report (e.g., see Page 26).

The main functions of the position papers are (*i*) to force the participants to think about the subject matter before the workshop, and (*ii*) to generate points of discussion during the workshops (through corresponding presentations). Both purposes can possibly also be achieved by setting up an email discussion before the workshop —although experience has shown that it takes a lot of time investment from the organiser to get such a discussion going.

- The time available for discussion during the workshop was clearly too short. Since the slots for the other parts of the programme were fine, it follows that half a day is too little time for a workshop along these lines. An entire day, on the other hand, may be too long, but part of the afternoon should definitely be available.
- The position statements serve two functions: to introduce the participants, and to give personal viewpoints. The first of these two can also take place immediately at the start of the programme, before the invited presentations: by giving all participant exactly 1 minute to introduce themselves, a more open atmosphere may be created right from the beginning. The second function did not work quite as well as intended, since only a single position paper had been submitted before the workshop; see above.

- The conclusion of the workshop was slightly unsatisfactory because there was no well-defined end product. In a way this is as it should be, because informal discussions rarely give rise to clear-cut conclusions, and any pressure to arrive at such a conclusion could harm the open exchange of opinions. One might consider, however, drawing up a list of opinions at the end of the day, collecting from each participant a statement on (for instance):
 - What did you learn today?
 - What was the most valuable new insight gained today?
 - What do you consider the most/least important aspect of the subject area?
 - What do you consider the hottest open question in the subject area?

Acknowledgements. Many people deserve thanks for making this workshop a reality: Catuscia Palamidessi for taking superb care of all local arrangements; Uwe Nestmann for encouragement and support; the invited speakers, Ugo Montanari and Frank de Boer, for interesting contributions that set the stage for the discussion session, and finally, all the participants for actively taking part in the workshop.

2000 International Conference on Concurrency Theory (Concur 2000)
WORKSHOP on "Why Object-Orientation", State College, Friday, August 25, 2000

Concurrency, Higher Order and Types. What Else?

Transparencies by Ugo Montanari
Dipartimento di Informatica, Pisa

Personal View of Object Orientation

Concurrency enthusiast

but no direct experience with object oriented research
(differently than Gérard Boudol)

OO highly relevant to real-life programming

Also deep and technical type theoretical foundations: e.g.

extensible objects with subtyping: FCT'99

Bono & Bugliesi

Boudol & Dal-Zilio

However WAN programming emphasizes concurrent, distributed aspects
need of new models/results/approaches

Concurrency, Higher Order and Types?

TOSCA (Italian acronym)

project supported by the Italian ministry of research
two years

*Rocco De Nicola, Mariangiola Dezani, Roberto Gorrieri,
Eugenio Moggi, Furio Honsell, Benedetto Intrigila,
Cosimo Laneve, Andrea Maggiolo, Ugo Montanari (coord.),
Nicoletta Sabadini, Vladimiro Sassone, Marisa Venturini, etc.*

tries to reconcile the three concepts
for models, programming, verification

Object orientation not directly addressed
but several theoretical issues relevant for OO actually studied

What is Peculiar About Object Orientation?

Remarkable combination of features
what is needed for (today?) programming

No precise definition of concepts
Emphasized by *Peter Wegner* in comparison with
5th generation *committed-choice* logic approach

Foundations mainly on type theory

Rather different than the analytic approach typical of *CONCUR* community
build sequences of calculi/models/languages/logics
with the minimal features needed to make a point
SOS, compositional, abstract, but moderate type theory
(but see *object calculi* by *Abadi & Cardelli* and *Fisher & Mitchell*)

Different Views of Object Orientation

Programming Languages and Methodology

Language design: *C++*, *Java*

Reuse: parameterization, polymorphism, information hiding, inheritance

Efficient implementation issues

Static and dynamic type checking

Flow and semantic analysis (e.g. pointer & escape analysis)

UML (unified modeling language) syntax and semantics

Software architectures

Design patterns

Aspect-oriented programming

Software engineering vs. *UML & OO*, e.g. *Rational Rose*

Semantics and Type Theory

Lambda calculus and higher-order functional programming

Denotational semantics

Domain theory

Curry-Howard formulas-as-types correspondence

Recursive types

Polymorphic types

Dependent types

Existential types (ADT)

Subtyping (containment/conversion)

Proof checking as typing

Signatures as proof development environments

Logic frameworks: *COQ*, *LF*, *ALF*

Operational Concurrency

Concurrency & distribution
Reactivity & interactivity
Synchronization & locking
Mobility
Openness
Finite & infinite state verification

Observation and Abstraction

Final semantics
Coalgebras
Semantics vs. logics adequacy

Combining Two Issues: Some New Scenarios

Programming & Semantics
POPL focus
Concurrency & Observation
CONCUR focus
Programming & Concurrency
WAN programming
Programming & Observation
coordination
Semantics & Concurrency
mobility & higher order processes
Semantics & Observation
compositional abstract semantics via algebras/coalgebras

Combining Sharp Focus with Wide Applicability

General frameworks with a natural and simple theory
Modeling several existing formalisms and application scenarios
Deriving new general results and innovative formalisms
Generic implementations for animation and rapid prototyping

Well-known examples: Petri nets, process algebras, constraint programming

More recent, developing frameworks

- rewriting logic
- action calculi & control structures
- graph transformations
- tile logic

New Challenges

mechanisms to support mobility of code and computations
effective infrastructures to support coordination and control
of dynamically loaded software modules

More generally

an abstract semantic framework to formalize the
model of computation of internet applications

Aims

to provide the formal basis to discuss and motivate controversial
design/implementation issues
to state and certify properties in a rigorous way

Synchronization in a Coordination Framework

Coordination: large number of software components

black boxes

independently programmed in different languages,

may change their configuration during execution

Most of the activity asynchronous, but some applications, e.g.

computer supported collaborative work

decision making among multiple partners

need complex computations by all partners on shared data before commit

Challenge: design the language primitives

provide a high level model for specification, validation and verification

implement via asynchronous protocols

Two Contributions

Zero-safe Petri nets (*Bruni and Montanari, Info and Co, Jan.2000*)

A zero-safe net in a stable state when certain places are empty

Step sequences from stable states to stable states

through non-stable states are transactions

They correspond to transitions of an abstract net.

Tile Logic (<http://www.di.unipi.it/~ugo/tiles.html>)

Extends *structured operational semantics (SOS)* by Gordon Plotkin

rewriting logic by Jose Meseguer

Tiles are inference rules which can be combined

horizontally to build transactions

vertically to build ordinary computations.

Finite State Verification with Name Creation (I)

Finite state verification: threads of control independent of the actual data
 automaton constructed
 minimized
 checked for properties
 applied to hardware and protocols.

In WAN programming we have creation of new names
 name extrusion in mobile calculi
 nonces generated during secure sessions
 dynamic process locations
 causal dependencies of events

Finite State Verification with Name Creation (II)

Control often independent from data
 redirecting streams
 connecting and disconnecting users
 transferring processes from an ambient to another
 resolving conflicts
 performing security checks

- ◇ Dynamic allocation & deallocation of names
 - ◇ expressive logical frameworks
 - ◇ efficient algorithms
- are needed for finite state checking of security
 and other global and local properties

History Dependent (HD) Automata *(Montanari & Pistore)*

Able to allocate and garbage collect names.

Verification of behavioral properties

dynamic network connectivity

locality of resources and processes

causality among events

Front end for the π -calculus *(CAV'98)*

Foundations

Higher-order abstract syntax for terms with binders *(LICS'99)*

Coalgebras with algebraic structure *(Turi & Plotkin; Corradini, Heckel & Montanari)*

HD-automata as coalgebras on permutation algebras *(Montanari & Pistore, MFCS 2000)*

Components in OO

F.S. de Boer

CONCUR 2000/YOO

Objects and classes

Objects

- **internal state** (data abstraction)
- **methods** (operations)
- **identity**

Classes **generic behavioral descriptions**

CONCUR 2000/YOO

1

Histories

Communication records

Method invocation

$(\underbrace{\alpha}_{\text{receiver}}, \underbrace{m}_{\text{method}}, \underbrace{\bar{\beta}}_{\text{parameters}}, \underbrace{\gamma}_{\text{result}})$

Request

$(\text{req}, \underbrace{m}_{\text{method}}, \underbrace{\bar{\beta}}_{\text{parameters}})$

Result

$(\text{res}, \underbrace{m}_{\text{method}}, \underbrace{\beta}_{\text{result}})$

CONCUR 2000/YOO

3

Observable behavior: Full abstraction

Context = collection of classes $P[\text{interface } C]$

Full abstraction

$C1 \simeq C2 \Leftrightarrow \forall P. \mathcal{O}(P[C1]) = \mathcal{O}(P[C2])$

Observables

$\mathcal{O}(P) \in \text{initial global state} \Rightarrow \text{set of global states}$

CONCUR 2000/YOO

2

Object creation

Main problem

Hiding of local objects

Conclusion

Objects do not have a clearly defined interface

CONCUR 2000/YOO

4

Components

Internal = Collection of Classes

External = (Mobile) Channels

CONCUR 2000/YOO

5

Channels

Channels: (unbounded) FIFO buffers

x !e output
 channel
 x ?y input
 channel

x contains a **reference** to a FIFO buffer

CONCUR 2000/YOO 7

The basic concepts

- **Objects**: basic computational entities
- **Classes**: generic behavioral descriptions
- **Channels**: basic external communication media
- **Components**: collection of classes

Main feature

Communication via channels is **anonymous**:
 Components do not have an **identity**.

CONCUR 2000/YOO 6

The language $\sigma\pi$

Basic actions

$x := e$

assignment

$\underbrace{x := \text{new}(C)}_{\text{chan}}$ component creation

$\underbrace{x!e}_{\text{chan}}$

output

$\underbrace{x?y}_{\text{chan}}$

input

CONCUR 2000/YOO

8

The local transition system

$\underbrace{\langle S, s \rangle}_{\text{configuration}}$
control state

Assignment

$\langle x := e, s \rangle \xrightarrow{\tau} \langle \text{nil}, s[s(e)/x] \rangle$

Component creation

$\langle x := \text{new}(C), s \rangle \xrightarrow{v} \langle \text{nil}, s[v/x] \rangle$
for any channel identity v

CONCUR 2000/YOO

9

The global transition system

$$\underbrace{\langle \lambda, \dots, (S, s), \dots \rangle}_{\text{configuration}}, \underbrace{\sigma}_{\text{channels}}$$

$\sigma \in \text{ChanId} \Rightarrow \text{Val}^*$ records

- the **existing** channels
- the **state** of each existing channel

CONCUR 2000/YOO

11

Communication

Output

$$\langle x!e, s \rangle \xrightarrow{v!w} \langle \text{nil}, s \rangle$$

where $s(x) = c$ and $s(e) = w$.

Input

$$\langle x?y, s \rangle \xrightarrow{v?w} \langle \text{nil}, s[w/y] \rangle$$

where $s(x) = v$ and w **arbitrary**.

CONCUR 2000/YOO

10

Internal computation and object creation

Internal computation

$$\frac{\langle S, s \rangle \xrightarrow{\tau} \langle S', s' \rangle}{\langle X \uplus \{ \langle S, s \rangle \}, \sigma \rangle \longrightarrow \langle X \uplus \{ \langle S', s' \rangle \}, \sigma \rangle}$$

Component creation

If v does not exist in σ then

$$\frac{\langle S, s \rangle \xrightarrow{v} \langle S', s' \rangle}{\langle X \uplus \{ \langle S, s \rangle \}, \sigma \rangle \longrightarrow \langle X \uplus \{ \langle S', s' \rangle, \langle S'', s'' \rangle \}, \sigma \rangle}$$

where

- σ' results from σ by adding v
- S'' is the corresponding statement
- $s''(\text{pipe}) = v$

CONCUR 2000/YOO

12

Communication

Asynchronous Input

If $\sigma(v) = w \cdot u$ for some value u

$$\frac{\langle S, s \rangle \xrightarrow{v?u} \langle S', s' \rangle}{\langle X \uplus \{ \langle S, s \rangle \}, \sigma \rangle \longrightarrow \langle X \uplus \{ \langle S', s' \rangle \}, \sigma' \rangle}$$

where $\sigma' = \sigma[w/v]$.

Asynchronous Output

$$\frac{\langle S, s \rangle \xrightarrow{v!w} \langle S', s' \rangle}{\langle X \uplus \{ \langle S, s \rangle \}, \sigma \rangle \longrightarrow \langle X \uplus \{ \langle S', s' \rangle \}, \sigma' \rangle}$$

where $\sigma' = \sigma[\sigma(v) \cdot w/v]$.

CONCUR 2000/YOO

13

Example

Mobility

```

ROOT ≡
x1: = new(SEND) ; x2: = new(REC) ; x1!x2

SEND ≡
y: = 0 ; pipe?x ; do x!y ; y: = y + 1 od

REC ≡
do pipe?y ; ... od

```

CONCUR 2000/YOO

14

The interface class

Interface of a component:

```

interface Interface
protected static CHAN x1, ..., xn

```

Each class *C* of a component extends Interface:

```
C extends Interface ←
```

$$S \begin{cases} x?y & y \notin \{x_1, \dots, x_n\} \\ x!e & \\ x: = \text{new } x \notin \{x_1, \dots, x_n\} & \end{cases}$$

CONCUR 2000/YOO

15

Adapter of a component

Adapter implements Interface \Leftarrow

$$S \left\{ \begin{array}{l} \text{root: = new}(C) \\ x_i: = \text{new} \\ \text{pipe?}x_i \\ \text{pipe!}x_i \end{array} \right. \begin{array}{l} \text{activation} \\ \text{initialization} \\ \text{importing} \\ \text{exporting} \end{array}$$

CONCUR 2000/YOO

16

Managers

Manager coordinates $(C_1, \dots, C_n) \Leftarrow S$

$$S \left\{ \begin{array}{l} x: = \text{new}(C_i) \\ x!y, x?y \end{array} \right. \begin{array}{l} \text{activation} \\ \text{reconfiguration} \end{array}$$

CONCUR 2000/YOO

17



Future work

Correctness of a coordination module:

Interface specification

- { input channels assumptions
- output channels commitments

correctness of a coordination module=

mutual consistency of the interface specification of its components.

Design methodology in UML

How Much of Objects do Processes Need? Or, Y use SOAP?

Uwe Nestmann, BRICS Aalborg, Denmark

YOO?, June 2000

Abstract

This position statement starts from the point of *Y,OO!* that OO, i.e., object-orientation, is good also in the context of concurrency. In this realm, we discuss briefly whether or not it is a good idea to use SOAP, i.e., to provide the “Semantics of Objects As Processes”, on which we have gained some experience over the past years. One of the building blocks of our work has been an enhanced “object-oriented process calculus” as a convenient target language for SOAP. The aim of this statement is to generate some discussion on whether SOAP is useful and on how object-oriented a process calculus needs to be, either to deserve to be called OO or to be useful as a SOAP target.

SOAP

This acronym describes a way to give semantics of OO languages (the source) by translating the syntax of the language into the syntax of a process calculus (the target) [5]. The aim of such an approach is to use the theory and intuitions of the target calculus to reason about terms, i.e., programs in the source language. We have recently experimented with some non-trivial OOL (OO language) derived from Obliq [1] as a subset including concurrency, cloning, and aliasing, which is enough to simulate a particular style of mobile objects within the language [4]. In order to prove, using an algebraic equation within the object language, that this style of object migration can be carried out in a safe manner [3], we used a semantics by translation into an enhanced *Local π -calculus* [2].

Among the many things learned from this non-trivial exercise of using SOAP it clearly proved useful to develop the semantics both at the implementation level of the target calculus as well as at the level of a direct structural operational semantics within the source language. In short, one could say that the semantics by translation enabled us to discover and explore how things do or do not work, while the direct semantics allowed us to afterwards explain the phenomena not only to “outsiders”, but also to ourselves from a more global point of view. (This seems to hold also for proofs.)

Object-Oriented Process Calculi

The target calculus that we used in [3] is an asynchronous π -calculus, (1) where all names are *local*, i.e., of which only the output capability can be transmitted in communication, (2) where messages can be structured using *labels* from a disjoint set of globally known names, and (3) enhanced with the syntactic category of *keys*, which are the only items that can be used in matching constructs, and which can not be used otherwise apart from being exchanged in communication. We consider this calculus to be object-oriented, because

- the locality of names captures the fact that the receivers of a name (aka: the object’s reference) are statically fixed upon the name’s declaration, while the send capability of the name—due to name-passing in the π -calculus—can be freely exchanged with potential clients of the object;

- the sending of labeled values is reminiscent of calling methods by using an object, one of its method labels, and (a list of) actual parameters.
- the matching on keys allows us to express the concept of self-infliction as used in Obliq, i.e., in languages that provide a means of protection and serialization based on the notion of an object's self.

Even calculi with fewer primitives have been coined “concurrent object calculus”, e.g., Vasconcelos' TyCO [6], a calculus of typed concurrent objects, in which *all* messages exchanged are labeled.

So, to repeat the title of this statement: “How Much of Objects do Processes Need?” There is already quite some machinery at hand in basic name-passing process calculi that captures certain OO features. In our opinion, the addition of more object-oriented flavor can be restricted to the addition of primitives as mentioned above. If not, what's missing? How could one measure whether a process calculus is object-oriented enough? Which OO features have to be representable within the process calculus, and according to which correctness criterion? For example, should there also be a requirement to express advanced synchronization primitives in order to cope with the inheritance anomaly, as experienced in concurrent OOLs?

References

- [1] L. Cardelli. A language with distributed scope. *Computing Systems*, 8(1):27–59, 1995. Short version in *Proceedings of POPL '95*. A preliminary version appeared as Report 122, Digital Systems Research, June 1994.
- [2] M. Merro. *Local π : A Model for Concurrent and Distributed Programming Languages*. PhD thesis, Ecole des Mines, France, 2000. To appear.
- [3] M. Merro, J. Kleist, and U. Nestmann. Local π -calculus at work: Mobile objects as mobile processes. In *Proceedings of TCS 2000*, LNCS. IFIP, Aug. 2000. To appear.
- [4] U. Nestmann, H. Hüttel, J. Kleist, and M. Merro. Aliasing models for mobile objects. Accepted for *Journal of Information and Computation*. Available from <http://www.cs.auc.dk/research/FS/ojeblik/>. An extended abstract has appeared as Distinguished Paper in the *Proceedings of EUROPAR '99*, LNCS 1685, 2000.
- [5] U. Nestmann and A. Ravara. Semantics of objects as processes (soap). In A. Moreira and S. Demeyer, editors, *ECOOP '99 Workshop Reader*, volume 1743 of *LNCS*, pages 314–325. Springer, Oct. 1999. An introduction to, and summary of, the 2nd International SOAP-Workshop.
- [6] V. T. Vasconcelos. *A process-calculus approach to typed concurrent objects*. PhD thesis, Keio University, 1994.

- 1) objects vs processes \rightarrow grouping
 Encapsulation
 identity = independence = concurrency
~~inheritance~~ \leftrightarrow reuse
-

- 2) synchronization among COs
 \rightarrow inheritance (even in sequential)
 \rightarrow inheritance anomaly
 \rightarrow mutual vs. self-recursion
-

- 3) probably yes, ... but no ^{intention}
 UML-notation "neutral" by ^{defia}
 (possibly offer various formal interpretations
 for "neutral" UML-syntaxes)
-

- 4) • advertise our work!
 • propose better primitives
 'max high-level

Treatment of classes

- Are classes just types?
- Name equivalence instead of structural equivalence for classes
 - closer to practice of OO
 - easier to deal with

process calculi + type theory →
classes

Summary

- What is OO?
 - identity (self)
 - encapsulated state
 - collection of methods
-
- What about TYPES and INHERITANCE?
- (parametric) polymorphism? PP vs. inheritance?
- Concurrency & UML? Shall UML be taken seriously?
- How can WE be useful?
 - Types
 - Operational semantics
 - Proof techniques