Semantics of Objects As Processes (SOAP)*

Uwe Nestmann[1] and António Ravara[2]

[1] BRICS***, Aalborg University, Denmark
uwe@cs.auc.dk
[2] IST, Technical University of Lisbon, Portugal
amar@math.ist.utl.pt

1. Introduction

One of the most popular programming paradigms today is that of object-oriented programming. With the growing popularity of the language C++ and the advent of Java as the language of choice for the World Wide Web, object-oriented programs have taken center stage. Consequently, the past decade has seen an exponentially increasing interest within the programming language research community for providing a firm semantic basis for object-oriented constructs.

Recently, there has been growing interest in studying the behavioral properties of object-oriented programs using concepts and ideas from the world of concurrent process calculi, in particular calculi with some notion of mobility. Not only do such calculi, as the well-known π-calculus by Milner, Parrow and Walker [26], have features like references and scoping in common with object-oriented languages; they also provide one with a rich vocabulary of reasoning techniques firmly grounded in structural operational semantics and static typing.

The process calculus view has therefore proven to be advantageous in many ways for semantics and verification issues. On the one hand, the use of encodings of object-oriented languages into existing typed mobile process calculi enables formal reasoning about the correctness of programs; on the other hand, using standard techniques from concurrency theory in the setting of calculi for objects may help in reasoning about objects, e.g. by finding appropriate and mathematically tractable notions of behavioral equivalences. Encodings may also help clarify the overlap and differences of objects and processes, and suggest how to integrate them best in languages with both.

The aim of the SOAP workshops is to bring together researchers working mainly in this area, but in related fields as well, where other process models or calculi are used as a basis for the semantics of objects.

Historical Remarks

The origin of the SOAP workshops may be found in the early *Fränkische OOrientierungstage 1993*, organized by Uwe Nestmann and Terry Stroup, where a

* http://www.cs.auc.dk/soap99/
*** Basic Research in Computer Science, Centre of the Danish National Research Foundation

A. Moreira and S. Demeyer (Eds.): ECOOP'99 Workshops, LNCS 1743, pp. 314–325, 1999.

programming tutorial by Benjamin Pierce [35] on objects in the π-calculus-based programming language Pict [36] was followed by an invited workshop on the semantics of objects as processes. The first open workshop, SOAP '98, organized by Hans Hüttel and Uwe Nestmann, then took place as a satellite event to ICALP'98 in Aalborg, Denmark. The proceedings can be downloaded from http://www.brics.dk/NS/98/Ref/BRICS-NS-98-Ref/, more information is accessible through the workshop web page at http://www.cs.auc.dk/soap99/.

Brief Summary of SOAP'99

For the '99 edition of SOAP, taking place as a satellite workshop of ECOOP '99, among nine submitted abstracts five were recommended by the programme committee (Hans Hüttel, Josva Kleist, Uwe Nestmann, and António Ravara) based on a formal refereeing process, and are summarized below. The proceedings can be downloaded via http://www.brics.dk/NS/99/Ref/BRICS-NS-99-Ref/, also accessible through http://www.cs.auc.dk/soap99/.

We would like to thank the organizers of ECOOP '99, in particular Ana Maria Moreira, for helping us logistically to set up the SOAP workshop, we thank BRICS, in particular Uffe Engberg Nielsen, for the publication of these proceedings, and we thank Massimo Merro and Silvano Dal-Zilio for their assistance in the refereeing process.

According to the specific topics of the accepted contributions, the workshop programme is composed of two complementary thematic building blocks.

The first block was addressing the motto 'objects *as* processes' literally in that objects are represented as a derived concept within a framework of processes; we welcomed Oscar Nierstrasz, Markus Lumpe, and Jean-Guy Schneider as invited speakers to present the work they have been accomplishing in this area—starting out from a mobile process calculus—and to let us learn about their conclusions. This session was rounded up by a verification approach using a temporal logic as a target setting for, in this case, UML-style objects.

The second building block, divided into a session on behavioral subtyping and another one on behavioral typing, is seen as an adaptation of the process-theoretic viewpoint to some object-oriented framework. While the typed λ-calculus is a firm ground to study typing for object-oriented languages, the typing of concurrent objects poses particular problems due to synchronization constraints. A static notion of typing is not powerful enough to capture dynamic properties of objects' behavior, like non-uniform service availability. Concurrency theory inspires dynamic notions of typing and subtyping, and this block of SOAP'99 exemplified the state of the art in the field.

The rest of this report contains brief summaries of workshop presentations (where the presenting speaker is indicated by a *), preceded by short overviews of the fields, and followed by concluding remarks and a list of selected references.

2. Objects as Processes

Background: An Annotated Bibliographic Overview

Rather soon after the development of process algebras, basically triggered by pioneering work of Tony Hoare [15] and Robin Milner [25], and enabled by first attempts to capture the semantics of parallel object-oriented programming within the POOL-family [3] of languages, Frits Vaandrager [46] started out to give a first explicit study on the semantics of objects as processes, where he encoded a variant of POOL into the process algebra ACP [4]. An encoding in this context usually simply means a translation of some high-level (object) syntax into some lower-level (process) syntax. An encoding is usually considered 'good', if it is compositional and exhibits some preservation and reflection properties concerning operational and behavioral relations of the languages or calculi involved.

Although some basic principles of concurrent objects could be clearly expressed in Vaandrager's approach by means of concurrent processes, several aspects of object-oriented programming, like the persistent identities of objects and certain inheritance features, were not modeled in a natural way. It needed another landmark invention, the π-calculus [26], to get a more suitable and, as it turns out, quite appropriate process model for objects. David Walker took over and extended Vaandrager's initial work, now targeting at the π-calculus [49]. Moreover, together with Xinxin Liu and Anna Philippou, in a series of papers, they investigated the possibilities of using the process-algebraic semantics for reasoning about concurrent object-oriented programs [22, 23, 24, 32, 33, 34], some of it in order to solve a tricky program transformation problem proposed by Cliff Jones [19, 20]. Related to this line of research, Xiaogang Zhang and John Potter provided some more understanding of class-based object-oriented programs using π-calculus [51]. These developments have not only been of theoretical nature, but gave also rise to implementations that closely follow the pencil-and-paper encodings, like Benjamin Pierce and David Turner exemplified in their Pict compiler [36].

Another strand of research has been opened up by the advent of the object calculus (OC) by Martín Abadi and Luca Cardelli [1], who provide a minimal model of typed objects. Again, the study of encodings has been the main tool for SOAP-investigations. Hans Hüttel and Josva Kleist gave a first encoding of the untyped object calculus into the π-calculus [17]. Davide Sangiorgi gave a different one for the typed object calculus [45], and extended this work together with Josva Kleist to the imperative object calculus [21]. Hans Hüttel, Josva Kleist, Massimo Merro and Uwe Nestmann extended this work to mobile objects [18].

There have also been other approaches to internalize object-oriented notions as primitives within process calculi. Here, we just list Oscar Nierstrasz' higher-order object calculus [31], Kohei Honda and Mario Tokoro's ν-calculus [16], Vasco Vasconcelos' TyCO [47], and Gérard Boudol's Blue Calculus [6]. Two further object calculi can be viewed either as extensions of OC with concurrency, or as extensions of process calculi with objects: one by Paolo Di Blasio and Kathleen Fisher [13] and another one by Andrew Gordon and Paul Hankin [14].

At SOAP '99, we also had presentations on how to represent objects within the framework of Petri Nets (see Section 3). In the remainder of the current section, however, we summarize the three presentations on objects as strongly typed name-passing processes, and one presentation on objects as cTLA agents.

Piccola — A Small Composition Language (Oscar Nierstrasz)

Although object-oriented languages are well-suited to implementing software components, they fail to shine in the construction of component-based applications, largely because object-oriented design tends to obscure a component-based architecture. We propose to tackle this problem by clearly separating component implementation and composition. Piccola is a small "composition language" that embodies the paradigm of "applications = components + scripts." Piccola models components and composition abstractions by means of a unifying foundation of communicating concurrent agents. Flexibility and extensibility are obtained by modeling both interfaces to components and the contexts in which they live by extensible records, or "forms". We illustrate the realization of an architectural style in Piccola and show how external components may be adapted and composed according to the style. We show how separating components from their composition can improve maintainability.

The πL-Calculus A Formal Foundation for Software Composition (Markus Lumpe)

In this talk, we present a formal language for software composition that is based on the π-calculus. More precisely, we present the πL-calculus, a variant of the π-calculus, in which agents communicate by passing extensible, labeled records, or so-called "forms", rather than tuples. This approach makes it much easier to model compositional abstractions than it is possible in the plain π-calculus, since the contents of communications are now independent of positions, agents are more naturally polymorphic since communication forms can be easily extended, and environmental arguments can be passed implicitly. The πL-calculus is developed in three stages: (i) we analyze whether the π-calculus is suitable to model composition abstractions, (ii) driven by the insights we got using the π-calculus, we define a new calculus that has better support for software composition (e.g., provides support for inherently extensible software construction), and (iii), we define a first-order type system with subtype polymorphism that allows us to statically check an agent system in order to prevent the occurrences of runtime errors.

Object Models in the πL-Calculus (Jean-Guy Schneider)

The development of concurrent object-based programming languages has suffered from the lack of any generally accepted formal foundation for defining their semantics, although several formal models have been proposed. Most of these models define objects and object-oriented abstractions as primitives, but they either do not incorporate important features found in object-based programming languages (e.g., they lack inheritance), hard-wire the underlying inheritance model, or integrate concepts in a non-orthogonal way. As an approach to overcome the problems of existing models, we present a (meta-level) framework for object models in the πL-calculus. We show that common object-oriented programming abstractions such as instance variables and methods, different method dispatch strategies as well as class features are most easily modeled when class meta-objects are explicitly reified as first class entities. We illustrate that various concepts which are typically merged (or confused) in object-oriented programming languages can be expressed in a more natural way by making a clear separation between functional elements (i.e. methods) and their compositions (i.e. inheritance). Furthermore, we show that the same concepts can also be applied for modeling mixins, mixin application, and mixin composition.

Composing Object-Oriented Specifications and Verifications with cTLA (Günter Graw*, Peter Herrmann, and Heiko Krumm)

In order to support formally correctness preserving refinement steps of object-oriented system designs, we refer at one hand to the practically well-accepted Unified Modelling Language (UML) and at the other hand to Leslie Lamport's Temporal Logic of Actions (TLA) which supports concise and precise notions of properties of dynamic behaviors and corresponding proof techniques. We apply cTLA which is an extension of TLA and supports the modular definition of process types. Moreover, in cTLA process composition has the character of superposition which facilitates the modular transformation of UML diagrams to corresponding formal cTLA process system definitions and their structured verification. We exemplify transformation and formal verification. Furthermore, we outline the application of this method for the establishment of domain-specific specification frameworks which can directly support the UML-based correct design of OO-systems.

3. Behavioral Typing and Subtyping

Background: An Annotated Bibliographic Overview

Behavioral typing and behavioral subtyping are notions of (respectively) typing and subtyping for concurrent object-oriented programming, which take into

account dynamic aspects of objects' behavior. In the beginning of the 90's, Nierstrasz argued that typing concurrent objects poses particular problems due to the non-uniform methods availability, since by synchronization constraints, the availability of a method depends upon the internal state of the object (which reflects the state of the system) [30]. Therefore, a static notion of typing, like interfaces-as-types, is not powerful enough to capture dynamic properties of the behavior of concurrent objects. Hence, Nierstrasz proposed the use of a regular language as types for active objects, to characterize the traces of menus offered by the objects. He also proposed a notion of subtyping, *request substitutability*, which is based on a generalization of the *principle of substitutability* by Wegner and Zdonick [50], according to the extension relation of Brinksma et. all [7]. It is a transition relation, close to the failures model.

Several researchers are working on this track, developing object-based process calculi and static typing disciplines which cope with non-uniform objects.

Jean-Louis Colaço and others [8, 9, 10, 11] propose a calculus of actors and a type system that aims at the detection of "orphan messages" (messages that may never be accepted in some actor's execution path, either because the requested service is not in the actor's interface, or due to dynamic changes in an actor's interface). Types are interface-like, with multiplicities (thus, without dynamic information), and the type system requires some complex operations on a lattice of types. A set-constraints algorithm does the type inference.

Elie Najm and Abdelkrim Nimour [27, 29, 28] propose several versions of a calculus of objects featuring dynamically changing interfaces and distinguishing private and public objects' interfaces. For each version of the calculus, they develop a typing system handling dynamic method offers in private interfaces, and guaranteeing some liveness properties. Types are sets of deterministic guarded parametric equations, equipped with a transition relation, and represent infinite state systems. They define an equivalence relation, a compatibility relation, and a subtyping relation on types, based on the simulation and on the bisimulation relations.

Franz Puntigam [38, 39, 40, 41] defines a calculus of concurrent objects, a process-algebra of types (with the expressiveness of a non-regular language), and a type system which guarantees that all messages that are sent to an object are accepted; sequencing of messages is enforced to achieve the purpose.

Gérard Boudol proposes a dynamic type system for the blue calculus (a variant of the π-calculus directly incorporating the λ-calculus) [5]. The types are functional, in the style of Curry-Church simple types, and incorporate Hennessy-Milner logic with recursion (thus, with modalities, interpreted as resources of names). Processes inhabit the types, and this approach captures some causality in the usage of names in a process, ensuring that messages to a name will meet a corresponding offer. Well-typed processes behave correctly, this correct behavior being preserved under computation.

In the context of the lazy λ-calculus [2], Laurent Dami proposes a liberal approach to potential errors [12]. He argues that the common notion of erroneous term is over-restrictive: some programs, in spite of having error terms inside

them, do not actually generate a run-time error when executed. Since there is a family of programming languages based on the lazy λ-calculus, Dami proposes a lazy approach to errors—a term is considered erroneous if and only if it always generates an error after a finite number of interactions with its context.

It seems quite natural to have a liberal approach to potential errors in the context of non-uniform concurrent objects. António Ravara and others claim that it allows a more flexible and behavior-oriented style of programming, and moreover, detects some deadlocks [44, 43, 42]. The type system they propose accepts all processes that 'traditional' systems [48] do, except for those that do not conform to the restriction that only the output-capability of names can be transmitted, and furthermore, rejects some deadlocked processes.

In the next section we summarize the four talks in this second building block—two on behavioral subtyping in the context of petri-nets and two on behavioral typing of non-uniform object calculi.

A Practical Approach to Behavioural Inheritance in the Context of Coloured Petri Nets (Charles Lakos and Glenn Lewis*)

There are a number of proposals for substitutability in the context of concurrent object-oriented systems. it is unclear whether these proposals are overly constrained for practical application.

In the context of colored petri nets, we propose a set of three incremental modifications which lie somewhere between weak an strong substitutability. The constraints that we impose can be checked statically and they have the property that if the refinement is *at least as live as* the abstraction, then strong substitutability holds. (This property *cannot* be checked statically.) An examination of case studies in the literature suggests that the above forms of refinement are applicable in practice.

While the above proposals were formulated in the context of colored petri nets, it turns out that if the colored petri nets are transformed into the corresponding (elementary) petri nets, then the three forms of refinement correspond to recognized net morphisms. The formal definition for these morphisms can be found elsewhere, as can the proofs that the composition of refinements is a refinement.

Current work is investigating the extent to which analysis techniques can take advantage of the structure implicit in the above incremental modifications in producing more efficient analysis.

Behavioural Types in CoOperative Objects (Nabil Hameurlain and Christophe Sibertin-Blanc)

Behavioral typing and subtyping has proved to be a very useful concept for the support of incremental reuse in the area of object-oriented (O-O) languages.

With the emergence of formalisms integrating the O-O approach and Petri nets, the question arises how behavioral subtyping may be supported by such formalisms. We present a formal framework for the definition of behavioral typing in CoOperative Objects, a concurrent Object Oriented language, based upon Client/Server Petri nets. This framework is based upon the preorder and equivalence relations which are considered in the study of concurrent systems, allowing to define various subtyping relations.

A Concurrent Object Calculus with Types that Express Sequences (Christof Peter* and Franz Puntigam)

Sequencing of messages specified by types of objects is desirable especially in concurrent systems. Types in popular concurrent object calculi cannot support sequencing of messages. We present a calculus that supports sequencing of messages and compare it to the calculus of Vasconcelos and Honda. Type safety in our calculus does not allow a certain kind of nondeterminism supported by other calculi.

Explicit Behavioral Typing for Object Interfaces (Elie Najm and Abdelkrim Nimour*)

In this paper we describe an approach for typing objects with non-uniform service availability. We define behavioral types for object interfaces based on labeled transition systems that specify the succession of available methods (services) at an interface. Each transition label is a method signature. In addition, each interface has to be declared public or private. A private object interface has only one client at a time and offers non-uniform services depending on the "protocol" the client and the server have agreed on. On the other hand, a public interface can have multiple clients at the same time and is required to perform the same services for all its potential clients: the services on a public interface are uniform.

4. Open Discussion

The last workshop session consisted of an open "round-table" discussion on the topics of the workshop: objects as processes and behavioral types. We summarize briefly the individual subjects:

- If one successfully studies objects as processes, why not also extend this to general forms of "component", and look for more general notions of plugging of objects/components?
- What is the border between types and programs, or type systems and process calculi, respectively? Should type declarations be added explicitly? What should be typed? Names or processes?

- What do we do with untypable, i.e., ill-typed programs? Can we provide simple fixes automatically? Can we provide over- or underestimates, or should we simply generate warnings?
- What is the best notion of error? Or, do we have to have many different ones for different application domains?
- What is the "real" problem domain for behavioral types? Non-uniform Objects? Protocol specification?
- What would be a good means for comparing the many systems for behavioral types? By encodings into some generic foundational system? By case studies?

Although we could not find an agreement on the individual questions raised above, we agreed at least on the fact that more work needs to be done and seems to be worth being carried out.

Participants: Gérard Boudol, Ilaria Castellani, Mauro Gaspari, Günter Graw, Glenn Lewis, Markus Lumpe, Uwe Nestmann, Oscar Nierstrasz, Abdelkrim Nimour, Christof Peter, António Ravara, Arend Rensink, Jean-Guy Schneider.

5. Conclusion

The summaries of workshop contributions presented, and summarized above, show that the field of object-oriented programming can substantially profit from both the representation of objects as processes as well as from the borrowing of process-theoretic concepts for adaptation to object-oriented settings.

References

[1] Martín Abadi and Luca Cardelli. *A Theory of Objects*. Monographs in Computer Science. Springer-Verlag, 1996.

[2] Samson Abramsky. The lazy lambda-calculus. In *Research Topics in Functional Programming*, pages 65–117. Addison Wesley, 1990.

[3] Pierre America. Issues in the design of a parallel object-oriented language. *Formal Aspects of Computing*, 1(4):366–411, 1989.

[4] Jos Baeten and Peter Weijland. *Process Algebra*, volume 18 of *Cambridge Tracts in Computer Science*. Cambridge University Press, 1990.

[5] Gérard Boudol. Typing the use of resources in a concurrent calculus. In *Asian Computing Science Conference*, volume 1345 of *lncs*, pages 239–253. sv, 1997.

[6] Gérard Boudol. The π-calculus in direct style. *Higher-Order and Symbolic Computation*, 11:177–208, 1998. Previously published in the *Proceedings of POPL '97*, pages 228–241.

[7] Ed Brinksma, Giuseppe Scollo, and Chris Steenbergen. LOTOS specifications, their implementations and their tests. *Protocol Specification, Testing and Verification VI, (IFIP)*, pages 349–360, 1987.

[8] Jean-Louis Colaço, Mark Pantel, and Patrick Sallé. CAP: an actor dedicated process calculus. In *Workshop Reader of the 10th European Conference on Object-Oriented Programming (ECOOP'96)*. Dpunkt Verlag, 1996.

[9] Jean-Louis Colaço, Mark Pantel, and Patrick Sallé. A set constraint-based analyses of actors. In *2nd IFIP Workshop on Formal Methods for Open Object-based Distributed Systems (FMOODS'97)*. Chapman & Hall, 1997.

[10] Jean-Louis Colaço, Mark Pantel, and Patrick Sallé. From set-based to multiset-based analysis: a practical approach. In *4th Workshop on Set Constraints and Constraint-based Program Analysis*, 1998. Satellite event of the *4th International Conference on Principles and Practice of Constraint Programming (CP'98)*.

[11] Jean-Louis Colaço, Mark Pantel, Fabien Dagnat, and P. Sallé. Static safety analyses for non-uniform service availability in actors. In *4th IFIP Workshop on Formal Methods for Open Object-based Distributed Systems (FMOODS'99)*. Kluwer, 1999.

[12] Laurent Dami. Labelled reductions, runtime errors and operational subsumption. In *24th International Colloquium on Automata, Languages and Programming (ICALP'97)*, volume 1256 of *Lecture Notes in Computer Science*, pages 782–793. Springer-Verlag, 1997.

[13] Paolo Di Blasio and Kathleen Fisher. A concurrent object calculus. In Ugo Montanari and Vladimiro Sassone, editors, *Proceedings of CONCUR '96*, volume 1119 of *Lecture Notes in Computer Science*, pages 655–670. Springer-Verlag, 1996. An extended version appeared as Stanford University Technical Note STAN-CS-TN-96-36, 1996.

[14] Andrew D. Gordon and Paul D. Hankin. A concurrent object calculus: Reduction and typing. In Uwe Nestmann and Benjamin C. Pierce, editors, *Proceedings of HLCL '98*, volume 16.3 of *Electronic Notes in Theoretical Computer Science*. Elsevier Science Publishers, 1998.

[15] Charles A. R. Hoare. Communicating sequential processes. *Communications of the ACM*, 21(8):666–677, 1978.

[16] Kohei Honda and Mario Tokoro. An object calculus for asynchronous communication. In Pièrre America, editor, *Proceedings of ECOOP '91*, volume 512 of *Lecture Notes in Computer Science*, pages 133–147. Springer-Verlag, July 1991.

[17] Hans Hüttel and Josva Kleist. Objects as mobile processes. Research Series RS-96-38, BRICS, October 1996. Presented at MFPS '96.

[18] Hans Hüttel, Josva Kleist, Massimo Merro, and Uwe Nestmann. Migration = cloning ; aliasing (preliminary version). In *Informal Proceedings of the Sixth International Workshop on Foundations of Object-Oriented Languages (FOOL 6, San Antonio, Texas, USA)*. Sponsored by ACM/SIGPLAN, 1999.

[19] Cliff Jones. Constraining interference in an object-based design method. In Marie-Claude Gaudel and Jean-Pierre Jouannaud, editors, *Proceedings of TAPSOFT '93*, volume 668 of *Lecture Notes in Computer Science*, pages 136–150. Springer-Verlag, 1993.

[20] Cliff Jones. Accomodating Interference in the Formal Design of Concurrent Object-Based Programs. *Formal Methods in System Design*, 8(2):105–122, 1996. To appear.

[21] Josva Kleist and Davide Sangiorgi. Imperative objects and mobile processes. In David Gries and Willem-Paul de Roever, editors, *Proceedings of PROCOMET '98*, pages 285–303. International Federation for Information Processing (IFIP), Chapman & Hall, 1998.

[22] Xinxin Liu and David Walker. Confluence of processes and systems of objects. In Peter D. Mosses, Mogens Nielsen, and Michael I. Schwarzbach, editors, *Proceedings of TAPSOFT '95*, volume 915 of *Lecture Notes in Computer Science*, pages 217–231. Springer-Verlag, 1995. Presented in the CAAP-section. Available as University of Warwick Research Report CS-RR-272, October 1994.

[23] Xinxin Liu and David Walker. Partial confluence of processes and systems of objects. *Theoretical Computer Science*, 1998.

[24] Xinxin Liu and David Walker. Concurrent objects as mobile processes. In Plotkin et al. [37]. To appear.

[25] Robin Milner. *A Calculus of Communicating Systems*. Springer-Verlag, 1980. LNCS 92.

[26] Robin Milner, Joachim Parrow, and David Walker. A calculus of mobile processes, part I/II. *Information and Computation*, 100:1–77, September 1992.

[27] Elie Najm, Abdelkrim Nimour, and Jean-Bernard Stefani. A calculus of object bindings. In *2nd IFIP Workshop on Formal Methods for Open Object-based Distributed Systems (FMOODS'97)*. Chapman & Hall, 1997.

[28] Elie Najm, Abdelkrim Nimour, and Jean-Bernard Stefani. Guaranteeing liveness in an object calculus through behavioral typing. In *IFIP Joint International Conference Formal Description Techniques For Distributed Systems and Communication Protocols & Protocol Specification, Testing, and Verification (FORTE/PSTV'99)*. Kluwer, 1999.

[29] Elie Najm, Abdelkrim Nimour, and Jean-Bernard Stefani. Infinite types for distributed objects interfaces. In *4th IFIP Workshop on Formal Methods for Open Object-based Distributed Systems (FMOODS'99)*. Kluwer, 1999.

[30] O. Nierstrasz. Regular types for active objects. In *Object-Oriented Software Composition*, pages 99–121. Prentice Hall, 1995.

[31] Oscar Nierstrasz. Towards an object calculus. In M[ario] Tokoro, O[scar] Nierstrasz, and P[eter] Wegner, editors, *Object-Based Concurrent Computing 1991*, volume 612 of *Lecture Notes in Computer Science*, pages 1–20. Springer-Verlag, 1992.

[32] Anna Philippou. *Reasoning about Systems with Evolving Structure*. PhD thesis, University of Warwick, December 1996.

[33] Anna Philippou and David Walker. On confluence in the π-calculus. In Pierpaolo Degano, Roberto Gorrieri, and Alberto Marchetti-Spaccamela, editors, *Proceedings of ICALP '97*, volume 1256 of *Lecture Notes in Computer Science*, pages 314–324. Springer-Verlag, 1997.

[34] Anna Philippou and David Walker. On transformations of concurrent object programs. *Theoretical Computer Science*, 195(2):259–289, 1998. An extended abstract appeared in *Proceedings of CONCUR '96*, LNCS 1119: 131–146.

[35] Benjamin C. Pierce. Fränkische OOrientierungstage 1993 (Rothenbühl, Fränkische Schweiz, Germany). Tutorial on programming in the π-calculus, 1993.

[36] Benjamin C. Pierce and David N. Turner. Pict: A programming language based on the pi-calculus. In Plotkin et al. [37]. To appear.

[37] Gordon Plotkin, Colin Stirling, and Mads Tofte, editors. *Proof, Language and Interaction: Essays in Honour of Robin Milner*. MIT Press, 1999. To appear.

[38] Franz Puntigam. Types for active objects based on trace semantics. In *1st IFIP Workshop on Formal Methods for Open Object-based Distributed Systems (FMOODS'96)*, pages 5–20. Chapman & Hall, 1996.

[39] Franz Puntigam. Coordination requirements expressed in types for active objects. In *11th European Conference on Object-Oriented Programming (ECOOP'97)*, number 1241 in lncs, pages 367–388. sv, 1997.

[40] Franz Puntigam. Coordination requirements expressed in types for active objects. In *4th International Euro-Par Conference*, number 1470 in lncs, pages 720–727. sv, 1998.

[41] Franz Puntigam. Non-regular process types. In *5th International Euro-Par Conference*, number 1685 in lncs, pages 1334–1343. sv, 1999.

[42] António Ravara and Luís Lopes. Programming and implementation issues in nonunifom TyCO. Technical report, Department of Computer Science, Faculty of

Sciences, University of Porto, 4150 Porto, Portugal, 1999. Presented at the *Workshop on Object-Oriented Specification Techniques for Distributed Systems and Behaviours (OOSDS'99)*. Satellite event of the *1st Conference on Principles, Logics and Implementations of high-level programming languages (PLI'99)*. Web page: http://www.tec.informatik.uni-rostock.de/IuK/congr/oosds99/program.htm.

[43] António Ravara, Pedro Resende, and Vasco T. Vasconcelos. An algebra of behavioural types. Technical report, Section of Computer Science, Department of Mathematics, Instituto Superior Técnico, 1049-001 Lisboa, Portugal, 1999. Preliminary version presented at the *1st Workshop on Semantics of Objects as Processes (SOAP'98)*. Satellite event of the *25th International Colloquium on Automata, Languages and Programming (ICALP'98)*. Web page: http://www.cs.auc.dk/soap99/index98.html.

[44] António Ravara and Vasco T. Vasconcelos. Behavioural types for a calculus of concurrent objects. In *3th International Euro-Par Conference*, number 1300 in lncs, pages 554–561. sv, 1997. Full version available as DM-IST Research Report 06/97.

[45] Davide Sangiorgi. An interpretation of typed objects into typed π-calculus. *Information and Computation*, 143(1):34–73, 1998. Earlier version published as Rapport de Recherche RR-3000, INRIA Sophia-Antipolis, August 1996.

[46] Frits Vaandrager. Process algebra semantics for POOL. Report CS-R862, Centre for Mathematics and Computer Science, Amsterdam, August 1986.

[47] Vasco T. Vasconcelos. *A process-calculus approach to typed concurrent objects*. PhD thesis, Keio University, 1994.

[48] Vasco T. Vasconcelos and Mario Tokoro. A typing system for a calculus of objects. In *1st International Symposium on Object Technologies for Advanced Software*, volume 742 of *lncs*, pages 460–474. sv, 1993.

[49] David Walker. Objects in the π-calculus. *Information and Computation*, 116(2):253–271, 1995.

[50] Peter Wegner and Stanley B. Zdonik. Inheritance as an incremental modification mechanism or what like is and isn't like. In *2nd European Conference on Object-Oriented Programming (ECOOP'88)*, number 322 in lncs, pages 55–77. sv, 1988.

[51] Xiaogang Zhang and John Potter. Class-based models in the pi-calculus. In Christine Mingins, Roger Duke, and Bertrand Meyer, editors, *Proceeding of The 25th International Conference in Technology of Object-Oriented Languages and Systems (TOOLS Pacific '97, Melbourne, Australia)*, pages 219–231, November 1997.