Welcome to the Jungle
A subjective guide to mobile process calculi

Uwe Nestmann

Technical University of Berlin, Germany

Abstract. Almost 30 years ago, the research on process calculi gained a lot of momentum with the invention of ACP, CCS and CSP. Later on, but also already 20 years ago, researchers started to consider so-called mobile variants of process calculi, in which communication channels were themselves treated as the exchanged data. The original Pi Calculus arose out of a reformulation and extension of CCS. In turn, it boosted the invention and study of a whole zoo of further process calculi.
In this tutorial, we provide a bird's-eye view on the jungle of results, techniques and subtleties about mobile process calculi. Next to a rough overview on the zoo of calculi, this includes the coverage of both semantic and pragmatic aspects, ranging from notions of equivalence and expressiveness to challenging application domains.

Disclaimer

This document does *not* intend to constitute yet another, possibly updated bibliographic article about mobile process calculi. There have been several already. To my knowledge, Kohei Honda did the first one in 1998, published online. Silvano Dal-Zilio did another one in 2001 [Dal01], integrating references to "truly mobile" calculi reminiscent of Mobile Ambients. Finally, during the years 1994–2003, Björn Victor and I actively co-maintained an online bibliography and web pages on the topic of Calculi for Mobile Processes [NV98]. When we stopped updating the bib-files, the corresponding LaTeX'ed version of the complete bibliography was 29 pages long, of course not even being complete at that time.

This document neither intends to constitute a typical technical tutorial-like introduction to mobile process calculi. There have been several already. The usual suspects that I would recommend are the ones listed on the mobility web pages, carefully written by Milner et. al. [MPW92,Mil99], Parrow [Par01], Pierce [Pie97], Sangiorgi [San01], Sewell [Sew00], etc.

This document, as well as the actual tutorial talk at CONCUR, rather tries to respond to typical critical questions that I often come across when having to defend mobile process calculi. The questions matter, especially when posed by "non-believers" who are very knowledgeable concerning *immobile* process calculi (like ACP, CCS and CSP) and who challenge that mobile process calculi would be truly foundational, canonical, elegant, or even useful at all. Especially when compared to the previous immobile calculi. The idea for such a document dates back to July 2003 when I proposed to the `moca` mailing list to develop a *Mobility FAQ*. Unfortunately, this idea had not taken off since, so here I go again.

To my colleagues in the Pi Calculus community, I apologize for possibly having missed to cite some of their own work, but there are simply too many interesting papers to include all that I probably should have. I also apologize for possibly trivializing too much on certain aspects within this informal guide.

The following questions (each listed as a separate section) represent just a starting point. Many more deserve to be posed and hopefully will be (cf. §10).

1 What's the relation between Ambient and Pi Calculi?

A welcome source of confusion. It is triggered by both families of calculi being equipped with the label "mobile", but interpreted on a quite different concept of "locations", i.e., places within a distributed system. The mobility difference is of course also manifest in the two quite different underlying basic computational paradigms. An obvious commonality is the role and treatment of names and the use of a restriction operator to dynamically create fresh names and to statically delimit and govern their scope during future computations. Both families of calculi are therefore nowadays also called *nominal* (cf. [Gor02]).

In Pi Calculi [MPW92], the location of a process can be understood as determined by its surrounding network configuration: the (current) set of active communication channels (a processes' communication interface), together with the set of partners reachable through these channels. Then, the location of a process changes whenever its surrounding network configuration changes. The role of name passing is precisely to implement such dynamic reconfigurations by means of explicit communication. This has been very clearly explained, e.g., by Milner [Mil99, §8]. The real expressive power of the Pi Calculus stems from the fact that also local (read: bound) names can be shipped in a controlled fashion.

In contrast, in Ambient Calculi [CG00], the location of a process is modeled explicitly by means of a syntactic entity. Locations usually appear as nested hierarchies. Processes themselves *are* locations. Computation occurs by having locations move around the hierarchy, thus by changing it. Depending on the set of Ambient primitives, location boundaries can also be dissolved dynamically.

In synthesis, there are also extensions of Pi Calculi with explicit concepts of location and movement, like the Distributed Pi Calculus or the Nomadic Pi Calculus. On the other hand, for convenience, also the Ambient Calculus and its descendants provide explicit communication concepts, although often bound to be used locally within a single ambient. To compare the expressive power of Pi and Ambient Calculi, mutual encodings have been studied in great detail.

With respect to this section, I also recommend Silvano Dal-Zilio's commented bibliography [Dal01]. In it, he structures the discourse by dividing mobile calculi into *labile* process systems (relating to chemistry) and *motile* (relating to biology) process systems. The former corresponds more to the first-order movement of names, the latter more to the higher-order movement of processes.

In this document, I will deliberately focus much more on Pi calculi.

2 The Pi Calculus is too complicated!

There are actually many different reasons why people conceive the Pi Calculus as being complicated. In fact, sometimes it really is. But it does not have to be.

The syntax is unreadable. Many different syntactic variants of the Pi Calculus have appeared over the years. However, for understanding how to send a value v over a channel a it should not matter that much whether you write it as $\overline{a}v$ or $\overline{a}\langle v\rangle$ or $a \leftarrow v$ or $a!v$. I do agree that the "bar" notation is not necessarily ideal, but it originates from CCS, and one can also get used to it. The arrow notation was introduced in Honda's ν-calculus, which is asynchronous (see below) and makes it resemble an assignment. The "bang" notation is closely related to ASCII syntax, which is useful when the goal is to discuss programming idioms, but you might confuse it with the replication bang (see below).

A quick word about names ... and (channel) constants ... and variables. In CCS with label passing [EN86], all of these entities were modeled as distinct syntactic concepts. In the original Pi Calculus [MPW92], they were unified into the single concept of names. Both approaches have their advantages and disadvantages. If you have just names, then many definitions get much more light-weight. If you have the distinction, then you get easier control when you have to distinguish names according to the positions in which they may occur. The differences are subtle, but most of the time just a matter of taste, philosophy or readability.

To be foundational, there are far too many primitives. When compared to the Lambda Calculus, there are definitely more primitives. But, the domain of concurrent computation is also more complicated. On the other hand, one of the smallest and in my opinion very readable Pi Calculus with well enough expressivity is the Asynchronous Pi Calculus, my personal favorite, with or without the *matching* construct $[x = y]P$. Its syntax is generated from this grammar:

$$P ::= \underbrace{\mathbf{0}}_{\textit{nil}} \quad | \quad \underbrace{P|P}_{\textit{parallel}} \quad | \quad \underbrace{\overline{a}\langle v\rangle}_{\textit{message}} \quad | \quad \underbrace{a(x).P}_{\textit{reception}} \quad | \quad \underbrace{!\, a(x).P}_{\textit{replication}} \quad | \quad \underbrace{(\boldsymbol{\nu} x)\, P}_{\textit{restriction}}$$

It can also get smaller, but this is rather a debate covered by the study of (mutual) encodings between the various calculi. See Section 6, but also [Yos02] for a study of minimality based on combinators appealing to the Lambda Calculus.

The labeled semantics has too many side-conditions. I got two answers.

Labels characterize the interactions with the observing environment. Under dynamic reconfigurations, the scope of binding structures may span across the interface between an observed process system and its concurrently observing process. As a consequence, a labeled semantics must carefully keep track of which names may occur in free or bound position. This justifies the side-conditions.

Now, please have a look at §4, and then come back here. In the spirit of the various incarnations of the Pi Calculus, the labeled semantics is only used for the purpose of providing an underlying deductively defined transition relation on which to define labeled bisimilarities—as proof techniques. For understanding the execution of processes, reduction semantics plainly suffices.

There are far too many kinds of bisimulation. Yes and no. The problem that is usually referred to, when complaining about the many bisimilarities in Pi Calculi, is the different versions called *ground*, *early*, *late*, *open*, ...

First, consider that also for immobile process calculi, a frightening number of process equivalences have been defined, studied and compared [Gla93]. There, the usually accepted defense seems to be that the choice of process equivalence depends on the application under study: the strongest equivalence that is satisfied by a pair of processes tells you a lot about their intuition and semantics. For the many bisimilarities of Pi Calculi, such an argumentation does not quite play well, because the notions are too close to each other. The question of which bisimulation scheme implies the more natural equations leads to debatably subjective answers on whether they matter in practice [SW01].

Now, ultimately, we are interested in process *congruences*. I simply take this point of view for granted, so I will not further argue for it. In process calculi with explicit passing of data, congruences also need to consider closure properties w.r.t. input prefix, i.e., the appearance of processes as continuations of input clauses, where input variables are to be substituted by the received data.[1] All of the above-mentioned notions are just variants of bisimulation that differ in the treatment of substitution of names (variables) by other names (data). Unfortunately, the moment on *when* to apply substitutions, gives rise to strictly different bisimilarities. Which one is the good one? There are at least three answers.

If you want a bisimilarity that constitutes a congruence by itself, then choose the *open* variant. It is also the one, for which more verification tools exist (cf. §7).

If you are of the opinion that the Pi Calculus is no good, because all these bisimilarities differ, then you should be happy to see that there are Pi Calculi in which the bisimilarities actually coincide ... and are congruences themselves. This is one of the many reasons for which I (very subjectively) very much "like" the Asynchronous Pi Calculus (cf. §2 and §4). There is also the Private Pi Calculus (formerly called "Pi Calculus with internal mobility") [SW01]. Both of these are strict subcalculi of the "standard" Pi Calculus that nevertheless retain sufficient expressive power (cf. §6) for most if not all practical purposes.

Last but not least, please (unless you have already had) have a look at §4, and then come back here. Assume you agree that barbed congruence or reduction congruence is what your bisimulation-based congruence[2] shall coincide with. Then, depending on the calculus at hand, chances are very good that the bisimulation scheme to go for (i.e., to base your congruence on) is the *early* one.

[1] Note that some problems already arise in the context of Value Passing CCS [Ing94], so it is not only a problem of name passing calculi.

[2] Usually, we take the largest congruence contained in a given bisimilarity by requiring its closure under substitution. The result is called *full bisimilarity* in [SW01].

3 Why are there so many different calculi?

Counter-question: why are there so many different programming languages?

Surely, there are several different driving forces that triggered this abundance of Pi Calculi. We may classify them according to several quests, namely for:

minimality Which primitives are needed to retain sufficient expressive power? Usually, particular (combinations of) primitives are best studied in isolation, each time giving rise to a new variation of the calculus. Typical primitives of interest are the basic communication paradigms, which can be synchronous [MPW92], asynchronous [Bou92], include selection labels as in TyCO [Vas94], and the inclusion of matching and mismatching primitives (there would be too many papers to cite, here).

applicability For every application (domain), one usually has the choice between the use of the bare base calculus, or some syntactically sugared version. Then, it is often a matter of efficiency of available implementations or the precision and comprehension of feedback from formal analysis that triggers the decision to come up with yet another extension of the calculus. Suitable data types are a popular candidate to conveniently extend existing calculi (see also the Applied Pi Calculus [AF01]). Sometimes, however, completely new dimensions are explicitly added to some base calculus, driven by the application. Explicit notions of space [RH98,WS00,Uny01], time [Ber04], stochastics [Pri95], probabilities [HP00], . . . , each give rise to new calculi.

implementability The interpretation of Pi Calculus as a computational formalism implies that it shall be implemented in order to execute its programs on computers, not just on paper. This quest contributed to the wide-spread use of (extended) sub-calculi often based on asynchronous communication like the ν-calculus [HT92], Lπ [MS04], and the Join Calculus [FG96]: synchronous communication, especially combined with general form of summation, is too hard to implement efficiently in a distributed setting [Pal03].

4 What are these "barbs"?

At the time of CCS, providing an operational semantics to a process calculus meant to write out deduction rules for a labeled transition relation, where the intuition is that labels characterize what a process is able to do. One distinguishes internal (invisible) from external (visible) steps. Visible steps model the interactions with an observer—a process that represents the actions of the observing environment. Technically, the labels were just copied from the occurrence of syntactic actions in process terms, with one exception: internal steps (labeled by τ) can also be triggered by the concurrent execution of two complementary (syntactic) actions, occurring at two different syntactic positions within the process. Back then, the world was mostly simple and elegant. ACP was a bit more flexible by allowing more varied new actions to be produced by means of combinations of concurrent actions. With the Pi Calculus, labeled semantics got more complicated since actions had to consider more structure: directed channels (subjects), data (objects) and even dynamically adaptable binding information. A labeled semantics to keep track of such aspects looks, at first, rather complicated to the outsider. Even worse, there seem to be several different but equally meaningful possibilities to characterize oberver actions through labels.

Let us take a step back. Initially, labeled transition semantics tried to prepare for (at least) three things at a time: (1) to define an execution semantics, which also considered actions at the interface to observers; (2) to define a sensible notion of observational congruence, and (3) to provide a tractable (co-inductive) proof technique for this notion. It is here that the Pi Calculus is too rich a setting in that it allows for many reasonable choices of how to define things (cf. §2). Now, the "barbed approach" [MS92] provides a simple and reasonably uniform solution, based on the separation of the above-mentioned three definitional tasks.

1. One defines a simple unlabeled transition semantics, called *reduction semantics* that only considers the execution of internal steps, which are independent of the observing environment. To this aim, and to cope with the syntactic distribution of the complementary parts of redexes, reduction is considered modulo structural rearrangement congruence (typically monoid laws for composition and summation, among others), according to

$$\frac{P \equiv Q \qquad Q \longrightarrow Q' \qquad Q' \equiv P'}{P \longrightarrow P'}$$

   The main reduction rule for the Asynchronous Pi Calculus above then is:

$$\overline{a}\langle v\rangle \mid a(x).P \longrightarrow \{{}^{v}\!/\!_{x}\}P$$

   In Pi Calculus, the treatment of fresh names is also conveniently pushed into the notion of structural congruence, essentially by the law

$$(\boldsymbol{\nu}x)\,(\,P \mid Q\,) \equiv ((\boldsymbol{\nu}x)P) \mid Q \qquad \text{if } x \text{ does not occur freely in } Q$$

   that allows to extend or syntactically shrink the scope of bound names. Thus, to model the so-called *scope extrusion*, one just first structurally extends the scope—reading the above law from right to left—and then applies the simple standard reduction rule within the scope. As a result, the reduction semantics for the Pi Calculus looks actually almost as simple as the one for CCS.[3]

2. Based on a reduction semantics, one defines a bisimulation-like notion of observational congruence on top. Since reductions do not contain information about the observing environment, the congruence is defined by explicit quantification over sets of *contexts*, i.e., processes with a single hole. Formally, a symmetric relation $\mathcal{B}$ is a *barbed bisimulation*, if whenever $P\ \mathcal{B}\ Q$, then
   – $P \longrightarrow P'$ implies that there is $Q \Rightarrow Q'$ with $P'\ \mathcal{B}\ Q'$, and

[3] Exercise: to get a feeling for structural congruence, it is instructive to try to find a minimal set of structural rearrangement laws ($\equiv$) that suffice to define CCS reduction in 1-1 correspondence to the labeled τ-transitions of the respective CCS-terms.

– P "has barb O" iff Q "has barb O".

The first item essentially requires the notion to be a *reduction bisimulation.* In the second item, *barb* O is usually instantiated by some *observation predicate*, often written as $\Downarrow$ (or $\Downarrow_a$), telling that one can reach a state where some (or a particular kind of action involving the name a) action is enabled.

As usual, two processes P and Q are called *barbed bisimilar*, written $P \approx_{\mathrm{B}} Q$, if there is a barbed bisimulation $\mathcal{B}$ that contains (P, Q). Barbed bisimulation as such is not very interesting, because it does not consider interactions with an observing environment. Thus, we add this information explicitly. Two processes are *barbed congruent*, written $P \cong_{\mathrm{B}} Q$, if, for all contexts $C[\cdot]$, $C[P] \approx_{\mathrm{B}} C[Q]$. Thus, barbed congruence is a *contextual* notion. To prove two processes congruent, in addition to the otherwise simple obligations of the definition, one has to consider an infinite number of contexts.

But, since quantification is done over an explicit set of contexts, we now also have a convenient definitional scheme at hand to define coarser congruences by cutting down the number of contexts considered. The biggest beneficiaries are the following two: (1) notions of *typed barbed congruence* (cf. [PS96]), where the class of contexts to consider is characterized by only those that yield well-typed terms when plugging some process into the hole; (2) notions of *congruence up to translated contexts* in the study of encodings (cf. §6).

3. One is then looking for a labeled transition semantics and a suitable labeled notion of bisimulation congruence that—as a proof technique—approximates the previously defined uniform notion of barbed congruence. In other words, barbed congruence should guide the quest for finding the "right notion of labeled congruence". Ideally, of course, one finds a labeled version that is not only sound, but also complete w.r.t. the barbed notion.

Note that the above story is actually independent of the Pi Calculus. However, it was the Pi Calculus in whose rich context one felt the need to establish some more uniform way to capture observational equivalences. Later on, it has also been followed by many other calculi, e.g., Ambient Calculi and Spi Calculus.

Accompanying the discussion of barbed congruence, one should also mention the arguably more canonical definition of so-called *reduction congruence* [HY95]: the main difference is that no notion of observation predicate (i.e., barb) needs to be justified. Instead, the general notion of *process insensitivity* is used to govern the definition of process congruence on top of reduction-closed equality (roughly corresponding to reduction bisimulation). In most calculi, reduction congruence is finer than barbed congruence. Roughly, reduction congruence corresponds to the requirement of quantification over contexts after every step in the bisimulation game. Fournet and Gonthier [FG04] have proved, for the Asynchronous Pi Calculus with matching, that barbed congruence and reduction congruence coincide, also with labeled asynchronous bisimulation (cf. §2).

5 Are there any applications? More precisely: beyond what you can do with CCS . . .

In the early years, there was the convincing analysis of (simple versions of) the GSM handover protocol, being reused in many papers, with more or less complicated scenarios. The situation reminds a bit of the numerous studies on the Alternating Bit Protocol within immobile process calculi and other concurrency formalisms during the 1980's. This is not necessarily bad, because it allows researchers to quickly get to the point and it allows referees to quickly understand the added value of the technique contributed by some submitted paper.

It is probably true that, for a too long time, the GSM handover has been the only truly mobile application example. Instead, those parts of the mobile process calculus community that were interested in "applications" went on studying the representations and implementations of high-level programming concepts by means of encoding them into a basic mobile process calculus (cf. §6). Examples include typed imperative, functional, object-oriented and even constraint-oriented programming, the latter two also in concurrent versions, whose semantics tends to be complex and difficult to get right. Although the modeling or encodings of high-level programming paradigms, as well as of typed data structures bears a number of similarities with assembler languages, the contributions can be stated as being surprisingly successful: the precision, depth and comprehensibility of results that could be achieved by translating concepts into Pi Calculi is astonishing, particularly in the functional and object-oriented domain. Also the Actor model—one of the sources of inspiration for the Pi Calculus—could be understood by representing it as a particular typed Pi Calculus [AT04].

Nevertheless, more and more truly mobile applications have been entering the landscape. The motivation to model Internet phenomena led to various distributed Pi Calculi, as well as versions that natively integrate XML datatypes, and also the Ambient Calculus. The Spi Calculus [AG99] has been used with success to model security protocols. Stochastic Pi Calculi have been used to model phenomena in Systems Biology [PRSS01]. This domain also inspired the development of tailor-made nominal calculi [DL04]. Finally, the domain of Business Process Modeling provides applications, where the standard original Pi Calculus has proved to be a useful modeling tool [PW05].

In most if not all of these applications, the expressive power and careful treatment of fresh name generation—a distinctive feature when compared to immobile process calculi—is the key to concise and successful modeling.

6 What's the point of studying encodings?

Very many encodings that use Pi Calculi as target languages have been studied. So many that I do not include any explicit references in this part. The use of encodings is actually a quite standard technique. If you want to compare the *expressive power* of two calculi, then it is convenient to study mutual encodings between them.[4] If you want to represent *high-level programming concepts* (cf. §5) within some basic calculus, then you do this by means of an encoding function. Such encodings actually represent formal abstract implementations.

Encodings can be analyzed in many respects. Typically, a "good" encoding preserves and reflects as many semantic properties as possible, e.g., notions of equivalence, the decidability of properties, or just the existence of matching transitions according to the operational semantics. Although many criteria for the quality of encodings have been proposed, there is still not yet an agreed-upon "theory of encodings" that tells how all of those criteria relate. Ongoing debates include the following questions: (1) How compositional does an encoding have to be? Is it acceptable to translate $[\![P|Q]\!]$ as $C[[\![P]\!]|[\![Q]\!]]$, where C could be some powerful arbiter, or should we insist on C being the empty context? (2) Is it acceptable to require the intended behavior of encoded terms w.r.t. encoded contexts only, or should we insist on good behavior w.r.t. all contexts available in the target language? (3) Is it acceptable that an encoding introduces divergence?

7 What about tools?

Actually, there are quite a few available (see an incomplete list at [Nes]). Some of them are compilers that allow one to efficiently run mobile programs written within some high-level programming language based on one the Pi Calculi, and this even in truly distributed settings. Others are analysis tools that allow one to simulate Pi Calculus executions, even for the Stochastic Pi Calculus. Further tools automate equivalence-checking, various forms of reachability analysis, some of them specialized for security protocols written in variants of the Spi Calculus. A member of the `moca` mailing list summarized the situation in 2003 as follows: "Process-calculists are not so interested in coding and coders don't read our papers, so cross fertilization has been somewhat lacking". I think it is fair to say that the situation has changed at least a bit, but we could still profit from more "efficient" analysis tools for mobile process calculi.

8 After 20 years, what are the main contributions?

This is a difficult question.

It is certainly the case that the fundamental role and pragmatic expressive power of fresh-name generation has proved to be useful for many practical programming and modeling problems (cf. §5).

The proliferation and extensive study of so many variations of the original Pi Calculus has helped to build up a large set of semantic techniques and results. It also helped to identify typical pitfalls and traps such that the development of further (domain-specific) variations is nowadays a much easier task. It has in fact

[4] Apparently, the question of expressive power is particularly appealing in the context of concurrency. The respective high quality 1-day workshop EXPRESS, which at its peak attracted 30 submissions, is now already running its 13th incarnation.

become so easy to design new calculi (with notions of execution and observational congruence, according to the scheme of § 4) that there is the running gag that we can "make up new calculi while having breakfast, several of them".

I think it is also fair to state that we have not yet discovered the process calculus that constitutes *the* foundation of mobile/global/distributed computation. Some sweet spots have been identified, e.g., where the many bisimulations collapse. Maybe, it could be stated as a contribution that there is reason to believe that there simply is no single one, e.g., with a fixed single concept of communication. Yet, if there is a single one, then it is likely a nominal calculus.

I also count as a main contribution the extensive study of name-based type systems in the context of mobile process calculi. Although they sometimes appear to be complicated, they are rich and flexible. The community even managed to convince an EU-official in the context of the GC-initiative of the importance of type systems for many concepts of distributed computation.

In this document, I have not at all covered the study of unifying models that are designed to capture the different notions of sequential, concurrent and distributed computation, e.g., Bi-Graphs. Their evolutionary development can certainly be counted as a main contribution, supported by mobile process calculi.

9 I want to use the Pi Calculus. Should I take on off the shelf, or should I assemble my own little calculus?

There are some obviously good reasons for choosing a calculus off-the-shelf. (1) You want to use one of the tools that have been written to analyze Pi Calculus terms. (2) You do not yourself have to work out the theoretical results that you might need for your application. (3) You need to be able to state that you are using some "standard" calculus in order to convince your application community. (4) You only need to read one or two papers on Pi Calculus yourself.

Yet, there are also very good reasons for using your own home-brewed calculus. (1) The typical argument for the use of domain-specific languages applies: to model your application, existing calculi might not offer the right set of primitives, be it at the proper level of abstraction, or be it a completely independent modeling dimension that has not yet been considered appropriately. (2) You have a Pi Calculus guru next door who knows about the traps and pitfalls, who might help you work out the theory, or might even do it for you. (3) You might get another paper published by carefully motivating the design of your domain-specific process calculus. (4) You may help the process calculus community by inspiring the development of new techniques and theories.

10 Will there be further work?

Concerning the Pi Calculi themselves, I am convinced that active research will continue for many more years, but I know of no roadmap that the community has agreed to pursue, possibly apart from the UK Grand Challenge on Sciences for Global Ubiquitous Computing. A list of open problems could be a good start.

Ideally, this document triggers many more questions that the "Pi Calculus community" should respond to. I will happily collect any such information—questions *and* responses—and provide an online resource, e.g., in the form of a Wiki, such that the guide will become less subjective as it appears herein.

Acknowledgments. Jos Baeten and Holger Hermanns for discussing related issues at the social event of CONCUR 2001. The members of the `moca` mailing list, in particular Martin Berger, for some discussion in July/August 2003.

References

[AF01] M. Abadi and C. Fournet. Mobile Values, New Names, and Secure Communication. In *Proceedings of POPL '01*, pages 104–115. ACM, Jan. 2001.

[AG99] M. Abadi and A. D. Gordon. A Calculus for Cryptographic Protocols: The Spi Calculus. *Information and Computation*, 148(1):1–70, 1999.

[AT04] G. Agha and P. Thati. An Algebraic Theory of Actors and Its Application to a Simple Object-Based Language. In *Essays in Memory of Ole-Johan Dahl*, volume 2635 of *LNCS*, pages 26–57. Springer, 2004.

[Ber04] M. Berger. Basic Theory of Reduction Congruence for Two Timed Asynchronous π-Calculi. In P. Gardner and N. Yoshida, eds, *Proceedings of CONCUR 2004*, volume 3170 of *LNCS*, pages 115–130. Springer, Aug. 2004.

[Bou92] G. Boudol. Asynchrony and the π-calculus (Note). Rapport de Recherche 1702, INRIA Sophia-Antipolis, May 1992.

[CG00] L. Cardelli and A. D. Gordon. Mobile Ambients. *Theoretical Computer Science*, 240(1):177–213, 2000.

[Dal01] S. Dal-Zilio. Mobile Processes: a Commented Bibliograhy. In *Proceedings of MOVEP'2K — 4th Summer school on Modelling and Verification of Parallel Processes)*, volume 2067 of *LNCS*, pages 206–222. Springer, 2001.

[DL04] V. Danos and C. Laneve. Formal Molecular Biology. *Theoretical Computer Science*, 325(1):69–110, 2004.

[EN86] U. Engberg and M. Nielsen. A Calculus of Communicating Systems with Label-passing. Technical Report DAIMI PB-208, Computer Science Department, University of Aarhus, Denmark, 1986.

[FG96] C. Fournet and G. Gonthier. The Reflexive Chemical Abstract Machine and the Join-Calculus. In *Proceedings of POPL '96*, pages 372–385. ACM, 1996.

[FG04] C. Fournet and G. Gonthier. A Hierarchy of Equivalences for Asynchronous Calculi. *Journal of Logic and Algebraic Programming*, 63(1):131–173, 2004.

[Gla93] R. Glabbeek. The Linear Time – Branching Time Spectrum II: The semantics of sequential systems with silent moves (Extended Abstract). In *Proceedings of CONCUR '93*, volume 715 of *LNCS*, pages 66–81. Springer, 1993.

[Gor02] A. D. Gordon. Notes on nominal calculi for security and mobility. In R. Focardi and R. Gorrieri, eds, *Foundations of Security Analysis and Design*, volume 2171 of *LNCS*, pages 262–330. Springer, 2002.

[HP00] O. M. Herescu and C. Palamidessi. Probabilistic Asynchronous π-Calculus. In J. Tiuryn, ed, *Proceedings of FoSSaCS 2000*, volume 1784 of *LNCS*, pages 146–160. Springer, 2000.

[HT92] K. Honda and M. Tokoro. On Asynchronous Communication Semantics. In M. Tokoro, O. Nierstrasz and P. Wegner, eds, *Object-Based Concurrent Computing 1991*, volume 612 of *LNCS*, pages 21–51. Springer, 1992.

[HY95] K. Honda and N. Yoshida. On Reduction-Based Process Semantics. *Theoretical Computer Science*, 152(2):437–486, 1995. An extract appeared in *Proceedings of FSTTCS '93*, LNCS 761.

[Ing94] A. Ingólfsdóttir. *Semantic Models for Communicating Processes with Value-Passing*. PhD thesis, University of Sussex, 1994.

[Mil99] R. Milner. *Communicating and Mobile Systems: the π-Calculus*. Cambridge University Press, May 1999.

[MPW92] R. Milner, J. Parrow and D. Walker. A Calculus of Mobile Processes, Part I/II. *Information and Computation*, 100:1–77, Sept. 1992.

[MS92] R. Milner and D. Sangiorgi. Barbed Bisimulation. In W. Kuich, ed, *Proceedings of ICALP '92*, volume 623 of *LNCS*, pages 685–695. Springer, 1992.

[MS04] M. Merro and D. Sangiorgi. On Asynchrony in Name-Passing Calculi. *Mathematical Structures in Computer Science*, 14(5):715–767, 2004.

[Nes] U. Nestmann. Calculi for Mobile Processes. http://move.to/mobility.

[NV98] U. Nestmann and B. Victor. Calculi for Mobile Processes: Bibliography and Web Pages. *EATCS Bulletin*, 64:139–144, Feb. 1998.

[Pal03] C. Palamidessi. Comparing the Expressive Power of the Synchronous and the Asynchronous π-calculus. *Mathematical Structures in Computer Science*, 13(5):685–719, 2003.

[Par01] J. Parrow. An Introduction to the pi-Calculus. In *Handbook of Process Algebra*, pages 479–543. Elsevier Science, 2001.

[Pie97] B. C. Pierce. Foundational Calculi for Programming Languages. In *Handbook of Computer Science and Engineering*, pages 2190–2207. CRC Press, 1997.

[Pri95] C. Priami. Stochastic π-Calculus. *The Computer Journal*, 38(6):578–589, 1995. Proceedings of PAPM '95.

[PRSS01] C. Priami, A. Regev, E. Y. Shapiro and W. Silverman. Application of a Stochastic Name-Passing Calculus to Representation and Simulation of Molecular Processes. *Information Processing Letters*, 80(1):25–31, 2001.

[PS96] B. C. Pierce and D. Sangiorgi. Typing and Subtyping for Mobile Processes. *Mathematical Structures in Computer Science*, 6(5):409–454, 1996. An extract appeared in *Proceedings of LICS '93*: 376–385.

[PW05] F. Puhlmann and M. Weske. Using the π-Calculus for Formalizing Workflow Patterns. In *Proceedings of BPM 2005*, volume 3649 of *LNCS*, pages 153–168. Springer, 2005.

[RH98] J. Riely and M. Hennessy. A Typed Language for Distributed Mobile Processes. In *Proceedings of POPL '98*. ACM, 1998.

[San01] D. Sangiorgi. Asynchronous process calculi: the first-order and higher-order paradigms (Tutorial). *Theoretical Computer Science*, 253(2):311–350, 2001.

[Sew00] P. Sewell. Applied Pi — A Brief Tutorial. Technical Report 498, Computer Laboratory, University of Cambridge, 2000.

[SW01] D. Sangiorgi and D. Walker. *The π-calculus: a Theory of Mobile Processes*. Cambridge University Press, 2001.

[Uny01] A. Unyapoth. *Nomadic Pi Calculi: Expressing and Verifying Infrastructure for Mobile Computation*. PhD thesis, University of Cambridge, June 2001.

[Vas94] V. T. Vasconcelos. Typed Concurrent Objects. In *Proceedings of ECOOP '94*, volume 821 of *LNCS*, pages 100–117. Springer, 1994.

[WS00] P. Wojciechowski and P. Sewell. Nomadic Pict: Language and Infrastructure Design for Mobile Agents. *IEEE Concurrency*, 8(2):42–52, 2000.

[Yos02] N. Yoshida. Minimality and separation results on asynchronous mobile processes — Representability theorems by concurrent combinators. *Theoretical Computer Science*, 274(1–2):231–276, 2002.