Case for Support: Novel type systems for concurrent programming languages

Dr S. J. Gay
Department of Computer Science,
Royal Holloway, University of London

1 Track Record and Previous Research

Background: Dr Simon Gay

I have been working in concurrency, with an emphasis on types, for several years. This research began when I was a PhD student at Imperial College, London, under the supervision of Professor Samson Abramsky. Initially I studied the π-calculus [24], and in particular the system of *sorting* proposed by Milner [23]. This work led to the development of a sort inference algorithm for the π-calculus [9], which continues to be cited in the literature on types for the π-calculus. Later my research moved in the direction of *interaction categories*, a semantic framework for typed concurrency proposed by Abramsky [1, 2, 4, 5]. My PhD thesis [11] dealt with the theory and applications of interaction categories. Its contributions included a new analysis of the semantics of synchronous dataflow programs, in the interaction categories framework; the development of a synchronous process calculus with a type system based on the structure of interaction categories, and a categorical denotational semantics; and a study of the semantic basis of a type system guaranteeing deadlock-freedom, in both synchronous and asynchronous settings.

After completing my PhD I worked as a research assistant with Professor Chris Hankin at Imperial College, on the Esprit Basic Research Action "Coordination". Our work on this project concerned the parallel programming language Gamma [7]; we applied Abramsky's *domain theory in logical form* [3] to develop program logics for the language. The result was two program logics: one based on a resumption semantics for Gamma [12] and the other based on a transition trace semantics [13]. The connection with types for concurrency is via the view of types as specifications or program properties, which is a theme of domain theory in logical form. Indeed, a broadening of the scope of type-theoretic techniques has always been an aim of interaction categories research.

Another strand of my research is related to programming language implementation, in particular concerning Yves Lafont's language *interaction nets* [20]. I have developed a sequential implementation of this (potentially parallel) language [8]; I also discovered that interaction nets can be compiled into a small number of fundamental combinators [10]. More recently, in collaboration with Dr Ian Mackie at Ecole Polytechnique, Paris, I have been developing a parallel implementation of interaction nets.

Intended PhD student: Mr Malcolm Hole

The proposal includes funding for a project studentship; indeed, this is the only funding requested for manpower, and is the primary motivation for the proposal. It is intended that the studentship will be used to fund Mr Malcolm Hole. Mr Hole is a final year undergraduate in computer science at Royal Holloway. He has a strong academic record, and it seems likely that he will graduate with first class honours. His interests lie in the area of the proposed project, and his final year options include a number

of relevant subjects: concurrent programming, compilers, and a project on adding type-checking to a Unix shell.

The Department of Computer Science

The Formal Methods Group consists of Dr Steve Schneider, Dr Simon Gay and a number of research students and research associates. The interests of the group include concurrency theory, formal analysis of communication protocols (including security protocols), and specification and verification of timing properties in concurrent systems. The proposed project will fit nicely into the current range of interests, and complement them by placing the type-theoretic view of specification in a more prominent position. The broader interests of the group mean that the funded PhD student will be exposed to a wider range of ideas and techniques in the formal methods area.

Another proposed project, involving collaboration between Dr Simon Gay, Dr Steve Schneider, and Dr Alan Jeffrey (University of Sussex), will study methods of combining temporal and data specifications in process calculi. The type-theoretic approach taken by the project now being proposed will provide a useful alternative view of that area.

In the Languages and Architectures Group, Dr Adrian Johnstone and Dr Elizabeth Scott are working on compiler theory. Their compiler generator is widely used, and is being actively developed [16, 17]. This local expertise will be a useful source of help with the language design aspects of the proposed project.

Links to other research groups

I have developed and maintained good links with other researchers in the area of type systems for concurrency, partly through the involvement of Professor Abramsky's group at Imperial in the Esprit Basic Research Action "CONFER", whose broad aim was to study the relationship between functional and concurrent computation. Because some of the proposed research is based on extending the Pict programming language, I anticipate discussing progress with the developers of that language, Dr Benjamin Pierce (University of Indiana) and Dr David N. Turner (An Teallach Ltd). In addition, Dr Peter Sewell (University of Cambridge) is actively working on extensions of Pict, and we expect to be able to share useful experience with his group. I also have close links with Professor Abramsky's current group at Edinburgh, especially with Dr Kohei Honda and Dr Nobuko Yoshida who are actively working on types for concurrency. They have both contributed to the pool of theoretical ideas which form the basis for the proposed project, and it will be easy to discuss applications and further developments with them.

Dr Bent Thomsen and Dr Lone Leth (ICL Research and Advanced Technology, Bracknell) were collaborators on the Esprit projects CONFER and Coordination and I have maintained contact with them. They are interested in the development of enhanced type systems for industrial-strength distributed programming languages. The proposed project will contribute to the foundations of such type systems, and show that various theoretical ideas can be practically implemented. Dr Thomsen and Dr Leth have provided a letter of support (included with this application), and we will find it useful to discuss our progress with them during the project. We hope to be able to identify areas of future collaboration, in order to bring the results of the project closer to industrial application.

2 Proposed Research

2.1 Introduction

This project is about the design and implementation of static type systems for concurrent programming languages. Compile time type checking for sequential languages is an established technology, and is of great assistance in software development. During the last few years, much research activity has been devoted to extending type theory to the description of communication behaviour in the context of process calculi and also concurrent object oriented languages. The general idea is to specify certain properties of interprocess communication by means of types, and use type checking to verify correctness.

The aim of the project is to use the concurrent programming language Pict [27] as a basis for experimental implementations of such type systems. The Pict type system already guarantees correct use of communication channels, in a sense which is described below, and also incorporates subtyping and higher order polymorphism. We will extend the type system to one which supports higher level descriptions of structured communications. This involves implementing some recent ideas in concurrent typing, and investigating how they interact with the existing Pict type system. Although a research language, Pict is complete enough to support serious large scale programming, so we will also develop case studies of practical programming with the new type systems.

When type-theoretic ideas are applied to concurrent and distributed systems rather than sequential systems, types describe communication behaviour rather than functional behaviour, and the purpose of type-checking is to verify that the components of a distributed system have compatible expectations of the patterns of communication between them. Because communication between such components is more complex than the interaction between a function and its arguments, the utility of static typing is correspondingly amplified. As interest continues to grow in safe programming languages for distributed applications, we can expect a matching growth of interest in type systems which can help to ensure safety.

One can seek to develop type systems for process calculi or more realistic programming languages, but in either case the fundamental assumption is the same: that processes communicate with each other by sending and receiving messages on certain channels. Runtime errors may arise when different processes attempt to use a channel or group of channels in incompatible ways, so the idea behind any type system is to avoid this by constraining the use of channels. There are various possible definitions of runtime error, and as always there is a tension between the constraints of typing and the desire for expressiveness in the language. These considerations have led to a range of proposals for concurrent type systems. Most of the type systems surveyed below have been proposed for the π-calculus or similar languages; since Pict is directly based on the π-calculus, these are the type systems which will be our immediate sources of inspiration.

The simplest way of constraining the use of channels is to specify an allowable message type for each channel and ensure that all messages sent or received on a channel have the correct type. Milner's system of *sorts* [23] for the π-calculus [24] takes this approach. Messages consists of tuples of channel names (the key feature of the π-calculus is the use of channel names as data) and the type of a channel is a tuple of types. Thus types have a recursive structure, ensuring that the receiver of a name will use it in the same way as the sender. This system has a notion of principal type, and type inference is possible in a similar way to functional languages [9, 15]. The kind of runtime errors which are avoided are attempts to receive a tuple of the wrong size.

Pierce and Sangiorgi [26] refined Milner's type system by introducing a distinction between input and output channels. The fact that a process respects a certain typing then provides a more accurate description of how it uses channels. This makes it easier to reason about process behaviour, because a process need only work correctly in a well-typed environment. They exploited this observation to prove correctness of Milner's encoding [22] of the λ-calculus into the π-calculus. They also introduced a notion

of subtyping: there is an unrestricted input/output type which is a subtype of both input and output, and can be used anywhere where one of the restricted types is expected.

Another refinement, due to Kobayashi, Pierce and Turner [19], introduces the idea of linear channel types which can only be used once for input and once for output. This idea again allows a more precise description of process behaviour; in typical applications of the π-calculus there are certain channels which are used for a single message only (for example, to return the result of a function call) and it is useful to capture this information as part of a type.

The Pict language [27] has an implementation of the π-calculus at its core, and provides various higher-level constructs which are defined by translation into the π-calculus. The Pict type system uses Pierce and Sangiorgi's input and output channel types, with subtyping; possible types of message include standard data types such as integers, tuples, and records; and these type features are combined with higher order polymorphism. Pict has been used for some substantial programming tasks, and the utility of its type system has been amply demonstrated. The proposed project will build on the success of Pict by extending its type system to include some more recent theoretical ideas.

Even if all channels are used correctly, there are still possibilities for undesirable runtime behaviour. For example, deadlocks can arise from cyclic dependencies among messages: the simplest example is if process P wants to send a message to process Q on channel x then receive a message on channel y, while Q wants to send on channel y before receiving on channel x. Kobayashi [18] has proposed a type system which records information about the order of channel use, and allows some deadlocks of this form to be detected statically. It also classifies channels as reliable or unreliable; the class of reliable channels is subdivided into linear channels, replicated input channels, and mutex channels. This again permits the desired usage of a channel to be specified more precisely, while also recognising that certain channels (the unreliable ones) cannot be described in great detail. Yoshida's system of graph types [33] for the π-calculus also captures information about order of channel use, and enables complex protocols to be described.

Nierstrasz [25] considers objects as processes, which respond to messages (requests for service) received on various channels. He introduces the idea of a regular type, which is a finite state process describing which messages can be accepted in various states. He is interested in subtyping, and uses the notion of request substituability (formulated in terms of a variation of failures [14]) to express the requirement that a subtype must guarantee acceptability of all messages accepted by a supertype.

In each of these type systems, the type assigned to a channel describes a single use of the channel; if that type permits the channel to be used more than once, then each use transmits the same type of information. A somewhat different approach has been taken by Takeuchi, Honda and Kubo [29]. They have proposed a language in which a channel can be used for a *session* consisting of a series of message transmissions of different types (for example, an integer followed by a boolean); this models a situation in which two components establish a private connection and then use it for an extended dialogue. The overall pattern of communication is described by a type, which must be respected by both parties. Furthermore, a session type can describe branching behaviour, in which a choice from a number of options determines the type of the subsequent interaction. Similar ideas are incorporated in the type system proposed by Puntigam [28]. The main difference is that he considers subtyping, motivated by object oriented considerations. Kobayashi *et al.* [19] have shown that a form of session types (which they call *linearized types*) can be encoded in a system combining linear types and recursive types. While such encodings often help one to understand higher level structures, the high level view remains worthy of study in its own right, since it provides a useful abstraction.

The research summarised above shows that types are being used to provide increasingly informative descriptions of system behaviour. The theory of interaction categories, developed by Abramsky, Gay and Nagarajan [1, 2, 4, 5] during the past several years, suggests a way of placing a range of type

systems within a single framework. Interaction categories are categories of typed concurrent processes, in which objects are types and morphisms are processes. The notion of *specification structure* allows the construction of a hierarchy of categories, corresponding to a hierarchy of type systems, each one obtained by adding more detailed information to the types of the last. Types at the bottom of such a hierarchy specify very simple properties such as compatibility of alphabets; types further up specify more complex properties such as deadlock-freedom or liveness. This suggests that it is fruitful to consider types and traditional specifications as facets of the same concept. Although the structure of interaction categories is not yet directly applicable to the π-calculus (name-passing is the technical problem), it may be possible to use the specification structure idea to organize the space of π-calculus type systems.

2.2 Relevance to Beneficiaries

The rapid growth of distributed computing, and in particular mobile code, has led to an increased need for guarantees of security and reliability in programming languages. One approach to addressing this need is to provide languages with appropriate type systems, which support static checking of desirable program properties such as safe execution. Many interesting ideas have emerged from recent research on type systems for concurrent programming, which could usefully be incorporated into real languages. While some have been implemented on a reasonably large scale, for example in Pict, the practicality of others still needs to be demonstrated before they can make their way into the next generation of industrial-strength languages. By experimenting with implementations, both of new type systems and combinations of features from existing type systems, this project will help to move more of these ideas towards commercial application. Because the implementation work will be based on Pict, the results will be directly relevant to existing Pict users, who will be able to assess the benefits of new typing features for their own applications. Further downstream, future language designers will be able to draw on the experience of Pict generally, and the results of this project in particular, when selecting appropriate language and typing features.

2.3 Dissemination and Exploitation

The results of the project will appear as a PhD thesis, and will also be published as journal and conference papers, and technical reports. The implementation work will result in an extended version of the Pict compiler, which will be made available to other researchers for further experimentation.

We hope that the interest shown in the project by Thomsen and Leth at ICL will lead to future collaboration. This could enable us to feed the ideas developed during the project into industrial research programmes.

2.4 Programme

The practical aim of the project is to produce an extension of the Pict language and its type system, so that extended dialogues between processes can be described in a high-level way. We will draw on ideas such as Takeuchi *et al.*'s session types, and Puntigam's type system. There is some existing work in this direction: the linear channel types of Kobayashi *et al.* [19] are able to encode at least some of the features of session types, and Pierce has added linear channel types to Pict (although this addition does not appear in the Pict release). The distinguishing feature of this project is that we will work directly with session types rather than using an encoding, so that type checking can be applied at the right level of abstraction. We will also integrate the new language features smoothly with the existing ones — in other words, we will produce a language *extension* rather than a variant.

In order to experiment with implementing type systems for concurrency, we have chosen to build on the

Pict language rather than designing a new language. There are two reasons for this decision. First, the existing implementation of Pict is available in source form, so by taking it as a starting point we will be able to subtantially reduce the work involved, and proceed straight to the implementation of extensions to the type system. Second, Pict is generally available to the research community, and is being used as the basis for experiments by a number of groups. Our work will contribute to the general development of Pict, and we may eventually be able to unify our extensions with those being studied elsewhere.

The project will involve theoretical as well as practical work. Because we aim to combine new type features with the existing Pict type system, we need to check that they work well together — for example, that subtyping and polymorphism can coexist with session types, and that type checking is still possible.

The main source of manpower for the project will be a PhD student, so the programme of work reflects the structure of a PhD programme.

Task 1 The student will spend most of the first year becoming familiar with the research area. He will need to learn the π-calculus, and type theory, both in general and as it applies to concurrency. He will also learn Pict, and develop an understanding of its compiler.

Output 1 The student will write a survey of existing type systems for concurrency. We will develop a suite of example Pict applications in which enhanced type systems would be useful.

Task 2 In the second year we will begin to modify the Pict language and compiler. The first step will be to disable some of the more complex features of the Pict type system (principally polymorphism and subtyping) and add a version of session types. We will also need to add some higher-level syntax to Pict, to indicate when session types are to be used. In accordance with the general design philosophy of Pict, the new language constructs will be defined by translation into the core language. The functionality provided by, for example, establishment of a private channel for a session, is already available in untyped core Pict; the point is that the modified type system will allow that functionality to be exploited safely. We expect to complete this initial modification six months into the second year.

Output 2 We will produce a definition of the modified language in the same style as the current Pict definition. We will also produce a technical report describing the language, its design and implementation, and giving examples of its use. The examples will be based on the applications developed during the first year. The results at this point should be suitable for presentation at a conference such as PARLE.

Task 3 In the next stage of the project we will restore the aspects of the Pict type system which were removed during the first phase of implementation work. In order to do this, we will need to study the interaction between session types, subtyping and higher order polymorphism, separately at first and then in combination. This is the main theoretical challenge of the project. Because a session type combines a number of message types, integrating the formation of session types with higher order polymorphism is likely to be technically difficult. At this point we will draw on the work of Takeuchi *et al.*, who considered session types in conjunction with principal type schemes (although not with higher order polymorphism as found in Pict) [29], and Puntigam, whose type system incorporates subtyping and shares some of the properties of session types [28]. This stage of the project will extend to six months into the third year. Dr Gay will begin to look at the theoretical issues earlier in the project, with the intention of identifying the key problems. The student will begin to work with Dr Gay on the theoretical issues, while Task 2 is in progress.

Output 3 We will produce a number of versions of Pict, progressively adding typing features, and develop example programs at each stage. The final output will be a version of Pict including session types, subtyping and higher order polymorphism. We will also produce a revised language definition and a suite of example programs.

Task 4 The final stage of the project will consist of writing the PhD thesis. This will make use of the documents produced earlier in the project.

Output 4 The thesis will be made available as a technical report, and published (in a suitably modified form) as a journal paper or papers. The final modified Pict system will also be made available electronically.

Most of the implementation work will be carried out by the student. Dr Gay will contribute to the theoretical study of the interaction between session types and other features. Our contact with Thomsen and Leth at ICL will help to keep our examples of programming with enhanced type systems in touch with practical concerns.

The transition from a theoretical type system to a practical implementation is by no means trivial, and careful design decisions need to be made. For example, Pierce (personal communication) has reported that naively adding linear channel types to Pict resulted in a major expansion of the language since linear and non-linear forms were required for every construct. There may be an analogous effect with session types, if a distinction is made between communication on a private channel (described by a session type) and more general communication. One approach to solving such problems is through type inference, which has been used to good effect in functional languages whose type systems capture linearity information. For example, Turner *et al.* [31] use a type inference algorithm to detect linear usage of expressions in a functional language; this information can be used to justify program optimisations (one of the motivations for the linear type system of Kobayashi *et al.*) but does not need to be explicitly provided by the programmer. Other linear functional languages have also used type inference to avoid annotating the syntax with linearity information [21, 32]. In the context of session types, it may be that a suitable type inference algorithm could determine which channels correspond to sessions and which to one-off communications. We are conscious of the need to produce a syntax which is as simple and flexible as possible, in order that the extended language can be used comfortably.

There is plenty of scope for extending this research beyond the programme described above. The most obvious possibility is to implement other proposed type systems for the π-calculus, again as extensions to Pict. It would be interesting to experiment with Yoshida's graph types [33] and Kobayashi's partially deadlock-free types [18]. It is very likely that other researchers will continue to propose refined type systems during the course of our project, and we will be in a good position to apply our experience to implementing them as well. Another possibility is to apply the idea of a specification structure [2], arising from work on interaction categories, to type systems in Pict. A specification structure defines a new type system as a refinement of an existing, less informative system; the result in general is a hierarchy of type systems providing increasingly detailed behavioural information. If a range of type systems for Pict could be structured in this way, it may be possible to develop a versatile type checker which can operate at any desired level of the hierarchy, thus allowing the programmer to choose the amount of static analysis being applied to a program.

Understanding the Pict compiler in sufficient depth to produce our proposed modifications is a significant challenge. However, we will be helped by the following factors.

- The Pict language has a formal definition, and the compiler follows the structure of that definition rather closely.
- We will only need to understand and modify the front end, including the parser and type-checker, and not the code-generating back end.
- We plan to share our experience of Pict development with Sewell's group in Cambridge, who are also experimenting with modifications of the compiler. The developers of Pict are willing to provide some help; in particular, a short visit to David Turner in Edinburgh will be useful in the early stages of our work.

The proposed project is ideal for a PhD programme. It will take place in the context of an established research area, whose importance is clear and certain to continue. A structured research programme has been identified, containing a number of technical challenges which the student will need to address. The project will combine theory and practice in such a way that the student will have the opportunity to develop a valuable, and somewhat uncommon, blend of expertise.

We have mentioned the interest of Thomsen and Leth (ICL) in this project. Since their work is based on the Facile language [30], we should now clarify the relationship between Facile and Pict and explain their interest. Facile is a multi-paradigm programming language combining functional and concurrent programming. It is intended for programming reactive and distributed systems, in particular the construction of systems based on the emerging mobile agent principle. Like Pict, it is based on a sound theory — in this case, an integration of functional language theory and concurrency theory. It is more complex than Pict, which is based on concurrency theory alone, but the two have been formally related [6]. Thomsen and Leth are interested in adding refined type systems to Facile, and the results of our work with Pict will be relevant as far as types describing concurrent behaviour are concerned. Facile has been used for a number of case studies, and we hope to use these case studies as a source of realistic examples of programming with our developments of Pict. While we are not proposing direct collaboration with Thomsen and Leth at this stage, we hope that this project will reveal opportunities for future collaboration.

2.5 Management and Resources

Management The project will be managed by Dr Gay, who will also supervise the funded PhD student. The management requirements of this project are to guide and supervise the student's research along the lines set out in the rest of this proposal, to collaborate with the student in a way which achieves the goals of the project but also allows him sufficient scope for independent work, to ensure that the student develops a sufficiently broad knowledge of the field within which the project is situated, and to ensure that the PhD thesis is completed on time.

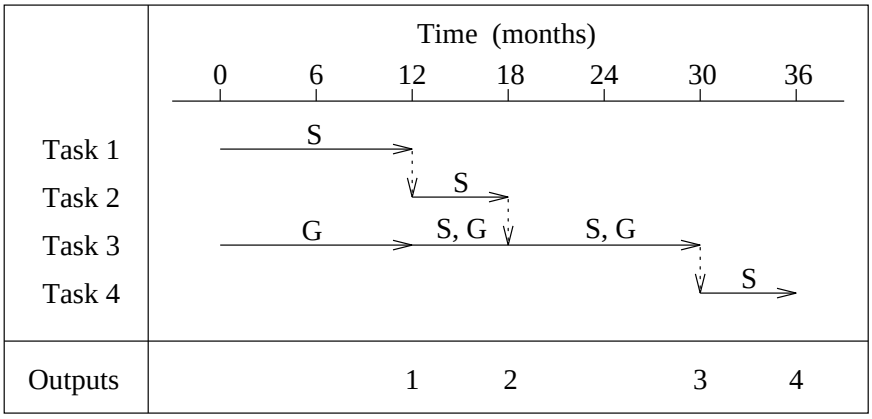
Manpower Funding is sought for a PhD studentship — this is the only manpower for which support is being requested. The PhD student will be supervised by Dr Gay, who will take an active part (4 hours per week) in the proposed programme of research.

Equipment We have requested funding for a Sun Ultra 1 Model 140 workstation for the use of the student, since he will be carrying out a substantial amount of implementation work. We have also included a budget for maintenance and consumables.

Exceptional item The exceptional item consists of the fees associated with the PhD studentship.

Travel We have budgeted for two conference visits, one to PARLE (Parallel Languages and Architectures Europe) and one to POPL (usually held in the USA). These conferences would be suitable forums for presenting the results of the project. We have also budgeted for a number of short visits to Cambridge, so that we can discuss Pict compiler modification experience with Sewell's group, and for three longer trips to Edinburgh, which will combine visits to Honda and Yoshida with visits to Turner for Pict discussion.

3 Project Plan



S = work carried out by the student; G = work carried out by Dr Gay.

Task and output numbers relate to the work plan in Section 2.4.

Vertical arrows indicate results fed between tasks.

References

[1] S. Abramsky, S. J. Gay, and R. Nagarajan. Interaction categories and foundations of typed concurrent programming. In M. Broy, editor, *Deductive Program Design: Proceedings of the 1994 Marktoberdorf International Summer School*, NATO ASI Series F: Computer and Systems Sciences. Springer-Verlag, 1995.

[2] S. Abramsky, S. J. Gay, and R. Nagarajan. Specification structures and propositions-as-types for concurrency. In G. Birtwistle and F. Moller, editors, *Logics for Concurrency: Structure vs. Automata—Proceedings of the VIIIth Banff Higher Order Workshop*, volume 1043 of *Lecture Notes in Computer Science*. Springer-Verlag, 1996.

[3] S. Abramsky. Domain theory in logical form. *Annals of Pure and Applied Logic*, 51:1–77, 1991.

[4] S. Abramsky. Interaction Categories (Extended Abstract). In G. L. Burn, S. J. Gay, and M. D. Ryan, editors, *Theory and Formal Methods 1993: Proceedings of the First Imperial College Department of Computing Workshop on Theory and Formal Methods*, pages 57–70. Springer-Verlag Workshops in Computer Science, 1993.

[5] S. Abramsky. Interaction Categories and communicating sequential processes. In A. W. Roscoe, editor, *A Classical Mind: Essays in Honour of C. A. R. Hoare*, pages 1–15. Prentice Hall International, 1994.

[6] R. Amadio, L. Leth, and B. Thomsen. From a concurrent λ-calculus to the π-calculus. In *Proceedings of FCT'95 — Tenth International Conference on Fundamentals of Computation Theory*, volume 965 of *Lecture Notes in Computer Science*. Springer-Verlag, 1995.

[7] J.-P. Banâtre and D. Le Métayer. The GAMMA model and its discipline of programming. *Science of Computer Programming*, 15:55–77, 1990.

[8] S. J. Gay. Interaction nets. Diploma in Computer Science Dissertation, University of Cambridge Computer Laboratory, 1991.

[9] S. J. Gay. A sort inference algorithm for the polyadic π-calculus. In *Proceedings, 20th ACM Symposium on Principles of Programming Languages*. ACM Press, 1993.

[10] S. J. Gay. Combinators for interaction nets. In C. L. Hankin, I. C. Mackie, and R. Nagarajan, editors, *Theory and Formal Methods 1994: Proceedings of the Second Imperial College Department of Computing Workshop on Theory and Formal Methods*. Imperial College Press, 1995.

[11] S. J. Gay. *Linear Types for Communicating Processes*. PhD thesis, University of London, 1995.

[12] S. J. Gay and C. L. Hankin. A program logic for Gamma. In J.-M. Andreoli, C. L. Hankin, and D. L. Métayer, editors, *Coordination Programming: Mechanisms, Models and Semantics*. Imperial College Press, 1996.

[13] S. J. Gay and C. L. Hankin. Gamma and the logic of transition traces. In A. Edalat, S. Jourdan, and G. McCusker, editors, *Advances in Theory and Formal Methods of Computing*. Imperial College Press, 1997.

[14] C. A. R. Hoare. *Communicating Sequential Processes*. Prentice Hall, 1985.

[15] K. Honda and V. Vasconcelos. Principal typing scheme for polyadic π-calculus. Unpublished report., 1992.

[16] A. Johnstone. `rdp` — a recursive descent compiler compiler. Technical Report CSD-TR-95-03, Department of Computer Science, Royal Holloway, University of London, March 1995.

[17] A. Johnstone. `rdp_supp` — support routines for the `rdp` compiler compiler. Technical Report CSD-TR-95-04, Department of Computer Science, Royal Holloway, University of London, March 1995.

[18] N. Kobayashi. A partially deadlock-free typed process calculus. In *Proceedings, Twelfth Annual IEEE Symposium on Logic in Computer Science*. IEEE Computer Society Press, 1997.

[19] N. Kobayashi, B. C. Pierce, and D. N. Turner. Linearity and the pi-calculus. In *Proceedings, 23rd ACM Symposium on Principles of Programming Languages*, 1996.

[20] Y. Lafont. Interaction nets. In *Proceedings of the Seventeenth ACM Symposium on Principles of Programming Languages*, pages 95–108. ACM, ACM Press, January 1990.

[21] I. Mackie. Lilac : A functional programming language based on linear logic. *Journal of Functional Programming*, 4(4):1–39, October 1994.

[22] R. Milner. Functions as processes. In *Proceedings of ICALP 90*, volume 443 of *Lecture Notes in Computer Science*, pages 167–180. Springer-Verlag, 1990.

[23] R. Milner. The polyadic π-calculus: A tutorial. Technical Report 91-180, Laboratory for Foundations of Computer Science, Department of Computer Science, University of Edinburgh, 1991.

[24] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, I and II. *Information and Computation*, 100(1):1–77, September 1992.

[25] O. Nierstrasz. Regular types for active objects. *ACM Sigplan Notices*, 28(10):1–15, October 1993.

[26] B. Pierce and D. Sangiorgi. Types and subtypes for mobile processes. In *Proceedings, Eighth Annual IEEE Symposium on Logic in Computer Science*. IEEE Computer Society Press, 1993.

[27] B. C. Pierce and D. N. Turner. Pict: A programming language based on the pi-calculus. Technical report in preparation, 1997.

[28] F. Puntigam. Synchronization expressed in types of communication channels. In *Proceedings of Euro-Par 96*, volume 1123 of *Lecture Notes in Computer Science*. Springer-Verlag, 1996.

[29] K. Takeuchi, K. Honda, and M. Kubo. An interaction-based language and its typing system. In *Proceedings of the 6th European Conference on Parallel Languages and Architectures*, number 817 in Lecture Notes in Computer Science. Springer-Verlag, 1994.

[30] B. Thomsen, L. Leth, and T.-M. Kuo. A Facile tutorial. In *Proceedings of CONCUR'96 — Seventh International Conference on Concurrency Theory*, volume 1119 of *Lecture Notes in Computer Science*. Springer-Verlag, 1996.

[31] D. N. Turner, P. Wadler, and C. Mossin. Once upon a type. In *7th International Conference on Functional Programming and Computer Architecture*, 1995. Also technical report TR-1995-8, Computing Science Department, University of Glasgow.

[32] P. Wadler. Is there a use for linear logic. Technical report, University of Glasgow, 1990.

[33] N. Yoshida. Graph types for monadic mobile processes. In *Proceedings of the Sixteenth Annual Conference on Foundations of Software Technology and Theoretical Computer Science*, Lecture Notes in Computer Science. Springer-Verlag, 1996.