Can Actors and π-Agents Live Together?*

Ugo Montanari[1] and Carolyn Talcott[2]

[1] Computer Science Laboratory, SRI International, Menlo Park, ugo@csl.sri.com
[2] Computer Science Department, Stanford University clt@sail.stanford.edu

Abstract. The syntax and semantics of actors and π-agents is first defined separately, using a uniform, "unbiased" approach. New coordination primitives are the added to the union of the two calculi which allow actors and π-agents to cooperate.

1 Modeling Actors and Agents

The syntax and semantics of actors and π-agents are first defined separately, using a uniform, "unbiased" approach. Since we aim at modeling concurrent distributed systems, and thus we are interested in asynchronous behavior, we choose for comparison with actors an asynchronous version of the π-calculus.

In the paper, the behavior of both actors and π-agents is defined by certain logic sequents called *tiles*. A tile is a rewrite rule which describes a possible evolution of a part s of the system which is matched by it. In addition, a tile also describes the evolution of the *interfaces* of s with the rest of the system. Thus two parts s and s' sharing an interface can be rewritten only by tiles which agree on the evolution of the common interface. This restriction introduces a powerful notion of synchronization[3] among tiles, and also makes possible to see the synchronization of two tiles as a (larger) tile. Eventually, all the possible evolutions are obtained by the repeated composition (synchronized, or in sequence, or in parallel) of certain small *basic* tiles called *rewrite rules.*

In the case of actors and π-agents, it is convenient to take a coordination [19] point of view and to distinguish between the behavior of agents in isolation and the behavior of *coordinators*, i.e. of system components whose role is to connect agents and to control their behavior. This approach allows us to abstract from the behavior *in the small* of agents, which is presented in a state transition, syntax-independent form, and to focus on the behavior of coordinators, which are the most characteristic feature of distributed systems. Correspondingly, we distinguish between two kinds of rewrite rules: *activity* rules and *coordination* rules. An activity rule describes an evolution of a single sequential agent, and may

* Research supported in part by CNR Integrated Project *Metodi e Strumenti per la Progettazione e la Verifica di Sistemi Eterogenei Connessi mediante Reti di Comunicazione*; and Esprit Working Groups *CONFER2* and *COORDINA*. The first author is on leave from Dipartimento di Informatica, Pisa, Italy.

[3] Even in an asynchronous system, synchronization is required to model the reception of messages and the extrusion of names.

produce some action at its interface with the rest of the system. A coordination rule describes an evolution of a coordinator, and may both require certain actions from the agents it controls, and produce actions for a coordinator operating at a higher level.

For actors and π-agents, the interfaces with the rest of the system contain the free names of the agent, or, equivalently, the acquaintances (including self) of the actor. We call both of them names. An important difference with the ordinary semantics of both calculi is that in our approach names have only a local scope. Thus if in a particular subsystem there are n names, we can just denote them with $x_1, x_2, \ldots, x_n$. This choice makes the handling of names much easier, especially in the presence of bound (restricted) names and of name extrusion steps: it avoids α-conversion, infinite branching and in general the need of making provisions for an infinite number of possible names when connecting with the external world.

In addition to names, the interfaces contain also *events*. Events are the mechanism we use for establishing concurrency control in a distributed system. Agents and messages include references to the events which generated them, and to the previous events which caused these events. For instance in the *event diagram* semantics [11, 32], when a message is received, a new event is created, and pointers to it and its causes are made available to all the components spawned by the step. The *causes* of this new event are the events both the message and the receiving agent pointed to. The causal relation determines the orderings in which the events can happen: all the sequential orderings compatible with the causal relation are possible, and they correspond to the same *concurrent* computation.

In the representation of agents, actions and events, the key notion is *sharing*. In fact, the only role of a name is to specify which agents share it, and similarly several events may share the same cause. Sharing is a well-studied notion from a formal point of view, in particular for the subterms shared by a term. For instance, terms can be broken into the parallel and sequential composition of term constructors and basic substitutions, modulo certain axioms. Sharing in its purest form is then represented by the basic substitution ∇ from one variable to two values:

$$\nabla : 1 \to 2 = \{y_1 := x_1, y_2 := x_1\}.$$

However, in the algebra of terms and substitutions, sharing is not a first class component, in the sense that it can be freely removed by copying the shared subterm. In the paper, instead of terms we use *term graphs*. They are more expressive than terms, since terms graphs with a different amount of sharing among subterms are actually different. As a consequence, in term graphs, the ∇ operator becomes a basic constructor.

2 Interoperability of Actors and Agents

In the ordinary syntax of actors and π-agents, we have three configuration operators: parallel composition, restriction and renaming. Parallel composition is very powerful, since references to the same name on both operands are automatically identified. In our approach, since names are only local, parallel composition considers all names as different and yields the union of the two subsystems without establishing any connection between them. Names are actually identified by the Δ *matching* operator, which is thus our second coordinator. The Δ operator is analogous but opposite to ∇, since it merges two names, or two events, into one, rather that creating two instances of the same variable. It replaces both variable substitution $\_ [\_ / \_]$ and parallel composition $\_ \mid \_$, which turn out, somewhat surprisingly, to have analogous meanings.

Renaming, which is of difficult interpretation in a distributed setting, becomes useless and is discarded[4]. Restriction is mantained, essentially with the same meaning.

The main difference between actor calculus and π-calculus resides, in our setting, in the different typing of names and in the different versions of Δ's. The free names of a π-agent are all typed c (for *channel*) and thus there is only one Δ, which we call Δ^c. Given an actor, its name is typed a, while its acquaintances are typed r (for *reference*). Also, all the acquaintances of a message, including its addressee, are typed r. There are only two Δ's: Δ^r accepting two references and yielding a reference, and Δ^a accepting an actor and a reference and yielding an actor. There is no Δ accepting two actors: this restriction fully enforces the uniqueness of actor names.

The behaviour of a Δ is determined by its permeability to input/output actions and to synchronization. It is easy to see that Δ^c must be permeable to everything, while Δ^r cannot be presented with any input action, and thus cannot synchronize either. The most interesting case is Δ^a, which is permeable to input and synchronization, but impermeable to output. The rationale under this restriction is that a message cannot exit a system if its addressee is inside the system.

When connecting actors and π-agents, it is reasonable to assume that a coordinator accepting a channel and a reference, called Δ^{cr}, behaves like Δ^c and thus yields a channel. On the other hand, a coordinator accepting a channel and an actor may behave either as Δ^c or as Δ^a. We thus introduce two different coordinators for the two alternatives, Δ^{ca} and Δ^{ac} respectively. Considering their behavior, we also sometimes overload coordinator denotations, calling Δ^c, Δ^{cr} and Δ^{ca} simply as Δ^c, and calling as Δ^a both Δ^a and Δ^{ac}. The permeability of the various Δ coordinators is summarized in Table 1.

Being both calculi equipped with mobility, the amount of name sharing established at configuration time can be modified, actually only increased, at run time. In our setting, new Δ's can be created essentially only during synchronizations[5],

[4] A weak notion of renaming, *permutations*, is used. They correspond exactly to substitutions of the form $\rho : 2 \to 2 = \{y_1 := x_2, y_2 := x_1\}$. However, they just describe a "wire twisting", they have no coordinating role, and no rewrite rule matches a substitution.

[5] An activity tile describing the forking of some actor or some π-agent may also create new Δ's. However they will connect only homogeneous subsystems, either both actors or both π-agents.

Connector	Δ^a	Δ^r	Δ^c	Δ^{cr}	Δ^{ca}	Δ^{ac}
Type	$ar \to a$	$rr \to r$	$cc \to c$	$cr \to c$	$ca \to c$	$ac \to a$
Permeability to :						
output	n	y	y	y	y	n
input	y	$-$	y	y	y	y
synchronization	y	$-$	y	y	y	y

Table 1. Permeability of coordinators for name sharing.

to equate the formal and the actual parameters. Within π-agents, only Δ^c's can be created. Within actors, while an ordinary synchronization will always create a Δ^r, a synchronization where the output action is extruding (similar to a *Close* step for π-calculus) will create a Δ^a and a restriction. How the new Δ^{cr}, Δ^{ca} and Δ^{ac} will behave? We think it is reasonable to assume that, when choice is given, they will create Δ's of the same type as themselves. For instance if a coordinator Δ^{ca} has been connected during the configuration phase to an actor and a channel name, it means that the actor has been given a lower priority, in the sense that its privilege of being the unique receptor of that name has been given up. It is thus reasonable that if an input object of type c had to be connected with an extruding output object of type a, a Δ^{ca} should also be used.

Since parallel composition and restriction have the same meaning for both actors and π-agents, the above case analysis for the Δ's covers all the possibilities of interconnections. In particular, the existence of two possible coordinators Δ^{ca} and Δ^{ac} allows for quite a freedom in putting together two collections of actors and π-agents. Notice that our distincion of the roles of parallel composition (system union vs. name identification) plays a key role in our construction. In fact, had parallel composition kept both roles, two parallel composition operators would have been introduced, one keeping the unique receptor privilege for actors, the other not keeping it. However the choice would have been the same for all names, while in our approach different choices are available for every pair of names.

3 Related Work

The actor model [20, 6, 1, 2] is one of the first and best known models for concurrent distributed systems and consists of independent computational agents which interact solely via asynchronous message passing. Semantic foundations for actor computation have been given in [4, 30, 31, 32]. An approach to specifying and implementing mechanisms for coordination of actors based on reflection is described in [3].

The π-calculus [27] is one of the best studied examples of *mobile* process calculi, namely calculi in which the communication topology among processes can dynamically evolve when computation progresses. The asynchronous version of the π-calculus has been introduced in [8, 21] and studied in [5].

The tile model, introduced in [16], is described in general terms in [17, 18]. Tiles are much like SOS inference rules [29], but they can be composed horizontally, vertically and in parallel to build larger proof steps. Tile systems generalize Kim Larsen and Liu Xinxin *context systems* [22] since they allow for more general rule formats. The tile model also extends rewriting logic [23, 24, 25] (in the nonconditional case), since it takes into account rewritings with side effects and rewriting synchronization. Tile systems can be seen as double categories [13] and tiles themselves as double cells. They can be equipped with observational equivalences and congruences.

The combined used of tiles and term graphs [7, 12] for modeling asynchronous π-calculus and CCS with locations [9] has been described in [14, 15]. Also coordination models equipped with flexible synchronization primitives are presented in [28, 10]. Ongoing work [26] aims at translating the tile model into rewriting logic, in order to take advantage of important features of rewriting logic, like execution strategies and reflective logics, and to employ its existing implementations.

References

1. G. Agha, *Actors: A Model of Concurrent Computation in Distributed Systems*, MIT Press 1986.
2. G. Agha, *Concurent Object Oriented Programming*, CACM 33 (9), pp.125-141, 1990.
3. G. Agha. Abstracting interaction patterns: A programming paradigm for open distribute systems. In E. Najm and J-B. Stefani, editors, *Formal Methods for Open Object-based Distributed Systems*, pages 135–153. Chapmn & Hall, 1997.
4. G. Agha, I.A. Mason, S.F. Smith and C.L. Talcott, *A Foundation for Actor Computation*, J. of Functional Programming, 7, pp.1-72, 1997.
5. R. Amadio, I. Castellani, D. Sangiorgi, *On Bisimulations for the Asynchronous π-calculus*, CONCUR'96, LNCS, 1996.
6. Henry G. Baker and Carl Hewitt. Laws for communicating parallel processes. In *IFIP Congress*, pages 987–992. IFIP, August 1977.
7. H.P. Barendregt, M.C.J.D. van Eekelrn, J.R.W. Glauert, J.R. Kennaway, M.J. Plasmeijer, M.R. Sleep, *Term Graph Reduction*, Proc. PARLE, Springer LNCS 259, 141–158, 1987.
8. G. Boudol, Asynchrony and the π-calculus (note), *Rapport de Recherche 1702*, INRIA Sophia-Antipolis, May 1992.
9. G. Boudol, I. Castellani, M. Hennessy and A. Kiehn, *Observing Localities*, Theoretical Computer Science, 114: 31–61, 1993.
10. R. Bruni and U. Montanari, *Zero-Safe Nets, or Transition Synchronization Made Simple*, in: Catuscia Palamidessi, Joachim Parrow, Eds, EXPRESS'97, ENTCS, Vol. 7, `http://www.elsevier.nl/locate/entcs/volume7.html`.
11. W. D. Clinger. *Foundations of Actor Semantics*. PhD thesis, MIT, 1981. MIT Artificial Intelligence Laboratory AI-TR-633.
12. A. Corradini, F. Gadducci, *A 2-Categorical Presentation of Term Graph Rewriting*, Proc. CTCS'97, Springer LNCS, to appear, 1997.

13. C. Ehresmann, *Catégories Structurées*: I and II, Ann. Éc. Norm. Sup. 80, Paris (1963), 349-426; III, Topo. et Géo. diff. V, Paris (1963).
14. G.L. Ferrari and U. Montanari, *A Tile-Based Coordination View of Pi-Calculus*, in: Igor Prvara, Peter Ruzicka, Eds., Mathematical Foundations of Computer Science 1997, Springer LNCS 1295, 1997, pp. 52-70.
15. G. Ferrari, U. Montanari, *Tiles for Concurrent and Located Calculi*, in: Catuscia Palamidessi, Joachim Parrow, Eds, EXPRESS'97, ENTCS, Vol. 7, `http://www.elsevier.nl/locate/entcs/volume7.html`.
16. F. Gadducci, *On the Algebraic Approach to Concurrent Term Rewriting*, PhD Thesis, Università di Pisa, Pisa. Technical Report TD-96-02, Department of Computer Science, University of Pisa, 1996.
17. F. Gadducci, U. Montanari, *The Tile Model*, in: Gordon Plotkin, Colin Stirling, and Mads Tofte, Eds., Proof, Language and Interaction: Essays in Honour of Robin Milner, MIT Press, to appear. Paper available from `http://www.csl.sri.com/ ugo/festschrift.ps`.
18. F. Gadducci and U. Montanari, *Tiles, Rewriting Rules and CCS*, in: *J. Meseguer, Ed., Procs. Rewriting Logic and Applications*, First International Workshop, ENTCS, Vol. 4 (1996), `http://www.elsevier.nl/locate/entcs/volume4.html`.
19. D. Garlan, D. Le Métayer, Eds., *Coordination Languages and Models*, LNCS 1282, 1997.
20. C. Hewitt, *Viewing Control Structures as Patterns of Passing Messages*, J. of Artificial Intelligence, 8(3), pp.323-364, 1977.
21. K. Honda, M. Tokoro, *An Object Calculus for Asynchronous Communication*, In: M. Tokoro, O. Nierstrasz, P. Wegner, Eds., Object-Based Concurrent Computing, Springer LNCS 612, 21–51, 1992.
22. K.G. Larsen, L. Xinxin, *Compositionality Through an Operational Semantics of Contexts*, in Proc. ICALP'90, LNCS 443, 1990, pp. 526-539.
23. J. Meseguer, *Conditional Rewriting Logic as a Unified Model of Concurrency*, Theoretical Computer Science **96**, 1992, pp. 73-155.
24. J. Meseguer, *Rewriting Logic as a Semantic Framework for Concurrency: A Progress Report*, in: U. Montanari and V. Sassone, Eds., *CONCUR'96: Concurrency Theory*, Springer LNCS 1119, 1996, 331-372.
25. J. Meseguer, Ed., *Procs. Rewriting Logic and Applications*, First International Workshop, ENTCS, Vol. 4 (1996), `http://www.elsevier.nl /locate/entcs/volume4.html`.
26. J. Meseguer and U. Montanari, *Mapping Tile Logic into Rewriting Logic*, paper submitted for publication. Draft available from `http://www. csl.sri.com/ ugo/wadt.ps`.
27. R. Milner, J. Parrow, D. Walker, *A Calculus of Mobile Processes* (parts I and II), Information and Computation, 100:1–77, 1992.
28. U. Montanari, F. Rossi, *Graph Rewriting and Constraint Solving for Modelling Distributed Systems with Synchronization*, in: Paolo Ciancarini and Chris Hankin, Eds., Coordination Languages and Models, LNCS 1061, 1996, pp. 12-27. Full paper submitted for publication available from `http://www.csl.sri.com/ ugo/graphs.ps`.
29. G. Plotkin, *A Structural Approach to Operational Semantics*, Technical Report DAIMI FN-19, Computer Science Department, Aarhus University, 1981.
30. C.L.Talcott, *An Actor Rewriting Theory*, in: *J. Meseguer, Ed., Procs. Rewriting Logic and Applications*, First International Workshop, ENTCS, Vol. 4 (1996), `http://www.elsevier.nl/locate/entcs/volume4.html`.

31. C. L. Talcott. Interaction semantics for components of distributed systems. In E. Najm and J-B. Stefani, editors, *1st IFIP Workshop on Formal Methods for Open Object-based Distributed Systems, FMOODS'96*, 1996. proceedings published in 1997 by Chapman & Hall.
32. C. L. Talcott. Composable semantic models for actor theories. In T. Ito M. Abadi, editor, *Theoretical Aspects of Computer Science*, number 1281 in Lecture Notes in Computer Science, pages 321–364. Springer-Verlag, 1997.