

Indirect interaction and decentralized coordination

David Keil, Dina Goldin

Computer Science & Engineering Department, University of Connecticut
371 Fairfield Rd., Unit 1155, Storrs, CT 06269-3155, USA
{dkeil, dqg}@engr.uconn.edu

Abstract. We distinguish two forms of interaction: *direct*, where *messages* are sent to pre-specified recipients, and *indirect*, consisting of *persistent, observable state changes*, where the recipients are any computing agents that observe the changes. Either type of interaction may occur in coordination, whose hallmarks are *anonymity* and *asynchrony*. While only the first type of interaction has been formalized as part of concurrency theory, our paper focuses on the role of the second. We argue that in self-organizing systems lacking centralized coordination, indirect interaction is needed to achieve anonymity and asynchrony. Therefore, coordination models for decentralized self-organizing systems must explicitly represent indirect interaction in order to be complete. In particular, *open* computational systems, characterized by mobility and dynamic reconfiguration, and operating in unpredictable environments, require decentralized approaches and hence coordination models that support indirect interaction.

1 Introduction

Coordination models were developed to complement “computational models” by providing a “glue” that connects computational activities. A coordination language gives support or form to a coordination model by furnishing operations to create, and enable interaction among, those activities [gel-car92]. From early on, work in coordination languages closely associated coordination with *open systems* where such distributed activities occur [kah-mil88]. This paper associates itself with earlier contributions [wegner96, kei-gol03, arbab98, zam-par02], that have linked coordination, open systems, and models of interactive computing.

It has long been recognized that interaction in such open systems often takes the form of creating “persistent objects” rather than sending transient messages [cargel89]. Communication among modules of single-thread computer programs, on the other hand, like much human communication, is direct, via *message passing*, in which the sender and recipient are identified to each other following a *handshake*. In concurrency theory, the message-passing metaphor has been adopted as a universal model, formalized in *process algebras* [milner93]. The underlying assumption of this predominant conceptual framework is that message passing completely captures interaction.

A central argument of this paper is that message passing does *not* capture some of the kinds of interaction with which coordination is most concerned.

Whereas handshake implies *synchrony* and sender knowledge of the recipient; *coordination* of autonomous and semi-autonomous computing agents often involves *asynchrony* (no handshake) and *anonymity* (no pre-identified destination). Coordination languages such as Linda [gel-car92] and Manifold [arbab98] rely on *tuple spaces* and *channels*, respectively, as coordination media, to produce the effect of anonymity and asynchrony. In the case of Manifold, a central coordinating agent communicates with computational activities synchronously and by message passing, but the computational entities linked in this way appear to be interacting asynchronously and anonymously.

Under conditions of rapid and unpredictable evolution of the environment in an *open* system (Section 4), a computing entity may be required to *adapt* itself, using the full concurrent computing power of its components, which must interact locally with each other. Examples are to be found in nature [kei-gol03, resnick94, bon-the00, ken-be01]. These *self-organizing* systems are by definition *decentralized*. In such self-reconfiguring systems, agents (components) interact by modifying their environments, not by exchanging messages with predetermined recipients.

An environment with persistent state can be altered by agents, thereby serving as the *medium* for interaction among the agents. This is known as *stigmergy*, a feature of natural systems in which agents' behavior is shaped by interactions with unknown other agents, as occurs in ant colonies [kei-gol03].

To express this in another way, for systems to exhibit a coherent behavior in the absence of a centralized coordinator, the typically mobile components must interact based on *where* they are (by locality), not *who* they are (by identifiers). Hence the interaction in such systems must be indirect. It follows that coordination models limited to a process-algebra framework, based on message passing, are incomplete. A theory of open systems must therefore incorporate *indirect* forms of interaction, which rely on *persistence* and *observability* of changes in the environment.

Outline. In Section 2 we define *interaction*, direct and indirect. The second type enables anonymous and asynchronous communication. We argue in Section 3 that *coordination* addresses a need for anonymity and asynchrony. Section 4 defines *openness* as a hierarchy of notions, the highest of which applies to systems most likely to require coordination. We show next that self-organized behavior and decentralized coordination in open systems require *indirect* communication to enable anonymity and asynchrony (Section 5). Finally we offer conclusions, including a claim that, to be adequate for design of open systems, models of coordination must explicitly represent indirect interaction (Section 6).

2 Indirect interaction

Here we distinguish and precisely define two forms of interaction, *direct* and *indirect*. We explain why a *persistent environment* enables indirect interaction and why *asynchrony* and *anonymity* are features of such interaction.

2.1 Direct interaction vs. algorithmic computation

This subsection defines interaction in general, interaction in its direct variant, and Milner's "new conceptual framework" for concurrency theory, and contrasts interactive to algorithmic computation.

Algorithms are characterized by a strict time ordering, in which finite input is followed by computation, which is followed by finite output. By contrast, interactive computation is characterized by the following features:

- (a) dynamic stream input/output;
- (b) interleaving of inputs and outputs during computation;
- (c) history-dependent dynamic behavior of the computing agent [wegner98].

Definition 2.1. *Interaction* is the ongoing two-way or multiway exchange of data among computational entities, such that the output of one entity may causally influence the later outputs of the another entity.

Our definition borrows from [wegner98]. Interaction is known under other names, such as *communication* and *concurrency*.

Algorithmic computation is associated with structured programming, batch systems, and closed computational systems; interactive computing is associated with object-oriented programming, graphical user interfaces, networked and distributed systems, and emerging open-systems formalisms [wegner97]. Whereas algorithmic computation executes a string-based mapping, interactive computation performs a stream-based transduction. Not only may output be a function of input, but I/O interleaving enables an output to influence a later input [wegner97].

[Milner75] noted the distinction between algorithms, which compute functions, and interactive systems, which do not. The *compositional* semantics of sequential programs are lost under concurrency. Whereas sequential processes cannot mutually interfere, concurrent ones may do so, and the semantic mapping from memories to memories does not apply in the presence of concurrency [milner93].

The notion of message passing provides the basis for a definition of one kind of interaction.

Definition 2.2. *Direct interaction* is interaction via messages; the identifier of the recipient is specified in the message.

Direct interaction may be *synchronous* or *asynchronous*. In the case of asynchrony in direct interaction, such as email, a computing agent synchronizes with an intermediate shared resource to receive or send a message.

2.2 Indirect interaction and persistence

Persistence is fundamental to most computational systems, including dynamic real-world-like environments and components that operate in them. Examples are the values of global variables, the states of objects and databases, the data stored in files, and the evolving states of the real world and what is in it. Due to persistence, multiple computing agents may interact with their common environment in such a way that they interact *indirectly* with each other, by changing the state of this environment or

by perceiving this persistent state as changed by others. This is a second category of interaction.

Definition 2.3. *Indirect interaction* is interaction via *persistent, observable state changes*; recipients are any computing entities that will observe these changes.

Whereas in direct interaction a message's destination is fixed by the identifier of the recipient at the time of sending, in indirect interaction the identity of the receiver depends on dynamically generated events such as the entry of agents into the vicinity of data emitted. Indirect interaction is therefore a natural model for systems of *mobile* agents whose ability to perceive and act upon their environment is localized, in contrast to systems with no spatial constraints on communication.

Example: Ant colonies solve the problem of efficiently foraging for food sources by a multiagent interaction in which each ant deposits *pheromones* (evaporating scent chemicals) as it walks, and each ant follows pheromone trails. Heavily traveled (hence strong, hence attractive) pheromone trails correspond to short paths to food. As food is exhausted at a site, the trails to it evaporate. Without a plan, the ants find a set of paths to the food that tends toward optimality [bon-the00]. The communication among the ants is indirect, via pheromone deposits that change the state of the environment, rather than via message passing with handshake.

Notably, as in the case of the ants, use of the real world as an interaction medium in indirect interaction is not precluded. In nature, interaction is often through actions that cause persistent and observable changes in the environment that later affect other organisms. Other social organisms, such as termites and slime amoebae, provide instances of communication via the environment [kei-gol03].

2.3 Properties of indirect interaction

By our definition, indirect interaction features *anonymity*, where the identity of the recipient is not known at the time the interaction is initiated.

Definition 2.4 *Anonymous interaction* occurs when computing entities communicate without knowledge of each other's identities. Anonymity is as opposed to *targeted send/receive* (TSR).

Note that anonymity does not guarantee that *no* entity knows the identities of the communicating processes; that stronger condition could be called *full anonymity*.

Indirect interaction can also involve *time delay (decoupling)*, that is, *asynchrony*, due to the persistence of the observable changes over time. Space decoupling can also be involved, when the agents participating in the interaction never share the same location because one leaves before the second arrives.

In data communications, a *synchronization* step occurs when one process stops and waits until the other has reached a certain point in the computation or interaction, the synchronization point. Similarly, use of *handshake* can enable communicating processes to begin an exchange at the same time.

Definition 2.5 *Asynchronous* processes are ones that do not perform synchronization with respect to each other.

Examples of synchronous interaction are music played to a rhythm set by a percussion instrument or a conductor, or a conversation in which each person waits for the other to finish. Asynchronous interaction includes exchanges of email and communication among ants via pheromone trails. *Message queues* enable asynchronous communication when interaction is direct.

Since no two computing agents in the physical world occupy the same location in space, they must communicate at a distance. Typically to close the distance, agents use *mobility*. Crossing a distance entails time delay, hence asynchrony, in principle. The fact that communication time is generally proportional to distance favors *local* interaction over interaction at a distance [zam-par02]. Thus only ants able to reach the same locale may communicate via the same pheromone trail.

Whereas digital computing agents may carry identifier tags and communicate using them, and may perform handshake for synchronization purposes, the physical world has different protocols. Atoms, cells, organisms, and colonies by their nature interact asynchronously, with timings of interactions and identities of partners fixed by *physical proximity*. Digital computing agents that operate in such environments, as in open systems, must adapt to and incorporate these features.

Not only does interaction in a natural context lack reference to identifiers, but the very notion of identity is problematic there. The set of molecules that are part of a wave of water changes by the second. Cells of an organism and members of a colony are constantly being created and destroyed. For an identifier to be useful, what it identifies must have a stable existence. Hence most interaction in the real world is asynchronous and anonymous.

Section summary: Interaction may be classified as *direct* or *indirect*, indirect interaction enables *asynchronous* and *anonymous* communication.

3 Coordination, asynchrony, and anonymity

Having shown that indirect interaction enables anonymous and asynchronous communication, we argue that *an essential feature* of coordination is the ability to enable anonymity and asynchrony. This will help us to argue in the next section that a certain kind of coordination, the decentralized or self-organizing kind, can *only* be modeled by use of indirect interaction.

3.1 What is coordination?

Coordination is described as enabling the management of dependencies among activities [ric-omi02] and involving the construction of protocols to realize dynamic topologies of interaction among computing entities [arbab98]. The *management* metaphor allows us to distinguish coordination from “computational activities” [gel-car92], just as management activities in a factory setting are distinguished from production activities. Our definition borrows from [car-gel89, arbab98].

Definition 3.1. *Coordination* is the management of interaction among computing entities in *multiagent* systems, including creation and destruction of such entities and of communication links among them.

Coordination only makes sense for *multiagent* systems because in strictly binary interaction, destruction or unlinking of entities destroys the system and creation of new entities produces a multiagent system.

Note that the distinction between coordination and “computation” uses a definition of computation as the execution of algorithms. It is becoming more and more accepted that computation incorporates interaction and therefore coordination.

Asynchrony (Section 2.3) is an essential feature of coordination. Likewise, in a setting where computing entities may come into being and disappear, and where mobile agents connect by virtue of proximity rather than mutual knowledge of identity, the feature of *anonymity* follows, from the definition, as an essential trait of coordination.

It is possible to distinguish two forms of anonymity in multiagent systems, that of senders and that of recipients.

3.2 Asynchrony and anonymity in existing coordination models

We show here that two existing coordination models assure anonymity and asynchrony and argue that any coordination model needs these features.

The idealized worker, idealized manager (IWIM) model for coordination [arbab96] was developed as an alternative to targeted send/receive (TSR) models used in CCS, CSP, Actor [Agha], Occam, and the PVM and MPI tools for parallel computation. IWIM is motivated by the observation that “ideal” workers have no need for information about the senders of communications to them or recipients of their outputs [arbab96]. All such communication is arranged by *manager* processes, which connect *ports* of worker processes with *channels*. The Manifold language for programming manager processes was developed to support the IWIM model.

Anonymity has a *simulated* character in Manifold because, though worker processes do not have access to sender and receiver identifiers, manager processes must have such access in order to create and break channel connections between processes. We will return in Section 5 to this *centralized* aspect of the coordination modeled by IWIM.

The Linda language and model, like IWIM and Manifold, supports anonymity, but enables communication through *tuple spaces* shared among the activities coordinated. We may say that Linda supports *true anonymity*, because once a tuple is inserted into or taken from a tuple space, record of the action is restricted to the sender or retriever alone. In other words, *no* entity, including a coordinator process, has access to identifier information. To model localities in mobile applications, multiple tuple spaces may be added as an extension to Linda under a mechanism of asynchronous communication [den-fer97].

The classic Dining Philosophers problem is one that entails anonymity and asynchrony. The diner/thinkers initiate all interactions locally and spontaneously without reference to identifiers or actions of others.

Section summary: Coordination exists to solve problems in multiagent systems that typically involve *anonymous* and *asynchronous* interaction.

4 Open computational systems and coordination

Coordination is useful in certain kinds of environments; we argue that these are the environments of *open* systems, systems at the top of a hierarchy of notions of openness. Later we will present a case that such environments present challenges that require *decentralized coordination* (Section 5).

4.1 Openness as interactivity

“Open” has various meanings in the literature. The first workshop on Theory and Practice of Open Computational Systems (TAPOCS), part of WETICE ’03, produced a report that defined open systems as “Collectivities/Communities of heterogeneous entities (humans and agents) situated in a (physical) unpredictable environment, collaborating toward the achievement of a mission, supported by a suitable software and hardware infrastructure” [tapocs03].

Here we classify systems involving coordination, such as “asynchronous ensembles, ” among three levels of of interaction in computation: *algorithmic computing*, *sequential interaction*, and *multiagent interaction*. We propose this as one dimension in a hierarchy of openness.

An instance of algorithmic computation is a single finite run of a halting Turing Machine, in which the sequence (input, execution output) occurs. Algorithmic computation is open in the sense that input is arbitrary and is determined by the Turing Machine’s environment. Such openness is low in the hierarchy of openness and is sometimes associated with closed systems. Algorithmic computation closes out the world after the input stage [wegner97].

A second level of openness as interactivity is *sequential interaction*, equivalent to repeated execution of a Turing Machine that is extended by having a *persistent work tape* alongside its input/work/output tape. The extended Turing machine treats both tapes as its input and produces in its computation outputs both to its normal tape and its persistent work tape, in effect updating the persistent “memory” of the device. Sequential interaction is open in the sense that not only does input affect output, but output at an early stage may affect input at a later stage. It is open to the world during computation.

To model, or even specify, sequential interaction, software engineering has required new notations such as the Unified Modeling Language (UML), which introduced *use case diagrams* to model the system user [gol-kei00]. Sequential interaction is open in that the user becomes part of the computation in an ongoing way.

A third level of openness, along the dimension of interactivity, is that of *multiagent interaction*. Such interaction may be, but is not necessarily, the composition of sequentially interactive computations. What allows it to be non-compositional is the possibility that interaction streams in a multiagent system may be created or destroyed

by the creation or destruction either of computing entities or of communication links among them. Process creation was a chief motivator that [car-gel89] gave for creating a language such as Linda. Indeed, a manager who cannot hire or fire, or set up and remove communication channels among subordinates, hardly has tools for adequate management.

We assert that this non-compositional (create/destroy, link/unlink) aspect of multiagent interaction is *central* to coordination and is *definitive* for discussions of openness at the level of multiagent systems.

4.2 Openness in physical environments

A second dimension along which to construct a hierarchy of notions of openness is that of embeddedness in the physical world, which has attributes such as *analog* (real-valued) states and behavior descriptions, *inaccessibility*, and *unpredictability*. Examples of systems that are open according to this criterion are sensor networks and colonies of autonomous mobile robots.

Work in robotics indicates that robust, scalable systems can only be built if designed and tested under conditions of situatedness and embodiedness [brooks91]. This is because the real world offers a qualitatively greater challenge than “toy” environments produced as artifacts by system designers.

[siegel99] shows that a model of analog computation, Analog Recurrent Neural Networks, is in principle a super-Turing one. Research in *hybrid automata* [lamport93] suggests models for partly-digital systems that are “open” in the sense that they interact with an environment whose inputs to these systems is analog. Examples are thermostats, vehicles, and embedded controllers in general. In [keigo03] we presented a proposition that the range of behaviors, using analog environments as media for indirect interaction, is wider than that in digital environments, and a conjecture that even without analog media, indirect interaction enables a richer set of system behavior than exclusively direct interaction. Future work will include verifying these claims.

Using the three tiers of openness defined in Section 4.1 and two tiers defined here suggests the following hierarchical table of notions of openness, filled in with some examples. The upper-left corner is least open; the lower-right is most.

	Digital environment	Physical environment
Algorithmic	Batch computation	Amnesic computation with real-valued inputs
Sequential	PC apps	Controllers, sensors
Multiagent	MAS software systems Internet	Sensor networks Robot societies

Section summary: In a hierarchy of notions of openness, systems in dynamic and unpredictable environments similar to the real world are at the top, and these are the ones most likely to require coordination.

5 Decentralized coordination and indirect interaction

This section focuses on the uppermost echelon of the hierarchy of notions of openness (Section 4): multiagent systems embedded in dynamic and unpredictable environments such as the real world. It presents an argument that coordination must be *decentralized* to solve problems of adaptation in such environments, and that in such cases, *indirect interaction* is needed to enable anonymity and asynchrony.

5.1 Centralized systems and the centralized world view

A “centralized mindset” that has influenced science, philosophy, and other fields and now faces challenge. For example, it has recently been determined, contrary to widespread belief, that behavior in insect colonies is not controlled by the queen and that V-shaped flocks of birds are not following the “leadership” of the bird in front [resnick94]. In the centralized mindset, patterns occur only if designed and orchestrated, and phenomena have single causes.

From [resnick94]’s critique, we obtain a definition that embraces two kinds of central control of systems.

Definition 5.1 A *centralized system* is a multiagent system whose components either respond to commands from an active director or manager component, or execute prespecified roles under a design or plan.

An alternative, more formal definition of centralization is suggested by [barbon03]: A *centralized system* is one described by a *causality graph* of components in which the outputs of a small number of nodes determine the outputs of most of the nodes. That is, it is describable by a causality graph whose number of cycles is relatively small (a graph approximating a tree). Such a *scale-free network*, e.g., the Internet, may be somewhat decentralized and is highly fault-tolerant, but unlike a random network is subject to disruption by coordinated attacks.

Thus in a centralized system, the need for feedback is minimized, interactions are local, and the causality present is unidirectional. Each agent may communicate with its partners in a predictable way via messages using agent identifiers.

In computer science, early work in open interactive systems (Turing’s choice machines, Mealy’s and Moore’s transducers or “sequential machines,” cybernetics and control theory, game theory, communications theory) gave way by the late 1960s or so to a nearly exclusive focus on closed, algorithmic systems, with centralized module hierarchies as the rule of software design. In software engineering, this focus has broadened with the use of object-oriented approaches and languages such as the UML.

Research in coordination has reflected the preference in computer science for centralized management; hence a directing “coordination medium” is often posited. This bias comes in part from the origin of coordination in efforts to solve problems of (predominantly synchronous and algorithmic) parallel computation. Increasingly, however, research in adaptive and multiagent systems has recognized the importance of *decentralized* systems and emergent behavior, as observed in nature [simon70, holland98].

5.2 Adaptation in complex dynamic environments

Systems in environments that are dynamic and unpredictable, such as the real world, reside in the upper rungs of a hierarchy of notions of openness (Section 4). The rapid and seemingly nondeterministic changes in the environment require the most powerful kind of adaptation; namely, ongoing reconfiguration, at many levels, of the components of systems operating in such environments.

Coordination is the performance of such reconfiguration, and entails anonymous and asynchronous interaction (Section 3). The Manifold language and model offer a centralized way to *simulate* anonymity and asynchrony: a central coordination mechanism or agent creates channels between computational processes, using direct interaction with full knowledge of identifiers of the coordinated processes. The coordinated processes interact anonymously, however.

A central coordinating mechanism that is responsible for constantly reconfiguring a system is unlikely to keep up with the requirements of an environment like the real world, filled as it is with more agile decentralized agents such as living organisms.

For the same reason, large organizations find that sending every decision to headquarters for approval deprives them of needed agility.

Hence to support anonymity and asynchrony, coordination in dynamic and unpredictable environments must be *decentralized*.

Conjecture 5.1 Computational agents operating in systems that are in the upper rungs of the openness hierarchy require adaptive features including a *decentralized* form of coordination.

One kind of evidence for this conjecture is that natural organisms, living in environments that are in the upper rungs of the openness hierarchy, have evolved with strong such features.

5.3 Self-organization and decentralized coordination

Emergent behavior is observed when the behavior of a whole system is more than the sum of the behaviors of the individual components [Simon]. Termites gathering chips, ants foraging for food, and slime molds dividing and aggregating are instances of this. Similarly, intelligent behavior emerges from unintelligent neurons in the brain, and humans collaborating can produce results of a higher *qualitative* nature than when working separately. The common feature is that the systems are organized locally rather than according to a global command mechanism or design.

Definition 5.2 A *self-organizing system* is a multiagent system whose global structure or pattern is coherent and is shaped by local interactions among components rather than by specific external causes [cam-den01, had-val03].

The close connection between self-organization and decentralized systems is indicated by the predominance of local interactions in this definition and by the fact that specific external causes are not decisive. That is, if centralized systems are those with either a leader or a design that controls behavior, then self-organization is precisely organization and coherence without centralization. We hold that in

coordination, from natural systems to business settings, adaptation to rapidly changing environments requires self-organization.

Pressures for flexibility, speed, and adaptiveness in the global market have produced a general trend away from hierarchical, rigid business-organization models. Interdisciplinary research in self-organization has offered insights usable in manufacturing control [had-val03, and-bar00]. It has been found that “cooperative activity” in self-organizing systems, defined as mutually beneficial interchange, may have the effect of coordination without explicit intention being present [gle-cam99].

5.4 Decentralized coordination and indirect interaction

A central coordination mechanism may possess identifiers of all components and may reconfigure them, using direct interaction. Lacking such central control, a system may organize its own internal communication to avoid loss of coherence. This interaction is necessarily *indirect* if new components come into being without identifiers known to the other components.

Coherent behavior – coordination – in a decentralized system depends on dynamically shifting connections among agents, in which shared locations in the environment, or intermediaries identified by their locations, are used rather than messages and identifiers are used but rather. In effect, the coordination is assured via the environment.

Definition 5.2 *Decentralized coordination* is the management of interaction among computational entities in a system that lacks centralized control.

We conjecture that decentralized coordination and self-organization amount to the same notion.

Section summary:

1. Adaptation in complex dynamic environments sometimes requires decentralized coordination.
2. To achieve anonymity and asynchrony, coordination that is decentralized must use indirect interaction.

6 Conclusion: Toward new formalisms for coordination

Having shown that indirect interaction is essential to decentralized coordination in real-world-like environments (Section 5), hence in truly open systems, we present a case for explicit modeling of indirect interaction.

6.1 Beyond direct interaction

Research in coordination has modeled the interaction of activities in multiagent systems using the tools of *concurrency theory*, based on *process algebras* and *labeled transition systems*. Work initiated by Robin Milner in the 1970s and 1980s found a

need for a new conceptual framework, because the semantics of running component processes concurrently may be different from the semantics of running them one after the other; concurrency has *non-compositional* semantics, due to nondeterminism of mutually interfering processes [milner93]. Concurrency, like complex systems, is nonlinear; the whole is different from the sum of the parts.

The conclusion drawn was that concurrent systems should be modeled by treating the interaction between program and memory as a special case of interaction among peers. Hence in this model, shared memory is represented as a process. The atom of behavior in CCS, for example, is the transmission of a single item of data between processes – message passing. All interactions are “treated in the same way,” because all entities accessed by names, whether values, processes, or objects, are modeled as processes.

Since the only mode of communication under this model is synchronous message passing, clearly the model only supports *direct* interaction (Definition 2.2), but not indirect interaction (Definition 2.3).

[arbab98] observed that existing formal models in concurrency theory, such as process algebras, CSP, CCS, π Calculus, “are more effective for describing closed systems,” and that the application context that gave rise to concurrency theory was different from that in the late 1990s, in which multiagent systems pose the newest challenges.

In the Manifold IWIM coordination model, as in the traditional concurrency model, communication is via *channels* between processes. This concurrency model and the use of process algebras and transition systems have been the foundation for Manifold’s and Linda’s support for anonymous and asynchronous communication. However, these models, based on direct interaction, especially Manifold, which only simulates anonymity of communication, may tend to break down when faced with the task of modeling *decentralized* interaction such as that in self-organizing systems. There, what central coordinating agent exists that could be modeled by Manifold’s channel-creating capability?

6.2 Toward new models

This paper has identified inadequacies in current formalisms for coordination models, leading to recognition of the need for new models that incorporate notions of indirect interaction.

A summary of our main points is as follows:

1. There exist two notions of interaction, direct and indirect;
2. Indirect interaction enables asynchronous and anonymous forms of communication;
3. The purpose of coordination is to address problems in multiagent systems in which anonymity and asynchrony are needed;
4. In a hierarchy of notions of openness, systems in dynamic and unpredictable environments similar to the real world are at the top, and these are the systems most likely to require coordination.

5. In such environments, adaptation may require decentralized coordination; such coordination must use indirect interaction to achieve anonymity and asynchrony.
6. Models of coordination, to be complete and adequate for design purposes, must represent indirect interaction in an explicit way.

A practical concern for system designers is the *scalability* of a modeling technique. Whereas the notion of persistent environment serving as a medium for indirect interaction is scalable, the representation of indirect interaction as direct interaction between active processes and passive shared variables scales poorly. On one end of the spectrum, it is unrealistic to model every shared variable in a discrete environment as a process with its own communication channels to each process that accesses it. On the other end, if computing entities communicate via the real world, then one must model an analog entity or set of entities unrealistically as a finitely specifiable discrete process.

Future work should address in more detail the requirements of a model of indirect interaction suitable for coordination.

7 References

- [and-bar00] Grl Anderson and John J. Bartholdi, III. Centralized versus decentralized control in manufacturing: Lessons from social insects. *Proceedings, Complexity and Complex Systems in Industry*, pp. 92-105, 2000.
- [arbab96] Farhad Arbab. The IWIM model for coordination of concurrent activities. *COORD 96*, pp. 34-56.
- [arbab98] Farhad Arbab. What do you mean, coordination? *Bulletin of the Dutch Association for Theoretical Computer Science*, March 1998.
- [bar-bon03] A. Barabasi and E. Bonabeau. Scale-free networks. *Scientific American*, May 2003, pp. 60-69.
- [bon-the00] Eric Bonabeau and Guy Theraulaz. Swarm smarts. *Scientific American*, March 2000, pp. 72-79.
- [brooks91] Rodney A. Brooks. Intelligence without reason. MIT AI lab A.I. Memo No. 1293, 1991.
- [car-gel89] Nicholas Carriero and David Gelernter. Linda in context. *Communications of the Association for Computing Machinery*. 32 (4), pp. 444-458, 1989.
- [den-fer97] Rocco De Nicola, GianLuigi Ferrari, Rosario Pugliese. Locality based Linda: Programming with explicit localities. *TAPSOFT 97, LNCS*, 1997.
- [dijkstra68] Edsger W. Dijkstra. Co-operating sequential processes. In F. Genuys, Ed., *Programming languages*, Academic Press, pp. 43-112, 1968.
- [gel-car92] David Gelernter and Nicholas Carriero. Coordination languages and their significance. *Communications of the Association for Computing Machinery* 35 (2), pp. 97-107, 1992.
- [goldin00] Dina Goldin. Persistent Turing Machines as a Model of Interactive Computation. FoIKS'00, Cottbus, Germany, Feb. 2000.
- [go-kei01] Dina Goldin, David Keil, Peter Wegner. An interactive viewpoint on the role of UML. In K Siau and T. Halpin, *Unified Modeling Language: Systems analysis, design and development issues*, Idea Group Publishing, 2001, pp. 249-263.

- [gol-smo00] Dina Q Goldin, Scott A. Smolka, Paul C. Attie, Peter Wegner. Turing machines, transition systems, and interaction. 8th Int'l Workshop on Expressiveness in Concurrency, Aarlborg, Denmark, August 2001.
- [had-val03] Hadeli, P. Valckenaers, C. B. Zamfirescu, H. Van Brussel, B. Saint Germain, T. Holvoet, E. Steegmans. Self-organising in multi-agent coordination and control using stigmergy, First International Workshop on Engineering Self-Organising Applications (ESOA 2003), 15th July 2003, Melbourne, Australia.
- [kah-mil88] Kenneth M. Kahn and Mark S. Miller. Language design and open systems. In B. A. Huberman, *The ecology of computation* (North-Holland, 1988), pp. 291ff.
- [kei-gol03] David Keil and Dina Goldin. Modeling indirect interaction in open computational systems. Workshop on Theory and Practice of Open Computational Systems. *Proceedings, WET ICE 03, Linz, Austria*.
- [ken-ebe01] James Kennedy and Russell C. Eberhart. *Swarm intelligence*. Morgan Kaufmann, 2001.
- [lamport93] Leslie Lamport. Hybrid systems in TLA+. In R. Grossman et al, Eds., *Hybrid systems* (Springer, 1993), pp. 77-102.
- [milner75] Robin Milner. Processes: A Mathematical Model of Computing Agents. In H. E. Rose and J. C. Shepherdson, Eds., *Logic Colloquium '73*. Amsterdam: North-Holland, 1975.
- [milner80] Robin Milner. *A calculus of communicating systems*. LNCS 92, Springer, 1980.
- [milner93] Robin Milner. Elements of Interaction. *Communications of the Association for Computing Machinery* 36 (1), pp. 78-89, 1993.
- [milner99] Robin Milner. *Communicating and mobile systems: the π calculus*. Cambridge: Cambridge University Press, 1999.
- [mos-ran03], S. K. Mostefaoui, O. F. Rana, N. Foukia, S. Hassas, G. Di Marzo, C. Van Aart, A. Karageorgos. Self-organising applications: A survey. ESOA-03.
- [omi-den01] Andrea Omicini and Enrico Denti. From tuple spaces to tuple centres. *Science of Computer Programming* 41, pp. 277-294, 2001.
- [pap-arb98] George A. Papadopoulos and Farhad Arbab. Coordination models and languages. In *Advances in computers*, Vol. 46, Academic Press, 1998.
- [resnick94] Mitchel Resnick. *Turtles, Termites, and Traffic Jams*. MIT Press, 1994.
- [ric-omi02] Alessandro Ricci, Andrea Omicini, Enrico Denti. Activity Theory as a Framework for MAS Coordination. Proceedings, ESAW'02.
- [siegel99] Hava Siegelmann. *Neural networks and analog computation: Beyond the Turing limit*. Birkhauser, 1999.
- [simon70] Herbert Simon. *The Sciences of the Artificial*. MIT Press, 1970.
- [tapocs03] M. Fredriksson, R. Gustavsson, A. Ricci, A. Omicini. TAPOCS 2003 Workshop report and roadmap (slides). <http://www.soclab.bth.se/workshops/tapocs2003>.
- [wegner96] Peter Wegner, Coordination as constrained interaction. *Coordination* 96, pp. 28-33.
- [wegner97] Wegner, Peter. Why Interaction is More Powerful than Algorithms. *Communications of the ACM* 40 (5), 1997.
- [wegner98] Peter Wegner. Interactive Foundations of Computing. *Theoretical Computer Science* 192, pp. 315-351, 1998.
- [vir-omi02] Mirko Viroli, Andrea Omicini. Specifying agent observable behavior. AAMAS'02, July 15-19, 2002, Bologna, Italy.
- [zam-par02] Franco Zambonelli and H. Van Dyke Parunak. Signs of a revolution in computer science and software engineering. Proceedings, ESAW02.