Developing Formal Specifications to Coordinate Heterogeneous Autonomous Agents

Munindar P. Singh[*]
Department of Computer Science
North Carolina State University
Raleigh, NC 27695-7534, USA

singh@ncsu.edu

Abstract

We have been developing an approach for the distributed coordination of heterogeneous, autonomous agents. This approach takes as input (a) agent skeletons, giving compact descriptions of the given agents in terms of their events that are significant for coordination, as well as (b) relationships among the events occurring in these skeletons. A natural question is how may the skeletons and relationships be produced in the first place. Parunak recently proposed a methodology for designing multiagent systems based on Dooley graphs from discourse analysis. We show how with a few key modifications, Dooley graphs can also be used to generate the skeletons and relationships required for coordination. This combines the benefits of an intuitive methodology with a formal and distributed framework for developing multiagent systems from autonomous agents.

1 Introduction

We have initiated a program of research on *interaction-oriented programming (IOP)*. IOP seeks to develop techniques and tools for the construction of multiagent systems by specifying the interactions among (usually) autonomous agents. We lack the space to review all of IOP here, but additional details may be found elsewhere [9, 10, 11].

Coordination is an important class of interactions, which deals with how different agents synchronize their activities. As part of IOP, we developed an approach for coordinating heterogeneous, autonomous agents [10]. Our approach specifies individual agents in terms of their *skeletons*, which give coarse (and therefore compact) descriptions of their behavior that capture the essential aspects of their behavior of significance to their potential coordination with other agents. Desired coordinations are specified by stating *relationships* among the events of different agents. These relationships are expressed in a formal language based on event algebra (a form of temporal logic), and can be automatically processed to yield distributed means of coordinating the events of different agents.

Applying this approach requires a means to specify coordinations intuitively and correctly. We consider Dooley graphs from discourse analysis [2], which were recently introduced to the multiagent community by Parunak [7]. Interestingly, Dooley graphs can be used to generate the skeletons and relationships that are input to our approach. The graph is processed to highlight the causal relationships among actions, and the structural properties of the interactions of agents.

Focus. There has been a large amount of good work on some related areas, especially agent communication languages and protocols. We shall not be getting into that subject here. For simplicity and ease of presentation, we follow Parunak's classification of speech acts and his set of "relationships" among utterances. However, we believe that our approach can be applied in other settings as well, provided they can identify the different "characters" played by an agent in a conversation. (The quoted terms are explained below.)

Organization. Section 2 describes the concepts of our coordination approach. Section 3 presents a brief exposition of Dooley graphs. Section 4 shows how we carry out the synthesis by working out an example from [7] to convert a Dooley graph into a set of agent skeletons.

[*]Munindar Singh is supported by the NCSU College of Engineering, the National Science Foundation under grants IRI-9529179 and IRI-9624425 (Career Award), and IBM corporation.

2 Coordination

We summarize the key concepts of our coordination service. Additional details are available in [10].

2.1 Coordination Model

There are two aspects of the autonomy of agents that concern us. One, the agents are designed autonomously, and their internal details may be unavailable. Two, the agents act autonomously, and may unilaterally perform certain actions within their purview. We assume that, in order to be able to coordinate them at all, the designer of the multiagent system has some limited knowledge of the designs of the individual agents. This knowledge is in terms of their externally visible actions, which are potentially significant for coordination. We call these the significant *events* of the agents. In other words, the only events we speak of are those publicly known—the rest are of no concern to the coordination service.

Event Classes Our metamodel considers four classes of events, which have different properties with respect to coordination. Events may be

- *flexible*, which the agent is willing to delay or omit
- *inevitable*, which the agent is willing only to delay
- *immediate*, which the agent performs unilaterally, that is, is willing neither to delay nor to omit
- *triggerable*, which the agent is willing to perform if requested.

The first three classes are mutually exclusive; each can be conjoined with triggerability. The category where an agent will entertain omitting but not delaying an event is empty, because unless the agent performs the event unilaterally, there must be some delay in receiving a response from the service.

Agent Skeletons It is useful to view the events as organized into a *skeleton* to provide a simple model of an agent for coordination purposes. Skeletons are well-known from logics of program, especially since Emerson & Clarke [3]. The skeletons are typically finite state automata. However, they can be anything as far as our formal system and implementation are concerned—neither looks at their structure. In particular, the skeletons may be sets of finite state automata, which can be used to model the different threads of a multithreaded agent. The set of events, their properties, and the skeletons of the agents are usually realized by an agent and, if so, in an application-specific manner. These can be viewed as requirements that are set by the protocol in which the designer wishes the agents to participate. Example 1 discusses two common skeletons.

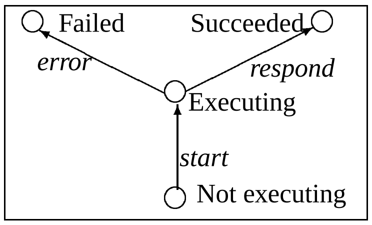

Figure 1. Example Skeleton for Querying

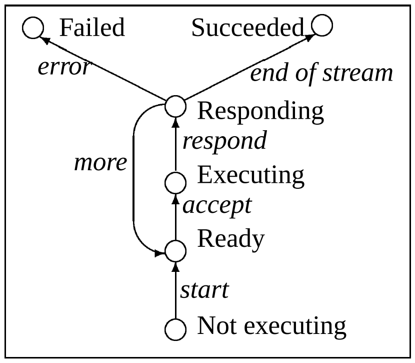

Figure 2. Example Skeleton for Information Filtering

Example 1 Figures 1 and 2 show two skeletons that arise in information search. The skeleton of Figure 1 is suited for agents who perform one-shot queries. Its significant events are *start* (accept an input and begin), *error*, and *respond* (produce an answer and terminate). he skeleton of Figure 2 is suited for agents who filter a stream or monitor a database. Its significant events are *start* (accept an input, if necessary, and begin), *error*, *end of stream*, *accept* (accept an input, if necessary), *respond* (produce an answer), *more* (loop back to expecting more input). In both skeletons, the application-specific computation takes place in the node labeled "Executing." We must also specify the categories of the different events. For instance, we may state that *error*, *end of stream*, and *respond* are immediate, and all other events are flexible, and *start* is in addition triggerable. ▮

Although the skeleton is not used explicitly by the coordination service during execution, it can be used to validate specified coordination requirements. More importantly, the skeleton is essential for understanding the public behavior of an agent, and for giving intuitive meaning to its actions.

2.2 Coordination Relationships

Coordinations are specified by expressing appropriate relationships among the events of different agents. Our formal language allows a variety of relationships to be captured. For reasons of space, we include the formal syntax and semantics of this language in Appendix A.

Table 1 presents some common examples. The events all carry parameter tuples, but we don't show them below to reduce clutter. Some of the relationships involve coordinating multiple events. For example, R8 captures requirements such as that if an agent does something (e), but another agent does not match it with something else (f), then a third agent can perform g. This is a typical pattern in applications with data updates, where g corresponds to an action to restore the consistency of the information (potentially) violated by the success of e and the failure of f. Hence the name *compensation*.

3 Dooley Graphs

Our presentation of Dooley graphs is based on the exposition in [7]. The key idea that interests us here is that Dooley graphs provide a natural way to present an actual conversation as it happened. By concentrating on specific conversations, Dooley graphs can separate the different *characters* played by a single agent. The characters can correspond to different components in an agent—roughly, this is what interests Parunak. However, we are also interested in the structure imposed on an agent's skeleton by the characters it plays. Further, the interactions among the characters lead to coordination relationships among the skeletons.

Agents act, both communicatively (i.e., using speech acts [1]) and physically. We are interested in the agents' interactions with one another. Typically, the agents' actions do not arise in isolation, but as parts of extended communicative activities. These activities can be thought of as protocols, dialogues, arguments, or negotiations among agents. Parunak uses the term *conversation* for a specific instance of these composite activities.

Conversations naturally include not only speech acts, but also some physical actions by means of which the agents deliver on their promises. Parunak allows the speech acts of Solicit (Request or Question) or Assert (Inform, Commit, and Refuse). He allows two physical acts: Ship and Pay.

In addition to the acts in a conversation, there is also knowledge of certain relationships among the speech acts. These relationships are restricted to be one of the following. Here, u_i and u_j refers to different utterances in a conversation. S_i refers to the sender of u_i.

- *Respond*. u_i responds to u_j iff (a) S_i previously received u_j, (b) u_j's impact on S_i caused S_i to send u_i, and (c) u_i is the first utterance of S_i to satisfy (a) and (b).

- *Reply*. u_i replies to u_j iff (a) S_i previously received u_j, (b) u_j's impact on S_i caused S_i to send u_i, and (c) u_i is the first utterance of S_i directed to S_j that satisfies (a) and (b).

- *Resolve*. u_i resolves u_j iff u_i replies to u_j and u_i follows the "rules of engagement" defined in u_j.

- *Complete*. u_i completes u_j iff u_j is a Commit and u_i either satisfies or cancels the associated commitment.

Respond, Reply, and Resolve are progressively more restrictive. Complete is mutually exclusive with Resolve—an act cannot both complete an utterance and resolve an utterance (not even a different one).

Example 2 Consider a request for proposals (RFP) from A to B, C, and D. The first act that any of them does that was caused by the RFP is a Response to it. If it is a message back to A, then it is also a Reply. If the Reply is a Commit or a Refuse, then it is also a Resolve. ▮

Example 3 Table 2 shows an example conversation from [7]. The #s partially order the utterances from early to late. In this conversation, A announces an RFP for 50 widgets to B, C, and D. B checks with C is C is bidding. C says it is. B then refuses A. C, however, makes a counter offer of 40 widgets. A accepts and C commits. In the meanwhile D offers to accept the initial RFP, which is more preferable to A. A then declines C, who cancels its commitment. D delivers, but the order is short (45 only). A informs it. When D complies, A pays. Table 2 also shows the discourse relations among the utterances.

Parunak's example is oversimplified in that D commits to supplying the widgets without bothering to check if A actually accepted its bid. However, this and other simplifications won't affect the main thrust of our paper. ▮

A Dooley graph is generated by analyzing a conversation in such a manner that the sets of utterances that are closely related to one another are brought closer. This is used to induce a set of *characters* from each participant in the conversation. The characters reflect the rhetorical structure of the conversation, and become the vertices of the graph.

Example 4 Figure 3 gives the Dooley graph for Table 2. The numbered utterances relate the characters that send and receive them. ▮

	Name	Description	Formal notation
R1	e is required by f	If f occurs, e must occur before or after f	$e \vee \overline{f}$
R2	e disables f	If e occurs, then f must occur before e	$\overline{e} \vee \overline{f} \vee f \cdot e$
R3	e feeds or enables f	f requires e to occur before	$e \cdot f \vee \overline{f}$
R4	e conditionally feeds f	If e occurs, it feeds f	$\overline{e} \vee e \cdot f \vee \overline{f}$
R5	Guaranteeing e enables f	f can occur only if e has occurred or will occur	$e \wedge f \vee \overline{e} \wedge \overline{f}$
R6	e initiates f	f occurs iff e precedes it	$\overline{e} \wedge \overline{f} \vee e \cdot f$
R7	e and f jointly require g	If e and f occur in any order, then g must also occur (in any order)	$\overline{e} \vee \overline{f} \vee g$
R8	g compensates for e failing f	if e happens and f doesn't, then perform g	$(\overline{e} \vee f \vee g) \wedge (\overline{g} \vee e) \wedge (\overline{g} \vee \overline{f})$

Table 1. Example Relationships

#	S	R	Utterance	Respond to	Reply to	Resolve	Complete
1	A	B,C,D	Request (RFP for 50)				
2	B	C	Question: bidding?	1			
3	C	B	Inform: yes	2	2	2	
4	B	A	Refuse	3	1	1	
5	C	A	Propose (take 40)	1	1		
6	A	C	Request (send 40)	5	5	5	
7	C	A	Commit (deliver 40)	6	6	6	
8	D	A	Commit (deliver 50)	1	1	1	
9	A	C	Assert (decline)	7, 8	7		
10	C	A	Refuse	9	9		7
11	D	A	Ship (deliver 45)	1	1		8
12	A	D	Assert (short) + Request	11	11		
13	D	A	Ship remainder, i.e., 5	12	12	12	
14	A	D	Pay	13	13	13	

Table 2. Example Conversation

4 Approach

Dooley graphs highlight the rhetorical structure of a conversation, but hide its causal structure or, in more mundane terms, the control flow among the agents where more than one character of an agent is involved. (Parunak's proposed extension also does not display the actual causal connections, and we don't consider it in detail here.)

Our approach proceeds as follows. We begin with a Dooley graph depicting the conversation being analyzed. We analyze this Dooley graph to explicitly identify the causal relationships among the various utterances. We separate out the histories of the different participants, but record the contribution of each character. Table 3 shows the histories derived from the Dooley graph of Figure 3. The different characters are highlighted in each history.

4.1 Inducing Agent Skeletons

We use the following conventions. An event type named "get ..." corresponds to the receipt of an utterance. We would expect an event type in another agent corresponding to the making of that utterance. There is no assumption that the two events happen in synchrony, and usually they would not. In the skeletons, we parenthetically show the corresponding utterance number from Table 2. A * indicates an action not in the given conversation.

Figure 4 shows possible skeletons for B. The first requires the agent to consult C before deciding whether to propose. It would be inappropriate in most settings, because it puts strong constraints on B's design. The second skeleton goes even farther and requires B's decision to depend on C. These skeletons place B's decision-making publicly in the protocol, and are clearly unacceptable. The last skele-

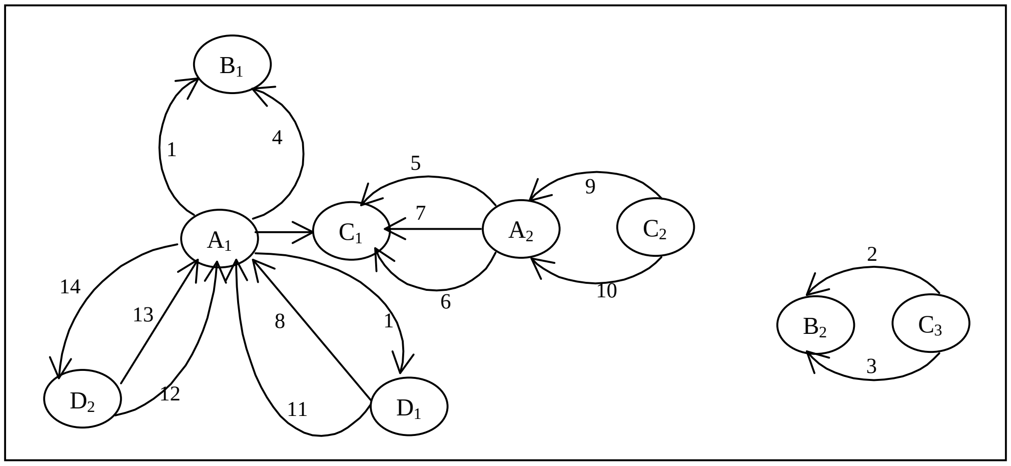

Figure 3. Example Conversation as a Dooley Graph

Role	History
B	$(A_1, 1, B_1); \boxed{(B_2, 2, C_3); (C_3, 3, B_2)}\,; (B_1, 4, A_1)$
C	$(A_1, 1, C_1); \boxed{(B_2, 2, C_3); (C_3, 3, B_2)}\,; (C_1, 5, A_2); (A_2, 6, C_1); (C_1, 7, A_2); \boxed{(A_2, 9, C_2); (C_2, 1, 0A_2)}$
D	$(A_1, 1, D_1); (D_1, 8, A_1); (D_1, 11, A_1); \boxed{(A_1, 12, D_2); (D_2, 13, A_1); (A_1, 14, D_2)}$
A	$(A_1, 1, B_1); (A_1, 1, C_1); (A_1, 1, D_1); (B_1, 4, A_1); \boxed{(C_1, 5, A_2); (A_2, 6, C_1); (C_1, 7, A_2)}\,; (D_1, 8, A_1);$ $\boxed{(A_2, 9, C_2); (C_2, 10, A_2)}\,; (D_1, 11, A_1); (A_1, 12, D_2); (D_2, 13, A_1); (A_1, 14, D_2)$

Table 3. The Histories of Agents in a Conversation

ton, however, leaves it up to B to decide whether to consult C and how to use its response. This skeleton captures the key intuition about character B_2, which is that it engages in a subdialogue with character C_3. This justifies selecting the last of the possible skeletons for B.

Notice that when B asks C, C's response is relevant to B's further actions. However, when B asks C, this query may have no consequence on C's actions (and in this protocol doesn't). Consequently, Figure 5 shows a skeleton for C in which C may get a query from B, but this query is structurally independent of how C handles RFPs. Similarly, the counter-proposal is kept as a separate loop but attached to the main flow. This too is an example where a character is modeled with a separate subskeleton (physically a thread) in the agent's skeleton. (For reasons of space, D is discussed when integrated below.)

The decision whether to have a separate thread or a loop in a single thread depends on how we understand the agents to be acting and interacting. Clearly, we must separate what the agents happen to do from what is essential for coordination in the given application. Dooley graphs, by focusing on a specific conversation are in tension with this process. However, in settings such as our present example, we can derive more information from the graph by recognizing that the same role is instantiated by multiple agents. Here, the multicast by A is a clue that B, C, and D are to be treated alike. In such a case, we can achieve the correct solution by integrating the skeletons. Figure 6 shows a composite skeleton assuming B, C, and D play the role of contractor. By integrating the skeletons, we can construct a single more complete skeleton than any of the agents in the given conversation indicates. The ship and get-error loop refers to character D_2 of Figure 3. In this case, giving it a separate loop would have caused the ship action to appear on two different transitions, and would have been less clear.

Figure 7 shows the skeleton for A. The main quirk in this is that A performs a multicast, and effectively keeps a separate thread to deal with each contractor. Note that it is not clear if only one bid can be accepted, because the bids may each be partial. If there were such a requirement, it would be captured as a disabling relationship (*a la* R2 in Table 1).

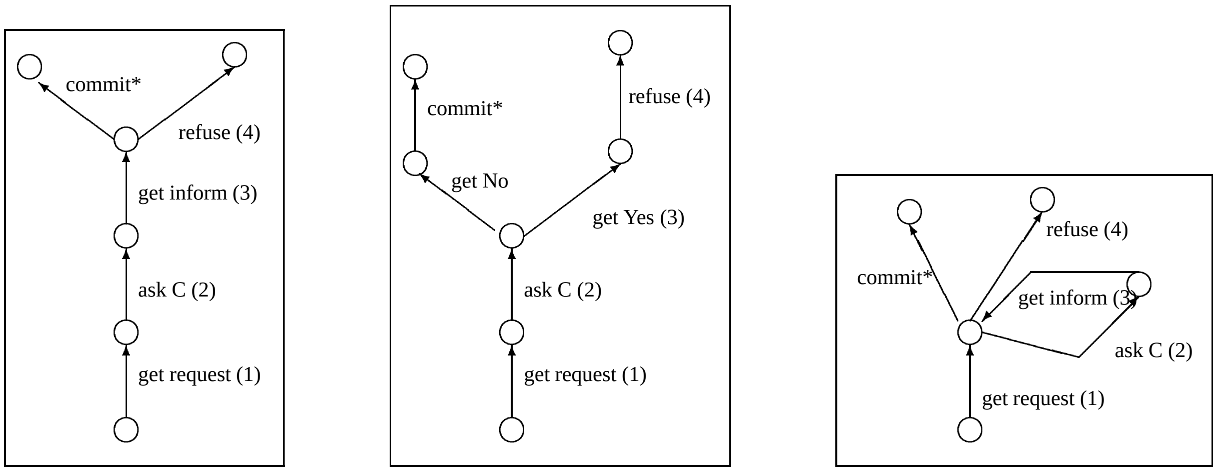

Figure 4. Possible Skeletons for Agent B

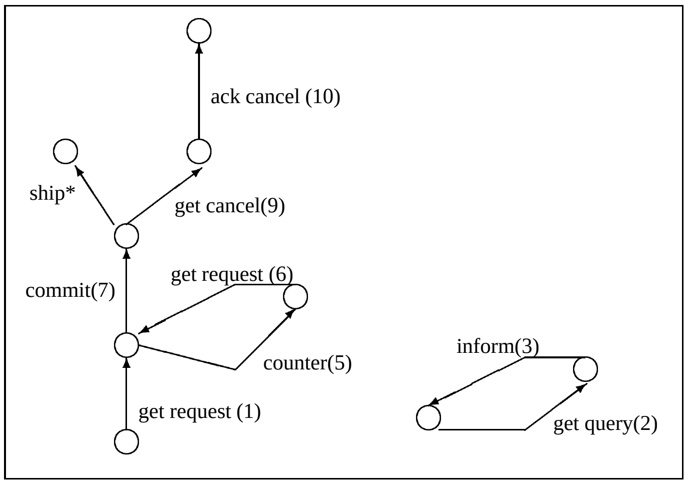

Figure 5. Possible Skeleton for Agent C

4.2 Inducing Event Classes

With the skeletons in hand, the properties of the events can be readily inferred. The "get" events for requests, queries, or cancelations are all triggerable, because that is how the agent is informed by others. In particular, unexpected events must conceptually be treated as triggerable. In some cases, the agent may be implemented so that triggerability is effected by polling, but that is a detail that is independent of our conceptual understanding.

Many of the agents' events may be modeled as immediate or at least inevitable, because the agents will perform them if they wish, although they may be willing to wait. For example, when A cancels, it cannot be told it should not. That is simply its prerogative as an autonomous agent. In some cases, however, when we wish to monitor the agents more closely, we might restrict events such as cancel so they may occur only after a commitment has been created, e.g., after a contractor has responded. In such a case, the event may be modeled as flexible.

4.3 Inducing Relationships

The above example does not involve enough variety of relationships to exercise all of our formal language. There are no important ordering constraints among the events of different agents, except for when triggering is involved. However, the following rules are easily identified—for convenience we refer to Table 1 below.

- Every Solicit is Replied to (R1). Replies may or may not be required in every protocol, however.
- The Replies must be enabled by the utterances they Reply to (R3).

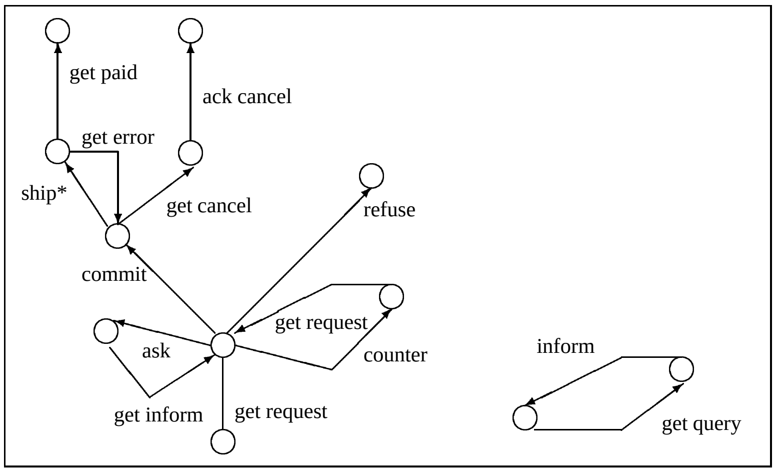

Figure 6. Integrated Skeleton for All Contractors

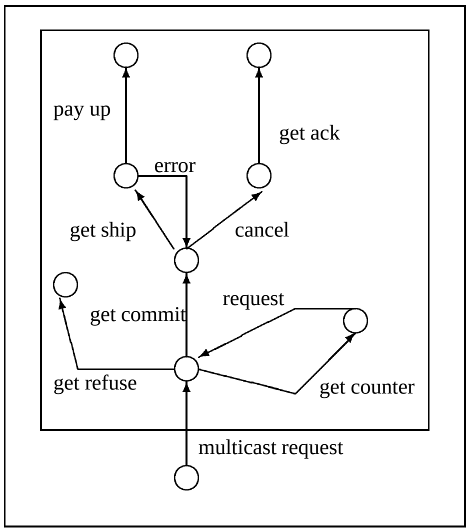

Figure 7. Skeleton for Agent A

- Every Commit is completed (R1).

- Responds is implemented in an application-specific manner. However, input Solicits can enable associated Responds. Sometimes, we may not wish to allow this, e.g., so B can ask C anyway.

- The presence of a non-Reply Respond to an utterance, e.g., $(B_2 2C_3)$, indicates that the Reply is not required right away. The non-Reply Respond itself is done unilaterally by the agent and must be modeled as immediate.

5 Conclusions and Future Work

We showed how we can begin with Dooley graphs and with some heuristics about well-formed skeletons and symmetry across agents in the same role, come up with reasonable skeletons for coordination. We can also infer many of the desired relationships among the skeletons. The process is not automatic, but can help a human designer create a good specification. A large-scale evaluation has not yet been performed, however.

We strongly believe that the nascent science of multiagent systems is inherently interdisciplinary. Accordingly, researchers in this area should be continually looking for useful ideas in other fields. We applaud Parunak for his efforts in recruiting ideas from applied linguistics.[1] For our part, we have pursued ideas from logics of program and databases in developing our coordination service. Here we showed how we can combine separately borrowed ideas to strengthen multiagent approaches still further!

There are a number of topics for future investigation. One is the consideration of conversations that are effectively nonterminating. If these are specifiable as finite state machines, we should be able to generalize Dooley graphs to accommodate them. Repeated interactions among the agents can help identify more of the branches of the possible conversations, but care must be taken so that unnecessary causal connections are not inferred. Another challenge is to use negative examples, i.e., graphs that describe failed conversations or conversations that do not meet some desired criteria. These tasks could be facilitated by a tool that incorporates some machine learning ideas.

We are intrigued by the distributed computing literature

[1]It is interesting that Dooley graphs don't feature in [6], which is a revised version of [5], so one wonders if the discourse analysis community found Dooley graphs not so useful.

on *potential causality*, and speculate that it will bear a fruitful relation to the present subject [4]. Potential causality is the idea that where there is an information flow across events within an agent or across events in different agents (through message passing), there may be a causal connection. A problem is that there can be far more potential causes than real causes [8]. An analysis of a number of conversations may help restrict the notion, however. Also, if we can use potential causality, we need reduced input from the designer or analyzer. This would not only simplify their task, but more readily incorporate heterogeneous agents, e.g., produced by different vendors, whose internal details are not known.

References

[1] J. L. Austin. *How to Do Things with Words*. Clarendon Press, Oxford, 1962.

[2] R. A. Dooley. Appendix B: Repartee as a graph. In *[5]*, pages 348–358. 1976.

[3] E. A. Emerson and E. C. Clarke. Using branching time temporal logic to synthesize synchronization skeletons. *Science of Computer Programming*, 2:241–266, 1982.

[4] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, July 1978.

[5] R. E. Longacre. *An Anatomy of Speech Notions*. Peter de Ridder, Lisse, 1976.

[6] R. E. Longacre. *The Grammar of Discourse*. Plenum Press, New York, 2nd edition, 1996.

[7] H. V. D. Parunak. Visualizing agent conversations: Using enhanced Dooley graphs for agent design and analysis. In *Proceedings of the 2nd International Conference on Multiagent Systems*, pages 275–282. AAAI Press, 1996.

[8] R. Schwarz and F. Mattern. Detecting causal relationships in distributed computations: In search of the holy grail. *Distributed Computing*, 7(3):149–174, 1994.

[9] M. P. Singh. Commitments among autonomous agents in information-rich environments. In *Proceedings of the 8th European Workshop on Modelling Autonomous Agents in a Multi-Agent World (MAAMAW)*, pages 141–155. Springer-Verlag, May 1997.

[10] M. P. Singh. A customizable coordination service for autonomous agents. In *Proceedings of the 4th International Workshop on Agent Theories, Architectures, and Languages (ATAL)*. Springer-Verlag, July 1997. Publication date: 1998.

[11] M. P. Singh. The intentions of teams: Team structure, endodeixis, and exodeixis. In *Proceedings of the 13th European Conference on Artificial Intelligence (ECAI)*. John Wiley, Aug. 1998.

A Formal Syntax and Semantics

We formalize interactions in an event-based linear temporal logic. $\mathcal{I}$, our specification language, is propositional logic augmented with the *before* $(\cdot)$ temporal operator. The literals denote event types, and can have parameters. A literal with all constant parameters denotes an event token. Crucially, $\mathcal{I}$ can express a remarkable variety of interactions, yet be compiled and executed in a distributed manner.

The syntax of $\mathcal{I}$ follows. Ξ includes all event literals (with constant or variable parameters); $\Gamma \subseteq \Xi$ contains only constant literals. A *dependency* is an expression in $\mathcal{I}$.

Syntax 1 $\Xi \subseteq \mathcal{I}$

Syntax 2 $I_1, I_2 \in \mathcal{I} \Rightarrow I_1 \vee I_2, I_1 \wedge I_2, I_1 \cdot I_2 \in \mathcal{I}$

Our formal semantics is based on traces, i.e., sequences of events. Our universe is $\mathbf{U}_{\mathcal{I}}$, which contains all consistent traces involving event tokens from Γ. Consistent traces are those in which an event token and its complement do not occur, and in which event tokens are not repeated. $[\![\,]\!] : \mathcal{I} \mapsto \wp(\mathbf{U}_{\mathcal{I}})$ gives the denotation of each member of $\mathcal{I}$. The specifications in $\mathcal{I}$ select the acceptable traces—specifying I means that the service may accept any trace in $[\![I]\!]$.

Let constant parameters be written as c_i etc.; variables as v_i etc.; and either variety as p_i etc. $e[c_1 \ldots c_m]$ means that e occurs appropriately instantiated.

Semantics 1 $[\![e[c_1 \ldots c_m]]\!] = \{\tau \in \mathbf{U}_{\mathcal{I}} : e[c_1 \ldots c_m]$ occurs on $\tau\}$

$\overline{e}$ refers to the complement of e. Since $[\![\,]\!]$ yields sets of traces, complementation is stronger than negation in other temporal logics. Intuitively, $\overline{e}[c_1 \ldots c_m]$ is established only when it is definite that $e[c_1 \ldots c_m]$ will never occur. Complemented literals are included in Ξ and need no separate syntax or semantics rule.

$I(v)$ refers to an expression free in variable v. $I(v ::= c)$ refers to the expression obtained from $I(v)$ by substituting every occurrence of v by c. Variable parameters are effectively universally quantified by:

Semantics 2 $[\![I(v)]\!] = \bigcap_{c \in \mathcal{C}} [\![I(v ::= c)]\!]$

$I_1 \vee I_2$ means that either I_1 or I_2 is satisfied. $I_1 \wedge I_2$ means that both I_1 and I_2 are satisfied (in any interleaving). $I_1 \cdot I_2$ means that I_1 is satisfied before I_2 (thus both are satisfied).

Semantics 3 $[\![I_1 \vee I_2]\!] = [\![I_1]\!] \cup [\![I_2]\!]$

Semantics 4 $[\![I_1 \wedge I_2]\!] = [\![I_1]\!] \cap [\![I_2]\!]$

Semantics 5 $[\![I_1 \cdot I_2]\!] = \{\tau_1\tau_2 \in \mathbf{U}_{\mathcal{I}} : \tau_1 \in [\![I_1]\!]$ and $\tau_2 \in [\![I_2]\!]\}$

Elsewhere [10], we present a set of equations that enable symbolic reasoning on $\mathcal{I}$ to determine when a certain event may be permitted, prevented, or triggered.