Communication Meaning: Foundations and Directions for Systems Research

Blue Sky Ideas Track

Amit K. Chopra
Lancaster University
Lancaster, UK
amit.chopra@lancaster.ac.uk

Samuel H. Christie V
North Carolina State University
Raleigh, NC, USA
schrist@ncsu.edu

ABSTRACT

The multiagent software research program envisaged putting multiagent abstractions and methodologies at the heart of designing intelligent distributed applications. In particular, with the aim of enabling flexible interactions between agents, it emphasized modeling *communication meaning*. After three decades of work, the program can claim little broad impact. Has the program been a failure?

We think the program has seen remarkable successes, the biggest being the recent work on information protocols, which finally makes it possible to realize the promise of modeling meaning. In this paper, in support of our claim, we set out how information protocols advance the cause of meaning. We then argue that these advances strike against conventional systems wisdom, including in fields such as networks, distributed systems, and programming languages. Finally, we lay out an ambitious research agenda that puts multiagent abstractions at the heart of systems research. Now is a good time to double down on MAS software research.

CCS CONCEPTS

• **Computing methodologies → Multi-agent systems**; **Distributed programming languages**; • **Networks → Layering**.

KEYWORDS

Asynchronous messaging, Protocols, Norms, Fault tolerance

ACM Reference Format:
Amit K. Chopra and Samuel H. Christie V. 2023. Communication Meaning: Foundations and Directions for Systems Research: Blue Sky Ideas Track. In *Proc. of the 22nd International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2023), London, United Kingdom, May 29 – June 2, 2023*, IFAAMAS, 6 pages.

1 INTRODUCTION

Historically, a central research program in multiagent systems (MAS) has had to do with general-purpose software that enables building intelligent distributed systems [24, 49]. It was motivated by the idea that multiagent software would capture autonomy—flexible, decentralized decision making—in a way that conventional software approaches could not [28, 31]. With decentralization in mind, the MAS software program focused on themes that emphasized modeling interactions between agents. These themes informed multiagent software engineering methodologies and inspired work on MAS platforms and programming models.

Today, the fate of multiagent software research hangs in the balance. Decades of work has had little recognizable impact. In particular, there appears to be a chasm between MAS abstractions and software practice, even though the kinds of applications and paradigms that were envisaged to benefit from MAS, e.g., the IoT, microservices, electronic business engagements, and so on, have proliferated.

Has the multiagent software program been a failure?

No—we think the program can boast notable successes. Like many research strands in AI, the MAS software program began with ambitious, open-ended questions. Progress has naturally been uneven and for sure there have been notable bumps (among them the idea of agent communication languages). However, in the last ten years or so, we think the ambition has paid off spectacularly via fundamental advances on the all-important question of *communication meaning* [40], which is the key to flexible, decentralized decision making. At the heart of these advances lies Singh's breakthrough of modeling MAS via declarative *information protocols* [41–43]. Before Singh came up with information protocols, there existed no general-purpose, operational abstraction for modeling decentralized systems [11]. We can now claim multiagent abstractions to be foundational to engineering distributed applications.

In this paper, we elaborate on our claims above, focusing on how information protocols make communication meaning practical and how that impacts research in important and well-established *systems* fields such as programming languages, networks, and distributed systems. Looking toward the future, we propose the idea of a *multiagent operating system* (MOS). A MOS facilitates an agent's participation in decentralized applications on the basis of communication meaning. Like any OS, a MOS abstracts over lower-level resources and services with the twin aims of making programming convenient and supporting monitoring and execution. We sketch out an ambitious research agenda—in terms of the programming abstractions and services a MOS could provide—that would put multiagent systems at the heart of systems research.

2 ADVANCES IN COMMUNICATION MEANING

Autonomy implies the ability of agents to interact flexibly with other agents. Recognizing this, early MAS software research focused on abstractions that enable flexible interactions. A prevailing

Proc. of the 22nd International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2023), A. Ricci, W. Yeoh, N. Agmon, B. An (eds.), May 29 – June 2, 2023, London, United Kingdom. © 2023 International Foundation for Autonomous Agents and Multiagent Systems (www.ifaamas.org). All rights reserved.

idea from AI was that if agents understood the meaning of their interactions, then they could interact flexibly. Early work on meaning in agent communication languages was influenced by a conception of *speech acts* that emphasized the mental states of the interacting agents [21, 22]. However, as Singh pointed out, an agent's mental state being unverifiable, such approaches were not conducive to building open MAS [40]. Singh instead proposed specifying protocols based on social commitments between agents. This led to a body work on protocols [1, 23, 35, 51, 52], whose significant consequence was to emphasize declarative representations in terms of social abstractions such as commitments, but more generally, in terms of norms [5]. In general, this body of work employed logic program-style representations such as the event calculus [33] and causal logic [25].

However, a thorny operational challenge remained: *How can agents in a MAS enact protocols in a decentralized manner—via asynchronous messaging, without relying on centralized state or coordination?* Traditional representations of protocols, e.g., state machines, specified constraints on message ordering via control flow abstractions. Underlying such representations were some operational intuitions about when communication events may occur. For example, it seems reasonable to model that a *Receipt* may only occur after *Payment* and that *Accept* and *Reject* (of an *Offer*) are mutually exclusive. Commitments didn't capture such operational intuitions. However, the problem with traditional representations is that only highly regimented protocols could be faithfully implemented in decentralized settings [3, 6, 7]. For example, one might imagine that is there nothing fundamentally wrong about *Payment* by B (buyer) and *Delivery* by S (seller) happening concurrently; however, control flow-based approaches reject a protocol that entertains such concurrency as ill-formed [11, 19]. In a nutshell, while there was a need for an operational layer to underpin commitments, the traditional approaches were unsuitable.

To address this challenge, Singh invented declarative information protocols. Information protocols eschew the specification of message ordering in favor of specifying *information causality*. Specifically, an agent can send any message as long as the information dependencies specified in the message's schema are satisfied by the agent's *information state* (its history of interactions).

Listing 1: An information protocol.

```
Flexible Purchase
 role B, S
 parameter out ID key, out item, out del, out paid

 B ↦ S: Request[out ID, out item]
 S ↦ B: Delivery[in ID, in item, out del]
 B ↦ S: Payment[in ID, in item, out paid]
```

For example, by Listing 1, S can send an instance of *Delivery* if it already knows the bindings for ID and item from prior interactions (both adorned ⌜in⌝); it can generate any binding for del (adorned ⌜out⌝). This protocol supports concurrent *Payment* and *Delivery*: once B has sent a *Request*, it can send the corresponding *Payment* and once S has received *Request*, it can send the corresponding *Delivery*. Specifying causality explicitly as described above means messages can be received in any order. This truly liberates decision making. For example, even though B can emit *Payment* only after emitting *Request*, *S* can receive *Payment* first. Further, whenever S receives *Payment*, regardless of whether it has received *Request*, *S* may emit *Delivery* (because its information dependencies would be satisfied). Further, retransmissions of messages by agents and receptions of duplicates are harmless because, information-wise, they are idempotent [42]. *Thus information protocols can be flexibly enacted over unordered, lossy communication services.*

Listing 2: A commitment specification.

```
commitment PurchaseCom B to S
 create Request
 detach Delivery within 1 day
 discharge Payment within 1 day
```

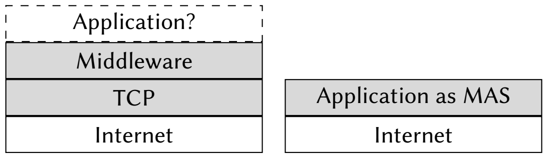
By enacting flexible information protocols, agents compute base-level communication events, e.g., *Payment* and *Delivery*, which may be thought of as the atoms of meaning. Higher-level meaning can be layered on top. Listing 2 shows a commitment, specified in Cupid [12], that refers to the communication events from the protocol in Listing 1; B and S compute commitment events (e.g., discharged or violated) locally based on the base events they have observed.

3 AGAINST CONVENTIONAL SYSTEMS WISDOM

MAS advances in meaning challenge the conventional wisdom in systems research, which emphasizes fixed message ordering and fault tolerance in communication services regardless of application requirements. E.g., TCP (the de facto transport service for Internet applications) [48] and message queues (the holy cow of business messaging) [2] both guarantee reliable, FIFO-ordered communication. Underlying such communication services is a mindset that ties programming convenience with synchrony achieved by enforcing a global ordering of the events the communications represent. However, the end-to-end argument [36] anticipated decades ago that (1) in providing complex guarantees communication services limit application-level decision making and hit performance; and (2) programming based on communication meaning ("application semantics") was the way to avoid these problems. For example, a FIFO service will not deliver *Payment* to S before it has delivered *Request*, which means that the possibility of *S* emitting *Delivery* upon the receipt of just *Payment* is ruled out. In essence, the FIFO service expends resources to implement synchronization thought to be needed by the application but in reality interferes with the application. In general, delaying the delivery of a message (to the application) pending the delivery of another is a deeply flawed idea.

Interestingly, the question of meaning was hotly debated in the systems community [10]. However, meaning-based representations of applications have remained elusive to systems researchers. As Internet pioneer Clark [17] makes clear (after noting that TCP was not ideal from the point of view of meaning): "*The alternative (to TCP) is to push to the app the implementation of the desired semantics (over UDP)..., but then the (app) designer is implementing the protocol... and we don't know how to do that.*" Meaning is no less than the holy grail of distributed systems. However, where systems research has given up, MAS research has excelled. Figure 1 captures the potentially transformative effect of MAS advances in meaning.

MAS advances in meaning challenge the conventional wisdom in work on communication-oriented abstractions in programming languages, which remains beset by a deep-seated, synchronous



(a) Typical current stack. **(b) Meaning-based stack.**

Figure 1: The typical network stack offers message ordering and reliability guarantees via complex transport and middleware layers but ignores applications. In doing so, it interferes with applications and introduces inefficiencies. MAS approaches can fix these problems by modeling communication meaning.

mindset reflected in the idea that sends and receives of messages must interlock as in a zipper. It is for this reason that Erlang [4] and Go [18], both popular as languages suited to programming distributed systems, assume FIFO communication between processes. Moreover, both languages enable application programmers to enforce arbitrary message reception orders by selectively receiving messages. The effect of selective reception is to potentially delay the reception of messages, which, as we argued before, is a flawed idea. The correct approach would be to make message receptions transparent to and decoupled from the application logic. Information protocols support such an approach by letting agents receive messages in any order.

The problem of structuring concurrent, distributed programs has inspired a vast body of work on languages for specifying interaction protocols and programming models based on them, best exemplified by the session types approach [26]. However, these approaches specify a protocol in terms of global message orders, which makes them unsuited to flexible decision making [11].

4 RESEARCH AGENDA

The layers lost in moving from the stack of Figure 1a to that of Figure 1b provided abstractions and services that supported the engineering and execution of distributed applications. *What replaces those layers in Figure 1b*? Building upon the recent advances in meaning, we propose the idea of a multiagent operating system (MOS) as a platform for developing, running, and monitoring MAS specified in terms of meaning. Conceptually, a MOS sits within the agent (Figure 2, a blowup of Figure 1b); it abstracts away the network (which may be unordered and unreliable) and maintains the agent's *local state* (its information state and the states of its commitments, norms, and so on); and it provides a high-level, meaning-oriented interface for plugging in the agent's internal decision making. Below, we elaborate on important research themes and questions.

4.1 Programming Models

What kind of programming model enables flexible decision making?

From the perspective of other agents in a MAS, an agent's communications are its decisions. Meaning specifications collectively define the decisions an agent can legally make in any local state. Since the MOS tracks the local state, it knows the available decisions. The challenge is to come up with a MOS-supported decision

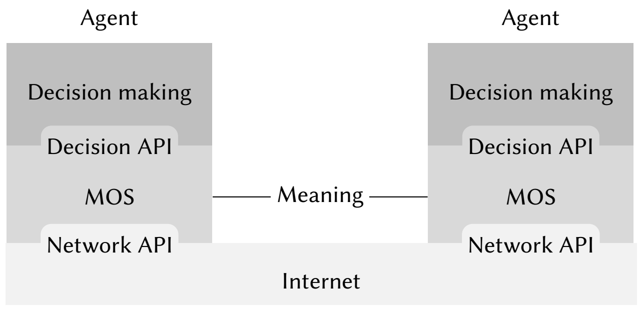


Figure 2: A MOS abstracts away the network and provides a high-level decision making-oriented interface for agent programming.

API that enables driving decisions based on arbitrary internal business logic. For example, a buyer agent's business logic may be that from all the available *Payment* decisions, it makes those that it is committed to make. In view of this, the programming model should natively support norm-based decision making. It should be expressive (it must not rule out reasonable patterns of business logic) and it should be convenient and natural to the programmer. Further, the programming model must not only guarantee correct communications, it should also make it virtually impossible to write incorrect business logic. Mandrake [13] and Kiko [16] represent preliminary work in this direction.

The programming model could be extended to business contracts, which domains such as finance are interested in from the point of view of automating trades. The advances in meaning promise a more suitable foundation for business contracts than blockchain and smart contracts [46].

Although the information protocol approach is more expressive than protocol languages based on control flow, from the perspective of meaning, it is at the level of assembly language, which means getting complex, flexible protocols right can be difficult. High-level languages from which information protocols can be generated, as exemplified by Clouseau [45], would be valuable. Naturally, a novel language would motivate novel programming models based on the constructs in the language.

4.2 Fault Tolerance

Faults are application-level phenomena tied to meaning and therefore agents must be programmed for fault tolerance. Today though, based on the thinking that faults should be transparent to application programmers, we generally lack programming model support for fault tolerance. *What are some common classes of faults and how can the MOS support fault tolerance?*

Faults could be characterized as the violation of communication expectations [14]. At the information protocol level, a fault could be missing or inconsistent information. At higher levels of meaning, a fault could be the expiry or violation of a norm. An idea worth exploring is extending the adapter with standardized fault handling protocols, e.g., for forwarding messages, for exercising accountability (in case of norm violations), and so on. Mandrake supports novel application-level fault tolerance mechanisms.

Is the traditional fault tolerance work relevant for multiagent systems? Fault tolerance in current distributed computing is dominated by an overarching concern: how to make a service both highly available and highly consistent in the presence of faults [8]. The solutions are centered on service replication and vary on how they tradeoff consistency and availability. It is not clear how the traditional work applies in a multiagent setting since a MAS is not a service but a decentralized system of agents who could be thought of as providing services. Perhaps then the traditional work could be applied toward replicating agents. We think a better bet is to take inspiration from real-world organizations, as MAS research has traditionally done, e.g., as in [27]. Specifically, a highly available agent could itself be an organization with an internal structure populated by agents who act on its behalf when dealing with external agents. The internal structure would support an agent taking over from a "failed" agent via organizational patterns such as delegation, escalation, and so on [32, 34, 47].

4.3 Performance

What innovations in MOS architecture, hardware support, algorithms, encodings, and so on, result in high performance agents and MAS?

Performance has traditionally been a neglected area in MAS programming work. Ideally, we want programming models that not only simplify programming but also offer adequately high performance and scalability. The activity concerns identifying performance bottlenecks and optimizations and tradeoffs suitable for certain classes of applications. For example, to save on storage and querying costs, information about "old" interactions could be moved to archival storage [15]. The tradeoff, of course, is that retreiving information from archive would be costly. Such a tradeoff might be worthwhile if retrievals were rare occurrences. To give another example, since message order does not matter, multiple messages could be packed in a network packet, thus saving on packet headers. Moreover, the size of messages themselves can be reduced based on what information the recipient is known to know [14].

A novel direction would be to explore performance in the context of specialized hardware, as with IoT applications, where storage, communication, computing, and power are highly constrained. Another idea would be to explore the use of specialized hardware adapted to the operation of MOS.

4.4 Native Language Support and Concurrency

How should communication and concurrency be supported in programming languages? Any communication between agents should be based on a protocol. Erlang and Go, even though they promote messaging-based coordination, do not support protocols. We believe this is the reason why messaging-based coordination remains just as difficult, if not more, than shared memory approaches [50]. Information protocols could be integrated into such languages through libraries (as Mandrake and Kiko do), but first-class information protocol support would likely replace the concepts of threads and processes with agents that communicate exclusively according to protocols through a MOS.

If a programming language supported the meaning-based programming model described in Section 4.1, then a language primitive for message reception (available both in Erlang and Go) becomes unnecessary, leading to simplicity and less buggy applications. This would be a major step forward in distributed systems programming. In fact, just as GOTO statements were considered harmful for undermining static analysis of programs and replaced by structured programming [20], receive statements should be considered harmful and replaced by agents structured as decision makers acting on information and meaning. We think the latter far better represents the ideal of *structured concurrent programming*.

4.5 Computer Networks

IoT applications require asynchronous communications [38, 44]. Applications that must operate with intermittent connectivity have motivated the idea of delay-tolerant networking [9]. Current network stacks are generally ill-suited to such applications. More mundanely, recognizing the limitations of TCP's single network interface-single stream model, enhancements such as multipath TCP [29] and QUIC [30] have been proposed. Since the MOS obviates significant parts of current networks stacks, including TCP (as Figure 1 highlights), and supports storage-backed, asynchronous communication, there is *a broad opportunity for revisiting significant problems and solutions studied by the networking community and coming up with novel MOS-supported solutions.*

In particular, TCP plays an integral role in congestion control, which is vital to the performance of the Internet [48]. *How would a MOS support congestion control and other network-level performance concerns?* The MOS offers an opportunity for reclaiming congestion control from infrastructure based on the use of application-level knowledge to manage information flows. For instance, an agent could prioritize certain actions, send explicit backoff messages, or implement pull-based active requests instead. Instead of using a default implementation that would eventually fossilize and become a layer in the geologic column of architecture (like TCP), the decisions would all be handled as first-class, application-level decisions. Common algorithms for making the decisions, such as LEDBAT (used for BitTorrent over UDP) [37], could be loaded from a library to reduce developer effort without reducing flexibility.

Will we require any additional layers in the stack of Figure 1b? For example, we may need a layer between the Internet and the MOS to handle large messages—to break them down into pieces small enough for transmission over the Internet and then reassemble those pieces into messages. In keeping with the end-to-end principle, for performance reasons, such a layer may even support reliability via retransmissions of the pieces. However, unlike TCP, it need not support FIFO delivery and hence it would be simpler.

5 CONCLUSION

The field of MAS software aspired to address distributed systems in a fundamental manner. MAS research rightly saw communication meaning (a problem Shannon appeared to have thought about [39]) as central to realizing this aspiration. Our message is that the work on information protocols means that MAS research can today claim to have laid the conceptual foundations for addressing the problem of communication meaning. And in that claim lies the potential to transform systems research broadly conceived, to not just give it a multiagent flavor, but a multiagent foundation.

Acknowledgments. Our vision is informed by two decades of joint work and discussions with Munindar Singh, to whom we owe an immense intellectual debt. We are grateful to the anonymous Senior Program Committee member for seeing merit in our submission. Chopra was supported by the EPSRC grant EP/N027965/1.

REFERENCES

[1] Marco Alberti, Federico Chesani, Marco Gavanelli, Evelina Lamma, Paola Mello, and Paolo Torroni. 2008. Verifiable agent interaction in abductive logic programming: The SCIFF framework. *ACM Transactions on Computational Logic* 9, 4 (2008), 43.

[2] AMQP. 2007. Advanced Message Queuing Protocol. http://www.nsf.gov/funding/.

[3] Davide Ancona, Daniela Briola, Angelo Ferrando, and Viviana Mascardi. 2015. Global Protocols as First Class Entities for Self-Adaptive Agents. In *Proceedings of the 14th International Conference on Autonomous Agents and Multiagent Systems*. IFAAMAS, Istanbul, 1019–1029.

[4] Joe Armstrong. 2003. *Making Reliable Distributed Systems in the Presence of Software Errors*. Ph.D. Dissertation. Royal Institute of Technology, Stockholm, Sweden.

[5] Alexander Artikis, Marek J. Sergot, and Jeremy V. Pitt. 2009. Specifying norm-governed computational societies. *ACM Transactions on Computational Logic* 10, 1 (2009), 42.

[6] Matteo Baldoni, Cristina Baroglio, Amit K. Chopra, Nirmit Desai, Viviana Patti, and Munindar P. Singh. 2009. Choice, Interoperability, and Conformance in Interaction Protocols and Service Choreographies. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems*. IFAAMAS, Budapest, 843–850.

[7] Matteo Baldoni, Cristina Baroglio, Alberto Martelli, and Viviana Patti. 2006. A Priori Conformance Verification for Guaranteeing Interoperability in Open Environments. In *Proceedings of the 4th International Conference on Service-Oriented Computing (LNCS, Vol. 4294)*. Springer, Chicago, 339–351.

[8] Ken Birman. 2015. Evolution of Fault Tolerance. In *SOSP History Day* (Monterey, California). ACM, New York, 7:1–7:32.

[9] Scott Burleigh, Adrian Hooke, Leigh Torgerson, Kevin Fall, Vint Cerf, Bob Durst, Keith Scott, and Howard Weiss. 2003. Delay-Tolerant Networking: An Approach to Interplanetary Internet. *IEEE Communications Magazine* 41, 6 (2003), 128–136.

[10] David R. Cheriton and Dale Skeen. 1993. Understanding the Limitations of Causally and Totally Ordered Communication. In *Proceedings of the 14th ACM Symposium on Operating System Principles*. Asheville, NC, 44–57.

[11] Amit K. Chopra, Samuel H. Christie V, and Munindar P. Singh. 2020. An Evaluation of Communication Protocol Languages for Engineering Multiagent Systems. *Journal of Artificial Intelligence Research* 69 (2020), 1351–1393.

[12] Amit K. Chopra and Munindar P. Singh. 2015. Cupid: Commitments in Relational Algebra. In *Proceedings of the 29th AAAI Conference on Artificial Intelligence*. Austin, Texas, 2052–2059.

[13] Samuel H. Christie V, Amit K. Chopra, and Munindar P. Singh. 2022. Mandrake: Multiagent systems as a basis for programming fault-tolerant decentralized applications. *Autonomous Agents and Multi-Agent Systems* 36, 16 (2022), 30.

[14] Samuel H. Christie V, Amit K. Chopra, and Munindar P. Singh. 2021. Bungie: Improving Fault Tolerance via Extensible Application-Level Protocols. *IEEE Computer* 54, 5 (May 2021), 44–53. https://doi.org/10.1109/MC.2021.3052147

[15] Samuel H. Christie V, Amit K. Chopra, and Munindar P. Singh. 2022. Bruno: Garbage-Collecting Business Information. In *Pre-proceedings of the 10th International Workshop on Engineering Multi-Agent Systems (EMAS)*. Auckland, 1–12.

[16] Samuel H. Christie V, Amit K. Chopra, and Munindar P. Singh. 2023. Kiko: Programming Agents to Enact Interaction Protocols. In *Proceedings of the 22th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. IFAAMAS, London, to appear.

[17] David Clark. 2015. The Network and the OS. In *SOSP History Day 2015* (Monterey, California). ACM, Article 11, 19 pages.

[18] Russ Cox, Robert Griesemer, Rob Pike, Ian Lance Taylor, and Ken Thompson. 2022. The Go Programming Language and Environment. *Commun. ACM* 65, 5 (2022), 70–78.

[19] Nirmit Desai and Munindar P. Singh. 2008. On the Enactability of Business Protocols. In *Proceedings of the 23rd Conference on Artificial Intelligence (AAAI)*. AAAI Press, Menlo Park, 1126–1131.

[20] Edsger W. Dijkstra. 1968. Letters to the Editor: Go to Statement Considered Harmful. *Commun. ACM* 11, 3 (mar 1968), 147–148. https://doi.org/10.1145/362929.362947

[21] Tim Finin, Richard Fritzson, Don McKay, and Robin McEntire. 1994. KQML as an Agent Communication Language. In *Proceedings of the International Conference on Information and Knowledge Management*. ACM Press, 456–463.

[22] FIPA. 2002. FIPA Agent Communication Language Specifications. FIPA: The Foundation for Intelligent Physical Agents, http://www.fipa.org/repository/aclspecs.html.

[23] Nicoletta Fornara and Marco Colombetti. 2004. A Commitment-Based Approach To Agent Communication. *Applied Artificial Intelligence* 18, 9-10 (2004), 853–866.

[24] Les Gasser. 1991. Social Conceptions of Knowledge and Action: DAI Foundations and Open Systems Semantics. *Artificial Intelligence* 47, 1–3 (Jan. 1991), 107–138.

[25] Enrico Giunchiglia, Joohyung Lee, Vladimir Lifschitz, Norman McCain, and Hudson Turner. 2004. Nonmonotonic causal theories. *Artificial Intelligence* 153, 1-2 (2004), 49–104.

[26] Kohei Honda, Nobuko Yoshida, and Marco Carbone. 2008. Multiparty asynchronous session types. In *Proceedings of the 35th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL)* (San Diego). 273–284.

[27] Jomi Fred Hübner, Jaime Simão Sichman, and Olivier Boissier. 2007. Developing Organised Multiagent Systems using the MOISE$^+$ Model: Programming Issues at the System and Agent Levels. *International Journal of Agent-Oriented Software Engineering* 1, 3/4 (2007), 370–395. https://doi.org/10.1504/IJAOSE.2007.016266

[28] Michael N. Huhns and Munindar P. Singh. 1998. Agents and Multiagent Systems: Themes, Approaches, and Challenges. In *Readings in Agents*, Michael N. Huhns and Munindar P. Singh (Eds.). Morgan Kaufmann, San Francisco, Chapter 1, 1–23.

[29] IETF. 2020. TCP Extensions for Multipath Operation with Multiple Addresses. https://www.rfc-editor.org/rfc/rfc8684.

[30] Jana Iyengar and Martin Thomson. 2021. QUIC: A UDP-Based Multiplexed and Secure Transport. https://datatracker.ietf.org/doc/rfc9000/.

[31] Nicholas R. Jennings. 2000. On agent-based software engineering. *Artificial intelligence* 117, 2 (2000), 277–296.

[32] Özgür Kafalı and Paolo Torroni. 2018. Comodo: Collaborative Monitoring of Commitment Delegations. *Expert Systems with Applications* 105 (Sept. 2018), 144–158. https://doi.org/10.1016/j.eswa.2018.03.057

[33] Robert Kowalski and Marek J. Sergot. 1986. A Logic-Based Calculus of Events. *New Generation Computing* 4, 1 (1986), 67–95.

[34] Timothy J. Norman and Chris Reed. 2001. Delegation and Responsibility. In *ATAL '00: Proceedings of the 7th International Workshop on Intelligent Agents VII. Agent Theories Architectures and Languages*. Springer, Berlin, 136–149.

[35] Jeremy Pitt and Abe Mamdani. 1999. A protocol-based semantics for an agent communication language. In *Proceedings of the International Joint Conference on Artificial Intelligence*. 486–491.

[36] Jerome H. Saltzer, David P. Reed, and David D. Clark. 1984. End-To-End Arguments in System Design. *ACM Transactions on Computer Systems* 2, 4 (Nov. 1984), 277–288. https://doi.org/10.1145/357401.357402

[37] Sea Shalunov, Greg Hazel, Janardhan Iyengar, and Mirja Kuehlewind. 2012. RFC 6817: Low extra delay background transport (LEDBAT). https://datatracker.ietf.org/doc/html/rfc6817

[38] Wentao Shang, Yingdi Yu, Ralph Droms, and Lixia Zhang. 2016. *Challenges in IoT Networking via TCP/IP Architecture*. Technical Report NDN-0038. NDN Project.

[39] Claude E. Shannon. 1948. A mathematical theory of communication. *The Bell System Technical Journal* 27, 3 (1948), 379–423.

[40] Munindar P. Singh. 1998. Agent Communication Languages: Rethinking the Principles. *IEEE Computer* 31, 12 (Dec. 1998), 40–47.

[41] Munindar P. Singh. 2011. Information-Driven Interaction-Oriented Programming: BSPL, the Blindingly Simple Protocol Language. In *Proceedings of the 10th International Conference on Autonomous Agents and MultiAgent Systems* (Taipei). IFAAMAS, 491–498.

[42] Munindar P. Singh. 2011. LoST: Local State Transfer—An Architectural Style for the Distributed Enactment of Business Protocols. In *Proceedings of the 9th IEEE International Conference on Web Services (ICWS)*. IEEE Computer Society, Washington, DC, 57–64.

[43] Munindar P. Singh. 2012. Semantics and Verification of Information-Based Protocols. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. IFAAMAS, Valencia, Spain, 1149–1156.

[44] Munindar P. Singh and Amit K. Chopra. 2017. The Internet of Things and Multiagent Systems: Decentralized Intelligence in Distributed Computing. In *Proceedings of the 37th IEEE International Conference on Distributed Computing Systems (ICDCS)*. IEEE, Atlanta, 1738–1747. https://doi.org/10.1109/ICDCS.2017.304 Blue Sky Thinking Track.

[45] Munindar P. Singh and Amit K. Chopra. 2020. Clouseau: Generating Communication Protocols from Commitments. In *Proceedings of the 34th Conference on Artificial Intelligence (AAAI)*. AAAI Press, New York, 7244–7252. https://doi.org/10.1609/aaai.v34i05.6215

[46] Munindar P. Singh and Amit K. Chopra. 2020. Computational Governance and Violable Contracts for Blockchain Applications. *IEEE Computer* 53 (Jan. 2020), 53–62. Issue 1.

[47] Munindar P. Singh, Amit K. Chopra, and Nirmit Desai. 2009. Commitment-Based Service-Oriented Architecture. *IEEE Computer* 42, 11 (2009), 72–79.

[48] W. Richard Stevens. 1994. *TCP/IP Illustrated, Volumes 1–3*. Addison-Wesley, Reading, Massachusetts.

[49] Katia P. Sycara. 1998. Multiagent Systems. *AI Magazine* 19, 2 (1998), 79–92.

[50] Tengfei Tu, Xiaoyu Liu, Linhai Song, and Yiying Zhang. 2019. Understanding Real-World Concurrency Bugs in Go. In *Proceedings of the Twenty-Fourth International*

Conference on Architectural Support for Programming Languages and Operating Systems (Providence, RI, USA). ACM, 865–878.

[51] Michael Winikoff, Wei Liu, and James Harland. 2005. Enhancing Commitment Machines. In *Proceedings of the 2nd International Workshop on Declarative Agent Languages and Technologies (DALT) (LNAI, Vol. 3476)*. Springer-Verlag, Berlin, 198–220.

[52] Pınar Yolum and Munindar P. Singh. 2002. Flexible Protocol Specification and Execution: Applying Event Calculus Planning using Commitments. In *Proceedings of the 1st International Joint Conference on Autonomous Agents and MultiAgent Systems* (Bologna). ACM Press, 527–534.