Multiagent Foundations for Distributed Systems: A Vision

Amit K. Chopra[1][0000−0003−4629−7594],
Samuel H. Christie V[1,2][0000−0003−1341−0087], and
Munindar P. Singh[2][0000−0003−3599−3893]

[1] Lancaster University, Bailrigg, Lancaster, UK
{amit.chopra,samuel.christie}@lancaster.ac.uk
[2] North Carolina State University, Raleigh, USA
{schrist,singh}@ncsu.edu

Abstract. Early works and retrospectives by the researchers who founded the network protocols underlying current distributed systems indicate they were aware of the importance of capturing application meaning but didn't know how to handle it programmatically. Therefore, those researchers introduced simplifications in the protocols that violated their own principle of the end-to-end argument in systems design.
The thesis of this vision paper is the following. First, the above-mentioned simplifications, especially the reliance on reliable, ordered communication protocols such as TCP have run their course. Modern applications demand flexibility that can only be achieved through modeling application meaning, and many applications (such as those based on the Internet of Things) cannot pay TCP's overhead. Second, the multiagent systems community has developed alternative meaning-based approaches that can provide a new foundation for distributed computing at large.

1 Introduction

As originally conceived, a multiagent system (MAS) is *decentralized* [14]: Agents in a MAS are autonomous computational entities that communicate and share information with each other. In many applications of MAS, an agent represents a real-world party, such as a human or organization, and the autonomy of the agent reflects the autonomy of the party it represents. Distinctly from other areas of computing (e.g., Web services, software engineering, and programming languages), MAS research emphasizes modeling the *meaning* of interactions [19]. Broadly, meaning refers to the information in an engagement between principals that is relevant to their decision making. The focus on meaning has led to a rich body of work on declarative abstractions such as commitments between agents [7, 8, 12, 24, 25]. The early work on commitments demonstrated that modeling meaning is the key to enabling flexible interactions between agents, and thus the key to accommodating their autonomy.

Pioneers in networked computing were aware of the importance of modeling distributed applications in terms of meaning [4]; however, they lacked the

abstractions to express meaning. Instead, to support programming, approaches in distributed computing focused on ordering and reliability guarantees in the infrastructure. Being application-agnostic, such guarantees are meaningless from the application perspective. But worse, the guarantees end up subverting autonomy by restricting the choices available to an agent in interacting with others. And in doing so, modern approaches end up violating the end-to-end argument (E2EA) [18], a fundamental principle of distributed systems. Indeed, Clark [11] explains as much in a retrospective on Internet protocols and distributed systems.

We claim that the MAS community's historical focus on autonomy and meaning has the potential to address a central quest in distributed systems. In a nutshell, the quest is for programming abstractions that enable programmers to easily build high-performance distributed applications based on *meaning* in a manner compatible with the E2EA. By applying ideas from MAS, we have the opportunity to fundamentally reshape how practitioners build distributed application.

2 The Dilemma Posed by Current Approaches

We discuss how each of the two major existing approaches (architectures) for distributed applications fails to satisfy crucial architectural desiderata, thus presenting developers with a *dilemma* (a situation with two equally bad choices).

2.1 Desiderata

Consider the two MAS architectures in Figures 1 and 2. In both, the agents communicate via asynchronous messaging via a communication infrastructure that offers an API for programming agents. Notice that there is no shared state between the agents. In the architecture of Figure 1, the communication infrastructure guarantees only that it delivers only sent messages. We refer to such an infrastructure (and architecture) as *bare-bones* because no real infrastructure guarantees less. In the architecture of Figure 2, the infrastructure provides the additional guarantee that all sent messages will be delivered and in FIFO order between any pair of agents. We refer to such an infrastructure (and architecture) as *reliable*. In practical systems, bare-bones and reliable infrastructures are exemplified by UDP over IP and TCP over IP, respectively.

Accommodating Autonomy. The end-to-end argument (E2EA) [18] is a guiding principle in the design of the Internet. The principle imagines a layered system architecture and draws our attention to the fact that if implementing some functionality fully and correctly requires knowledge only available at some system layer, then that functionality cannot be implemented in a lower system layer. Partial implementation of the functionality in a lower layer, however tempting, should generally be avoided as the layer would impose a model upon the higher layer (by constraining the choices available at the higher layer) and

Fig. 1: MAS where agents communicate via a bare-bones infrastructure that guarantees neither message ordering nor delivery. The API represents programming abstractions offered by the infrastructure.

Fig. 2: MAS where agents communicate via a reliable infrastructure that guarantees message delivery and in FIFO order.

likely result in a performance hit as well. The E2EA famously argues against reliable infrastructures (as defined above), among other things.

For example, suppose that we wanted to make a medical prescription application reliable in the sense that a prescription written by the doctor in response to a patient complaint should be fulfilled by the pharmacy in a timely manner. To distinguish application reliability from reliability at the infrastructure level, let's refer to the former as *a-reliability*. We can imagine a few measures to increase a-reliability. One, we could specify a contract that stipulates that the pharmacy fulfill valid prescriptions in a timely manner. Further, we could support reminders and acknowledgments between the parties. A reliable infrastructure would be oblivious of such measures—they would necessarily have to be supported at the application level. In particular, no infrastructure-level retransmission or acknowledgment of messages can provide a-reliability, because it depends on the cooperation of multiple higher-level endpoints (the agents).

More insidiously perhaps, FIFO delivery interferes with the application by delaying the delivery of messages pending the arrival of an earlier message. For example, if a doctor sends two prescriptions to the pharmacy, one after the other, then until the first prescription is delivered to the pharmacy, the infrastructure won't deliver the second, thus interfering with the pharmacy's fulfillment of its commitments and its autonomy (the idea that infrastructures could interfere with agent autonomy was first articulated in [5]). Consider another example where the doctor can cancel a prescription after issuing it to the pharmacy. FIFO delivery would mean that the pharmacy can't process the cancellation before receiving the prescription, even though handling cancellation first might avoid wasting effort and so would be desirable from the pharmacy's point of

view. In essence, a-reliability does not require reliable infrastructure, but the reliable infrastructure gets in the way of a-reliability.

From the application developer's perspective, the E2EA promotes the idea of representing application meaning (e.g., the meaning of a prescription and its cancellation) and implementing agents based on such meaning rather than some expected message ordering.

Programming Convenience. Historically though, reliable infrastructures have been favored over bare-bones ones for building applications. The reason is that in the mind of an application developer, the application is represented as a unitary (as opposed to decentralized) state machine. For example, the developer implicitly models a state machine where the doctor's cancellation of a prescription happens after the prescription is issued (Figure 3). A state machine is a convenient abstraction from the point of view of programming. A reliable infrastructure helps implement such a state machine in a distributed manner by making it impossible for the pharmacy to observe and process the cancellation before observing the prescription.

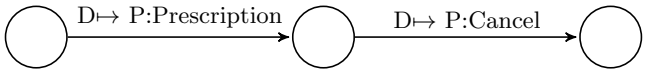

Fig. 3: Fragment of a state machine representing a medical prescription scenario.

Imagine programming such an application over bare-bones infrastructure. Now, the application developer must implement the pharmacy to deal with cancellation arriving before the prescription. In general, prevalent techniques offer no alternative but to implement business logic for each possible message sequence, a cumbersome and error prone task at best, especially when several messages may be in transit at once.

In addition to preventing arbitrary message orders (or *nondeterminism*), a reliable infrastructure also saves the application developer from writing logic to recover from lost messages. Over bare-bones infrastructure, the developer would have to implement acknowledgments, retransmissions, and the handling of duplicates. It is no wonder then that application developers prefer reliable infrastructures.

Loose Coupling. Loosely-coupled architectures are better because a component can be replaced with fewer modifications to other components [17, 23]. From the point of view of loose coupling, the bare-bones architecture is better. This is because the applications that work over a bare-bones infrastructure will also work over the reliable infrastructure. However, the reverse is not true. If agents relied on the reliability offered by the infrastructure, then switching to

bare-bones infrastructure would result in errors. As we saw above, if the pharmacy were implemented to expect Cancel after Prescription, then the pharmacy wouldn't work over a bare-bones infrastructure.

2.2 The Dilemma

From the foregoing discussion, it would seem that agent programmers are faced with a dilemma.

1. Either build flexible applications in a manner compatible with the E2EA—over bare bones infrastructure—but without the benefit of high-level, meaning-based programming abstractions. In particular, programmers would have to implement complex code to track the state of the interaction.
2. Or benefit from some programming convenience, but at the cost of violating the E2EA and subverting autonomy.

Neither alternative is ideal since we really want both high-level communication abstractions *and* compatibility with the E2EA. Work in distributed systems, however, has historically favored Alternative 2, as evidenced by work on *middleware*. Message queues (e.g., MQTT) support reliable, FIFO messaging. Causal delivery generalizes FIFO delivery to more than two endpoints. RPC (remote procedure call), a technique whose limitations were laid bare decades ago, has made a comeback with microservices.

3 Meaning-Based MAS Architecture

In contrast to traditional approaches for creating distributed applications, a strand of MAS research has emphasized modeling a MAS in terms of the *meaning* of interactions between agents. The motivation behind modeling meaning is to support flexible decision-making by enabling flexible interactions between agents.

In current work, the meaning is usually modeled in terms of how messages affect the states of the normative expectations (*norms*, e.g., *commitments*) between agents [7, 8, 12, 24, 25]. Recent work has demonstrated how the decentralized computation of norms may be operationalized over information protocols [16, 20, 21]. In a nutshell, agents compute the atoms of meaning, or *base events*, by enacting information protocols. The base events an agent has observed are materialized in its *local state*. Each agent computes higher-level meanings as views on the local state.

Figure 4 describes a promising meaning-based MAS architecture schematically. Several things are notable about the architecture. One, the application is specified by norms and information protocols; collectively, the *interaction specification*. The specification would be jointly determined by application stakeholders following some design process, e.g., [6]. Two, the interaction specification is the extent of the coupling between the agents. There is no hidden coupling between the agents. In particular, nothing is assumed of the communication infrastructure except that it respects physical causality; that is, the infrastructure delivers only sent messages.

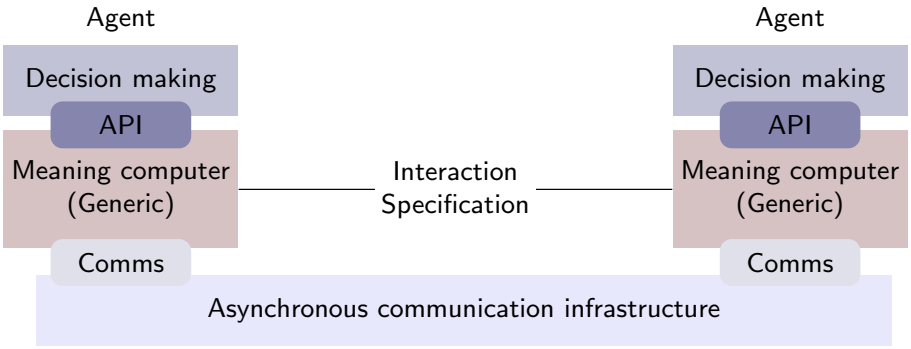

Fig. 4: Meanings-based MAS. An agent's local meaning computer computes meanings based upon a specification of interactions, here, norms and information protocols. The meaning computer offers a high-level API which a developer can use to plug in decision making policies. The meaning computer is generic and interfaces with an asynchronous communication infrastructure via a low-level communication interface (Comms).

Each agent consists of two components: *Meaning Computer* (MC) and *Decision making*. The MC is a generic component. It interprets the information protocol and ensures that the agent is compliant with it. It also records the incoming and outgoing messages, collectively the base events that the agent has observed. Further, the MC interprets norm specifications over the base events to infer the states of the norms. An agent developer would plug in the agent's decision making policies via an API to the MC.

Returning to the example of prescription cancellation, the architecture in Figure 4 enables the possibility of the pharmacy handling the prescription cancellation before receiving the prescription if the cancellation is received first. As modeled in the information protocol, the cancellation would refer to the prescription being canceled via a unique identifier. If the cancellation is received before the prescription, then the pharmacy's MC disables the fulfillment of the prescription (based on the information protocol) so that when the prescription eventually arrives, there is nothing to do.

The information protocol approach represents a key breakthrough in protocol languages in that it supports meaning and flexible interaction far better than choreographic (message ordering-based) approaches [5]. Remarkably, although a choreography is an application-level abstraction, its reliance on message ordering for correctness recreates the problems of reliable infrastructures at the application level.

4 Directions

4.1 Specification

Natural language *contracts* (agreements) capture the high-level meaning of engagements between parties by setting out the relevant *norms*, e.g., commitments, authorizations, prohibitions, and powers. The virtue of a contract is that it supports both autonomy and correctness. That is, a party may decide to act as it pleases; however, if it violates a norm, that would amount to an observable violation.

Smart contracts (the blockchain variety) have caught the world's attention for not having that virtue. Their motivation is to cut out the social aspects of decision making by touting inviolability [22]. Inviolability though is antithetical to autonomy, which is probably one reason why smart contracts have not caught on as a general purpose technology. Herein lies a great opportunity for MAS research—declarative representations of violable contracts—to make a real-world impact.

Our contributions include *Cupid*, a declarative language for specifying and computing norms over a database of business events [7, 8]. *Clouseau* [21] shows how to leverage Cupid contracts in a decentralized setting with several agents, each with its own local database. We also developed a proof-of-concept implementation of Cupid for the R3 Corda distributed ledger [22]. More is needed for practical applications. In particular, a contract bundles norms and has its own lifecycle (it enters into force upon parties signing up to it and it may be amended, breached, and terminated). Further, contracts involve operations such as delegation and assignment and notions such as jurisdictions that need to be properly formalized.

We need methodologies for specifying and verifying contracts. One important question is how may stakeholders starting from their requirements arrive at a contract. Protos [6], a methodology for refining requirements into commitments, offers some ideas. A broader question is that of *governance*, which requires taking into account the actual outcomes from enacting a contract in the process of revising a contract. Further, contracts need to be to related to multiagent organizations (institutions). An organization (itself an agent) would normally serve as the arbiter of disputes and provide other services such as identity, discovery, and reputation. An organization may further help enforce contracts by sanctioning agents, e.g., by expelling an agent for repeated violations.

A related question is what constitutes a *fair* contract? Consider a contract between a lender and borrower, whereas a notification sent by the lender counts when the lender sends it, whereas a notification sent by the borrower counts when the lender receives it. All other things being equal (e.g., they are using the same communication infrastructure), such a contract seems unfair to the borrower because all decisions are made from the lender's perspective. Other questions relate to the enactability of a contract. For example, a contract may only be partially enactable or it may be enactable only in odd ways (e.g., a commitment which comes into force only after it is already satisfied).

4.2 Programming Models

A challenge for any interaction-based approach of specifying applications is how to facilitate the implementation of agents based on contracts and protocols. In contrast to traditional approaches [1, 2, 3], a suitable programming model would be based on information, thus abstracting away the challenges of asynchrony. It would also ensure that the interactions progress in decentralized yet consistent manner. We have explored some initial ideas in Stellar [13] and PoT [10], which demonstrate that an information-based programming model saves significant programming effort and avoids errors. A yet uncharted area is how to support programming based on contracts.

Early work on commitment machines identified semantic exception handling as a benefit [25]. Exception handling naturally relates to the theme of fault tolerance. The remarkable thing is that application-level fault tolerance is not optional; any application must ensure that it achieves its own objectives. Although properly addressing causes may reduce the probability of failure and improve performance, what ultimately matters to an application is success. However, current approaches (following a long tradition) focus on handling faults as close to their causes as possible, and thus encourage delegating fault tolerance to the infrastructure. For example, in a paradigm as new as microservices, fault tolerance is left to the underlying service mesh [15]. The focus on infrastructure has meant that today we lack the tools to program fault tolerance effectively at the application level. PoT and Bungie [9] present some initial ideas about how to implement fault tolerance at the application level—in the agents.

Most future applications will be programmed to run in the cloud, possibly as a composition of microservices. It would be timely for MAS researchers to consider how their techniques could benefit from cloud-based mechanisms (e.g., for scalability) and what they might in turn have to offer to application developers. Programming models such as Function-as-a-Service (FaaS) intend to make programming cloud applications easier. However, such models currently offer neither any programming abstractions for managing state nor composition mechanisms for building realistic applications. Meaning-based programming models can help address these gaps and potentially enable highly concurrent agent implementations that can take advantage of scalability mechanisms in the cloud.

5 Conclusion

Current approaches for building distributed applications pose a dilemma: Either build applications in violation of the E2EA or build them without any programming support. A fundamentally multiagent approach based on meaning has the potential to provide the way out: satisfy the E2EA by enabling the deployment of applications on bare-bones infrastructure, and facilitate programming via high-level programming abstractions.

The multiagent systems community has long expressed angst about the lack of direct impact on systems development practice. We suggest here that perhaps

it is because we have sought to make small incremental changes, which since they don't align well with traditional thinking are largely disregarded by practitioners. We suggest that it would be worth (1) understanding the foundational problems in distributed systems, that is, those that lie beyond the capacity of current approaches and (2) showing how multiagent systems can address those problems in a natural manner. The history of distributed computing indicates that the founders were quite aware of the simplifications, and revisiting those design decisions could be a pathway toward introducing multiagent systems into practice.

Acknowledgments. Grants from the NSF (IIS-1908374) and EPSRC (EP/N027965/1) supported this research.

Bibliography

[1] Baldoni, M., Baroglio, C., Capuzzimati, F.: A commitment-based infrastructure for programming socio-technical systems. ACM Transactions on Internet Technologies **14**(4), 23:1–23:23 (Dec 2014)

[2] Baldoni, M., Baroglio, C., Capuzzimati, F., Micalizio, R.: Type checking for protocol role enactments via commitments. Autonomous Agents and Multi-Agent Systems **32**(3), 349–386 (2018)

[3] Boissier, O., Bordini, R.H., Hübner, J.F., Ricci, A., Santi, A.: Multi-agent oriented programming with JaCaMo. Science of Computer Programming **78**(6), 747–761 (Jun 2013)

[4] Cheriton, D.R., Skeen, D.: Understanding the limitations of causally and totally ordered communication. In: Proceedings of the 14th ACM Symposium on Operating System Principles (SOSP). pp. 44–57. ACM Press, Asheville, North Carolina (Dec 1993). https://doi.org/10.1145/168619.168623

[5] Chopra, A.K., Christie V, S.H., Singh, M.P.: An evaluation of communication protocol languages for engineering multiagent systems. Journal of Artificial Intelligence Research **69**, 1351–1393 (2020)

[6] Chopra, A.K., Dalpiaz, F., Aydemir, F.B., Giorgini, P., Mylopoulos, J., Singh, M.P.: Protos: Foundations for engineering innovative sociotechnical systems. In: Proceedings of the 18th IEEE International Requirements Engineering Conference. pp. 53–62 (2014)

[7] Chopra, A.K., Singh, M.P.: Cupid: Commitments in relational algebra. In: Proceedings of the 29th AAAI Conference on Artificial Intelligence. pp. 2052–2059 (2015)

[8] Chopra, A.K., Singh, M.P.: Custard: Computing norm states over information stores. In: Proceedings of the 15th International Conference on Autonomous Agents and Multiagent Systems (AAMAS). pp. 1096–1105. IFAAMAS, Singapore (May 2016)

[9] Christie V, S.H., Chopra, A.K., Singh, M.P.: Bungie: Improving fault tolerance via extensible application-level protocols. IEEE Computer **54**(5), 44–53 (May 2021)

[10] Christie V, S.H., Smirnova, D., Chopra, A.K., Singh, M.P.: Protocols over Things: A decentralized programming model for the Internet of Things. IEEE Computer **53**(12), 60–68 (2020)

[11] Clark, D.: The network and the OS. In: SOSP History Day. pp. 11:1–11:19. ACM, Monterey, California (Oct 2015). https://doi.org/10.1145/2830903.2830912

[12] Fornara, N., Colombetti, M.: Operational specification of a commitment-based agent communication language. In: Proceedings of the 1st International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS). pp. 535–542. ACM Press (Jul 2002)

[13] Günay, A., Chopra, A.K.: Stellar: A programming model for developing protocol-compliant agents. In: Preproceedings of the 6th International

Workshop on Engineering Multi-Agent Systems. LNCS, vol. 11375, pp. 117–136. Springer, Stockholm (2018)
[14] Huhns, M.N. (ed.): Distributed Artificial Intelligence. Pitman/Morgan Kaufmann, London (1987)
[15] Istio: Introducing Istio: A robust service mesh for microservices. `https://istio.io/v0.1/blog/istio-service-mesh-for-microservices.html` (May 2017), accessed: 16 Jun 2021
[16] King, T.C., Günay, A., Chopra, A.K., Singh, M.P.: Tosca: Operationalizing commitments over information protocols. In: Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI). pp. 256–264 (2017)
[17] Microsoft: Microservice architecture style. `https://docs.microsoft.com/en-us/azure/architecture/guide/architecture-styles/microservices` (Oct 2019), accessed: 16 Jun 2021
[18] Saltzer, J.H., Reed, D.P., Clark, D.D.: End-to-end arguments in system design. ACM Transactions on Computer Systems **2**(4), 277–288 (Nov 1984). https://doi.org/10.1145/357401.357402
[19] Singh, M.P.: Agent communication languages: Rethinking the principles. IEEE Computer **31**(12), 40–47 (Dec 1998)
[20] Singh, M.P.: Information-driven interaction-oriented programming: BSPL, the Blindingly Simple Protocol Language. In: Proceedings of the 10th International Conference on Autonomous Agents and MultiAgent Systems. pp. 491–498 (2011)
[21] Singh, M.P., Chopra, A.K.: Clouseau: Generating communication protocols from commitments. In: Proceedings of the 34th AAAI Conference on Artificial Intelligence. pp. 7244–7252. AAAI Press, New York (2020)
[22] Singh, M.P., Chopra, A.K.: Computational governance and violable contracts for blockchain applications. IEEE Computer pp. 53–62 (Jan 2020)
[23] Singh, M.P., Huhns, M.N.: Service-Oriented Computing: Semantics, Processes, Agents. John Wiley & Sons, Chichester, UK (2005)
[24] Winikoff, M.: Implementing commitment-based interactions. In: Proceedings of the 6th International Conference on Autonomous Agents and Multiagent Systems. pp. 1–8 (2007)
[25] Yolum, P., Singh, M.P.: 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. pp. 527–534. ACM Press (2002)