Interaction-Oriented Software Engineering: Programming Abstractions for Autonomy and Decentralization

Amit K. Chopra
School of Computing and Communications, Lancaster University, United Kingdom
E-mail: amit.chopra@lancaster.ac.uk

Abstract. We review the main ideas and elements of Interaction-Oriented Software Engineering (IOSE), a program of research that we have pursued for the last two decades, a span of time in which it has grown from philosophy to practical programming abstractions. What distinguishes IOSE from any other program of research is its emphasis on supporting autonomy by modeling the meaning of communication and using that as the basis for engineering decentralized sociotechnical systems. Meaning sounds esoteric but is the basis for practical decision making and a holy grail for the field of distributed systems. We describe our contributions so far, directions for research, and the potential for broad impact on computing.

Keywords: Autonomy, Decentralization, Meaning, Sociotechnical systems, Multiagent systems, Protocols, Norms, Programming Models, Causality

1. Introduction

We conduct research on the software engineering of *sociotechnical systems*, that is, systems that facilitate interactions between multiple autonomous principals. A principal would normally be either a human or an organization. In contrast to AI approaches that understand autonomy in terms of intelligence and reduce it to automation, our program of research understands autonomy as decentralized decision making and is concerned with software abstractions that enable principals in sociotechnical systems to make decisions in a maximally flexible and loosely-coupled manner.

Conceptually, sociotechnical systems are commonplace. Any business transaction, whether, e.g., in health, finance, or ebusiness, occurs in a sociotechnical system (STS). Internet of Things (IoT) applications, e.g., smart cities, typically span multiple organizations who collaboratively govern devices and the information they produce. The industry-led microservices paradigm represents a real shift away from monolithic application implementations to decentralized systems of autonomous components.

Today we lack programming abstractions that enable systematically realizing sociotechnical systems as decentralized systems. In particular, we lack abstractions that accommodate autonomy, understood as decentralized decision making. *How can we systematically realize sociotechnical systems in a way that accommodates autonomy?* The question concerns practical decision making. It also goes to the heart of the computing discipline, which has traditionally been preoccupied with realizing a system via a conceptually central machine (Figure 1). Traditional software engineering is largely about specifying, composing, verifying, and implementing machines. The field of distributed systems, in particular, is largely concerned with the distributed implementation of conceptually central machines. This becomes obvious when one considers that synchrony—achieved by enforcing a global event ordering—is central to current distributed systems approaches.

Our program of research, *Interaction-Oriented Software Engineering* (IOSE), is a methodology for engineering sociotechnical systems. Its insight is twofold. (1) Decentralization presupposes modeling the rules of encounter

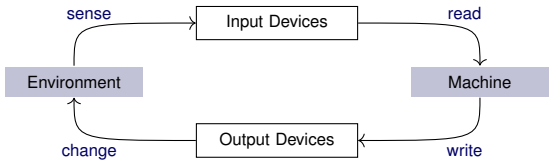


Fig. 1. Sociotechnical system realized via a conceptually central machine [1]

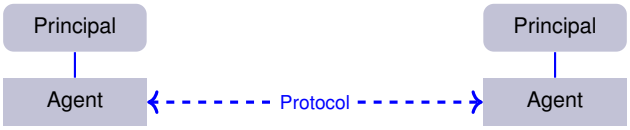


Fig. 2. Sociotechnical system realized as a decentralized multiagent system.

between the principals in a system via *interaction protocols*. Protocols abstract away from the decision-making apparatus of any particular principal. Each principal implements an agent to represent itself in the STS (Figure 2). Further, to support the building of complex systems, systems-as-protocols may be composed. If we understand a protocol as specifying an architecture, then what IOSE supports is composition of architectures. (2) Users make decisions based on the meaning of their interactions. Meaning refers to conceptual objects in the application domain, e.g., the commitments that result from interaction. Supporting flexible decision making therefore requires modeling such meaning. Notably, meaning is a declarative notion founded in information.

As we describe below, our program is founded in languages for modeling protocols, composition and verification of protocols, and programming models for implementing agents as protocol endpoints. It is a program that brings together software engineering, distributed systems, and AI toward the engineering of decentralized sociotechnical systems and in doing so reshapes them all. What makes our research program unique is its emphasis on modeling the meaning of interactions.

IOSE stands apart from Agent-Oriented Software Engineering (AOSE) methodologies in its emphasis on interactions. Some AOSE methodologies included limited forms of control flow-based protocols; however, for the most part, they embody a centralized mindset that deemphasizes autonomy and interactions.

2. The Elements of Meaning

An *interaction protocol* models a decentralized application by specifying the coordination constraints on agents. Supporting flexible decision making requires protocols that support *social meaning* [2], which is what users care about and base their decisions upon. E.g., in an ebusiness transaction between a buyer and a seller, an Offer message specifying an item and a price means the corresponding real-world offer, which itself means a real-world commitment from the seller to the buyer for Delivery of the item if Payment of the price occurs.

We elaborate on important themes below.

2.1. Norms as High-Level Protocols

A norm is a directed social expectation that one principal (the expector) has of another (the expectee) [3]. Norms yield accountability: the expector has standing to demand an account from the expectee for violations [1]. Commitment, power, authorization, and prohibition are exemplar kinds of norms. Accommodating autonomy requires specifying the norms that govern the interactions between the principals. The norms in fact capture the social meaning of interaction, which is the level at which principals understand interactions. Whereas autonomy means a principal may decide as it pleases, norms yield a correctness standard for decisions.

Norms specify high-level protocols and support flexible interactions. E.g., the seller may violate the commitment to do Delivery or do Delivery before it receives Payment. In case of a violation, norms that capture sanctions may be triggered, e.g., that seller should refund 110% of the Payment as compensation. For repeated violations, the seller's trading credentials may be revoked by authorities in the STS. Representing a system in software requires representation of the relevant norms and supporting reasoning about them.

The alternative to specifying norms is regimenting the interactions via mechanisms so that violation is not possible [4]. However, given that our setting is decentralized, such regimentation is generally neither possible nor is it desirable. A decision to violate a commitment is as much an expression of autonomy as a decision to discharge it. Violations may, in fact, be judged as having been beneficial and lead to improvements in the system.

2.2. Information Protocols as Operational Protocols

Once we specify the system in terms of norms, the question of where the states of the norms are being computed arises naturally. Since the system is decentralized, there isn't a distinguished locus of computation that computes the norms. Indeed, the norms must be computed locally by the agents in the system based on shared events they have observed separately. It is not difficult to see in fact that at least the expector and expectee of a norm must compute it, otherwise they will scarcely be interoperable.

To enable the decentralized computing of norms, we need to specify operational (low-level) interaction protocols that produce the base-level events such as Offer, Delivery, and Payment. Focusing on meaning makes clear that the operational constraints on communication may be based only on the information content of messages, specifically: (i) *information causality*, which captures the information dependencies that must be satisfied to emit a message; and (ii) *information integrity*, which captures that no two messages may contain inconsistent information. Examples of both come from the domain. E.g., in any transaction, the seller must know the item in order to do Delivery, and Payment and Delivery must refer to the same item.

Accordingly, we adopt the idea of declarative information protocols [5], a novel approach for specifying protocols in terms of the information content of messages and causality and integrity constraints on them. By enacting information protocols, agents communicate shared events, based upon whose observations, they compute the norms. Notably, information protocols depart from traditional notations and languages for specifying interactions via UML sequence diagrams [6] and choreographies [7].

Focusing on information lets us avoid irrelevant operational restrictions, the most egregious of which is message reception order. This ability is truly liberating because it opens up decision making: If Delivery conveys the information needed to send Payment (e.g., the price), then the buyer may send Payment without receiving Offer. Meaning thus obviates ordered-delivery communication services, which are not only expensive but cause delays and hide meaning in low-level control structures.

Figure 3 shows the resulting architecture schematically. The computation of norms is layered on top of base events produced by information protocols. Each agent's decision making (private) takes advantage of a database of norm events. Messages between agents are transported by an ordered, unreliable (lossy) communication service, e.g., UDP.

2.3. The Triangle of Meaning: Decisions, Communications, Information

An agent's communications represents its *public* decisions—an expression of its autonomy. It complements *internal* decision making based on intelligent algorithms. E.g., internally, a seller's algorithm may determine the price at which it should offer an item for sale. However, the public decision is made when it communicates the Offer. And from the social perspective, it is public decisions that count. The history of an agent's public decisions represents where it stands in the world. Fig 4 captures the centrality of meaning: it lies in information and connects decision making with communication.

High-level meanings are not limited to norms, even if they have been the primary focus of study in multiagent systems. Meaning encompasses any high-level event in the application domain that is relevant to decision making. Importantly, meaning is a matter of specification via protocols.

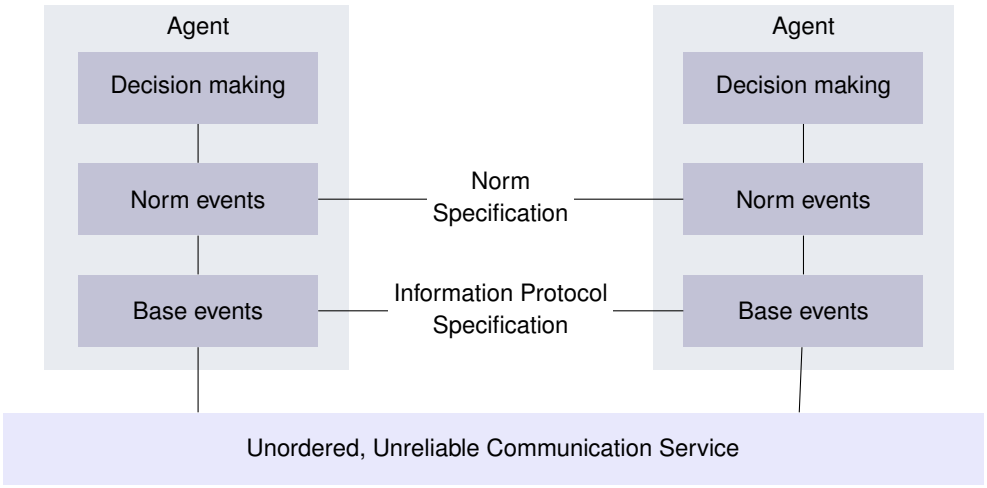


Fig. 3. Layers in a decentralized MAS, schematically.

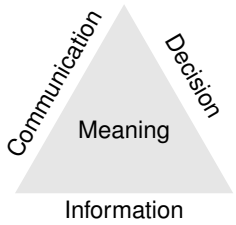


Fig. 4. Meaning Intuition.

3. Contributions

We now describe some technical contributions.

3.1. Norms

We have contributed a novel and expressive language, called Cupid, for specifying and computing norms over databases of business events [8, 9]. The motivating idea is that information in any organizational database would reflect the contractual relationships that the organization is involved in. For example, information in a seller's database would reflect the states of the commitments to buyers. Today, if the seller wanted to generate reports about commitments by their state (e.g., pending, discharged, and violated), she would have to rely on a manual translation from the relevant natural language contract to highly-complicated database queries.

Cupid instead enables declaratively specifying the norms (commitments, prohibitions, authorizations, and powers) and automatically generating the required queries in SQL. The generated queries are typically an order of magnitude longer and significantly more complex than the specifications they are generated from, thereby highlighting the practical significance and value of the language. In fact, it is not unusual to obtain a thousand lines of SQL (as formatted by MySQL Workbench) for five lines of a Cupid specification.

Listing 1: Commitment specification in Cupid.

```
base events
   quote(S, B, ID, item, price, t)
   accept(S, B, ID, item, price, addr, t)
   pay(S, B, ID, item, price, amt, t)
   deliver(S, B, ID, addr, status, t)
   refund(S, B, ID, amt, rAmt, t)

commitment PurchaseCom S to B
   create quote
   detach (accept and pay) within quote + 5d
        where amt >= price
   discharge deliver within detached PurchaseCom + 10d

commitment Compensation S to B
   create quote
   detach violated PurchaseCom
   discharge refund within violated PurchaseCom + 2d
       where rAmt >= amt
```

Listing 1 gives a Cupid specification. It specifies the base events (the underlined attributes are keys and attribute *t* represents timestamp). PurchaseCom specifies a commitment from seller S to buyer B: Every quote occurrence creates a PurchaseCom commitment; the occurrence of both accept and pay within five days of quote detaches the commitment provided that the amt (in pay) is at least as high as the quoted price; and the occurrence of deliver within 10 days of detachment discharges the commitment. Compensation specifies a commitment from S to B that if a PurchaseCom commitment is violated, then a refund will be issued to B within two days of the violation.

Cupid enables interpreting operational (base or protocol) events in terms of normative events, as illustrated in Figure 3. We have built implementations of Cupid over both MySQL (a relational database) and CouchDB (a document-oriented database) [10].

Cupid enables a norm-oriented view over a single database of communication events. The problem is in a decentralized application, each agent has its own local database. The challenge is to keep these databases sufficiently in sync so that agents don't draw incompatible conclusions about the states of the norms they are involved in. For example, if a seller inferred from its database that a buyer were committed to pay, but the buyer did not infer the commitment from its own database, then the parties are misaligned, which could lead to a breakdown in interoperability. To address this challenge, we formulated the novel property of *alignment* [11] and gave a distributed algorithm that each agent runs locally to guarantee it.

3.2. Operational Protocols

Although originally we studied message ordering-based protocol specification approaches [12], we quickly abandoned them—as complicated as they were, they were unsuited to decentralization because they required the agents to move in lockstep. In a recent evaluation of protocol languages [13], we have in fact demonstrated that information protocols offer significant advantages over well-known protocol specification approaches based on traces and session types. E.g., there is no instrinsic reason why Payment by buyer and Delivery by seller cannot happen concurrenctly. However, the alternative approaches cannot express such concurrency.

Listing 2 gives the information protocol Purchase. Purchase is enacted between roles B and S and each enactment corresponds to a tuple of information as specified in the parameter line. This tuple is progressively computed as the roles exchange the messages specified, such as quote, accept, and so on. The adornments in and out capture causality by specifying information dependencies. For e.g., to send an accept, the bindings for ID and item must be known from prior interactions but addr can be bound to any value. Notice how information enables concurrency: once both buyer and seller have observed accept, the information dependencies for both pay and deliver are satisfied and they can happen concurrently.

Listing 2: An information protocol.

```
Purchase {
  role B, S
  parameter out ID, out item, out amt, out status

  S ↦ B: quote[out ID, out item, out price]
  B ↦ S: accept[in ID, in item, in price, out addr]
  B ↦ S: pay[in ID, in item, in price, out amt]
  S ↦ B: deliver[in ID, in addr, out status]
  S ↦ B: refund[in ID, in amt, out rAmt, out status]
}
```

We have introduced expressiveness-related extensions (e.g., multicast, multiple agents playing a role, and dynamic role binding) [14], and novel properties such as *atomicity* and *refinement* of protocols and their verification [15, 16].

Notice how the message names correspond to event names in Listing 1. The idea is that the base events correspond to message observations, based on which higher-level meanings such as commitments can be computed. Tosca [17] explores the layering of commitments on top of information protocols from the point of view of alignment. Clouseau [18] aims at a protocol language that unifies the operational and higher-level aspects in a single language.

3.3. Blockchain

Smart contracts are motivated as representations of real-world contracts. Their unique selling point is that they are inviolable. Despite the rhetoric, a smart contract is no substitute for a real contract [19]. A contract describes a decentralized system that is embedded in the social fabric via notions such as norms, accountability, and trust. Violability is central to the notion of a contract. A smart contract is a glorified program—a unitary machine—removed from any social relationship. In fact, the cardinal error in the philosophy of smart contracts is the notion that social structures should and can be done away with. Autonomy necessitates social structures; what we must do is represent them computationally.

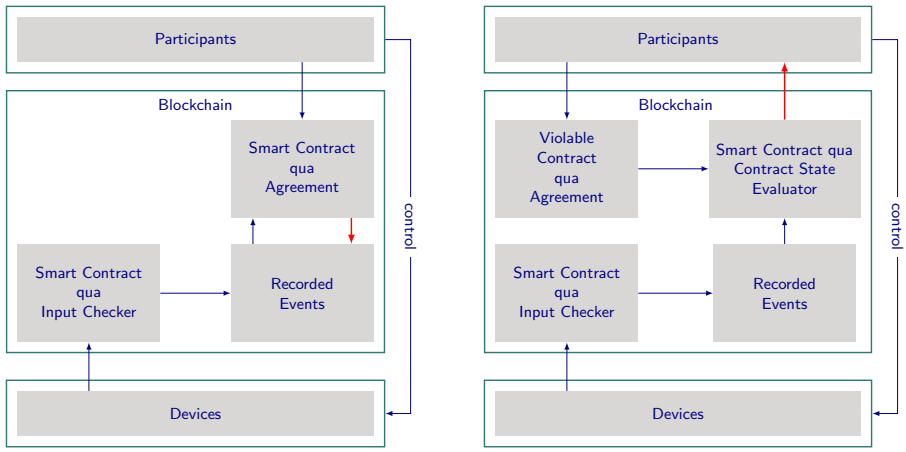
We have applied Cupid toward representing contracts on blockchain [19]. Figure 5 captures our contribution. We put violable contracts specified in a Cupid-like language (as described above) on the blockchain. As events are recorded on the blockchain, a contract state evaluator enables determining the state of the contract, e.g., whether satisfied or violated. In contrast to the traditional smart contract approach, the contract machinery does not modify the blockchain by recording new events, thus fully supporting the autonomy of the participants and at the same time supporting an immutable history of contract states.

3.4. Systems and Programming

In earlier work, we focused on methodologies for specifying interactions and engineering agents. Given that we now understand better the architectural constraints that decentralization imposes and we also have languages that work within those constraints, the ideas behind those methodologies have become crisper and evolved into a *programming model* for decentralized applications.

Specifically, the programming model enables engineering an agent given a protocol. The heart of the model is a generic protocol adapter that exposes a programming interface for sending messages and handling received messages. The interface is specifically generated for the protocol and enables a programmer to easily plug in decision making (the business logic) in the appropriate places. An agent developed following the programming model is guaranteed to communicate correctly with other agents. In recent work, we have instantiated the programming model in Java [20], Node-RED (a popular IoT programming framework) [21], and on AWS Lambda (a popular cloud-based serverless platform) [22]. We have demonstrated that our programming model saves significant programming effort and avoid errors. Most notably, our programming model relies on nothing more than UDP for communications between agents; that is, our technique for coordinating agents works over lossy, unordered infrastructure.

In recent work, we have turned our attention to systems themes. One such theme is fault tolerance, which we investigated first in the Node-RED instantiation of the programming model. There we consider the fault of *missing*



(a) Smart contracts (b) Violable contracts

Fig. 5. Smart contracts and violable contracts based on norms, schematically [19].

information, which may be due to message loss or due to the fact that an agent never sent the message. To handle such faults, agents implement application-level retransmission policies in the agents. We contrasted our approach with the traditional IoT approach, which is MQTT over TCP (in contrast to UDP, TCP offers reliable ordered delivery). We found that for different loss rates, our application-level retransmission over UDP strategy performed favorably compared to MQTT over TCP in terms of throughput measured as the numbers of interactions completed. Mandrake [23] goes further in proposing a policy language that takes advantage of information protocols and shows how to implement fault tolerance policies at the application level. Bungie [24] gives succinct annotations for augmenting information protocols with features that support fault tolerance.

In the instantiation of our programming model on AWS Lambda, the agents are highly modular, each module being a *function* (as in Function as a Service). Again, the modularity naturally follows from the programming model being protocol based. Preliminary experiments on AWS Lambda demonstrated that several AWS functions in fact had several concurrent instances running at the same time, thus establishing the potential scalability of our design.

4. Directions

We highlight some important directions for research.

4.1. Contract-Based Computing

A (business) contract enables engagements between autonomous principals by declaratively laying out the *norms* (e.g., commitments, prohibitions, authorizations, powers, and so on) that capture expectations about the behavior of the principals. Now the beauty of a contract is that it supports both autonomy and correctness. That is, a principal may do decide to act as it pleases, however, if it violates a norm, that would amount to incorrect behavior. Further, a violation would in principle be *verifiable* from the observation of the relevant events. Given the centrality of contracts to engagements between autonomous principals, one would imagine that the fundamental software representation of a system that supports such engagements would be in terms of business contracts. But where currently do we represent contracts when we build software?

My vision is that a declarative contract (based on norms, and therefore violable) would yield everything needed for interoperation between the principals who sign up to it. Naturally, the engineering of a particular principal's agent would be facilitated by the implementation of programming models based on contracts. Our ongoing work on protocol-based programming models and fault tolerance and efficiency would be subsumed in the contract-based model. For example, the contract would tell an agent which messages are crucial and therefore worth retransmit-

ting in the absence of an expected event. Contract-based computing stands in stark contrast with what passes for interoperation today—based on either APIs or message formats (which are prevalent in finance and health).

We have recently elaborated upon this vision, detailing how the aforementioned contributions already constitute a foundation for realizing it and how it represents an alternative to smart contracts [19]. We have also developed a proof-of-concept implementation of violable contracts for the R3 Corda distributed ledger. Our contributions in this line of work have attracted attention from industry. However, more needs to be done in terms of languages needed to express real business contracts and methodology, including design and verification tools, and programming and deployment tools. In particular, blockchain is just one platform for carrying out contract-based computing. A more appealing alternative is to support contract-based computing in the cloud, e.g., on serverless and container-based platforms.

4.2. Microservices

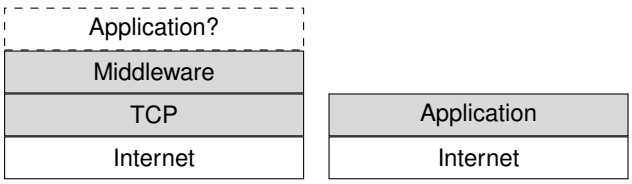
In the last few years, microservices have emerged as a dominant paradigm for building loosely-coupled distributed applications. What is interesting about microservices is that each microservice is treated as if it were autonomous—developed, deployed, and scaled independently of other microservices in the application. Microservices are here to stay, since interest in them in underpinned by advances in cloud computing, especially containerization.

Building a loosely-coupled system requires explicitly modeling the extent of the coupling between system components. In other words, it requires modeling the interactions between components. Surprisingly though, current microservice approaches are not based upon a model of interactions. This leads to overly complex microservices sitting atop overly complex and specialized communication infrastructures (services meshes, event buses, and so on). The overall result is—contrary to aims—a tightly-coupled system.

We want to apply protocols toward realizing an application as a truly loosely-coupled system of microservices. A protocol would capture explicitly the extent of the coupling between the microservices in an application. Further, since a protocol may be enacted asynchronously—without ordering guarantees from the infrastructure—no specialized communication infrastructure is required.

4.3. Programming Models for Distributed Applications

Focusing on meaning represents a radical departure from conventional systems wisdom, which emphasizes doing ordering and fault tolerance in communication services. E.g., TCP (the de facto service for Internet applications) and message queues (the holy cow of business messaging) 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, as the end-to-end argument [25] anticipated, in providing ordering guarantees, communication services limit decision making at the application level and that programming based on meaning ("semantics") was the way out [26]. However, a meaning-based programming model has remained deeply elusive. As Internet pioneer David Clark [27] makes clears (after noting Ken Birman's remark that TCP wasn't good for anything): "*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.*" Even Shannon appeared to be aware of the importance of communication meaning [28]. Meaning is the forgotten holy grail of systems research.



(a) Current network stack. (b) IOSE network stack.

Not focusing on meaning has given rise to today's complex system stack (Figure 6a) where every layer in the stack except the application is modeled as a protocol. In modeling applications in terms of meaning-based protocols, IOSE is unique among existing software development methodologies in harboring the potential to obviate layers of complexity in the stack and making it possible to develop Internet-native applications, that is, applications that sit directly on top of the Internet (possibly through a shim such as UDP) (Figure 6b). The stack of Figure 6a is threatened by many developments, including data centers and the IoT, but few have dared to imagine a stack as simple as Figure 6b.

Current programming models, which bake in assumptions of ordering and fault tolerance, are entrenched. Programmers have gotten used to all kinds of complex middleware and the client-server mindset. Changing this mindset will require showing that protocol-based application programming models make it possible to build flexible, fault-tolerant, and loosely-coupled decentralized applications just as conveniently as existing programming models. Further, the implemented applications should offer comparable performance and opportunities for scalability. Although we have started investigating these directions, much more needs to done to realize this vision.

We are currently exploring several ideas for making protocol adapters more efficient. In our AWS Lambda implementation, scalability is limited by the fact that the protocol adapter may have only one instance (because it must process messages serially). Under reasonable assumptions about identifiers though, we can in fact run several instances of the adapter. We are currently exploring how protocols may support redundancy for fault tolerance and how redundancy trades off with efficiency. We are exploring code generation-supported programming models that make it easy for a microservice developer to specify fault-tolerance policies.

More broadly, we are examining how important ideas in distributed computing map to IOP. One of them is the notion of *potential causality* [29], along with the family of timestamp-based coordination approaches it has inspired. We believe that the information protocols approach, being based on a specification of causality, and therefore *true* causality, is a superior alternative to potential causality. In fact, work on causality in distributed systems has historically been concerned with detecting causal relationships when really the correct approach—as information protocols make clear—is to specify the relationships. We tend to forget that computing is a *science of the artificial*.

5. Concluding Remarks

IOSE is informed by practical requirements. The requirements inform the development of methodologies and languages, which in turn inform programming models and systems work. Recently, as described above, we have focused on supporting IOSE on popular platforms such as blockchain and serverless, with an eye on making an impact on practice.

Recent discussions and consulting engagements with industry highlight the exciting potential of IOSE. In the short term, our focus is on developing software and engaging with industrial partners to seek out joint work with them that applies IOSE toward addressing their problems. For example, colleagues at IBM Research are interested in developing a version of our decentralized serverless technology [22] targeted at Kubernetes, the motivation being to simplify the development of Kubernetes applications.

Alan Kay thought the big idea (in OOP) was messaging; but also that it wasn't done properly—in fact, according to us, it wasn't done at all. The bigger idea is protocols and we aim to do it properly. Via novel abstractions that combine decision and communication via information, we seek to revolutionize programming practice and research. Information-based representations of interaction, decision abstractions based on them, and native language support for them will become central themes in programming.

Industry is waking up to the advantages of messaging-based coordination, as evidenced by growing interest in actor model implementations such as Akka and languages such as Erlang. However, without embracing protocols, messaging-based coordination will remain just as complex as shared-memory coordination [30]. As applications are devised increasingly in terms of microservices, the problem of coordination will become ever more acute and protocols will come to the fore. The problem of representing contracts has also entered the broader imagination thanks to blockchain. Coupled with the fact that we are focused on producing practical technology, these developments in industry give us hope that the novel research program we have pursued for the past two decades is poised for impact.

Today, as many in the MAS community reflect on what the community has contributed to computing and software development over the last three decades and how to stay relevant, we are confident that the MAS work on communication meaning is significant and has the potential to reshape computing and software engineering.

Acknowledgments. Munindar Singh laid the foundations of IOSE is his seminal work on the meaning of communication. He and his research group at NC State University continue to elaborate on the theme. Pınar Yolum, in particular, made substantial contributions via her work on commitment protocols. Today, for the most part, IOSE is a collaboration between Singh's group and the author's group at Lancaster. Samuel Christie, who is a member of both groups, currently leads the programming model efforts.

Protocols, commitments, and norms are historically important research themes in multiagent systems. The current paper is not a survey of the field; it is merely a summary of the main ideas and implications of IOSE. The author has benefitted greatly from interaction with members of the EMAS (software engineering) and the COIN (coordination) communities. Discussions (and in some cases joint work) with Matteo Baldoni, Cristina Baroglio, Fabiano Dalpiaz, Angelo Ferrando, Viviana Mascardi, John Mylopoulos, and Michael Winikoff have been especially illuminating. Matthew Arrott, an expert in business messaging and financial applications, has been a rich source of support and encouragement for the past decade.

References

[1] A.K. Chopra and M.P. Singh, From Social Machines to Social Protocols: Software Engineering Foundations for Sociotechnical Systems, in: *Proceedings of the 25th International World Wide Web Conference*, ACM, Montréal, 2016, pp. 903–914.

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

[3] G.H. von Wright, *Norm and Action: A Logical Inquiry*, Routledge & Kegan Paul, London, 1963.

[4] A.J.I. Jones and M.J. Sergot, On the Characterisation of Law and Computer Systems: The Normative Systems Perspective, in: *Deontic Logic in Computer Science: Normative System Specification*, J.-J.C. Meyer and R.J. Wieringa, eds, John Wiley and Sons, Chichester, UK, 1993, pp. 275–307, Chapter 12.

[5] M.P. Singh, Information-Driven Interaction-Oriented Programming: BSPL, the Blindingly Simple Protocol Language, in: *Proceedings of the 10th International Conference on Autonomous Agents and MultiAgent Systems*, 2011, pp. 491–498.

[6] J.J. Odell, H.V.D. Parunak and B. Bauer, Representing Agent Interaction Protocols in UML, in: *Agent-Oriented Software Engineering*, Lecture Notes in Computer Science, Vol. 1957, Springer, 2001, pp. 201–218.

[7] WS-CDL, Web Services Choreography Description Language Version 1.0, 2005, www.w3.org/TR/ws-cdl-10/.

[8] A.K. Chopra and M.P. Singh, Cupid: Commitments in Relational Algebra, in: *Proceedings of the 29th AAAI Conference on Artificial Intelligence*, 2015, pp. 2052–2059.

[9] A.K. Chopra and M.P. Singh, Custard: Computing Norm States over Information Stores, in: *Proceedings of the 15th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, IFAAMAS, Singapore, 2016, pp. 1096–1105.

[10] S.H. Christie V, A.K. Chopra and M.P. Singh, Hercule: Representing and Reasoning about Norms as a Foundation for Declarative Contracts over Blockchain, *IEEE Internet Computing (IC)* **25**(4) (2021), 67–75. doi:10.1109/MIC.2021.3080982.

[11] A.K. Chopra and M.P. Singh, Generalized Commitment Alignment, in: *Proceedings of the 14th International Conference on Autonomous Agents and Multiagent Systems*, IFAAMAS, 2015, pp. 453–461.

[12] M. Baldoni, C. Baroglio, A.K. Chopra, N. Desai, V. Patti and M.P. Singh, 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, 2009, pp. 843–850.

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

[14] A.K. Chopra, S.H. Christie V and M.P. Singh, Splee: A Declarative Information-Based Language for Multiagent Interaction Protocols, in: *Proceedings of the 16th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, IFAAMAS, São Paolo, 2016, pp. 1054–1063.

[15] S.H. Christie V, A.K. Chopra and M.P. Singh, Compositional Correctness for Multiagent Interactions, in: *Proceedings of the 17th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, IFAAMAS, São Paolo, 2018, pp. 1159–1167.

[16] S.H. Christie V, A.K. Chopra and M.P. Singh, Multiagent protocol refinement, in: *Proceedings of the 19th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, IFAAMAS, 2020, 258–266.

[17] T.C. King, A. Günay, A.K. Chopra and M.P. Singh, Tosca: Operationalizing Commitments Over Information Protocols, in: *Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI)*, 2017, pp. 256–264.

[18] M.P. Singh and A.K. Chopra, Clouseau: Generating Communication Protocols from Commitments, in: *Proceedings of the 34th AAAI Conference on Artificial Intelligence*, AAAI Press, New York, 2020, pp. 7244–7252.

[19] M.P. Singh and A.K. Chopra, Computational Governance and Violable Contracts for Blockchain Applications, *IEEE Computer* (2020), 53–62.

[20] A. Günay and A.K. Chopra, Stellar: A Programming Model for Developing Protocol-Compliant Agents, in: *Preproceedings of the 6th International Workshop on Engineering Multi-Agent Systems*, LNCS, Vol. 11375, Springer, Stockholm, 2018, pp. 117–136.
[21] S.H. Christie V, D. Smirnova, A.K. Chopra and M.P. Singh, Protocols over Things: A Decentralized Programming Model for the Internet of Things, *IEEE Computer* **53**(12) (2020), 60–68.
[22] S.H. Christie V, A.K. Chopra and M.P. Singh, Deserv: Decentralized Serverless Computing, in: *Proceedings of the 19th IEEE International Conference on Web Services (ICWS)*, IEEE Computer Society, Virtual, 2021, pp. 51–60. doi:10.1109/ICWS53863.2021.00020.
[23] S.H. Christie V, A.K. Chopra and M.P. Singh, Mandrake: Multiagent Systems as a Basis for Programming Fault-Tolerant Decentralized Applications, *Journal of Autonomous Agents and Multi-Agent Systems (JAAMAS)* **36**(1) (2022), 1–30. doi:10.1007/s10458-021-09540-8.
[24] S.H. Christie V, A.K. Chopra and M.P. Singh, Bungie: Improving Fault Tolerance via Extensible Application-Level Protocols, *IEEE Computer* **54**(5) (2021), 44–53. doi:10.1109/MC.2021.3052147.
[25] J.H. Saltzer, D.P. Reed and D.D. Clark, End-To-End Arguments in System Design, *ACM Transactions on Computer Systems* **2**(4) (1984), 277–288. doi:10.1145/357401.357402.
[26] A.K. Chopra, S.H. Christie V and M.P. Singh, Multiagent Foundations for Distributed Systems: A Vision, in: *Proceedings of the 9th International Workshop on Engineering Multi-Agent Systems (EMAS 2021)*, Lecture Notes in Artificial Intelligence, Springer, London, 2022, pp. 62–71. doi:10.1007/978-3-030-97457-2_4.
[27] D. Clark, The Network and the OS, in: *SOSP History Day 2015*, ACM, 2015.
[28] C.E. Shannon, A mathematical theory of communication, *The Bell System Technical Journal* **27**(3) (1948), 379–423.
[29] L. Lamport, Time, Clocks, and the Ordering of Events in a Distributed System, *Communications of the ACM (CACM)* **21**(7) (1978), 558–565. doi:10.1145/359545.359563.
[30] T. Tu, X. Liu, L. Song and Y. Zhang, Understanding Real-World Concurrency Bugs in Go, in: *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, ACM, 2019, pp. 865–878.