Interaction-Oriented Programming: An Application Semantics Approach for Engineering Decentralized Applications

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

Samuel H. Christie V
Lancaster University
Lancaster, UK
samuel.christie@lancaster.ac.uk

Munindar P. Singh
North Carolina State University
Raleigh, USA
singh@ncsu.edu

ABSTRACT

Interaction-Oriented Programming (IOP) refers to multiagent concepts, languages, and programming models for engineering applications that are characterized by interactions between *autonomous* parties. Such applications arise in domains such as e-commerce, health care, and finance. Owing to the autonomy of the principals involved, such applications are conceptually *decentralized*.

We demonstrate how to specify a decentralized application flexibly and how to engineer correct, fault-tolerant endpoints (agents) for the principals in a straightforward manner. Notably, the entire application is realized as agents communicating over an unordered, unreliable messaging infrastructure (our implementations in fact use UDP). IOP departs from traditional distributed systems approaches that rely on guarantees in the application's communication infrastructure, e.g., for ordering and fault tolerance. Notably, IOP shows how to address *application semantics*, the holy grail of distributed systems.

CCS CONCEPTS

• **Computing methodologies → Multi-agent systems**.

KEYWORDS

Commitments, information protocol, programming model

ACM Reference Format:
Amit K. Chopra, Samuel H. Christie V, and Munindar P. Singh. 2021. Interaction-Oriented Programming: An Application Semantics Approach for Engineering Decentralized Applications. In *Proceedings of the 2021 ACM Symposium on Principles of DistributedComputing (PODC '21), July 26–30, 2021, Virtual Event, Italy*. ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3465084.3467486

1 APPLICATION SEMANTICS: THE HOLY GRAIL

Despite decades of research, programming robust distributed applications remains incredibly difficult. The reason for this state of affairs is quite remarkable: Research and practice in distributed systems continues to focus on guarantees in the application's communication infrastructure (e.g., for ordering and fault tolerance) in contravention of their own fundamental principles, especially the end-to-end argument [8]. Evidence the notion of a service mesh, a middleware for microservice-based applications that transparently provides fault tolerance and other functionality, and is, in essence, trying to prop up the conceptually flawed RPC paradigm for building distributed applications. Evidence QUIC [7], an alternative to TCP, which in trying to address problems arising from TCP's FIFO delivery model, ends up introducing a more complicated delivery model consisting of multiple FIFO streams.

This continued focus on infrastructure mechanisms stands in stark contrast to the scant attention paid to *application semantics* [1], which is the central challenge motivated by the end-to-end argument. Broadly, the challenge recognizes that application users base their decision making on the meaning (the content) of their communications and, therefore, incorporating meaning is the key to getting rid of complicated infrastructure solutions. For example, in an e-commerce application, what may be meaningful to a seller for purposes of fulfilling a buyer's purchase-order (PO) is an identifier for the PO and the correlated item, price, and address. The seller cannot fulfill a PO lacking such information, and once it has this information, the seller can fulfill the PO whenever it wants—independently of the order in which it observes communications. Meaning, grounded in information, thus obviates the need for ordered delivery infrastructures. However, incorporating meaning requires representing it and therein lies the crux of the challenge: How can we model distributed applications in a way that captures the meaning of communications?

More concretely, the quest is for a general method (including application models and programming abstractions) that enables and facilitates implementing robust distributed applications directly over the Internet (or UDP, which is merely a thin shim over IP), which guarantees neither ordered nor guaranteed delivery. This means that all concerns stemming from ordering, correlation, and faults must now be handled at the application level. This quest is really the *holy grail* of distributed systems. As David Clark [6] says, UDP (not TCP) is designed to support application semantics; however, with the current state of programming abstractions, building applications over UDP is impractical.

2 APPLICATION SEMANTICS IN MULTIAGENT SYSTEMS

Independently, to address issues about communication semantics in multiagent systems, Singh [9] emphasized modeling the social meaning of communications via abstractions such as commitments between agents. Developing this theme further, Yolum and Singh [11] advanced the idea that commitments support more flexible interactions than protocols that specified rigid message sequences.

PODC '21, July 26–30, 2021, Virtual Event,Italy

ACM ISBN 978-1-4503-8548-0/21/07…$15.00
https://doi.org/10.1145/3465084.3467486

To understand their contribution, consider Figure 1, which shows a simple e-commerce protocol specified as a state machine (S and B refer to the seller and the buyer, respectively). Suppose S, for whatever reason, wanted to send *deliver* before receiving *pay*. Such a move would violate the protocol.

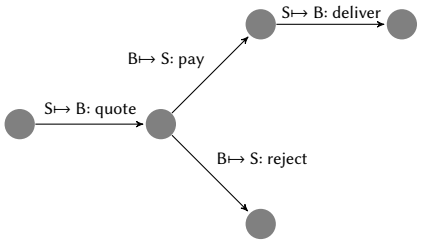

Figure 1: A protocol specified as an FSM.

To overcome this inflexibility, one could simply declaratively specify the meaning of the interaction between S and B via a commitment, e.g., that *quote* creates a commitment from S to B that if *pay* occurs, then *deliver* will occur. Clearly, S not sending *deliver* if it had sent *quote* and received *pay* would be a violation of the commitment; however, the commitment itself does not prescribe any relative ordering of events. For example, S may send *deliver* before receiving *pay*, even though it doesn't have to, and B may send *pay* after receiving *deliver*, even though it doesn't have to. Listing 1 shows how something like the foregoing commitment would be expressed in Cupid [4], a language for specifying and interpreting commitments—via queries—over information in a relational database. For example, a Cupid-generated SQL query, given a database of base events, yields all the *violated* instances of PurchaseCom.

Listing 1: A commitment specification in Cupid.

```
base events
  quote(S, B, ID key, item, price, timestamp)
  reject(S, B, ID key, item, price, timestamp)
  pay(S, B, ID key, item, price, addr, timestamp)
  deliver(S, B, ID key, item, price, addr, timestamp)

commitment PurchaseCom S to B
  create quote
  detach pay within quote + 5 days
  discharge deliver within detached PurchaseCom + 10 days
```

A multiagent system, however, is decentralized; there is no central database of events. Each agent has its own database of interaction state (*local state*), over which it interprets the relevant commitments [3]. How are these local states being populated? How are the atoms of meaning (the base events) determined?

To address these questions, Singh invented the idea of declarative information protocols [10]. Unlike the protocol shown in Figure 1, an information protocol specifies the computation of an information object via explicit causality (to capture information dependencies) and integrity constraints on messages. An agent can send a message in any local state that satisfies the constraints. Message reception though is unconstrained; that is, a message can be received no matter what the receiver's local state. Listing 2 illustrates how messages are specified in an information protocol. For example, for S to send a *reject*, it must already know the ID and the correlated item and price from prior communication (all adorned **in**), but it can generate any values for dec and fin if it does not already know them (both adorned **out**).

Listing 2: An information protocol.

```
S ↦ B: quote[out ID key, out item, out price]
B ↦ S: reject[in ID key, in item, in price, out dec, out
    fin]
B ↦ S: pay[in ID key, in item, in price, out addr, out dec]
S ↦ B: deliver[in ID key, in item, in price, in addr, out
    fin]
```

Information protocols offer fundamental advantages over alternative protocol representations [2]. We have proposed a programming model for implementing fault-tolerant multiagent systems based on information protocols [5]. Its centerpiece is a generic protocol adapter that sits within each agent and (1) enforces protocol constraints and therefore ensures that the agent communicates correctly, (2) provides a local state-based (information-based) API that abstracts away actual network interfaces and facilitates the development of agent; and (3) supports fault tolerance by retransmitting and forwarding information already in the agent's local state. This programming model can be used to develop distributed applications as multiagent systems directly over UDP, thus paving the way to addressing the challenge of application semantics.

3 TUTORIAL

The tutorial elaborates on the foregoing themes. It introduces desiderata for decentralized applications and the idea of application semantics. It covers Cupid and information protocols in depth. The tutorial demonstrates a programming model that enables realizing decentralized applications as multiagent systems in a manner compatible with the idea of application semantics.

Acknowledgments. Support from the EPSRC (Grant EP/N027965/1) and the NSF (grant IIS-1908374) is gratefully acknowledged.

REFERENCES

[1] 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.
[2] 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 (JAIR)* 69 (Dec. 2020), 1351–1393.
[3] Amit K. Chopra and Munindar P. Singh. 2009. Multiagent commitment alignment. In *Proceedings of the Eighth International Conference on Autonomous Agents and Multiagent Systems*. IFAAMAS, 937–944.
[4] Amit K. Chopra and Munindar P. Singh. 2015. Cupid: Commitments in Relational Algebra. In *Proc. 29th AAAI Conference on Artificial Intelligence*. 2052–2059.
[5] Samuel H. Christie V, Daria Smirnova, Amit K. Chopra, and Munindar P. Singh. 2020. Protocols Over Things: A Decentralized Programming Model for the Internet of Things. *IEEE Computer* 53, 12 (Dec. 2020), 60–68.
[6] David Clark. 2015. The Network and the OS. In *SOSP History Day 2015* (Monterey, California). ACM, Article 11, 19 pages.
[7] Jana Iyengar and Martin Thomson. 2021. QUIC: A UDP-Based Multiplexed and Secure Transport. https://datatracker.ietf.org/doc/html/draft-ietf-quic-transport-34.
[8] 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.
[9] Munindar P. Singh. 1998. Agent Communication Languages: Rethinking the Principles. *IEEE Computer* 31, 12 (Dec. 1998), 40–47.
[10] 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*. 491–498.
[11] 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*. ACM Press, 527–534.