Programming Decentralized Decision Making in Business Processes

Amit K. Chopra, [1] **Samuel H. Christie V,** [2] **Munindar P. Singh** [2]

[1] Lancaster University
[2] North Carolina State University
amit.chopra@lancaster.ac.uk, schrist@ncsu.edu, singh@ncsu.edu

Abstract

In a business process, a principal's communications represents its public decisions. This insight leads us to conceptualize a business process as a decentralized multiagent system, wherein each principal is represented by an agent. Current programming models, however, offer poor decision making-oriented abstractions for implementing agents. In particular, they lack abstractions that bridge an agent's internal (and private) decision logic with its public decisions.

We present *Kiko*, a programming model for business processes that combines decision making and communications. *Kiko*'s main abstraction is that of a *decision maker*. To implement an agent, a programmer writes one or more decision makers, each of which chooses from among a set of valid decisions (possible messages it could send) and makes some mutually compatible decisions (by sending the corresponding messages). *Kiko*'s abstractions enables developing sophisticated, practical agents and moreover enables agent developers to focus on the business logic—more so than any alternative programming model.

Introduction

Driven by technologies such as the cloud and the IoT, there is growing interest in realizing business processes as systems of *autonomous* components that interact via arms-length communications. We refer to the components as *agents* and to the business processes as being *decentralized*. The industry-led microservices paradigm exemplifies this trend.

Today we lack programming abstractions that enable systematically engineering decentralized business processes. In particular, we lack abstractions that accommodate autonomy, which refers to the ability of agents to make decentralized decisions flexibly—as principals (humans, organizations) would make them. *How can we systematically realize business processes as decentralized 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 centralized and synchronous abstractions (e.g., state machines, shared memory, and event ordering) that are incompatible with autonomy.

Approach: Business Meaning as a Basis for Decision Making

An *interaction protocol* models a decentralized application by specifying the coordination constraints on agents. Supporting flexible decision making requires protocols that support *business meaning*, which is what principals 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.

Meaning supports flexible decision making. E.g., the seller may violate the commitment or do Delivery before it receives 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.

Meaning lets us avoid irrelevant operational restrictions, including 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.

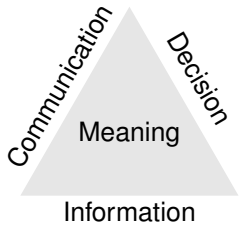

Figure 1: Meaning Intuition.

Fig 1 captures our key intuition: Meaning supports decision making by bringing together information and communication. In particular, an agent's communications do not only convey information, but they also represents its (public) decisions. The challenge for a programming model is to provide first-class decision making abstractions that constitute a bridge from its internal decision logic to its public decisions.

We adopt the idea of declarative information protocols (Singh 2011), a novel approach for specifying protocols in terms of the information content of messages and causality and integrity constraints on them (as described above). Each information protocol specifies a business process. An agent may adopt a role in several business processes at once, e.g., it may play buyer and patient in ebusiness and healthcare, respectively. The overarching challenge we address is to design and implement a programming model that enables implementing agents given the roles they adopt. Viewing agents as decision makers, the task is to come up with natural decision-making abstractions (including for handling faults) in light of the information-based communication constraints.

Contributions

We present an information protocol-based programming model that enables systematically and effortlessly engineering fault-tolerant, loosely-coupled, and flexible decentralized business processes.

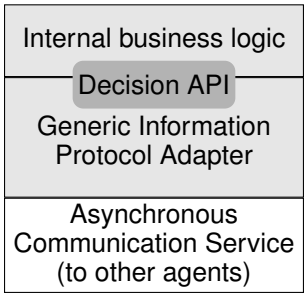

Figure 2: *Kiko* agent (in gray).

The programming model's core component is a generic *adapter* for information protocols that sits within each agent. The adapter abstracts away the underlying communication service and provides a decision API that enables programmers to plug in the internal logic for making decisions (Fig 2). Specifically, the API enables programmers to select some of the *available* decisions (which are tracked by the adapter based on the agent's history and the roles it plays) and flesh them out. The adapter emits the messages corresponding to the chosen decisions thus recording them in the history as made. E.g., Accept or Reject decisions for some Offers may be available to the buyer; based on internal logic (e.g., some optimization), buyer may Accept some, Reject some, and leave some open. The adapter ensures integrity by rejecting decisions that would make the agent's history inconsistent (e.g., buyer doing both Accept and Reject for an Offer or doing Accept for an Offer rescinded by seller). The adapter can use any communication service, including unordered, unreliable ones such as UDP (part of the TCP/IP suite). The adapter obviates the need for message queues (AMQP 2007), a holy cow in business messaging.

Kiko guarantees an agent's compliance with the roles its plays and supports a variety of practical and interesting decision making patterns.

Correlation. An agent may simultaneously be involved in several enactments of a protocol. *Kiko* automatically correlates communications by enactment. For example, buyer may be concurrently engaged with seller in several distinct enactments, each for some item at some price.

Cross-enactment reasoning. *Kiko* enables agents to use information across enactments in their decision making. For example, *Kiko* enables expressing the decision where the buyer accepts the offer with the lowest price from among all outstanding offers.

Emission sets. As a variant of the above, buyer wants to accept a set of offers that maximizes the number of items it gets given some budget and reject all others. *Kiko* enables expressing the emission of multiple messages to different agents as a single decision step.

Multiple protocols. *Kiko* enables implementing agents that concurrently play roles in conceptually unrelated protocols. For example, buyer may be purchasing goods on behalf of an organization; if so, it must get approval from another agent in the organization before accepting any offer.

Notably, in providing a decision-based interface for programming agents, the programming model abstracts away the interface with the communication service that transports messages between agents. In particular, decision making is oblivious to the order in which messages are received by an agent, making it possible to use an unordered asynchronous communication service such as UDP for messaging. Actual message emission is also handled transparently in the programming model. The common refrain one often hears (Little 2017; Smith 2021) is that programming models should support programmers in focusing on the business logic; *Kiko* take this support to new heights.

Our programming model offers an alternative to conventional business processes wisdom, which emphasizes a rigid ordering of steps and relies on complex communication services. Although the idea of business protocols is considered important in traditional work on business processes (e.g., in the notion of choreography (Peltz 2003)), existing approaches specify a protocol in terms of global message orders, which limits their value as abstractions that enable intelligent decision making (Chopra, Christie V, and Singh 2020). In adopting meaning grounded in information as the basis for structuring and implementing decentralized business processes, we depart fundamentally from current business process approaches based on protocols.

Paper Details

The programming model is fully implemented and the paper that describes it is currently under review at a major conference.

References

AMQP. 2007. Advanced Message Queuing Protocol. Http://www.nsf.gov/funding/.

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

Little, M. 2017. Virtual Panel: Microservices in Practice. https://www.infoq.com/articles/microservices-in-practice/.

Peltz, C. 2003. Web Service Orchestration and Choreography. *IEEE Computer*, 36(10): 46–52.

Singh, M. P. 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.

Smith, B. 2021. Getting started with serverless for developers: Part 2 - The business logic. https://aws.amazon.com/blogs/compute/getting-started-with-serverless-for-developers-part-2-the-business-logic/.