Kiko: Programming Agents to Enact Interaction Protocols

Samuel Christie, Munindar P. Singh (singh@ncsu.edu),
Amit K. Chopra (amit.chopra@lancaster.ac.uk)

Aim

Business processes involve interactions between autonomous principals. Interactions have traditionally been viewed in terms of message ordering. Interactions, are, however, about decentralized decision making: A principal's communications represent its (public) decisions, which it makes on the basis of some (private) internal logic. The challenge Kiko addresses is to enable the programming of a business process on the basis of decision making abstractions that combine internal logic and communications.

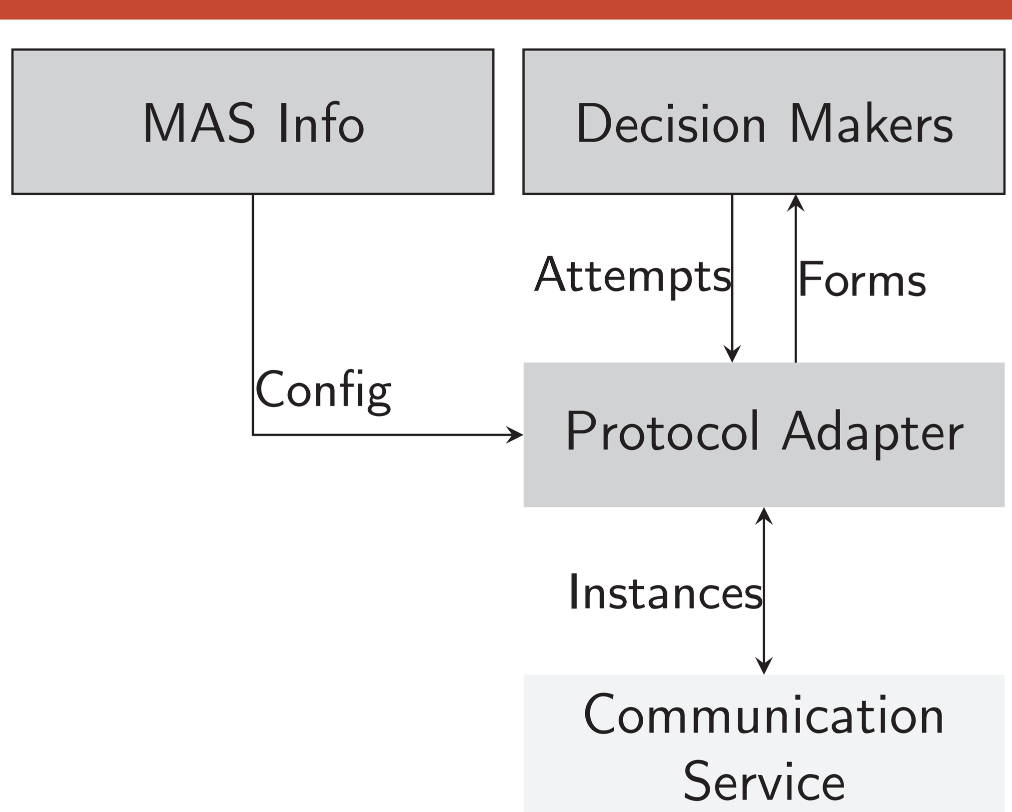
Interaction Protocols

```
Purchase
  roles (B)uyer, (S)eller
  parameters out ID key, out item, out
       price, out done

  B -> S: RFQ[out ID key, out item]
  S -> B: Quote[in ID key, in item, out
       price]
  B -> S: Buy[in ID key, in item, in
       price, out done]
  B -> S: Reject[in ID key, in price, out
       done]
```

A business process is modeled as a multiagent system (MAS): An interaction protocol specifies information dependencies that constrain decentralized decision making by agents. E.g., to send a *Quote* in some enactment, the Buyer must know the bindings of the enactment identifier ID (annotated key) and the item (both adorned ⌜in⌝); however, it can apply its internal logic to generate a binding for price (adorned ⌜out⌝). Further, any enactment may have at most one binding for a parameter, thus supporting integrity.

Programming An Agent



Write decision makers that are invoked by agent's protocol adapter. The adapter supplies a decision maker with the possible decisions (*forms*) that could be made (filled) given the interaction state. The programmer writes internal logic to fill out some forms. These forms are known as *attempts*, which if validated by the adapter for consistency, are emitted as (message) *instances* and recorded as decisions.

Agent Configuration

Each agent is configured with information about the MAS it plays roles in.

```
self = "Bob"
systems = {
 "5feceb66": {
    "protocol": Purchase,
    "roles": {Buyer: self, Seller:
         "Sally"}}}
agents = {
  self: [("192.168.1.100", 1111)]
  "Sally": [("192.168.1.102", 1111),
       ("152.1.27.202", 1111)]}
```

Initiating Enactments

```
@adapter.decision(event=InitEvent)
def start(forms):
    for item in ["ball", "bat"]:
        ID = str(uuid.uuid4())
        for m in forms.messages(RFQ):
            m.bind(ID=ID, item=item)
```

Invoked upon Bob's initialization and leads to the emission of two *RFQ*s.

Exercising Choice

```
@adapter.decision
def start(forms):
  for m in forms.messages(Buy):
    if(m["price"] < 20)
        m.bind(done="cool")
    else reject =
         next(forms.messages(Reject,
         ID=m["ID"]))
         reject.bind(done="rejected")
```

Contradictions Blocked

A set of attempts is rejected by the adapter if it contains mutually-exclusive messages.

```
@adapter.decision
def indecisive(forms):
  buy = next(forms.messages(Buy))
  reject = next(forms.messages(Reject,
       system=buy.system, ID=buy["ID"]))
  buy.bind(done="accepted")
  reject.bind(done="rejected")
```

Simple Selection

```
@adapter.decision
def cheapest(forms):
  buys = forms.messages(Buy)
  cheapest = min(buys, key=lambda b:
       b["price"])
  cheapest.bind(done=True)
```

Optimization

Maximize number of Buys given budget; reject the other Quotes.

```
@adapter.decision
def select_gifts(forms):
    best, rest = best_combo(forms)
    for b in best: # buy the best items
        b.bind(done=True)
    for r in rest: # reject the rest
        r.bind(done=True)
```

Multiprotocol Logic

```
Approval
  roles (R)equester, (A)pprover

  R -> A: Ask[out aID key, out request]
  A -> R: Approve[in aID, in request, out
       approved]
```

*Ask*ing (approval) for each *Buy*, copying *Buy*'s payload into request.

```
@adapter.enabled(Buy)
def request_approval(buy):
    ask = next(adapter.enabled_messages.
         messages(Ask), None)
    return ask.bind(ID=str(uuid.uuid4()),
         request=buy.payload)
```

Intuitive Semantics

$$\textsc{Recv}\ \frac{\widehat{m} \in I_a \qquad \widehat{m} \notin H_a \qquad \mathsf{check}_r(\widehat{m}, H_a)}{a\langle H_a, I_a, O_a\rangle \longrightarrow a\langle H_a \cup \{\widehat{m}\}, I_a, O_a\rangle}$$

For a message in the agent's inbox, if it does not already belong to its history and passes validty checking, add it to the history.

$$\textsc{Decide}\ \frac{\begin{array}{c} Q := \mathsf{forms}(a, H_a) \\ T := d(Q) \qquad \mathsf{check}_s(T, H_a) \end{array}}{a\langle H_a, I_a, O_a\rangle \longrightarrow a\langle H_a \cup T, I_a, O_a \cup T\rangle}$$

Let decision maker (d) compute attempt set T given forms Q. Then if T passes validity checking, add it to the agent's history.

Noteworthy Benefits

- Enables programming an agent as a set of decision makers
- Empowers programmers by abstracting away communication services and enabling them to focus on the business logic
  - Neither message emission or reception are programming primitives
- Works over any asynchronous communication service, even unordered ones
- Supports complex decision-making patterns involving multiple communications in multiple business process instances

Directions

- Enable decision making based on business meaning, e.g., contracts
- Support fault tolerance, security, and other concerns today handled in business meaning-agnostic middleware
- Support IoT devices

Acknowledgments

Christie and Chopra were supported by the EPSRC grant EP/N027965/1 and Singh was partially supported by the NCSU Laboratory of Analytic Sciences.

Play: https://gitlab.com/masr/bspl/-/tree/kiko

Read: In the AAMAS 2023 Proceedings