



Engineering Multiagent Systems



Munindar P. Singh² Amit K. Chopra¹ Samuel H. Christie V²
(with contributions from Matteo Baldoni³ and Omkar Joshi²)

¹Lancaster University

²North Carolina State University

³University of Torino

AgntCon 2026

Motivation

Motivating Interaction Protocols

Commitments and Other Norms

Specifying and Verifying Protocols

LLM-Based Agents for Protocols

Conclusion

Outline

Motivation

Motivating Interaction Protocols

Commitments and Other Norms

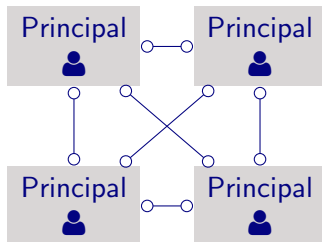
Specifying and Verifying Protocols

LLM-Based Agents for Protocols

Conclusion

Before Computing: Decentralization was Natural

Long-lived engagements between autonomous principals

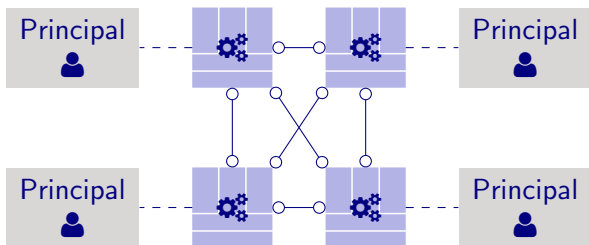


- ▶ In business, health, finance, ...
- ▶ Conceptually decentralized

Multiagent Systems: Agents Help Principals

Realize a decentralized, loosely coupled system to promote flexibility

Heterogeneous agents encode decision making of their respective principals



What we need: Protocols at two levels

Operational (Regulative) Constraints on the ordering and occurrence of interactions

Constitutive The meaning of an interaction on the social tier

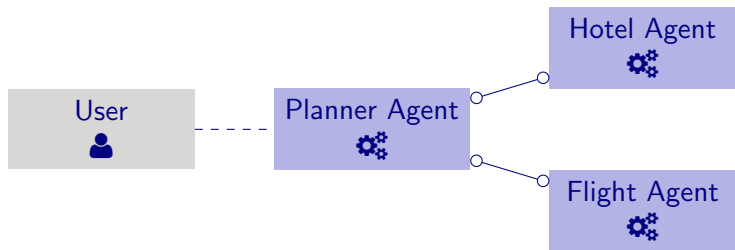
Challenges for Agentic AI Addressed in This Talk

Points raised in keynotes at AgntCon 2026

- ▶ Manik Surtani: agreement is the plumbing
 - ▶ Computational model for agreements
- ▶ David Soria Parra: asynchronous calls (also human elicitation)
 - ▶ Ensuring an agent can interact despite results arriving unexpectedly
 - ▶ Supporting flexible enactments and potential delays
- ▶ Bradley Axen: agents are attached to a human
 - ▶ Accountability and power via agents representing principals
- ▶ Thomas Sarlandie: information has to be discarded from overlong schemas
 - ▶ Compact and precise specifications

Agentic AI Today: Multiagent Lite

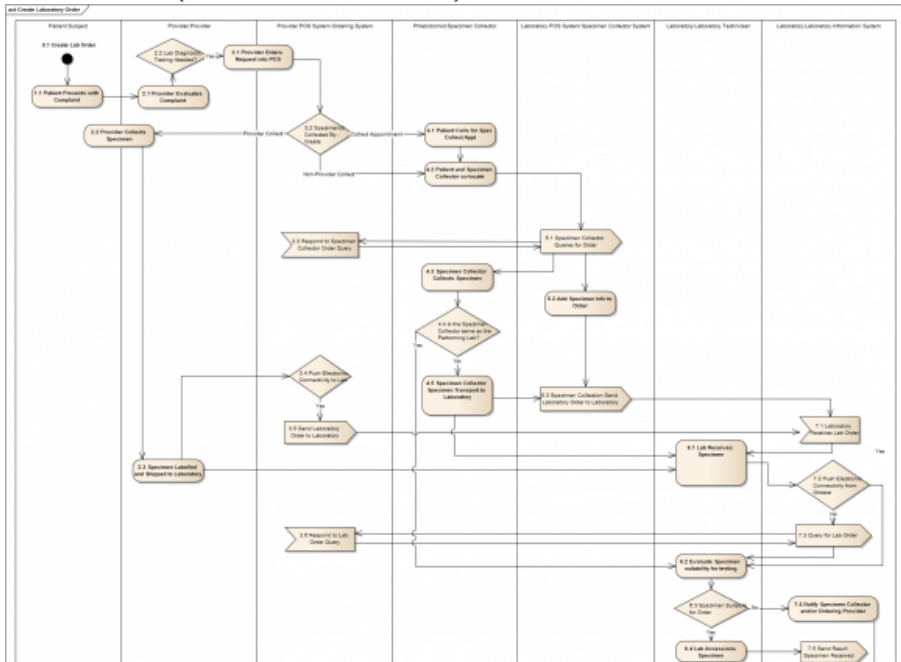
Flexible, generative AI-powered agents that make real-world decisions



Inflexible coordination defeats flexibility

- ▶ Workflows and task graphs adapt the conception of traditional scripting languages
- ▶ No meaning
- ▶ Inflexible operationalization

Workflows (This is from HL7): Don't try to read!



Interaction-Oriented Programming (IOP)

Multiagent system

Agents + Interaction protocol

Method

- ▶ Model a multiagent system in terms of interactions
 - ▶ Meaning via norms to compute accountability
 - ▶ Operational protocols to determine information exchange
- ▶ Verify these models
- ▶ Use the models to realize agents

High-level abstractions: Natural language is imprecise

- ▶ Reflect stakeholder intuitions
- ▶ Let programmers (or LLMs) focus on the reasoning

Outline

Motivation

Motivating Interaction Protocols

Commitments and Other Norms

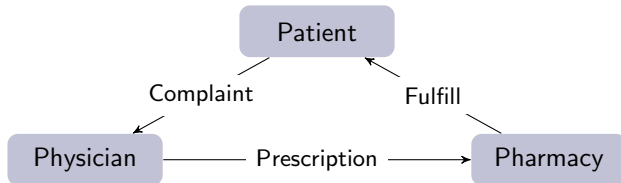
Specifying and Verifying Protocols

LLM-Based Agents for Protocols

Conclusion

Protocol for a Healthcare Application

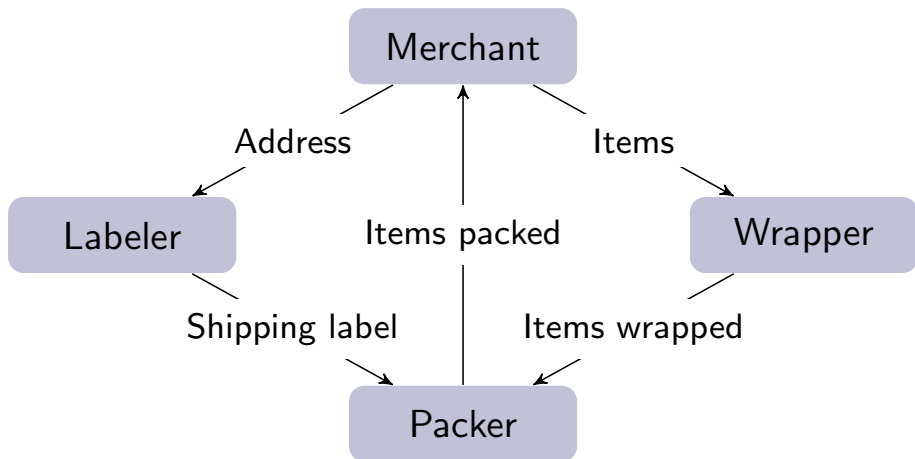
Patient sends a Complaint to Physician, who sends a Prescription to Pharmacy, who sends Fulfill to Patient



- ▶ Autonomy means no one needs to send any message!
- ▶ Three parties, not client server
- ▶ Healthcare standards: Health Level 7 (HL7), Integrate the Healthcare Enterprise (IHE)
 - ▶ Informally described interactions: difficult to implement correctly

Protocol for Purchase Order (PO) Fulfillment Application

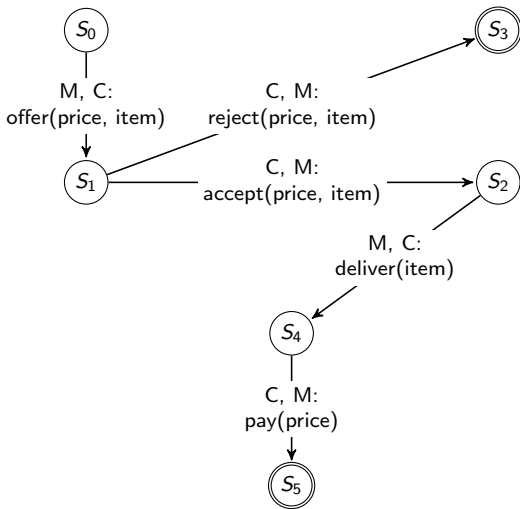
Several items in a PO that may be wrapped and packed independently to create a shipment



Finite State Machine (NetBill, Early Commerce Protocol)

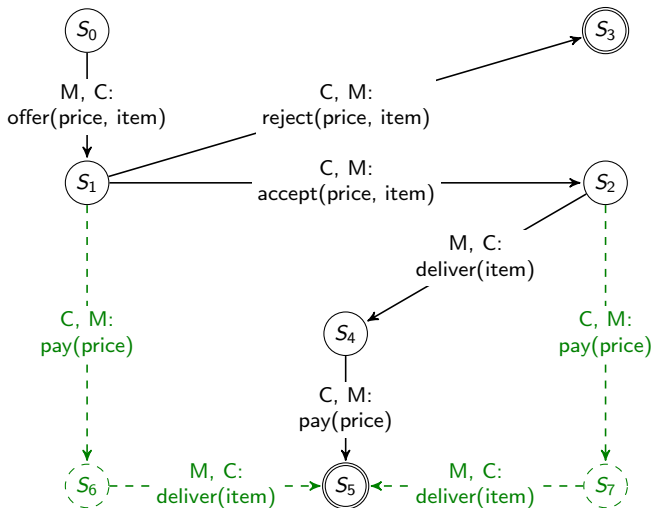
Legitimate protocol: specifies interactions, not internal decision making

- ▶ Roles: merchant (M) and customer (C)
- ▶ Transitions: messages sender, receiver
- ▶ Enactment: *offer, reject*
- ▶ Enactment: *offer, accept, deliver, pay*
- ▶ Correctness: purely operational terms (sequences of messages, not meanings)



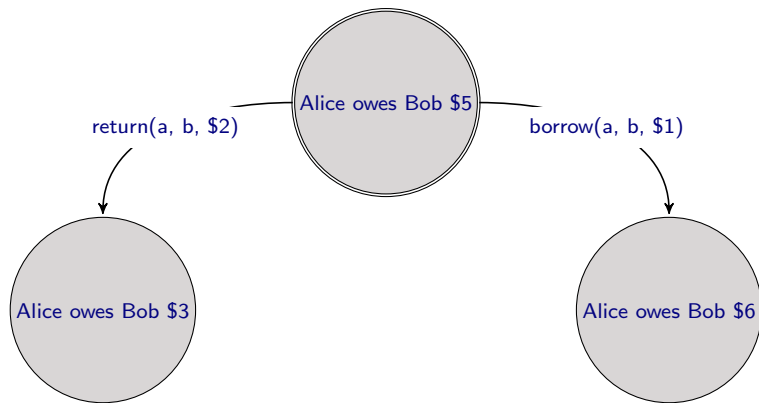
State Machine Example: Generalized

How do we determine some states and transitions are legitimate



Social Machine

Literally a computational system but the states and transitions have meaning
Provides a basis for distinguishing correct from incorrect interactions



Inference based on application meaning produces the machine for free
Overlaid on the events observed

Outline

Motivation

Motivating Interaction Protocols

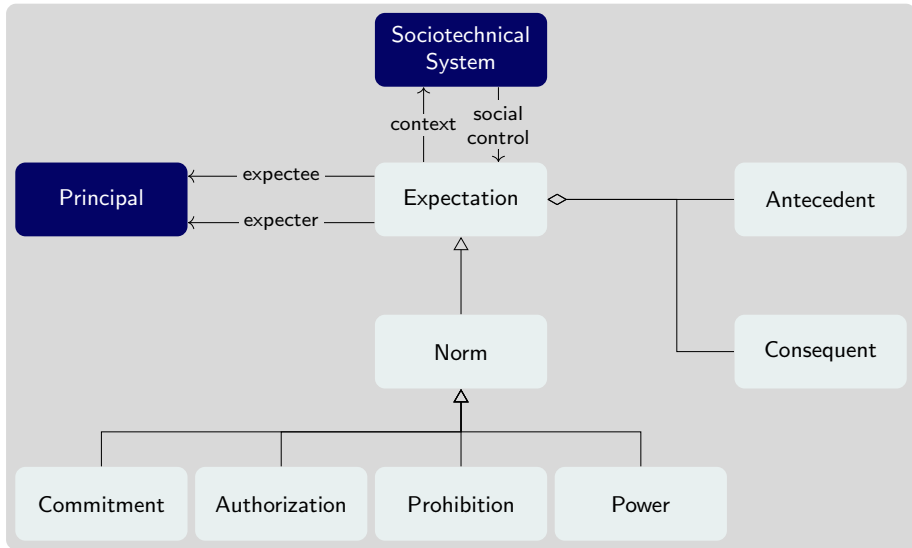
Commitments and Other Norms

Specifying and Verifying Protocols

LLM-Based Agents for Protocols

Conclusion

Specify Social Expectations as Norms



Norms are Directed and Characterize Accountability

Specify the social architecture of a sociotechnical system

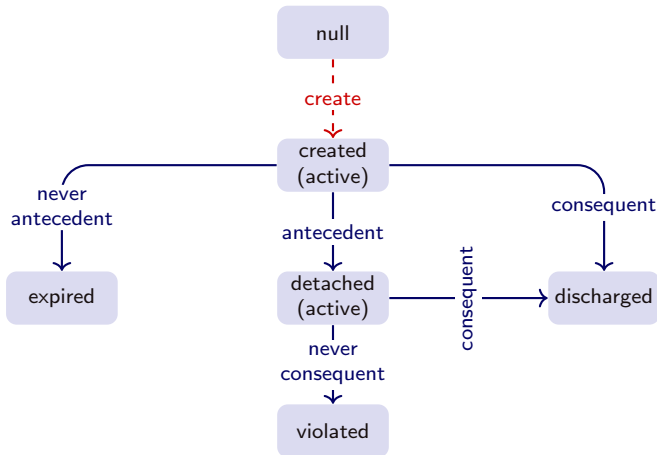
Norm Type	Accountable (Expectee)	Privileged (Expecter)
<i>Commitment</i>	Debtor	Creditor
<i>Prohibition</i>	Prohibited	Prohibiter
<i>Authorization</i>	Authorizer	Authorizee
<i>Power</i>	Empowering	Empowered

Lifecycle of a Commitment Instance

Some transitions are logic-based (based on antecedent, consequent)

Other transitions are purely autonomous (e.g., create, cancel, release)

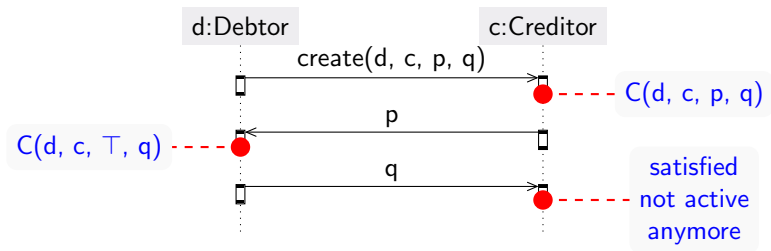
commitment(Alice, Bob, pay \$1, deliver water bottle)



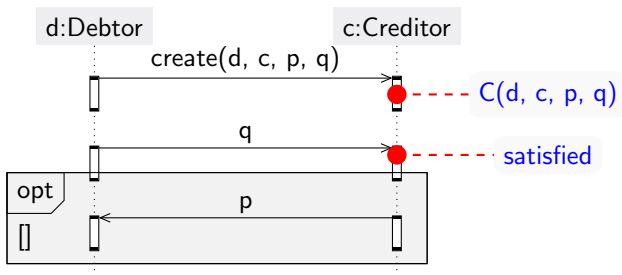
Operationalizing Commitments: Strengthening by Creditor

$C(\text{debtor}, \text{creditor}, \text{antecedent}, \text{consequent})$

p detaches and strengthens the commitment to make it unconditional

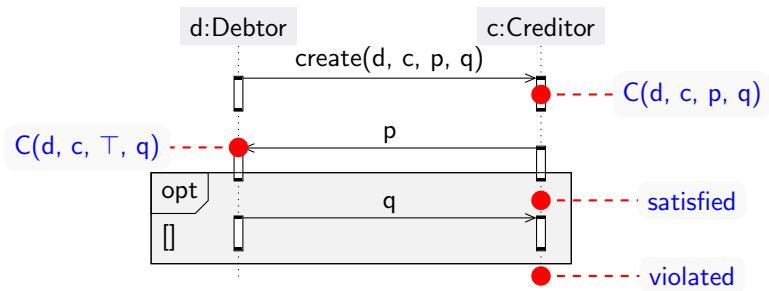


Operationalizing Commitments: Discharge First



Once it is discharged (with q), the detach (with p) is a noop

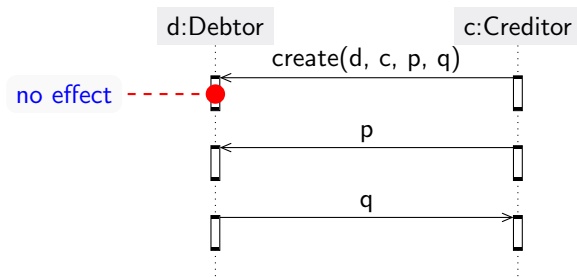
Operationalizing Commitments: Optional Discharge



Violated if never discharged

Operationalizing Commitments: Creation by Creditor

$C(\text{debtor}, \text{creditor}, \text{antecedent}, \text{consequent})$



Bogus! Violates autonomy

A typical failing of traditional models

Cupid: Commitment Specification Based on Information

E-commerce setting

```
1 schema
2   Quote(S, B, ID, item, uPrice) with key ID
3   Accept(B, S, ID, qty, addr) with key ID
4   Payment(B, S, ID, pPrice) with key ID
5   Shipment(S, B, ID, addr) with key ID
6   Refund(S, B, ID, rAmount) with key ID

1 commitment DiscountQuote S to B
2   create Quote
3   detach Accept[, Quote + 4] and Payment[, Quote + 4]
4     where pPrice >= 0.9 * uPrice * qty
5   discharge Shipment[, detached DiscountQuote + 4]
```

Discharged and Violated Instances as of 16 May 2026

Based on event instances for Quote, Accept, Payment, and Shipment

Quote			
ID	item	uPrice	t
T1	fig	1	1 May 2026
T2	pear	1	1 May 2026

Accept			
ID	qty	addr	t
T1	1	Lancaster	2 May 2026
T2	1	Raleigh	2 May 2026

Payment		
ID	pPrice	t
T1	1	2 May 2026
T2	1	2 May 2026

Shipment		
ID	addr	t
T1	Lancaster	3 May 2026

discharged	
ID	t
T1	3 May 2026

violated	
ID	t
T2	7 May 2026

Example: Compensation

A commitment that refers to the state of another commitment

```
1 commitment Compensation S to B
2   create Quote
3   detach violated(DiscountQuote)
4   discharge Refund[, violated(DiscountQuote) + 9] where
      rAmount = pPrice
```

Generate Canonical Queries for Each Norm Type

For a norm's created, detached, expired, discharged, and violated instances

- ▶ Generated SQL is long and complicated
 - ▶ The query for *violated DiscountQuote* is 413 lines long
 - ▶ Nested commitments are more complex
 - ▶ Other norms are similar
- ▶ Generated NoSQL (CouchDB) is also long and complicated
- ▶ Good argument for neurosymbolic architecture, leaving this hard reasoning to a symbolic layer

Outline

Motivation

Motivating Interaction Protocols

Commitments and Other Norms

Specifying and Verifying Protocols

LLM-Based Agents for Protocols

Conclusion

Declarative Operational Protocols

Correct operations are needed to ensure

- ▶ Each agent can perform any legitimate action for which it has (or ought to have) the information
- ▶ The interactions are safe and live

Langshaw, declarative language for highly flexible protocols

- ▶ Succinct *and* flexible
- ▶ New primitive for *sayso* over information in the *social state*
- ▶ Formal semantics of social state updates

Tool support for Langshaw

- ▶ Efficient verification despite high flexibility
- ▶ Automatic translation to asynchronously enactable protocol to flexibility

Example Protocol in Langshaw

Most innovative part is Sayso

```
1 Purchase // Name of the protocol
2 who Buyer, Seller, Shipper // Roles
3 what ID key, Reject or Deliver // Completion (for liveness)
4 // Action schemas to change the social state
5 do Buyer: RFQ(ID, item)
6 do Seller: Quote(ID, item, price)
7 do Buyer: Accept(ID, item, price, address)
8 do Buyer: Reject(ID, Quote)
9 do Seller: Instruct(ID, Accept, item, address, fee)
10 do Shipper: Deliver(ID, Instruct, item, address)
11 // Social authority over information (may generate bindings)
12 sayso Buyer > Seller: item // Buyer has higher authority
13 sayso Seller > Buyer: price // Seller's is higher
14 sayso Buyer: address
15 sayso Seller: fee
16 // What action pairs are incompatible
17 nono Accept Reject
18 nono Reject Deliver
19 // What action prevents another
20 nogo Reject ↗ Instruct
```

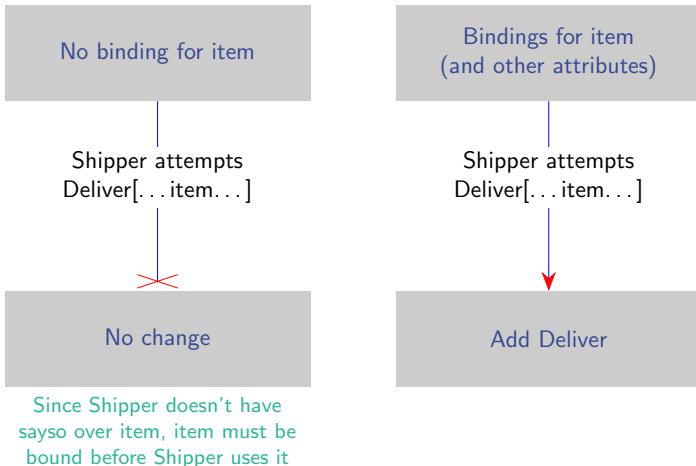
Operational Semantics: Social State $\rightarrow$ Next Social State

Social state = set of all (social) actions that have occurred

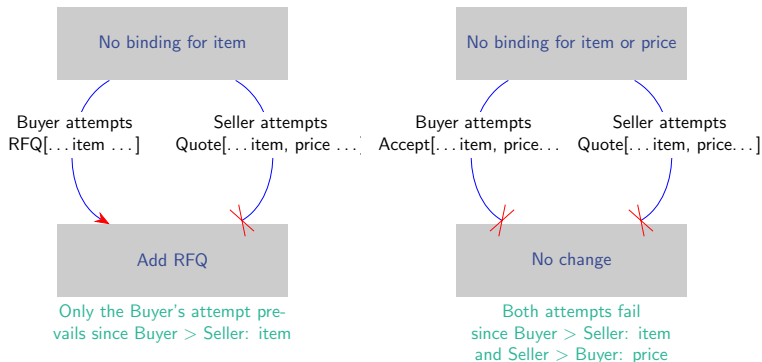
Agents concurrently attempt many actions each

The semantics tells us which attempts succeed

Some simple cases below for intuition



Operational Semantics: Social State $\rightarrow$ Next Social State



Safety: A good protocol avoids errors of concurrency

Individually correct agents may conflict: we check they don't

Mapping to asynchrony requires communication to prevent races

Generating an Asynchronously Enactable Protocol

A generated protocol includes the communication necessary for sayso, nono, nogo
Sayso example: given $\text{BUYER} > \text{SELLER}$: item, BUYER has first dibs on binding item

Buyer

Seller

RFQ[Didn't bind item, Delegating authority]

Quote[Bound item, Exercised authority]

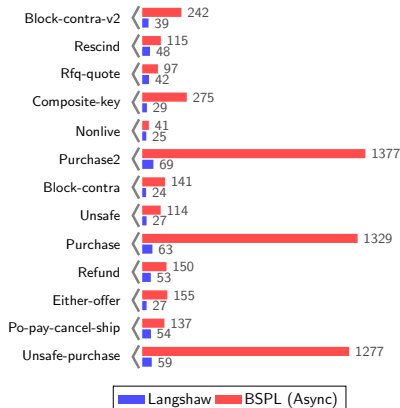
Buyer

Seller

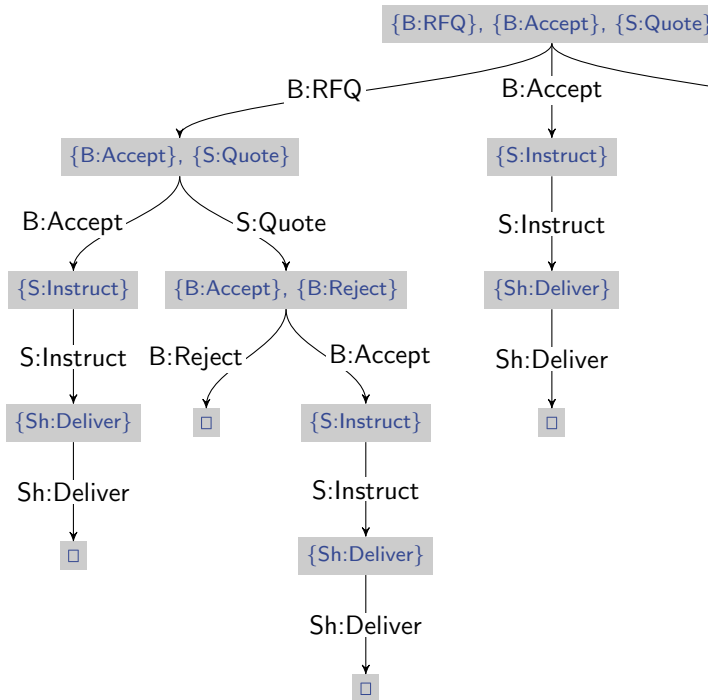
RFQ[Bound item, Exercised authority]

Quote[Used binding, Respected authority]

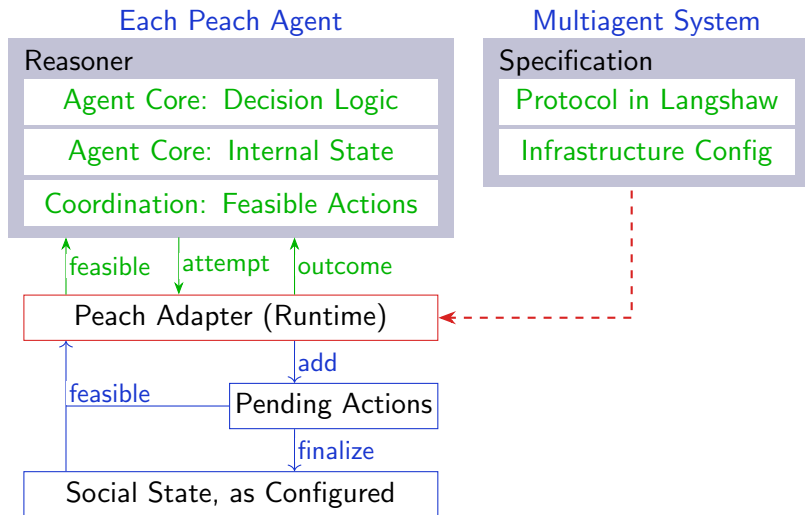
Reduced tokens in spec



- ▶ **Fragment of a tableau generated for verification:**
node labels are enabled action sets
- ▶ To verify a property:
assert proposition at root; expand until contradiction
- ▶ **Sizable reduction over naive approach**

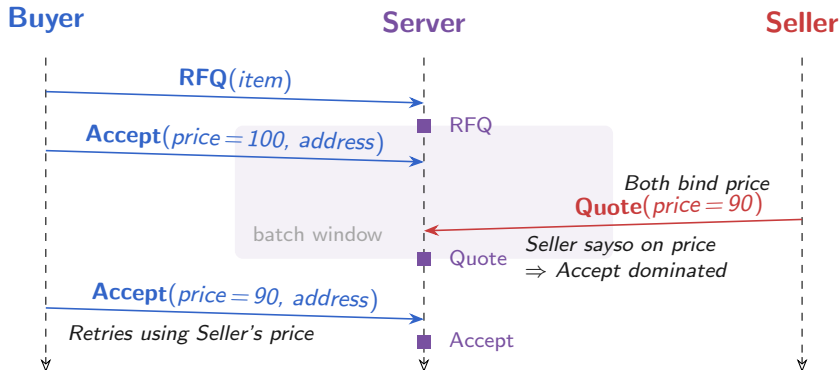


Architecture: How an Agent Participates in a Protocol

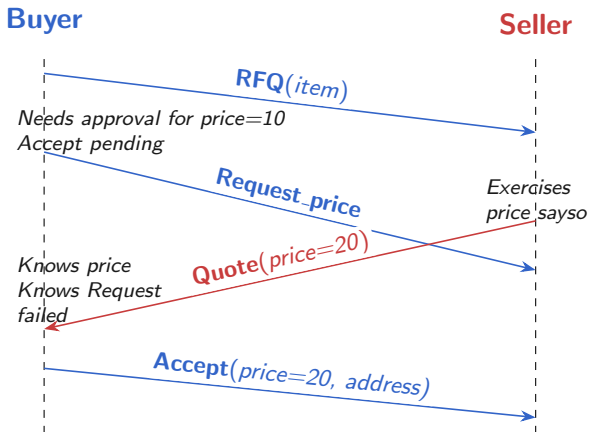


Synchronous Batched Enactment

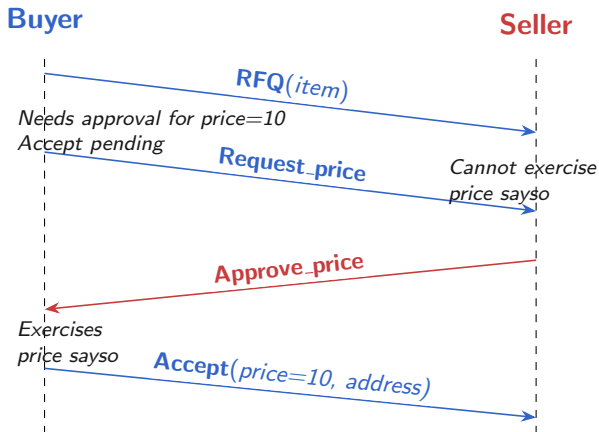
Quote dominates Accept



Asynchronous Enactment: Quote dominates Accept



Asynchronous Enactment: Accept dominates Quote

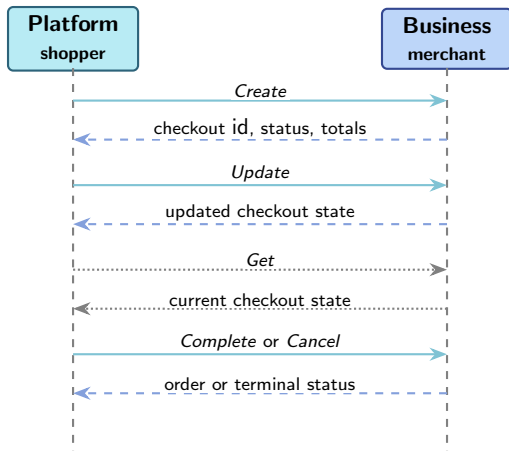


Case Study: UCP Checkout

Message schemas

Sequence of request-response exchanges

No formal representation of meaning



Operations

Create

Start a checkout; BUSINESS generates the session id

Update

Supply or overwrite checkout data

Get

Recover the current session state

Complete

Pay and create the order

Cancel

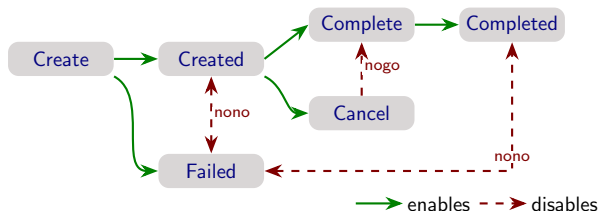
Abandon the checkout

Simplified UCP in Langshaw

```
1 SimpleUCP
2 who Platform , Business
3 what cid key , Completed or Failed
4 do Platform: Create(cid , line_items , currency , buyer ,
    payment_pref , discount_codes , fulfillment_pref)
5 do Business: Created(cid , Create , id , status , totals , payment
    , discounts , fulfillment)
6 do Business: Failed(cid , reason)
7 do Platform: Complete(cid , Created , id , payment_data ,
    risk_signals)
8 do Platform: Cancel(cid , Created , id)
9 do Business: Completed(cid , Complete , order)
10 sayso Platform: line_items , currency , buyer , discount_codes ,
    fulfillment_pref , payment_pref , payment_data ,
    risk_signals
11 sayso Business: id , status , totals , payment , discounts ,
    fulfillment , order , reason
12 nono Completed Failed
13 nono Created Failed
14 nogo Cancel ↗ Complete
```

SimpleUCP Highlights

- ▶ `cid` gives every enactment an identity before HTTP assigns an id
- ▶ **sayso** records authority
- ▶ **nono** and **nogo** make conflicts explicit



Parismony: the resulting PLATFORM and BUSINESS agents are order of magnitude smaller than Google's reference implementations

Programming a Peach PLATFORM Agent

The adapter offers semantically acceptable actions;

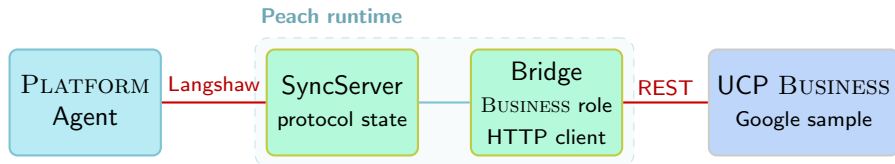
the agent selects an action and determines what is needed to perform it

```
1 @adapter.on_enabled('Create') # reasons about protocol state
2 async def on_create_enabled(fa):
3     await adapter.attempt([fa.bind( # submits action instance to
4         SyncServer
5         line_items=SAMPLE_LINE_ITEMS,
6         currency="USD",
7         buyer=SAMPLE_BUYER,
8         payment_pref={},
9         discount_codes={"codes": []},
10        fulfillment_pref=SAMPLE_FULFILLMENT_PREF)])
11 @adapter.on_enabled('Complete') # reasons about protocol state
12 async def on_complete_enabled(fa):
13     await adapter.attempt([fa.bind( # submits action instance to
14         SyncServer
15         payment_data=SAMPLE_PAYMENT_DATA,
16         risk_signals={"ip": "127.0.0.1"})])
```

- This programming model abstracts away the protocol semantics

Architecture

Ideal for Langshaw endpoints; interoperates with legacy REST endpoints

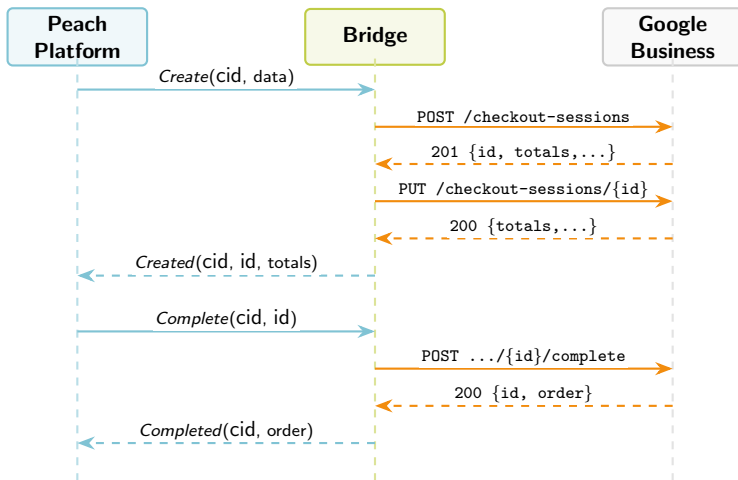


SyncServer provides runtime support

- ▶ Maintains protocol state
- ▶ Determines which actions are enabled to guide the agent's reasoning
- ▶ Verifies actions attempted by the agent and denies invalid ones
- ▶ Accommodates infrastructure variants
 - ▶ Request-response
 - ▶ Batched as in common ledgers (e.g., banking ACH or blockchain)
 - ▶ Fully asynchronous

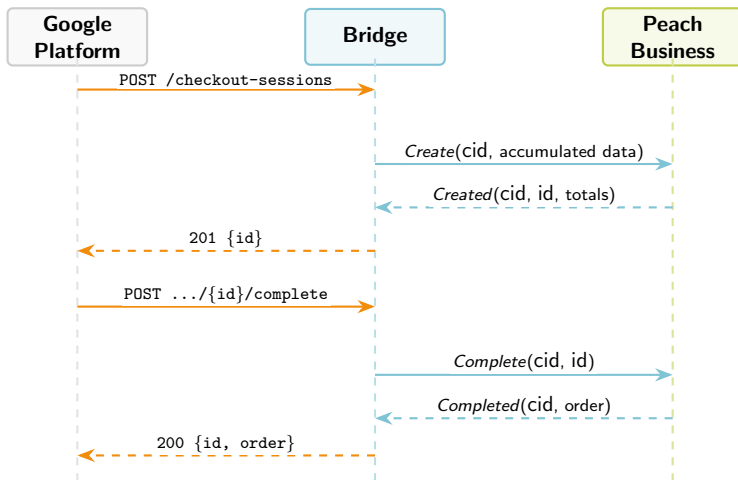
Completed Checkout

Peach PLATFORM agent, Google's reference BUSINESS agent



Completed Checkout

Google's reference PLATFORM agent, Peach BUSINESS agent



Outline

Motivation

Motivating Interaction Protocols

Commitments and Other Norms

Specifying and Verifying Protocols

LLM-Based Agents for Protocols

Conclusion

Multiagent System

Protocol captures interaction constraints

- ▶ Supports interoperation and loose coupling
- ▶ Lays out the possible interactions

Agent represents a principal and captures their internal reasoning

- ▶ Helps automate interactions
- ▶ Chooses from the possible interactions

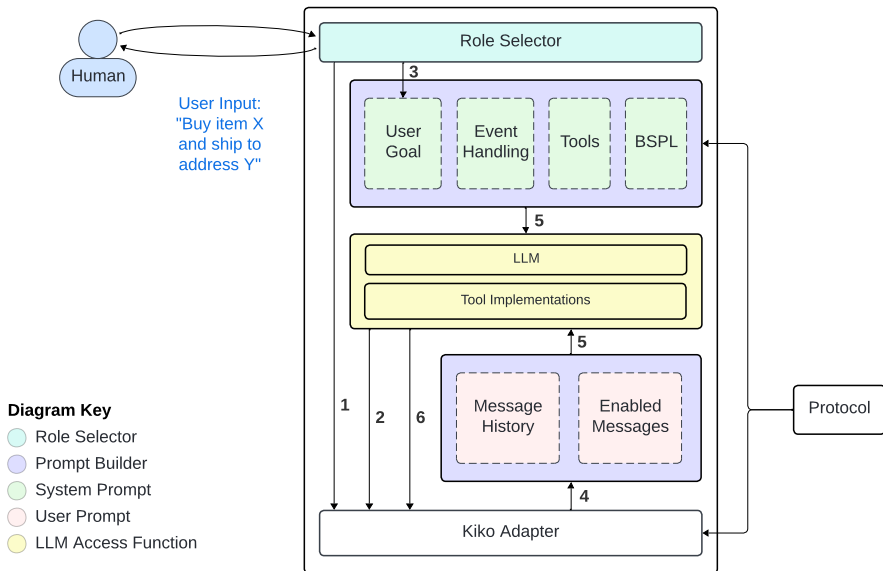
LLM agents that select and participate in a protocol

- ▶ *How can we exploit LLMs to create appropriately flexible agents without programming them?*

LLM Agents Promise Flexibility But Lack Rigor

```
1 ## Pure LLM agent
2 response = client.chat.completions.create(
3     model="gpt-4",
4     messages=[
5         {"role": "system", "content": "You are a helpful -
6           assistant and you need to achieve -" + [User Goal]},
7         {"role": "user", "content": [Decision Context] + "Decide
8           - what the next step is."}
9     ]
10 )
11
12 print(response.choices[0].message.content)
13
14 ## Example output
15 ## (Not a message defined in the protocol)
16 {
17     "message": "buy",
18     "item": "pen",
19     "price": 20,
20 }
```

Ahoy Architecture: LLM Agent Designed for Interaction



Configuring an Ahoy Agent

User goal

```
Describe the scenario or goal you'd like ahoy to participate in.  
(Enter multiple lines. Type 'END' on a new line to finish)
```

```
I need to buy a pen for less than $20.  
Delivery location: Raleigh, NC 27606  
END
```

Available protocols (from a repository)

```
=====
Available Protocols
=====

1. Purchase: [Buyer, Seller, Shipper]
2. CreditPurchase: [CreditBuyer, CreditSeller, CreditShipper]
3. Logistics: [Merchant, Wrapper, Labeler, Packer]
4. FlexiblePurchase: [FlexibleCustomer, FlexibleMerchant]
5. NetBill: [Customer, Merchant]

Select a protocol (number or name):
```

Ahoy Agent Chooses a Message and Binds Its Parameters

```
=====
2026-04-27 20:03:53,667 - DEBUG - RAW LLM RESPONSE
2026-04-27 20:03:53,667 - DEBUG - =====
2026-04-27 20:03:53,667 - DEBUG - Looking at my goal and the current state:

**Goal Analysis:**
- Need to buy a pen for less than $20
- Delivery location: Raleigh, NC 27606

**Current Situation:**
- I've already requested a quote for a pen
- The seller quoted $3 for a pen
- The price ($3) is well under my $20 budget ✓
- I need to accept this offer and proceed with delivery

**Decision:**
Option 1 (Purchase/accept) is the appropriate next step. I should accept the quote with my delivery address.

```json
{
 "choice": 1,
 "params": {
 "address": "Raleigh, NC 27606",
 "resp": "Accepted - proceeding with purchase"
 },
 "tool_requests": []
}
```
```

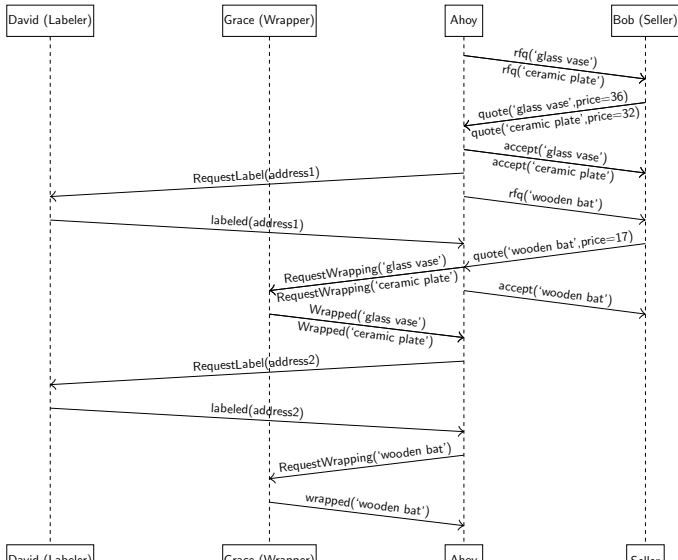
Programming Not Needed

Task: “Buy a pen for \$20 and deliver it to address ...”.

```
rfq(ID=[ID],item='pen'){protocol='Purchase'}  
accept(ID=[ID],item='pen',price=15,address=[user_address],resp='Accepted.  
    Please proceed with shipment to Raleigh, NC 27606.'){protocol='  
    Purchase'}  
completed(ID=[ID],item='pen',price=15,satisfaction='Transaction completed  
    successfully. Pen received in good condition at [user_address].'){  
    protocol='Purchase'}
```

Concurrent Participation in Two Protocols

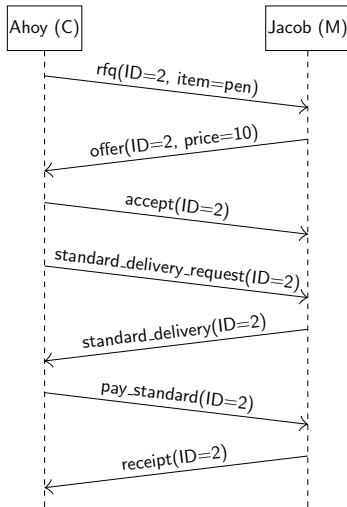
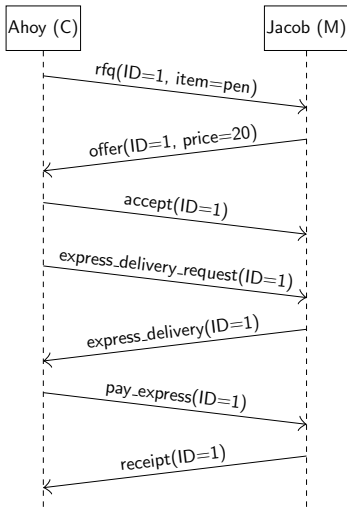
"Buy a glass vase and a ceramic plate and ship to address1, and buy a bat and ship to address2. The total budget is \$200."



Flexibly Enacting Branching Protocols: Alternative Paths

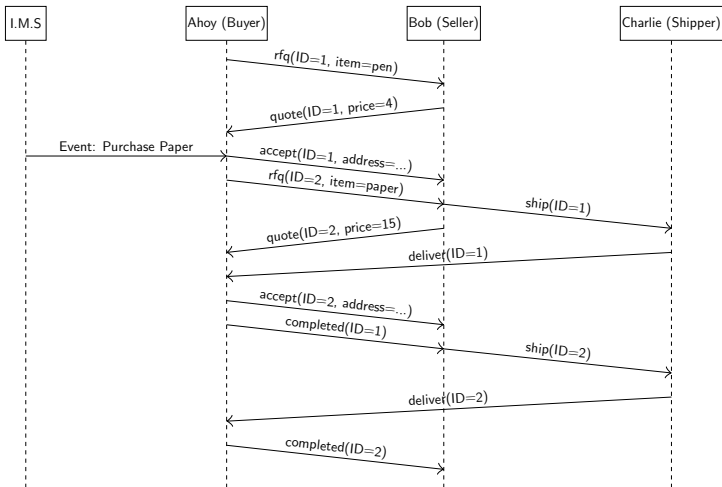
I want a pen delivered urgently

I want a pen delivered cheaply



External Event Handling

"Buy a pen and deliver to address."



Evaluation: Ahoy Agents

- ▶ Respect protocol semantics
- ▶ Select actions intelligently given user goals
- ▶ Can interleave multiple protocol enactments
- ▶ Handle exogenous events

| Problem | Solution |
|---|---|
| Poor understanding of interaction:
mapped goal to incorrect protocol | Ongoing work to support norms |
| Hallucination: hallucinated ID values | Generated and passed IDs |
| Insufficient commonsense knowledge:
Bought pen for \$10k | Budget given in goal |
| Poor knowledge of business practices:
Kept checking with seller repeatedly | Explicitly instructed to not
spam seller |

Outline

Motivation

Motivating Interaction Protocols

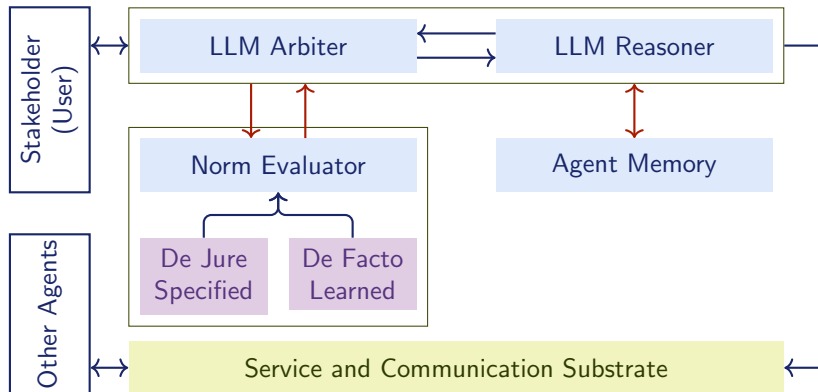
Commitments and Other Norms

Specifying and Verifying Protocols

LLM-Based Agents for Protocols

Conclusion

Agentic Approach to Norms



Norms Ground Accountability

Zero-agent and one-agent obligations are inadequate

Accountability: ≥ 2 parties: one to call another to account for its actions

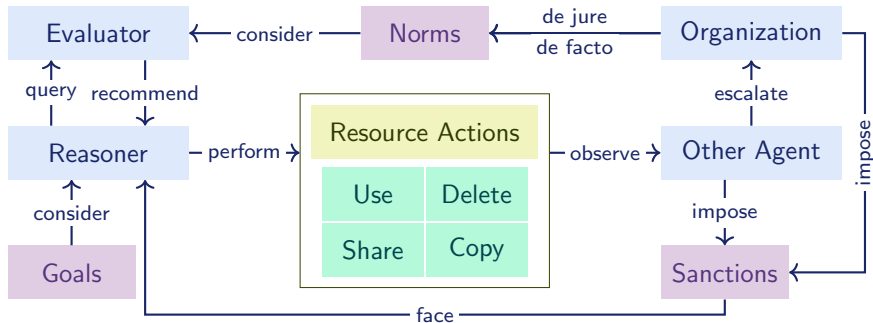
Alternatives vary on accountability



► Explanations and arguments to override prima facie expectations

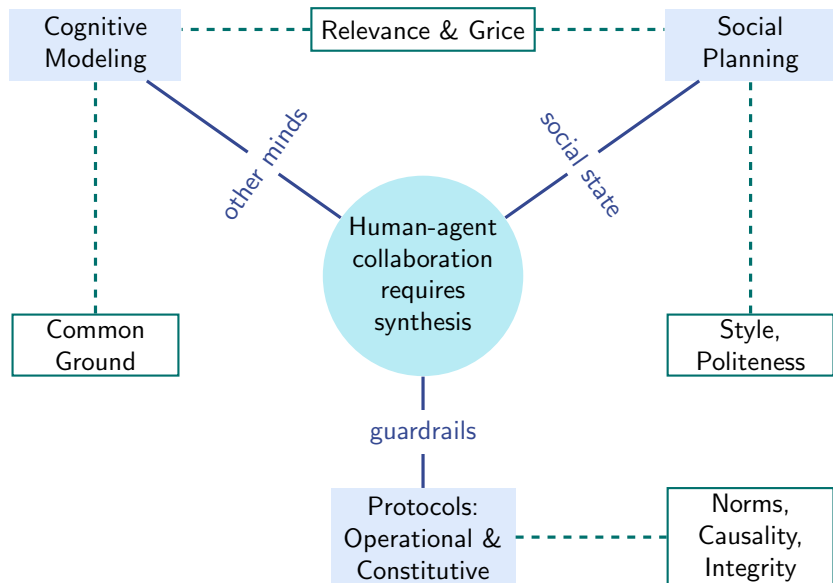
An Agent May Violate Norms

An agent is guided by organizational de jure or de facto norms



Sanctions may come directly from agents or via escalation to an organization

Human-AI Cooperation



Norms and Declarative Protocols $\mapsto$ Multiagent AI

| Conventional Agentic AI | Interaction Orientation |
|--------------------------------|-------------------------------|
| Orchestration | Interactions |
| Natural language: ambiguous | Formal, declarative, flexible |
| Over-constrained communication | Flexible communication |
| Soft guardrails | Hard guardrails |
| LLM agents | Theory based (LLM ...) agents |

Engineering is a science of the artificial (Herb Simon)

Models are important because they precisely capture requirements of the engineered artifact

Thanks!

- ▶ National Science Foundation
- ▶ EPSRC
- ▶ Consortium for Ocean Leadership



Engineering Multiagent Systems



Munindar P. Singh² Amit K. Chopra¹ Samuel H. Christie V²
(with contributions from Matteo Baldoni³ and Omkar Joshi²)

¹Lancaster University

²North Carolina State University

³University of Torino

AgntCon 2026