Chapter 1
Tools for Implementing Multiagent Systems Based on Protocols

Amit K. Chopra[0000-0003-4629-7594],
Samuel H. Christie V[0000-0003-1341-0087], and
Munindar P. Singh[0000-0003-3599-3893]

Abstract Interaction-Oriented Programming (IOP) is an approach to building a multiagent system by modeling the interactions between its roles via a flexible interaction protocol and implementing agents to realize the interactions of the roles they play in the protocol.

In recent years, we have developed an extensive suite of software that enables multiagent system developers to apply IOP. These include tools for efficiently verifying protocols for properties such as liveness and safety and middleware that simplifies the implementation of agents. This paper presents some of that software suite.

1.1 Motivation

Software systems increasingly support interactions among autonomous principals (humans and organizations) in various settings, e.g., e-commerce, health-care, finance, and so on. Owing to autonomy, it is natural to realize such systems as multiagent systems in which principals are represented by agents who interact by exchanging messages.

The main challenges in realizing multiagent systems can be divided into broad categories. One, how can we model a multiagent system in a manner that enables realizing it as a loosely-coupled system of agents who can interact with each other with maximal flexibility. Intelligent actions by agents presumes flexibility; it means an agent can take into account the relevant circumstances in making decisions. Loose coupling [52, 61] means minimizing and making explicit

Lancaster University, UK,
e-mail: amit.chopra@lancaster.ac.uk
North Carolina State University,
e-mail: schrist@ncsu.edu
North Carolina State University,
e-mail: singh@ncsu.edu

the architectural assumptions that agents (as system components) make of each other. Loose coupling thus promotes architectural simplicity, clarity, and efficiency. Crucially, it enables implementing an agent without knowing how other agents are implemented (which in open systems may not be possible anyway).

Two, given a model of a multiagent system, we are faced with the problem of implementing an agent to play a role in it. How can we exploit the model to facilitate such engineering? In particular, how can we engineer flexible, fault-tolerant agents that are correct with respect to the model in a manner that enables programmers to focus on encoding an agent's decision making.

Modeling multiagent systems, first and foremost, in terms of *interaction protocols* is the most promising way we know of addressing these challenges. This approach offers several benefits. First, a protocol supports autonomy by capturing communication constraints but otherwise leaving the principals free to apply their own business logic and engage flexibly. Second, a protocol helps principals implement their software agents by providing role-based interfaces. Third, protocols enable realizing software in terms of loosely coupled, decentralized components (the agents). Fourth, protocols may be composed and verified, which enables reasoning about a system before any agents are implemented [69]. Fifth, they support high-level abstractions such as social commitments [63] and, more generally, norms, which inform agent decision making.

The importance of interaction protocols was recognized early on in multiagent systems research [36, 39], which spurred research on languages for specifying protocols. AUML [50] was a notable early result of this activity and was highly influential, finding application in communication standards and software methodologies. However, AUML and most of the other work (reviewed in Section 1.2) that followed it did not enable realizing the above-mentioned benefits.

The situation changed with the development of information protocols [65, 66, 67, 68], a novel declarative approach for specifying multiagent protocols. Traditionally, protocols specified message ordering. Information protocols depart from this tradition by specifying information causality and integrity constraints on communication. In recent years, we have built a software suite that enables verifying information protocols and implementing flexible, robust agents based on high-level information-based abstractions.

Our goal in this contribution is to show via concrete examples how this suite enables realizing all of the above-mentioned benefits.

1.2 Literature

We review the literature on related themes.

1.2.1 Agent Communication Languages

Agent communication is usually understood as building on elements of the philosophy of language, especially *speech act theory* [4]. Here, a communication is an action performed by the speaker (depending on context). For example, if a referee in a soccer match says "foul," it's a foul (and changes the state of the game), but if a player says "foul," it has no such effect. If a fan skeets "foul by Smith" they may inform (or misinform) their readers, but cannot cause a foul to be recorded.

Speech act theory separates propositional *content* (e.g., "foul") from *illocutionary type* (e.g., *declarative* to make the content true or *informative* to report on the content). Traditional AI approaches define a handful of message formats, one for each major illocutionary type, which is inherently limited since there are potentially as many sets of illocutions as multiagent systems [17, Section 7]. KQML (Knowledge Query and Manipulation Language) [32] was designed for agents viewed as homogeneous knowledge bases (KBs). The agents can query each other's KBs and tell each other facts to be believed, as well as give commands to be achieved. FIPA [54] is a successor to KQML: streamlined but similar in spirit to it [63].

Modeling meaning is essential for reasoning about interactions. For example, a "quote" may refer to

1. the last trading price on a stock exchange
2. an offer to sell

Yet, current approaches specify meaning only informally. The early formal models, such as FIPA and KQML, follow a *mentalist* semantics [60], interpreting communicative acts in terms of the beliefs and intentions of the participants. Such models are unsuited to autonomous and heterogeneous agents, whose internal representations are hidden.

We established the contrary *social* semantics in AI [63], about the social semantics of a communicative act based on the commitments it presupposes and alters [64]. Modern philosophy of language is reviving elements of Austin's theory pertaining to conventions and norms [12, 58, 59], which were downplayed in the purely cognitive approaches. When the social state is expressed via norms, communications map to changes in norms and can be formally reasoned about [23]. For example, an offer by a seller creates a commitment; a buyer's acceptance of an offer makes the buyer committed to paying the offered price and advances the seller's commitment to one where the seller has to supply the goods.

1.2.2 Protocol Languages

Protocols are crucial to multiagent systems engineering methodologies [15, 29, 51, 56]. However, protocols are traditionally expressed in informal UML-inspired notations such as *Agent UML* [40]. FIPA [34] specifies a select few protocols using AUML. An agent plays a role in a protocol and communicates accordingly.

However, FIPA and UML provide no formal model of the protocol. Since they lack a formal semantics, it is not possible to verify protocols for desirable properties, provide a principled programming model, or check whether each message respects the protocol semantics. Moreover, the few protocols that FIPA specifies cannot meet the requirements of the potentially infinite variety of multiagent systems. Thus, they fall short of the goals for engineering multiagent systems [79].

Formal protocol specification approaches generally express the information content implicitly and coordination explicitly, which limits flexibility. Baldoni et al. [6, 5] specify protocols as state machines where the transitions represent messages. Ferrando et al. [31] specify protocols as trace expressions over messages. Winikoff et al. [80] specify protocols via a notation reminiscent of statecharts and augmented with information constraints. ASEME [73] is another model inspired by statecharts. Although these approaches can be applied toward engineering nominally decentralized multiagent systems, these approaches are neither conducive to loose coupling nor flexibility.

1.2.3 Agent Programming

JADE [8, 9], a programming model for multiagent systems, is noteworthy for its early support for FIPA protocols [34]; however, as discussed above, the FIPA approach is long outdated [63] and the FIPA protocols are limited to a few patterns of interaction specified in terms of message ordering. SARL [35] is an imperative language for agents; it lacks support for protocols.

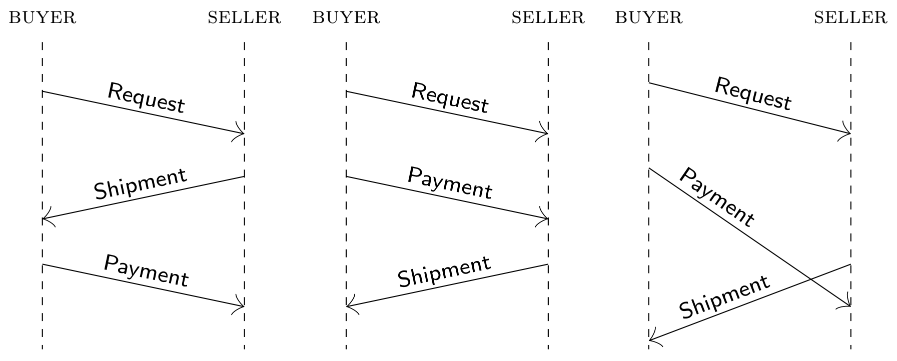
Agent-oriented programming models such as Jason [11] and JaCaMo [10] provide cognitive abstractions for encoding an agent's internal reasoning but do not support protocols. Jason uses KQML-inspired communication abstractions; JaCaMo includes Jason and, in addition, supports communication between agents via Web services-style artifacts.

1.3 Toolkit Description

We first introduce the idea of information protocols. Then we introduce related tooling. First, we describe a tool for verifying information protocols. Then, we describe protocol-based programming models, specifically Kiko [26] and Mandrake [25]. Kiko demonstrates how a generic information-based adapter abstracts over the network and presents a simple information-based interface for programming agents. Mandrake shows how agent developers may specify application-level message forwarding policies to deal with potentially lost messages.

1.3.1 Information Protocols

We introduce a simple ebusiness scenario in which BUYER may *Request* some item from a SELLER. After sending the *Request*, BUYER may send *Payment* at any time; after receiving *Request*, SELLER may send *Shipment* at any time. Notice, therefore, that it is possible that BUYER and *seller* send *Payment* and *Shipment* concurrently. Indeed, all enactments of Figure 1.1 are possible, highlighting the flexibility we desire.



(a) Shipment first. (b) Payment first. (c) Concurrent.

Fig. 1.1: Three possible enactments in an ebusiness scenario.

The scenario described above cannot be captured in traditional protocol specification approaches [20]. In the information protocols paradigm, it is straightforward to capture it. Listing 1 gives an information protocol for this scenario.

Listing 1: An information protocol.

```
Flexible Purchase {
 role B, S
 parameter out ID key, out item, out status, out paid

 B ↦ S: Request[out ID, out item]
 S ↦ B: Shipment[in ID, in item, out status]
 B ↦ S: Payment[in ID, in item, out paid]
}
```

In the listing, ⌜in⌝ and ⌜out⌝ capture information dependencies. An agent can send any message whose information dependencies are satisfied by its *local state* (the set of messages sent and received by the agent). The adornment ⌜in⌝ for a parameter means that the binding for the parameter must exist in the agent's local state; ⌜out⌝ means that the binding for the parameter must not exist in the local state, but the act of sending the message generates it. Thus, enactments of Figure 1.1 are all entertained because sending and receiving *Request* satisfy the information dependencies for sending *Payment* and *Shipment*, respectively. Notice that specifying information dependencies as described above means messages can be received in any order. This operational flexibility truly liberates decision making.

For example, even though BUYER can emit *Payment* only after emitting *Request*, SELLER can receive *Payment* first. Further, whenever SELLER receives *Payment*, regardless of whether it has received *Request*, *S* may emit *Shipment* (because its information dependencies would be satisfied). Further, retransmissions of messages and receptions of duplicate messages are harmless because, information-wise, they are idempotent. What this means is that information protocols can be flexibly enacted over unordered, lossy communication services such as the Internet.

1.3.2 Tango

Before a protocol may be used to implement agents, we would like to verify that it has certain desirable properties. Tango is an approach for verifying the safety and liveness of information protocols [70]. A protocol is *safe* if no enactment may generate more than one binding for a parameter. A protocol is *live* if any enactment is able to progress to completion. Tango is implemented in a command line tool called `bspl`, available at https://gitlab.com/masr/bspl/

To verify a property, we must check that it holds for all possible protocol enactments. Because information protocols can be flexibly and fully asynchronously enacted (it requires no assumption about message ordering), a protocol may have a large number of enactments, which can make verification inefficient. An important feature of the Tango approach is that it reduces the set of enactments of protocol to a set of *canonical* enactments and then performs the checking against the set of canonical enactments. For example, all the enactments in Figure 1.1 reduce to one of them. Thus, instead of checking several enactments, we need to check only one of them. This reduction leads to vastly improved verification performance compared to earlier verification approaches for information protocols [67].

Listing 2 gives the results of executing liveness and safety queries for *Flexible Purchase*, the protocol in Listing 1. A *maximal* path is one that cannot be extended by further emissions or receptions by an agent. The third query (`all_paths`) makes it clear that there are 12 maximal paths; the tooling reduces them to one for purposes of liveness and safety checking.

Listing 2: Executing Tango on Flexible Purchase (Listing 1). Output shown in blue.

```
>bspl verify liveness Flexible-Purchase.bspl
```

```
{'live': True, 'checked': 7, 'maximal paths': 1, 'elapsed':
    0.0010309999343007803}

>bspl verify safety Flexible-Purchase.bspl
{'safe': True, 'checked': 7, 'maximal paths': 1, 'elapsed':
    0.0009660000214353204}

>bspl verify all_paths Flexible-Purchase.bspl
40 paths, longest path: 6, maximal paths: 12, elapsed: 0.005732199992053211
(B!Request, S!Shipment, S?Request, B!Payment, S?Payment, B?Shipment)
(B!Request, S!Shipment, S?Request, B!Payment, B?Shipment, S?Payment)
(B!Request, S!Shipment, S?Request, B?Shipment, B!Payment, S?Payment)
(B!Request, S!Shipment, B?Shipment, B!Payment, S?Payment, S?Request)
(B!Request, S!Shipment, B?Shipment, B!Payment, S?Request, S?Payment)
(B!Request, S!Shipment, B?Shipment, S?Request, B!Payment, S?Payment)
(B!Request, S?Request, B!Payment, S?Payment, S!Shipment, B?Shipment)
(B!Request, S?Request, B!Payment, S!Shipment, B?Shipment, S?Payment)
(B!Request, S?Request, B!Payment, S!Shipment, S?Payment, B?Shipment)
(B!Request, S?Request, S!Shipment, B!Payment, S?Payment, B?Shipment)
(B!Request, S?Request, S!Shipment, B!Payment, B?Shipment, S?Payment)
(B!Request, S?Request, S!Shipment, B?Shipment, B!Payment, S?Payment)
```

Listing 3 gives a buggy variant of *Flexible Purchase*. It is not live because status, which must be bound for the completion, cannot be bound in any enactment. It is unsafe because both B and S can send *Payment* and *Shipment* concurrently, thus concurrently binding paid.

Listing 3: An unsafe and nonlive version of the protocol in Listing 1.

```
Buggy Flexible Purchase {
  roles B, S
  parameters out ID key, out item, out status, out paid

  B -> S: Request[out ID key, out item]
  S -> B: Shipment[in ID key, in item, out paid]
  B -> S: Payment[in ID key, in item, out paid]
}
```

Listing 4 shows the outcomes of liveness and safety queries for this protocol, giving counterexamples for both. For liveness, it gives a enactment that is deadlocked. For safety, it reports the parameter (paid) as being multiply bound.

Listing 4: Executing Tango on *Buggy Flexible Purchase*

```
>bspl verify liveness Buggy-Flexible-Purchase.bspl
{'live': False, 'reason': 'Found path that does not extend to completion', '
    path': (B!Request, B!Payment, S?Payment, S?Request), 'checked': 5, '
    maximal paths': 1, 'elapsed': 0.0013166000135242494}

>bspl verify safety Buggy-Flexible-Purchase.bspl
{'safe': False, 'reason': 'Found parameter with multiple sources in a path',
     'path': (B!Request, B!Payment, S?Request, S!Shipment), 'parameter': '
    paid', 'checked': 7, 'maximal paths': 1, 'elapsed':
    0.0017665999475866556}
```

1.3.3 Kiko

Kiko is a protocol-based programming model for agents. Specifically, given an information protocol, it enables implementing agents that play roles in the protocol. To make agent development easy, Kiko includes a middleware that exposes an event-driven, information-based interface that may be used to implement an agent's internal reasoning.

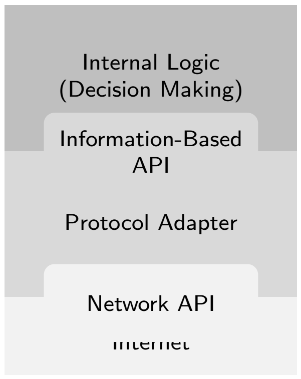


Fig. 1.2: Agent architecture in the Kiko programming model.

As Figure 1.2 shows, each Kiko agent has an information protocol adapter that sits between the network and the agent's decision making, that is, its internal logic. An agent's Kiko adapter maintains its local state. Based on the local state and the protocol specification, it keeps track of information-enabled *forms*. The forms are necessarily partial message instances that would be legal to send if completed. Specifically, a form's ⌜in⌝ parameters are bound (from the local state) and the ⌜out⌝ parameters are unbound (because they don't exist in the local state). (Information protocols may also feature the ⌜nil⌝ adornment, which we omit from this discussion for simplicity.) Figure 1.3 gives a possible local state for a BUYER agent and the forms available to it in that state.

Request(1, fig)
Request(2, jam)
Payment(1, fig, $10)

(a) Local state

Request(ID, item)
Payment(2, jam, paid)

(b) Enabled forms

Fig. 1.3: A possible local state for a BUYER agent and the enabled forms in that state.

To create a Kiko agent, a developer writes a set of *decision makers*. A decision maker is an event-triggered piece of code that gets the set of enabled forms and completes some subset via some logic. The completed forms are emitted by the adapter as messages and added to the local state.

Listing 5 shows a decision maker for a BUYER agent. Its logic is to complete those *Payment* forms for which *Shipment* has been received. The completed *Payment* forms are sent by the adapter as messages. The decision maker is triggered at 1700 hours every day. In other words, it processes enabled *Payment*s in a batch every day.

Listing 5: A BUYER agent's decision maker that sends *Payment* only in those enactments (as identified by ID) in which *Shipment* has been received.

```
@adapter.schedule_decision(00 17 * * *)
def payment(enabled, state):
  payments = enabled.messages(Payment)
  for p in payments:
     if(next(state.messages(Shipment, system=p.system,
         ID=p["ID"])))
        p.bind(paid="10")
```

Listing 6 informally describes the logic of the adapter. In essence, the adapter runs a loop in which it either responds to a trigger by invoking the corresponding decision maker or receives a message (if one is available).

Attempts refer to completed forms. This terminology reflects the fact that completed forms may be mutually inconsistent and therefore must be checked by the adapter before emission. Notice how the adapter abstracts away the actual emission and reception of messages from the agent developer. Unlike alternative approaches, a Kiko developer never needs to *selectively receive* messages—that is, specify which message to receive next and block until it arrives. As in the actor model [38], messages are received and added to the local state as they are made available by the network. A feature of Kiko is that it needs nothing more than UDP (part of the Internet Protocol suite) for transport. UDP, notably, supports only best-effort (unreliable, unordered) message delivery.

Listing 6: Adapter logic.

```
//$t_i$:$d_i$ represents a decision maker $d_i$ with trigger $t_i$
//$c_i$ represents a channel on which messages from another agent
    may be received
while()
    r = listen($t_1$,...,$t_m$, $c_1$,...,$c_n$)
    if (r is a $t_i$)
        attempts = invoke($d_i$, forms)
        if (check(attempts))
            updateLocalState(attempts)
            emit(attempts)
    else
        m = receive(r)
        if(check(m))
            updateLocalState(m)
```

The Kiko adapter, documentation, and sample code are available at:
https://gitlab.com/masr/bspl/-/tree/kiko/

1.3.4 Mandrake

Mandrake is another protocol-based programming model. Kiko's enablement-based programming is more general and convenient to use than Mandrake's *reactive* programming model (an agent reacts to the reception of a message by sending a message). Where Mandrake goes beyond Kiko is in the handling of delayed (potentially lost) messages, a kind of fault that may arise when using network transports such as UDP. Specifically, Mandrake enables writing agent-level policies for dealing with such faults.

Mandrake is inspired by the end-to-end principle [57]. Although it is today customary to assume a reliable transport layer such as TCP (also part of the Internet Protocol suite), the end-to-end principle advises that lower-layer reliability guarantees are inadequate for building reliable multiagent systems. Instead, the agents must encode the necessary reliability mechanisms.

A simple example highlights the inadequacy of lower-level reliability guarantees. Suppose the SELLER agent has sent *Shipment* but has not received *Payment* even after waiting a considerable amount of time. There are two reasons why the SELLER has not received *Payment*. Either BUYER sent the *Payment* but it was delayed in transit, or BUYER never sent *Payment*. These reasons are indistinguishable to the SELLER.

Whereas TCP can help address network problems via retransmission, it is of no help if an agent never sends a message. Therefore, the SELLER must implement some fault handling logic. For example, as is often the case in real life, it may remind the BUYER about its *Shipment*. Listing 7 shows such a retransmission policy for a SELLER agent. The policy specifically is to send a daily reminder of the *Shipment* to the BUYER until it receives the *Payment* or it has sent five reminders.

Listing 7: A SELLER's reminder policy.

```
- action: remind Buyer of Shipment until Payment
  when: 0 0 * * * // daily
  max tries: 5
```

Notice that the reminders amount to application-level retransmission of messages. As explained above, they are necessary, but once we have them at the application level, lower-level reliability mechanisms, such as those provided by TCP, are obviated, thus leading to potentially improved performance. Mandrake is available at https://gitlab.com/masr/bspl/-/tree/mandrake/

1.3.5 Reflections

The autonomy of entities motivates protocols, which are thus inherent to multiagent systems. There are, of course, multiagent systems that are implemented without *specifying* a protocol. In such cases, the agent programmer implements the protocol in low-level agent code. For anything but the most trivial protocols, such an exercise is bound to be complex and error-prone. By enabling the specification of protocols and implementing protocol-based agents, our tooling addresses this gap.

We started with a simple protocol *Flexible Purchase*; verified it for safety and liveness using Tango, and showed how one might implement agents using Kiko and Mandrake. Keeping in mind the alternative approaches described in Section 1.2, the following points are noteworthy about our approach.

- Our approach is formal, and our tooling enables implementing a verified protocol. That is, the model you verify is the model you implement.
- *Flexible Purchase*, as simple as it is, is not even expressible as a well-formed protocol in alternative protocol specification approaches [20].
- In general, protocol specifications can be much more complex than our examples. The number of possible enactments of a protocol can, in the worst case, grow exponentially with the size of the protocol. Kiko's enablement-based approach abstracts away much of this complexity.
- Our approach does not require message ordering guarantees from the communication infrastructure. Both the actor model [1, 38] and the end-to-end principle [57] argue against such guarantees. Mandrake enables agent programmers to encode policies for handling potential communication failures in a straightforward manner.
- Our approach leads to decentralized multiagent systems—agents communicate with each other via asynchronous messaging. A "centralized" multiagent system would be a special case in which the protocol is such that there is one role such that all communications are between it and the other roles.
- High-level meaning is important. Although not the focus of this contribution, Table 1.1 gives pointers to tools that show how protocols are a substrate that supports meaning. Both are necessary: whereas protocols define the legal communicative moves with respect to causality and integrity, notions such as commitments capture their meaning.

1.4 Summary of Tools for Interaction-Oriented Programming

This paper introduces publicly available software that embodies the principles of Interaction-Oriented Programming (IOP) and should help facilitate the adoption of multiagent systems by developers. The software is centered on the idea of information protocols, a novel approach for modeling flexible multiagent systems. The software enables verifying protocols and implementing protocol-based agents. Table 1.1 summarizes the various tools for engineering multiagent systems based on models of interaction that we have developed and which are still active. We have explained Tango, Kiko, and Mandrake above. We briefly explain the others below.

Table 1.1: Tools for engineering multiagent systems based on interaction: information protocols and commitments. Internals refers to the language the tool is constructed using and Usage refers to the language it outputs or analyzes.

		Language		
Tool	**Main purpose**	Internals	Usage	**Status**
Tango	Verifying protocols	Python	BSPL	Stable
Kiko	Enablement-based imperative agent programming	Python	Python	Stable
Mandrake	Reactive, fault-tolerant agent programming	Python	Python	Stable
Orpheus	Enablement-based cognitive agent programming	Jason	Jason	Beta
Cupid	Compile commitments into database queries	Cupid	SQL	Beta
Azorus	Enablement-based cognitive agent programming with commitments	Jason	Jason	Alpha

Orpheus [7] is a programming model inspired by Kiko but geared toward implementing cognitive agents in Jason. Orpheus provides a tool that generates much of the general-purpose reasoning needed by an agent that is based on the given protocol (and the BSPL semantics). A programmer needs to provide only the business logic comprising the agent's goals (presumably to support the agent's principal's requirements) and the mapping of these goals to the messages the agent needs to send.

Communication meaning is a defining theme in multiagent systems. A superior alternative to the FIPA ACL [33] and KQML [16] is specifying social meaning [64] in terms of commitments and kinds of norms. Cupid is a language for specifying the meaning of communicative acts in terms of commitments [22]. Cupid is accompanied by a compiler that translates commitment specifications into SQL queries over a database of communicative acts.

Azorus [18] extends Orpheus by incorporating support for commitment-based reasoning in Jason. Specifically, Azorus includes a Cupid compiler targeted to Jason. Jason programmers can now write agents that query states of commitments and take protocol-based actions accordingly. Interestingly, though commitment protocols [77, 81] precede information protocols by over a decade, the connection between commitments and information protocols for the purpose of programming multiagent systems was not fleshed out until recently. Azorus begins to fill that gap.

1.5 Artificial Intelligence Context

Our current techniques fall squarely in the so-called Good Old-Fashioned AI (GOFAI) camp in that we seek to model and verify multiagent systems on the basis of requirements and then implement them using middleware-supported programming abstractions.

We seek representations that are intuitive to both stakeholders and programmers. Our whole enterprise is, in fact, motivated by the need for explicitly modeling the high-level meaning of interactions via notions such as commitments and other norms. It is worth keeping in mind that interoperability is an important goal of engineering multiagent systems, and because interoperability between agents depends on them having a common understanding of such meaning, there is no alternative to specifying it explicitly. In this regard, ours is no different from any existing multiagent systems engineering approach. Our approach has room for exploiting semantic descriptions of objects [45] and planning-based agents [47, 76].

The Theory of Mind (ToM) is the notion that agents model other agents as having minds [13, 55]. ToM justifies the use of folk psychological concepts (e.g., beliefs and goals) in AI, e.g., [28, 49, 62].

Although the current paper deemphasizes representing agents in terms of BDI, our other work (discussed above) addresses this theme. There is also an opportunity to combine our approach with learning agents [75, 2]. For example, an agent may learn to act in ways that represent violations of the specified norms but lead to overall societal benefit. We are agnostic on the representation of an agent's decision making except where it interfaces with communications. Those communications must be made based on protocols (for interoperability) and in the light of their specified meaning (ignoring which could result in unexpected sanctions and missed opportunities).

Large Language Models (LLMs) and other generative AI have upended many intellectual fields. LLMs perform comparably to humans in tasks such as question answering and code generation [48], though with well-known limitations in reasoning [78].

The *agentic* methods form "agents" by combining generative AI with information and abilities to sense and act, e.g., via web services [82]. Current frameworks emphasize workflows to execute compositions of agents specified as task graphs [44, 30]. Formulating agent coordination in task graphs sounds attractive, but its shortcomings have been known for decades [27, 72, 3]. Choreographies [53] go beyond workflows in considering multiple loci of action, and they too are limited [65, 67].

A major shortcoming of workflows and choreographies is their rigidity, which limits agent autonomy and responsiveness to exceptions [46]. Thus, their adoption in modern agentic frameworks is particularly unfortunate—the capabilities of agents built with modern AI techniques would be stymied by poor coordination. Thus, a challenge is to develop agents who can reason about and interact based on protocols. LLMs have shown capabilities comparable to humans on some ToM tasks, mostly focused on beliefs [43, 74], whereas humans (even children) understand desires more readily than beliefs [37]. Enhancements to LLMs for collaboration via ToM could help go beyond today's trivial agentic models.

1.6 Blue Skies All Around!

A vision that inspired the field of multiagent systems was that agents would better capture autonomy in distributed systems. A key enabling idea in this direction was that of interaction meaning: If agents could understand the meaning of their interactions, then they could make decisions intelligently. Early approaches emphasize mentalist semantics, e.g., as in KQML and FIPA ACL. Despite their many and well-documented shortcomings (as discussed in Section 1.2), their usage continues within popular multiagent programming frameworks. The work highlighted in this paper highlights a practical basis for intelligent, decentralized decision making without the shortcomings of the mentalist approaches. A direction for the entire engineering multiagent systems (EMAS) community is to reorient their abstractions and tooling to incorporate protocols and norms. Additionally, we need standards for developing multiagent systems that are based on these ideas. We think such standards would galvanize the EMAS community and make its work attractive to practitioners.

Interaction-Oriented Programming is very much an ongoing effort, and we expect to augment and improve the software suite over the coming years. We highlight some specific directions here. An additional Blue Sky discussion is in [19].

Higher-Level Protocol Languages. Although BSPL is foundational and higher-level than alternative protocol languages, it captures individual messages and may be thought of as an assembly language for operational protocols. In particular, it does not distinguish information that is essential to message meaning from information that is purely coordinative. For example, in the context of an alternative purchase protocol, imagine that a BUYER may send either *Accept* or *Reject* in response to an *Offer*. Parameters such as ID, item, price are related to the meaning of *Accept* and *Reject* and thus would feature in both. To make *Accept* and *Reject* mutually exclusive in BSPL, we would need another parameter which would be ⌜out⌝ in both. This parameter is purely coordinative; it has nothing to do with the meanings of *Accept* and *Reject*. The coordination requirements could, of course, be more complex than mutual exclusion, and a protocol designer would have to map those requirements to the elementary notions in BSPL. This motivates protocol languages that are higher-level than BSPL in that they are focused on meaning and enable generating the necessary coordination. Langshaw [71] is a start in that direction. Langshaw protocols are synchronous and specify agent actions and *sayso*s (the capability of generating it) over information. BSPL protocols are compiled out from Langshaw protocols.

Fault Tolerance and scalability. These are both of practical importance but are not particularly well-addressed in the EMAS community. Mandrake requires the specification of fault tolerance policies. Ideally, we should exploit the meaning specifications to automate fault tolerance. For example, commitment specifications could tell an agent when and which communications are important and, therefore, worth retransmitting.

Modern architectural paradigms, such as the cloud, are motivated by scalability. Though there have been efforts linking information protocols and cloud computing paradigms such as serverless computing [24] and microservices [42], research on engineering multiagent systems, including IOP, lacks a story for how to realize highly scalable multiagent systems. The actor model is known for its scalability; developing a synthesis of protocols and actors would be a valuable direction.

Types. A type theory for protocols would further enhance the programming mode by catching errors at compile time and guiding the implementation of agents. For example, we could capture at compile time the error of an agent attempting to send both *Accept* and *Reject* in an enactment. The notion of dependent types (e.g., as in Idris [41]) could be interesting for this purpose.

Methodologies. Developing methodologies for specifying meaning, protocols, and implementing agents will be important. We need to understand how requirements map to the specification of meaning in terms of norms, e.g., in the spirit of Protos [21]. We need to extend Tango to support the verification of a broader class of properties, e.g., encoding the stakeholder requirements on protocols. We need to understand how to map agent requirements into agent code in programming models such as Kiko. Kiko guarantees an agent's compliance with the protocol— statically, in the case of sequential agents. We may also want to verify that the agent's internal logic meets requirements via formal methods and testing.

1.7 Reproducibility

The entire codebase referenced in this paper, as well as other related tools, are available online at https://gitlab.com/masr. Software developed in [42] is available at https://gitlab.com/masr/information-protocols-dapr-emas-2022. Software developed in [24] is available at https://gitlab.com/masr/deserv.

Acknowledgments

Thanks to the NSF (grant IIS-1908374) and EPSRC (grant EP/N027965/1) for support.

References

1. Agha, G.A.: Actors. MIT Press, Cambridge, Massachusetts (1986). https://doi.org/10.7551/mitpress/1086.001.0001
2. Agrawal, R., Ajmeri, N., Singh, M.P.: Socially intelligent genetic agents for the emergence of explicit norms. In: Proceedings of the 31st International Joint Conference on Artificial Intelligence (IJCAI). pp. 10–16. IJCAI, Vienna (Jul 2022). https://doi.org/10.24963/ijcai.2022/2
3. Attie, P.C., Singh, M.P., Sheth, A.P., Rusinkiewicz, M.: Specifying and enforcing intertask dependencies. In: Proceedings of the 19th International Conference on Very Large Data Bases (VLDB). pp. 134–145. Morgan Kaufmann, Dublin (Aug 1993), http://www.vldb.org/conf/1993/P134.PDF
4. Austin, J.L.: How to Do Things with Words. Clarendon Press, Oxford (1962)
5. Baldoni, M., Baroglio, C., Chopra, A.K., Desai, N., Patti, V., Singh, M.P.: Choice, interoperability, and conformance in interaction protocols and service choreographies. In: Proceedings of the 8th International Conference on Autonomous Agents and MultiAgent Systems (AAMAS). pp. 843–850. IFAAMAS, Budapest (May 2009). https://doi.org/10.5555/1558109.1558129
6. Baldoni, M., Baroglio, C., Martelli, A., Patti, V.: A priori conformance verification for guaranteeing interoperability in open environments. In: Proceedings of the 4th International Conference on Service-Oriented Computing (ICSOC). Lecture Notes in Computer Science, vol. 4294, pp. 339–351. Springer, Chicago (Dec 2006). https://doi.org/10.1007/11948148_28
7. Baldoni, M., Christie V, S.H., Singh, M.P., Chopra, A.K.: Orpheus: Engineering multiagent systems via communicating agents. In: Proceedings of the 39th AAAI Conference on Artificial Intelligence (AAAI). pp. 23135–23143. AAAI, Philadelphia (Feb 2025). https://doi.org/10.1609/aaai.v39i22.34478
8. Bellifemine, F., Caire, G., Greenwood, D.: Developing Multi-Agent Systems with JADE. Wiley, Chichester, UK (2007). https://doi.org/10.1002/9780470058411
9. Bergenti, F., Caire, G., Monica, S., Poggi, A.: The first twenty years of agent-based software development with JADE. Journal of Autonomous Agents and Multi-Agent Systems (JAAMAS) **34**(2), 36 (2020). https://doi.org/10.1007/s10458-020-09460-z
10. Boissier, O., Bordini, R.H., Hübner, J.F., Ricci, A., Santi, A.: Multi-agent oriented programming with JaCaMo. Science of Computer Programming **78**(6), 747–761 (Jun 2013). https://doi.org/10.1016/j.scico.2011.10.004
11. Bordini, R.H., Hübner, J.F.: Semantics for the Jason variant of AgentSpeak (plan failure and some internal actions). In: Proceedings of the 19th European Conference on Artificial Intelligence (ECAI). Frontiers in Artificial Intelligence and Applications, vol. 215, pp. 635–640. IOS Press, Lisbon (Aug 2010). https://doi.org/10.3233/978-1-60750-606-5-635
12. Caponetto, L., Labinaz, P. (eds.): Sbisà on Speech as Action. Philosophers in Depth, Palgrave Macmillan, Cham, Switzerland (2023). https://doi.org/10.1007/978-3-031-22528-4
13. Carruthers, P., Smith, P.K.: Introduction. In: Theories of Theories of Mind [14], chap. 1, pp. 1–8. https://doi.org/10.1017/CBO9780511597985
14. Carruthers, P., Smith, P.K. (eds.): Theories of Theories of Mind. Cambridge University Press, Cambridge (1996). https://doi.org/10.1017/CBO9780511597985
15. Cernuzzi, L., Zambonelli, F.: Experiencing AUML in the GAIA methodology. In: Proceedings of the 6th International Conference on Enterprise Information Systems (ICEIS). pp. 283–288. Science and Technology Publications, Lda, Porto, Portugal (Apr 2004). https://doi.org/10.5220/0002618802830288
16. Chalupsky, H., Finin, T., Fritzson, R., McKay, D., Shapiro, S., Wiederhold, G.: An overview of KQML: A knowledge query and manipulation language. TR, University of Maryland Computer Science Department, Baltimore (Apr 1992)

17. Chopra, A.K., Artikis, A., Bentahar, J., Colombetti, M., Dignum, F., Fornara, N., Jones, A.J.I., Singh, M.P., Yolum, P.: Research directions in agent communication. ACM Transactions on Intelligent Systems and Technology (TIST) **42**(2), 20:1–20:23 (Mar 2013). https://doi.org/10.1145/2438653.2438655
18. Chopra, A.K., Baldoni, M., Christie V, S.H., Singh, M.P.: Azorus: Commitments over protocols for BDI agents. In: Proceedings of the 24th International Conference on Autonomous Agents and MultiAgent Systems (AAMAS). pp. 490–499. IFAAMAS, Detroit (May 2025)
19. Chopra, A.K., Christie V, S.H.: Communication meaning: Foundations and directions for systems research. In: Proceedings of the International Conference on Autonomous Agents and Multiagent Systems. pp. 1786–1791. ACM, London (2023)
20. Chopra, A.K., Christie V, S.H., Singh, M.P.: An evaluation of communication protocol languages for engineering multiagent systems. Journal of Artificial Intelligence Research (JAIR) **69**, 1351–1393 (Dec 2020). https://doi.org/10.1613/jair.1.12212
21. Chopra, A.K., Dalpiaz, F., Aydemir, F.B., Giorgini, P., Mylopoulos, J., Singh, M.P.: Protos: Foundations for engineering innovative sociotechnical systems. In: Proceedings of the 22nd IEEE International Requirements Engineering Conference (RE). pp. 53–62. IEEE Computer Society, Karlskrona, Sweden (Aug 2014). https://doi.org/10.1109/RE.2014.6912247
22. Chopra, A.K., Singh, M.P.: Cupid: Commitments in relational algebra. In: Proceedings of the 29th Conference on Artificial Intelligence (AAAI). pp. 2052–2059. AAAI Press, Austin, Texas (Jan 2015). https://doi.org/10.1609/aaai.v29i1.9443
23. Chopra, A.K., Singh, M.P.: Custard: Computing norm states over information stores. In: Proceedings of the 15th International Conference on Autonomous Agents and MultiAgent Systems (AAMAS). pp. 1096–1105. IFAAMAS, Singapore (May 2016). https://doi.org/10.5555/2936924.2937085
24. Christie V, S.H., Chopra, A.K., Singh, M.P.: Deserv: Decentralized serverless computing. In: Proceedings of the 19th IEEE International Conference on Web Services (ICWS). pp. 51–60. IEEE Computer Society, Virtual (Sep 2021). https://doi.org/10.1109/ICWS53863.2021.00020
25. Christie V, S.H., Chopra, A.K., Singh, M.P.: Mandrake: Multiagent systems as a basis for programming fault-tolerant decentralized applications. Journal of Autonomous Agents and Multi-Agent Systems (JAAMAS) **36**(1), 16:1–16:30 (Apr 2022). https://doi.org/10.1007/s10458-021-09540-8
26. Christie V, S.H., Singh, M.P., Chopra, A.K.: Kiko: Programming agents to enact interaction protocols. In: Proceedings of the 22nd International Conference on Autonomous Agents and MultiAgent Systems (AAMAS). pp. 1154–1163. IFAAMAS, London (May 2023). https://doi.org/10.5555/3545946.3598758
27. Chrysanthis, P.K., Ramamritham, K.: Synthesis of extended transaction models using ACTA. ACM Transactions on Database Systems **19**(3), 450–491 (Sep 1994). https://doi.org/10.1145/185827.185843
28. Dennett, D.C.: The Intentional Stance. MIT Press, Cambridge, Massachusetts (1987)
29. Desai, N., Mallya, A.U., Chopra, A.K., Singh, M.P.: OWL-P: A methodology for business process development. In: Proceedings of the 7th International Bi-Conference on Agent-Oriented Information Systems (AOIS). Lecture Notes in Computer Science, vol. 3529, pp. 79–94. Springer, Utrecht, The Netherlands (Jul 2005). https://doi.org/10.1007/11916291_6
30. Dibia, V., Chen, J., Bansal, G., Syed, S., Fourney, A., Zhu, E., Wang, C., Amershi, S.: AutoGen Studio: A no-code developer tool for building and debugging multi-agent systems. CoRR **abs/2408.15247** (Aug 2024). https://doi.org/10.48550/ARXIV.2408.15247
31. Ferrando, A., Winikoff, M., Cranefield, S., Dignum, F., Mascardi, V.: On enactability of agent interaction protocols: Towards a unified approach. In: Proceedings of the 7th International Workshop on Engineering Multi-Agent Systems (EMAS). Lecture

Notes in Computer Science, vol. 12058, pp. 43–64. Springer, Montréal (May 2019). https://doi.org/10.1007/978-3-030-51417-4_3
32. Finin, T., Fritzson, R., McKay, D., McEntire, R.: KQML as an agent communication language. In: Proceedings of the 3rd International Conference on Information and Knowledge Management. pp. 456–463. ACM Press, Gaithersburg, Maryland (Dec 1994). https://doi.org/10.1145/191246.191322
33. FIPA: FIPA agent communication language specifications. http://www.fipa.org/repository/aclspecs.html (2002), FIPA: The Foundation for Intelligent Physical Agents. Accessed 2025-01-20
34. FIPA: FIPA interaction protocol specifications. http://www.fipa.org/repository/ips.html (2003), FIPA: The Foundation for Intelligent Physical Agents. Accessed 2024-11-24
35. Galland, S., Rodriguez, S., Gaud, N.: Run-time environment for the SARL agent-programming language: The example of the Janus platform. Future Generation Computer Systems **107**, 1105–1115 (Jun 2020). https://doi.org/10.1016/j.future.2017.10.020
36. Gasser, L.: Social conceptions of knowledge and action: DAI foundations and open systems semantics. Artificial Intelligence **47**(1–3), 107–138 (Jan 1991). https://doi.org/10.1016/0004-3702(91)90052-L
37. Harris, P.: Desires, beliefs, and language. In: Carruthers and Smith [14], chap. 13, pp. 200–220. https://doi.org/10.1017/CBO9780511597985
38. Hewitt, C.: Viewing control structures as patterns of passing messages. Artificial Intelligence **8**(3), 323–364 (Jun 1977). https://doi.org/10.1016/0004-3702(77)90033-9
39. Hewitt, C.: Open information systems semantics for distributed artificial intelligence. Artificial Intelligence **47**(1–3), 79–106 (Jan 1991). https://doi.org/10.1016/0004-3702(91)90051-K
40. Huget, M.P., Odell, J.: Representing agent interaction protocols with agent UML. In: Proceedings of the 5th International Workshop on Agent-Oriented Software Engineering (AOSE). Lecture Notes in Computer Science, vol. 3382, pp. 16–30. Springer, New York (Jul 2004). https://doi.org/10.1007/978-3-540-30578-1_2
41. Idris: Idris: A language for type-driven development (Mar 2025), www.idris-lang.org, accessed 2025-03-05
42. Khadse, A.K., Christie V, S.H., Chopra, A.K., Singh, M.P.: Protocol-based engineering of microservices. In: Proceedings of the 11th International Workshop on Engineering Multi-Agent Systems (EMAS). pp. 61–77. No. 14378 in Lecture Notes in Artificial Intelligence, London (May 2023). https://doi.org/10.1007/978-3-031-48539-8_4
43. Kosinski, M.: Evaluating large language models in theory of mind tasks. Proceedings of the National Academy of Sciences **121**(45), e2405460121 (Oct 2024). https://doi.org/10.1073/pnas.2405460121
44. LangGraph: LangGraph: Building language agents as graphs (Dec 2024), https://langchain-ai.github.io/langgraph/, accessed 2024-12-05
45. Lemée, J., Vachtsevanou, D., Mayer, S., Ciortea, A.: Signifiers for conveying and exploiting affordances: From human-computer interaction to multi-agent systems. Annals of Mathematics and Artificial Intelligence **92**(4), 815–835 (2024)
46. Lichtenstein, T., Chopra, A.K., Singh, M.P., Weske, M.: From visual choreographies to flexible information protocols. In: Proceedings of the 22nd International Conference on Service-Oriented Computing (ICSOC). pp. 354–369. No. 15404 in Lecture Notes in Computer Science, Springer, Tunis, Tunisia (Dec 2024). https://doi.org/10.1007/978-981-96-0805-8_25
47. Meneguzzi, F., Telang, P.R., Singh, M.P.: A first-order formalization of commitments and goals for planning. In: Proceedings of the 27th Conference on Artificial Intelligence (AAAI). pp. 697–703. AAAI Press, Bellevue, Washington (Jul 2013). https://doi.org/10.1609/aaai.v27i1.8632

48. Mozannar, H., Bansal, G., Fourney, A., Horvitz, E.: When to show a suggestion? Integrating human feedback in AI-assisted programming. In: Proceedings of the 38th AAAI Conference on Artificial Intelligence (AAAI). pp. 10137–10144. AAAI Press, Vancouver (Feb 2024). https://doi.org/10.1609/AAAI.V38I9.28878
49. Newell, A.: The knowledge level. Artificial Intelligence **18**(1), 87–127 (Jan 1982). https://doi.org/10.1016/0004-3702(82)90012-1
50. Odell, J., Parunak, H.V.D., Bauer, B.: Representing agent interaction protocols in UML. In: Proceedings of the 1st International Workshop on Agent-Oriented Software Engineering (AOSE 2000). Lecture Notes in Computer Science, vol. 1957, pp. 121–140. Springer, Toronto (Jun 2001). https://doi.org/10.1007/3-540-44564-1_8
51. Padgham, L., Winikoff, M.: Prometheus: A practical agent-oriented methodology. In: Henderson-Sellers, B., Giorgini, P. (eds.) Agent-Oriented Methodologies, chap. 5, pp. 107–135. Idea Group, Hershey, Pennsylvania (2005). https://doi.org/10.4018/978-1-59140-581-8.ch005
52. Parnas, D.L.: Information distribution aspects of design methodology. In: Proceedings of the International Federation for Information Processing Congress. vol. TA-3, pp. 26–30. North Holland, Amsterdam (1971)
53. Peltz, C.: Web service orchestration and choreography. IEEE Computer **36**(10), 46–52 (Oct 2003). https://doi.org/10.1109/MC.2003.1236471
54. Poslad, S.: Specifying protocols for multi-agent systems interaction. ACM Transactions on Autonomous and Adaptive Systems (TAAS) **2**(4), 15:1–15:24 (Nov 2007). https://doi.org/10.1145/1293731.1293735
55. Premack, D., Woodruff, G.: Does the chimpanzee have a theory of mind? Behavioral and Brain Sciences **1**(4), 515–526 (Dec 1978). https://doi.org/10.1017/S0140525X00076512
56. Rooney, C., Collier, R.W., O'Hare, G.M.P.: VIPER: A VIsual protocol editoR. In: Proceedings of the 6th International Conference on Coordination Models and Languages COORDINATION. Lecture Notes in Computer Science, vol. 2949, pp. 279–293. Springer, Pisa (Feb 2004). https://doi.org/10.1007/978-3-540-24634-3_21
57. Saltzer, J.H., Reed, D.P., Clark, D.D.: End-to-end arguments in system design. ACM Transactions on Computer Systems **2**(4), 277–288 (Nov 1984). https://doi.org/10.1145/357401.357402
58. Sbisà, M.: How to read Austin. Pragmatics **17**(3), 461–473 (Sep 2007). https://doi.org/10.1075/prag.17.3.06sbi
59. Sbisà, M.: Varieties of speech act norms. In: Witek, M., Witczak-Plisiecka, I. (eds.) Normativity and Variety of Speech Actions, Poznań Studies in the Philosophy of the Sciences and the Humanities, vol. 112, pp. 23–50. Brill Rodopi, Leiden, Netherlands (2018). https://doi.org/10.1163/9789004366527_003
60. Searle, J.R.: Speech Acts: An Essay in the Philosophy of Language. Cambridge University Press, Cambridge, United Kingdom (1969). https://doi.org/10.1017/CBO9781139173438
61. Shaw, M., Garlan, D.: Software Architecture: Perspectives on an Emerging Discipline. Prentice-Hall, Upper Saddle River, New Jersey (1996)
62. Singh, M.P.: Multiagent Systems: A Theoretical Framework for Intentions, Know-How, and Communications. No. 799 in Lecture Notes in Computer Science, Springer, Heidelberg (1994). https://doi.org/10.1007/BFb0030531, http://www.csc.ncsu.edu/faculty/mpsingh/books/MAS/
63. Singh, M.P.: Agent communication languages: Rethinking the principles. IEEE Computer **31**(12), 40–47 (Dec 1998)
64. Singh, M.P.: A social semantics for agent communication languages. In: Proceedings of the 1999 IJCAI Workshop on Agent Communication Languages. pp. 31–45. No. 1916 in Lecture Notes in Artificial Intelligence, Springer, Berlin (2000). https://doi.org/10.1007/10722777_3
65. Singh, M.P.: Information-driven interaction-oriented programming: BSPL, the Blindingly Simple Protocol Language. In: Proceedings of the 10th International Con-

ference on Autonomous Agents and MultiAgent Systems (AAMAS). pp. 491–498. IFAAMAS, Taipei (May 2011). https://doi.org/10.5555/2031678.2031687

66. Singh, M.P.: LoST: Local State Transfer—An architectural style for the distributed enactment of business protocols. In: Proceedings of the 9th IEEE International Conference on Web Services (ICWS). pp. 57–64. IEEE Computer Society, Washington, DC (Jul 2011). https://doi.org/10.1109/ICWS.2011.48
67. Singh, M.P.: Semantics and verification of information-based protocols. In: Proceedings of the 11th International Conference on Autonomous Agents and MultiAgent Systems (AAMAS). pp. 1149–1156. IFAAMAS, Valencia, Spain (Jun 2012). https://doi.org/10.5555/2343776.2343861
68. Singh, M.P.: Bliss: Specifying declarative service protocols. In: Proceedings of the 11th IEEE International Conference on Services Computing (SCC). pp. 235–242. IEEE Computer Society, Anchorage, Alaska (Jun 2014). https://doi.org/10.1109/SCC.2014.39
69. Singh, M.P., Chopra, A.K.: Programming multiagent systems without programming agents. In: Proceedings of the 7th International Workshop on Programming Multiagent Systems (ProMAS 2009). pp. 1–14. No. 5919 in Lecture Notes in Artificial Intelligence, Springer (May 2010). https://doi.org/10.1007/978-3-642-14843-9_1, invited paper
70. Singh, M.P., Christie V, S.H.: Tango: Declarative semantics for multiagent communication protocols. In: Proceedings of the 30th International Joint Conference on Artificial Intelligence (IJCAI). pp. 391–397. IJCAI, Online (Aug 2021). https://doi.org/10.24963/ijcai.2021/55
71. Singh, M.P., Christie V, S.H., Chopra, A.K.: Langshaw: Declarative interaction protocols based on sayso and conflict. In: Proceedings of the 30th International Joint Conference on Artificial Intelligence (IJCAI). pp. 202–210. IJCAI, Jeju, Korea (Aug 2024). https://doi.org/10.24963/ijcai.2024/23
72. Singh, M.P., Huhns, M.N.: Automating workflows for service order processing: Integrating AI and database technologies. IEEE Expert **9**(5), 19–23 (Oct 1994). https://doi.org/10.1109/64.331480
73. Spanoudakis, N.I., Moraitis, P.: The ASEME methodology. International Journal of Agent-Oriented Software Engineering **7**(2), 79–107 (May 2022). https://doi.org/10.1504/IJAOSE.2022.122600
74. Strachan, J.W.A., Albergo, D., Borghini, G., Pansardi, O., Scaliti, E., Gupta, S., Saxena, K., Rufo, A., Panzeri, S., Manzi, G., Graziano, M.S.A., Becchio, C.: Testing theory of mind in large language models and humans. Nature Human Behaviour **8**, 1285–1295 (Jul 2024). https://doi.org/10.1038/s41562-024-01882-z
75. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. Adaptive Computation and Machine Learning, MIT Press, Cambridge, Massachusetts, 2nd edition edn. (2018)
76. Telang, P.R., Meneguzzi, F., Singh, M.P.: Hierarchical planning about goals and commitments. In: Proceedings of the 12th International Conference on Autonomous Agents and MultiAgent Systems (AAMAS). pp. 877–884. IFAAMAS, St. Paul, Minnesota (May 2013). https://doi.org/10.5555/2484920.2485059
77. Venkatraman, M., Singh, M.P.: Verifying compliance with commitment protocols: Enabling open Web-based multiagent systems. Journal of Autonomous Agents and Multi-Agent Systems (JAAMAS) **2**(3), 217–236 (Sep 1999). https://doi.org/10.1023/A:1010056221226
78. Verma, M., Bhambri, S., Kambhampati, S.: On the brittle foundations of ReAct prompting for agentic large language models. CoRR **abs/2405.13966** (May 2024). https://doi.org/10.48550/ARXIV.2405.13966
79. Winikoff, M.: Implementing commitment-based interactions. In: Proceedings of the 6th International Joint Conference on Autonomous Agents and MultiAgent Systems (AAMAS). pp. 868–875. IFAAMAS, Honolulu (May 2007). https://doi.org/10.1145/1329125.1329283

80. Winikoff, M., Yadav, N., Padgham, L.: A new Hierarchical Agent Protocol Notation. Journal of Autonomous Agents and Multi-Agent Systems (JAAMAS) **32**(1), 59–133 (Jan 2018). https://doi.org/10.1007/s10458-017-9373-9
81. Yolum, P., Singh, M.P.: Commitment machines. In: Proceedings of the 8th International Workshop on Agent Theories, Architectures, and Languages (ATAL 2001). pp. 235–247. No. 2333 in Lecture Notes in Artificial Intelligence, Springer, Seattle (2002). https://doi.org/10.1007/3-540-45448-9_17
82. Zhang, J., Lan, T., Zhu, M., Liu, Z., Hoang, T., Kokane, S., Yao, W., Tan, J., Prabhakar, A., Chen, H., Liu, Z., Feng, Y., Awalgaonkar, T.M., Murthy, R., Hu, E., Chen, Z., Xu, R., Niebles, J.C., Heinecke, S., Wang, H., Savarese, S., Xiong, C.: xLAM: A family of large action models to empower AI agent systems. CoRR **abs/2409.03215** (Sep 2024). https://doi.org/10.48550/ARXIV.2409.03215