MULTIAGENT SYSTEMS

MIT Press, 2011

Contents

List of Figures

List of Tables

Chapter 3

Agent Communication

Amit K. Chopra and Munindar P. Singh

1 Introduction

Multiagent systems are distributed systems. Engineering a multiagent system means rigorously specifying the communications among the agents by way of interaction protocols. What makes specifying the protocols for agent interaction especially interesting and challenging is that agents are *autonomous* and *heterogeneous* entities. These properties of agents have profound implications on the nature of protocol specifications. As we shall see, protocols for multiagent systems turn out to be fundamentally different from those for other kinds of distributed systems such as computer networks and distributed databases.

We conceptualize all distributed systems in architectural terms—as consisting of components and connectors between the components. The components of the Internet are all nodes with IP addresses. The main connector is the Internet Protocol, which routes packets between the nodes. The components of the Web are the clients (such as browsers) and servers and the connector is the HTTP protocol. The components in a distributed database are the client databases and the coordinator and a connector is the two-phase commit protocol. We can discern a pattern here: the connectors are nothing but the interaction protocols among the components. Further, we can associate protocols with the application it facilitates. For example, the Internet Protocol facilitates routing; HTTP facilitates access to

a distributed database of resources; and the two-phase commit protocol facilitates distributed transactions.

The same applies for multiagent systems except that the components are autonomous and heterogeneous agents, and applications are typically higher-level—for example, auctions, banking, shipping, and so on. Each application would have its own set of requirements and therefore we would normally find different protocols for each application. Below, the term *traditional distributed systems* refers to non-multiagent distributed systems such as the Internet, the Web, and so on.

The importance of protocols is not lost upon industry. Communities of practice are increasingly interested in specifying standard protocols for their respective domains. RosettaNet [40] (e-business), TWIST [53] (foreign exchange transactions), GDSN [33] (supply chains), and HITSP [34] and HL7 [31] (health care) are just a few examples.

Our objectives in this chapter are to help the reader develop a clear sense of the conceptual underpinnings of agent communication and to help the reader learn to apply the concepts to the extent possible using available software. The chapter is broadly structured according to the following subobjectives.

Requirements for protocol specifications The inherently open nature of multiagent systems places certain requirements on protocol specifications. Meeting these requirements is the key to designing good protocols.

Protocol specification approaches There are many diverse approaches for specifying protocols. We evaluate some approaches widely practiced in software engineering and some historically significant ones from artificial intelligence. We also study an approach that is particularly promising.

Directions in agent communication research The last fifteen years have seen some exciting developments in agent communication. However, many practical concerns remain to be addressed. We discuss these briefly.

1.1 Autonomy and its Implications

Protocols are modular, potentially reusable specifications of interactions between two or more components. The interactions are specified in terms of the *messages* the components exchange. To promote reusability, a protocol is specified abstractly with reference to the *roles* that the interacting components may adopt. A protocol is designed with a certain *application* in mind. An *enactment* refers to an execution of the protocol by the components.

In distributed systems, the chief concern is *how can distributed components work together effectively?* In other words, how can we ensure their *interoperation*? In engineering terms, protocols are the key to interoperation. The idea is that as long as components are individually *conformant*, that is, follow their respective roles in the protocol, they will be able to work together no matter how they are implemented. Interoperation makes great engineering sense because it means that the components are loosely coupled with each other; that is, we can potentially replace a component by another conformant one and the modified system would continue to function. You would have noticed that Web browsers and servers often advertise the versions of the HTTP standard with which they are conformant.

The same concepts and concerns apply to multiagent systems. However, agents are not ordinary components. They are components that are autonomous and heterogeneous. Below, we discuss exactly what we mean by these terms, and how autonomy and heterogeneity naturally lead to requirements for agent interaction protocols that go beyond protocols for traditional distributed systems.

Each agent is an autonomous entity in the sense that it itself is a domain of control: other agents have no direct control over its actions (including its communications). For instance, consider online auctions as they are conducted on web sites such as eBay. Sellers, bidders, and auctioneers are all agents, and none of them exercises any control over the others. If an auctioneer had control over bidders, then (if it chose to) it could force any of the bidders to bid any amount by simply invoking the appropriate method. Such a setting would lack any resemblance to real life.

There is a subtle tension between the idea of a protocol and autonomy. With protocols, we seek to somehow constrain the interaction among agents so that they would be interoperable. Autonomy means that the agents are free to interact as they please (more precisely, each agent acts accordingly to the rationale of its principal). From this observations follows our first requirement. *We must design protocols so that they do not overconstrain an agent's interactions.*

In traditional distributed systems, interoperation is achieved via low-level coordination. The protocols there would specify the flow of messages between the participants. In the case of the two-phase commit protocol, the controller coordinates the commit outcome of a distributed transaction. In the first phase, a controller component collects votes from individual databases about whether they are each ready to commit their respective subtransactions. If they unanimously respond positively, the controller, in the second phase, instructs each to commit its respective subtransaction; otherwise, it instructs each to abort its subtransaction.

The above discussion of autonomy implies the following.

The irrelevance of intelligence Contrast the notion of agent autonomy discussed above with the one where autonomy is interpreted as the ability of an agent to perform high-level reasoning (intelligent agents) or as the degree to which an agent can operate without the supervision of its principal (autonomic agents). Consider that you want to automate your purchases on the Web. On the one hand, you can design a simple bidding agent that takes input from you about the things you want, the maximum prices you are willing to pay, and the reputation thresholds of the sellers and auctioneers you are willing to deal with. On the other hand, you can design a sophisticated bidding agent that mines your communications to discover the items you desire and what you are willing to pay for them and can figure out on its own which auctions to bid in on your behalf. From the agent communication perspective, however, the latter's sophistication does not matter—they are both autonomous agents.

Logical versus physical distribution Because of their autonomy, agents are the logical units of distribution: they can neither be aggregated nor decomposed into processes. Whenever an application involves two or more agents, there simply is no recourse but to consider their interactions. Constructs such as processes, by contrast, are physical units of distribution. The choice of whether an application is implemented as a single process or multiple ones is often driven by physical considerations such as geographical distribution, throughput, redundancy, number of available processors and cores, and so on. An agent itself may be implemented via multiple physical units of distribution; that choice, however, is immaterial from a multiagent systems perspective.

Heterogeneity refers to the diversity of agent implementations. The software engineering approach for accommodating heterogeneity is to make public the interdependencies among the components. A component can then be implemented based on what it depends on other components for (what it assumes) and what others depend on it for (what it guarantees) without concern for how the others are implemented. The same approach applies to agents. The specification of the interdependencies is essentially a protocol.

In traditional distributed systems, to accommodate heterogeneity, it is enough that protocols specify the schemas of the messages exchanged as well as their legal flows, that is, their ordering and occurrence. However, such a specification is inadequate for multiagent systems, wherein accommodating heterogeneity entails

also specifying the semantics of the interaction. As an example, consider the finite state machine in Figure 3.1. It specifies the part of a purchase protocol that deals with making offers. This protocol involves two roles: buyer (b) and seller (s). The transitions are labeled with the messages. First, the seller sends an offer to the buyer. The buyer may then accept or reject the offer. After the buyer accepts, the seller may send an updated offer.

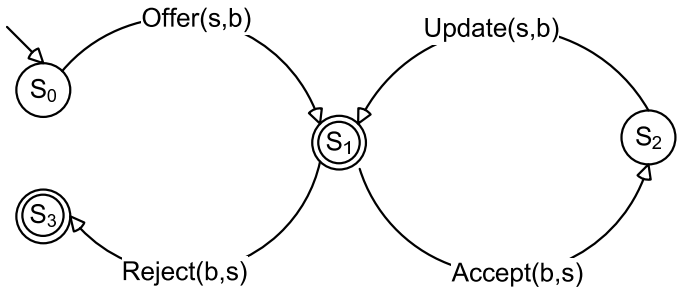

Figure 3.1: Updating an offer.

There is, however, an important element of the specification that is missing from this protocol. That element is what the messages mean in the real world. Making an offer in many settings would count as making a public, in other words, *social commitment* (more on social commitments later). Thus when the seller offers some book to the buyer for some price, it would mean that he is socially committed to the buyer for the offer. Consequently, updating an offer, for instance, by raising the price of the book, counts as updating the commitment. Specifically, it means that the old commitment is canceled and in its place a new one is created. Clearly, a protocol that specifies only the flow of messages, such as the one in Figure 3.1 does not capture such subtleties of meaning.

If the meanings of messages are not public, that would potentially make the agent noninteroperable. For example, this would happen if the buyer interprets the seller's offer as a commitment, but the seller does not. Their interaction would potentially break down. Accommodating semantic heterogeneity presupposes that we make the meanings of messages public as part of the protocol specification.

In practice, many multiagent protocols are specified as flows without reference to the message meanings. And, they seem to work fairly well. In such cases, the designers of the agents agree offline on how to interpret and process the messages

and build this interpretation into the agents, thereby tightly coupling the agents.

1.2 Criteria for Evaluation

Communication has been studied in software engineering, distributed systems, and distributed artificial intelligence. Consequently, there are many approaches for specifying protocols. Later in the chapter, we discuss the major classes of approaches. Let us now motivate broad criteria by which to evaluate each approach.

Software engineering Ideally, protocols should be specified in terms of high-level abstractions that appeal to their stakeholders. In other words, protocol specifications should not be far removed from the expression of stakeholder requirements. Protocol specifications should be modifiable, easily understandable, and composable. Further, they should promote loose coupling among agents.

Flexibility Agents should be able to enact protocols flexibly. Flexibility is especially important in dynamic settings where agents may come and go, and exceptions and opportunities may arise. Ideally, protocol specifications should constrain agents no more than is necessary to ensure correctness, where correctness is understood in connection with the application domain of interest.

Compliance checking An important standard of correctness is compliance. Checking an agent's compliance with a protocol means determining if the agent is following the protocol. To make such a determination presupposes both that a protocol be precise and that its standard of correctness be based on information that is accessible to the agents involved.

2 Conceptual Foundations of Communication in MAS

2.1 Communicative Acts

An important theme in the study of communication is *speech act theory*, better called *communicative act theory*, since it has little specific connection with spoken communication. The main insight behind communicative act theory, due to the philosopher of language, John Austin, is that communication is a form of action. Specifically, we can think of communicative acts as those where "saying makes it

so." For example, when a judge declares a couple married, the judge is not merely reporting on some privately or publicly known fact; instead, the judge is bringing the fact into existence. The same may be said for a soccer umpire who ejects a player from the game. The umpire is not merely stating that the player is not allowed on the field for the duration of the game, the umpire is causing the player's permission to enter the field during the current game to be withdrawn. The judge and the umpire rely upon lower level means to carry out the communicative acts. The judge may merely speak in public or sign a marriage certificate and affix his seal on it. The umpire may flash a red card at the player and speak out the player's jersey number. The physical means exist and information is transferred but what makes the communication a true communication is the convention in place in the given setting. Informally, we can think of the judge as saying "I declare this couple man and wife" and the umpire as saying "I declare this player as ejected from the game."

Austin argued that all communications could be phrased in the above declarative form through the use of appropriate *performative* verbs. Thus a simple *informative* such as "the shipment will arrive on Wednesday" can be treated as if it were "I inform you that the shipment will arrive on Wednesday." A *directive* such as "send me the goods" can be treated as if it were "I request that you send me the goods" or "I demand that you send me the goods." or other such variations. A *commissive* such as "I'll pay you $5" can be treated as if it were "I promise that I'll pay you $5."

The above stylized construction has an important ramification for us as students of multiagent systems. It emphasizes that although what is being informed, requested, or promised may or may not be within the control of the informer, requester, or promiser, the fact that the agent chooses to inform, request, or promise another agent is entirely within its control. The above construction thus coheres with our multiagent systems thinking about autonomy and reflects the essence of the autonomous nature of communication as we explained above.

The above stylized construction has another more practical and arguably more nefarious ramification. Specifically, this is the idea that we can use the performative verb in the above to identify the main purpose or *illocutionary point* of a communication separately from the propositional content of the communication. The underlying intuition is that the same propositional content could be coupled with different illocutionary points to instantiate distinct communicative acts. In computer science terms, the illocutionary points map to message types, and may be thought of as being the value of a message header. Following the shipment example above, we would associate the proposition "the shipment will

arrive on Wednesday" with different message types, for example, *inform*, *request*, and *query*.

2.2 Agent Communication Primitives

As a result of the naturalness of the above mapping from illocutionary points to message types, it has been customary in agent communication languages to specify a small number of specialized message types as primitives. Having message types appears reasonable, but a pitfall lurks in this thinking. Because the literature describes a few broad-brush illocutionary points, existing approaches reflect the assumption that only a small number of primitives is adequate. They account for the meaning of each of these primitives. The above assumption proves erroneous because the applications of multiagent systems are manifold. In each application, the meanings that we need can be potentially distinct from the others. Thus the official meaning supplied by the agent communication language is insufficient, and developers end up adopting additional ad hoc meanings, which they hard-code into their agents. As a result, the agents become tightly coupled with each other. Such coupling makes it difficult to change a multiagent system dynamically, by swapping out one agent for another as it were. Thus the potential benefit of using an agent communication language is lost.

In response to the above challenges, the newer approaches dispense with a fixed set of primitives based on illocutionary points. Instead, they provide an underlying set of abstractions that can be used to provide a formal semantics for any domain-specific primitives that a multiagent system may need. In other words, each domain is different but there is an underlying logic-based representation in which the meanings of the terms used in the domain may be expressed.

For business applications, today, commitments are the key abstractions employed in the underlying representation. For example, in the stock-trading domain, we would see primitives such as *request stock quote* and *provide stock quote*. And, in the electronic commerce domain, we would see primitives such as *quote price*, *quote delivery charges*, and so on. The semantics of the primitives would be expressed in commitments. Notice that even apparently similar primitives may end up with completely different meanings, reflecting the needs and practices of the applicable domains. For example, in typical practice, a price quote is an offer to sell, meaning that the seller becomes committed to providing the specified item at the quoted price. In contrast, in typical practice, a stock quote carries no such connotation of an offer to sell—all it means is that the quoted price is the price at which the previous transaction was completed on the specified stock

symbol, not that the brokerage who provided the quote is offering to sell you the stock for the quoted price. As you can well imagine, the meanings can easily be made more subtle and involved to capture the nuances of practical application scenarios.

Therefore, in a nutshell, it appears misguided to have a few (about a dozen or so) primitives with their unique definitions, hoping that they would cover all practical variations. For the above reason, we suggest that you read the literature on the primitives motivated from the illocutionary points, merely as showing illustrative examples—possibly even as important patterns but definitely not as an adequate basis for building a multiagent system for an arbitrary application.

3 Traditional Software Engineering Approaches

We referred above to low-level distributed computing protocols as a way to explain architectures in general. We argued that we need to consider multiagent systems and high-level protocols as a way to specify architectures that yield interoperability at a level closer to application needs. However, traditional software engineering arguably addresses the challenges of interoperability too. Would it be possible to adopt software engineering techniques as a basis for dealing with agent communication?

The above view has received a significant amount of attention in the literature. Partly because of the apparent simplicity of traditional techniques and largely because of their familiarity to researchers and practitioners alike, the traditional techniques continue to garner much interest in the agents community.

The traditional techniques leave the formulation of the message syntax open—a message could be any document and in common practice is an XML document. And, they disregard the application meaning of the messages involved. Instead, these techniques focus on the operational details of communication, mostly concentrating on the occurrence and ordering of messages.

Thus a protocol may be specified in terms of a finite state machine that describes its states and legal transitions from a centralized perspective. Formally, this may be done in a variety of ways, including state machines [58, 8], Petri Nets [18], statecharts [24], UML sequence diagrams [35], process algebras such as the pi-calculus [9], and logic-based or declarative approaches [47, 54]. All of these approaches specify a set of message occurrences and orderings that are deemed to capture the protocol being specified. We discuss a few of these below.

The above-mentioned traditional representations have the advantage of there

being a number of formal tools for verifying and even validating specifications written in those representations. Thus a protocol designer would be able to determine if a protocol in question would satisfy useful properties such as termination. Implementing the endpoints or agents to satisfy such specifications is generally quite straightforward. Checking compliance with the specification is also conceptually straightforward. As long as the messages observed respect the ordering and occurrence constraints given by a protocol, the enactment is correct with respect to the protocol; otherwise, an enactment is not correct.

However, the value of such tools is diminished by the fact that in the traditional representations there is no clear way to describe the meanings of the interactions. In other words, these approaches lack an independent application-centric standard of correctness. For example, let us suppose that a protocol happens to specify that a merchant ships the goods to the customer and then the customer pays. Here, if the customer happens to pay first, that would be a violation of the protocol. In informal terms, we should not care. It should be the customer's internal decision whether to pay first. If the customer does (taking the risk of paying first or losing bank interest on the money paid), that is the customer's prerogative. However, given the traditional, operational specification, any such deviation from the stated protocol is equally unacceptable. Notice that it may in fact be in the customer's interest to pay first, for example, to include the expense in the current year's tax deductions. But we have no way of knowing that.

Instead, if the protocol could be specified in terms of the meanings of the communications involved, we would naturally express the intuition that all we expect is that the customer eventually pays or that the customer pays no later than some other crucial event. If the customer fails to pay, that would be a violation. But if the customer pays early, so much the better.

3.1 Choreographies

The service-oriented computing literature includes studies of the notion of a *choreography*. A choreography is a specification of the message flow among the participants. Typically, a choreography is specified in terms of *roles* rather than the participants themselves. Involving roles promotes reusability of the choreography specification. Participants *adopt* roles, that is, bind to the roles, in the choreography.

A choreography is a description of an interaction from a shared or, more properly, a *neutral* perspective. In this manner, a choreography is distinguished from a specification of a *workflow*, wherein one party drives all of the other parties. The

latter approach is called an *orchestration* in the services literature.

An advantage of adopting a neutral perspective, as in a choreography, is that it better applies in settings where the participants retain their autonomy: thus it is important to state what each might expect from the others and what each might offer to the others. Doing so promotes loose coupling of the components: centralized approaches could in principle be equally loosely coupled but there is a tendency associated with the power wielded by the central party to make the other partners fit its mold. Also, the existence of the central party and the resulting regimentation of interactions leads to implicit dependencies and thus tight coupling among the parties.

A neutral perspective yields a further advantage that the overall computation becomes naturally distributed and a single party is not involved in mediating all information flows. A choreography is thus a way of specifying and building distributed systems that among the conventional approaches most closely agrees with the multiagent systems way of thinking. But important distinctions remain, which we discuss below.

WS-CDL [57] and ebBP [25] are the leading industry supported choreography standardization efforts. WS-CDL specifies choreographies as message exchanges among partners. WS-CDL is based on the pi-calculus, so it has a formal operational semantics. However, WS-CDL does not satisfy important criteria for an agent communication formalism. First, WS-CDL lacks a theory of the meanings of the message exchanges. Second, when two or more messages are performed within a given WS-CDL choreography, they are handled sequentially by default, as in an MSC. Third, WS-CDL places into a choreography actions that would be private to an agent, such as what it should do upon receiving a message. Fourth, for nested choreographies, WS-CDL relies upon local decision-making by an agent, such as whether to forward a request received in one choreography to another [50].

3.2 Sequence Diagrams

The most natural way to specify a protocol is through a message sequence chart (MSC), formalized as part of UML as Sequence Diagrams [28]. The roles of a protocol correspond to the lifelines of an MSC; each edge connecting two lifelines indicates a message from a sender to a receiver. Time flows downward by convention and the ordering of the messages is apparent from the chart. MSCs support primitives for grouping messages into blocks. Additional primitives include alternatives, parallel blocks, or iterative blocks. Although we do not use

MSCs extensively, they provide a simple way to specify agent communication protocols.

FIPA (the Foundation of Intelligent Physical Agents) is a standards body, now part of the IEEE Computer Society, that has formulated agent communication standards. FIPA defines a number of interaction protocols. These protocols involve messages of the standard types in FIPA. Each FIPA protocol specifies the possible ordering and occurrence constraints on messages as a UML Sequence Diagram supplemented with some informal documentation.

Figure 3.2 shows the FIPA Request Interaction Protocol in FIPA's variant of the UML Sequence Diagram notation [26]. This protocol involves two roles, an INITIATOR and a PARTICIPANT. The INITIATOR sends a *request* to the PARTICIPANT, who either responds with a *refuse* or an *agree*. In the latter case, it follows up with a detailed response, which could be a *failure*, an *inform-done*, or an *inform-result*. The PARTICIPANT may omit the *agree* message unless the INITIATOR asked for a notification.

The FIPA Request protocol deals with the operational details of when certain messages may or must be sent. It does not address the meanings of the messages themselves. Thus it is perfectly conventional in this regard. Where it deviates from traditional distributed computing is in the semantics it assigns to the messages themselves, which we return to below. However, the benefit of having a protocol is apparent even in this simple example: it identifies the roles and their mutual expectations and thus decouples the implementations of the associated agents from one another.

3.3 State Machines

Figure 3.3 shows a state machine between two roles, merchant (mer) and customer (cus) as a state machine. The transitions are labeled with messages; the prefix mer, cus indicates a message from the merchant to the customer, and cus, mer indicates a message from the customer to the merchant. This state machine supports two executions. One execution represents the scenario where the customer rejects the merchant's offer. The other execution represents the scenario where the customer accepts the offer, following which the merchant and the customer exchange the item and the payment for the item. In the spirit of a state machine, Figure 3.3 does not reflect the internal policies based upon which the customer accepts an offer.

Consider the state machine in Figure 3.4. The dotted paths indicate two additional executions that are not supported by the state machine in Figure 3.3. The executions depict the scenarios where the customer sends the payment upon re-

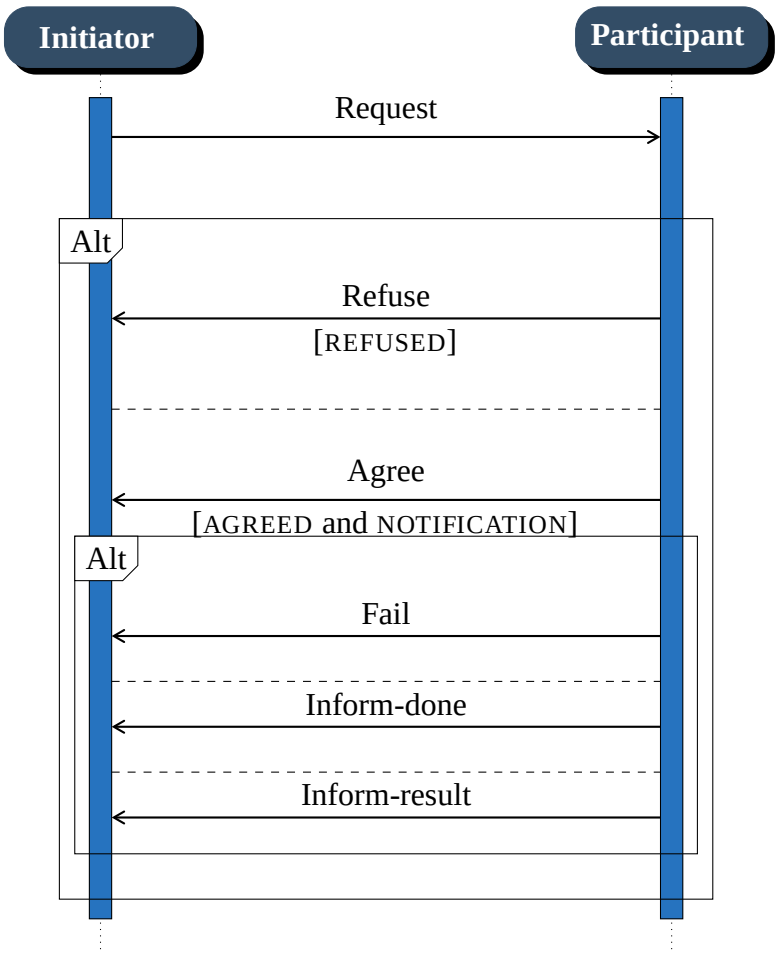

Figure 3.2: FIPA Request Interaction Protocol, from the FIPA specification [26], expressed as a UML Sequence Diagram.

ceiving an offer and after sending an accept, respectively. These additional executions are just as sensible as the original ones. However, in the context of the state machine in Figure 3.3, these executions are trivially *noncompliant*. The reason is that checking compliance with choreographies is purely syntactical—the messages have to flow between the participants exactly as prescribed. Clearly, this curbs the participants' autonomy and flexibility.

We can attempt to ameliorate the situation by producing ever larger FSMs that include more and more paths. However, doing so complicates the implementation

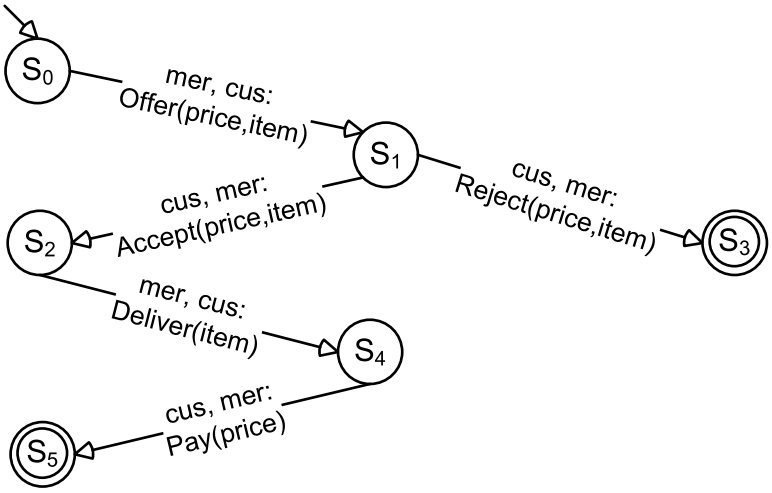

Figure 3.3: A protocol specified as a state machine.

of agents and the task of comprehending and maintaining protocols, while not supporting any real runtime flexibility. Further, any selection of paths will remain arbitrary.

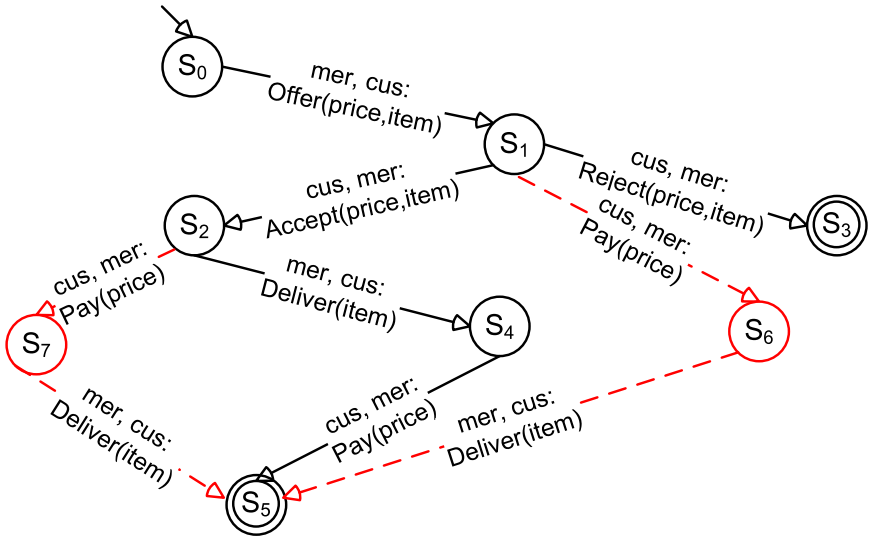

Figure 3.4: An alternative, more flexible state machine.

3.4 Evaluation with Respect to MAS

Traditional software engineering approaches for specifying protocols are operational in nature. Instead of specifying the meaning of a communication, they specify the flow of information among agents. The lack of meaning leads to the following observations about protocols produced following traditional approaches.

Software engineering Because the protocols specify the set of possible enactments at a low level of abstraction, any but the most trivial are difficult to design and maintain. It is difficult to map the business requirements of stakeholders to the protocols produced.

Flexibility Agents have little flexibility at runtime; the protocols essentially dictate agent skeletons. Any deviation from a protocol by an agent, no matter how sensible from a business perspective, is a violation. Further, to enable interoperation, the protocols are specified so that they produce lock-step synchronization among agents, which also limits flexibility.

Compliance Checking an agent's compliance with the protocol is easy: computationally, it is akin to verifying whether a string is accepted by an FSM. However, that ease comes at the expense of flexibility.

4 Traditional AI Approaches

The traditional AI approaches to agent communication begin from the opposite extreme. These approaches presume that the agents are constructed based on cognitive concepts, especially, beliefs, goals, and intentions. Then they specify the communication of such agents in terms of how the communication relates to their cognitive representations.

The AI approaches came from two related starting points, which has greatly affected how they were shaped. The first starting point was of human-computer interaction broadly and natural language understanding specifically. The latter includes the themes of discourse understanding from text or speech, and speech understanding. What these approaches had in common was that they were geared toward developing a tool that would assist a user in obtaining information from a database or performing simple transactions such as booking a train ticket. A key functionality of such tools was to infer what task the user needed to perform and to help the user accordingly. These tools maintained a user model and were

configured with a domain model upon which they reasoned via heuristics to determine how best to respond to their user's request, and potentially to anticipate the user's request.

Such a tool was obviously cooperative: its raison d'être was to assist its user and failure to be cooperative would be simply unacceptable. Further, it was an appropriate engineering assumption that the user was cooperative as well. That is, the tool could be based on the idea that the user was not purposefully misleading it, because a user would gain nothing in normal circumstances by lying about his needs and obtaining useless responses in return.

As the tools became more proactive they began to be thought of as agents. Further, in some cases the agents of different users could communicate with one another, not only with their users. The agents would maintain their models of their users and others based on the communications exchanged. They could make strong inferences regarding the beliefs and intentions of one another, and act and communicate accordingly. These approaches worked for their target setting. To AI researchers, the approaches these agents used for communicating with users and other agents appeared to be applicable for agent communication in general.

The second body of work in AI that related to agent communication came from the idea of building distributed knowledge-based systems (really just expert systems with an ability to communicate with each other). The idea was that each agent would include a reasoner and a knowledge representation and communication was merely a means to share such knowledge. Here, too, we see the same two assumptions as for the human interaction work. First, that the member agents were constructed with the same knowledge representations. Second, that the agents were largely cooperative with each other.

4.1 KQML

Agent communication languages began to emerge in the 1980s. These were usually specific to the projects in which they arose, and typically relied on the specific internal representations used within the agents in those projects.

Somewhat along the same lines, but with some improved generality, arose the Knowledge Query and Manipulation Language or KQML. KQML was created by the DARPA Knowledge Sharing Effort, and was meant to be an adjunct to the other work on knowledge representation technologies, such as ontologies. KQML sought to take advantage of a knowledge representation based on the construct of a knowledge base, such as had become prevalent in the 1980s. Instead of a specific internal representation, KQML assumes that each agent maintains a knowledge

base described in terms of knowledge (more accurately, belief) assertions.

KQML proposed a small number of important primitives, such as *query* and *tell.* The idea was that each primitive could be given a semantics based on the effect it had on the knowledge bases of the communicating agents. Specifically, an agent would send a *tell* for some content only if it believed the content, that is, the content belonged in its knowledge base. And, an agent who received a *tell* for some content would insert that content into its knowledge base, that is, it would begin believing what it was told.

Even though KQML uses knowledge as a layer of abstraction over the detailed data structures of the internal implementation of agent, it turns out to be overly restricted in several ways. The main assumption of KQML is that the communicating agents are cooperative and designed by the same designers. Thus the designers would make sure that an agent sent a message, such as a *tell*, only under the correct circumstances and an agent who received such a message could immediately accept its contents. When the agents are autonomous, they may generate spurious messages—and not necessarily due to malice.

KQML did not provide a clear basis for agent designers to choose which of the message types to use and how to specify their contents. As a result, designers all too often resolved to using a single message type, typically *tell*, with all meanings encoded (usually in some ad hoc manner) in the contents of the messages. That is, the approach is to use different *tell* messages with arbitrary expressions placed within the contents of the messages.

The above challenges complicated interoperability so that it was in general difficult if not impossible for agents developed by different teams to be able to successfully communicate with one another.

4.2 FIPA ACL

We discussed the FIPA interaction protocols in Section 3.2. FIPA has also produced the FIPA ACL, one of the motivations behind which was to address the challenges with KQML. A goal for the FIPA ACL or Agent Communication Language was to specify a definitive syntax through which interoperability among agents created by different developers could be facilitated. In addition, to ensure interoperability, the FIPA ACL also specified the semantics of the primitives. Like KQML's, the FIPA ACL semantics is mentalist, although it has a stronger basis in logic. The FIPA ACL semantics is based on a formalization of the cognitive concepts such as the beliefs and intentions of agents.

Beliefs and intentions are suitable abstractions for designing and implementing agents. However, they are highly unsuitable as a basis for an agent communication language. A communication language supports the interoperation of two or more agents. Thus it must provide a basis for one agent to compute an abstraction of the local state of another agent. The cognitive concepts provide no such basis in a general way. They lead to the internal implementations of the interacting agents to be coupled with each other. The main reason for this is that the cognitive concepts are definitionally internal to an agent. For example, consider the case where a merchant tells a customer that a shipment will arrive on Wednesday. When the shipment fails to arrive on Wednesday, would it be any consolation to the customer that the merchant sincerely believed that it was going to? The merchant could equally well have been lying. The customer would never know without an audit of the merchant's databases. In certain legal situations, such audits can be performed but they are far from the norm in business encounters.

One might hope that it would be possible to infer the beliefs and intentions of another party, but it is easy to see with some additional reflection that no unique characterization of the beliefs and intentions of an agent is possible. In the above example, maybe the merchant had a sincere but false belief; or, maybe the merchant did not have the belief it reported; or, maybe the merchant was simply unsure but decided to report a belief because the merchant also had an intention to consummate a deal with the customer.

It is true that if one developer implements all the interacting agents correctly, the developer can be assured that an agent would send a particular message only in a particular internal state (set of beliefs and intentions). However such a multiagent system would be logically centralized and would be of severely limited value.

It is worth pointing out that the FIPA specifications have ended up with a split personality. FIPA provides the semiformal specification of an agent management system, which underlies the well-regarded JADE system [7]. FIPA also provides definitions for several interaction protocols (discussed in Section 3.2), which are also useful and used in practice, despite their limitations. FIPA provides a formal semantics for agent communication primitives based on cognitive concepts, which gives a veneer of rigor, but is never used in multiagent systems.

4.3 Evaluation with Respect to MAS

The traditional AI approaches are mentalist, which render them of limited value for multiagent systems.

Software engineering The AI approaches offer high-level abstractions, which is a positive. However, because the abstractions are mentalist, the approaches cannot be applied to the design of multiagent systems except in the restricted case where one developer designs all the agents (as explained above). Since there is no interaction in the sense of interaction Further, recall the discussion from Section 2.2 regarding the unsuitability of a small set of primitives. Both KQML and FIPA suffer from this problem.

Flexibility The flexibility of agents is severely curtailed because of restrictions on when agents can send particular communications.

Compliance It is impossible for an observer to verify the cognitive state of an agent. Hence verifying agent compliance (for example, if the agent has the requisite cognitive state for sending a particular message) is impossible.

5 Commitment-Based Multiagent Approaches

In contrast with the operational approaches, commitment protocols give primacy to the *business meanings* of service engagements, which are captured through the participants' *commitments* to one another [60], [11, 52, 22, 56], [20]. Computationally, each participant is modeled as an *agent*; interacting agents carry out a service engagement by creating and manipulating commitments to one another.

5.1 Commitments

A commitment is an expression of the form $\mathsf{C}(\mathit{debtor}, \mathit{creditor}, \mathit{antecedent}, \mathit{consequent})$, where *debtor* and *creditor* are agents, and *antecedent* and *consequent* are propositions. A commitment $\mathsf{C}(x, y, r, u)$ means that x is committed to y that if r holds, then it will bring about u. If r holds, then $\mathsf{C}(x, y, r, u)$ is *detached*, and the commitment $\mathsf{C}(x, y, \top, u)$ holds ($\top$ being the constant for truth). If u holds, then the commitment is *discharged* and does not hold any longer. All commitments are *conditional*; an unconditional commitment is merely a special case where the antecedent equals $\top$. Examples 1–3 illustrate these concepts. In the examples below, EBook is a bookseller, and Alice is a customer.)

Example 1 (Commitment) $\mathsf{C}(\mathit{EBook}, \mathit{Alice}, \$12, \mathit{BNW})$ means that EBook commits to Alice that if she pays $12, then EBook will sends her the book *Brave New World*.

Example 2 (Detach) If Alice makes the payment, that is, if \$12 holds, then $\mathsf{C}(EBook, Alice, \$12, BNW)$ is detached. In other words, $\mathsf{C}(EBook, Alice, \$12, BNW) \wedge \$12 \Rightarrow \mathsf{C}(EBook, Alice, \top, BNW)$.

Example 3 (Discharge) If EBook sends the book, that is, if *BNW* holds, then both $\mathsf{C}(EBook, Alice, \$12, BNW)$ and $\mathsf{C}(EBook, Alice, \top, BNW)$ are discharged. In other words, $BNW \Rightarrow \neg\mathsf{C}(EBook, Alice, \$12, BNW) \wedge \neg\mathsf{C}(EBook, Alice, \top, BNW)$.

Importantly, commitments can be manipulated, which supports flexibility. The commitment operations [45] are listed below; CREATE, CANCEL, and RELEASE are two-party operations, whereas DELEGATE and ASSIGN are three-party operations.

- CREATE(x, y, r, u) is performed by x, and it causes $\mathsf{C}(x, y, r, u)$ to hold.
- CANCEL(x, y, r, u) is performed by x, and it causes $\mathsf{C}(x, y, r, u)$ to not hold.
- RELEASE(x, y, r, u) is performed by y, and it causes $\mathsf{C}(x, y, r, u)$ to not hold.
- DELEGATE(x, y, z, r, u) is performed by x, and it causes $\mathsf{C}(z, y, r, u)$ to hold.
- ASSIGN(x, y, z, r, u) is performed by y, and it causes $\mathsf{C}(x, z, r, u)$ to hold.
- DECLARE(x, y, r) is performed by x to inform y that the r holds.

DECLARE is not a commitment operation, but may indirectly affect commitments by causing detaches and discharges. In relation to Example 2, when Alice informs EBook of the payment by performing DECLARE$(Alice, EBook, \$12)$, then the proposition \$12 holds, and causes a detach of $\mathsf{C}(EBook, Alice, \$12, BNW)$.

Further, a commitment arises in a social or legal context. The context defines the rules of encounter among the interacting parties, and often serves as an arbiter in disputes and imposes penalties on parties that violate their commitments. For example, eBay is the context of all auctions that take place through the eBay marketplace; if a bidder does not honor a payment commitment for an auction that it has won, eBay may suspend the bidder's account.

A formal treatment of commitments and communication based on commitments is available in the literature [48, 15].

Table 3.1: A commitment protocol.

$\mathit{Offer}(\mathit{mer}, \mathit{cus}, \mathit{price}, \mathit{item})$ means $\text{CREATE}(\mathit{mer}, \mathit{cus}, \mathit{price}, \mathit{item})$
$\mathit{Accept}(\mathit{cus}, \mathit{mer}, \mathit{price}, \mathit{item})$ means $\text{CREATE}(\mathit{cus}, \mathit{mer}, \mathit{item}, \mathit{price})$
$\mathit{Reject}(\mathit{cus}, \mathit{mer}, \mathit{price}, \mathit{item})$ means $\text{RELEASE}(\mathit{mer}, \mathit{cus}, \mathit{price}, \mathit{item})$
$\mathit{Deliver}(\mathit{mer}, \mathit{cus}, \mathit{item})$ means $\text{DECLARE}(\mathit{mer}, \mathit{cus}, \mathit{item})$
$\mathit{Pay}(\mathit{cus}, \mathit{mer}, \mathit{price})$ means $\text{DECLARE}(\mathit{cus}, \mathit{mer}, \mathit{price})$

5.2 Commitment Protocol Specification

Table 3.1 shows the specification of a commitment protocol between a merchant and a customer (omitting sort and variable declarations). It simply states the meanings of the messages in terms of the commitments arising between the merchant and customer. For instance, the message $\mathit{Offer}(\mathit{mer}, \mathit{cus}, \mathit{price}, \mathit{item})$ means the creation of the commitment $\mathsf{C}(\mathit{mer}, \mathit{cus}, \mathit{price}, \mathit{item})$, meaning the merchant commits to delivering the item if the customer pays the price; $\mathit{Reject}(\mathit{cus}, \mathit{mer}, \mathit{price}, \mathit{item})$ means a release of the commitment; $\mathit{Deliver}(\mathit{mer}, \mathit{cus}, \mathit{item})$ means that the proposition *item* holds.

Figure 3.5(A) shows an execution of the protocol and Figure 3.5(B) its meaning in terms of commitments. (The figures depicting executions use a notation similar to UML interaction diagrams. The vertical lines are agent lifelines; time flows downward along the lifelines; the arrows depict messages between the agents; and any point where an agent sends or receives a message is annotated with the commitments that hold at that point. In the figures, instead of writing CREATE, we write *Create*. We say that the *Create* message realizes the CREATE operation. Likewise, for other operations and DECLARE.) In the figure, the merchant and customer role are played by EBook and Alice, respectively; c_B and c_{UB} are the commitments $\mathsf{C}(\mathit{EBook}, \mathit{Alice}, \$12, \mathit{BNW})$ and $\mathsf{C}(\mathit{EBook}, \mathit{Alice}, \top, \mathit{BNW})$ respectively.

5.3 Evaluation with Respect to MAS

Compliance Protocol enactments can be judged correct as long as the parties involved do not violate their commitments. A customer would be in violation if he keeps the goods but fails to pay. In this manner, commitments support business-level compliance and do not dictate specific operationalizations [22].

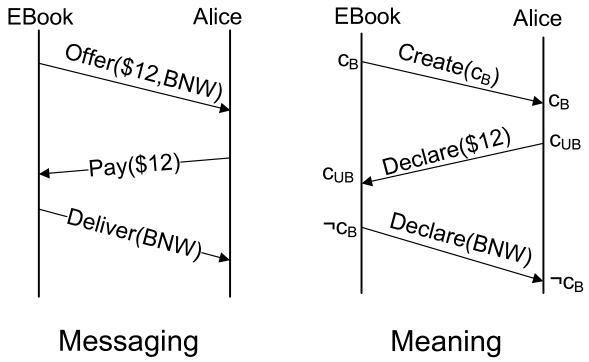

Figure 3.5: Distinguishing message syntax and meaning: two views of the same enactment.

Flexibility The above formulation of correctness enhances flexibility over traditional approaches by expanding the operational choices for each party [13]. For example, if the customer substitutes a new way to make a payment or elects to pay first, no harm is done, because the behavior is correct at the business level. And, the merchant may employ a new shipper; the customer may return damaged goods for credit; and so on. By contrast, without business meaning, exercising any such flexibility would result in noncompliant executions.

Software Engineering Commitments offer a high-level abstraction for capturing business interactions. Further, a commitment-based approach accommodates the autonomy of the participants in the natural manner: socially, an agent is expected to achieve no more than his commitments. Commitments thus also support loose coupling among agents. Commitment-based approaches offer a compelling alternative to the traditional SE approaches described in Section 3 for building systems comprised of autonomous agents.

Figure 3.6 shows some of the possible enactments based on the protocol in Table 3.1. The labels c_A and c_{UA} are $\mathsf{C}(\mathit{Alice}, \mathit{EBook}, \mathit{BNW}, \$\mathit{12})$ and $\mathsf{C}(\mathit{Alice}, \mathit{EBook}, \top, \$\mathit{12})$, respectively. Figure 3.6(B) shows the enactment where the book and payment are exchanged in Figure 3.3. Figures 3.6(A) and (C) show the additional executions supported in Figure 3.4; Figure 3.6(D) reflects a new execution that we had not considered before, one where Alice sends an *Accept* even before receiving an offer. All these executions are compliant executions in terms of commitments, and are thus supported by the protocol in Table 3.1.

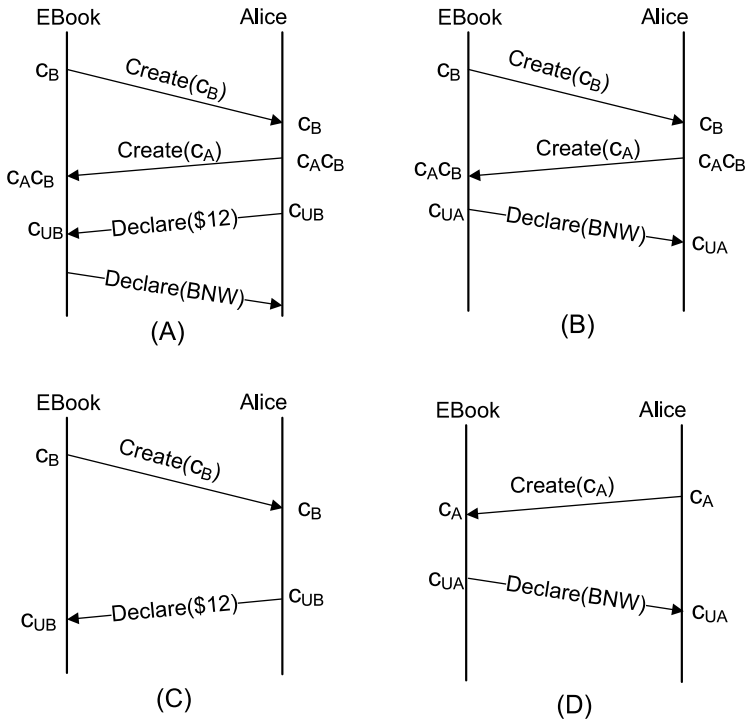

Figure 3.6: Flexible enactment.

Table 3.2: Comparison of agent communication approaches.

	Traditional SE	**Traditional AI**	**Commitment Protocols**
Abstraction	control flow	mentalist	business relationship
Compliance	lexical basis	unverifiable	semantic basis
Flexibility	low	low	high
Interoperability	message-level	integration	business-level

Table 3.2 summarizes the three approaches.

6 Engineering with Agent Communication

Protocols support the development of distributed systems. A natural way to apply protocols is to derive from them the specifications of the roles that feature in them. The idea is to use these role specifications as a basis for designing and implementing the agents who would participate in the given protocol. Role specifications are sometimes termed *role skeletons* or *endpoints*, and the associated problem is called *role generation* and *endpoint projection*.

The above motivation of implementing the agents according to the roles suggests an important quality criterion. We would like the role specifications to be such that agents who correctly implement the roles can interoperate successfully without the benefit of any additional messages than those included in the protocol and which feature in the individual role specifications. In other words, we would like the agents implementing the roles to only be concerned with satisfying the needs of their respective roles without regard to the other roles: the overall computation would automatically turn out to be correct.

Role generation is straightforward for two-party protocols. This is so because any message sent by one role is received by the other. Thus it is easy to ensure their joint computations generate correct outcomes. But when three or more roles are involved, because any message exchange involves two agents (neglecting multicast across roles for now) leaves one or more roles unaware of what has transpired. As a result, no suitable role skeletons may exist for a protocol involving three or more parties. We take this nonexistence to mean that the protocol in question is causally ill-formed and cannot be executed in a fully distributed manner. Such a protocol must be corrected, usually through the insertion of messages that make sure that the right information flows to the right parties and that potential race conditions are avoided.

In a practical setting, then, the role skeletons are mapped to a simple set of method stubs. An agent implementing a role—in this metaphor, by fleshing out its skeleton—provides methods to process each incoming message and attempts to send only those messages allowed by the protocol. Role skeletons do not consider the contents of the messages. As a result, they can be expressed in a finite state machine too. Notice this machine is different from a state machine that specifies a protocol. A role's specification is very much focused on the perspective of the role whereas the machine of a protocol describes the progress of a protocol enactment from a neutral perspective.

6.1 Programming with Communications

The Java Agent Development Framework (JADE) is a popular platform for developing and running agent-based applications. It implements the FIPA protocols discussed earlier. JADE provides support for the notion of what it terms *behaviors*. A behavior is an abstract specification of an agent that characterizes important events such as the receipt of specified messages and the occurrence of timeouts. To implement an agent according to a behavior involves defining the methods it specifies as callbacks. In particular, a role skeleton can be implemented by defining the handlers for any incoming methods. The JADE tutorial online offers comprehensive instructions for building JADE applications.

6.2 Modeling Communications

It is not trivial to specify the *right* commitments for particular applications. For instance, Desai et al. [19] show how a scenario dealing with foreign exchange transactions may be formalized in multiple ways using commitments, each with different ramifications on the outcomes. The challenge of specifying the right commitments leads us to the question: *How can we guide software engineers in creating appropriate commitment-based specifications?*

Such guidance is often available for operational approaches such as state machines and Petri nets that describe interactions in terms of message order and occurrence. For instance, Figure 3.7 shows two common patterns expressed as (partial) state machines, which can aid software engineers in specifying operational interactions. Here, *b* and *s* are buyer and seller, respectively. (A) says that the seller may accept or reject an order; (B) says the buyer may confirm an order after the seller accepts it.

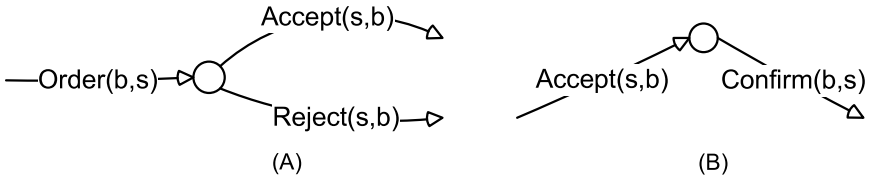

Figure 3.7: Example operational patterns

By contrast, commitment protocols abstract away from operational details,

focusing on the meanings of messages, not their flow. Clearly, operational patterns such as the above would not apply to the design of commitment protocols. What kinds of patterns would help in the design of commitment protocols? By and large, they would need to be *business patterns*—characterizing requirements, not operations—that emphasize meanings in terms of commitments. In contrast with Figure 3.7, these patterns describe what it *means* to make, accept, reject, or update an offer, not when to send specific messages.

Business patterns support specifying business protocols. These patterns are motivated by the following principles.

Autonomy compatibility Autonomy broadly refers to the lack of control: no agent has control over another agent. To get things done, agents set up the appropriate commitments by interacting. Any expectation from an agent beyond what the agent has explicitly committed would cause hidden coupling.

Explicit meanings The meaning ought to be made public, not hidden within agent implementations.

6.2.1 Business Patterns

Business patternspattern encode the common ways in which businesses engage each other. Below is an example of the *compensation* pattern.

• COMPENSATION

Intent To compensate the creditor in case of commitment cancellation or violation by the debtor.

Motivation It is not known in advance whether a party will fulfill its commitments; compensation commitments provide some assurance to the creditor in case of violations.

Implementation $Compensate(x,y,r,u,p)$ means $Create(x,y,violated(x,y,r,u),p)$.

Example $Compensate(mer,cus,price,item,discount)$; it means that the merchant will offer the customer a discount on the next purchase if the item is paid for but not delivered.

Consequences A commitment (even a compensation commitment) should ideally be supported by compensation; however, at some level, the only recourse is escalation to the surrounding business *context*—for example, the local jurisdiction [51].

6.2.2 Enactment Patterns

Whereas a business pattern describes the meaning of communication, an enactment pattern describes the conditions under which an agent should enact a business pattern, that is, *when* to undertake the corresponding communication. A locus of such enactments may serve as the basic agent skeleton.

• COUNTER OFFER

Intent One party makes an offer to another, who responds with a modified offer of its own.

Motivation Essential for negotiation.

When Let $\mathsf{C}(x,y,r,u)$ be the commitment corresponding to the original offer. Making a counteroffer would amount to creating the commitment $\mathsf{C}(y,x,u',r')$ such that $u' \vdash u$ and $r \vdash r'$, in other words, if the consequent is strengthened and the antecedent is weakened. An alternative implementation includes doing $\mathit{Release}(x,y,r,u)$ in addition.

Example Let's say $\mathsf{C}(\mathit{EBook},\mathit{Alice},\$12,\mathit{BNW})$ holds. Alice can make the counter offer $\mathsf{C}(\mathit{Alice},\mathit{EBook},\mathit{BNW} \wedge \mathit{Dune},\$12)$ meaning that she wants *Dune* in addition to *BNW* for the same price.

Consequences When $u \equiv u'$ and $r \equiv r'$, the counter offer amounts to a mutual commitment.

6.2.3 Semantic Antipatterns

Semantic antipatterns identify forms of representation and reasoning to be avoided because they conflict with the autonomy of the participants or with a logical basis for commitments.

• COMMIT ANOTHER AS DEBTOR

Intent An agent creates a commitment in which the debtor is another agent.

Motivation To capture delegation, especially in situations where the delegator is in a position of power of over the delegatee.

Implementation The sender of $\mathsf{Create}(\mathsf{y},\mathsf{z},\mathsf{p},\mathsf{q})$ is x (x and y are different agents), thus contravening the autonomy of the y.

Example Consider two sellers EBook and BookWorld. EBook sends $Create(BookWorld, Alice, \$12, BNW)$ to Alice, which violated BookWorld's autonomy.

Consequences A commitment represents a public undertaking by the debtor. A special case is when $x = z$. That is, x unilaterally makes itself the creditor.

Criteria Failed y's autonomy is not respected.

Alternative Apply delegation to achieve the desired business relationship, based on prior commitments. In the above example, BookWorld could have a standing commitment with EBook to accept delegations. EBook can then send a delegate "instruction" to BookWorld upon which BookWorld commits to Alice.

The above are some examples of patterns. For a more exhaustive list of patterns, see [16].

6.3 Communication-Based Methodologies

Because of the centrality of agent communication to multiagent systems, a number of methodologies for designing and implementing multiagent systems are based on communications. We point out a few such methodologies in the further readings section.

The common idea behind these methodologies is to identify the communications involved in the system being specified and to state the meanings of such communications. The main protocol concepts are roles, messages, and message meanings. Below we briefly outline the high-level considerations involved in designing a protocol.

- Identify stakeholder requirements.
- Identify the roles involved in the interaction. Let's say the roles identified are customer, merchant, shipper, and banker.

- If a suitable protocol is available from a repository, then choose it and we're done. After all, one of key benefits of protocols is reusability. For instance, suppose the stakeholders wanted to design a purchase protocol. If the protocol of Table 3.1 fits their requirements, we're done.

- Often the required protocol may be obtained by *composing* existing protocols. For example, the desired protocol could potentially be obtained by combining *Ordering*, *Payment*, and *Shipping* protocols.

- Sometimes the protocol or parts of its may need to be written up from scratch. Identify the communications among the roles. For example, there would be messages between the customer and the merchant that would pertain to ordering items. The messages between the customer and bank would pertain to payment, and so on.

- Identify how the messages would affect their commitments. For example, the *Offer* message could be given a meaning similar to the one in Table 3.1. The customer's payment to the bank would effectively discharge his commitment to pay the merchant. Similarly, the delivery of the goods by the shipper would effectively discharge the merchant's commitment to pay, and so on.

7 Advanced Topics and Challenges

This section describes some important current directions in agent communication.

7.1 Primacy of Meaning

As we outlined in the foregoing, there is an unfortunate tendency to specify communication protocols in operational terms at the cost of the meanings that they convey. However, agent communication should be understood at the level of the "social state" of the parties involved and how it affects and is affected by communications. Adopting a meaning-based stance protects one's models from inadvertent dependencies upon implementation and yields the highest flexibility for the participating agents while maintaining correctness.

The earlier meaning-based approaches to agent communication tended to combine assertions regarding the meanings of communications with operational details, such as the conditions under which what communication must occur and

how the communications must be mutually ordered. Such operational details interfere with an application of meaning-based reasoning because they require that the agents maintain not only the meanings of the communication and the changing social state but also additional, otherwise irrelevant, dependencies with the decisions of other agents.

We have not been able to find even a single compelling "natural" situation where such details are necessary. Any requirement that an agent produce a message is a violation of its autonomy. When we think of meaning properly, there is never a natural need for ordering constraints—the only ordering constraints that might arise are those based on artificial grounds such as arbitrary conventions in a particular domain. Such conventions are fine and an approach for agent communication should support them. However, they do not explain the large number of ordering constraints that traditional specifications tend to include.

Although the operational details interfere with reasoning about meaning, they are essential to ensure that each party obtains the information it needs at the right time so as to proceed effectively. The recent approach termed the Blindingly Simple Protocol Language [49] provides a simple resolution to this tension by capturing the necessary operational details in a declarative manner. The declarative representation of messages facilitates producing assertions regarding social state from them, and using such assertions as a basis for reasoning about the meanings of the messages.

A research challenge, then, is to develop languages and methodologies in which (and with which to formulate) proper meanings for communications, so as to capture the needs of domain settings precisely.

7.2 Verifying Compliance

Because agent communication involves the interactions of two or more autonomous parties it inherently has the weight of a "standard"—albeit a minor, nonuniversal standard. In other words, when two agents talk to one another, they must agree sufficiently on what they are talking about and they must be able to judge if their counterparty is interacting in a manner that they would expect. To the first point, the traditional approaches missed stating expectations properly.

Just as a standard in any domain of practice is worthless if we cannot judge whether the parties subject to the standard are complying with it or not, so it is with agent communication. Any approach for agent communication must support the statement of the mutual expectations of the parties involved *and* do so in a manner that supports each party verifying if the others are complying with its

expectations of them. This is an obvious point in retrospect. However, the mentalist approaches disregarded the problem of compliance. Despite this point having been explained over a decade ago [44], there remains a tendency to disregard it in approaches to communication, especially as such approaches are applied within software engineering methodologies.

A research challenge here is to design specification languages that promote the verification of compliance and, more importantly, to develop algorithms by which an agent or a set of cooperating agents could verify the compliance of others based on the communications it can monitor.

7.3 Protocol Refinement and Aggregation

If we are to treat communication as a first-class abstraction for specifying multiagent systems, we must be ready to support dealing with conceptual modeling using that abstraction. Classically, two conceptual modeling relations are known: refinement and aggregation. *Refinement* deals with how a concept refines another in the sense of the is-a hierarchy. *Aggregation* deals with how concepts are put together into composites in the sense of the part-whole hierarchy. Refinement and aggregation are well-understood for traditional object-oriented design and are supported by modern programming languages.

However, dealing with refinement in particular has been nontrivial for communication protocols. Recent work on session types is promising in this regard [32] as is work on refinement with respect to commitment-based protocols [30]. An important challenge is to produce a generalized theory and associated languages and tools that would support refinement and aggregation of protocols for more powerful meaning specifications.

7.4 Role Conformance

As we stated above, the meaning of communication captures the expectations that the parties involved can have of each other. Accordingly, an important engineering challenge is to develop agents who would meet such expectations. An agent can potentially apply complex reasoning and, therefore, verifying that an agent (implementation) would meet the expectations of another agent is nontrivial.

A natural way to approach the problem is to formulate a role description or a role *skeleton* based on the specification of a communication protocol. A skeleton describes the basic structure of a role. An agent who plays (and hence implements) a role would provide additional details so as to flesh out the structure that

is the skeleton. Since a protocol involves two or more roles the challenge is to determine sufficient structural properties of each role, in terms of what messages it can receive and send under what circumstances and any constraints on how the local representation of the social state should progress in light of the messages received and sent. We can then publish the descriptions of each role in a protocol along with the protocol specification.

At the same time, one can imagine that software vendors may produce agent implementations that are compatible with different roles. A vendor would not and should not provide the internal details but would and should provide the public "interface" of the agent in terms of its interactions. In other words, a vendor would describe a role that its agent would be able to play. In general, an agent may need to participate in more than one protocol. Thus it would help to know if the role as published by a vendor conforms with the role as derived from a protocol. This is the problem of *role conformance*. Solving this problem for a particular language would help automate part of the task of creating a multiagent system from disparate agents while ensuring that the agents, even if implemented heterogeneously, would be able to interoperate with respect to a specified protocol.

An important research challenge is to identify formal languages for specifying roles along with algorithms for determining whether a role conforms with another.

8 Conclusions

It should be no surprise to anyone that communication is at the heart of multiagent systems, not only in our implementations but also in our conception of what a multiagent system is and what an agent is.

To our thinking, an agent is inherently autonomous. Yet, autonomous, heterogeneously constructed agents must also be interdependent on each other if they are to exhibit complex behaviors and sustain important real-world applications. A multiagent system, if it is any good, must be loosely coupled and communication is the highly elastic glue that keeps it together. Specifically, communication, understood in terms of agents and based on high-level abstractions such as those we explained above, provides the quintessential basis for the arms-length relationships desired in all modern software engineering as it addresses the challenges of large decentralized systems.

The foregoing provided a historical view of the agent communication, identifying the main historical and current ideas in the field. This chapter has only scratched the surface of this rich and exciting area. We invite the reader to delve

deeper and to consider many of the fundamental research problems that arise in this area. An important side benefit is that, faced with the challenges of open systems such as on the Web, in social media, in mobile computing, and cyber-physical systems, traditional computer science is now beginning to appreciate the importance and value of the abstractions of agent communication. Thus progress on the problems of agent communication can have significant impact on much of computer science.

Further Reading

Agent communication is one of the most interesting topics in multiagent systems, not only because of its importance to the field but also because of the large number of disciplines that it relates to. In particular, it touches upon ideas in philosophy, linguistics, social science (especially organizations and institutions), software engineering, and distributed computing. The readings below will take the reader deeper into these subjects.

Philosophical foundations. Some of the most important works on the philosophy of language undergird the present understanding of communication. Austin [6] introduced the idea of communication as action. Searle developed two accounts of communication, one emphasizing the mental concepts of the parties involved [41] and the second the notion of social reality that sustains and is sustained by language [42]. Some recent works by Chopra, Singh, and their colleagues have exploited the distinction between constitution and regulation that Searle described [14, 38].

Organizations and institutions. Several researchers in multiagent systems have studied the notions of organizations and institutions. These works provide computational bases for agents to participate in structured relationships. The works of Vázquez-Salceda and the Dignums [55, 3] and of Fornara and Colombetti [27] highlight important conceptual and practical considerations in this area.

Norms, conventions, and commitments. The notions of organizations and institutions are defined based on the normative relationships that arise among their participants. Artikis, Jones, Pitt, and Sergot have developed formalizations of norms that are worth studying as influential papers [5, 37]. Jones and Parent [36] formalize conventions as a basis for communication.

Singh proposed the notion of social commitments [43, 45] as an important normative concept to be used for understanding social relationships. He proposed commitments as basis for a social semantics for communication [46]. A related idea has been developed by Colombetti [17]. A formal semantics for commitments [48] and the proper reasoning about commitments in situations with asynchronous communication among decoupled agents [15] are significant to practice and promising as points of departure for important research in this area.

Software engineering. A number of approaches apply communications as central to the development of multiagent systems [10, 39, 29, 21, 16]. Further, several design and verification tools for communication protocols and agent communication generally have been proposed [59, 4, 2, 1, 56, 23]. The development of well-principled tools is an important research direction because of their potential impact on computer science—if they could lead to the expanded deployment of multiagent system.

Challenges. The agent communication manifesto is a collection of short essays by several researchers that seek to articulate the main challenges and directions in this area [12]. The reader should consult it before embarking on research in this area.

9 Exercises

1. Level 1 Which of the following statements are true?

   (a) Communications are an important class of interactions because they support the autonomy of the parties involved

   (b) The three elements of a communicative act are locution, illocution, and perlocution

   (c) Unlike traditional settings, perlocutions provide the right basis for communicative acts in open, service-oriented settings

   (d) Unlike in a traditional finite state machine, the states of a commitment machine are specified using logic and each transition corresponds to the meaning of the message that labels the transition

(e) In an open environment, two agents might sometimes need to combine their local observations in order to determine that a third agent is complying with its commitments

2. Level 1 Which of the following statements are true?

   (a) In an open environment, we can typically ensure compliance based upon the implementations of the interacting agents

   (b) The benefit of employing a commitment protocol is that it exactly specifies the order of the messages without regard to their meaning

   (c) Using the meanings of the messages, we can compute whether a message may be sent in the current state, and the next state that would result from doing so

   (d) Ideally, each participant in a protocol should be able to verify if any of the commitments where it is the creditor are violated

3. Level 1 Which of the following statements are true about interaction and communication?

   (a) Perlocutions are considered the core aspect of a communicative act

   (b) The same proposition, e.g., reserve(Alice, UA 872, 14 May 2020), may feature in a *request* and a *declare*

   (c) We may not be able to decide if a statement such as *Shut the door!* is true or false but we can decide whether such a statement was made

   (d) A statement such as *Shut the door!* becomes true if the door in question is shut on *purpose*, not accidentally

4. Level 1 Identify all of the following statements that are true about commitments and commitment protocols

   (a) If the debtor of a commitment delegates it simultaneously with the creditor of the same commitment assigning it, additional messages are in general needed for the new debtor and the new creditor to learn about each other

   (b) If the debtor of a commitment discharges it simultaneously with the creditor of the same commitment assigning it, no additional messages are needed for the new creditor to learn that the debtor is compliant

(c) A protocol for payment through a third party could naturally be specified using the delegate of a commitment to pay

(d) Forward-going interactions such as ordering and payment may be modeled as commitment protocols, but not backward-going interactions such as returning goods for a refund

(e) Even though a commitment protocol captures the meanings of the messages involved, the participants must accept the protocol in order for it to work

5. Level 2 We say that a commitment is *discharged* when the consequent holds, *expired* when the antecedent cannot ever hold, and *violated* when the antecedent holds but the consequent cannot ever hold.

   Let $\mathbf{E} = \{e_0, e_1, e_2, \ldots, \overline{e}_0, \overline{e}_1, \overline{e}_2, \ldots\}$ be a set of events such that $\overline{e}_i$ is the complement of e_i. For instance, if e_0 means *package was delivered by 5PM*, $\overline{e}_0$ means *package was not delivered by 5PM*. Further, $\overline{\overline{e}}_0 = e$.

   Let $\langle v_0, v_1, ..., v_n \rangle$ represent an event trace, that is, the sequence of events that have been recorded, where all the v_i are variables that range over $\mathbf{E}$. Further, in any event trace, for any event, only the event or its complement may occur, but not both (for example, the package was either delivered by 5PM or it was not, but not both). Thus, for example, $\langle e_0, e_3, \overline{e}_5 \rangle$ is a valid trace, but $\langle e_0, e_3, \overline{e}_5, \overline{e}_0 \rangle$ is not.

   Assume that the commitment $\mathsf{C}(x, y, e_0, e_1 \wedge e_2)$ holds right before we start recording events (x commits to y that if e_0 occurs, both e_1 and e_2 will occur).

   For each the following event traces, indicate whether the commitment is (a) satisfactorily resolved (via discharge or expiration), (b) violated, or (c) continues to hold.

   (a) $\langle e_0, e_1, e_5 \rangle$

   (b) $\langle \overline{e}_1, e_0, e_2 \rangle$

   (c) $\langle e_1, \overline{e}_0, e_3 \rangle$

   (d) $\langle e_1, e_0, e_2 \rangle$

   (e) $\langle \overline{e}_0, \overline{e}_1, \overline{e}_2 \rangle$

6. Level 2 Examine Figure 3.1. Now create an FSM for the commitment compensate pattern discussed in the chapter.

7. Level 2 Examine Figure 3.1. Now specify a commitment pattern that captures the idea of updating commitments.

8. Level 2 Create an FSM corresponding to the FIPA Request protocol shown in Figure 3.2.

9. Level 3 Create a WS-CDL specification for the FIPA Request protocol.

10. Level 3 Consider the following outline of a process for buying books. A merchant offers an online catalog of books with price and availability information. A customer can browse the catalog and purchase particular books from the catalog or the merchant may contact the customer directly with offers for particular books. However, the customer must arrange for shipment on his own: in other words, he must arrange for a shipper to pick up the books from the merchant's store and deliver them to him. All payments—to the merchant for the books and to the shipper for delivery—are carried out via a payment agency (such as PayPal).

    (a) List the roles and messages involved in the protocol underlying the above business process

    (b) Specify the messages in terms of communicative acts

    (c) Specify the protocol in three different ways: as an FSM with messages as the transitions, (2) an MSC, and (3) a commitment protocol

    (d) Show a simplified MSC representing one possible enactment where the books have been delivered and the payments have been made.

    (e) Based on the commitment protocol you specified above, annotate points in the above described enactment with commitments that hold at those points

11. Level 4 Suppose the business process described in Question 10 above also supported returns and refunds for customers.

    (a) As we did above, specify the underlying protocol as an FSM, as an MSC, and as a commitment protocol

    (b) Show both a synchronous and an asynchronous return-refund enactment.

    (c) Annotate both with the commitments at various points. (Hint: for the asynchronous enactment, read [15])

12. Level 3 Specify role skeletons for the purchase process with returns and refunds.

    (a) In the JADE style.

    (b) In the rule-based style. (Hint: read [22])

13. Level 3 Map Figure 3.6 to an FSM and an MSC.

14. Level 3 Compare the FSM and MSC from Question 13 to the commitment protocol specification of Table 3.1 with respect to compliance, ease of creation, and ease of change.

15. Level 4 Implement the logic for practical commitments described in [48].

16. Level 4 Implement a commitment-based middleware based on the postulates given in [15].

Acknowledgments

We have benefited from valuable discussions about agent communication with several colleagues, in particular, our coauthors on papers relating to agent communication: Matteo Baldoni, Cristina Baroglio, Nirmit Desai, Scott Gerard, Elisa Marengo, Viviana Patti, and Pınar Yolum.

Some parts of this chapter have appeared in previous works by the authors [16, 38].

Amit Chopra was supported by a Marie Curie Trentino award. Munindar Singh's effort was partly supported by the National Science Foundation under grant 0910868. His thinking on this subject has benefited from participation in the OOI Cyberinfrastructure program, which is funded by NSF contract OCE-0418967 with the Consortium for Ocean Leadership via the Joint Oceanographic Institutions.

References

[1] Marco Alberti, Federico Chesani, Marco Gavanelli, Evelina Lamma, Paola Mello, Marco Montali, and Paolo Torroni. Web service contracting: Specification and reasoning with SCIFF. In *Proceedings of the 4th European Semantic Web Conference*, pages 68–83, 2007.

[2] Marco Alberti, Marco Gavanelli, Evelina Lamma, Paola Mello, and Paolo Torroni. Modeling interactions using social integrity constraints: A resource sharing case study. In *Proceedings of the International Workshop on Declarative Agent Languages and Technologies (DALT)*, volume 2990 of *LNAI*, pages 243–262. Springer, 2004.

[3] Huib Aldewereld, Sergio Álvarez-Napagao, Frank Dignum, and Javier Vázquez-Salceda. Making norms concrete. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 807–814, Toronto, 2010. IFAAMAS.

[4] Alexander Artikis, Marek J. Sergot, and Jeremy Pitt. An executable specification of a formal argumentation protocol. *Artificial Intelligence*, 171(10–15):776–804, 2007.

[5] Alexander Artikis, Marek J. Sergot, and Jeremy V. Pitt. Specifying norm-governed computational societies. *ACM Transactions on Computational Logic*, 10(1), 2009.

[6] John L. Austin. *How to Do Things with Words*. Clarendon Press, Oxford, 1962.

[7] Fabio Luigi Bellifemine, Giovanni Caire, and Dominic Greenwood. *Developing Multi-Agent Systems with JADE*. Wiley-Blackwell, 2007.

[8] Boualem Benatallah, Fabio Casati, and Farouk Toumani. Analysis and management of web service protocols. In *Conceptual Modeling ER 2004*, volume 3288 of *LNCS*, pages 524–541. Springer, 2004.

[9] Carlos Canal, Lidia Fuentes, Ernesto Pimentel, José M. Troya, and Antonio Vallecillo. Adding roles to CORBA objects. *IEEE Transactions on Software Engineering*, 29(3):242–260, 2003.

[10] Christopher Cheong and Michael P. Winikoff. Hermes: Designing flexible and robust agent interactions. In Virginia Dignum, editor, *Handbook of Research on Multi-Agent Systems: Semantics and Dynamics of Organizational Models*, chapter 5, pages 105–139. IGI Global, Hershey, PA, 2009.

[11] Amit Chopra and Munindar P. Singh. Nonmonotonic commitment machines. In Frank Dignum, editor, *Advances in Agent Communication: Proceedings of the 2003 AAMAS Workshop on Agent Communication Languages*, volume 2922 of *LNAI*, pages 183–200. Springer, 2004.

[12] Amit K. Chopra, Alexander Artikis, Jamal Bentahar, Marco Colombetti, Frank Dignum, Nicoletta Fornara, Andrew J. I. Jones, Munindar P. Singh, and Pınar Yolum. Research directions in agent communication. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 2011. To appear.

[13] Amit K. Chopra and Munindar P. Singh. Contextualizing commitment protocols. In *Proceedings of the 5th International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 1345–1352, 2006.

[14] Amit K. Chopra and Munindar P. Singh. Constitutive interoperability. In *Proceedings of the 7th International Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*, pages 797–804, Estoril, Portugal, May 2008. IFAAMAS.

[15] Amit K. Chopra and Munindar P. Singh. Multiagent commitment alignment. In *Proceedings of the 8th International Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*, pages 937–944, Budapest, May 2009. IFAAMAS.

[16] Amit K. Chopra and Munindar P. Singh. Specifying and applying commitment-based business patterns. In *Proceedings of the 10th International Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*, Taipei, May 2011. IFAAMAS.

[17] Marco Colombetti. A commitment-based approach to agent speech acts and conversations. In *Proceedings of the Autonomous Agents Workshop on Agent Languages and Communication Policies*, pages 21–29, May 2000.

[18] R. Cost, Y. Chen, T. Finin, Y. Labrou, and Y. Peng. Modeling agent conversations with colored petri nets. In *Working Notes of the Workshop on Specifying and Implementing Conversation Policies*, pages 59–66, Seattle, Washington, May 1999.

[19] Nirmit Desai, Amit K. Chopra, Matthew Arrott, Bill Specht, and Munindar P. Singh. Engineering foreign exchange processes via commitment protocols. In *Proceedings of the 4th IEEE International Conference on Services Computing*, pages 514–521, Los Alamitos, 2007. IEEE Computer Society Press.

[20] Nirmit Desai, Amit K. Chopra, and Munindar P. Singh. Representing and reasoning about commitments in business processes. In *Proceedings of the 22nd Conference on Artificial Intelligence*, pages 1328–1333, 2007.

[21] Nirmit Desai, Amit K. Chopra, and Munindar P. Singh. Amoeba: A methodology for modeling and evolution of cross-organizational business processes. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 19(2):6:1–6:45, October 2009.

[22] Nirmit Desai, Ashok U. Mallya, Amit K. Chopra, and Munindar P. Singh. Interaction protocols as design abstractions for business processes. *IEEE Transactions on Software Engineering*, 31(12):1015–1027, December 2005.

[23] Nirmit Desai and Munindar P. Singh. On the enactability of business protocols. In *Proceedings of the 23rd Conference on Artificial Intelligence (AAAI)*, pages 1126–1131, Chicago, July 2008. AAAI Press.

[24] Hywel R. Dunn-Davies, Jim Cunningham, and Shamimabi Paurobally. Propositional statecharts for agent interaction protocols. *Electronic Notes in Theoretical Computer Science*, 134:55–75, 2005.

[25] ebBP. Electronic business extensible markup language business process specification schema v2.0.4, December 2006. docs.oasis-open.org/ebxml-bp/2.0.4/OS/.

[26] FIPA. FIPA interaction protocol specifications, 2003. FIPA: The Foundation for Intelligent Physical Agents, http://www.fipa.org/repository/ips.html.

[27] Nicoletta Fornara, Francesco Viganò, Mario Verdicchio, and Marco Colombetti. Artificial institutions: A model of institutional reality for open multiagent systems. *Artificial Intelligence and Law*, 16(1):89–105, March 2008.

[28] Martin Fowler. *UML Distilled: A Brief Guide to the Standard Object Modeling Language*. Addison-Wesley, Reading, MA, 3rd edition, 2003.

[29] Juan C. Garcia-Ojeda, Scott A. DeLoach, Robby, Walamitien H. Oyenan, and Jorge Valenzuela. O-MaSE: A customizable approach to developing multiagent processes. In *Proceedings of the 8th International Workshop on Agent Oriented Software Engineering (AOSE)*, 2007.

[30] Scott N. Gerard and Munindar P. Singh. Formalizing and verifying protocol refinements. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 2011. In press.

[31] HL7 reference information model, version 1.19. www.hl7.org/ Library/ data-model/ RIM/ C30119/ Graphics/ RIM_billboard.pdf, 2002.

[32] Kohei Honda, Nobuko Yoshida, and Marco Carbone. Multiparty asynchronous session types. In *Proceedings of the 35th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL)*, pages 273–284. ACM, 2008.

[33] http://www.gs1.org/productssolutions/gdsn/. GDSN: Global Data Synchronization Network.

[34] http://www.hitsp.org/. Hitsp: Healthcare information technology standards panel.

[35] Marc-Philippe Huget and James Odell. Representing agent interaction protocols with agent UML. In *Agent-Oriented Software Engineering V*, volume 3382 of *LNCS*, pages 16–30. Springer, 2005.

[36] Andrew J. I. Jones and Xavier Parent. A convention-based approach to agent communication languages. *Group Decision and Negotiation*, 16(2):101–141, March 2007.

[37] Andrew J. I. Jones and Marek Sergot. A formal characterisation of institutionalized power. *Journal of the IGPL*, 4(3):429–445, 1996.

[38] Elisa Marengo, Matteo Baldoni, Amit K. Chopra, Cristina Baroglio, Viviana Patti, and Munindar P. Singh. Commitments with regulations: Reasoning about safety and control in REGULA. In *Proceedings of the 10th International Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*, Taipei, May 2011. IFAAMAS.

[39] Lin Padgham and Michael Winikoff. Prometheus: A practical agent-oriented methodology. In Brian Henderson-Sellers and Paolo Giorgini, editors, *Agent-Oriented Methodologies*, chapter 5, pages 107–135. Idea Group, Hershey, PA, 2005.

[40] RosettaNet. Home page, 1998. www.rosettanet.org.

[41] John R. Searle. *Speech Acts*. Cambridge University Press, Cambridge, UK, 1969.

[42] John R. Searle. *The Construction of Social Reality*. Free Press, New York, 1995.

[43] Munindar P. Singh. Social and psychological commitments in multiagent systems. In *AAAI Fall Symposium on Knowledge and Action at Social and Organizational Levels*, pages 104–106, 1991.

[44] Munindar P. Singh. Agent communication languages: Rethinking the principles. *IEEE Computer*, 31(12):40–47, December 1998.

[45] Munindar P. Singh. An ontology for commitments in multiagent systems: Toward a unification of normative concepts. *Artificial Intelligence and Law*, 7(1):97–113, March 1999.

[46] Munindar P. Singh. A social semantics for agent communication languages. In *Proceedings of the 1999 IJCAI Workshop on Agent Communication Languages*, volume 1916 of *Lecture Notes in Artificial Intelligence*, pages 31–45, Berlin, 2000. Springer.

[47] Munindar P. Singh. Distributed enactment of multiagent workflows: Temporal logic for service composition. In *Proceedings of the 2nd International Joint Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*, pages 907–914, Melbourne, July 2003. ACM Press.

[48] Munindar P. Singh. Semantical considerations on dialectical and practical commitments. In *Proceedings of the 23rd Conference on Artificial Intelligence (AAAI)*, pages 176–181, Chicago, July 2008. AAAI Press.

[49] Munindar P. Singh. Information-driven interaction-oriented programming. In *Proceedings of the 10th International Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*, pages 491–498, Taipei, May 2011. IFAAMAS.

[50] Munindar P. Singh. LoST: Local state transfer—An architectural style for the distributed enactment of business protocols. In *Proceedings of the 7th IEEE International Conference on Web Services (ICWS)*, pages 57–64, Washington, DC, 2011. IEEE Computer Society.

[51] Munindar P. Singh, Amit K. Chopra, and Nirmit Desai. Commitment-based service-oriented architecture. *IEEE Computer*, 42(11):72–79, November 2009.

[52] Munindar P. Singh, Amit K. Chopra, Nirmit Desai, and Ashok U. Mallya. Protocols for processes: Programming in the large for open systems. *ACM SIGPLAN Notices*, 39(12):73–83, December 2004.

[53] Transaction workflow innovation standards team, February 2006. http://www.twiststandards.org.

[54] Wil M. P. van der Aalst and Maja Pesic. DecSerFlow: Towards a truly declarative service flow language. In *Proceedings of the 3rd International Workshop on Web Services and Formal Methods*, volume 4184 of *LNCS*, pages 1–23. Springer, 2006.

[55] Javier Vázquez-Salceda, Virginia Dignum, and Frank Dignum. Organizing multiagent systems. *Autonomous Agents and Multi-Agent Systems*, 11(3):307–360, 2005.

[56] Michael Winikoff, Wei Liu, and James Harland. Enhancing commitment machines. In *Proceedings of the 2nd International Workshop on Declarative Agent Languages and Technologies (DALT)*, volume 3476 of *LNAI*, pages 198–220, Berlin, 2005. Springer-Verlag.

[57] WS-CDL. Web services choreography description language version 1.0, November 2005. www.w3.org/TR/ws-cdl-10/.

[58] Daniel M. Yellin and Robert E. Strom. Protocol specifications and component adaptors. *ACM Transactions on Programming Languages and Systems*, 19(2):292–333, 1997.

[59] Pınar Yolum. Design time analysis of multiagent protocols. *Data and Knowledge Engineering Journal*, 63:137–154, 2007.

[60] Pınar Yolum and Munindar P. Singh. Flexible protocol specification and execution: Applying event calculus planning using commitments. In *Proceedings of the 1st International Joint Conference on Autonomous Agents and MultiAgent Systems*, pages 527–534. ACM Press, July 2002.

Subject Index