Agent Communication

Amit K. Chopra and Munindar P. Singh

University of Trento

North Carolina State University

May 23, 2012

MAS as Distributed Systems

- Agents
  - Autonomous: independently acting
  - Heterogeneous: independently designed
- Agents communicate with each other
  - Protocols define how the agents ought to communicate with one another
    - A protocol is a modular, potentially reusable specification of the interactions between two or more entities
    - Defining a protocol helps ensure *interoperability*, i.e., being able to work together
  - Communities of practice define appropriate protocols
    - RosettaNet: manufacturing
    - Foreign exchange transactions: TWIST
    - Health care: HL7

Exercise

Identify the agents and communications (including protocols) involved in the specific setting of consumer-to-consumer auctions

Objectives of this Chapter

Study the key conceptual underpinnings of agent communication

- What are the main requirements for protocol specifications?
- How can we specify a communication protocol?
- Which way is the field headed?

Traditional Distributed Computing

- Ignore autonomy and heterogeneity
- Specify interaction in low-level operational terms via message order and occurrence
- Specify interoperation in low-level terms
- A system may be fragile because of its interoperation depending upon low-level details that can easily change when one of the parties modifies its internals

Autonomy

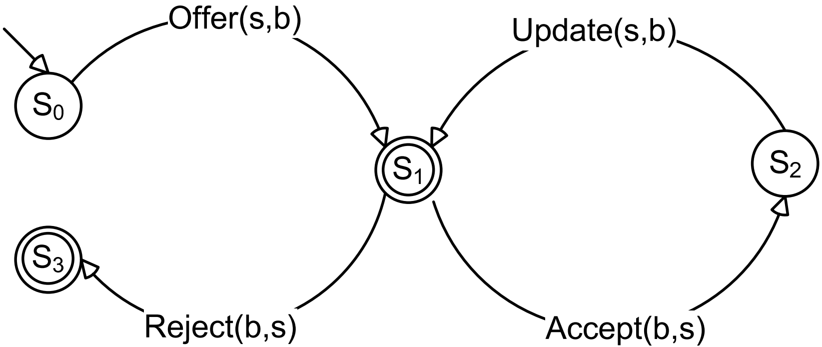
- Each agent is free to act as it pleases
  - We must design protocols so that they do not over-constrain an agent's interactions
  - Intelligence is irrelevant in a protocol: must design a protocol whose correctness does not depend upon the agents' internal reasoning
- The agents are the logical units of distribution
  - Physical distribution is based on considerations such as geographical distribution, throughput, redundancy
  - Cannot treat two or more agents as a single operating system process, even though that's how they may be realized, e.g., within the same virtual machine in an agent platform

Heterogeneity

- In traditional systems, it is enough that protocols specify the
  - Schemas of the messages exchanged
  - Legal flows, that is, their ordering and occurrence
- In multiagent systems, protocols must specify the meaning of the messages
  - Logically, agents interoperate on the basis of meanings of their communications
    - Since the meanings determine their *social state*, i.e., state of their interaction
- Whatever is in the protocol
  - Becomes the standard to which agents are implemented
  - Defines the level of heterogeneity: the agents can be heterogeneous with regard to everything else
  - Giving prominence to low-level concerns (such as ordering and occurrence of messages) couples the agent designs at the corresponding low level
    - Even though such concerns are appropriate for lower levels of the implementation

Example Finite State Machine Representation

Part of a purchase protocol that deals with making offers



- Roles: buyer (b) and seller (s)
- Transitions labeled with messages
  - Specify legal message flows

Critique of the FSM Representation

- The FSM specification does not account for meanings of messages
- Implicit meanings can cause violation of interoperability because the parties may interpret messages differently
- Designers agree offline regarding the meanings, thereby limiting the heterogeneity of their agents

Criteria for Evaluating Protocols

- Software engineering of systems: Use representations close to stakeholder requirements
- Flexibility of agents
- Compliance checking of an agent with a protocol

Communicative Act Theory

Speech act theory in philosophy

- Communication is a form of action
  - Goes beyond traditional logic, which deals with assertions (true or false)
- Canonical example: when a judge declares a couple married, the judge
  - Does not merely report on some privately or publicly known fact
  - Brings the fact into existence
  - Assumption: the judge has suitable powers and acts autonomously
- The above is an example of a *declarative*

Performatives: 1

All communications can be expressed as declaratives

- Informatives
  - “the shipment will arrive on Wednesday” maps to
  - “I inform you that the shipment will arrive on Wednesday”
- Directives
  - “send me the goods” maps to
  - “I request that you send me the goods”
- Commissives
  - “I’ll pay you $5” maps to
  - “I promise that I’ll pay you $5”

Related to Multiagent Systems

- Emphasizes autonomy of the sending agent (speaker)
  - May not control the real world
  - But controls when it informs, requests, promises, ...
- The performative provides type information on a communication separately from its content
- Consider the proposition "the door is open"
  - "I inform that" + "the door is open"
  - "I request that" + "the door is open"
  - "I promise that" + "the door is open"
- That is, we see a modular structure separating types from the content

Agent Communication Primitives

- Customary to consider a small set of primitives based on the performative types
  - KQML, FIPA ACL, and the lesser known languages do so (with small variations)
  - Give a unique meaning for the types (sometimes only informally)
- The above proves problematic
  - MAS applications are diverse
  - The standard, broad-brush meaning is rarely adequate
  - Developers build in additional layers of meaning but leave it undocumented
- Dispense with a fixed set of primitives
  - Define application-specific primitives
  - Provide suitable meaning based on social state primitives such as commitments

Traditional Software Engineering Approaches

- Emphasize operational details, mostly concentrating on the occurrence and ordering of messages
  - Leave open the formulation of the message syntax (good)
  - Disregard the meanings of the messages (bad)
- Traditional representations capture occurrence and ordering of messages, mostly in procedural terms
  - Finite state machines (procedural)
  - Petri nets (procedural)
  - State diagrams or statecharts (procedural)
  - Pi-calculus (procedural)
  - Temporal logic (declarative)

Traditional Software Engineering Tradeoffs

- Benefits
  - Formal tools for verification
  - Natural to implement agents who satisfy protocol requirements
  - Easy to check compliance
- Shortcomings
  - No account of meaning
  - No application-centric standard of correctness
  - No support for flexibility based on meanings

Choreography

A specification of the message flow among the participants from a neutral perspective

- Benefits
  - Decentralized nature agrees with the MAS way of thinking
- Current approaches: WS-CDL and ebBP
- Shortcomings of current approaches
  - No encoding of the meaning
  - Focus on ordering and occurrence
  - Makes private actions of agents visible
  - No support for composition of choreographies

Sequence Diagrams

Used by FIPA (Foundation for Intelligent Physical Agents)

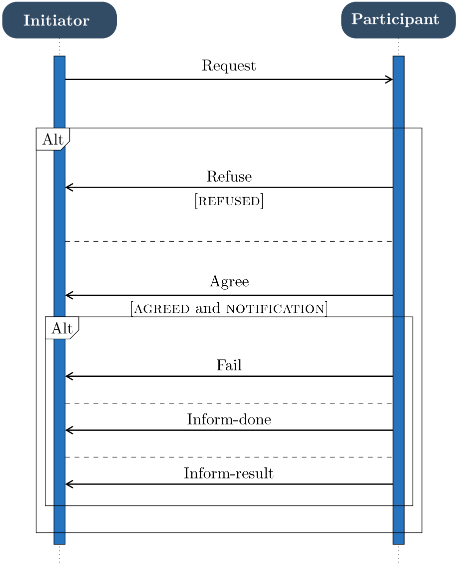
Also known as Message Sequence Charts (MSCs)

- Procedural constructs: sequencing (default), alternative, parallel, loop
- FIPA uses UML Sequence Diagrams to specify its interaction protocols
- FIPA added constructs that have subsequently become part of the UML 2.0 standard

FIPA Request Interaction Protocol

- Roles: INITIATOR and PARTICIPANT
- The INITIATOR sends a *request* to the PARTICIPANT
- The PARTICIPANT either responds with a *refuse* or an *agree*.
- If it agrees, 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

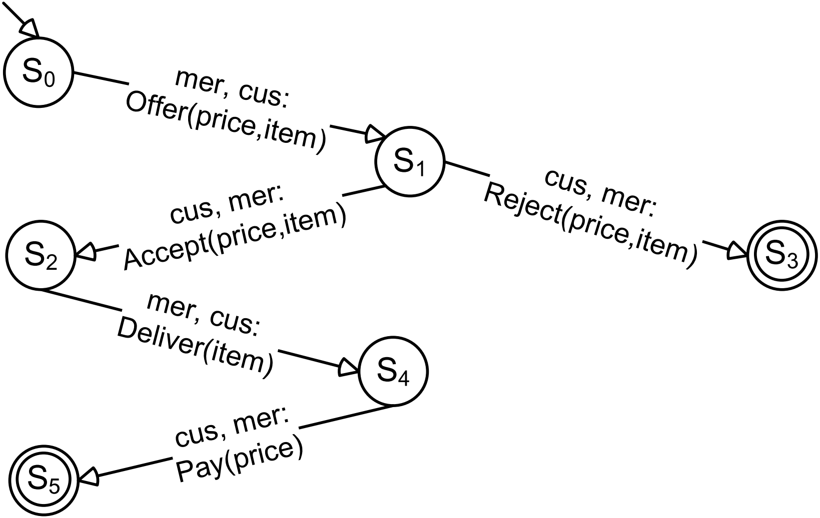
FIPA Request Interaction Protocol



FIPA Request Interaction Protocol

- Highlights benefits of a protocol
  - Clear roles
  - Decouples agents from one another
- Ignores meanings specific to the protocol
  - FIPA offers a semantics for the message types that we review below

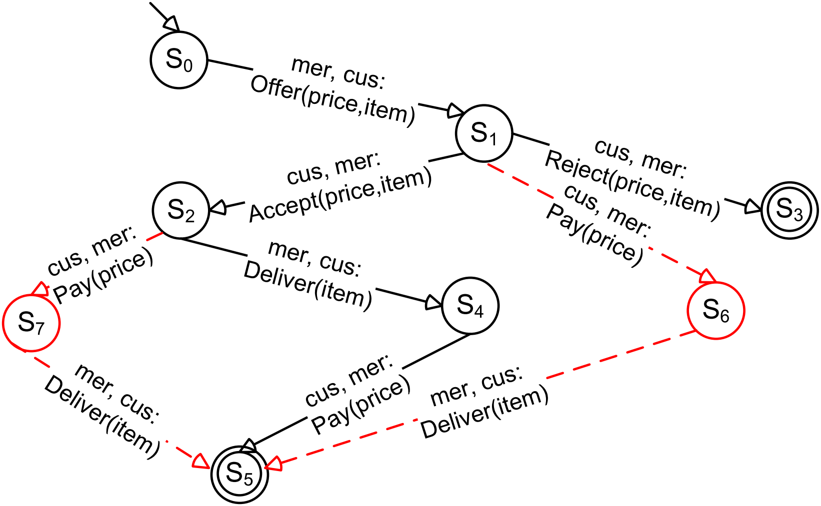
State Machine Example: 1



State Machine Example: 2

- Roles: merchant (mer) and customer (cus)
- Transitions: labeled with messages as sender, receiver
- No representation of internal decision policies: hence it describes a protocol
- Two executions
  - The customer rejects the merchant's offer
  - The customer accepts the merchant's offer, following which the merchant and the customer exchange the item and the payment for the item
- Shortcomings
  - Syntactic notion of correctness
  - Omits additional paths (next picture) that are equally reasonable

State Machine Example: 3



Produce Larger FSMs?

Can we not use FSMs to capture all reasonable paths?

- Producing ever-larger FSMs with additional paths
  - Complicates the agent implementation
  - Does not support runtime flexibility
  - Presupposes an arbitrary selection of paths: which path is reasonable, which is not?
- The same argument holds against merely expanding declaratively-specified—though conceptually low-level protocols
  - For example, those specified using temporal logic

Evaluation with Respect to MAS

- Software engineering: low-level abstractions
- Flexibility: limited because of the protocols tending to over-specify message order and occurrence
- Compliance checking: easy since the protocol is explicit about message order and occurrence but failure to comply may not indicate an application-level problem

AI Approaches

Human assisting tools

- Based on work on tools for assisting humans
  - Human-computer interaction
  - Natural language understanding for helping users
- Assume cooperative settings, based on the above
  - Seek to infer what the user wants
  - Assume the user wants to be helped
- Give prominence to mental or cognitive concepts
  - Model the user's cognitive state
  - Project a cognitive state to the user

AI Approaches

Distributed knowledge-based systems

- Expert systems that communicate with each other
- Leading to agents with a reasoner and a knowledge base
- All the agents would be built by the same party
  - Cooperative
  - Not quite autonomous
  - Largely homogeneous, although potentially with different reasoning rules and knowledge

KQML: Knowledge Query and Manipulation Language

- Underlying assumptions
  - Each agent maintains a knowledge (belief) base or KB
  - The agents are cooperative, sincere, credulous
  - Beliefs provide an abstraction over the implementation details of agents
- The name reflects a control perspective
  - An agent cannot query the knowledge of another
  - Much less manipulate it
- Provides a small set of primitives, each defined in relation to the agents' KBs
  - *tell*: sender takes some beliefs from its KB and tells another; receiver inserts the received beliefs into its KB
  - *query*: receiver responds with a *tell* of the query result
- Evaluation
  - KQML doesn't provide a basis for choosing among the message types
  - Most times, developers would use *tell* and encode (in an ad hoc way) the necessary information within the body of the *tell*
  - The above led to reduced interoperability because the semantics offered by the language had no value as such to a MAS

FIPA ACL

Agent Communication Language

- Provides primitives for message types along with their syntax
- States the semantics of each primitive
  - In terms of the beliefs and intentions of the participants
  - Including their beliefs and intentions about each other's beliefs and intentions
  - That is, incorporating assumptions of sincerity and cooperation

Evaluating Cognitive Concepts for Communication

- Cognitive concepts provide a natural way to capture the internal representation and reasoning of an agent
  - Good way to capture stakeholder wishes
  - High-level way of describing agent reasoning independent of low-level details of data structures and such
- Cognitive concepts cannot be used as a basis for interoperation, which is what communication is about
  - Internally focused
  - One designer cannot determine the beliefs or intentions of another designer's agents
    - Without making unrealistic assumptions, e.g., one designer controls all designs, thereby abolishing heterogeneity
  - One agent cannot determine another agent's beliefs or intentions
    - Without making unrealistic assumptions, e.g., abolishing autonomy and heterogeneity

FIPA Evaluated

Split personality

- Practically valuable
  - Discussion of multiagent architecture and interoperations
  - Implementation of powerful systems, such as JADE
  - Description (though limited in style and scope) of useful interaction protocols
- Nonsense
  - Misguided, cognitive approach to formal semantics
  - Irrelevant assumptions
  - *Never* used (fortunately)
- What we should do: discard the second and strengthen the first

AI Approaches Evaluated

- Software engineering:
  - High-level abstractions are a positive
  - Mentalism in the abstractions is a negative
- Flexibility: curtailed through the assumptions underlying the semantics
  - In FIPA, to inform another agent the sender must believe the receiver doesn't already know the content
- Compliance: impossible under mentalism

Commitment-Based Multiagent Approaches

Give primacy to business meanings of service engagements

- Identify messages
- Identify their meanings in terms of their effect on the social state
  - Creation of the commitments among the participants
  - Manipulation of commitments
  - Changes to parts of the state relevant to commitments
- Instead of explicit state transitions, consider inference on the social state based on the messages

Example: Commitment Progression

Via explicit operations or because of logical properties

C(Buyer, Seller, goods, pay) signifies an active and conditional commitment

- If *goods* $\wedge$ C(Buyer, Seller, goods, pay) Then
  - Active and detached (or unconditional or base)
  - C(Buyer, Seller, T, pay)
- If C(Buyer, Seller, T, pay) Then
  - If *pay* Then Satisfied
  - If never *pay* Then Violated
- If C(Buyer, Seller, goods, pay) Then
  - If *pay* Then Satisfied
  - If never *pay* and never *goods* Then Expired

Can be nested:
C(Seller, Buyer, pay, C(Shipper, Buyer, T, deliverGoods))

Example Commitment Protocol

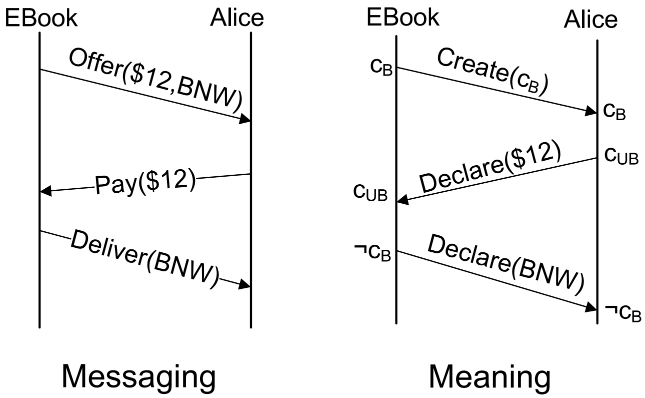
Purely declarative specification

$Offer(mer, cus, price, item)$ means $\textsc{create}(mer, cus, price, item)$
$Accept(cus, mer, price, item)$ means $\textsc{create}(cus, mer, item, price)$
$Reject(cus, mer, price, item)$ means $\textsc{release}(mer, cus, price, item)$
$Deliver(mer, cus, item)$ means $\textsc{declare}(mer, cus, item)$
$Pay(cus, mer, price)$ means $\textsc{declare}(cus, mer, price)$

- Specifies how each message affects the social state
  - By acting on a commitment explicitly
  - By bringing about a social fact via DECLARE that may cause commitments to detach or discharge
- The social state is conceptual
- In general, no centralized store of social state
  - Raises the challenge of commitment alignment in distributed systems

Distinguishing Message Syntax and Meaning

Two views of the same enactment

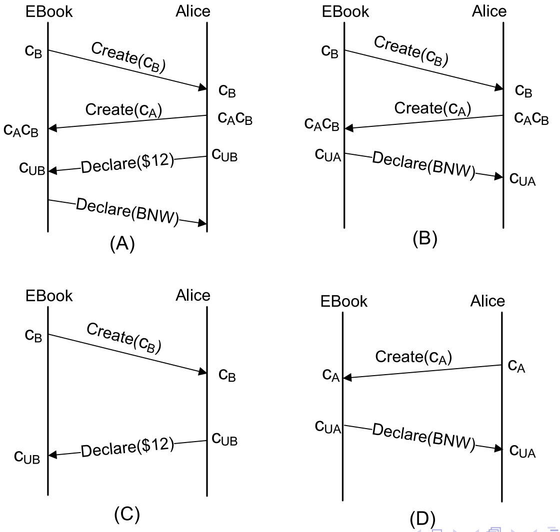


Evaluation with respect to MAS

- Compliance: At the business level. A protocol enactment is correct as long as the parties involved do not violate their commitments
- Flexibility: Enhanced by expanding the operational choices for each party, e.g., discharge a commitment when convenient (even sooner); delegate or assign
- Software engineering: Commitments are a high-level abstraction for capturing business interactions
  - Support loose coupling among agents
  - Accommodate the autonomy of each participant

Illustrating Flexible Enactment

These are compliant executions in terms of commitments, and thus realize the above protocol



Comparing 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

Engineering with Agent Communication

- Beginning from a protocol
- Generate role skeletons (or endpoints) from the protocol
- Challenge: Generating role skeletons such that implementing agents ensures interoperation
  - Not trivial when a protocol involves more than two roles
  - The protocol must be such that such skeletons are derivable from it
- For each role skeleton, implement one or more agents who realize ("flesh out") it
  - Map each skeleton to a set of incoming and outgoing messages and the changes each message induces in the local state
  - Implement methods to process each incoming message
  - Send messages allowed by the protocol

Programming with Communications

Java Agent Development Framework or JADE is a leading platform

- Behavior: a specification of a role skeleton that characterizes important events such as the receipt of specified messages and the occurrence of timeouts
- Implement an agent according to a behavior by defining the methods it specifies as callbacks
  - Define the handlers for any incoming methods

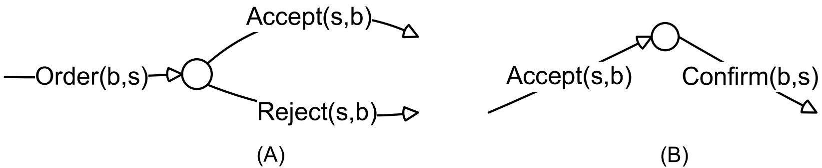
Modeling Communications

Coming up with the right specifications

- Need for a methodology to elicit stakeholder requirements
- We advocate a pattern-based approach
- Operational patterns are easy but often trivial and miss business meanings
- Commitment-based *business* patterns help capture ways in which agents may interact at a high level

Example Operational Patterns

Patterns such as these can help a designer in specifying a protocol



Motivating Principles Behind our Patterns

- Autonomy compatibility: no agent controls another's actions
- Explicit meanings: The business meaning ought to be public and explicit
  - Not hidden within implementations
  - Not hidden within offline agreements between designers

Patterns

Encode the common ways in which agents interact

- Business patterns: what relationships to express
- Enactment patterns: the conditions under which an agent should enact a business pattern
- Semantic antipatterns: the relationships antithetical to our principles
- We write each type of pattern in a template specific to that type

Compensation

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

Motivation Compensation commitments provides 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 $Compensate(mer, cus, price, item, discount)$, i.e., the merchant will offer the customer a discount on the next purchase if the item is paid for but not delivered

Consequences The only recourse a creditor may have is escalation to the surrounding business *context* such as the applicable jurisdiction

Counter Offer

Example of an enactment pattern

Intent Responding to an offer via an offer

Motivation Supporting negotiation

When
- Original offer: $\mathsf{C}(x, y, r, u)$
- Counter offer: $\mathsf{C}(y, x, u', r')$
  - Flips debtor and creditor and antecedent and consequent
  - Antecedent is stronger than original consequent
  - Consequent is weaker than original antecedent
- Alternative counter offer: above plus $Release(x, y, r, u)$

Example Assume $\mathsf{C}(EBook, Alice, \$12, BNW)$
- Alice makes the counter offer $\mathsf{C}(Alice, EBook, BNW \wedge Dune, \$12)$ 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 results in a mutual commitment

Semantic Antipatterns

Forms of representation and reasoning to be avoided

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 where the delegator holds a power over the delegatee

Implementation The sender of Create(y, z, p, q) is x ($x \neq y$), thus contravening the autonomy of y

Example EBook makes an offer *Create*(*BookWorld*, *Alice*, $12, *BNW*) to Alice, which violates BookWorld's autonomy

Criteria Failed The debtor's autonomy is not respected

Consequences Calls into question the idea of modeling with agents

Alternative Apply delegation to achieve the desired business relationship, based on prior commitments

- 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

Communication-Based Engineering Methodologies

How to design a protocol

- Identify stakeholder requirements
- Identify the roles involved
  - customer, merchant, shipper, and banker
- If possible, select a suitable protocol from a repository
  - The purchase protocol shown earlier
- Otherwise, compose existing protocols if possible
  - Compose the *Ordering*, *Payment*, and *Shipping* protocols
- Otherwise, specify a protocol or parts of it from scratch
  - Identify the communications among the roles
    - Messages for ordering items and messages for payment
- Identify how the messages affect commitments
  - *Offer* could create a commitment, as shown earlier
  - A delivery by the shipper would discharge the merchant's commitment to provide the goods

Primacy of Meaning

Understand agent communication in terms of the participants' *social state*

- Helps avoid inadvertent dependencies upon implementation and yields flexibility
- Older meaning-based work combines meanings and operational details on message ordering and occurrence
  - Operational details interfere with reasoning about meaning
- No compelling natural situation where operational details, outside of commitments, are necessary
  - Occurrence of a message: requiring an agent to send a message violates its autonomy—it may choose to violate its commitments, for example
  - Nonoccurrence of a message: where it is necessary for integrity, we should model it via commitments
  - Ordering messages for conventions: reasonable and should be encoded within the antecedents and consequents of commitments
  - Ordering messages otherwise: almost never useful and merely included just by habit
- The Blindingly Simple Protocol Language declaratively captures the necessary operational details, facilitating assertions about social state

Verifying Compliance

Each protocol functions as a small standard

- Agents must be able to judge if their counterparties are interacting as codified in their agreed upon protocol
- Worthless otherwise
- The mentalist approaches preclude such verification
- Despite long research on this point, several researchers return to mentalism repeatedly
- Challenges
  - Design specification languages that promote the verification of compliance
  - Develop algorithms by which one or more cooperating agents could verify the compliance of others based on the communications they can monitor

Protocol Refinement and Aggregation

Apply traditional conceptual modeling relations to communication

- Refinement: how a concept refines another (*is-a* hierarchy)
- Aggregation: how concepts are put together into composites (*part-whole* hierarchy)
- Well-understood for traditional object-oriented design and supported by programming languages (as type checking)
- Nontrivial for communication protocols (especially, refinement)
- Challenge: produce a generalized theory and associated languages and tools that would support refinement and aggregation of protocols for more powerful meaning specifications

Role Conformance

Developing an agent that conforms to a role specification

- Produce a role skeleton from a protocol specification
- An agent who plays (and hence implements) a role fleshes out the skeleton
  - A challenge is to determine sufficient constraints on messages an agent playing a role can receive and send and any constraints on how the local representation of the social state should progress
  - We can then publish role skeletons along with the protocol specification
- Software vendors produce agent implementations
- An agent vendor does not reveal internal details but specifies what roles the agent can play
- Conformance means that an agent can play a particular protocol role
- Challenge: identifying formal languages for specifying roles along with algorithms for checking conformance

Conclusions

Communication lies at the heart of multiagent systems

- Autonomous agents depend on each other, i.e., interoperate, to realize important real-world applications
- A good multiagent system must be loosely coupled; communication is the highly elastic glue that keeps it together

Digging Deeper

Relevant topics to explore further

- Philosophical foundations
- Organizations and institutions
- Norms, conventions, and commitments
- Software engineering