Specification and verification of agent interaction protocols

WOA 2005

Alberto Martelli
Dipartimento di Informatica
Università di Torino

COFIN

Sviluppo e verifica di sistemi multiagente basati sulla logica (MASSIVE)

20/11/2003 – 20/11/2005

Coordinatore: Alberto Martelli

Partecipanti:

Alessandria: Laura Giordano

Bologna: Paola Mello

Torino: Alberto Martelli

Obiettivo della ricerca:

individuare formalismi, strumenti e tecniche per modellare, sviluppare e analizzare sistemi ad agenti.

• Modellare la dinamica interna al singolo agente, con le sue credenze, i suoi obiettivi le sue capacità di ragionamento.

• Modellare la dinamica dell'interazione e delle comunicazione fra agenti in un ambiente che evolve dinamicamente.

L'attività di ricerca ha riguardato principalmente:

1. Modelli formali di sistemi ad agente

Sono state studiate tecniche basate sulla logica computazionale per la descrizione del comportamento individuale e collettivo di agenti che interagiscono in società aperte. E' stato definito un modello di società basato su logica computazionale nel quale ogni protocollo di interazione può essere definito da un insieme di vincoli logici abduttivi.

E' stata proposta una teoria delle azioni basata sulla logica DLTL per modellare la comunicazione tra agenti. La logica DLTL combina la logica temporale "linear time" con la logica dinamica, usata per modellare azioni complesse (programmi regolari).

2. Linguaggi di specifica basati sulla logica computazionale

Al formalismo per modellare società aperte è stata associata una controparte operazionale, ottenuta come estensione della procedura logica abduttiva IIF(Fung e Kowalski) che consente la verifica delle interazioni fra agenti della società.

E' stata completata la definizione di un linguaggio di programmazione logica per ragionare su azioni, DyLOG, basato su un approccio modale, estendendo il linguaggio con azioni comunicative.

E' stata studiata l'integrazione di DyLOG con l'ambiente di prototipazione DcaseLP.

3. Verifica

Sono stati affrontati diversi problemi di verifica di proprietà di agenti e di protocolli di comunicazione, mediante i formalismi citati nei punti precedenti.

I problemi considerati riguardano: proprietà di protocolli, comportamento corretto di agenti comunicanti, correttezza dell'implementazione di un singolo agente, interoperabilità di un insieme di agenti.

Le soluzioni proposte utilizzano varie tecniche, come il ragionamento abduttivo o il *model checking*.

Le attività di ricerca nell'ultima fase del progetto si sono orientate principalmente verso le problematiche relative alla specifica formale e verifica di proprietà di protocolli, prendendo in considerazione settori applicativi come le linee guida mediche o i servizi web.

Nel seguito verrà dato un inquadramento generale delle problematiche della specifica e verifica di protocolli di comunicazione fra agenti.

Gli approcci specifici adottati nel progetto verranno descritti negli interventi seguenti.

Multi-agent systems

A **multiagent system** contains a number of agents which interact with one another through **communication**.

Multiagent environments have the following characteristics:

- they provide an infrastructure specifying communication and interaction protocols;
- they are typically open and have no centralized designer;
- they contain agents which are autonomous and distributed, and may be self-interested or cooperative.

Agents communicate in order to better achieve the goals of themselves or of the society/system in which they exist.

Agent Communication Languages (ACL)

An ACL provides agents with a means of exchanging information and knowledge.

ACLs stand usually at a higher level with respect to the communication tools of distributed systems, such as remote procedure call or method invocation.

ACLs handle propositions, rules, and actions instead of simple objects

An ACL message describes a desired state in a declarative language, often in terms of mental attitudes.

KQML

Knowledge Query and Manipulation Language

KQML is a high-level communication language independent of:

- the transport mechanism (tcp/ip, RMI, …)
- the content language (KIF, Prolog, …)
- the ontology

A KQML message specifies the type of message (performative)

KQML ignores the content portion of a message.

KQML messages

The syntax of a KQML message is based on Lisp-like lists, and consists of a *performative* followed by a number of keyword/value pairs (syntax is not important).

Example (a query from Joe about the price of a share of IBM):

```
(ask-one                                   ← performative
     :sender   joe
     :receiver   stock-server
     :reply-with   ibm-stock
     :language   LPROLOG                   ← the representation language of the content
     :content   (PRICE IBM ?price)
     :ontology   NYSE-TICKS
)
```

Some KQML performatives

achieve	S wants R to make something true
advertise	S claims to be suited to processing a performative
ask-one	S wants one of R's answers to question C
ask-all	S wants all of R's answers to question C
reply	communicates an expected reply
sorry	S cannot provide a more informative reply
tell	S informs R that it knows C

S: sender R: receiver C: content

FIPA ACL

The Foundation for Intelligent Physical Agents (FIPA) was formed in 1996 to produce software standards for heterogeneous and interacting agents and agent-based systems.

FIPA operates through the open international collaboration of member organizations, companies, research centers and universities.

Among other specifications, FIPA has defined an agent communication language similar to KQML.

FIPA ACL provides 22 communicative acts, like inform, request, agree, query-if, …

The syntax for FIPA ACL messages closely resembles that of KQML:

```
(inform
      :sender     agent1
      :receiver  agent2
      :language Prolog
      :content   "weather(today, raining)"
)
```

However the semantics of the two languages are rather different.

FIPA ACL does not include the facilitation primitives of KQML.

Semantics of speech acts

Speech acts can be considered as actions.

They can be modeled by using techniques for reasoning about actions and change developed in AI, in particular in planning research.

The usual approach is to describe an action by means of its preconditions and effects.

Modeling speech acts

Cohen and Perrault gave an account of the semantics of speech acts by using techniques developed in AI planning research. In particular they used a STRIPS-like notation, by means of preconditions and effects of actions on a mental state.

Request(S, H, α)

Preconditions: (S BELIEVE (H CANDO α)) $\wedge$
(S BELIEVE (H BELIEVE (H CANDO α)))

Effects: (H BELIEVE (S BELIEVE (S WANT α)))

Successful completion of the *Request* ensures that the hearer is aware of the speaker's desires, but does not guarantee that action α will actually be performed.

Semantics of KQML

Initially KQML had no formal semantics.

A semantics was provided later on by Labrou and Finin in terms of *preconditions*, *postconditions*, and *completion conditions*.

Given a sender **A** and a receiver **B**, *preconditions* indicate the necessary state for A to send a performative, **Pre(A)**, and for B to accept and successfully process it, **Pre(B)**.

Postconditions describe the states of **A** after utterance of a performative, **Post(A)**, and of **B** after receipt of a message, **Post(B)**.

A *completion condition* for a performative indicates the final state, after, for example, a conversation has taken place.

Preconditions, postconditions and completion conditions describe states of agents in a language of mental attitudes (belief, knowledge, desire, and intention) and action descriptions (for sending and processing a message).

No semantic models for the mental attitudes are provided.

Semantics of *tell(A, B, X)*

Pre(A): BEL(A, X) $\wedge$ KNOW(A, WANT(B, KNOW(B,S)))

Pre(B): INT(B, KNOW(B, S))

where S may be any of BEL(B, X) or $\neg$BEL(B, X)

Post(A): KNOW(A, KNOW(B, BEL(A, X)))

Post(B): KNOW(B, BEL(A, X))

Completion: KNOW(B, BEL(A, X))

An agent cannot offer unsolicited information. A *proactive tell* might have **Pre(A):** BEL(A, X) and empty **Pre(B)**.

Semantics of FIPA ACL

The Semantic Language (SL) is the formal language used to define FIPA ACL's semantics. It is mainly due to Sadek.

SL is a quantified multimodal logic with modal operators:

- $B_i\varphi$ i believes that φ
- $U_i\varphi$ i is uncertain about φ but thinks that φ is more likely than $\neg\varphi$
- $C_i\varphi$ i desires (choice, goal) that φ currently holds

To enable reasoning about actions, the universe of discourse involves sequences of events (actions).

The following operators are introduced for reasoning about actions:

- Feasible(a, φ) a can take place and if it does φ will be true after that
- Done(a, φ) a has just taken place and φ was true just before that
- Agent(i, a) i is the only agent that ever performs action a

From belief, choice and events, the concept of *persistent goal* $PG_i\varphi$ can be defined. *Intention* $I_i\varphi$ is defined as a persistent goal imposing the agent to act.

The semantics of a communicative act is specified as sets of SL formulas that describe the act's *feasibility preconditions* and *rational effects*.

Feasibility preconditions (FP): conditions that must hold for the sender to properly perform the communicative act

Rational effects (RE): the effects that an agent can expect to occur as a result of performing the action (the reasons for which the act is selected). The receiving agent is not required to ensure that the expected effect comes about.

Conformance with the FIPA ACL means that when agent A sends communicative act *c*, the *FP(c)* for A must hold. The unguaranteed *RE(c)* is irrelevant to the conformance issue.

<i, INFORM(j, φ)>

FP: $B_i\varphi \wedge \neg B_i(Bif_j\varphi \vee Uif_j\varphi)$

RE: $B_j\varphi$

FP means that *i* must believe φ, and *i* must not believe that *j* already knows φ or ¬φ, or *j* is uncertain about φ or ¬φ.

<i, REQUEST(j, a)>

FP: $FP(a)[i\backslash j] \wedge B_i\ Agent(j,a) \wedge B_i \neg PG_j\ Done(a)$

RE: Done(a)

where FP(a)[i\j] denotes the part of the FPs of *a* which are mental attitudes of *i*.

Most of the other communicative actions are derived from INFORM and REQUEST. For instance:

<i, QUERY-IF(j, φ)> $\stackrel{\text{def}}{=}$

<i, REQUEST(j, <j, INFORM-IF(i, φ)>)>

<i, INFORM-IF(j, φ)> $\stackrel{\text{def}}{=}$

<i, INFORM(j, φ)> | <i, INFORM(j, ¬φ)>

Social semantics

The above semantic definitions constitute the *mental* approach, because they define semantics of speech acts in terms of the mental states of participants. Using mental states to define speech acts may be adequate on cooperative multiagent systems, but presents some problems when the multiagent system is composed of competitive, heterogeneous agents. In this case it is impossible to trust other agents completely or to make strong assumptions about their internal way of reasoning.

The *social* approach, instead, considers the social consequences of performing speech acts. The approach recognizes that communication is inherently public, and thus depends on the agent's social context.

This approach is based on *commitments* between agents: an agent (the *debtor*) is committed to another agent (the *creditor*) to make some fact true or to carry out some action.

According to the social approach, the various speech acts can be seen in terms of the social commitments the participants are entering. This is obvious for an act like a promise, where a commitment is explicitly made, but holds also for other speech acts. For instance, in an assertion, the speaker is committed to the truth of the proposition.

Using the mental approach it is very difficult to verify the compliance of an agent with the semantics of speech acts. How can we show that an agent believes what it says if it is not a BDI agent?

Communication in the social approach is inherently public.

Singh was probably the first to clearly emphasize the need to define the semantics of ACLs in terms of "social notions". He proposed a social semantics based on the views of the philosopher Habermas.

He proposed three level of semantics for each act. For instance, by informing j that p, i gets committed towards j that p holds (objective claim), that he believes that p (subjective claim), and that ha has reasons to believe p (practical claim). Singh admits the mentalistic approach at the subjective level, but embedded within a social attitude (the claims that leads to a commitment).

Technically, the semantics of commitments is expressed in a branching-time logic (CTL).

Colombetti has proposed an approach in the same vein as Singh.

There can be various types of commitments:

C(a, b, p) a is committed to b that p (p can be a fact or an action)

CC(a, b, p, q) *conditional commitment:* if q holds then C(a, b, p)

PC(a, b, p) *precommitment*: is a kind of conditional commitment (e.g. a request pre-commits the agent to which it is addressed, meaning that this agent will be committed in case of acceptance).

For instance, execution of the communicative action

inform(a, b, p)

creates a commitment CC(a, b, p)

Execution of

request(a, b, p)

creates a precommitment PC(b, a, p)

If b replies with an *accept*, the precommitment is transformed in an active commitment,

if b replies with *reject*, the precommitment is cancelled.

To summarize:

Communicative actions can be modeled by means of preconditions and effects on a state.

The state can be a mental state, defined in terms of BDI attitudes, or a social state, defined in terms of commitments and social facts.

Formalization of speech acts requires the use of time (temporal logics or temporal constraints). In fact, the semantics of both mental attitudes and commitments refers to time (eventually the intention will be satisfied, eventually the commitment will be fulfilled).

Social semantics is more suitable for verifying compliance with the semantics in the case of open systems, where internal states of agents are inaccessible.

Protocols

Agents cannot take part in a dialogue simply by exchanging ACL messages.

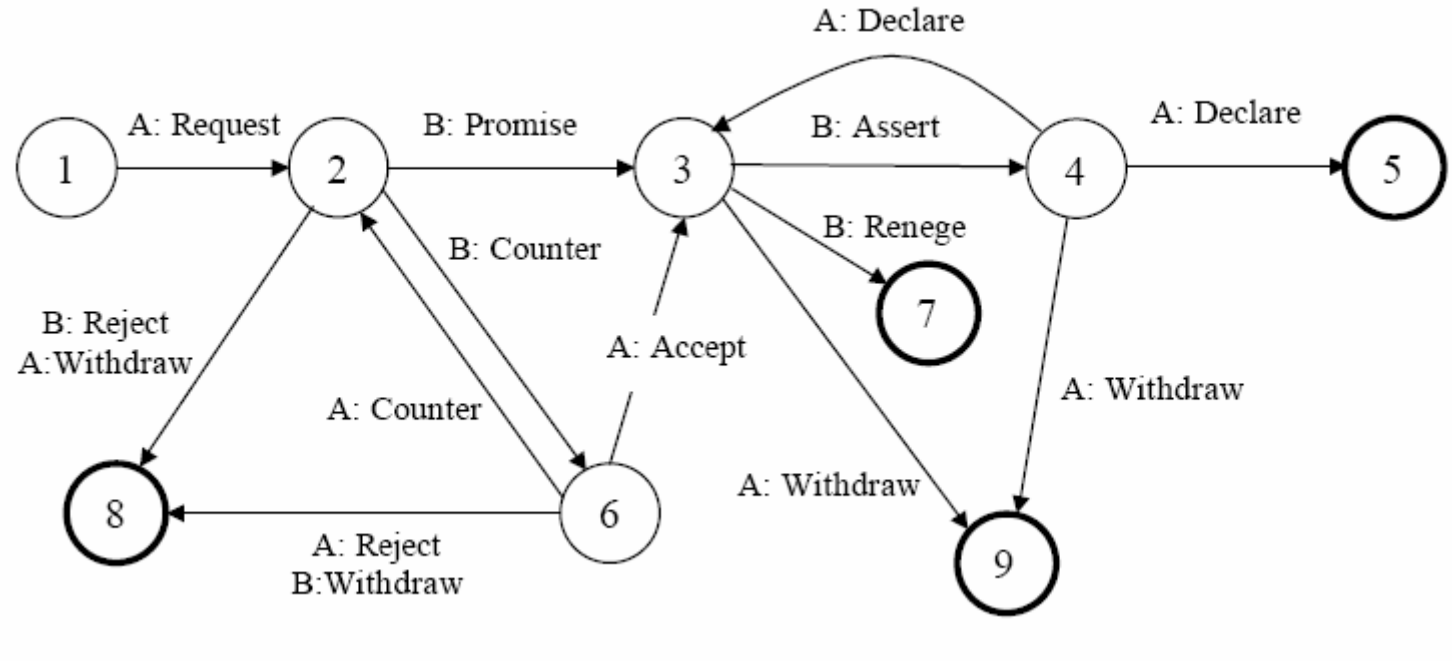
Analysis of many human *conversations* shows that there is often a pattern which frequently occurring conversations follow (example, phone calls).

The mentalistic semantics of communicative acts is too complex to determine the possible answer to a message by just reasoning on mental states.

An agent must implement tractable decision procedures that allow it to select and produce ACL messages that are appropriate to its intentions: **conversation policies** or **protocols**.

Protocol specification

Usually protocols are modeled as finite state machines.



Request for action protocol

Other formalisms have been proposed for protocol specification:

Petri nets

Definite Clause Grammars (DCG): have been used by Labrou and Fining for KQML protocol specification. DCGs are extensions of Context Free Grammars where non-terminals may be compound terms, and the body of a rule may contain procedural attachments.

AUML: FIPA specifications define protocols by means of AUML, an extension of UML for agents.

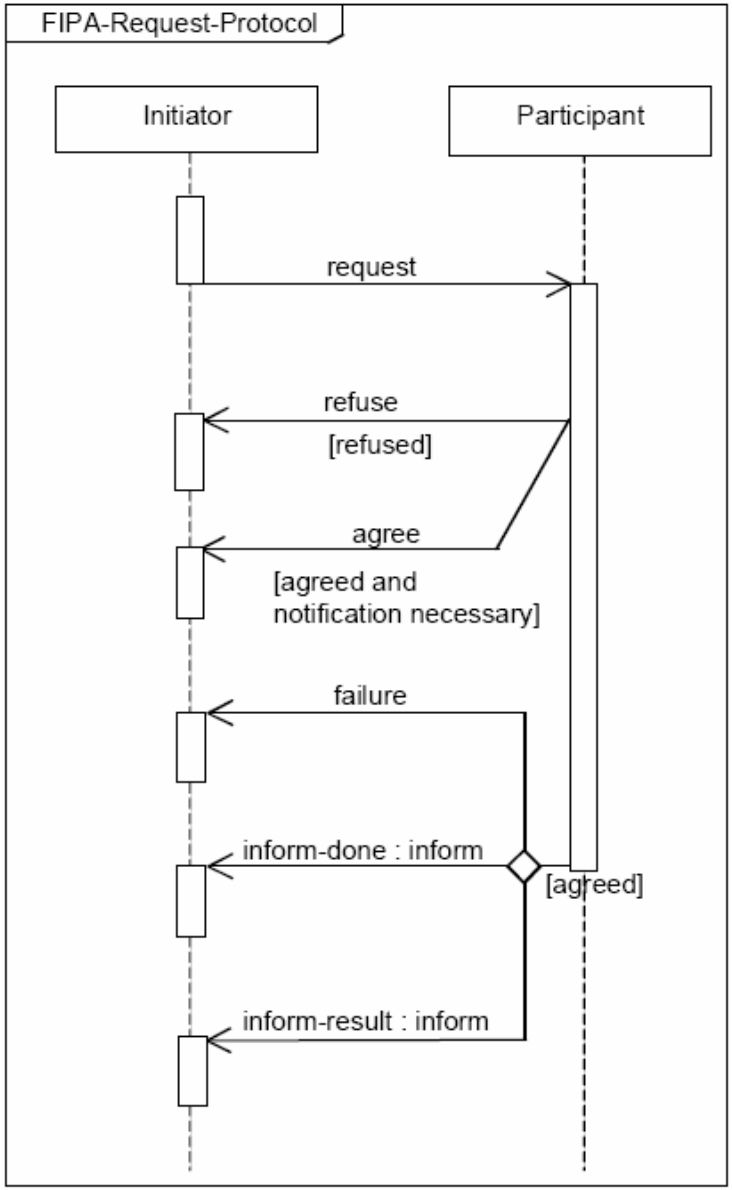


Figure 1: FIPA Request Interaction Protocol

Contract Net protocol

Many protocols have been defined for cooperation among agents.

The best known and most widely applied is the **Contract Net Protocol**: an interaction protocol for cooperative problem solving among agents. It is modeled on the contracting mechanism used by business to govern the exchange of goods and services.

An agent wanting a task to be solved is called the *manager*; agents that might be able to solve the task are called potential *contractors*.

From a manager's perspective:

- Announce a task that needs to be performed
- Receive and evaluate bids from potential contractors
- Award a contract to a suitable contractor
- Receive and synthesize results

From a contractor's perspective:

- Receive task announcements
- Evaluate my capabilities to respond
- Respond (decline, bid)
- Perform the task if my bid is accepted
- Report my results

A contractor for a specific task may act as a manager by soliciting the help of other agents in solving parts of that ask.

An expiration time gives a deadline for receiving bids.

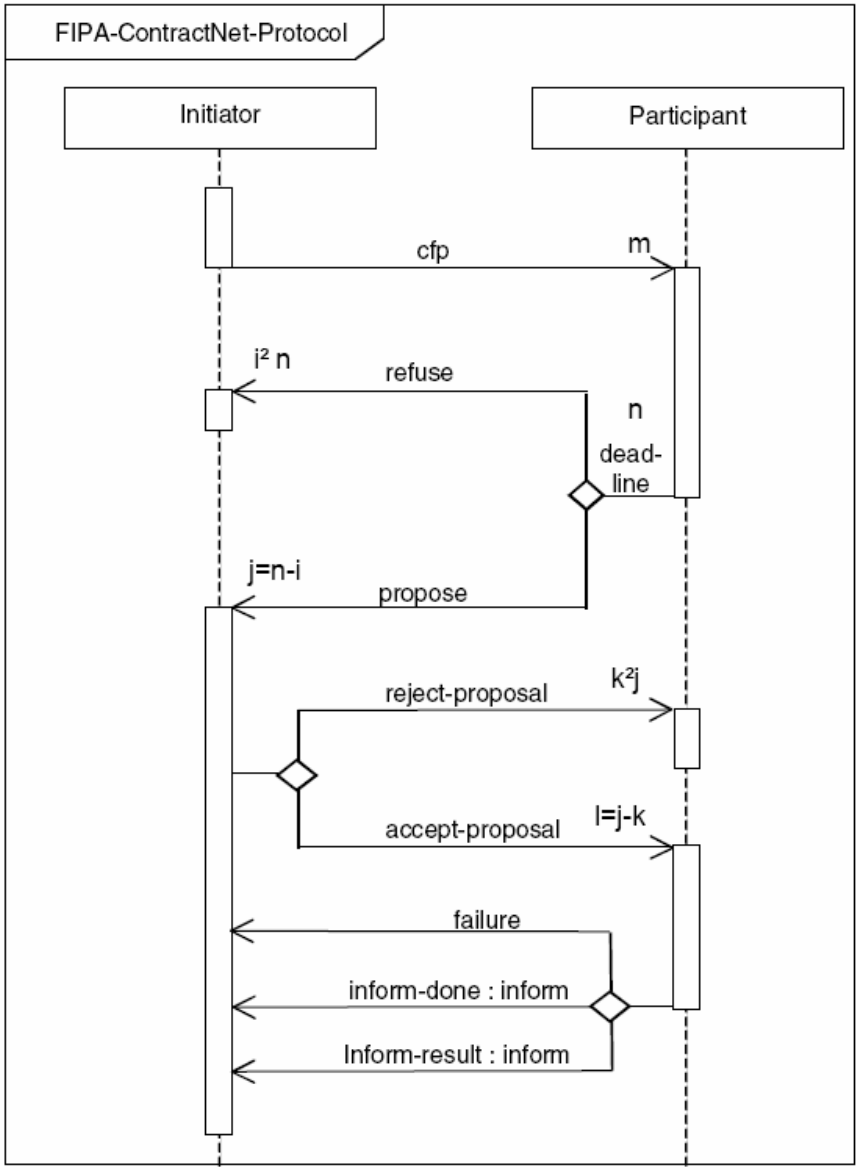


Figure 1: FIPA Contract Net Interaction Protocol

Protocols in the social approach

According to Singh, protocols can be specified as *sets of commitments* rather than as finite state machines.

Agents play different roles within a society, and the roles define the associate social commitments or obligations to other roles. For instance, A will honor a price quote, provided B responds within a specified period.

In general, agents can operate on their commitments by manipulating or canceling them.

Because protocol requirements would be expressed solely in terms of commitments, agents could be tested for compliance on the basis of their communications (it is not necessary to know the implementation).

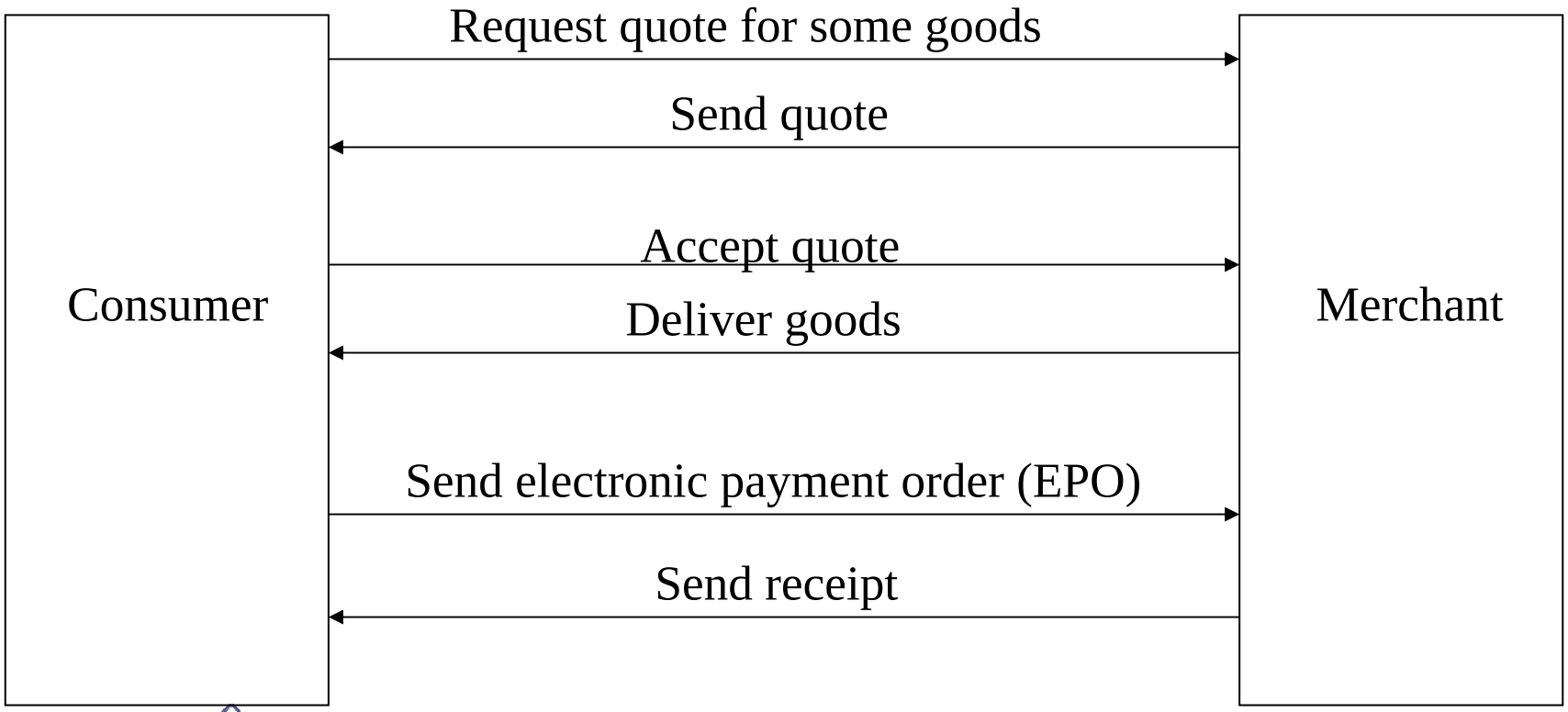
Rigid vs. freeform protocols

Rigid protocols define a set of pre-designed sequences of actions in a conversation. They specify which actions are allowed in each state of the conversation.

Freeform protocols give the agents more freedom in choosing their actions. The protocol does not specify the precise action that must be executed at each step, but rather the effect which must be achieved. For instance, in a social approach, all commitments must be fulfilled by the end of the protocol.

An example: The NetBill protocol

We consider a simplified form of the NetBill protocol developed for buying and selling goods on the Internet.



This protocol can be executed in many different ways.

For instance, the merchant may send a quote before the customer asks for it, to advertise his goods,

or the merchant wants to send the goods without a prior acceptance of the customer, similar to the trial version of software products which lasts a certain period of time.

Due to this flexibility, the standard approaches to protocol representation, e.g. finite state automata, are inadequate.

Protocol specifications

Many issues are involved in the specification of protocol:

1. Adopt a formalism which allows more flexibility than finite state machines (take into account also the semantics of communicative actions)
2. Consider unexpected (or exceptional) messages within the protocol
3. Specify protocols at a high level of abstraction
4. Adopt a declarative approach
5. Provide formal properties of the protocol proposed.

Multiagent systems verification

Several different types of verification are possible, depending on the type of ACL (mental or social language), the information available (internal state, external state, language specification) and whether we wish to verify at design time or at run time.

Design time verification is important when we want to prove some properties to guarantee certain behaviors or outcomes of the system.

Run time verification is used to determine if agents are misbehaving in a certain run of the system. It is important in an open system because it may be the only way to identify rogue agents.

Verification in open systems

The following types of verification are useful in an open system:

1. Verify that an agent will always satisfy its social facts

Suppose we are using a social language and we have access to the agent's program code, with its internal states. We can verify at design time that the agent will always respect its social commitments, regardless of what other agents will do. To do this we have to prove that, for all computations of the system, social facts that are true for agent i (e.g. commitments) will be satisfied.

2. Interoperability

Verify that a set of agents will interact correctly according to a given protocol. This requires to check compliance of each agent with the protocol.

3. Prove a property of a protocol

In this case we do not know the internals of the agents. We will reason on all possible observable sequences of states, by assuming that all agents are compliant.

4. Determine if an agent is not respecting its social facts at runtime

In this case we will reason on an observable history of messages exchanged by one agent or by the entire system. With this information it may be possible to determine if agents have complied with the ACL thus far, but not to determine if they will comply in the future.

Approaches to verification

Guerin and Pitt have proposed a general agent communication framework within which several notions of verification can be investigated. Informally, the framework consists of:

An agent programming language: allowing to program sets of agent which communicate via message passing. Agents have an internal state, which can model the mental state of the agent.

Social state: describes all publicly observable phenomena including propositions representing social facts (commitments), control variables (roles), the rules governing interaction and the history of transmitted messages.

An Agent Communication Language, whose semantics accounts both for the mental part and the social part.

Event calculus

Yolum and Singh proposed an approach based on *event calculus*, a formalism to reason about *events* (actions). Events can initiate or terminate *fluents*, and time appears explicitly in event calculus formulas.

Some of the predicates for reasoning about events are:

Initiates(a, f, t) fluent f holds after event a at time t

Terminates(a, f, t) fluent f does not hold after event a at time t

Initially(f) fluent f holds from time 0

Happens(a, t) event a happens at time t

HoldsAt(f, t) fluent f holds at time t

Commitments are represented as fluents. There are

base-level commitments C(x, y, p)

conditional commitments CC(x, y, p, q): if condition p is brought out, x will be committed to y to bring about q.

Some rules can be defined to reason about commitments. For instance

$$Terminates(e, C(x, y, p), t) \leftarrow$$
$$HoldsAt(C(x, y, p), t) \wedge Happens(e, t) \wedge Initiates(e, p, t)$$

(A commitment is no longer in force if the condition committed to holds)

$$Initiates(e, C(x, y, q), t) \wedge Terminates(e, CC(x, y, p, q), t) \leftarrow$$
$$HoldsAt(CC(x, y, p, q), t) \wedge Happens(e, t) \wedge Initiates(e, p, t)$$

(A conditional commitment is transformed into a base-level commitment)

For the NetBill protocol, let MR be the merchant and CT the customer.

Some fluents are:

request(i): the customer has requested a quote for item i
goods(i): the merchant has delivered the goods i
pay(m): the customer has paid the amount m

Some abbreviations for commitments:

accept(i, m) ≡ CC(CT, MR, goods(i), pay(m))

the customer is willing to pay if he receives the goods

promiseGoods(i, m) ≡ CC(MR, CT, accept(i, m), goods(i))

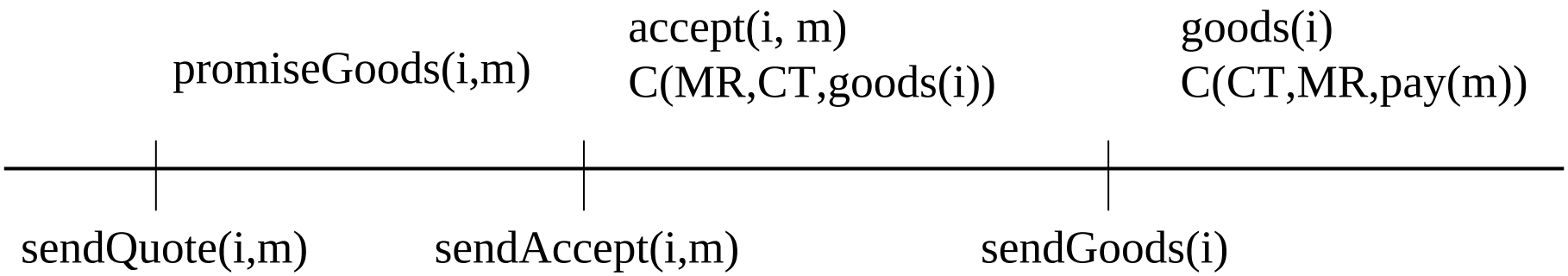
the merchant is willing to send the goods if the customer promises to pay the agreed amount

Protocols are specified by a set of Initiates and Terminates clauses. For instance:

Initiates (sendQuote(i, m), promiseGoods(i, m), t)

Initiates(sendAccept(i, m), accept(i, m), t)

Initiates(sendGoods(i), goods(i), t)



remember:

accept(i, m) ≡ CC(CT, MR, goods(i), pay(m))

promiseGoods(i, m) ≡ CC(MR, CT, accept(i, m), goods(i))

Yolum and Singh show how to reason about protocols in their event calculus approach. Using an event calculus planner, they can generate complete protocol runs, i.e. sequences of actions of the protocol such that there are no pending base-level commitments.

Within MASSIVE we have investigated different approaches to protocol verification:

A model of agent society based on computational logic, where interaction protocols can be described by a set of integrity constraints.

An approach based on the logic DLTL, where communicative actions are modeled by means of preconditions and effects on the social state, and protocols can be described by means of rules defining permissions to execute actions, and commitment fulfillment.

The problem of interoperability has been studied for protocols expressed by finite state automata.

Integrity constraints

A model of agent interaction based on *social integrity constraints*. Social integrity constraints are used to model protocols.

The multiagent system records in a "history" file the observable events for the society: **H**(e), event *e happened*.

A course of events might give rise to social expectations about the future behavior of the agents: **E**(e) (event *e* is expected to happen) or **NE**(e) (event *e* is expected not to happen).

Integrity constraints link events and expectations. For instance:

$$\mathbf{H}(\text{tell}(x,y,\text{start})) \rightarrow \mathbf{E}(\text{pass}(y))$$

if x told y "start", then we expect a social event pass(y).

Expectations can be used to represent *commitments*. In the NetBill:

$$\mathbf{H}(\text{sendQuote}(m,i,t_q)) \wedge \mathbf{H}(\text{sendAccept}(m,i,t_a)) \wedge t_q < t_a \rightarrow$$

$$\mathbf{E}(\text{sendGoods}(i,t_g)) : t_g \leq t_a + \tau$$

where the last argument of actions represents time.

If the merchant has sent a quote and the customer has accepted, the merchant is committed to send the goods with a maximum delay τ.

Backward expectations allow to express preconditions:

$$\mathbf{H}(\text{sendReceipt}(m,i,t_r)) \rightarrow \mathbf{E}(\text{sendEPO}(i,t_g)) : t_g < t_r$$

If the merchant sends a receipt and the customer has not sent an EPO before, the constraint will be violated.

In this framework it is possible to determine if some agent is not respecting its social facts at runtime, by checking compliance of the "history" of observable events with the specifications.

It is also possible to verify compliance of agents to protocols, for a restricted class of programs and protocols, by specifying both agents and protocols in terms of integrity constraints.

Dynamic linear time logic

DLTL combines linear time logic with dynamic logic, by indexing the *until* operator with regular programs of dynamic logic. It can be used to reason about composite actions (programs) and to express temporal properties.

Given an alphabet Σ of primitive actions, formulas of DLTL are:

$$p \mid \neg\alpha \mid \alpha \vee \beta \mid \alpha \, \mathbf{U}^{\pi} \, \beta$$

where π is a program over Σ.

Protocols are formulated as sets of *action laws*, specifying effects of actions. For instance:

- □[sendQuote(i, m)] promiseGoods(i, m)
- □[sendAccept(i, m)] accept(i, m)

The protocol can specify constraints (*permissions*) on the execution of actions by giving preconditions to the actions:

$\Box(\neg \text{paid} \rightarrow [\text{sendReceipt}]\ \bot)$ if the payment has not been done, sendReceipt cannot be executed by the merchant.

An agent *i* satisfies its commitments when, for all commitments *C(i, j, a)* in which agent *i* is the debtor, the formula

$$\Box_i(C(i, j, a) \rightarrow \Box_i \langle a \rangle_i\ \top)$$

holds. When an agent is committed to execute action *a*, then it must eventually execute *a*.

Reasoning about protocols in DLTL

The verification problems mentioned before can be formulated as satisfiability or validity of DLTL formulas.

Since the logic allows to formulate programs, it is also possible to prove properties regarding compliance of an agent program with a given protocol.

In some cases it is necessary to reason simultaneously on the behavior of more than one agent. This requires to use the product version of DLTL.

Model checking

One particularly successful approach to the verification of concurrent systems is *model checking*. This approach can be used for verifying multiagent systems.

Model checking is a semantic approach: given a model M in a given logic L, and a formula φ of L, determine whether or not φ is valid in M.

In particular, practical model checking techniques are based on temporal logics and on the close relationships between models for temporal logic and finite-state machines describing computations.

Given a (concurrent) program π and a temporal logic formula φ (describing a specification or property), to show that φ holds for program π, we proceed as follows:

• take π and generate from it a *Kripke structure* M_π. A Kripke structure consists of a set of states, a set of transitions between states, and a function that labels each state with a set of propositions that are true in that state. Paths in a Kripke structure model computations of π;

• show that M_π is a model of φ, i.e. that $M_\pi \models \varphi$.

If φ is an LTL (linear time temporal logic) formula, model checking can be performed using automata. The advantage of this approach is that both the modeled system and the specification are represented in the same way.

In fact, given an LTL formula φ, we can construct a Büchi automaton $\mathcal{B}_\varphi$ such that the language $\mathcal{L}(\mathcal{B}_\varphi)$ accepted by $\mathcal{B}_\varphi$ is non-empty iff φ is satisfiable. Furthermore it is easy to build an automaton $\mathcal{B}_\pi$ which directly corresponds to M_π.

This formulation suggest the following model checking procedure:

- Construct the two automata $\mathcal{B}_{\pi}$ and $\mathcal{B}_{\neg\varphi}$.
- Construct the automaton which accepts the intersection of the languages $\mathcal{L}(\mathcal{B}_{\pi})$ and $\mathcal{L}(\mathcal{B}_{\neg\varphi})$ (the product of the two automata).
- If the intersection is empty, then φ holds for π, otherwise a run in the intersection provides a counterexample.

In general this problem is PSPACE-complete, but efficient techniques have been proposed and implemented.

Model checking for ACL compliance

Wooldridge et al. have developed an approach to the verification of properties of multi-agent systems using model checking, based on the language MABLE.

MABLE is essentially a conventional imperative programming language, enriched by constructs from the agent-oriented programming paradigm. Agents in MABLE have a mental state consisting of beliefs, desires and intentions, and communicate using FIPA-like performatives.

MABLE systems may be augmented by addition of *formal claims* about the system. Claims are expressed using a (simplified) version of the BDI logic *LORA*, called *MORA*.

The MABLE language has been implemented by making use of SPIN, a freely available model-checking system based on LTL.

MABLE has been used to verify ACL compliance.

Communication is realized by means of send and receive instructions:

```
send(inform agent2 of (a ==10))
```

Programmers can define their own semantics for communicative acts, separately from a program, and then verify the *compliance* of the program with the semantics. The semantics is expressed in a STRIPS-style pre/post-conditions formalism. for instance:

inform(i, j, φ)

Pre: (Bel i φ) (if i is *sincere*)

Post: (Bel j (Int i (Bel j φ)))

The following *LORA* formula expresses the property that an inform performative satisfies its preconditions:

$$\mathbf{A}\,\Box\,(\text{Happens inform}(i, j, \varphi)) \Rightarrow (\text{Bel } i\ \varphi)$$

i.e. whenever agent *i* sends an inform message to agent *j* with content φ, then *i* believes φ (*i* is sincere).

This formula can be expressed as a MABLE claim, and added to the MABLE program describing the multi-agent system we want to verify.

The same approach can be used to verify rational effects, e.g.

$$\mathbf{A}\,\Box\,(\text{Happens inform}(i, j, \varphi)) \Rightarrow \Diamond\,(\text{Bel } j\ \varphi)$$

Within MASSIVE we have experimented the use of model checking for proving protocol properties.

In particular in the approach based on DLTL, it is possible to carry out the above proofs using model checking techniques, by extracting a model from the formulas expressing the domain descriptions (action laws), and then checking the other formulas on it.

An efficient technique for obtaining Büchi automata from DLTL formulas has been developed, by extending the construction defined for LTL.

Experiments have been done using the model checker SPIN, based on LTL.

Recent research activities within MASSIVE, and future work, aim at exploiting the techniques and tools developed in the project in various application areas, such as:

- Medical guidelines
- Web services

A further research topic regards the translation of languages which have been proposed for specifying interaction protocols, in particular graphical languages, into the formalisms developed in the project, to give a formal semantics to those languages.