A temporal approach to the specification and verification of Interaction Protocols

Laura Giordano[1], Alberto Martelli[2],
Paolo Terenziani[1], Alessio Bottrighi[1], Stefania Montani[1]

[1]Dipartimento di Informatica, Università del Piemonte Orientale, Alessandria
[2]Dipartimento di Informatica, Università di Torino, Torino

Summary of the paper

- The paper addresses the problem of *specifying* and *verifying* systems of communicating agents.
- We present an approach developed in the context of the project PRIN 2003 *“Logic-based development and verification of multi-agent systems"*.
- We discuss the applicability of this approach to the specification of clinical guidelines.

Summary of the paper

- We adopt a *social approach* to agent communication, where communication is described in terms of changes to the social state, and interaction protocols in terms of *permissions* and *commitments* among agents.
- In particular, we make use of a *temporal action theory*, where a protocol is defined as a set of temporal constraints, which specify the *effects* and *preconditions* of the *communicative actions* on the social state.
- The action theory is based on a *Dynamic Linear Time Temporal Logic (DLTL)*

The logical framework

- A theory for reasoning about communicative actions based on Dynamic Linear Time Temporal Logic ($DLTL$), an extension of LTL (the propositional linear time temporal logic).
- $DLTL$ extends LTL by strengthening the *until* operator by indexing it with the *regular programs* of dynamic logic. It is, essentially, a dynamic logic equipped with a linear time semantics.
- $DLTL$ has an exponential time decision procedure based on Büchi automata.

DLTL (Henriksen, Thiagarajan)

Σ be a finite non-empty alphabet of **actions**.
Σ^ω the set of infinite words on Σ.

$Prg(\Sigma)$ the set of **programs** (regular expressions)

$$Prg(\Sigma) ::= a \mid \pi_1 + \pi_2 \mid \pi_1; \pi_2 \mid \pi^*, \text{ where } a \in \Sigma$$

$[[\pi]]$ represents the set of executions of π

The set of formulas of DLTL(Σ):

$$\text{DLTL}(\Sigma) ::= p \mid \neg\alpha \mid \alpha \vee \beta \mid \alpha \mathcal{U}^\pi \beta$$

where $p \in \mathcal{P}$ are atomic propositions

DLTL

A model of DLTL(Σ) is a pair $M = (\sigma, V)$ where $\sigma \in \Sigma^\omega$ and $V : prf(\sigma) \to 2^{\mathcal{P}}$ is a valuation function.

Given a model $M = (\sigma, V)$, and a prefix τ of σ, we can define the satisfiability of a formula at τ in M. In particular:

$M, \tau \models \alpha \mathcal{U}^\pi \beta$ iff there exists $\tau' \in [[\pi]]$ such that

- $\tau\tau' \in prf(\sigma)$
- $M, \tau\tau' \models \beta$
- $M, \tau\tau'' \models \alpha$ for every prefix τ'' of τ'

DLTL

We can define derived modalities

- $\langle \pi \rangle \alpha \equiv \top \mathcal{U}^{\pi} \alpha$
- $[\pi] \alpha \equiv \neg \langle \pi \rangle \neg \alpha$
- $\bigcirc \alpha \equiv \bigvee_{a \in \Sigma} \langle a \rangle \alpha$ (next)
- $\Diamond \alpha \equiv \top \mathcal{U}^{\Sigma^*} \alpha$
- $\Box \equiv \neg \Diamond \neg \alpha$

Specifying protocols

We adopt a *social approach* to protocol specification, where communicative actions affect the *social state* of the system, rather than the internal states of the agents. The social state records the social facts, like the *permissions* and the *commitments* of the agents, which are created and modified in the interactions among them.

This formulation does not require the rigid specification of all the allowed action sequences, e.g. by means of finite state diagrams.

The action theory can provide a high level specification of the protocol.

Action theory

For each action a we can define
Action laws

$$\Box(\alpha \rightarrow [a]\beta)$$

Precondition laws

$$\Box(\alpha \rightarrow [a]\bot)$$

Causal laws

$$\Box((\alpha \wedge \bigcirc\beta) \rightarrow \bigcirc\gamma)$$

Persistency is modeled by a completion construction.

Contract net

The Contract Net protocol begins with an agent (the manager) broadcasting a task announcement (call for proposals) to other agents viewed as potential contractors (the participants). Each participant can reply by sending either a proposal or a refusal. The manager must send an accept or reject message to all those who sent a proposal. When a contractor receives an acceptance it is committed to perform the task.

Contract net

We assume to have only two **agents**: M, the manager, and P, the participant.

Actions:
cfp, $accept$, $reject$, $end\_protocol$ whose sender is the merchant
$refuse$, $propose$, $inform\_done$ whose sender is the participant

Fluents:
CN (which is true during the execution of the protocol)
task (whose value is true after the task has been announced),
replied (the participant has replied),
proposal (the participant has sent a proposal),
acc_rej (the manager has sent an accept or reject message)
accepted (the manager has accepted the proposal of participant)
done (the participant has performed the task).

Action laws

Action laws describe the effects of actions on the social state.

$$\Box[cfp](task \wedge CN \wedge CC(M, P, proposal, acc\_rej))$$
$$\Box[accept]acc\_rej$$
$$\Box[reject]acc\_rej$$
$$\Box[refuse]replied$$
$$\Box[propose](replied \wedge proposal \wedge CC(P, M, accepted, done))$$
$$\Box[inform\_done]done$$
$$\Box[end\_protocol(CN)]\neg CN$$

Permissions

The permissions to execute communicative actions in each state are represented by *precondition laws*.

$$\Box(\neg CN \vee \neg proposal \vee acc\_rej \rightarrow [accept]\bot)$$

Action *accept* cannot be executed outside the protocol, or if a proposal has not been done, or if the manager has already replied

$$\Box(\neg CN \vee task \rightarrow [cfp]\bot)$$
$$\Box(\neg CN \vee \neg proposal \vee acc\_rej \rightarrow [reject]\bot)$$
$$\Box(\neg CN \vee \neg task \vee replied \rightarrow [refuse]\bot)$$
$$\Box(\neg CN \vee \neg task \vee replied \rightarrow [propose]\bot)$$
$$\Box(\neg CN \vee \neg accepted \vee done \rightarrow [inform\_done]\bot)$$
$$\Box(\neg CN \vee \neg task \rightarrow [end\_protocol(CN)]\bot)$$

$Perm_i$: permissions of agent i

Commitments

Commitments are special fluents

- **base-level commitments**: $C(ag_1, ag_2, \alpha)$ (agent ag_1 is committed to agent ag_2 to bring about α)
- **conditional commitments**: $CC(ag_1, ag_2, \beta, \alpha)$ (agent ag_1 is committed to agent ag_2 to bring about α, if the condition β is brought about)

In the Contract Net we have the commitments:

$C(P, M, replied)$ and $C(M, P, acc\_rej)$

and conditional commitments

$CC(P, M, task, replied)$ and
$CC(M, P, propose, acc\_rej)$.

Rules for commitments

Some reasoning rules have to be defined for cancelling commitments when they have been fulfilled and for dealing with conditional commitments. We introduce the following *causal laws*:

$$\Box(\bigcirc\alpha \rightarrow \bigcirc\neg C(i,j,\alpha))$$
$$\Box(\bigcirc\alpha \rightarrow \bigcirc\neg CC(i,j,\beta,\alpha))$$
$$\Box((CC(i,j,\beta,\alpha) \wedge \bigcirc\beta) \rightarrow (\bigcirc(C(i,j,\alpha) \wedge \neg CC(i,j,\beta,\alpha))))$$

A commitment (or a conditional commitment) to bring about α is cancelled when α holds, and a conditional commitment $CC(i,j,\beta,\alpha)$ becomes a base-level commitment $C(i,j,\alpha)$ when β has been brought about.

Fulfilling commitments

We are interested in those execution of the ContractNet protocol in which all commitments have been fulfilled. We can express the condition that the commitment $C(i, j, \alpha)$ will be *fulfilled* within the execution of the ContractNet protocol by the constraint:

$$\Box(C(i, j, \alpha) \rightarrow (CN \ \mathcal{U} \ \alpha))$$

We call Com_i the set of constraints of this kind for all commitments of agent i.

Com_i states that agent i will fulfill all the commitments of which he is the debtor.

Starting the protocol

We define the *initial state* **Init** of the protocol as follows:

$$\{\neg CN, \neg task, \neg replied, \neg proposal, \neg done,$$
$$CC(P, M, task, replied), C(M, P, task)\}$$

Domain description

$D = (Comp(\Pi), \mathcal{C})$ is the domain description of The ContractNet protocol, where:

- $Comp(\Pi)$ is the completion of the set Π of the action and causal laws given above
- $\mathcal{C} = Init \wedge \bigwedge_i (Perm_i \wedge Com_i)$.

The *runs* of the system according the protocol are the linear models of D. In these protocol runs all permissions and commitments have been fulfilled.

Verifying agents compliance at runtime

We are given a history $\tau = a_1, \ldots, a_n$ of the communicative actions executed by the agents, and we want to check the compliance of that execution with the protocol.

This problem can be formalized as a *satisfiability problem*. The formula

$$(Comp(\Pi) \wedge Init \wedge \bigwedge_i (Perm_i \wedge Com_i)) \wedge < a_1; a_2; \ldots; a_n > \top$$

is satisfiable if it is possible to find a run of the protocol starting with the action sequence $a_1, \ldots, a_n$.

Verifying protocol properties

Proving that the protocol satisfies a given (temporal) property φ can be formalized as a em validity check. The formula

$$(Comp(\Pi) \wedge Init \wedge \bigwedge_i (Perm_i \wedge Com_i)) \rightarrow \varphi. \qquad (1)$$

is valid if all the runs of the protocol satisfy φ. Observe that, all the agents are assumed to be compliant with the protocol.

For instance,

$$\varphi = \Box[cfp]\Diamond\neg CN$$

after a call for proposal has been issued by the manager, the protocol will eventually reach a state in which the proposition *CN* is false, i.e. the protocol is finished, for all possible runs of the protocol.

Verifying the compliance at compile-time

Verify that an agent is compliant with the protocol, given the program executed by the agent itself.

We can specify the behavior of an agent by making use of *complex actions (regular programs)*. The following program π_P describes the behavior of the participant:

$$[\neg end?; ((cfp; eval\_task; (\neg ok?; refuse+ \\ ok?; propose))+ \\ reject+ \\ (accept; do\_task; inform\_done)+ \\ (end\_protocol(CN); exit))]^*; end?$$

Local fluents: $done$, $exit$, and ok.
Local actions: $eval\_task$, do_task, $exit$, $done?$ and $ok?$.

Domain description

The *program of the participant* can be specified by a domain description $Prog_P = (\Pi_P, \mathcal{C}_P)$, where Π_P is a set of action laws describing the effects of the private actions of the participant.

For instance, the action $exit$ sets the proposition end to true:

$$\Box[exit]end$$

The set of constraints

$$\mathcal{C}_P = \{\langle \pi_P \rangle \top, \neg end, \neg ok\}$$

provides the initial values for the local fluents as well as the formula $\langle \pi_P \rangle \top$ stating that the program of the participant is executable in the initial state.

Compliance verification

The verification that the participant is compliant with the protocol can be formalized as a validity check. Let $D = (\Pi, \mathcal{C})$ be the domain description describing the protocol, as defined above. The formula

$$(Comp(\Pi) \wedge Init \wedge Perm_M \wedge Com_M \wedge Comp(\Pi_P) \wedge \mathcal{C}_P) \rightarrow (Perm_P \wedge Com_P)$$

is valid if in all the behaviors of the system, in which the participant executes its program π_P and the manager (whose internal program is unknown) respects the protocol specification (in particular, its permissions and commitments), the permissions and commitment of the participant are also satisfied.

Decision problem in DLTL

The **satisfiability** problem for DLTL can be solved in EXPTIME, as for LTL, by constructing for each formula α a **Büchi automaton** $\mathcal{B}_\alpha$ such that there is a one to one correspondence between models of the formula and infinite words accepted by $\mathcal{B}_\alpha$.

For instance, every infinite word accepted by $\mathcal{B}_D$ corresponds to a possible run of the ContractNet protocol.

To **prove a property φ of the protocol**, we can build the automaton $\mathcal{B}_{\neg\varphi}$ and check that the language accepted by the product of $\mathcal{B}_D$ and $\mathcal{B}_{\neg\varphi}$ is empty.

Verification in SPIN

- An alternative way for applying this approach in practice, is to make use of existing model checking tools, such as for instance SPIN.
- We have done some experiments with the model checker SPIN on proving properties of protocols expressed according to the approach presented in this paper.
- The domain description is formulated as a PROMELA program, which describes all possible runs allowed by the domain theory.
- Properties and constraints are expressed as LTL formulas.

An application to clinical guidelines

- Clinical guidelines can be roughly defined as frameworks for specifying the "best" clinical procedures and for standardizing them.
- Many different systems and projects have been developed in recent years in order to realize computer-assisted management of clinical guidelines.
- **GLARE** (Guidelines Acquisition, Representation and Execution) is one of such domain-independent systems, developed by a group of computer scientists from UPO and UNITO, in collaboration with Azienda Ospedaliera S. Giovanni Battista in Torino.

An application to clinical guidelines (contd.)

- We have started to analyze
  - (i) how clinical guidelines can be *modelled* in our framework
  - (ii) how the *reasoning* facilities provided by a model checker can be exploited within the clinical application environment

Modelling clinical guidelines

- Clinical guidelines can be regarded as *hierarchically structured* protocols.
- At the lower level, they are basically composed by sequences of *elementary actions* and *decision actions* needed to choose among alternative paths
- Elementary actions and decision actions can be combined in structured actions by constructs like *sequence*, *choice* and *iteration*.
- The overall structure of a clinical guideline can be modelled as the *composition* of different protocols.

Reasoning about clinical guidelines

Model checking capabilities can be used in order to:

- instantiate a guideline on a specific patient;
- contextualize guidelines to specific hospitals, considering locally available laboratories and resources;
- look for executable paths which satisfy a given set of requirements (concerning e.g. costs, execution times and specific goals).

Conclusions

- We have developed an approach to the specification and verification of interaction protocols in a multiagent system, developed in the context of the national project PRIN 2003 *“Logic-based development and verification of multi-agent systems"*.
- We are currently investigating the applicability of the approach, on the one hand to the specification and verification of *clinical guidelines* and, on the other hand, to the specification and verification of *Web Services*, with a particular regard to the problem of service *composition* and *coordination*.