A temporal approach to the specification and verification of Interaction Protocols

L. Giordano *, A. Martelli†, P. Terenziani*, A. Bottrighi* and S. Montani*
* Dipartimento di Informatica, Università del Piemonte Orientale, Alessandria, Italy
† Dipartimento di Informatica, Università di Torino, Torino, Italy

***Abstract*— The paper presents a proposal for the specification and verification of systems of communicating agents in a temporal logic. The proposal is based on a social approach to agent communication, where communication is described in terms of changes to the social state, and interaction protocols are defined by a set of temporal constraints, which specify the effects and preconditions of the communicative actions on the social state. The paper addresses the problem of combining protocols to define new more specialized protocols and exploits this idea in the specification of clinical guidelines.**

I. Introduction

Agent technology has been rapidly developing in the last decade to answer the needs for new conceptual tools for modelling and developing complex software systems and it has given rise to a large amount of literature [6]. Autonomous agents can communicate, cooperate and negotiate using commonly agreed communication languages (ACLs) and protocols. The issue of interoperability has lead to the development of standardized agent communication languages, including KQML [21] and FIPA-ACL [4]. One of the central issues in the field concerns the specification of conversation policies, which govern the communication between software agents in an agent communication language (ACL). Conversation policies (or interaction protocols) define stereotypical interactions in which ACL messages are used to achieve communicative goals.

The specification of interaction protocols has been traditionally done by making use of finite state machines, but the transition net approach has been soon recognized to be too rigid to allow for the flexibility needed in agent communication [24], [16]. For these reasons, several proposals have been put forward to address the problem of specifying (and verifying) agent protocols in a flexible way. One of the most promising approaches to agent communication, first proposed by Singh [28], is the social approach [1], [8], [18], [24]. In the social approach, communicative actions affect the "social state" of the system, rather than the internal (mental) states of the agents. The social state records social facts, like the permissions and the commitments of the agents.

In this paper we present a temporal approach to the specification and verification of interaction protocols among agents. Temporal logics are extensively used in the area of reasoning about actions and planning [2], [14], [11], [26], [3], and, in particular, they have been used in the specification and in the verification of systems of communicating agents. In [34], [22] agents are written in MABLE, an imperative programming language, and the formal claims about the system are expressed using a quantified linear time temporal BDI logic and can be automatically verified by making use of the SPIN model checker. Guerin in [17] defines an agent communication framework which gives agent communication a grounded declarative semantics. In such a framework, temporal logic is used for formalizing temporal properties of the system. Our theory for reasoning about communicative actions is based on the Dynamic Linear Time Temporal Logic ($DLTL$) [19], which extends LTL by strengthening the *until* operator by indexing it with the regular programs of dynamic logic. As a difference with [34] we adopt a social approach to agent communication. The dynamics of the system emerges from the interactions of the agents, which must respect permissions and commitments (if they are compliant with the protocol). The social approach allows a high level specification of the protocol, and it is well suited for dealing with "open" multi-agent systems, where the history of communications is observable, but the internal states of the single agents may not be observable.

The paper provides an overview of the approach developed in [12], [13], and describes the different kinds of verification problems which can be addressed, which can be formalized either as validity or as satisfiability problems in DLTL. These verification tasks can be automated by making use of Büchi automata. In particular, we can make use of the tableau-based algorithm presented in [10] for constructing a Büchi automaton from a DLTL formula. The construction of the automata can be done on-the-fly, while checking for the emptiness of the language accepted by the automaton. As for LTL, the number of states of the automata is, in the worst case, exponential in the size of the input formula. We discuss the applicability of this approach to the specification of clinical guidelines.

II. Dynamic Linear Time Temporal Logic

In this section we shortly define the syntax and semantics of DLTL as introduced in [19]. In such a linear time temporal logic the next state modality is indexed by actions. Moreover, (and this is the extension to LTL) the until operator is indexed by programs in Propositional Dynamic Logic (PDL).

Let Σ be a finite non-empty alphabet. The members of Σ are actions. Let Σ^* and Σ^ω be the set of finite and infinite words on Σ, where $\omega = \{0, 1, 2, \ldots\}$ and let ϵ denote the empty word. Let $\Sigma^\infty = \Sigma^* \cup \Sigma^\omega$. We denote by σ, σ' the words over Σ^ω and by τ, τ' the words over Σ^*. Moreover, we denote by

$\leq$ the usual prefix ordering over Σ^* and, for $u \in \Sigma^\infty$, we denote by *prf(u)* the set of finite prefixes of u.

We define the set of programs (regular expressions) $Prg(\Sigma)$ generated by Σ as follows:

$$Prg(\Sigma) ::= a \mid \pi_1 + \pi_2 \mid \pi_1; \pi_2 \mid \pi^*$$

where $a \in \Sigma$ and π_1, π_2, π range over $Prg(\Sigma)$. A set of finite words is associated with each program by the mapping $[[]] : Prg(\Sigma) \rightarrow 2^{\Sigma^*}$, which is defined as usual.

Let $\mathcal{P} = \{p_1, p_2, \ldots\}$ be a countable set of atomic propositions. The set of formulas of DLTL(Σ) is defined as follows:

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

where $p \in \mathcal{P}$ and α, β range over DLTL(Σ) and π ranges over $Prg(\Sigma)$.

A model of DLTL(Σ) is a pair $M = (\sigma, V)$ where $\sigma \in \Sigma^\omega$ and $V : prf(\sigma) \rightarrow 2^{\mathcal{P}}$ is a valuation function. Given a model $M = (\sigma, V)$, a finite word $\tau \in prf(\sigma)$ and a formula α, the satisfiability of a formula α at τ in M, written $M, \tau \models \alpha$, is defined as follows:

- $M, \tau \models p$ iff $p \in V(\tau)$;
- $M, \tau \models \neg\alpha$ iff $M, \tau \not\models \alpha$;
- $M, \tau \models \alpha \vee \beta$ iff $M, \tau \models \alpha$ or $M, \tau \models \beta$;
- $M, \tau \models \alpha \mathcal{U}^\pi \beta$ iff there exists $\tau' \in [[\pi]]$ such that $\tau\tau' \in prf(\sigma)$ and $M, \tau\tau' \models \beta$. Moreover, for every τ'' such that $\varepsilon \leq \tau'' < \tau'$[1], $M, \tau\tau'' \models \alpha$.

A formula α is satisfiable iff there is a model $M = (\sigma, V)$ and a finite word $\tau \in prf(\sigma)$ such that $M, \tau \models \alpha$.

The formula $\alpha \mathcal{U}^\pi \beta$ is true at τ if "α until β" is true on a finite stretch of behavior which is in the linear time behavior of the program π.

The derived modalities $\langle\pi\rangle$ and $[\pi]$ can be defined as follows: $\langle\pi\rangle\alpha \equiv \top\mathcal{U}^\pi\alpha$ and $[\pi]\alpha \equiv \neg\langle\pi\rangle\neg\alpha$.

Furthermore, if we let $\Sigma = \{a_1, \ldots, a_n\}$, the $\mathcal{U}$, $\bigcirc$ (next), $\Diamond$ and $\Box$ operators of LTL can be defined as follows: $\bigcirc\alpha \equiv \bigvee_{a\in\Sigma}\langle a\rangle\alpha$, $\alpha\mathcal{U}\beta \equiv \alpha\mathcal{U}^{\Sigma^*}\beta$, $\Diamond\alpha \equiv \top\mathcal{U}\alpha$, $\Box\alpha \equiv \neg\Diamond\neg\alpha$, where, in $\mathcal{U}^{\Sigma^*}$, Σ is taken to be a shorthand for the program $a_1 + \ldots + a_n$. Hence both LTL(Σ) and PDL are fragments of DLTL(Σ). As shown in [19], DLTL(Σ) is strictly more expressive than LTL(Σ). In fact, DLTL has the full expressive power of the monadic second order theory of ω-sequences.

III. Protocol Specification

In the social approach an interaction protocol is specified by describing the effects of communicative actions on the social state, and by specifying the permissions and the commitments that arise as a result of the current conversation state.

Let us shortly recall the action theory developed in [11] that we use for the specification of interaction protocols.

Let $\mathcal{P}$ be a set of atomic propositions, the *fluents*. A *fluent literal* l is a fluent name f or its negation $\neg f$. Given a fluent literal l, such that $l = f$ or $l = \neg f$, we define $|l| = f$. We will denote by Lit the set of all fluent literals.

[1]We define $\tau \leq \tau'$ iff $\exists\tau''$ such that $\tau\tau'' = \tau'$. Moreover, $\tau < \tau'$ iff $\tau \leq \tau'$ and $\tau \neq \tau'$.

A *domain description* D is defined as a tuple $(\Pi, \mathcal{C})$, where Π is a set of *action laws* and *causal laws*, and $\mathcal{C}$ is a set of *constraints*.

Action laws in Π have the form: $\Box(\alpha \rightarrow [a]\beta)$, with $a \in \Sigma$ and α, β arbitrary formulas, meaning that executing action a in a state where precondition α holds causes the effect β to hold.

Causal laws in Π have the form: $\Box((\alpha \wedge \bigcirc\beta) \rightarrow \bigcirc\gamma)$, meaning that if α holds in a state and β holds in the next state, then γ also holds in the next state. Such laws are intended to expresses "causal" dependencies among fluents.

Constraints in $\mathcal{C}$ are arbitrary temporal formulas of $DLTL$. In particular, the set of constraints includes *precondition laws* of the form: $\Box(\alpha \rightarrow [a]\bot)$, meaning that the execution of an action a is not possible if α holds. (i.e. there is no resulting state following the execution of a if α holds). Observe that, when there is no precondition law for an action, the action is executable in all states.

Action laws and causal laws describe the changes to the state. All other fluents which are not changed by the actions are assumed to persist unaltered to the next state. To cope with the *frame problem*, the laws in Π, describing the (immediate and ramification) effects of actions, have to be distinguished from the constraints in $\mathcal{C}$ and given a special treatment. In [11], we defined a completion construction which, given a domain description, introduces frame axioms in the style of the successor state axioms introduced by Reiter [27]. The completion construction is applied only to the action laws and causal laws in Π and not to the constraints. In the following we call $Comp(\Pi)$ the completion of a set of laws Π.

Let us now provide the specification of the Contract Net protocol [4].

Example 1: 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.

Let us consider first the simplest case where we have only two agents: the manager (M) and the participant (P). The two agents share all the communicative actions, which are: *cfp* (the manager issues a call for proposals for task T), *accept* and *reject* whose sender is the manager, *refuse* and *propose* whose sender is the participant, *inform_done* by which the participant informs the manager that the task has been executed and *end_protocol* by which the manager declares the completion of the protocol.

The social state contains the following domain specific 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 to the participant) *accepted* (the manager has accepted the proposal of participant) and *done* (the participant

has performed the task). Such fluents describe observable facts concerning the execution of the protocol.

We also introduce special fluents to represent *base-level commitments* of the form $C(i,j,\alpha)$, meaning that agent i is committed to agent j to bring about α, where α is an arbitrary formula, or they can be *conditional commitments* of the form $CC(i,j,\beta,\alpha)$ (agent i is committed to agent j to bring about α, if the condition β is brought about). The two kinds of base-level and conditional commitments we allow are essentially those introduced in [35]. For modelling the Contract Net example we introduce the following commitments

$C(P,M,replied) \quad C(M,P,acc\_rej)$
$C(i,M,done) \quad C(M,P,task)$

and conditional commitments

$CC(P,M,task,replied)$
$CC(M,P,proposal,acc\_rej)$
$CC(i,M,accepted,done)$.

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.

Let us now describe the effects of communicative actions by the following *action laws*:

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

The laws for action cfp add to the social state the information that a call for proposal has been done for the $task$, and that, if the manager receives a proposal, it is committed to accept or reject it.

The permissions to execute communicative actions in each state are determined by social facts. We represent them by precondition laws. Preconditions on the execution of action *accept* can be expressed as:

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

meaning that action *accept* cannot be executed outside the protocol, or if a proposal has not been done, or if the manager has already replied. Similarly we can give the *precondition laws* for the other actions:

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

The precondition law for action $propose$ ($refuse$) says that a proposal can only be done if a task has already been announced and the participant has not already replied. The last law says that the manager cannot issue a new call for proposal if a task has already been announced.

In the following we will denote $Perm_i$ (permissions of agent i) the set of all the precondition laws of the protocol pertaining to the actions of which agent i is the sender.

Assume now that we want the participant to be committed to reply to the task announcement. We can express it by adding the following conditional commitment to the initial state of the protocol: $CC(P,M,task,replied)$. Furthermore the manager is committed initially to issue a call for proposal for a task. We can define the *initial state Init* of the protocol as follows:

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

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

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

We will 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.

Given the above rules, the domain description $D = (\Pi, \mathcal{C})$ of a protocol is defined as follows: Π is the set of the action and causal laws given above, and $\mathcal{C} = Init \wedge \bigwedge_i (Perm_i \wedge Com_i)$ is the set containing the constraints on the initial state, the permissions $Perm_i$ and the commitments Com_i of all the agents (the agents P and M, in this example).

Given a domain description D, let the completed domain description $Comp(D)$ be the set of formulas $(Comp(\Pi) \wedge Init \wedge \bigwedge_i (Perm_i \wedge Com_i))$. The runs of the system according the protocol are the linear models of $Comp(D)$. Observe that in these protocol runs all permissions and commitments are fulfilled. However, if Com_j is not included for some agent j, the runs may contain commitments which have not been fulfilled by j.

IV. Protocol verification

Different kinds of verification problems can be addressed, given the specification of a protocol by a domain description.

A. 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. Namely, we want to verify that the history τ is the prefix of a run of the protocol, that is, it respects the permissions and commitments of the

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

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

(where i ranges on all the agents involved in the protocol) is satisfiable if it is possible to find a run of the protocol starting with the action sequence $a_1, \ldots, a_n$.

B. Verifying protocol properties

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

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

is valid if all the runs of the protocol satisfy φ. Observe that, all the agents are assumed to be compliant with the protocol. As an example of property to be checked, we consider the property of termination of the protocol. After the manager has announced a task, the protocol will eventually arrive to completion. This property can be formalized by the temporal formula:

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

meaning that, always, 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.

C. Verifying the compliance of an agent with the protocol at compile-time

When the program executed by an agent is given (or, at least, its logical specification is given), we are faced with the problem of verifying if the agent is compliant with the protocol, that is, to verify if the agent's program respects the protocol. Solving this problem requires: first to provide an *abstract* specification of the behavior (program) of the agent; and, second, to check that all the executions of the agent program satisfy the specification of the protocol, assuming that the other agents are compliant with the protocol.

In the general case, addressing this problem requires to move to the Product Version of DLTL [13]. However, for protocols involving two agents, where all fluents and all actions of the social state are shared by both agents, this verification problems can be represented in DLTL.

In DLTL the behavior of an agent can be specified by making use of complex actions (regular programs). Consider for instance the following program π_P for the participant:

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

The participant cycles and reacts to the messages received by the manager: for instance, if the manager has issued a call for proposal, the participant can either refuse or make a proposal according to his evaluation of the task; if the manager has accepted the proposal, the participant performs the task; and so on.

The state of the agent is obtained by adding to the fluents of the protocol the following local fluents: end, which is initially false and is made true by action $exit$, and ok which says if the agent must make a bid or not. The local actions are $eval\_task$, which evaluates the task and sets the fluent ok to true or false, do_task and $exit$. Furthermore, $end?$ and $ok?$ are test actions.

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_i to true:

$$\Box[exit]end$$

The set of constraints $\mathcal{C}_P$ contains $Init_P$ which provides the initial values for the local fluents ($\neg end, \neg ok$)of the participant as well as the formula $\langle \pi_P \rangle \top$ stating that the program of the participant is executable in the initial state.

To prove that the participant is compliant with the protocol, i.e. that all executions of program π_P satisfy the specification of the protocol, we cannot consider the program π_P alone. In fact, it is easy to see that the correctness of the behavior of the participant depends on the behavior of the manager. Since we don't know its internal behavior, we will assume that the manager respects its public behavior, i.e. that it respects its permissions and commitments in the protocol specification.

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.

D. Proofs and model checking in DLTL

The above verification and satisfiability problems can be solved by extending the standard approach for verification and model-checking of Linear Time Temporal Logic, based on the use of Büchi automata. An approach for constructing a Büchi automaton from a DLTL formula making use of a tableau-based algorithm has been proposed in [10]. The construction of the states of the automaton is similar to the standard construction for LTL [9], but the possibility of indexing until formulas with regular programs puts stronger constraints on the fulfillment of until formulas than in LTL, requiring more complex acceptance conditions. The construction of the automaton can be done on-the-fly, while checking for the emptiness of the language accepted by the automaton. As for LTL, the number of states of the automaton is, in the

worst case, exponential in the size if the input formula, but in practice it is much smaller.

Standard model checking techniques [5] cannot be immediately applied to our approach, because protocols are formulated as sets of properties rather than as programs. Furthermore, in principle, with DLTL we do not need to use model checking, because programs and domain descriptions can be represented in the logic itself, as we have shown in the previous section. However representing everything as a logical formula can be rather inefficient from a computational point of view. In particular all formulas of the domain description are universally quantified, and this means that our algorithm will have to propagate them from each state to the next one, and to expand them with the tableau procedure at each step.

Therefore we have adapted model checking to the proof of the formulas given in the previous section, by deriving the model from the domain theory in such a way that the model describes all possible runs allowed by the domain theory. In particular, we can obtain from the domain description a function $next\_state_a(S)$, for each action a, for transforming a state in the next one, and then build the model (an automaton) by repeatedly applying these functions starting from the initial state. We can then proceed as usual to prove a property φ by taking the product of the model and of the automaton derived from $\neg\varphi$, and by checking for emptiness of the accepted language.

An alternative way for applying this approach in practice, is to make use of existing model checking tools. In particular, by translating DLTL formulas into LTL formulas, it would be possible to use LTL-based model checkers such as for instance SPIN [20]. Although in general DLTL is more expressive than LTL, many protocol properties, such as for instance fulfillment of commitments, can be easily expressed in LTL.

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 model is obtained as suggested above by formulating the domain description as a PROMELA program, which describes all possible runs allowed by the domain theory. Properties and constraints are expressed as LTL formulas. In the case of verification of compliance of an agent implementation with the protocol, we have used different PROMELA processes for representing the agent and the protocol. The representation of the agent is derived from its regular program.

V. An application to clinical guidelines

Clinical guidelines can be roughly defined as frameworks for specifying the "best" clinical procedures and for standardizing them. Clinical guidelines play different roles in the clinical process: for example, they can be used to support physicians in the treatment of diseases, or for critiquing, for evaluation, and for education purposes. Many different systems and projects have been developed in recent years in order to realize computer-assisted management of clinical guidelines (see e.g., [15], [7]). GLARE (Guidelines Acquisition, Representation and Execution) [29], [31] is one of such domain-independent systems. GLARE is being developed by a group of computer scientists from Universita' del Piemonte Orientale and Universita' di Torino, in collaboration with Azienda Ospedaliera S. Giovanni Battista in Torino, one of the largest hospitals in Italy. Despite the system is basically a research product, whose features are continuously refined and updated, the facilities it embeds have been formally tested or at least carefully examined by physicians. Some of the peculiar features of GLARE (with respect to the other computer-based approaches to clinical guidelines in the literature) are its decision-making facilities, which also involve advanced decision theory features [32], and its treatment of temporal constraints [30]. Despite the fact that several specialized "reasoning" facilities are provided by GLARE (see, e.g., [30], [32]), extensive logical reasoning capabilities such as the one which can be provided by theorem proving and/or model checking techniques can provide critical advances (see also [23]). We thus started to analyze (i) how clinical guidelines (such as the one represented by the GLARE system) can be modeled in our framework (ii) how the reasoning facilities provided by the model checker can be exploited within the clinical application environment.

As regards modeling, clinical guidelines are a hierarchical description of clinical procedures. At the lower level, they are basically composed by sequences of elementary actions (corresponding to actions to be executed on the specific patient) and decision actions needed to choose among alternative paths. All the elementary actions in a chosen path must be necessarily executed, unless their preconditions are not satisfied by the patient's data. This can be easily modeled by making use of precondition laws and obligations. Decisions are the core elements in clinical guidelines and are preceded by a data acquisition phase, which can be modeled as an interaction between the physician executing the guideline, the clinical record containing the patient data and, possibly, laboratories, which can be modeled as follows. The physician sends a data request to the database containing clinical records, which is committed to send back the requested data (if available) together with a timestamp stating their time of validity. If the data are not available or not up to date, the physician asks for them to the proper laboratories and waits for the answers. When all the up to date data are available, the decision process can start. In the GLARE approach, decision is modeled as an interaction between the system and the physician. On the basis of the decision criteria embodied in the guideline, and of the patient's data, the system proposes to the physician the subset of alternative paths suggested for the given patient. The physician can commit to one of the suggested alternatives or even to a non suggested one. In the latter case, however, the system sends a warning to the physician.

As regards reasoning, model checking can be used in order to instantiate a guideline on a specific patient, for instance, by checking, on the basis of the patient data, whether there are executable paths. Analogously, guidelines can be contextualized to specific hospitals, considering locally available laboratories and resources. Moreover, model checking capabilities can be

used to look for executable paths which satisfy a given set of requirements (concerning e.g. costs, execution times, goals and intention).

VI. Conclusions

In the paper we have presented an approach to the specification and verification of interaction protocols in a multiagent system that has been 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.

References

[1] M. Alberti, D. Daolio and P. Torroni. Specification and Verification of Agent Interaction Protocols in a Logic-based System. *SAC'04*, March 2004.

[2] F. Bacchus and F. Kabanza. Planning for temporally extended goals. in *Annals of Mathematics and AI*, 22:5–27, 1998.

[3] D. Calvanese, G. De Giacomo and M.Y.Vardi. Reasoning about Actions and Planning in LTL Action Theories. In Proc. *KR'02*, 2002.

[4] FIPA Contract Net Interaction Protocol Specification, 2002. Available at `http://www.fipa.org`.

[5] E.M.Clarke, O.Grumberg, and D. Peled, Model Checking, MIT Press, 2000.

[6] F.Dignum and M.Greaves. Issues in Agent Communication: An Introduction". In F.Dignum and M.Greaves (Eds.), Issues in Agent Communication, LNAI 1916, pp. 1-16, 1999.

[7] Special Issue on Workflow Management and Clinical Guidelines, D.B. Fridsma (Guest ed.), JAMIA, 22(1), 1-80, (2001).

[8] N. Fornara and M. Colombetti. Defining Interaction Protocols using a Commitment-based Agent Communication Language. *Proc. AAMAS'03*, Melbourne, pp. 520–527, 2003.

[9] R. Gerth, D. Peled, M.Y.Vardi and P. Wolper. Simple On-the-fly Automatic verification of Linear Temporal Logic. In *Proc. 15th Work. Protocol Specification, Testing and Verification*, Warsaw, June 1995, North Holland.

[10] L. Giordano and A. Martelli. On-the-fly Automata Construction for Dynamic Linear Time Temporal Logic. *TIME 04*, June 2004.

[11] L. Giordano, A. Martelli, and C. Schwind. Reasoning About Actions in Dynamic Linear Time Temporal Logic. In *The Logic Journal of the IGPL*, Vol. 9, No. 2, pp. 289-303, March 2001.

[12] L. Giordano, A. Martelli, and C. Schwind. Verifying Communicating Agents by Model Checking in a Temporal Action Logic. JELIA 2004, Lisbon, Portugal, September 27-30, 2004, pp. 57-69.

[13] L. Giordano, A. Martelli, and C. Schwind. Specifying and Verifying Interaction Protocols in a Temporal Action Logic. *Journal of Applied Logic (Special issue on Logic Based Agent Verification)*, Accepted for publication, 2005.

[14] F. Giunchiglia and P. Traverso. Planning as Model Checking. In *Proc. The 5th European Conf. in Planning (ECP'99)*, pp.1–20, Durham (UK), 1999.

[15] C. Gordon and J.P. Christensen, Health Telematics for Clinical Guidelines and Protocols. IOS Press, Amsterdam, 1995.

[16] M. Greaves, H. Holmback and J. Bradshaw. What Is a Conversation Policy?. *Issues in Agent Communication*,LNCS 1916 Springer, pp. 118-131, 2000.

[17] F. Guerin. Specifying Agent Communication Languages. PhD Thesis, Imperial College, London, April 2002.

[18] F. Guerin and J. Pitt. Verification and Compliance Testing. *Communications in Multiagent Systems*, Springer LNAI 2650, pp. 98–112, 2003.

[19] J.G. Henriksen and P.S. Thiagarajan. Dynamic Linear Time Temporal Logic. in *Annals of Pure and Applied logic*, vol.96, n.1-3, pp.187–207, 1999

[20] G.J. Holzmann *The SPIN Model Checker. Primer and Reference Manual*. Addison-Wesley, 2003

[21] Y. Labrou and T. Finin, A semantic approach for KQML - a general purpose communication language for software agents. In *3rd Int Conf. on Information and Knowledge Management, CIKM'94*, pp.447-455, 1994.

[22] M.P. Huget and M. Wooldridge. Model Checking for ACL Compliance Verification. *ACL 2003*, Springer LNCS 2922, pp. 75–90, 2003.

[23] M. Marcos, M. Balser, A. ten Teije, F. van Harmelen, C. Duelli Experiences in the formalisation and verification of medical protocols, AIME'03.

[24] N. Maudet and B. Chaib-draa. Commitment-based and dialogue-game based protocols: new trends in agent communication languages. In *The Knowledge Engineering Review*, 17(2):157-179, June 2002.

[25] S. Narayanan and S. McIlraith. Simulation, Verification and Automated Composition of Web Services. In *Proceedings of the Eleventh International World Wide Web Conference (WWW-11)*, May, 2002.

[26] M.Pistore and P.Traverso. Planning as Model Checking for Extended Goals in Non-deterministic Domains. Proc. IJCAI'01, Seattle, pp.479-484, 2001.

[27] R. Reiter. The frame problem in the situation calculus: a simple solution (sometimes) and a completeness result for goal regression. In *Artificial Intelligence and Mathematical Theory of Computation: Papers in Honor of John McCarthy*, V. Lifschitz, ed.,pages 359–380, Academic Press, 1991.

[28] M. P. Singh. A social semantics for Agent Communication Languages. In *IJCAI-98 Workshop on Agent Communication Languages*, Springer, Berlin, 2000.

[29] P. Terenziani, G. Molino, and M. Torchio, A Modular Approach for Representing and Executing Clinical Guidelines. Artificial Intelligence in Medicine 23, 249-276, 2001.

[30] P. Terenziani, C. Carlini, S. Montani. Towards a Comprehensive Treatment of Temporal Constraints in Clinical Guidelines. Proc. TIME'02, Manchester, UK, IEEE Press, 20-27, 2002.

[31] Terenziani, P., Montani, S., Bottrighi, A., Torchio, M., Molino, G., Correndo, G. A context-adaptable approach to clinical guidelines. Proc. MEDINFO'04, M. Fieschi et al. (eds), Amsterdam, IOS Press, (2004), 169-173.

[32] P. Terenziani, S. Montani, A. Bottrighi. Exploiting Decision Theory for Supporting Therapy Selection in Computerized Guidelines. Proc. Int'l Conf. Artificial Intelligence in Medicine Europe, LNCS, Sprinter Verlag, 2005.

[33] P.Traverso and M.Pistore. Automated Composition of Semantic Web Services into Executable Processes. Proc. Third International Semantic Web Conference (ISWC2004), November 9-11, 2004, Hiroshima, Japan.

[34] M. Wooldridge, M. Fisher, M.P. Huget and S. Parsons. Model Checking Multi-Agent Systems with MABLE. In *AAMAS'02*, pp. 952–959, Bologna, Italy, 2002.

[35] P. Yolum and M.P. Singh. Flexible Protocol Specification and Execution: Applying Event Calculus Planning using Commitments. In *AAMAS'02*, pp. 527–534, Bologna, Italy, 2002.