ELSEVIER

Artificial Intelligence in Medicine 703 (2003) 1–26

Artificial Intelligence in Medicine

www.elsevier.com/locate/artmed

Automated monitoring of medical protocols: a secure and distributed architecture

T. Alsinet*, C. Ansótegui, R. Béjar, C. Fernández, F. Manyà

Department of Computer Science, University de Lleida Jaume II 69, E-25001 Lleida, Spain

Received 15 October 2002; received in revised form 15 October 2002; accepted 18 October 2002

Abstract

The control of the right application of medical protocols is a key issue in hospital environments. For the automated monitoring of medical protocols, we need a domain-independent language for their representation and a fully, or semi, autonomous system that understands the protocols and supervises their application. In this paper we describe a specification language and a multi-agent system architecture for monitoring medical protocols. We model medical services in hospital environments as specialized domain agents and interpret a medical protocol as a negotiation process between agents. A medical service can be involved in multiple medical protocols, and so specialized domain agents are independent of negotiation processes and autonomous system agents perform monitoring tasks. We present the detailed architecture of the system agents and of an important domain agent, the database broker agent, that is responsible of obtaining relevant information about the clinical history of patients. We also describe how we tackle the problems of privacy, integrity and authentication during the process of exchanging information between agents.

Keywords: Medical protocols; Specification and representation; Multi-agent system architecture; Distributed monitoring; Secure communication layer

1. Introduction

In recent years physicians have made an important effort to formalize medical protocols (MP) for the diagnosis and treatment of diseases. MPs published by medical associations, like the American Heart Association, the European Resuscitation Council and the Australian Medical Association, can be seen as the consensus opinion of experts based

* Corresponding author. Tel.: +34-973-702-734; fax: +34-973-702-702
E-mail addresses: tracy@eup.udl.es (T. Alsinet), carlos@eup.udl.es (C. Ansótegui), ramon@eup.udl.es (R. Béjar), cesar@eup.udl.es (C. Fernández), felip@eup.udl.es (F. Manyà).

0933-3657/03/$ – see front matter © 2003 Published by Elsevier Science B.V.
doi:10.1016/S0933-3657(03)00010-1

on systematic review of the scientific literature. This effort is devoted to improve both decision making in healthcare processes and optimal limited resource management.

According to Gordon [14] MPs can be defined as systematically developed statements to assist practitioner and patient decisions about appropriate health care for specific clinical circumstances. To help physicians on the right application of MPs in real hospital environments, the Artificial Intelligence community has considered the challenge of automatic management of MPs. For achieving this objective two different problems have been considered.

On the one hand, it is necessary to tackle the automated application and the monitoring problem. That is, to develop a general-purpose system devoted both to coordinate practitioners and patients during a real-time application of a MP and to warn about forbidden actions and decisions. On the other hand, for performing the monitoring task, it is necessary to have a suitable and well-defined representation of MPs. For this purpose we propose a domain-independent language for their representation and so, a user-friendly specification tool for allowing physicians to model MPs and translating the MP models into the representation language.

There exist different approaches and tools in the literature for representing and modeling MPs, but not for the automated application and the monitoring problem when considering real hospital environments. In such environments, one has to develop a system able to run in a real-time distributed frame and, because of the nature of the information exchanged in the distributed system, one has also to provide a secure communication layer. Some well-known approaches for modeling MPs are those of DGPM [15], GLIF [26,28]. It should be noticed that those approaches, except the extension of GLIF [28], are not necessarily well-suited to support the automated application and monitoring of the MPs. As to the automated application of the MPs, Boxwala et al. [5] have defined a system based on the GLIF representation model. Their approach is aimed at clinical decision-support and medical education. Other approaches facing this aim are proforma [10,9] and the projects Asgaard [37] and ISAAC [4]. The PRODIGY system [30] is a guideline-based decision-support system designed to assist general practitioners in choosing the appropriate therapeutic action for their patients and has been extended [21] for encoding clinical guidelines for managing patients. Although these approaches help physicians to test and evaluate knowledge of MPs recommendations, they do not consider the integration of a MP agreed by consensus into the medical practice, i.e. the automatic application and monitoring of the MP for a given patient in a hospital environment with existing medical records. In this framework, Terenziani et al. [42] and Alsinet et al. [2,3] have defined modular systems for representing and applying MPs. Both systems [42,2] consist of a user-friendly environment for representing and acquiring MPs involving temporal issues as the treatment of composite, concurrent and repeated actions. However, [42] defines a centralized system architecture (i.e. to be executed in a local way) while [3] defines a multi-agent system architecture for distributed hospital environments.

The objective of this paper is to define a system for the assistance and supervision of the real-time application of MPs in distributed hospital environments with computerized-based medical records. A MP specifies possible sequences of composite, concurrent and repeated actions that could be performed when a patient had a particular pathology and, depending on clinical results, establishes which of them are forbidden. In hospital

environments, every staff person plays one or more roles. A role specifies a particular service; for instance, infirmary, surgery, radiology or hospital management. So, a MP specifies possible interactions between medical services in front of a particular pathology and forbids some medical decisions depending on the medical history and evolution of the patient. Our system can be used for managing both suggested actions and constrained actions. To be precise, suggestions are modeled as multiple possible actions and constraints as forbidden actions. So, it can be used for managing both medical guidelines and MPs. From now on, we refer to the more general term, i.e. MP.

To define a system for monitoring MPs, we make an abstraction of the real world, i.e. hospital environment, in terms of a multi-agent system (MAS) [41,45]. The main idea is to model medical services in hospitals as specialized domain agents and interactions between different services as electronic communication processes. From this point of view, a MP describes a negotiation process between multiple specialized domain agents for treating a particular pathology and specifies behavior rules depending on specific symptoms.

To tackle this problem, the first step is to develop a tool and a representation model for hospital environment specification in terms of MASs which, in particular, allow us to model medical services as specialized domain agents and MPs as restricted negotiation processes. In this context, we have implemented a user-friendly environment (called JAFDIS) [2] to graphically represent negotiation processes in MASs based on the notion of dialogical institutions. The second step is to develop a suitable MAS architecture for real-time monitoring MPs in distributed hospital environments. In distributed MASs, agent interactions are performed as electronic communications through a communication layer. Because of the private nature of most of the information exchanged in a hospital environment, the communication layer must provide privacy, integrity and authentication during the process of exchanging information between agents. Therefore, the third step is to define a robust communication interface. Finally, in order to make our system fully compliant with the existing medical records, we develop a database broker agent for automatically recovering the medical history and evolution of patients.

The remainder of the paper is organized as follows. Section 2 formalizes the concept of dialogical institution and defines its two basic components, dialogical framework and performative structure. Section 3 shows how a MP can be specified using this formalism. Section 4 describes how a dialogical institution can be modeled using JAFDIS. Section 5 presents the architecture of our MAS for monitoring MPs. Section 6 deals with all necessary security and communication aspects to ensure the privacy of the information exchanged between agents. Sections 7 and 8 describe the functional requirements of two main components of the MAS architecture: the server and supervisor agents of MPs. Section 9 shows the mediator functionalities of the interface between the MAS architecture and the medical services. Section 10 describes the functionalities of the database broker agent. Finally, in Section 11, we summarize our main results and point out our future work.

2. Dialogical institutions

In this section we provide a new definition of dialogical institution, based on [2], which is *formed by a dialogical framework* and a *performative structure*. This definition allow us

to model a traditional institution (say a hospital) and its set of institutional protocols (say MPs) by detailing what kind of agents exist and what type of messages or illocutions are interchanged between agents. The dialogical framework describes the general context of the institution and the performative structure describes the set of protocols. In [29,32,7,35] the authors propose different approaches for modeling some particular institutions.

In what concerns our work, we think of agent-based institutions as a computational realization of a traditional institution, which intuitively amounts to a set of clearly established conventions that somehow restrict participating agent's interactions. Therefore, a dialogical institution can be understood as a set of agents and a set of protocols so that, for every protocol, there exists a well-defined behavior of the involved agents. An agent is an entity that plays a specific institution role. Moreover, an agent can play different roles in different protocols and in different scenes of a same protocol. A protocol is a non-empty set of well-defined scenes.

In a dialogical institution agents are ''dialogical entities'' that interact with other agents in a multi-agent context through illocutions. An agent is an entity that expresses illocutions, reacts to illocutions addressed to it and, furthermore, only the emission and reception of illocutions constitute its observable behavior.

Definition 1 (Dialogical institution). A dialogical institution is a pair $\langle \mathscr{DF}, \mathscr{PS} \rangle$, where

- $\mathscr{DF}$ is a dialogical framework and
- $\mathscr{PS}$ is a performative structure that defines the set of protocols of the institution in which the system agents are involved.

2.1. Dialogical framework

The dialogical framework describes the context of a dialogical institution. On the one hand, it specifies the basic roles and the agents that take place inside the institution. On the other hand, it formalizes the communication language of the system agents and the metalanguage used to model the rules of behavior inside a protocol.

Definition 2 (Dialogical framework). A dialogical framework is a tuple of the form $\langle \text{Roles}, \text{Agents}, L, \text{CL}, \text{TL}, \text{ML}, T \rangle$, where

- Roles is a set of different kinds of participants associated with the institution,
- Agents denotes a set of role occurrences,
- L is an object language and describes the syntactic rules for generating the system identifiers, the agents names, and the illocutions content,
- CL is a communication language and defines the set of illocutions that can be exchanged between the system agents,
- TL is the transition language that allows one to define the movements of agents between scenes within a specific protocol,
- ML is the metalanguage used for restricting the agents behavior within a specific protocol, and
- T is a model of time.

At the specification phase, one selects a specific model of time, i.e. a specific time format, by means of a unique identifier. The monitoring system must provide a suitable implementation of every model of time. For example, a model of time can be the usual time format indicating seconds, minutes and hours referring to the Greenwich Meridian Time together with the set of arithmetic and relational operators. Another possible model of time[1] is the one based on the notion of days, weeks and months, where the day is the basic unit of time.

Definition 3 (Communication language). The syntax of an illocution of CL is as follows: illocution_id (agent$_1$, agent$_2$, φ, t), where

- illocution_id $\in L$ is the illocution identifier,
- agent$_1$ $\in$ Agents is the sender of the illocution and agent$_2$ $\in$ Agents is the recipient of the illocution,
- $\varphi \in L$ is the content of the illocution, and
- t is the illocution time expressed according to the time model T.

Definition 4 (Transition language). The syntax of a transition of TL is as follows: move (kind_of_movement, $\{a_1, \ldots, a_n\}$, t), where

- move $\in L$ is the label of the transition,
- kind_of_movement $\in L$ is either the label IN $\in L$ or the label OUT $\in L$ and allows one to model arrivals/departures to/from the scenes within a specific protocol,
- $\{a_1, \ldots, a_n\} \subseteq$ Agents describes the set of agents that should move between the scenes of a specific protocol, and
- t is the transition time expressed according to the time model T.

At the specification phase, the illocution time and the transition time denote time variables which are instantiated by the monitoring system when executing a protocol of the institution. The time variables allow us to specify forbidden actions based on temporal constraints and timeout execution errors.

2.2. Performative structure

The performative structure is a set of protocols. A protocol is a main scene and a set of interdependent sub-scenes and states that allows us to establish causal and temporal co-dependencies among them.

Intuitively, a protocol of a dialogical institution can be formally specified by means of an extension of a Petri net. A Petri net [6] has four basic components: places, transitions, arcs, and tokens. A place represents a state in which the system may be. Tokens indicate the current states of the system. A transition has zero or more input arcs, coming from its input places, and zero or more output arcs, going to its output places. A transition is enabled if there is at least one input token in each of its input places. Any enabled transition is executed by removing one token from each input place and depositing a token in each

[1] The example of Section 3 uses this model of time.

output place. The execution of MPs in our system follows the same basic rules as in the execution of Petri nets.

Translating Petri nets into our setting, tokens are associated with agents. Petri nets states are associated with scenes and states of our protocols. We identify three different types of states: atomic, synchronism and parallel states. Petri nets arcs between states are associated with illocutions and agent movements. Specifically, each scene and each state is defined as a set of active agents, where each agent plays an institutional role. Each pair of agents that exchange an illocution are subject to a common atomic interaction represented as an illocution arc between two states. The movement arcs are reserved for representing the arrival/departure of agents to/from a scene. This means that movement arcs are only allowed between scenes and synchronism or parallel states. A synchronism state models a synchronized arrival of some agents into the current scene from a different scene. That is, the current scene is suspended until the arrival movements of agents from the other scene are executed. A parallel state models a parallel departure of some agents from the current scene to a different scene. That is, the current scene and the new scene are concurrently executed. Finally, to formalize the interaction between the dialogical institution and the outside we assume the existence of an external scene. The arrival of external agents into a scene is modeled by a synchronism node and the departure is modeled by a parallel node.

Definition 5 (Performative structure). A performative structure of a dialogical institution is a set of protocols. A protocol is a tuple of the form $\langle \text{id\_protocol}, S, s_0 \rangle$, where

- id_protocol $\in L$ is the protocol identifier,
- S is the non-empty set of scenes involved with the protocol,
- $s_0 \in S$ is the initial (main) scene of the protocol. The execution of a protocol starts by executing the initial scene.

Definition 6 (Scene). A scene is a tuple of the form

$$\langle \text{id\_scene}, \{a_1, \ldots, a_n\}, \mathcal{SDG}, \mathcal{BR} \rangle,$$

where

- id_scene $\in L$ is the scene identifier,
- $\{a_1, \ldots, a_n\} \subseteq$ Agents describes the non-empty set of initial agents (i.e. the set of agents that must be active when starting the execution of the scene),
- $\mathcal{SDG}$ is the scene dependence graph,
- $\mathcal{BR}$ is a set of rules that restrict the behavior of agents within the scene. Every behavior rule is defined over the metalanguage ML and has the following general syntax:

  $$\text{if } C \text{ then } \{\neg\delta_1, \ldots, \neg\delta_n\},$$

  where C is a Boolean expression built over the language $L \cup \text{CL} \cup \text{TL}$, and δ_i, $i \in \{1, \ldots, n\}$ with $n \geq 1$, is a transition of the scene dependence graph.

All the system restrictions, as the rules of behavior and the initial set of agents, are checked during the monitoring phase when executing a protocol.

Definition 7 (Scene dependence graph). A scene dependence graph is a tuple of the form $\langle W, \delta, w_0, W_T \rangle$, where

- W is a set of nodes formed by a set of scenes, a set of states and the designated external scene,
- δ is the set of transitions of the dependence graph,
- $w_0 \in W$ is the initial node of the scene dependence graph, and
- $W_T \subseteq W$ is the non-empty set of final nodes of the scene dependence graph.

The execution of a scene starts at the initial node and finishes at a final node.

Definition 8 (State). A state is a tuple of the form $\langle \text{id\_state}, \text{kind\_of\_state}, \text{timeout} \rangle$, where

- id_state $\in L$ is the state identifier,
- kind_of_state $\in L$ is either the label ATOMIC $\in L$, or the label SYNCHRONISM $\in L$, or the label PARALLEL $\in L$ (i.e. a state is either an atomic state, or a synchronism state, or a parallel state),
- *timeout* is a time constant expressed according to the time model T of the dialogical institution.

The initial node of a scene dependence graph is an atomic state. A scene cannot be used as a final node of a scene dependence graph. When monitoring a protocol, its execution starts in the initial scene at its initial state. Moreover, the initial set of agents of the scene is the set of active agents of the initial state. Therefore, the set of active agents for each other state of the scene is dynamically computed from the initial set and the execution trace. Finally, the timeout of a state specifies the maximum delay allowed for leaving the state during the execution phase (i.e. the maximum delay allowed for performing a valid transition). The time model of the dialogical institution must contain a special time constant for specifying that there is no timeout constraint.

In the following definition S denotes a finite set of scenes, A denotes a finite set of atomic states, E denotes the designated external scene, Sync denotes a finite set of synchronism states, and Pll denotes a finite set of parallel states.

Definition 9 (Dependence graph transitions). A set of transitions δ of a scene dependence graph is a pair of mappings $\langle \rho_1, \rho_2 \rangle$ between the graph nodes. The mapping ρ_1 specifies the allowed transitions inside the current scene. The mapping ρ_2 specifies the allowed transitions between the current scene and other scenes of the protocol.

The mapping ρ_1 is formalized as follows:

$$\rho_1 : (A \cup \text{Pll} \cup \text{Sync}) \times \text{CL} \rightarrow A$$

The mapping ρ_2 is formalized as follows:

$$\rho_2 : (A \cup \text{Pll} \cup \text{Sync}) \times (S \cup E) \times \text{TL} \rightarrow (\text{Pll} \cup \text{Sync}),$$

The mapping ρ_2 must also satisfy the following constraints:

- If $\rho_2(a, s, \text{transition}) \in \text{Pll}$, where $a \in A \cup \text{Pll} \cup \text{Sync}$, $s \in S \cup E$ and transition $\in \text{TL}$, then transition is a departure movement (i.e. an output transition) and

- If $\rho_2(a, s, \text{transition}) \in \text{Sync}$, where $a \in A \cup \text{Pll} \cup \text{Sync}$, $s \in S \cup E$ and transition $\in$ TL, then transition is an arrival movement (i.e. an input transition).

The set of constraints imposed by the mappings ρ_1 and ρ_2 are checked during the protocol specification.

Notice that neither mapping ρ_1 nor mapping ρ_2 restrict the number of transitions for a given state. Therefore, it is possible to reach a state from different source states and from a state one can reach multiple target states. When a state acts as the source of multiple transitions, the state has to be understood as a decision state of the scene. At the specification phase, multiple transitions are constrained by the behavior rules. However, it is during the execution phase of a protocol that one can know which of these multiple transitions are allowed by the behavior rules. Finally, during the execution phase, if more than one transition is allowed, the active agents at the decision state have to select one of them. Moreover, if one maintains a history about agents decisions for protocol executions, this database could be used as knowledge source for refining the protocol specification.

3. Specification of a medical protocol for controlling hypertension

For illustrative purposes, we next describe an example of the use of our modeling language to specify a real medical protocol for detecting and controlling hypertension.[2] The protocol involves the participation of a patient, a physician and a nurse. After an initial diagnostic phase of the severity of the high pressure condition, the patient must start a controlling phase for checking the high pressure evolution. Since the controlling phase is general enough to exemplify the use of all the elements of our specification language, we concentrate on this phase. Fig. 1 shows an algorithmic description of the controlling phase of the hypertension protocol. Pressure is determined by means of two measures: siastolic blood pressure (SBP) and diastolic blood pressure (DBP). Good pressure levels are assumed when the patient SBP and DBP are less than 140 and 90, respectively. Acceptable pressure levels are assumed when the patient SBP and DBP are in the integer and semi-open interval $[140, 160)$ and $[90, 95)$, respectively. Dangerous pressure levels are assumed when the patient SBP and DBP are greater than 160 and 95, respectively. For good and acceptable pressure levels, patients must follow a periodic basic test, every 3 months, for checking their pressure. For dangerous pressure levels, patients must follow a periodic and exhaustive test, every 2 weeks, for checking their pressure and possible side effects. Usually, the basic test is performed by the nurse and the exhaustive test is performed by the physician. Once the patient has successfully passed three consecutive basic tests, he must follow an annual checkup program.

In our modeling language, the controlling phase of the hypertension protocol is specified by means of an scene, called the controlling scene. Fig. 2 shows the scene dependence graph (SDG) of the controlling scene. Tables 1 and 2 show the illocutions and transitions of the SDG, respectively.

[2] This protocol has been provided by a clinic of Tarragona, Spain

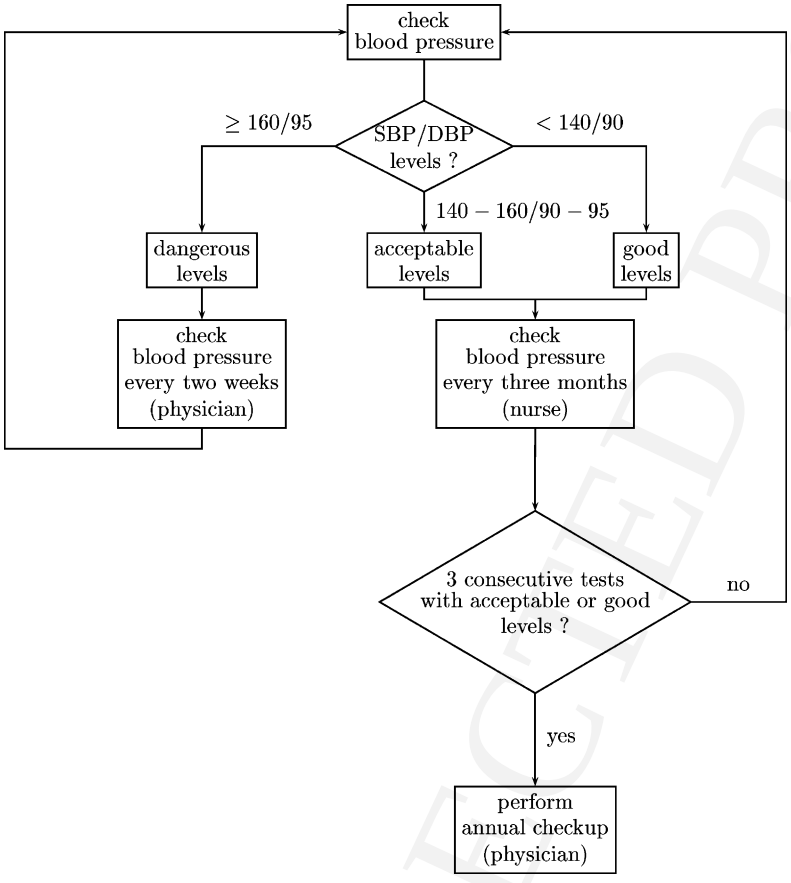

Fig. 1. Algorithm of the controlling phase for the hypertension protocol.

The agents involved in the controlling scene are a patient, a physician and a nurse, and the initial agent of the scene is the nurse. The first action that takes place in the scene is the synchronization between the nurse and the patient (transition T_1). The nurse is located in the initial state S_0 and the patient comes from the diagnostic scene (scene D at the SDG). At the synchronization state Sync_1, the nurse checks the blood pressure of the patient.

Table 1
Illocutions of the SDG of the controlling scene

Identifiers	Illocution
i_1, i_4, i_9, i_{14}, i_{19}	Inform (nurse, patient, ''dangerous levels'')
i_2, i_5, i_{10}, i_{15}, i_{20}	Inform (nurse, patient, ''acceptable levels'')
i_3, i_6, i_{11}, i_{16}, i_{21}	Inform (nurse, patient, ''good levels'')
i_7, i_8, i_{12}, i_{13}, i_{17}, i_{18}	Request (nurse, patient, ''perform basic test'')
i_{22}	Request (nurse, patient, ''perform exhaustive test'')

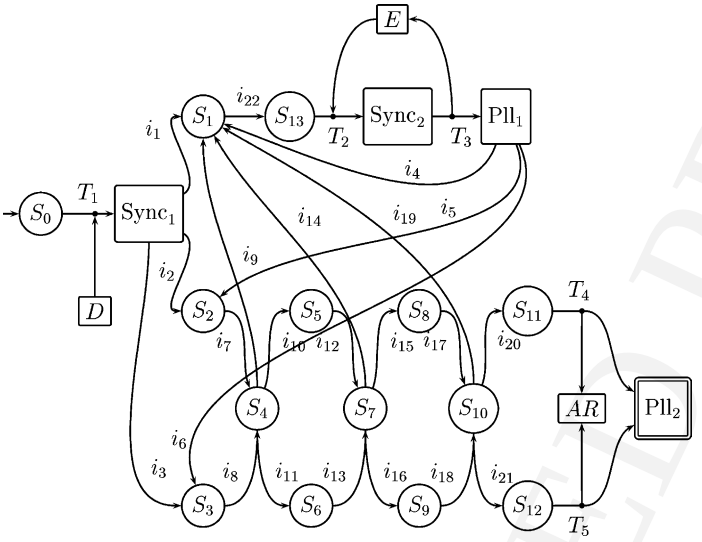

Fig. 2. SDG of the controlling scene for the hypertension protocol.

Depending on the SBP and DBP levels, the nurse emits either illocution i_1, or i_2, or i_3. The right illocution is determined by means of behavior rules.

On the one hand, when the patient SBP and DBP levels are dangerous, the nurse should emit illocution i_1 and then illocution i_{22} with a maximum delay of 2 weeks. This fact is specified by means of a finite timeout attached to state S_1. Since the exhaustive test must be performed by the physician, he must come in the scene (transition T_2). After performing this test, as in state Sync_1 and depending on the SBP and DBP levels, the nurse emits either illocution i_4, or i_5, or i_6, but, in any case, the physician abandons the scene and goes to the external scene (scene E at the SDG) by means of transition T_3.

On the other hand, when the SBP and DBP levels of the patient are acceptable, the nurse should emit illocutions i_2 and then i_7. If levels are good, the nurse should emit illocutions i_3 and then i_8. In both cases, illocutions must have a maximum delay of 3 months. This fact is specified by means of a finite timeout attached to states S_2 and S_3. After performing the basic test, as in state Sync_1 and depending on the SBP and DBP levels, the nurse emits either illocution i_9, or i_{10}, or i_{11}. If the patient successfully passes[3] two more consecutive basic tests, the system reaches either state S_{11}, if the SBP and DBP levels for the last basic test are acceptable, or state S_{12}, if the levels are good. In both cases, the patient must be sent to the annual checkup scene (scene AR at the SDG) by means of transactions T_4 and T_5.

Finally, the behavior rules attached with state Sync_1 are the following ones:

- If (SBP ≥ 160 and DBP ≥ 95) then $\{\neg i_2, \neg i_3\}$.
- If (SBP $\in [140, 160)$ and DBP $\in [90, 95)$) then $\{\neg i_1, \neg i_3\}$.
- If (SBP < 140 and DBP < 90) then $\{\neg i_1, \neg i_2\}$.

The behavior rules attached with states Pll_1, S_4, S_7 and S_{10} are defined in an analogous way.

[3] A patient successfully passes a basic test whenever the SBP and DBP levels are either acceptable or good.

Table 2
Transitions of the SDG of the controlling scene

Identifiers	Transition
T_1	Move (IN, {patient})
T_2	Move (IN, {physician})
T_3	Move (OUT, {physician})
T_4, T_5	Move (OUT, {patient})

4. Modeling a dialogical institution with JAFDIS

JAFDIS [2] offers the possibility of working either in graphical mode or in text mode. That is, the scene dependence graph can be specified either by loading a textual description in a formal language or by drawing the graph using a graph editor. The graphical mode is supported by a full-featured graphical user-interface which incorporates MDI technology (multiple document interface) in order to display the context of all the scenes that configure a protocol in a common frame.

The definition of a new dialogical institution can be performed in two different ways: static and dynamic. The static way refers to the fact of explicitly specifying the dialogical framework by means of menu commands before specifying the performative structure. The dynamic way refers to the fact of implicitly specifying the dialogical framework while specifying the performative structure.

JAFDIS provides a unified frame for specifying all the components of a dialogical institution. The frame is divided into four areas (see Fig. 3):

(1) The *menu* and the *toolbar* area.
(2) The *tabPane* area with two options: S (Scenes) and P (Paint).
   - The S option shows the hierarchy structure of scenes that make up a protocol.
   - The P option shows a drawing tool for graphically designing the scenes dependence graphs.
(3) The *WorkSpace* area shows each scene dependence graph in a different window.
(4) The *Debugger* area shows errors generated when checking the consistency of the scenes dependence graphs.

The dialogical framework describes the basic components of the dialogical institution. JAFDIS provides different dialog boxes for graphically specifying these components.

For illustrative purposes, we present the JAFDIS dialog boxes for agents management. Fig. 4 shows the dialog box for specifying the set of agents. The *Properties* option allows us to specify their main features (see Fig. 5) identification, description and role. As an agent can play different roles in the institution and a role can be associated with different agents, JAFDIS provides a dialog box (see Fig. 6) for specifying the relations between roles and agents.

The performative structure is the set of protocols of the dialogical institution. When specifying a new protocol the initial scene is created by default. Every scene has a graphical representation of the scene dependence graph. The basic components of a scene dependence graph are nodes and transitions. Nodes are atomic states, synchronism states, parallel

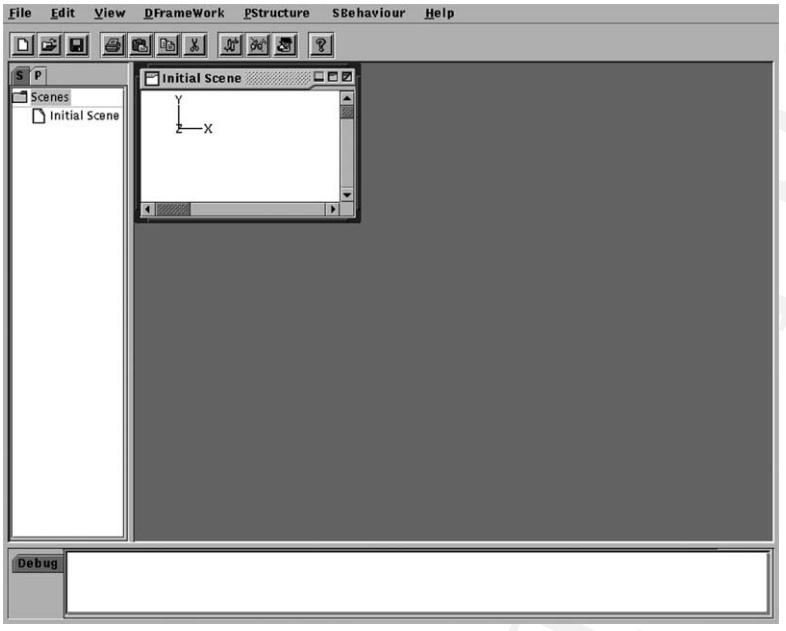

Fig. 3. JAFDIS frame distribution.

states and scenes. Transitions are illocutions and movements. Figs. 7 and 8 show the dialog boxes for specifying nodes and illocutions, respectively. When specifying nodes one can establish the behavior rules (*R. Behavior* option) that restrict the available transition during the execution phase and the timeout parameter (*Constraints* option). When specifying transitions one can establish, during the execution phase, whether the time in which the transition is performed must be recorded (*mtime* option).

Every scene dependence graph must satisfy the following constraints: the scene dependence graph must be connected, must have at least one initial and one final node, and every node must be accessible and useful. A node is accessible when there is at least a

Fig. 4. Agents dialog box.

Fig. 5. Agent properties dialog box.

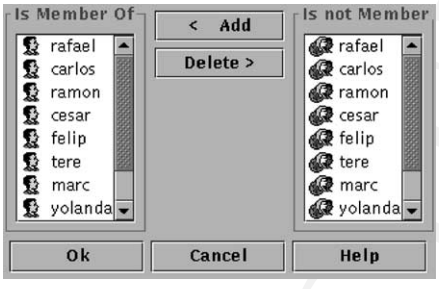

Fig. 6. Agent roles dialog box.

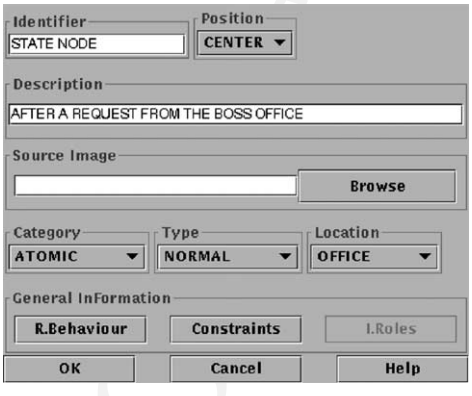

Fig. 7. Nodes properties dialog box.

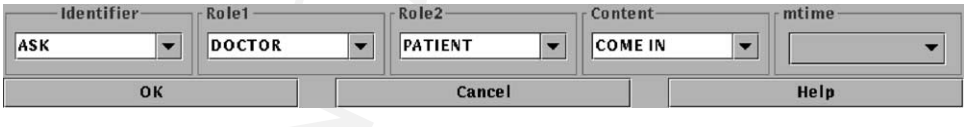

Fig. 8. Illocution properties dialog box.

path from the initial node to this node, and a node is useful when there is at least a path from the node to a final node. A depth-first search algorithm is used to check if the scene dependence graph satisfies these constraints.

5. Multi-agent system architecture

Nowadays, MASs play a central role in the field of distributed artificial intelligence. MAS applications go from electronic commerce [31,25], aircraft maintenance [38] and business process management [17,19] to e_mail filtering [23]. In the following, an agent has to be understood as a software component that exhibits some aspects of intelligent human behavior [44] and that interacts with its environment. In what concerns our work, we identify three key issues in the design and construction of MASs: negotiation models between autonomous software agents [8,27,45,18,20], agent architectures [33,34,12] and multi-agent architectures [41].

In Section 2 we have defined a representation model for specifying hospital institutions in terms of MASs which, in particular, allows us to model medical services as specialized domain agents (SDAs) and MPs as restricted negotiation processes. For identifying the suitable MAS architecture for monitoring MPs we have considered two different approaches:

(1) To provide SDAs with domain and monitoring knowledge [16]. This approach is based on a layered architecture in which every layer should manage knowledge of a different nature. For instance, the base layer could contain the specific domain knowledge (medical specialty); the middle layer, the specification of MPs related to the specialty; and the upper layer, the monitoring and decision control.
(2) To distribute the domain and monitoring knowledge [41] among system agents. On the one hand, SDAs could manage the specific domain knowledge and the decision control, i.e. at least a two layer agent architecture. On the other hand, autonomous system agents (ASAs) could control monitoring tasks and manage MP specifications.

As a MP can involve multiple medical services, the first approach should keep a same MP specification in multiple SDAs. Moreover, to check forbidden medical decisions could be necessary to know the result of all SDA interactions. So, we can easily see that this approach should maintain a great amount of duplicated information and SDAs should work in a synchronized way, i.e. at every monitoring step all SDAs should know the same information about previous agent interactions. The distributed knowledge approach avoids these problems and allows SDAs to concurrently execute multiple MPs by assigning a different ASA to every monitoring process.

In the rest of the paper we make a step towards the formalization of a MAS architecture, based on the distributed knowledge approach, for monitoring MPs by (i) defining a secure communication interface and a certification agent; (ii) providing the system with a server agent and supervisor agents of MPs; and (iii) introducing mediator agents, called interagents, that act as a bridge between SDAs and the rest of the system. SDAs can be of different kinds, i.e. from autonomous software agents to human agents. To support human agents, interagents offer a user-friendly interface accessible from any point of the Internet using a browser.

We address the problem of monitoring MPs using a distributed knowledge MAS that allows us to define a distributed environment with secure and robust interactions. Our proposal is based on a computational system composed of several SDAs—one for every medical service in hospital environments—and a set of ASAs that will offer specific services for solving the MPs monitoring problem.

Before describing the architecture of the MAS, we identify the fundamental services that the system should provide to agents:[4]

Dynamic location: The system must be accessible from any point of the Internet (or of the hospital intranet) and allowing agents to interact independently of their physical location (the network address). Thus, we must provide the system with distributed location tables which will be dynamically maintained.

Information confidentiality: Agent interactions can involve private information about patients. For instance, urine and blood tests, X-rays or biopsy results. Moreover, we have seen that the communication channel of the system is the Internet, which does not guarantee privacy and integrity. Therefore, in order to ensure that only the addressee agent of a message can interpret the content, all messages will be sent encrypted.

Information integrity: During the communication processes between agents, the system must be able to detect a third party message manipulation. Information manipulation includes such things as insertion, deletion and substitution.

Agents authentication: In order to ensure the identity of the sender agent during the communication processes between agents, all messages will be digitally signed. Message origin authentication implicitly provides information integrity.

Medical protocols monitoring: A MP describes a negotiation process between multiple SDAs for treating a particular pathology and forbids some SDA actions depending on the result of previous interactions. So, when executing a MP, the system must supervise all SDA interactions in order to register invalid actions and to assist SDA decisions.

Medical protocols delivery: As a MP can involve multiple SDAs, the system must manage MP specifications and deliver them with a suitable format.

These services are provided by a secure communication interface and some ASAs. SDAs interact with ASAs through interagents allowing SDAs concurrently execute multiple MPs. The secure communication interface ensures that (i) messages are sent encrypted and digitally signed, and (ii) received messages are authentic. The remaining services are provided by the following ASAs:

- The **certification agent** (**CA**) acts as a certification authority. The first time an agent interacts with the system it must contact with the CA to obtain its certificate. Agents need their certificates for signing messages.
- **Supervisor agents** (**SA**) track interactions between SDAs and verify their validity.
- The **medical protocols server agent** (**MPSA**) performs two main system services. On the one hand, it distributes the list of available MPs to SDAs depending on their hospital roles. On the other hand, it distributes MP specifications to SAs.

We group agent interactions in two phases. The first one (Fig. 9) includes agents certification, agent addresses distribution and MPs delivery. The second one (Fig. 10)

[4] From now on, by an agent we understand a SDA or an ASA.

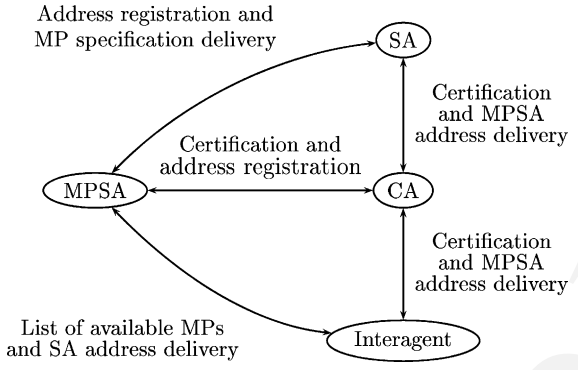

Fig. 9. Certification and information delivery phase.

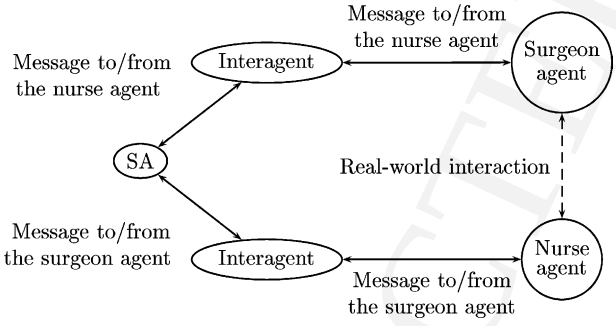

Fig. 10. Monitoring phase.

corresponds to the MPs monitoring process. During the first phase, the central component of the system is the CA. All agents must contact with the CA to obtain their certificates. The first agent that must contact with the CA is the MPSA. The CA stores the MPSA address. A SA contacts with the CA for obtaining the MPSA address. At this point the SA provides its address to the MPSA. This strategy allows us to dynamically distribute the system monitoring load depending on the number of active SAs. A SDA contacts, through an interagent, with the CA for obtaining the MPSA address. Next, the interagent contacts with the MPSA for obtaining the list of available MPs depending on the SDA hospital role. The SDA decides which MP has to be executed. When all SDAs involved in the MP have contacted with the MPSA (following the described protocol), the MPSA assigns a SA for monitoring the MP execution, delivers to interagents the SA address and the monitoring phase starts. From now on, interagents forward SDAs interactions to the SA.

6. Communication interface

The private nature of most of the information exchanged in a hospital environment and the need of authentication led us to use cryptographic tools that we incorporated into a

communication interface. Furthermore, these requirements are imposed, in most countries, by governmental laws related to the electronic processing of personal data. An excellent analysis of the types of threats that can impact on a MAS can be found in [43]. Authors also provide architectural solutions to deal with security and trust issues.

Our objective, as in most cryptographic systems, has been to provide confidentiality, integrity and authentication to the dialog between agents of our MAS. These features have to be transparent to the user of the communication layer. As it is difficult to find cryptographic classes written in Java, without distribution and license restrictions, we have implemented a complete cryptosystem based on the Secure Socket Layer specification [11].

Our approach is based on a certification authority architecture [22]. This architecture has been used, instead of using a public key server approach, in order to avoid a full storage of the public key of agents and a bottleneck in the system. The potential bottleneck would obey to the fact that an agent, each time it must interact with a new one, must obtain the public key of the new agent through the public key server. The certification authority (in our system, the CA) creates certificates that are used by agents to exchange public keys without contacting a public key server and avoiding congestion scenarios. An agent conveys its public key information to another by transmitting its certificate. Then, the receiver agent verifies that the certificate was created by the CA.

The main features of our implementation, outlined in Fig. 11, are the following ones:

- Communications are symmetrically encrypted by using the triple DES algorithm to provide confidentiality. The symmetric encryption key is negotiated in every session.
- The negotiation of the symmetric key is performed by using the public key RSA algorithm. This algorithm allows an authenticated communication between the involved agents. Therefore, every agent has a public and a private key.
- The public and private keys are generated from the login and password of the agent.
- Two agents transfer one to each other their public key through their certificates. As explained in Section 5, certificates are delivered to agents by the CA.
- For the sake of increasing security, symmetric key negotiation requires a double exchange of nonces.
- A Message Authentication Code (MAC) is used to compute a digest of the plain message. This digest is encrypted with the private key of the sender agent and appended to the plain message in order to provide authentication and integrity.
- The MAC is computed using a Message Digest (MD5) algorithm [36,24].

Once explained how the communication among agents achieves certain degree of security, we will explain how certificates are distributed.

A certificate for an agent A, say C_A, can be expressed as

$$C_A = [\mathrm{ID}_A, \mathrm{KU}_A, T, E_{\mathrm{KR}_{\mathrm{CA}}}[D(M)]],$$

where ID_A is the identification for the agent A, KU_A its public key, T a time stamp and $E_{\mathrm{KR}_{\mathrm{CA}}}[D(M)]$ is the digest calculated over the three previous components of C_A encrypted with the private key of the CA. This last term provides authentication and integrity when a certificate is requested by an agent, but only if the public key of the CA has been delivered using a secure channel.

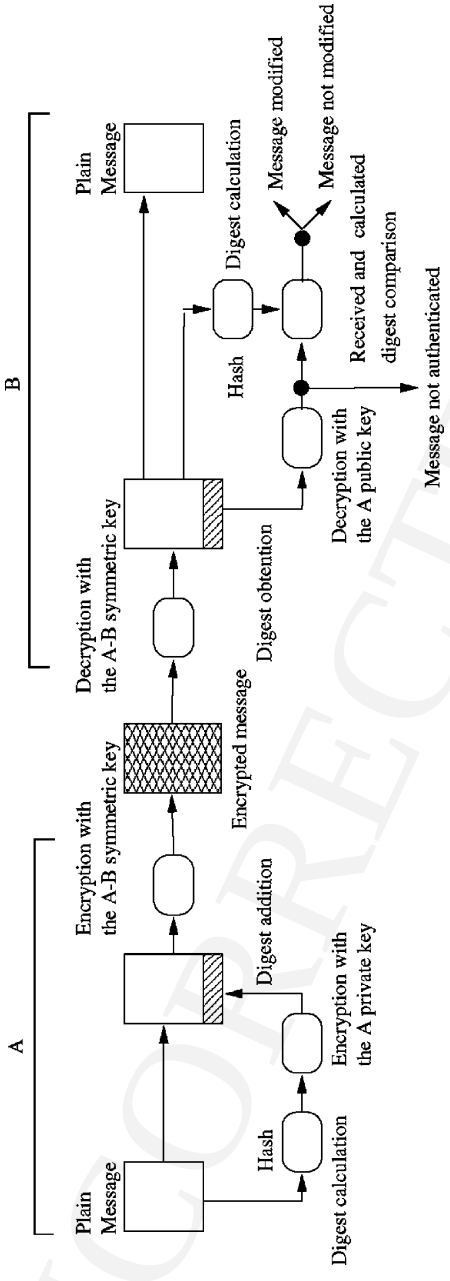

Fig. 11. Block diagram of the secure communication process.

The main objective of this implementation has been to establish great similarities with the usual transport level interfaces, i.e. sockets. The use of our interface differs from others in the following points:

- Additional parameters must be specified; for instance, sender keys and certificates.
- Additional error conditions have to be managed; for instance, authentication errors.

Finally, we describe how an agent should use the secure communication interface. The steps that it must follow are the following ones:

(1) Open a socket as in conventional transport level interfaces.
(2) Set up the socket with some parameters: the agent identification, the agent private key and the public key of the CA.
(3) Execute the handshake protocol. This is the core operation of the secure communication layer. When executing the handshake protocol, certificates of peer agents are exchanged and a session symmetric encryption key is established. Then, two nonces are exchanged following the scheme detailed in [40].
(4) If the handshake protocol is successfully concluded, the socket is ready to accept messages.

7. Medical protocols server agent

Before specifying the services provided by the medical protocols server agent (MPSA), we summarize the system specification of a MP. A MP is a unique identifier, a non-empty set of scenes and an initial scene.

The components of a scene are:

- A unique **identifier**.
- A **scene dependence graph** which specifies a negotiation process between SDAs. States represent points in a negotiation process (SDAs involved in the process and their condition) and transitions between states represent SDA actions. A scene dependence graph contains one initial state and at least one final state. We distinguish two kinds of actions: dialog actions and movement actions. Dialog actions model information exchange between SDAs. Because of the nature of hospital environment, when a SDA is executing a scene of a MP it can be necessary to start a new medical task which will be described by means of a different scene. Therefore, movement actions model arrivals and departures of SDAs to/from scenes.
- A set of **behavior rules** that describes forbidden SDA actions depending on the result of previous actions.
- A non-empty set of **starting agents**. It refers to the set of SDAs that must be present in the system to execute a scene. For instance, to perform a surgical operation at least an anesthesiologist and a surgeon must be present.
- A set of **dynamic joining agents**. It refers to the set of SDAs that could take part in a scene. Their entrance will depend on the monitoring process state. For instance, during a surgical operation, a second surgeon can be requested depending on the evolution of the patient.

The MPSA allows us to define a distributed and scalable MAS in which SDAs can be developed independently of MPs features by:

- Storing the address of every SA. New SAs can be dynamically created depending on the system monitoring load.
- Creating new MP instances. A MP is started by a SDA from its set of starting agents (the starting agents of the initial scene). At this point, the MPSA creates a new instance for the MP and blocks it until the rest of starting agents are present. A MP instance contains the MP specification and the monitoring state. When creating a new MP instance the monitoring state is empty.
- Hanging up MP scene instances. A MP scene can be hanged up by two reasons: a new dynamic joining agent is needed or a SDA involved in the execution of the scene decides to temporally hang it up. In both cases, the MPSA stores the monitoring state provided by the SA and frees it from the monitoring task.
- Providing SDAs with the list of available MPs depending on SDAs hospital roles. This list will be composed of blocked and hanged up MP scene instances, and all MP that they can start.
- Providing SAs with MP scene instances. When a blocked or hanged up MP scene instances is ready (all required SDAs are present) the MPSA selects one SA and delivers to it the instance to be monitored. Moreover, as a SA must intercept through interagents all SDA interactions, the MPSA delivers to interagents the SA address and the MP scene instance identifier. This strategy allows us to distribute the system monitoring load between SAs, to move a SDA between MP scenes and to concurrently execute multiple scenes of a same MP or of different MPs.

8. Medical protocols supervisor agents

The main services offered by medical protocols supervisor agents (SAs) are the supervision and assistance of MP executions. When monitoring MPs, SAs assist SDAs in the sense that they inform SDAs about all the possible and forbidden actions specified. Although SDAs behavior is not restricted by SAs, SAs record SDAs anomalous decisions. In some cases, a MP can specify a timeout for executing critical medical actions; for instance, a biopsy has to be performed before 2 days. Thus, SAs control the elapsed time between SDAs actions.

The most important benefit of defining specialized agents for monitoring MPs is that SDAs do not need to be synchronized. To check forbidden and critical SDAs actions, it is necessary to know the monitoring state, which must be the same for every SDA involved in the MP execution. This information is centralized and maintained by SAs in our approach.

A more detailed description of SA functionalities is the following one:

- As we have seen, the number of active SAs is not fixed in the system. Therefore, a SA must be able to supervise the execution of several MP instances. To this end, interagents transparently forward to SAs, using the MP scene instance identifier, interactions between SDAs.

- For each MP instance, a SA must compute the monitoring state, i.e. for each active scene it must compute the current state by means of the scene dependence graph, the sequence of SDA actions from the initial state to the current state and, if necessary, behavior rules of the current state. To check behavior rules, we have defined a *general data type* that encapsulates data and their operations. The content of a message must belong to a data type that derives from this general data type; for instance, a biopsy result should be provided together with its testing operations. This feature is easily achieved in the object oriented programming paradigm of Java.
- After computing a monitoring state, SAs assist SDAs by informing them, through interagents, about possible and forbidden actions.
- Finally, SAs must manage arrival and departure actions. On the one hand, when a new dynamic joining agent is required (arrival action), SAs hang up the MP scene instance following the process described in Section 7. On the other hand, a SDA can abandon a MP scene execution in different ways:
  (1) A SDA decides to temporally hang up a MP scene execution. The SDA contacts with the MPSA who checks if the MP scene instance is already hanged up. In this case, the SDA is simply added to the list of dynamic joining agents. Otherwise, the MPSA interacts with the SA for obtaining the MP scene instance.
  (2) When executing a MP scene, a SDA reaches a final state. The SA informs the interagent about the exit condition.
  (3) A SDA can, temporally or definitively, leave the current MP scene to execute a new one. The SA provides the interagent with the identifier of the new scene. The interagent interacts with the MPSA for obtaining the list of blocked and hanged up instances of the MP scene. Finally, the SDA decides to execute one of them or a new one.

In the next section we show how these system interaction processes are transparent to SDAs. This allows SDAs to focus on the decision control based on the specific domain knowledge.

9. Interagents

Two functional requirements of our system are that the monitoring process must be transparent to SDAs and SDAs must be able to concurrently execute multiple MPs. Interagents allow us to achieve such objectives by forwarding SDAs interactions to SAs and assigning a different interagent to each MP scene concurrent execution. In order to execute a MP scene, a SDA interacts with other SDAs through one interagent; this interagent is the same for all interactions in the MP scene.

When a SDA is executing a MP scene it can be necessary, temporally or definitively, to interrupt the current scene and then execute a new MP scene. In Section 7 and 8 we have described how the MPSA and SAs interact with interagents to perform entrances to MPs scenes. Now, we describe how interagents perform exits from MPs scenes. As we have seen, the system uniquely identifies every MP scene execution. When a SDA enters to a MP scene execution, the interagent pushes the scene instance identifier and the SA address

(provided by the MPSA) onto a stack. When a SDA reaches a final state (the SA informs the interagent about the exit condition), the interagent pops the current MP scene instance information from the stack. After performing a pop operation, if the stack is not empty (i.e., it contains temporally interrupted executions), the interagent contacts with the MPSA for coming back to the last hanged up instance which is located at the top of the stack.

An additional feature of interagents is to provide human SDAs with a user-friendly interface accessible using a web browser. This interface represents the content of SDAs messages in a suitable way and assists human agents in making decisions by showing the possible and forbidden actions in every step when executing a MP scene. The operations encapsulated by the *general data type* include not only operations to check behavior rules, but also operations to graphically represent the involved message information. For instance, biopsy and X-rays results should be shown to the user in a different way.

10. Database agent

According to [13] the integration of the existing medical records within the guidelines specified by means of a MP is a key issue for automatically executing and monitoring MPs. The effective integration requires consistent information structures and content. However, in real hospital environments we identify two key problems. On the one hand, the information structures and content of the medical records are not uniform among different hospital environments. On the other hand, typically, the MPs specifications involve high level information structures and content while medical records management systems store and retrieve information at a lower level. Therefore, to achieve an effective integration it is necessary to provide a *bridge* between the execution and monitoring system and the medical records management system.

In our framework, the bridge is provided as an autonomous SDA called *database broker agent*. The functionalities of the database broker agent are the following ones:

- Provides to the monitoring system a unified frame for exchanging information between the system agents and the database broker agent. To be precise, the database broker agent answers queries by using the already proposed *general data type* that encapsulates data and their operations.
- Provides a uniform high level language for querying the medical records. For instance, to obtain the results of biopsies and X-rays.
- Translates the high level queries into the corresponding set of low level queries for the medical records management system. For instance, the results of biopsies and X-rays could be composed of different information elements as patient identification data, images, indexes results, ... These elements can be stored with different data models: object oriented, relational, hierarchical, ... And within a model, the information elements can be stored with different management systems and information structures. Nowadays, we have developed this functionality for management systems with relational data models and with the SQL interrogation language. To interact with different management systems, the SDA should be extended with new modules that implement low level access to these systems.

11. Conclusions and future work

In this paper, we have first defined a formal specification language for modeling dialogical institutions which, in particular, can be used to model hospital environments and their MPs. Second, we have defined a graphical framework (called JAFDIS) developed in Java for specifying dialogical institutions according to the specification language. Third, we have defined a MAS architecture for the assistance and supervision of the execution of MPs in hospital environments. The architecture is based on a knowledge distributed approach that allows us to distribute the load among agents and to define a dynamically scalable system. SDAs model specific medical services and manage the involved domain knowledge, and ASAs manage MP specifications and supervise their execution. When executing a MP, the system assists SDAs by reporting to them about possible and forbidden actions and SDAs perform the decision control. For supervising executions, interactions between SDAs are transparently forwarded by interagents to SAs allowing SDAs to concurrently execute multiple MPs. Messages are automatically digitally signed and encrypted by means of the services provided by the secure communication interface. All system components have been implemented in Java and this allows us to define messages as objects that encapsulate data and processing operations. Finally, we have defined a database broker agent for automatically bridging medical records management systems and the monitoring system.

For the time being, we have developed a prototype of the MAS architecture which has been implemented in Java. This prototype consists of the database broker agent, the secure communication interface and all required ASAs, i.e. a certification agent, a MP server agent and MP supervisor agents, and interagents.

The prototype of the MAS architecture has been developed using a Java based representation of MPs. That is, each MP is represented by a Java class hierarchy where each class represents a scene of the MP. This representation model is automatically generated by the graphical specification tool JAFDIS. The Java based representation of MPs is a general representation model in the sense that one can easily define a translator from other MPs representation models (for instance, GEM [39], GLIF [26,28] or UMLS [1]) to our Java class hierarchy model.

Our ongoing research is directed toward two main directions. On the one hand, the development of a user-friendly interface tool between the medical records administrator and the database broker agent. The objective of this tool is to allow the medical records administrator to dynamically maintain the available set of high level queries. On the other hand, the design and development of medical services as autonomous intelligent software agents. For instance, it would be interesting to provide a suitable implementation of an agent for automatically scheduling the resources of the hospital taking into account different quality criteria.

Acknowledgements

We would like to thank the anonymous referees for their comments and suggestions. This research was partially supported by project SMASH (TIC96-1038-C04-03) and

project LOGFAC (TIC2001-1577-C03-03) funded by the Spanish *Ministerio de Ciencia y Tecnología*.

References

[1] Achour SL, Dojat M, Rieux C, Bierling P, Lepage E. UMLS-based knowledge acquisition tool for rule-based clinical decision-support system development. J Am Med Inform Assoc 2001;8:351–60.
[2] Alsinet, T, Béjar, R, Ansótegui, C, Fernàndez, C, Manyà F. JAFDIS, a Java framework for dialogical institution specification. Technical Report DIEI-98-RT-2, Universitat de Lleida, 1998. Available at http://fermat.eup.udl.es/∼cesar/recerca/jafdis_ev.ps.gz.
[3] Alsinet T, Béjar, R, Fernández, C, Manyà, F. A multi-agent system architecture for monitoring medical protocols. In: Sierra, C, Gini, M, Rosenschein, JS, editors. Proceedings of the Fourth International Conference on Autonomous Agents, Barcelona, Spain. New York, USA: ACM Press; 2000. p. 499–505.
[4] Boon W, Nardi R, Bertelli S. Computerized guidelines for preventative care in general practice. Stud Health Technol Inform 1995;16:45–56.
[5] Boxwala, AA, Greenes, RA, Deibel, SR. Architecture for a multipurpose guideline execution engine. In: Lorenzi, NM, editor. Proceedings of AMIA Annual Symposium, Greenes, PA, USA. Philadelphia, USA: Hanley & Belfus; 1999. p. 701–5.
[6] Danthine AAS. Protocol representation with finite-state models. IEEE Trans Commun 1980;COM-20:632–43.
[7] Esteva, M, Rodríguez-Aguilar, JA, Arcos, JLL, Sierra, C, Garcia, P. Institutionalising open multi-agent systems. In: Kraus, S, Nakashima, H, Tambe, M, editors. Proceedings of the Fourth International Conference on Multiagent Systems, Boston, MA, USA. Los Alamitos, CA, USA: IEEE Press; 2000. p. 381–2.
[8] Faratin P, Sierra C, Jennings NR. Negotiation decision functions for autonomous agents. Int J Robot Autonom Agents 1998;24(3/4):159–82.
[9] Fox, J, Das, S. Safe, Sound: artificial intelligence in hazardous situations. Menlo Park, CA, USA: AAAI Press; 2000.
[10] Fox J, Johns N, Rahmanzadeh A. Disseminating medical knowledge: the PROforma approach. Artif Intel Med 1998;14:157–81.
[11] Freier, AO, Karlton, P, Kocher, PC. The SSL Protocol, Version 3.0, 1996.
[12] Georgeff, MP, Lansky, AL. Reactive reasoning and planning. In: Forbus, K, Shrobe, H, editors. Proceedings of the Sixth National Conference on Artificial Intelligence, Seattle, WA, USA. Menlo Park, CA, USA: AAAI Press; 1987. p. 677–82.
[13] Glowinski A. Integrating guidelines and the clinical record: the role of semantically constrained terminologies. Stud Health Technol Inform 1995;16:207–18.
[14] Gordon, C. Practice guidelines and healthcare telematics: towards an alliance. In: Gordon, C, Christensen, JP, editors. Health telematics for clinical guidelines and protocols. Amsterdam, The Netherlands: IOS Press; 1995. p. 3–15.
[15] Herbert, SI. Informatics for care protocols and guidelines: towards a european knowledge model. In: Gordon, C, Christensen, JP, editors. Health telematics for clinical guidelines and protocols. Amsterdam, The Netherlands: IOS Press; 1995. p. 27–42.
[16] Huang, J, Jennings, NR, Fox, J. An agent architecture for distributed medical care. In: Wooldridge, M, Jennings, NR, editors. Proceedings of the ECAI'94 Workshop on Gent Theories, Architectures and Languages, Amsterdam, The Netherlands, LNCS 890. Heidelberg, Germany: Springer; 1994. p. 219–232.
[17] Jennings, NR, Faratin, P, Johnson, MJ, O'Brien, PO, Wiegand, ME. Using intelligent agents to manage business processes. In: Proceedings of the First International Conference on The Practical Application of Intelligent Agents and Multi-Agent Technology, London, UK, 1996, p. 345–60.
[18] Jennings NR, Faratin P, Lomuscio AR, Parsons S, Wooldridge M, Sierra C. Automated negotiation: prospects, methods and challenges. Group Decision Negotiation 2001;10:199–215.
[19] Jennings NR, Norman TJ, Faratin P. ADEPT an agent-based approach to business process management. ACM SIGMOD Record 1998;27(4)(4):32–9.

[20] N.R. Jennings, S. Parsons, C. Sierra, P. Faratin, Automated negotiation. In: Proceedings of The Fifth International Conference on The Practical Application of Intelligent Agents and Multi-Agent Technology, Manchester, UK, 2000, p. 23–30.

[21] Johnson, PD, Tu, S, Booth, N, Sugden, B, Purves, IN. Using scenarios in chronic disease management guidelines for primary care. In: Overhage, JM, editor. Proceedings of AMIA Annual Symposium, Los Angeles, CA, USA. Philadelphia, USA: Hanley & Belfus; 2000. p. 389–93.

[22] Kohnfelder, L. Towards a practical public-key cryptosystem. Bachelor's Thesis, MIT, 1978.

[23] Maes P. Agents that reduce work and information overload. Commun ACM 1996;37(7):31–40.

[24] Menezes, A, Oorschot, P, Vanstone, S. Handbook of applied cryptography. London, UK: CRC Press; 1997.

[25] Nwana, HS, Rosenschein, J, Sandholm, T, Sierra, C, Maes, P, Guttmann, R. Agent-mediated electronic commerce: issues, challenges and some viewpoints. In: Sycara, KP, Wooldridge, M, editors. Proceedings of the Second International Conference on Autonomous Agents, Minneapolis, USA. New York, USA: ACM Press; 1998. p. 189–96.

[26] Ohno-Machado L, Gennari JH, Murphy SN, Jain NL, Tu SW, Oliver DE, Pattison-Gordon E, Greenes RA, Shortliffe EH, Barnett GO. The guideline interchange format: a model for representing guidelines. J Am Med Inform Assoc 1998;5:357–72.

[27] Parsons SD, Sierra C, Jennings NR. Agents that reason and negotiate by arguing. J Logic Comput 1998;8(3):261–92.

[28] Peleg, M, Boxwala, AA, Ogunyemi, O, Zeng, Q, Tu, SW, Bernstam, E, Ohno-Machado, L, Shortliffe, EH, Greenes, RA. GLIF3: the evolution of a guideline representation format. In: Overhage, JM, editor. Proceedings of AMIA Annual Symposium, Los Angeles, CA, USA. Philadelphia, USA: Hanley & Belfus; 2000. p. 645–9.

[29] Plaza, E, Noriega, P, Sierra C. The conference center as an agent-mediated institution. In: Proceedings of the First International Workshop on Agents in Community Ware, Paris, France, 1998, p. 73–82.

[30] Purves, IN, Sugden, B, Booth, N, Sowerby, M. The prodigy project—the interactive development of the release one model. In: N.M. Lorenzi, editor. Proceedings of AMIA Annual Symposium, Greenes, PA, USA. Philadelphia, USA: Hanley & Belfus; 1999. p. 359–63.

[31] Rodríguez, JA, Noriega, P, Sierra, C, Padget, J. A Java-based electronic auction house. In: Proceedings of the Second International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology, 1997, p. 207–224.

[32] Rodriguez-Aguilar, JA, Martin, FF, Garcia, P, Noriega, P, Sierra, C. Towards a formal specification of complex social structures in multi-agent systems. In: Padget, JA, editor. Collaboration between human and artificial societies: coordination and agent-based distributed computing, LNAI 1624. Heidelberg, Germany: Springer; 1999. p. 284–300.

[33] Rosenschein S. Formal theories of knowledge in AI and robotics. New Generation Comput 1985; 3(4):345–57.

[34] Rosenschein, S, Kaelbling, L. The synthesis of digital machines with provable epistemic properties. In: Halpern, JY, editor. Proceedings of the Conference on Theoretical Aspects of Reasoning About Knowledge. New York, USA: Morgan Kaufmann; 1986. p. 207–224.

[35] Sabater, J, Sierra, C. Regret: reputation in gregarious societies. In: Andre, E, Sen, S, Frasson, C, Müller, JP, editors. Proceedings of The Fifth International Conference on Autonomous Agents, Montreal, Canada. New York, USA: ACM Press; 2001. p. 194–5.

[36] Schneier, B. Applied cryptography: protocols, algorithms, and source code in C. 2nd ed. Hoboken, NJ, USA: Wiley; 1995.

[37] Shahar, Y, Miksch, S, Johnson, P. The asgaard project: a task-specific framework for the application and critiquing of time-oriented clinical guidelines. Artif Intel Med 1998;14:29-51.

[38] Shehory, O, Sycara, K, Sukthankar, G, Mukherjee, V. Agent aided aircraft maintenance. In: Etzioni, O, Müller, JP, editors. Proceedings of the Third International Conference on Autonomous Agents, Seattle, WA, USA. New York, USA: ACM Press; 1999.

[39] Shiffman RN, Karras BT, Agrawal A, Chen R, Marenco L, Nath S. GEM: a proposal for a more comprehensive guideline document model using xml. J Am Med Inform Assoc 2000;7:488–98.

[40] Stallings, W. Network and Internetwork Security. New Jersey, USA: Prentice-Hall; 1995.

[41] Sycara K, Decker K, Pannu A, Williamson M, Zeng D. Distributed intelligent agents. IEEE Expert 1996;11(6):36–45.
[42] Terenziani P, Molino G, Torchio M. A modular approach for representing and executing clinical guidelines. Artif Intel Med 2001;23:249–76.
[43] Wong, HC, Sycara, K. Adding security and trust to multi-agent systems. In: Proceedings of the Agents'99 Workshop on Deception, Fraud and Trust in Agent Societies, Seattle, WA, USA, 1999, p. 149–61.
[44] Wooldridge, M, Jennings, NR, Agent theories, architectures, and languages: a survey. In: Wooldridge, M, Jennings, NR, editors. In: Proceedings of the ECAI'94 Workshop on Agent Theories, Architectures and Languages, Amsterdam, The Netherlands, LNCS 890. Heidelberg, Germany: Springer; 1994. p. 1–32.
[45] Wooldridge MJ, Jennings NR. Cooperative problem solving. J Logic Comput 1999;9(4):563–92.