META-GLARE's Supports to Agent Coordination

META-GLARE's supports to agent coordination

Luca Anselma[1], Alessio Bottrighi[2], Luca Piovesan[2] and Paolo Terenziani[2]

[1]Dipartimento di Informatica, Università di Torino, Via Pessinetto 12, Torino, Italy
anselma@di.unito.it
[2]DISIT, Università del Piemonte Orientale, Viale Teresa Michel 11, Alessandria, Italy
{alessio.bottrighi, luca.piovesan, paolo.terenziani}@uniupo.it

Abstract. Clinical Guidelines (GLs) provide evidence-based recommendations to suggest to physicians the "best" medical treatments, and are widely used to enhance the quality of patient care, and to optimize it. In many cases, the treatment of patients cannot be provided by a unique healthcare agent, operating in a unique context. For instance, the treatment of chronic patients is usually performed not only in the hospital, but also at home and\or in the general practitioner's ambulatory, and many healthcare agents (e.g., different specialist, nurses, family doctor) may be involved. To grant the quality of the treatments, all such agents must cooperate and interact. A computer-based support to GL execution is important to provide facilities for coordinating such different agents, and for granting that, at each time, the actions to be executed have a "proper" person in charge and executor, and are executed in the correct context. Additionally, also facilities to support the delegation of responsibility should also be considered. In this paper we extend META-GLARE, a computerized GL management system, to support such needs providing facilities for (1) treatment continuity (2) action contextualization, (3) responsibility assignment and delegation (4) check of agent "appropriateness". Specific attention is also devoted to the temporal dimension, to grant that each action is executed according to the temporal constraints possibly present in the GL. We illustrate our approach by means of a practical case study.

Keywords: Computer-Interpretable Guidelines, Agent Coordination, Temporal reasoning.

1 Introduction

The rationalization of the healthcare system is a task of fundamental importance in order to grant both the quality and the standardization of healthcare services, and the minimization of costs. *Evidence-based* medicine and clinical guidelines (GLs) have gained a major role in this context. Clinical guidelines (GLs) are "*systematically developed statements to assist practitioner and patient decisions about appropriate healthcare under specific clinical circumstances*" [1]. In short, GLs provide evidence-based recommendations to suggest to physician the set of diagnostic/therapeutic actions to be executed to cope with a specific disease, supporting their decision-making activity. Thousands of GLs have been devised in the last years. For instance, the Guideline

International Network (http://www.g-i-n.net) groups 103 organizations representing 47 countries from all continents, and provides a library of more than 6400 CPGs. Despite such a huge effort, CPGs have not provided all the expected advantages to clinical practice yet. Recent research has shown that Computer Science can help to drastically improve the impact of GLs in the clinical practice. The adoption of ***computer-interpretable clinical guidelines*** (**CIG**s henceforth) and computerized approaches to acquire, represent, execute and reason with CIGs can further increase the advantages of CPGs, providing crucial advantages to:

(1) patients, enabling them to receive the best quality medical treatments;
(2) physicians, providing them with a standard reference which they may consult, as well as advanced support for their decision-making and\or coordination activities;
(3) hospitals, and healthcare centers, providing them with tools to enable the quality and the standardization of their services.

Many different systems and projects have been developed to this purpose (see e.g. [2–4]). Such systems usually provide facilities to acquire, represent and/or execute CIGs, and are mainly developed to support physicians in patient care.

Different forms of support may be provided. In particular, a lot of attention has been devoted to decision support facilities (such as "what if" analysis [5] or cost-benefit analysis [6]). However, CIG systems have quite neglected the problem of properly supporting the coordination of different healthcare agents in the execution of CIGs (see, however, the discussion in Section 7). In most cases, CIG systems assume that the CIG is executed in a unique location (usually, in a hospital) under the responsibility of a unique agent (usually, a physician). However, such a simplified assumption does not hold for many different GLs. For example, GLs dealing with chronic disorders require that the patient treatment is continued in time, and is carried on in *different contexts* (e.g. at home, or in the general practitioner's ambulatory), under the responsibility of *different agents* (not only physicians, but also nurses, family doctors, family members). In such cases, it is crucial that the different healthcare agents communicate and coordinate themselves in an effective way, so that, at any moment, there is a "proper" person in charge (henceforth, "responsible person" or "responsible" for short) for the GL actions to be executed, operating in the right context, and having the appropriate role and qualification. Additionally, the responsible might have the possibility of transferring the responsibility to other agents, or to delegate the responsibility or the execution of actions. None of the available computerized GL systems fully addresses such needs. Notably, there are several multi-agent approaches for healthcare in the literature (see e.g. the survey in [7]), but they consider agents as autonomous software entities. In our approach, we consider agents as a representation of real persons and use a *multi-agents view* to describe and support a distributed GL execution.

In this paper, we describe how we have extended META-GLARE, [8], a recent evolution of GLARE (Guideline Acquisition, Representation and Execution), a domain-independent CIG system for GL acquisition and execution [9], to provide such support to healthcare agents. However, although we have implemented our approach in META-GLARE**,** it is worth stressing that the methodology we propose is general and application-independent.

First, we identify the extensions to the CIG formalism needed to represent the different pieces of information required to manage the above phenomena. In particular, our representation formalism supports the specification, for each action, of the *context* in which it must be executed, and of the *qualification* and *competences* required to its *responsible*, to its *delegate* (if any) and to its *executor*.

Then, we describe the facilities we have provided to support the coordination of the different agents involved in the execution of a CIG on a patient. The main goal of such facilities is to grant that a proper treatment is assured to the patient. While "standard" CIG systems support physicians by suggesting the proper action to be executed at a given time on the patient, our approach further enhances such a support to grant that the actions are executed in the proper context, under the responsibility of a proper (i.e., having the correct qualification and competences) agent, and are executed by a proper agent. To achieve such a goal, our facilities support:

(1) treatment continuity,
(2) action contextualization,
(3) responsibility assignment and delegation,
(4) check of agent and executor "appropriateness".

Last, but not least, our approach is the first one in the literature that explicitly manages the ***temporal dimension*** within the coordination process. Indeed, actions are delegated and executed at specific times, so that, to implement a realistic and useful coordination process, systems have to explicitly manage the "window" of time in which actions have to be executed (given the temporal constraints in the CIG, and given the execution times of the previous actions in the CIG). For instance, before accepting a delegation\execution, it is important (and realistic) that the agents have also the information about when they will have to manage such an action. Without such an information, their acceptance is just a "generic" one (and not a strong commitment to manage the action) since they cannot grant to be available at the time when the action will have to be executed. Therefore, our approach also proposes temporal facilities to

(5) associate with each action (during the coordination process) the window of time when it has to be executed, and
(6) check that the actions are executed at "proper" times (i.e., at times consistent with the temporal constraints in the CIG).

Notably, the facilities (5) and (6) constitute a major advance with respect to the other approaches in the literature, since they require not only the representation of the temporal constraints in the CIG, but also the development of temporal reasoning algorithms, and their integration into the delegation\execution process. In particular, temporal reasoning is necessary in order to associate with each GL action a "window" of time in which it has to be executed. Notably, such a "window" cannot be evaluated once-and-for-all, since the execution of other actions may further "restrict" the window of execution of a given one. In our approach, we suggest that such a temporal window has to be evaluated (i) whenever an agent is looking for a responsible\delegate\executor of an action, to check whether the new agent accepts the task in the specified window of time, and (ii) whenever an action has to be effectively executed, to grant that it is executed in a time consistent with the temporal constraints in the CIG and with the times of execution of the previously executed actions in the CIG.

As a running example, in this paper we show the application of our approach to the "management of harmful drinking and alcohol dependence in primary care" GL developed by the Scottish Intercollegiate Guidelines Network (SIGN) [10], which we have adapted to the Italian context.

The paper is structured as follows: in Section 2 we describe the main features of GLARE and META-GLARE. In Section 3 we describe our extensions to the META-GLARE representation formalism. In Section 4, we specifically discuss the temporal reasoning support. In Section 5 we describe the different facilities we provide to support the distributed execution of a CIG, and the coordination of the involved agents, with specific emphasis on the temporal issues. In Section 6 we exemplify a practical application of our approach considering the treatment of alcohol-related disorders. Finally, in Section 7 we address related work and concluding remarks.

2 GLARE and META-GLARE

META-GLARE is an evolution of GLARE, a domain-independent system for acquisition and execution of GLs [9], which we have been developing since 1997, in collaboration with the physicians of Azienda Ospedaliera San Giovanni Battista in Torino, Italy.

The main goals of the GLARE systems are:

(i) to be domain-independent. In particular, GLARE has been already applied to deal with a wide range of CIGs ranging from asthma to ischemic stroke;

(ii) to be user-friendly. GLARE is intended to be a tool to support physicians (in particular, regarding decision making), and not to replace them. Such a goal has several implications, including the facts that (i) CIGs representation language must be as close as possible to the physicians' way of looking at it, and (ii) the "technological" complexity of the system must be hidden to the user-physicians, through a user-friendly and easy-to-use interface;

(iii) to be easily extendable. Indeed, physicians should be provided with plenty of facilities concerning CIGs. Thus, the system architecture must be modular and easily extendable.

2.1 GLARE's kernel

The core of GLARE (see the box on the left of Fig. 1) is based on a modular architecture. CG_KRM (Clinical Guidelines Knowledge Representation Manager) is the main module of the system: it manages the internal representation of CIG, and operates as a domain-independent and task-independent knowledge server for the other modules; moreover, it permanently stores the acquired CIG in a dedicated Clinical Guidelines Database (CG-DB).

The Clinical Guidelines Acquisition Manager (CG_AM) provides expert-physicians with a user-friendly graphical interface to introduce the CIG into the CG_KRM and to describe them. It may interact with four databases: the Pharmacological DB, storing a structured list of drugs and their costs; the Resources DB, listing the resources that are available in a given hospital; the ICD DB, containing an international coding system of diseases; the Clinical DB, providing a "standard" terminology to be used

when building a new CIG, and storing the descriptions and the set of possible values of clinical findings.

GLARE's acquisition module provides expert-physicians with a user-friendly and easy-to-use tool for acquiring a CIG. In order to achieve these goals, GLARE provides: (i) a graphical interface, which supports primitives for drawing the control information within the CIGs, and ad-hoc windows to acquire the internal properties of the objects; (ii) facilities for browsing the CIGs; (iii) an "intelligent" help and consistency-checking facilities including name and range checking, logical design criteria checks.

The execution module (CG-EM) executes a CIG for a specific patient, considering the patient's data (retrieved from the Patient DB). It adopts the "agenda technique" (see [11]). Basically, through the "agenda technique", GLARE is able to identify all the next actions to be executed in the current CIG, and the window of time when they have to be executed.

Notably, the schema of the Patient DB mirrors the schema of the Clinical DB. Therefore, the interaction with the Clinical DB during the acquisition phase makes it possible to automatically retrieve data from the Patient DB at execution time. CG-EM stores the execution status in another DB (CG Instances) and interacts with the user-physician via a graphical interface (CG-IM).

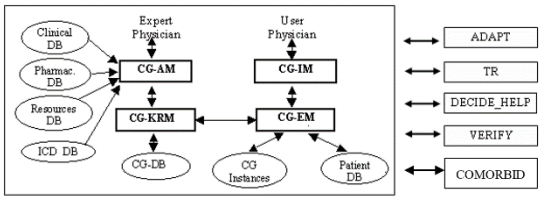

Fig. 1. Architecture of GLARE. Rectangles represent computation modules, and ovals data/knowledge bases (original figure in [12]).

2.2 GLARE's Extended Architecture

GLARE's architecture is open: new modules and functionalities can be easily added if\when necessary. In the last years, several new modules and\or methodologies have been added to cope with different phenomena.

The **ADAPT** module [13] copes with automatic resource-based contextualization. Indeed, contextualization is an essential step to be taken before a guideline manager is actually adopted in clinical practice. One of the most relevant obstacles to the exploitation and dissemination of GLs is the gap between the generality of the GLs themselves (as defined, e.g., by physicians' committees) and the peculiarities of the specific contexts of application. ADAPT supports the adaptation of general guidelines to the local setting, by (automatically) taking into account local resources (e.g. diagnostic instrumentation) unavailability, and locally applied procedures, which may require to discard some alternatives.

The temporal reasoning module **TR** [11] deals with the temporal constraints in the CIGs. As already discussed in the introductory section, the temporal dimension is essential in the management of CIGs. A detailed description of this module, and how it is exploited in the coordination process, is presented in Sections 4 and 5 of this paper.

The decision making support module **DECIDE_HELP** [14] provides further support to physicians for their decision-making activities, focusing on therapy selection. In many cases, when executing a CIG on a given patient, a physician can be faced with a choice among different therapeutic alternatives, and identifying the most suitable one is often not straightforward. In several situations no alternative is actually "better" than the others from a strictly clinical viewpoint, and CIGs are only meant to present all the range of choices, leaving to the user the responsibility of selecting the "right" one.

In clinical practice, various selection parameters (such as the costs and effectiveness of the different procedures) are sometimes available when executing a CIG, but the task of comparing and balancing them is typically left to the physician.

Decision theory seems a natural candidate as a methodology for affording this analysis; to this end, we have realized a mapping between the CIG representation primitives and decision-theory concepts (in particular we represent CIGs as Markov Decision Processes), and we have developed a decision-theory tool for supporting therapy selection in GLARE. In particular, the GLARE decision-theory facility enables the user to: (1) identify the optimal policy, and (2) calculate the expected utility along a path, by exploiting classical dynamic programming algorithms.

The module **VERIFY** [15] supports *model-based verification* of CIGs. Indeed, CIGs are very large and complex bodies of knowledge, so that, after they have been acquired, it is important to be able to check automatically whether they satisfy different types of properties (e.g., their eligibility to treat specific classes of patients). The general methodology we propose is to exploit the capabilities of a model checker (specifically, we adopt SPIN [16]) by loosely integrating with it a CIG system. Such a loose integration is achieved by defining a module for the automatic translation of any GLARE CIG into the corresponding CIG represented in the model-checker format. The translation of a CIG can be performed a priori and once and for all. After that, the model-checker can be used in a standard way in order to verify any property (that can be expressed in the model-checker property language) on any of the translated CIGs. It is important to stress that the properties need not to be defined a-priori: the user can directly express a new property and ask the model-checker to verify it.

Finally, the **COMORBID** module [17, 18] is devoted to the treatment of comorbid patients, i.e., of patients having more than one disease.

Computer Interpretable Guidelines (CIGs) are consolidated tools to support physicians with evidence-based recommendations in the treatment of patients affected by a specific disease. However, the application of two or more CIGs on comorbid patients is critical, since dangerous interactions between (the effects of) actions from different CIGs may arise. **COMORBID** is the *first* tool supporting, in an integrated way, (i) the knowledge-based detection of interactions, (ii) the management of the interactions, and (iii) the final "merge" of (part of) the CIGs operating on the patient. **COMORBID** is characterized by being very supportive to physicians, providing them support for focusing, interaction detection, and for an "hypothesize and test" approach to manage the detected interactions. To achieve such goals, it provides advanced Artificial Intelligence techniques.

Testing. GLARE has been already tested considering different domains, including bladder cancer, reflux esophagitis, heart failure, and ischemic stroke. The acquisition

of a GL using GLARE is reasonably fast (e.g., the acquisition of the GL on heart failure required 3 days).

2.3 GLARE REPRESENTATION FORMALISM

In the GLARE project, a GL is represented through the set of actions composing it. GLARE distinguishes between *atomic* and *composite* actions. Atomic actions can be regarded as elementary steps in a CIG, in the sense that they do not need a further decomposition into sub-actions to be executed. Composite actions are composed by other actions (atomic or composite). Four different types of atomic actions can be distinguished in GLARE: *work* actions, *query* actions, *decisions* and *conclusions*. Work actions are basic atomic actions which must be executed on the patient, and can be described in terms of a set of attributes, such as name, (textual) description, cost, time, resources, goals. Query actions are requests of information, which can be obtained from the outside world (physicians, databases, knowledge bases). Decision actions are specific types of actions embodying the criteria which can be used to select alternative paths in a CIG. In particular, diagnostic decisions are represented as an open set of triples <diagnosis, parameter, score> (where, in turn, a parameter is a triple <data, attribute, value>), plus a threshold to be compared with the different diagnoses' scores. On the other hand, therapeutic decisions are based on a pre-defined set of parameters: *effectiveness*, *cost*, *side-effects*, *compliance*, *duration*. Finally, conclusions represent the output of a decision process. Composite actions are defined in terms of their components, via the "has-part" relation. Control relations establish which actions might be executed next and in what order. We distinguish among four different control relations: *sequence*, *constrained*, *alternative* and *repetition*. The description of sequences usually involves the definition of the minimum and maximum delay between actions. Complex temporal constraints between actions (e.g., *overlaps*, *during*) can be specified using constrained control relations. In particular, action *parallelism* is also supported through this feature.

2.4 META-GLARE

In the last years, a new CIG system, META-GLARE, has been designed, on top of GLARE. Basically, METAGLARE takes in input a CIG formalism (based on the Task-Network Model approach), and provides as output a CIG system for acquiring and executing CIGs expressed through the input language. The core idea of our meta-approach is:

(i) To define an open library of elementary components (e.g., textual attribute, Boolean condition, Score-based decision), each of which is equipped with methods for acquiring, consulting and executing them,

(ii) To provide system-designers with an easy way of aggregating such components to define node and arc types (constituting the representation formalism of a new system),

(iii) To devise general and basic tools for the acquisition, consultation and execution of CIGs, represented by hierarchical directed graphs which are parametric over

the node and arc types (in the sense that the definitions of node and arc types are an input for such tools).

In such a way, META-GLARE provides users with several advantages:

- Using META-GLARE, users can easily define their own systems, basically by defining the nodes and arcs types as an aggregation of components (called *attributes*) from the library. No other effort (e.g., building acquisition or execution modules) is needed.
- The extension of a system can be easily achieved by adding new node/arc types, or adding components to already existing types (with no programming effort at all).
- User programming is needed only in case a new component (*attribute type*) has to be added to the component library. However, this addition is modular and minimal: the programmer has just to focus on the component to be added, and to provide the code for acquiring, consulting, and (if needed) executing it (while the "general" acquisition, consultation and execution engines have not to be modified).

In short, META-GLARE is a "meta" system, in that it takes in input a CIG representation formalism, and automatically generates a CIG system to acquire and execute CIGs expressed in the input formalism. To test it, an extended formalism has been used (see [8]). In the following, we refer to such an extended formalism as "META-GLARE formalism". In particular, the META-GLARE formalism extends GLARE's one with the possibility of specifying not only 1:1 arcs (i.e., arcs with just one input action and one output action), but also 1:n, n:1 and n:n arcs. Such an additional feature is very useful to easily model the parallelism between actions.

GLARE and META-GLARE have been developed in Java, to take advantage of its portability. As a consequence, GLARE can run similarly on any hardware/operating-system platform.

3 CIG annotations

Our approach aims at supporting the coordination of multiple healthcare agents in the execution of a CIG on a given patient, granting at the same time that each action is managed by "proper" agents. To achieve such a goal, each action has to specify a set of properties (that we call annotations), to specify what are the characteristics of a "proper" execution in terms of (i) what are the "proper" (in terms of qualification and competences) agents that can manage the action (either the responsible, or the executor), (ii) what are the "proper" execution contexts, and (iii) when applicable, also additional constraints to cope with the continuity of treatments (in terms of the agents managing them). In META-GLARE, such properties are managed as additional attributes to be added to the description of actions.

In the following, we describe such additional attributes in detail.

First of all, META-GLARE actions can be annotated with the possible *contexts* in which they can be executed.

- **Context** annotation**:** it specifies where the action can be executed (e.g. in-patient care, community medicine). Notably, a context is not necessarily a physical

place, but it might also indicate an operative environment. For example, community medicine can refer to the patient's home or to the general practitioner's ambulatory. In META-GLARE, each action can be annotated with a set of contexts. In such a case, we intend that the action can be executed in any one of the contexts in the set;

Regarding the agents that have to take care of a given action (or set of actions) in a CIG, META-GLARE supports the distinction among three different ***roles***: ***responsible***, ***delegate***, and ***executor***.

The ***responsible*** of an action (or of a whole part of a CIG; for instance, the head physician of a hospital department) is, indeed, the agent that retains the whole responsibility of the management of the action, and who also has the responsibility of determining the responsible of the next actions. Of course, a responsible can delegate an action (or a set of actions) to a delegate agent. A delegate has the "local" responsibility of the action, but, after managing it, s\he has to return the responsibility to the responsible of the action. Both responsibles and delegates are not necessarily the physical executors of actions. Indeed, in many cases, the responsibility (and, possibly, the delegation) is given to physicians, but the physical execution is demanded to nurses or lab technicians.

To be a "proper" responsible, delegate or executor of an action, an agent must have a given ***qualification***, and, possibly, some specific ***competences***. To cope with such issues, we extended the META-GLARE formalism with six additional annotations:

- **responsible_qualification** (e.g. neurologist, gastroenterologist, …)**, delegate_qualification** (e.g. neurologist, gastroenterologist, …)**, executor_qualification** (e.g., nurse): such annotations specify what the qualification of the responsible, delegate, and executor of the action must be. A list of qualifications can be specified, meaning that the agent must have (at least) one of the qualifications in the list;
- **responsible_competence, delegate_competence, executor_competence:** they specify that the agent must have specific abilities (e.g. expert in the alcohol-related disorders management). Such an attribute is optional. A list of competences can be specified, meaning that the agent must have *all* the competences in the list.

When a CIG is being acquired, we impose that each action in it is annotated with a specification of a list of possible contexts and of a list of possible qualifications of responsibles and executors. This is mandatory, and the acquisition module is extended to support the acquisition of such annotations (see Section 4). On the contrary, competence annotations are optional. In case the competence list is empty, no specific restriction needs to be applied; otherwise, only the agents having the required competences are allowed to be responsible for or to execute the action at hand.

Example 1. The action "Brief intervention for hazardous and harmful drinking" (see action 11 in Fig. 4) in the alcohol-related disorders treatment GL [10] is described as follows:

- responsible_qualification: physician
- responsible_competence: \
- delegate_qualification: physician
- delegate_competence: \
- executor_ qualification: physician, nurse
- executor_competence: \

- context: community medicine, SERT medicine (SERT is the acronym for "SERvizio per le Tossicodipendenze", an Italian service similar to the Mental Health Service in U.S.A.), in-patient care, hospital ambulatory care.

In many practical cases, behind coordination of agents, it is also important to support the continuity of treatments. Indeed, it is preferable to assign "homogeneous" sets of actions in a CIG to the same responsible (or, in some cases, executor). For instance, it might be preferable that the same neurologist is responsible of all the neurological activities performed on a given patient, and that the different EMG examinations of a patient are executed by the same specialist. Notably, continuity constraints are interpreted as "preferential" constraints by our system (see Section 4), and admit violations (e.g., after a period, a physician may, for any reason, be unable to continue to treat a given patient). In META-GLARE, we support such a need through an independent data structure, which annotates CIGs with ***"continuity constraints"***, which specify sets of actions that should have (preferably) the same responsible, delegate and\or executor.

In the next sections, we will present how *annotations* are formalized in our approach, and how they are treated by META-GLARE.

3.1 Ontology of annotations

GL *annotations* can be modeled on the basis of three taxonomies, and of the relations between them. Part of the taxonomies and relations are graphically shown in Fig. 2. The ontology of *contexts* is a "part-of" taxonomy, in which each context can be further specified by its components. For instance, in Fig. 2, FAMILY, HOSPITAL and COMMUNITY MEDICINE are three possible contexts, and NEUROLOGY, GASTROENTOROLOGY and INTERNAL MEDICINE are some of the departments that are part of a hospital. *Qualifications* can be modeled through a standard "isa" taxonomy, in which each qualification can be further refined (*isa* relation) by its specializations. For instance, in Fig. 2, NURSE and PHYSICIAN are two possible qualifications. In turn, NEUROLOGIST, GASTROENTEROLOGIST and INTERNIST are specializations of PHYSICIAN. Analogously, also *competences* are modeled by an "is-a" taxonomy. For instance, in Fig. 2, (the competence in) PERIPHERAL NEUROPATHY is a specialization of (competence in) NEUROLOGY, which is a specialization of MEDICAL COMPETENCE.

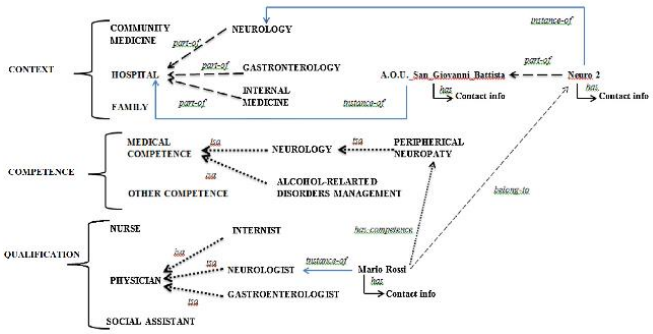

Fig. 2. Ontology of contexts, qualifications and competences and their instances (original figure in [12]).

Besides concepts, which denote classes of entities, the ontology also includes instances, denoting specific entities. Each instance is connected to its class through an "instance-of" relation. For example, in Fig. 2, Neuro2 is an instance of NEUROLOGY in Azienda Ospedaliera San Giovanni Battista (which is an instance of Hospital); Mario Rossi is an instance of NEUROLOGIST (thus, given the transitivity of the *isa* relation, Mario Rossi is also an instance of PHYSICIAN). It is worth noticing that, while the entities in the context and qualification taxonomies have instances, competences do not have them (they are individual concepts). Besides *part-of, isa* and *instance-of* relations, other relations are useful to represent the domain. *Agents* (which are instances of Qualifications) are related to the contexts they belong to by the *belong-to* relation. Additionally, agents may have competences, and this fact is represented by the *has-competence* relation. For instance, in Fig. 2, Mario Rossi belongs to Neuro2, and has specific competence about Peripheral Neuropathy. Contexts and persons have *contacts* (usually phone numbers).

Continuity constraints model the fact that, preferably, "homogeneous" sets of actions in a GL should be performed by the same agent. Such constraints are typically GL-dependent, and can be easily formalized in an independent data structure, to store three different partitions of the GL actions into subsets, to model:

(i) the sets of actions which should (preferably) have the same responsible;
(ii) the sets of actions which should (preferably) have the same delegate;
(iii) the sets of actions which should (preferably) have the same executor.

The definition of continuity constraints is a refinement process. First, the users can define the continuity constraints concerning the responsibles. Then, within each responsible-level continuity group, they can further specify continuity groups for possible delegates. Finally, continuity execution groups can be defined, within the delegate-level continuity groups.

Temporal constraints play a fundamental role in clinical guidelines. For example, temporal indeterminacy, constraints about duration, delays between actions, and periodic repetitions of actions are essential in order to cope with clinical therapies. As a matter of fact, in most therapies, actions have to be performed according to a set of temporal constraints concerning their relative order, their duration, and the delays between them. Additionally, in many cases, actions must be repeated at regular (i.e., periodic) times. For instance, consider Ex.1. In Ex. 1, the instances of the melphalan treatment must respect the temporal pattern "twice a day, for 5 days", but such a pattern must be repeated for six cycles, each one followed by a delay of 23 days, since the melphalan treatment is part of the general therapy for multiple myeloma.

Example 2. The therapy for multiple myeloma is made by six cycles of 5-day treatments, each one followed by a delay of 23 days (for a total time of 24 weeks). Within each cycle of 5 days, two inner cycles can be distinguished: the melphalan treatment, to be provided twice a day, for each of the 5 days, and the prednisone treatment, to be provided once a day, for each of the 5 days. These two treatments must be performed in parallel.

Coping with such temporal constraints involves two main challenges:

(1) Developing a representation formalism to capture them,

(2) Developing temporal reasoning algorithms to reason with them (e.g., to check their consistency, and to make explicit the implied constraints).

Notably, only in case the temporal reasoning algorithms are proved to be correct and complete (with respect to the representation formalism) one can assure that:

(i) the constraints are consistent (notably, if they are not, there is no execution of the CIG containing them that can satisfy all of the constraints)

(ii) the window of time associated with CIG actions are correct with respect to the constraints (i.e., do not contain any time that is not "legal", given the temporal constraints in the CIG

(see the discussion in [11] for more details).

As a consequence, a fundamental step in the development of a manager for temporal constraints is the analysis of the trade-off between the expressiveness of the representation formalism and the tractability of correct and complete temporal reasoning algorithms [19].

4 Temporal Reasoning

4.1 Representing CIG temporal constraints

As regarding the representation formalism, the main challenge is how to represent the constraints about repetitions, and how to integrate such a representation with the representation of the "standard" temporal constraints (i.e., the temporal constraints between non-repeated actions).

"Standard" temporal constraints. We have chosen to model the temporal constraints concerning "standard" (i.e., non-repeated) actions in the CIGs, using a well-known and widely-used AI framework, namely STP [20].

In STP, a set of constraints is modeled as a conjunction of bounds on differences of the form $c \leq x - y \leq d$, which have an intuitive temporal interpretation, namely that the temporal distance between the time points x and y is between c (minimum distance) and d (maximum distance). In STP the correct and complete propagation of the constraints (e.g., for consistency checking) can be performed in a time cubic in the number of time points, and the *minimal network* of the constraints is provided as output (i.e., the minimum and maximum distance between each pair of points) [20]. The minimal network can be computed by an "all-to-all shortest paths" algorithm such as the Floyd-Warshall's one. The STP framework can be used to model precise or imprecise temporal locations (dates), durations, delays between points, and different forms of qualitative temporal constraints between time points and/or time intervals (see [21, 22]). Unfortunately, constraints about repeated actions cannot be easily captured by the STP framework.

Temporal constraints about repetitions. In our approach, the constraints on repetitions and periodicities are temporal constraints of the form

<nRepetitions, I-Time, repConstraints, conditions>.

nRepetitions represents the number of times that the action must be repeated; **I-Time** represents the time span in which the repetitions must be included; **repConstraints** may impose a pattern that the repetitions must follow; **conditions** allows to

express conditions that must hold so that the repetition can take place. Informally, we can roughly describe the semantics of the quadruple as the natural language sentence "*repeat the action **nRepetitions** times in exactly **I-Time** according to **repConstraints**, if **conditions** hold*".

repConstraints$_i$ is a (possibly empty) set of pattern constraints, representing possibly imprecise repetition patterns. Pattern constraints may be of type:

- **fromStart(min, max)**, representing a (possibly imprecise) delay between the start of the I-Time and the beginning of the first repetition;
- **toEnd(min, max)**, representing a (possibly imprecise) delay between the end of the last repetition and the end of the I-Time;
- **inBetweenAll(min, max)** representing the (possibly imprecise) delay between the end of each repetition and the start of the subsequent one;
- **inBetween((min$_1$, max$_1$), …, (min$_{nRepetitions_{i-1}}$, max$_{nRepetitions_{i-1}}$))**, representing the (possibly imprecise) delays between each repetition and the subsequent one.

For example, the temporal constraint modeling the repetition of the chemotherapy (as a whole) in Ex.1 is

<6, 6wk, {inBetweenAll(23d, 23d), toEnd(23d, 23d)}, ∅>.

Notably, our formalism to represent repetitions is recursive, in the sense that one can specify repetitions of repetitions, as a list of specifications <**<nRepetitions$_1$, I-Time$_1$, repConstraints$_1$, conditions$_1$>**, … ,**<nRepetitions$_k$, I-Time$_k$, repConstraints$_k$, conditions$_k$>>** as above.

For example, the temporal constraint modeling the melphalan cycle is the following:

<<5, 5d, ∅, ∅>, <2, 1d, ∅, ∅>>.

Integrated representation of temporal constraints. The formalism we propose to consider both "standard" constraints and constraints about repetitions is the STP-tree [11]. In an STP-tree, we model the constraints regarding repeated actions into separate STPs, one for each repeated action. The root of the tree (node N1 in the example in Fig. 3) is the STP which homogeneously represents the constraints (including the ones derived from the control-flow of actions in the guideline) between all the actions in the guideline (e.g., in N1, the fact that the duration of the chemotherapy is 168 days), except repeated actions. Each node in the tree is an STP, and has as many children as the number of repeated actions it contains. Each edge in the tree connects a pair of endpoints in an STP (the starting and ending point of a repeated action) to the STP containing the constraints between its subactions, and it is labeled with the list of properties describing the temporal constraints on the repetitions. For example, in Fig. 3, we show the STP-tree representing the temporal constraints involved by the example Example 2 above.

Additionally, an independent STP must be used in order to represent the temporal constraints about the specific instances of the actions of the guidelines, as emerging from the executions of the guidelines on specific patients.

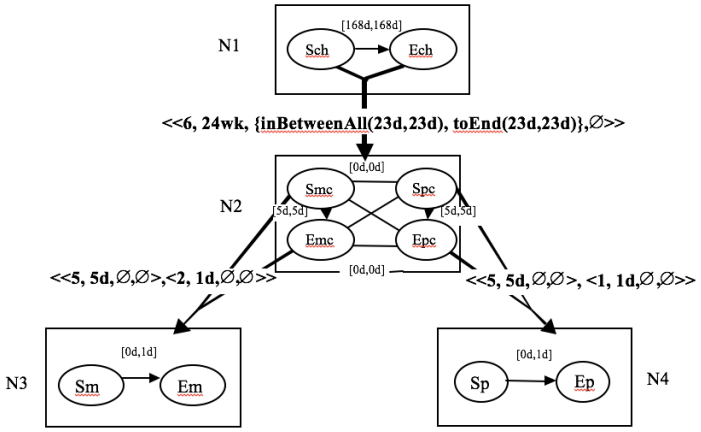

Fig. 3. STP-tree for the multiple myeloma chemotherapy guideline in Example 2. Edges inside STPs represent STP constraints; edges between STPs represent repetitions. *Sch*, *Ech*, *Smc*, *Emc*, *Spc*, *Epc*, *Sm*, *Em*, *Sp* and *Ep* stand for the starting (*S*) and ending (E) points of chemotherapy, melphalan cycle, prednisone cycle, melphalan treatment and prednisone treatment, respectively.

4.2 Temporal reasoning

In order to propagate the temporal constraints in the STP-tree, and to check their consistency, it is not sufficient to check the consistency of each node separately. In such a case, in fact, we would neglect the repetition/periodicity information. Temporal consistency checking, thus, proceeds in a top-down fashion, starting from the root of the STP-tree. Basically, the root contains a "standard" STP, so that the Floyd-Warshall's algorithm can be applied to check its consistency. Thereafter, we proceed top-down towards the leaves of the tree. For each node *X* in the STP-tree (except the root), we progress as shown in the algorithm *STPs_tree_consistency* in Algorithm 1.

```
function STP_tree_consistency(X: STPNode, RepSpec = (R1 =
<nRepetitions1, I-Time1, repConstraints1, conditions1>, …,
Rn = <nRepetitionsn, I-Timen, repConstraintsn, condi-
tionsn>)) : STP
   1. check that the repetition/periodicity constraint is
      well-formed (i.e., that repetitions nest properly)
   2. compute Max, i.e. the maximum duration of a single
      repetition of X according to RepSpec
   3. impose in X that the maximum distance between each
      pair of points is less or equals Max
   4. X ← FloydWarshall(X)
   5. if X = INCONSISTENT then return INCONSISTENT else
      return X
```

Algorithm 1. Algorithm for checking the consistency of a guideline (represented as an STP-tree).

STP_tree_consistency takes in input the STP-node that must be checked (i.e. *X*) and the repetition/periodicity constraint (i.e., the repetition specification in the arc of the STP-tree entering node *X*), and gives as an output an inconsistency or, in the case of consistency, the local minimal network of the constraints in *X* considering also the repetition/periodicity constraints.

In step 1 it checks whether the repetition/periodicity constraint is "well-formed", i.e. if it is consistent when it is taken in isolation (e.g., in a constraint such as R_1 = <*nRepetitions*$_1$, *I-Time*$_1$, *repConstraints*$_1$, *conditions*$_1$> and R_2 = <*nRepetitions*$_2$, *I-Time*$_2$, *repConstraints*$_2$, *conditions*$_2$>, *I-Time*$_2$ must be contained into *I-Time*$_1$). In step 2 it computes the maximum duration of a single repetition. This is obtained by considering the time that allows to perform a repetition assuming that all the other repetitions have the minimum possible duration. In step 3 it adds to the STP *X* the constraints stating that the maximum "duration" of *X* must be the computed maximum duration of a single repetition of *X*. Finally, in step 5 it checks the consistency of the "augmented" STP *X* via the Floyd-Warshall's algorithm.

Complexity. Considering that the number of nesting levels, in the worst case, is less than the number of classes, the algorithm is dominated by step 4, that is O(C^3), where *C* is the number of actions in the guideline.

Property 1. The top-down visit of the STP-tree is complete as regards consistency checking of the constraints in the STP-tree.

Proof (sketch). The all-to-all-shortest-paths algorithm is complete for STP frameworks. The only constraints relating actions (or, better, time points) in different STPs in the tree are located on the arcs of the tree, and, by definition, the STP-tree does not allow loops between STPs. Since, in our approach, I-Times (if any) must be provided in a precise (exact) way, they cannot be further "restricted" by constraint propagation. Thus, there is no need to propagate forward and backward the constraints along the tree, and a top-down visit of the tree is sufficient[1].

5 Agent coordination facilities

In the previous section, we have discussed the extensions to (i) the META-GLARE language and to (ii) the META-GLARE basic ontology to cope with the "annotations" required to support agent coordination. Indeed, such extensions to the knowledge representation languages must be paired with software modules to acquire and navigate them, and to use them to support agents in the coordinated execution of CIGs. In the following, we detail such modules. In Section 5.1, we discuss the tools we provide to navigate the ontological knowledge. Such tools are exploited by the acquisition module (discussed in Section 5.2) to acquire proper and "standardized" annotations for the actions of the CIGs. In Section 5.3, we discuss how we have extended the META-GLARE execution engine in order to support agent coordination. Last, but not least, in Section 5.4 we discuss the facilities we provide to the healthcare agents.

[1] Notice, however, that the above reasoning mechanism does not provide the minimal network between all the actions in the STP-tree.

5.1 Navigation Module

The ontology pointed out in Section 3.1 is important for two main tasks:
(i) during the acquisition of CIGs, to support a "standardized" annotations of CIGs with qualifications, contexts and competences taken from the ontology, and
(ii) during the execution of a CIG action, to support the search of agents having the properties required for the management (as a responsible, delegate, or executor) of such an action (as specified by the action's annotations).

To manage such needs, we have developed two different facilities to navigate the ontology:
(1) schema browsing,
(2) instance browsing.

The *schema browsing* facility is used at acquisition time, and allows experts to navigate the ontology (using the *part-of* and *is-a* relations) and to find qualifications/contexts/competences needed to annotate the CIG actions.

The *instance browsing* facility is used at execution time to support agent coordination in the selection of "proper" agents. It allows users to find a specific agent on the basis of the relations *part-of*, *is-a*, *instance-of*, *has-competence*, *belong-to*. For example, it supports the search of an agent on the basis of a qualification (e.g. Physician), a context (e.g. Neurology) and, possibly, a competence (e.g. Peripheral Neuropathy). This facility can give in output (i) one or more agents (and their contact information) satisfying the requirements, or (ii) one or more specific contexts, in which agents having the required qualification and competences operate.

5.2 Acquisition

We have extended META-GLARE with an ***annotation support***, supporting the acquisition of CIG annotations. We have developed a user-friendly Graphical User Interface (GUI). Such a GUI takes advantage of the schema browsing facility discussed in Section 5.1 to find in the ontology the proper qualifications, contexts and competences needed to annotate actions.

Moreover, we have developed an ad-hoc module to support the definition and the acquisition of ***continuity constraints***. The user can use this module to browse the CIG and to specify the set of actions in the CIG which belong to a continuity constraint by selecting such actions in the graphical representation of the CIG.

5.3 Execution Engine

The execution engine of META-GLARE has been deeply extended in order to support coordination in the execution of CIGs. Specifically, we provide facilities to support the identification of the responsible(s), of the delegate(s) and of the executor(s) for the next action(s) according to the CIG annotations, and to consider the window of time when the actions have to be executed, according to the temporal constraints in the CIG and to the times when the previous CIG actions have been executed.

The main data structure supporting CIG execution is an ***agenda*** [13], containing a set of pairs $\{(A_1,T_{A1}), \ldots,(A_k,T_{Ak})\}$ representing the actions to be executed next

(A_1,…,A_k), and the window of time within which the actions have to be executed (T_{A1}, …,T_{Ak}). Notably, (i) more than one pair may appear in the set, to support concurrent execution, and (ii) the time window of the action is evaluated by a temporal reasoner, as described in Section 4 above.

To support the management of responsibilities\delegations\executions, the agenda has to be integrated with additional data structures. First of all, to grant the continuity of treatments (as modeled by the continuity constraints) at each time, the system has not only to support the management of the actions in the agenda, but also of each action belonging to the Responsibility Continuity Group ("RCG" in the following algorithm) of such actions. For each action A of such actions, a new data structure (called ***agent stack*** of A) has to be introduced, to store two different types of information: (i) the time window in which the action has to be executed, and (ii) a stack of the form $Stack_A$: $<(X_1, role_1), \ldots,(X_k, role_j)>$ where X_i is a specific agent, and $role_h$ her role in the management of the action A (i.e., responsible (R), delegate (D), or executor (E))[2] storing the different agents involved in the management of such an action, and the window of time.

An important issue regards the evaluation of the time windows associated with (i) the actions in the agenda, and with (ii) the actions in the RCG of the agenda actions. Both windows (i) and (ii) are evaluated by the temporal reasoner, as discussed in Section 4. However, there is a subtle but important difference. The formers are evaluated whenever an action is entered in the agenda, i.e., whenever it has actually to be executed. Therefore, the time window exactly indicates to the executor the range of time when the action has to be performed. On the other hand, the actions in the RCG groups are actions which do not necessarily have to be executed soon: for instance, the RCG of a neurologist visit may include all the following neurologist's visits included in the CIG. From one side, it is important that the window of time in which such visits will have to be executed is evaluated soon. In such a way, when asking a neurologist to accept the responsibility\execution of such actions, we can also indicate her\him the (approximate) window of execution. This is an important information, since the agent may accept or reject also on the basis of her\him time availabilities. However, such a window is generally a quite approximate one. Notably, in principle, whenever a new CIG action is executed, the temporal window of each RCG action can be restricted (by adding the time of execution of the action executed last to the previous set of constraints, and propagating the resulting set of constraints through the temporal reasoner).

Thus, an important choice we have to take in our approach is whether and when we have to update the time windows associated with actions. Since temporal reasoning is quite computational expensive, we have chosen to evaluate such windows only twice:

(i) First, they are evaluated whenever an action becomes part of an RCG group (so that we can show the window to agents while asking them to accept to manage them);

[2] At the time of the execution of an action A, its agent stack $Stack_A$ should contain the responsible (bottom of the stack), a certain number of delegates (zero delegates in case no delegation has been performed; more than one delegate is possible, to support delegation of delegations), and one executor (which might also be the last delegate, or the responsible).

(ii) Second, they are evaluated whenever actions enter into the agenda (i.e., when actions have actually to be executed next). At such a time, it is fundamental to give to the executor the precise window of execution (considering the constraints in the CIGs and its current execution status).

The execution of a CIG starts with an initialization phase (see Algorithm 2 in the following). All the initial actions are inserted into the agenda, together with the window of time in which they must be executed. For each one of such actions, and for each action belonging to the Responsibility Continuity Group ("RCG" in the following algorithm) of such actions, the approximate time window is evaluated, and the agent stack is initialized.

Notably the responsibles of the first actions are predetermined and provided as input to the execution engine.

The CIG execution engine operates as described by Algorithm 3. For each action A in the agenda, the CIG execution engine starts its execution by sending the *execute* message (line 2) to the agent on the top of the agent stack $Stack_A$, asking her to manage (according to her role) the action A in the time window T_A. Notably, as discussed above, the time window T_A is re-evaluated by META-GLARE at the time when A is inserted in the agenda (i.e., when the execution engine determines the CIG actions to be executed next). In case action A is executed (line 3), A is removed from the Agenda (line 4). Thus, the execution engine evaluates the set S of the next actions in the CIG to be executed, using the *get_next* function (line 5). Notably, identifying the next actions which have to be executed during the execution of a CIG is a standard operation (see [23]), and consider [24] as regards the META-GLARE approach).

Each action B belonging to S is pushed onto the Agenda. Then, in the case that B has not yet a responsible (i.e., $Stack_B$ has not been created yet), the responsible of A has to find the responsible for B, and for all the actions in RCG(B). To find such a responsible, we first call the temporal reasoner to evaluate, for each action $C \in$ RCG(B), its temporal window. A message "next_responsible?" containing the resulting set of pairs (action, time-window) (denoted by $RCG^T(B)$ in the algorithm -see line 12) is sent by the execution engine to the responsible of A (i.e. the agent stored at the bottom of $Stack_A$) to advise her\him to search for a responsible for B and the other actions in $RCG^T(B)$ (line 12). Notably, since we manage continuity groups, the responsible of an action X in the CIG can be already determined before the time when X is inserted in the agenda (due to the fact that X belongs to the responsibility continuity group of another action already inserted in the agenda). Finally, the stack of A is deleted (line 13).

```
1.  let S={(A1,MA1), …, (Ak,MAk)} be the set of the start-
     ing actions of CIG, and of their responsibles.
2.  for each action Ai∈S
      evaluate its temporal window Ti;
      push (Ai,Ti) onto the Agenda
3.  for each (A, TA) in Agenda do
      for each B ∈ RCG(A) do
        evaluate the temporal window TB
        initialize(StackB, MA, "R", TB)
```

Algorithm 2. Pseudocode of the initialization of the **CIG executor engine**.

```
1.   for each (A, TA) in Agenda do
2.      OUT←send(top(StackA),execute(A,TA))
3.      if (OUT == OK) then
4.         Remove A from Agenda
5.         S ← get_next(A)
6.         for each B in S do
7.             Evaluate the time window TB
8.             push  (B, TB) onto the Agenda
9.             if B has no responsible then
10.               for each C∈RCG(B)
11.                 evaluate its time window
12.                 send(next_responsible?
                      (bottom(StackA), RCGT(B))
13.       Delete StackA
14.     else
15.       pop(stack of A)
16.       goto 2
```

Algorithm 3. Pseudocode of **CIG executor engine**.

Otherwise, in case A is not executed (i.e. the agent on the top of StackA rejects its role), a pop on $Stack_A$ is performed (line 15). Thus, A remains in Agenda and the engine executor has to handle it again sending an execute message to the new top of the stack of A.

Notably, we support the fact that an agent accepts the responsibility, the delegation or the execution of a set of actions (all the actions in a continuity group) but, later on, stops to operate on some of the accepted actions. In such a case, the CIG execution engine "goes up" in the agent stack of the "rejected" actions to find new delegates or responsibles. Notably, though the current responsible may decide not to operate any more on the actions she previously accepted, before "retiring" she has to find a new responsible for them.

5.1 Support to Agents

As discussed above, in META-GLARE we distinguish among three different "roles" that agents can play in the management of a CIG action: *responsibles*, *delegates*, and *executors*. Each role has different rights and duties, so that META-GLARE provides a different support for each role.

A first set of facilities has the goal of supporting agents to find proper responsibles, delegates and executors of one or more CIG actions.

The ***find_responsible*** function allows an agent to find a "proper" responsible for an action or a set of actions (in a continuity group). First of all, it calls the *instance browsing* facility which returns the set of "candidate" agents that satisfy the requirements expressed by the CIG annotations (plus possible additional requirements introduced by the agent). The agent has to find in such a set a candidate willing to accept the new responsibility. Therefore, the agent selects one of the candidates and sends her\him an ***accept_responsibility?***$(\{(A_1,T_{A1})\ldots,(A_k,T_{AK})\})$ message, indicating all the actions and their time windows. If the candidate accepts them, the *agent stacks* of A_1, …, A_K are created, specifying the new responsible, otherwise the search for a responsible goes on, considering (in the order chosen by the agent) the following candidate.

The ***find_delegate?*** and ***find_executor***? functions operate similarly, supporting the identification of appropriate delegates (if desired) and executors (compulsory) to actions (through the use of ***accept_delegation?*** and ***accept_execution?*** messages) and taking into account continuity groups.

In our approach, each agent has the possibility to receive and send different types of messages, depending on her current role (responsible, delegate, executor).

Responsible.

Receipt of an execute(A,T_A) message. In our approach, the CIG execution engine only sends *execute(A,T_A)* messages to agents indicated in the agent stack of the action A, i.e., to agents that have already explicitly accepted to execute such an action. However, the acceptance time could be long before the time when the execute message is received. As a matter of facts, an agent may have accepted to execute all the actions in a continuity group, along a quite long range of time. Therefore, despite the fact that the agent has already accepted, we also support the case that, for any reason, at the time when A must be executed, the responsible wants\needs to decline (e.g., the responsible of a patient with a chronic disease may retire or move away). We allow her to do so, but with a restriction: before "retiring", the current responsible must find a new responsible for the action A and the other actions (not executed yet) in the responsibility continuity group of A (using the *find_responsible* function).

On the other hand, if the responsible still retains her responsibility, she still has several options: she can

(i) **delegate** DCG(A) (i.e., A and all the other actions in the Delegate Continuity Group of A), through the *find_delegate* function,

(ii) **find an executor** for ECG(A) (i.e., A and all the other actions in the Executor Continuity Group of A, through the *find_executor* function,

(iii) directly **execute** A herself.

Receipt of a next_responsible?({(A1,T_{A1}),…, (Ak,T_{Ak})} message.

The current responsible is in charge of identifying an appropriate responsible for the actions A_1… A_k. To support her in this task, we provide the *find_responsible* function, described above.

Receipt of an accept_responsibility?({(A1,T_{A1}),…, (Ak,T_{Ak})} message.

The agent may accept or reject the new responsibility.

Notably, soon after the acceptance of the responsibility of a set of actions {(A1,T_{A1}),…, (Ak,T_{Ak})} (a Responsibility Continuity Group of actions), the new responsible can soon search for delegates or executors for such actions (considering their Delegate and Executor Continuity Groups respectively), using the *find_delegate* and *find_executor* facilities. In such a way, the mechanism of determining delegates and executors can proceed in a (partially) asynchronous way with respect to the actual execution of actions in the CIG.

Delegate.

When a delegate receives an ***execute(A,T_A)*** *message,* she may decline. Such a situation is directly managed by the execution engine (see Algorithm 2), which pops the delegate from $Stack_A$ and sends the *execute(A,T_A)* message to the new top of the stack. On the other hand, if the delegate retains her role, she can **delegate** DCG(A), **find an executor** for ECG(A) or directly **execute** A herself. Additionally, she may accept or reject an ***accept_delegation?({(A1,T_{A1}),…, (Ak,T_{Ak})}*** *request.*

Notably, as in the case of responsibles, soon after the acceptance of the delegation of a set of actions {(A1,T_{A1}),…, (Ak,T_{Ak})} the new delegate can look for delegates or executors for such actions.

Executor.

When an executor receives an ***execute(A,T_A)*** *message,* she may decline. Such a situation is directly managed by the execution engine, as described above (concerning delegates). Otherwise, she must execute action A within the time interval T_A. Additionally, she may accept or reject an ***accept_execution?({(A1,T_{A1}),…, (Ak,T_{Ak})}*** *request.*

6 A Case Study

In this section, we present an application of our approach to a GL for alcohol-related problems [10], adapted to the Italian context (see Figg. 4 and 5). First, we describe the first part of CIG that has been acquired in META-GLARE. Then, we show META-GLARE in action.

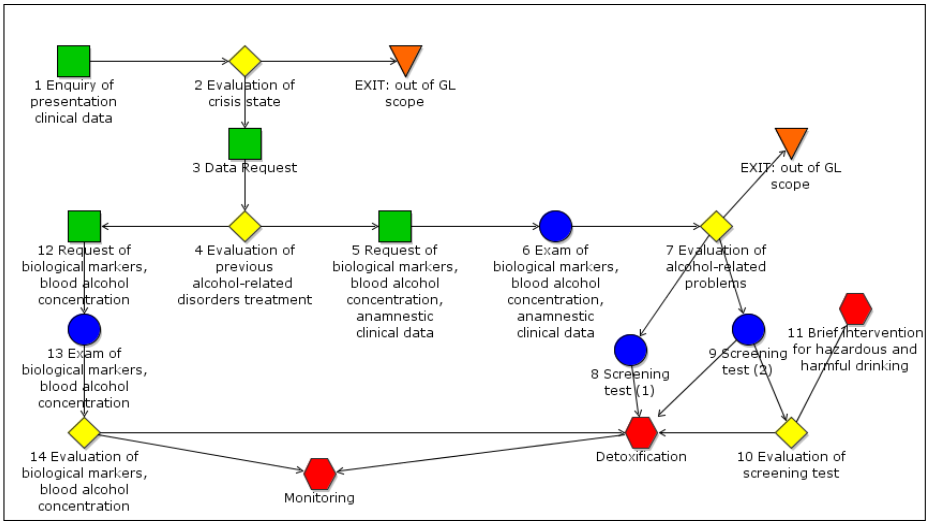

Fig. 4. META-GLARE graphical representation of part of the CIG on the treatment of alcohol-related disorders (original figure in [12]).

Responsibility Continuity Group : Responsible Qualification	Delegation Continuity Group : Delegate Qualification	Execution Continuity Group : Executor Qualification	Context	Action number
RG1 : R1, R4	DG1 : R1, R4	EG1 : R1, R2, R4	C1, C2, C3, C4, C5	1
			C1, C2, C3, C4, C5	2
	DG2 : R1, R4	EG2 : R1, R2, R4	C1, C2, C3, C4, C5	3
			C1, C2, C3, C4, C5	4
RG2 : R1	DG3 : R1	EG3 : R1	C1, C2, C3, C4	5
			C1, C2, C3, C4	7
RG3 : R1	DG4 : R1	EG4 : R5	C4	6
RG4 : R1	DG5 : R1	EG5 : R1	C1, C2, C3, C4	12
			C1, C2, C3, C4	14
RG5 : R1	DG6 : R1	EG6 : R5	C4	13
RG6 : R1	DG7 : R1	EG7 : R1, R2	C1, C2, C3, C4	8
	DG8 : R1	EG8 : R1, R2	C1, C2, C3, C4	9
			C1, C2, C3, C4	10
	DG9 : R1	EG9 : R1, R2	C1, C2, C3, C4	11

Fig. 5. The annotations of actions in Fig. 4 (original figure in [12]).

The CIG starts with the collection of the patient clinical data (query action 1). The data are used to diagnose whether the patient is currently experiencing a crisis state (decision action 2). Since the management of an alcohol-related crisis is outside the GL

scope, the CIG ends. If, on the other hand, the patient is not experiencing a crisis, her clinical history is collected (query action 3), to distinguish whether it is the first time that the patient is in treatment for alcohol-related problems, or not (decision action 4). New patients require the collection of biological markers, blood alcohol concentration and anamnestic data (data request 5 and work action 6). On the other side, patients who were already cared for alcohol-related disorders need that only biological markers and blood alcohol concentration will be collected and examined (data request 12 and work action 13). For such patients, an evaluation of such data (decision action 14) is performed to decide whether monitoring them or proceed with a detoxification. Focusing on new patients, after the collection of clinical information, a diagnostic decision about the presence of alcohol-related problems must be taken (decision action 7). In the case that the patient does not show alcohol-related problems, the CIG execution is ended. Otherwise, two different treatments can be applied, depending on the severity of alcohol-related problems; both start with a screening test (work actions 8 and 9 respectively). Focusing on patients with a mild alcohol-dependence (work action 9), after evaluating the screening test results (decision action 10), the patient can be selected for a brief intervention for hazardous and harmful drinking (composite action 11), which basically consists in a set of motivational interviews.

In Table 1 and Table 2 we show temporal information about the first part of the CIG that is used to exemplify the CIG execution. In particular, Table 1 shows the (minimal and maximal) duration of actions and Table 2 shows the (minimal and maximal) delay between actions (i.e. the delay between two actions is the (minimal/maximal) duration of the arc connecting them).

Table 1. Duration of the actions in the CIG.

Action number	Duration (min-max)
1	10-20 minutes
2	10-20 minutes
3	20-30 minutes
4	10-20 minutes
5	30-60 minutes
6	1-3 days
7	10-30 minutes
8	20-40 minutes
9	20-40 minutes
10	10-30 minutes
11	1-3 hours
12	30-60 minutes
13	1-3 days
14	10-30 minutes

Table 2. Delay between actions in the CIG.

Delay between actions (from - to)	Duration (min-max)
1– 2	0–20 minutes
2– 3	0–30 minutes
3 – 4	20–60 minutes
4 – 5	1 - 2 days
4 – 12	1 - 2 days
5– 6	1 - 2 days
6 – 7	1 – 2 days
7– 8	1-6 hours
8 – 9	1-6 hours
9 – 10	1-2 hours
10 – 11	1-3 days
12 – 13	1 - 2 days
13 – 14	1-3 days

Exploiting the annotation support (see Section 4.2), we have *annotated* all the actions defining the possible qualification(s) of its responsible, delegate (if any) and executor. Moreover, we have identified and specified the continuity groups in the CIG. In this specific application, possible values for the attributes in the annotations are the following:

- context: Community medicine (C1), SERT medicine (C2), in-patient care (C3), hospital ambulatory care (C4), social services (C5);
- qualification: physician (R1), nurse (R2), healthcare assistant (R3), social assistant (R4), laboratory technician (R5).

The treatment continuity criteria demand that all the actions corresponding to the initial evaluation of the patient status (actions 1-4) must have a unique responsible (responsibility continuity group RG1; see Fig. 5), which must be a physician (R1) or a social assistant (R4). The continuity group RG1 is further divided into two "subparts", corresponding to the two delegation continuity groups DG1 and DG2. In particular, DG1 corresponds to the identification of a crisis currently in progress (actions 1 and 2) and DG2 corresponds to the identification of previous alcohol-related disorder treatments (actions 3 and 4). Moreover, due to the execution continuity constraints, action 1 and action 2 belong to a single execution continuity group (EG1) and the actions 3 and 4 belong to a single execution continuity group (EG2). The executor of the actions in EG1 must be a physician (R1), or a social assistant (R4), or a nurse (R4). The actions in EG2 have the same constraints and the same annotations. After action 4, there are two alternative treatment paths. One path manages patients who are treated for alcohol correlated problems for the first time. Such a path is annotated with a responsibility continuity group RG2 on the actions to evaluate the patient's problem (action 5 and 7) and with a responsibility group RG3 on the exams needed for such an evaluation (action 6). RG2 and RG3 require a physician (R1) as responsible. Notably, a continuity group can contain non-contiguous actions (e.g., RG2 is composed by actions 5 and 7 which are not contiguous in the CIG).

In the following, we exemplify how META-GLARE extended execution engine can work on the above part of the CIG.

STEP 0: We suppose that the execution of the CIG starts at day 1 hour 10 minute 0. At the beginning, the META-GLARE executor engine identifies the first action (i.e. action 1) and puts it in the agenda. We suppose that agent A, a social assistant in the social services SS1, is the responsible of action 1. Since action 1 is in RG1, the responsibility of A regards all the actions belonging to such a continuity group (i.e., actions 1, 2, 3 and 4). Thus, the agent stacks of the four actions are created and initialized with X as responsible and with their time windows. In the initial step, the executor engine creates the agent stacks for actions 1-4 as follows (line 3 of Algorithm 1):
agent $stack_1$: (<(1, 10,0), (1, 10, 0), (1, 10, 10), (1, 10, 15)>,<(A,R)>);
agent $stack_2$: (<(1, 10, 10), (1, 10, 25), (1, 10, 20), (1, 10, 45)>,<(A,R)>);
agent $stack_3$: (<(1, 10, 20), (1, 11, 15), (1, 10, 50), (1, 11, 55)>,<(A,R)>);
agent $stack_4$: (<(1, 11, 10), (1, 12, 55), (1, 11, 20), (1, 13, 15)>,<(A,R)>).

The agenda of the execution engine contains only action 1 and its temporal window, represented by the quadruple <(1, 10, 0), (1, 10, 0), (1, 10, 10), (1, 10, 15)>, that means that the action has to start between day 1 hour 10 minute 0 (i.e. the first triple) and day 1 hour 0 minute 0 (i.e. the second triple) and has to end between day 1 hour 10 minute 10 (i.e. the third triple) and day 1 hour 10 minute 15 (i.e. the last triple).

Agenda: <1,<(1, 10,0), (1, 10, 0), (1, 10, 10), (1, 10, 15)>.

STEP 1: the executor engine sends an *execute* message for each action in the agenda (line 2 on Algorithm 2). In the agent there is only action 1, therefore the executor engine sends a message to the agent in the top element of the $stack_1$ (i.e., to A) to perform the execution of action 1. A receives the *execute* message and she decides to execute action 1 (i.e. she takes also the role of executor). Thus, A is put in the agent stack of action 1 as executor, and since actions 1 and 2 belong to the execution continuity group EG1, she has also the role of executor of action 2 (i.e. A is pushed as executor also onto the agent stack of action 2). At this point, the status of agent stacks and the agenda is the following:
agent $stack_1$: (<(1, 10,0), (1, 10, 0), (1, 10, 10), (1, 10, 15)>,<(A,R), (A,E)>);
agent $stack_2$: (<(1, 10, 10), (1, 10, 25), (1, 10, 20), (1, 10, 45)>,<(A,R), (A,E)>);
agent $stack_3$: (<(1, 10, 20), (1, 11, 15), (1, 10, 50), (1, 11, 55)>,<(A,R)>);
agent $stack_4$: (<(1, 11, 10), (1, 12, 55), (1, 11, 20), (1, 13, 15)>,<(A,R)>).

A executes action 1 returning "OK" (line 2). The execution of action 1 lasts 10 minutes. Since action 1 has been executed, the executor engine removes it from the agenda (line 4). Then (line 5), the next action of the CIG is found (i.e. action 2) and it is put in the agenda with its time window and the agent stack of action 2 is updated with such a new time window (line 8). Action 2 has already a responsible, thus the agent stack of action 1 is simply deleted.

agent $stack_2$: (<(1, 10, 10), (1, 10, 20), (1, 10, 20), (1, 10, 40)>,<(A,R), (A,E)>);
agent $stack_3$: (<(1, 10, 20), (1, 11, 15), (1, 10, 50), (1, 11, 55)>,<(A,R)>);
agent $stack_4$: (<(1, 11, 10), (1, 12, 55), (1, 11, 20), (1, 13, 15)>,<(A,R)>).

Agenda: <2, <(1, 10, 10), (1, 10, 20), (1, 10, 20), (1, 10, 40)>>.

STEP 2: the above procedure is similarly repeated for action 2 in the Agenda. We suppose that, after receiving the message, A, who is registered as executor of action 2, executes it. A decides that the patient is not experiencing a crisis. Action 2 starts at day 1 hour 10 minute 10 and ends at day 1 hour 10 minute 30 (i.e. action 2 lasts 20 minutes). Thus, the next action is action 3 and then action 3 is put in the agenda with its time window and as consequence the time window in agent $stack_3$ is updated. Also in this case, action 3 has its responsible already defined (i.e. A).
agent $stack_3$: (<(1, 10, 30), (1, 11, 0), (1, 11, 0), (1, 11, 50)>,<(A,R)>);
agent $stack_4$: (<(1, 11, 10), (1, 12, 55), (1, 11, 20), (1, 13, 15)>,<(A,R)>).
Agenda: <3, <(1, 10, 30), (1, 11, 0), (1, 11, 0), (1, 11, 50)>>.

STEP 3: Action 3 is the only action in the Agenda and is managed by sending an *execute* message to its responsible A (i.e. A is on the top of stack in agent $stack_3$). In this case we suppose that A decides to ***delegate*** such an action. Exploiting the instance browsing facility of our navigation tool (see Section 4.1), A searches for an agent satisfying the requirements for action 3 (i.e. a social assistant or a physician in her context). Through the navigation tool, A obtains a list of possible agents. She selects a preferred one from the list and asks for acceptance, until she receives a positive reply. We suppose that (possibly after some negative replies of social assistants due to the time window associated to the action) the social assistant B accepts. Since actions 3 and 4 belong to the same delegation continuity group (i.e. DG2), B is also delegated for action 4.

agent $stack_3$: (<(1, 10, 30), (1, 11, 0), (1, 11, 0), (1, 11, 50)>,<(A,R), (B,D)>);
agent $stack_4$: (<(1, 11, 10), (1, 12, 55), (1, 11, 20), (1, 13, 15)>,<(A,R), (B,D)>).

Agenda: <3, <(1, 10, 30), (1, 11, 0), (1, 11, 0), (1, 11, 50)>>.

B decides to be the executor of action 3. Since actions 3 and 4 belong to the same execution continuity group EG2, B is nominated also as the executor of action 4 and she is put in the two stacks as executor.
agent $stack_3$: (<(1, 10, 30), (1, 11, 0), (1, 11, 0), (1, 11, 50)>,<(A,R), (B,D), (B,E))>);
agent $stack_4$: (<(1, 11, 10), (1, 12, 55), (1, 11, 20), (1, 13, 15)>,<(A,R), (B,D), (B,E)>).

Agenda: <3, <(1, 10, 30), (1, 11, 0), (1, 11, 0), (1, 11, 50)>>.

Y executes action 3. Action 3 starts at day 1 hour 10 minute 40 and lasts 40 minutes. Then action 4 is put in the agenda as next action with it is time window and as consequence the time windows in agent $stack_4$ are updated.

agent $stack_4$: (<(1, 11, 40), (1, 12, 20), (1, 11, 50), (1, 12, 40)>, <(X,R),(B,D),(B,E)>).

Agenda: <4,<(1, 11, 40), (1, 12, 20), (1, 11, 50), (1, 12, 40)>>.

STEP 4: the engine takes action 4 from the agenda, then it notifies to B (i.e. B is on the top of stack in agent $stack_4$) to execute action 4. Exploiting the instance browsing facility B identifies the agent C as executor of action 4. C satisfies the action annotations (i.e. she is a nurse and operates in SS1). When she accepts the assignments, she is put on agent $stack_4$ as executor.
agent $stack_4$: (<(1, 11, 40), (1, 12, 20), (1, 11, 50), (1, 12, 40)>, <(A,R),(B,D),(C,E)>).
Agenda: <4, <(1, 11, 40), (1, 12, 20), (1, 11, 50), (1, 12, 40)>>.

C executes action 4 at day 1 hour 12 minute 0 and its execution lasts 20 minutes. C identifies that the patient is treated for alcohol-related problems for the first time (i.e. the next action is action 5). Thus, action 5 is put in the agenda with its time window. Action 5 has not yet a responsible (line 9), thus the system asks A, the responsible of action 4 (i.e., the element at the bottom of the stack in agent $stack_4$) to find a responsible for such a new action. A must find a responsible who satisfies the action annotation (i.e. an agent who is a physician (R1) and works either in a Community medicine (C1) or in a SERT medicine (C2) or in-patient care (C3) or in a hospital ambulatory care (C4)). Using the instance browsing facilities, A finds a physician D, who works in the community medicine CM1, and asks her for the responsibility of action 5. D accepts the responsibility and, since action 7 belongs to the same responsibility continuity group (RG2), D is also nominated as responsible of both the actions in RG2.

agent $stack_5$: (<(2, 12, 00), (3, 12, 00), (2, 12, 30), (3, 13, 00), <(D,R)>);
agent $stack_7$:(<(5, 12, 30), (10, 13, 00), (5, 12, 40), (10, 13, 30), <(D,R)>).

Agenda: <5,< <(2, 12, 00), (3, 12, 00), (2, 12, 30), (3, 13, 00)>.

Then, the CIG execution goes on in a similar way.

7 Comparisons and Conclusions

Clinical guidelines encode the "best" evidence-based procedures to treat specific diseases, and are important in order to ensure the quality of healthcare treatments. However, they usually involve the activity of multiple healthcare agents, operating in different contexts, having different qualifications and competences, and playing different roles (e.g., responsible vs. executor of an action). In our research, we have extended META-GLARE, our system to manage CIGs, to provide users with a comprehensive and homogeneous set of facilities to manage the above phenomena. In particular, META-GLARE allows to distinguish among different "roles" that an agent can play in the management of a CIG action: *responsible*, *delegate* and *executor*. For each action, META-GLARE supports the specification of the *qualification* and of the specific *competences* that each responsible, delegate and executor must have to manage it, as well as the *context* in which the action has to be managed. We also support a set of *facilities for agent coordination* to grant that, at each time during the execution of a CIG on a specific patient, each action to be executed "next" has a "proper" responsible and an executor (and, possibly, some delegate), and is executed in a "proper" context. Additionally, we also support "continuity constraints", to grant that , whenever possible, actions requiring the same qualification and competences are managed by the same agent. Last, but not least, our approach deeply takes into account the temporal dimension. Temporal reasoning is exploited in order to indicate the time window when actions have to be executed. Such an information is provided to executors, but is also used to find responsibles and delegates (accepting to manage the actions in the indicated time window).

Notably, we have described our approach on the basis of META-GLARE, but it is worth stressing that our methodology is completely general and system\ application-independent (i.e. other CIG systems can be extended applying our approach).

The approach described in this paper is the result of a stream of research in the area of CIG agent coordination, that we started already in [25]. In such an approach, we only considered the *responsibles* of actions. We have proposed to annotate (*colour*, in the terminology in [25]) CIG actions with *context*, *qualification* and *competences* for the responsible, and we have proposed several facilities to acquire and query annotations, and to execute coloured CIGs. In [26] we have extended such an initial approach along two main directions:

(1) we have considered the distinction between responsibles, delegates, and executor, and proposed facilities to support the coordination of such different "roles";

(2) we have taken into account continuity constraints, representing them and considering them in the agent coordination process.

In this paper we propose a further step forward of our approach: to make our support fully applicable in the real practice, we also considered the ***temporal dimension***. Considering time is essential for the practical applicability of CIG systems, but deeply affects the complexity of the phenomena to be coped with. First of all, we had to enrich our approach with temporal reasoning capabilities, to evaluate the window of time in which the CIG actions have to be managed, according with the temporal constraints in the CIG, and with the time of execution of the previously executed CIG actions. Second, we had to properly integrate such temporal reasoning capabilities within the agent coordination process. Indeed, we suggest the agent coordination process is deeply affected by the temporal analysis, since the temporal windows of execution have to be evaluated not only when actions enter in the agenda (i.e., when actions have to be actually executed), but also at coordination time, when agents are requested to accept the responsibility, delegation, or execution of CIG actions, so that they can accept or reject also considering the estimated time when such actions will have to be carried on. To consider such fundamental issues, in this paper we have proposed the temporal reasoning facility (Section 4), and we have deeply extended the coordination facilities (Section 5), and the CIG executor engine (Section 5) presented in our previous work in [26].

While in the literature there is no other approach to CIGs that has provided a support considering all the aspects above, several approaches have faced at least few of them.

First of all, the treatment of the temporal dimension of CIG has attracted quite a lot of attention within the medical informatics community. For instance, GLIF [27, 28] deals both with temporal constraints on patient data elements and with duration constraints on actions and decisions. In PROforma [29], guidelines are modelled as plans, and each plan may define constraints on the accomplishment of tasks, as well as task duration and delays between tasks. Moreover, temporal constructs can also be used in order to specify the preconditions of actions. DILEMMA and PRESTIGE [30] model temporal constraints within conditions. EON [31] uses temporal expressions to allow the scheduling of guideline steps, and deals with duration constraints about activities. Moreover, by incorporating the RESUME system, it provides a powerful approach to cope with temporal abstraction. In EON, the Arden Syntax allows the representation of

delays between the triggering event and the activation of a Medical Logic Module (MDL), and between MDLs [32].

Within the AI community, it is widely recognized that temporal constraints are almost useless if they are not paired with adequate temporal reasoning mechanism to draw proper inferences from them [19], and many temporal reasoning approaches have been devised (see, e.g. the survey in [19]. Indeed, also the authors of this paper have been very active in the area [33–42]). However, the CIG community has paid only a (very) limited attention to temporal reasoning. This is quite surprising to us since, as discussed in [11] and in this paper, temporal reasoning is *necessary* to determine the time when next actions have to be executed (given the temporal constraints in the CIGs). Notable exceptions are represented by the approaches by Shahar [43] and by Duftschmid et al. [44].

In Shahar's approach, the goal of temporal reasoning is not to deal with temporal constraints (e.g., to check their consistency), but to find out proper temporal abstractions to data and properties. Therefore, temporal reasoning is not based on constraint propagation techniques, in fact, e.g., interpolation-based techniques and knowledge-based reasoning are used.

Miksch et al. have proposed a comprehensive approach based on the notion of temporal constraint propagation [43, 44]. In particular, in Miksch et al.'s approach, different types of temporal constraints – deriving from the scheduling constraints in the guideline, from the hierarchical decomposition of actions into their components and from the control-flow of actions in the guideline – are mapped onto an STP framework [20]. Temporal constraint propagation is used in order to (1) detect inconsistencies, and to (2) provide the minimal constraints between actions. In [44], there is also the claim that (3) such a method can be used by the guideline interpreter in order to assemble feasible time intervals for the execution of each guideline activity. Moreover, advanced visualization techniques are used in order to show users the results of temporal reasoning [45].

Notably, however, none of the above approaches have considered temporal reasoning in conjunction with the agent coordination phenomena discussed in this paper. Moreover, while several approaches to agent coordination have been proposed in the CIG literature (as widely discussed below), none of them takes into account the temporal reasoning facilities we propose in this paper.

In the formalism in [46], Fox's group has proposed an extension of the PROforma formalism to specify who will execute an action. However, the authors do not focus on agent coordination and action execution in different contexts. The goal of such a work is that of exploiting the pieces of information concerning agents for the sake of CIG contextualization (considering local human resources), and for flexibly adjusting them through delegation.

[47] have proposed a workflow-based solution to manage chronic patients over long time periods. Such an approach aims at allowing patients to obtain the necessary healthcare services by accessing different locations/organizations, which have to exchange/communicate health data whenever needed. The final goal of such an approach is to support cooperative work between different healthcare organizations; moreover, the authors model organizational knowledge (i.e. qualifications, resources etc.) by means of ontologies – as we do. However, their approach is not as flexible as ours,

because interactions between agents, and allocations of the next action to a specific responsible, are strictly predetermined by a contract and cannot be determined dynamically during the CIG execution. On the other hand, we allow the responsible of the current action to navigate the ontology, and to dynamically and freely identify the responsible and\or the executor of the next action on the basis of the available knowledge and constraints. Moreover, they do not support delegation, and do not take into account temporal reasoning issues.

[48] propose an ontology-driven execution of CIGs. Their approach relies on a *multi-agent* system, where agents in a medical center include not only persons, but also structures. In such an approach, the description of each CIG action includes a *hasResponsible* relation relating the action to one agent or to a set of agents. The main contribution of such an approach regards the delegation issue in a supervised fashion and the automation of the coordination of internal activities using a medical-organisational ontology. Since their approach is meant to be applied within a specific medical centre, it is focused on supporting interaction in a distributed environment, where the coordination between actors cannot be managed automatically.

Grando et al. [49] formalize cooperative work in CIG execution (but not distributed executions across different contexts). Their approach mainly focuses on the delegation of tasks to specific members of the working team, on the basis of their competences. Specific attention is devoted to cope with the assignment of the responsibility to enact services, and to exception handling. Specifically, they extend the design pattern framework introducing the types *role* (qualification in our approach) and *actor*, and a set of relations between key concepts. Since actors have roles and competences, they recall our notion of *agent.* Therefore, the authors take into account notions that are quite similar to our annotations. However, they do not consider the use of an ontology to formalize and standardize them. On the other hand, Grando et al. do not consider contexts: in this sense, their approach is more limited than ours, and not straightforwardly extendable to deal also with *distributed* (and not just *cooperative*) CIG executions. Moreover, temporal reasoning issues are not taken into account by Grando et al.

[50] have proposed a framework to support coordinated CIG execution by interdisciplinary healthcare teams. They distinguish among team managers, practitioner assistants, patient representatives (notably, such classes do not correspond either to our *qualification* classes, or to our roles). They have the concept of *capability*, which strictly resembles the notion of *competence* in our approach. They annotate actions, but their annotations are only related to the capability requirements. A main difference with respect of our approach is that Wilk et al. only consider CIG action *executors*, while they do not take into account *responsibles* and *delegates*. Moreover, they have the concept of team, i.e. a set of agents (defined using a hybrid approach) who manage the execution and are coordinated by a team manager, i.e. the responsible of execution for the whole CIG. The identification of executor of an action is not as general as the one in our approach: only the team manager can identify the executors and she has to take into account the agents in her team first. Only in the case that there are not any suitable and available agents in the team, she can search an external agent to execute the action and can add it to the team. Additionally, they do not cope with the notion of context of execution, as well as with temporal reasoning.

In summary, despite a huge interest of the Medical Informatics community in the CIG agent coordination phenomena, no approach in the literature has been devised until now coping with all the different issue covered by the approach we have proposed in this paper. In particular, to the best of our knowledge, our approach is the first CIG approach to agent coordination facing the temporal dimension through temporal reasoning, to determine the execution time of CIG actions.

Acknowledgments

The authors are very grateful to Prof. Gianpaolo Molino and Dr. Mauro Torchio of Azienda Ospedaliera San Giovanni Battista in Turin (one of the largest hospitals in Italy) for their constant support, and for their help in the definition of the case study.

References

1. Committee to Advise the Public Health Service on Clinical Practice Guidelines, Institute of Medicine: Clinical practice guidelines directions for a new program. National Academy Press, Washington, D.C. (1990).
2. Fridsma, D.B.: Special Issue on Workflow Management and Clinical Guidelines. J. Am. Med. Inform. Assoc. 22, 1–80 (2001).
3. Gordon, C., Christensen, J.P. eds: Health telematics for clinical guidelines and protocols. IOS Press, Amsterdam ; Washington, D.C (1995).
4. Peleg, M.: Computer-interpretable clinical guidelines: A methodological review. J. Biomed. Inform. 46, 744–763 (2013).
5. Terenziani, P., Montani, S., Bottrighi, A., Torchio, M., Molino, G.: Supporting physicians in taking decisions in clinical guidelines: the GLARE" what if" facility. Proc. AMIA Symp. 772 (2002).
6. Montani, S., Terenziani, P.: Exploiting decision theory concepts within clinical guideline systems: Toward a general approach. Int J Intell Syst. 21, 585–599 (2006).
7. Isern, D., Moreno, A.: A systematic literature review of agents applied in healthcare. J. Med. Syst. 40, 43 (2016).
8. Bottrighi, A., Terenziani, P.: META-GLARE: A meta-system for defining your own computer interpretable guideline system - Architecture and acquisition. Artif. Intell. Med. 72, 22–41 (2016).
9. Terenziani, P., Montani, S., Bottrighi, A., Molino, G., Torchio, M.: Applying artificial intelligence to clinical guidelines: the GLARE approach. Stud. Health Technol. Inform. 139, 273–282 (2008).
10. Scottish Intercollegiate Guidelines Network: Management of harmful drinking and alcohol dependence in primary care, https://lx.iriss.org.uk/sites/default/files/resources/sign74.pdf.
11. Anselma, L., Terenziani, P., Montani, S., Bottrighi, A.: Towards a comprehensive treatment of repetitions, periodicity and temporal constraints in clinical guidelines. Artif. Intell. Med. 38, 171–195 (2006).
12. Bottrighi, A., Piovesan, L., Terenziani, P.: Supporting Multiple Agents in the Execution of Clinical Guidelines. In: Zwiggelaar, R., Gamboa, H., Fred, A.L.N., and Badia, S.B. i (eds.) Proceedings of the 11th International Joint Conference on Biomedical Engineering Systems

and Technologies (BIOSTEC 2018) - Volume 5: HEALTHINF. pp. 208–219. SciTePress (2018).
13. Terenziani, P., Montani, S., Bottrighi, A., Torchio, M., Molino, G., Correndo, G.: A context-adaptable approach to clinical guidelines. Stud. Health Technol. Inform. 107, 169–173 (2004).
14. Montani, S., Terenziani, P., Bottrighi, A.: Exploiting decision theory for supporting therapy selection in computerized clinical guidelines. In: Conference on Artificial Intelligence in Medicine in Europe. pp. 136–140. Springer (2005).
15. Bottrighi, A., Giordano, L., Molino, G., Montani, S., Terenziani, P., Torchio, M.: Adopting model checking techniques for clinical guidelines verification. Artif. Intell. Med. 48, 1–19 (2010).
16. Holzmann, G.: SPIN model checker, the: primer and reference manual. Addison-Wesley Professional (2003).
17. Piovesan, L., Molino, G., Terenziani, P.: Supporting Physicians in the Detection of the Interactions between Treatments of Co-Morbid Patients. In: Healthcare Informatics and Analytics: Emerging Issues and Trends. pp. 165–193. IGI Global (2014).
18. Piovesan, L., Terenziani, P., Molino, G.: GLARE-SSCPM: an Intelligent System to Support the Treatment of Comorbid Patients. IEEE Intell. Syst. In press (2018).
19. Vila, L.: A Survey on Temporal Reasoning in Artificial Intelligence. AI Commun. 7, 4–28 (1994).
20. Dechter, R., Meiri, I., Pearl, J.: Temporal Constraint Networks. Artif Intell. 49, 61–95 (1991).
21. Brusoni, V., Console, L., Terenziani, P., Pernid, B.: Later: managing temporal information efficiently. IEEE Expert. 12, 56–64 (1997).
22. Meiri, I.: Combining qualitative and quantitative constraints in temporal reasoning. In: Proceedings of the ninth National conference on Artificial intelligence-Volume 1. pp. 260–267. AAAI Press (1991).
23. Isern, D., Moreno, A.: Computer-based execution of clinical guidelines: a review. Int. J. Med. Inf. 77, 787–808 (2008).
24. Terenziani, P., Bottrighi, A., Rubrichi, S.: META-GLARE: a meta-system for defining your own CIG system: Architecture and Acquisition. In: 6th International Workshop on Knowledge Representation for Health Care. pp. 92–107 (2014).
25. Bottrighi, A., Molino, G., Montani, S., Terenziani, P., Torchio, M.: Supporting a distributed execution of clinical guidelines. Comput. Methods Programs Biomed. 112, 200–210 (2013).
25. Bottrighi, A., Piovesan, L., Terenziani, P.: Supporting Multiple Agents in the Execution of Clinical Guidelines. In: Proc. of the 11th International Joint Conference on Biomedical Engineering Systems and Technologies (BIOSTEC 2018) - Volume 5: HEALTHINF. pp. 208–219. SciTePress (2018).
27. Ohno-Machado, L., Gennari, J.H., Murphy, S.N., Jain, N.L., Tu, S.W., Oliver, D.E., Pattison-Gordon, E., Greenes, R.A., Shortliffe, E.H., Barnett, G.O.: The guideline interchange format: a model for representing guidelines. J. Am. Med. Inform. Assoc. 5, 357–372 (1998).
28. Peleg, M., Boxwala, A.A., Ogunyemi, O., Zeng, Q., Tu, S., Lacson, R., Bernstam, E., Ash, N., Mork, P., Ohno-Machado, L., others: GLIF3: the evolution of a guideline representation format. In: Proceedings of the AMIA Symposium. p. 645. American Medical Informatics Association (2000).

29. Fox, J., Johns, N., Rahmanzadeh, A.: Disseminating medical knowledge: the PROforma approach. Artif. Intell. Med. 14, 157–182 (1998).
30. Gordon, C., Herbert, I., Johnson, P.: Knowledge representation and clinical practice guidelines: the DILEMMA and PRESTIGE projects. Stud. Health Technol. Inform. 511–515 (1996).
31. Musen, M.A., Tu, S.W., Das, A.K., Shahar, Y.: EON: A component-based approach to automation of protocol-directed therapy. J. Am. Med. Inform. Assoc. 3, 367–388 (1996).
32. Sherman, E.H., Hripcsak, G., Starren, J., Jenders, R.A., Clayton, P.: Using intermediate states to improve the ability of the Arden Syntax to implement care plans and reuse knowledge. In: Proceedings of the Annual Symposium on Computer Application in Medical Care. p. 238. American Medical Informatics Association (1995).
33. Terenziani, P., Carlini, C., Montani, S.: Towards a comprehensive treatment of temporal constraints in clinical guidelines. In: Proceedings Ninth International Symposium on Temporal Representation and Reasoning. pp. 20–27 (2002).
34. Anselma, L., Bottrighi, A., Montani, S., Terenziani, P.: Extending BCDM to Cope with Proposals and Evaluations of Updates. IEEE Trans Knowl Data Eng. 25, 556–570 (2013).
35. Anselma, L., Piovesan, L., Terenziani, P.: A 1NF temporal relational model and algebra coping with valid-time temporal indeterminacy. J. Intell. Inf. Syst. 47, 345–374 (2016).
36. Anselma, L., Terenziani, P., Snodgrass, R.T.: Valid-Time Indeterminacy in Temporal Relational Databases: Semantics and Representations. IEEE Trans. Knowl. Data Eng. 25, 2880–2894 (2013).
37. Anselma, L., Bottrighi, A., Montani, S., Terenziani, P.: Managing proposals and evaluations of updates to medical knowledge: theory and applications. J. Biomed. Inform. 46, 363–376 (2013).
38. Anselma, L., Stantic, B., Terenziani, P., Sattar, A.: Querying now-relative data. J. Intell. Inf. Syst. 41, 285–311 (2013).
39. Piovesan, L., Anselma, L., Terenziani, P.: Temporal detection of guideline interactions. In: HEALTHINF 2015 - 8th International Conference on Health Informatics, Proceedings; Part of 8th International Joint Conference on Biomedical Engineering Systems and Technologies, BIOSTEC 2015. pp. 40–50 (2015).
40. Anselma, L., Piovesan, L., Terenziani, P.: Temporal detection and analysis of guideline interactions. Artif. Intell. Med. 76, 40–62 (2017).
41. Anselma, L., Mazzei, A., De Michieli, F.: An artificial intelligence framework for compensating transgressions and its application to diet management. J. Biomed. Inform. 68, 58–70 (2017).
42. Anselma, L., Piovesan, L., Sattar, A., Stantic, B., Terenziani, P.: A General Approach to Represent and Query Now-Relative Medical Data in Relational Databases. In: Artificial Intelligence in Medicine - 15th Conference on Artificial Intelligence in Medicine, AIME 2015, Pavia, Italy, June 17-20, 2015. Proceedings. pp. 327–331 (2015).
43. Shahar, Y., Miksch, S., Johnson, P.: The Asgaard project: a task-specific framework for the application and critiquing of time-oriented clinical guidelines. Artif. Intell. Med. 14, 29–51 (1998).
44. Duftschmid, G., Miksch, S., Gall, W.: Verification of temporal scheduling constraints in clinical practice guidelines. Artif. Intell. Med. 25, 93–121 (2002).

45. Kosara, R., Miksch, S.: Metaphors of movement: a visualization and user interface for time-oriented, skeletal plans. Artif. Intell. Med. 22, 111–131 (2001).
46. Sutton, D.R., Fox, J.: The syntax and semantics of the PRO forma guideline modeling language. J. Am. Med. Inform. Assoc. 10, 433–443 (2003).
47. Leonardi, G., Panzarasa, S., Quaglini, S., Stefanelli, M., Van der Aalst, W.M.: Interacting agents through a web-based health serviceflow management system. J. Biomed. Inform. 40, 486–499 (2007).
48. Sánchez, D., Isern, D., Rodríguez-Rozas, Á., Moreno, A.: Agent-based platform to support the execution of parallel tasks. Expert Syst. Appl. 38, 6644–6656 (2011).
49. Grando, A., Peleg, M., Glasspool, D.: Goal-based design pattern for delegation of work in health care teams. In: MedInfo. pp. 299–303 (2010).
50. Wilk, S., Astaraky, D., Michalowski, W., Amyot, D., Li, R., Kuziemsky, C., Andreev, P.: MET4: supporting workflow execution for interdisciplinary healthcare teams. In: International Conference on Business Process Management. pp. 40–52. Springer (2014).