A General Framework for the Distributed Management of Exceptions and Comorbidities

Alessio Bottrighi, Luca Piovesan and Paolo Terenziani
DISIT, Computer Science Institute, Università del Piemonte Orientale, Italy

Keywords: Computer Interpretable Guideline (CIG), Concurrent Execution of Multiple CIGs, CIG-exception Handler, System Architecture.

Abstract: In the last decades, many different computer-assisted management systems for Computer Interpretable Guidelines (CIGs) have been developed. While CIGs propose a "standard" evidence-based treatments of "typical" patients, exceptions may arise, as well the need to cope with comorbidities. The treatment of deviation from "standard" execution has attracted a lot of attention in the recent literature, but the approaches proposed are focused on the treatment either of exceptions or of comorbities. However, this is a clear limitation, since during a CIG execution, both these issues can occur. In this paper, we propose the first approach which supports the integrated treatment of both exceptions and comorbidities. To achieve such a goal, we propose a modular client-server architecture supporting the concurrent execution of multiple guidelines. The architecture proposed has been designed as a further layer building upon "traditional" execution engines for a single CIG. Thus, our methodology is general and can be used to extend the CIG systems in the literature. Finally, we describe our approach in action on a case study, in which a comorbid patient is treated for Peptic Ulcer and for deep Venous Thrombosis and, during the treatment, she manifests a heart failure.

1 INTRODUCTION

Clinical Practice Guidelines (CPGs) represent the current understanding of the best clinical practice. CPGs are gaining a major role to improve the quality and to reduce the cost of health care. The ICT technology can further enhance the impact of CPGs. Many different systems have been developed to manage Computer Interpretable clinical practice Guidelines (CIGs for short). Such approaches are characterized by a specifc formalism to represent CPGs. CIG formalisms are usually based on a Task Network Model: a (hierarchical) model of the CPG control flow as a network of specific tasks. Such formalisms are "formal" and allow one to unambiguously represent guideline procedures and recommendations. Besides supporting formal languages to acquire and represent CPGs, CIG systems usually also provide execution engines that allow user physicians to "instantiate" general guidelines on specific patients: by accessing the patient clinical data, the execution engine shows to the user physicians only those paths of actions that are applicable to the patient at hand. In such a way, they provide patient-oriented recommendations to physicians, allowing them to fulfill the gap between the CPG generality and the specificity of the patient at hand. A survey and/or a comparative analysis of these systems is outside the goals of this paper, but a comparison of Asbru, EON, GLIF, Guide, PROforma, PRODIGY can be found in (Peleg et al., 2003). Bottrighi et al. (Bottrighi et al., 2009) extends it to consider also GLARE and GPROVE. (Peleg, 2013; Anselma et al., 2015) are recent surveys of the state-of-the-art.

One of the main goals of CPGs and CIGs is to capture medical evidence and to put it into practice. However, from one side, evidence is essentially a form of statistical knowledge, and it is used to capture the generalities of classes of patients, rather than the peculiarities of a specific patient. From the other side, demanding to expert committees the elicitation of all possible executions of a CPG on any possible specific patient in any possible clinical condition is an infeasible task. Thus, several conditions are usually implicitly assumed by experts building a CPG: (i) ideal patients, i.e., patients that have *only* the disease considered in the CPG (thus excluding the concurrent application of more than one CIG), and (ii) "statistically relevant" patients not presenting rare peculiarities/side-effects; (iii) ideal context of execution, so that all necessary resources are available. However, when a specific physician applies a given CIG to a specific patient, unexpected conditions may show up. Such si-

Bottrighi, A., Piovesan, L. and Terenziani, P.
A General Framework for the Distributed Management of Exceptions and Comorbidities.
DOI: 10.5220/0006552800660076
In *Proceedings of the 11th International Joint Conference on Biomedical Engineering Systems and Technologies (BIOSTEC 2018) - Volume 5: HEALTHINF*, pages 66-76
ISBN: 978-989-758-281-3

tuations are unexpected, and, as such, cannot be specified a priory in the CIGs. However, especially in case of unexpected life threatening problems, the physician must start soon to cope with the new problems (possibly suspending or ending the "standard" execution of the current CIG, or concurrently with it). Such problems have been usually indicated with the term "*exceptions*" within the CIG community, since they are exceptions with respect to the "standard" execution of a CIG.

Another challenging problem that might involve deviations from the "standard" execution of a CIG is the treatment of comorbid patients. The problem is that, by definition, CIGs address specific clinical circumstances (i.e., specific pathologies), and, unfortunately, in comorbid patients the treatments of single pathologies may dangerously interact with each other. Also, the approach of proposing an ad-hoc "combined" CIG to cope with each possible comorbidity does not scale up (Michalowski et al., 2013). For these reasons, new methodologies have been recently introduced to manage multiple CIGs on comorbid patients (see, e.g., the survey in (Fraccaro et al., 2015)).

1.1 Related Work

Within the CIG community, the "exceptions" and comorbidities have been always managed in isolation.

Several frameworks have been already proposed to cope with "exceptions" (see, e.g. (Fox et al., 1998; Leonardi et al., 2012; Tu and Musen, 1999; Grando et al., 2010; Quaglini et al., 2001; Peleg et al., 2009)). In most of such approaches, the "standard" executor of a CIG is extended with some mechanism to trigger exceptions (on the basis of the patient's data) and to activate their treatment, synchronizing such a treatment with the execution of the current CIG. Different mechanisms of synchronization have been proposed.

On the other hand, a range of different technical solutions have been proposed to cope with comorbidities, spanning from the use of constraint logic programming (Michalowski et al., 2013; Wilk et al., 2017) to answer set programming (Merhej et al., 2016), from rules (López-Vallverdú et al., 2013) to agents (Sánchez-Garzón et al., 2013). Notably, some of such approaches focus on the automatic generation of a unique "merged" CIG avoiding the undesired CIG interactions (consider, e.g. (López-Vallverdú et al., 2013; Sánchez-Garzón et al., 2013)). In such a way, a "standard" CIG executor can be used to enact the "merged" CIG. However, in the clinical practice, (1) there are usually different ways to manage interactions, and physicians want and *must* (for ethical reasons) be the protagonists of such a decision. And, more importantly for the current work, (2) though in clinical practice the interactions between CIGs are managed, physicians do not look at the solution as a single "merged" CIG: they still look at the treatment of a comorbid patient as the concurrent execution of multiple CIGs.

In conclusion, there are several approaches managing either "exceptions" or comorbidities, but no one provides a integrated support of them. This is a major and clear limitation to their application in practice, also given the occurrence frequency of the above phenomena. For instance, several studies demonstrate a prevalence of comorbidities on an average of 25% of the population, ranging from 10% (in younger people) to 78% (in older people) (van den Akker et al., 1998; Barnett et al., 2012). Thus, we aim at fulfilling the gap between the support provided by the current CIG systems and real world requirements.

1.2 Goals and Original Contributions

Until now, within the CIG literature, exceptions and comorbidities have been treated as ***separate*** phenomena[1], so that current approaches cope either with exceptions, or with comorbidities. This is a clear limitation of the state of the art, since both phenomena may co-occur on specific patients. In this paper, we first propose a CIG approach facing both phenomena, thus overcoming such a relevant limitation of the current literature. Additionally, unlike (López-Vallverdú et al., 2013; Sánchez-Garzón et al., 2013), we aim at maintaining the distinction between CIGs, even in the case of comorbid patients.

Our goal is to propose the first general framework that copes with **both** *exceptions* and *comorbidities*. The starting point of our approach is that the management of patients affected by multiple problems requires

(i) A support for the ***concurrent*** and ***distributed*** (i.e., carried on by different agents) ***execution of CIGs***, and to synchronize them. In the case of comorbid patients, it will support the execution of one CIG for each one of the patient's diseases; in the case of an "exception", it will support the execution of the original CIG plus the plan

[1]This choice is, in our opinion, quite surprising, since there does not seem to be a clear cut between the two phenomena. Just as one prototypical example, in (Leonardi et al., 2012) heart failure is considered as an "exception" for a patient treated with a CIG for trauma. But, when a patient with a trauma manifests a heart failure, s/he becomes a comorbid patient, and attention must be paid to avoid dangerous interactions between the treatment (CIG) for the trauma and the treatment (CIG) for the hearth failure.

(which can be formalized as a CIG) to manage the exception.

(ii) A support for ***detecting*** the possible ***interactions*** between such concurrent CIGs, and for ***managing*** them (avoiding dangerous interactions)

(iii) A support for ***detecting new patient's problems*** (i.e., changes in the status of the patient that require new treatments – thus, new CIGs).

In the rest of the paper, we propose the first CIG framework providing all such supports in an integrated way. While (1)-(3) are the main goals of our approach, we also take into account a further, more "technical" goal. The specialized literature proposes several "consolidated" execution engines, to execute a CIG for a specific patient. Many of such approaches (including GLARE and its recent extension META-GLARE (Terenziani et al., 2014; Bottrighi and Terenziani, 2016) have achieved good results, providing physicians with friendly environments to execute CIGs. We think that it would really be a pity to waste such an amount of good work, building from scratch a new concurrent execution engine. Thus, an additional main goal of our approach is that of devising a ***modular*** approach for the concurrent execution, in which the execution engine of a CIG in isolation is maintained, and it is extended and integrated in a general framework supporting synchronization and concurrency. Notably, although we are building our framework on top of META-GLARE (in the sense that each "Exec" module - see Fig.3 below - is an instantiation of META-GLARE execution module), our methodology is general and can be adapted for similar CIG systems (such as, e.g., (Fox et al., 1998; Shahar et al., 1998)).

Furthermore, notice that the framework we have developed also supports the fact that multiple healthcare agents may be involved in the execution of each single CIG for a specific patient. We cope with multiple agents using the methodology in (Bottrighi et al., 2013) and generalizing it to the context of multiple CIGs. For the sake of brevity, such a topic is no further discussed within this paper.

Notably, the main contribution of this paper is the definition of (the architecture of) a system-independent framework for the distributed management of exceptions and comorbidities. We are currently implementing a prototype of the framework on top of META-GLARE and of its module for managing the interactions (Anselma et al., 2017).

2 A GENERAL VIEW OF THE BEHAVIOR OF OUR FRAMEWORK

While the architecture of our framework is proposed in Section 3, here we informally discuss the basic data/knowledge sources (ovals in Fig. 1) managed by our framework, and its general behavior (see Fig. 1). First, though in the literature (see, e.g., (Fox et al., 1998; Grando et al., 2010; Leonardi et al., 2012; Peleg et al., 2009; Quaglini et al., 2001; Tu and Musen, 1999)) different types of exceptions have been identified, for the sake of brevity in this paper we focus only on the most "common" ones, i.e., on the exceptions arising because of unexpected changes in the status of the patient, requiring a (new) treatment (i.e., with *CIG-independent patient-exceptions*, in the terminology in (Leonardi et al., 2012)).

Second, it is important to clarify that there is a main "practical" difference between the management of exceptions and the one of interactions (e.g., the interactions that may arise between actions of different CIGs operating on the same patient). Exceptions cannot be avoided: the status of the patient has already changed, and an exception arises because of such a change (i.e., it is *triggered* by the new status of the patient). Moreover, of course, the exception must be managed, usually by starting a new treatment (which, indeed, may be represented by a CIG) for it.

In our approach, the treatment of exceptions is modeled by a KB (called "**Exception KB**"; see Fig. 1) consisting of triggering rules of the form $\langle$Condition, Manag$\rangle$, where "*Condition*" indicates a Boolean condition (called "triggering" conditions) on the ***status of the patient***, and "*Manag*" represents the actions to cope with such a condition (representable as a new CIG) plus constraints about how such new actions have to be synchronized with (the execution of) the current CIG(s). Notably, in our approach, the "Exception KB" is static, in the sense that the rules it contains are patient-independent, and are permanently stored.

On the contrary, undesired interactions should be detected a-priori and avoided (through some management operation). META-GLARE, for instance, is provided with a framework (see the *Interaction Analysis* module in Fig. 1; rectangles represent computational modules) for (i) supporting physicians in the focusing on specific parts of the CIGs[2] (Piovesan et al.,

[2]CIGs may consist of hundreds of actions and/or alternative paths. An extensive check of all interactions could provide a combinatorial number of cases, most of which are not interesting for the patient at hand. Physician-driven focusing is an essential step to avoid an unnecessary com-

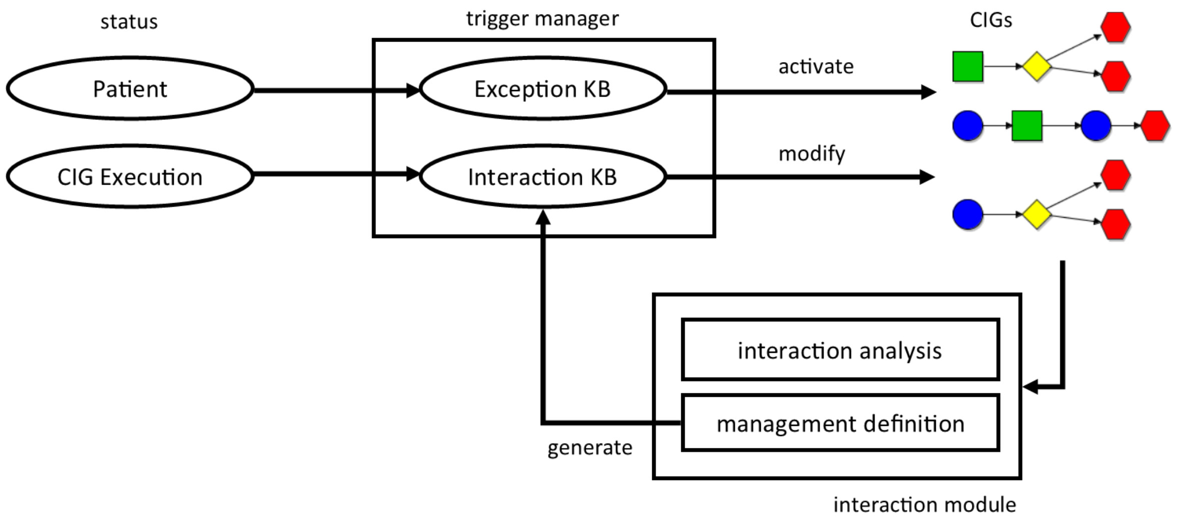

Figure 1: Graphical representation of our treatment of exceptions and interactions.

2015), (ii) automatically detecting (based on CIG-independent ontological knowledge) the interactions between the focused action (Piovesan et al., 2014), as well as a suite of management options (derived from the medical literature) to manage them (e.g., avoiding them by delaying some actions, or managing them through local modifications of the involved CIGs (Piovesan and Terenziani, 2015); see the *Management Definition* module in Fig. 1). Although the analysis of interaction should be performed *a priori*, generally the management option chosen by physicians has not to be enacted soon. Indeed, it had to be enacted only if and when, during the execution of the CIGs, the conditions identifying the onset of the interactions arise[3]. As a consequence, the treatment of interactions may be modelled by a KB (called "**Interaction KB**"; see Fig. 1) containing ⟨Condition, Manag⟩ pairs. However, differently from the rules for exceptions discussed above, here:

(i) triggering conditions have as input the ***status of execution*** of the ***CIG***s

(ii) "Manag" indicates the operations to implement the management option chosen to cope with an interaction (e.g., operations to locally modify a CIG)

The "Interaction KB" is ***dynamic***: the system dynamically add rules into it whenever a new interaction has been detected and a management for it has been chosen, and it deletes such rules when they are not useful any more. Indeed, there would be no reason to permanently store such rules, since they are specific to the execution of a set of CIGs for a specific patient.

In Fig. 1, we show that a unique manager, the "**Trigger Manager**" can uniformly operate on both types of rules. Fig. 1 also shows that the output of the execution of an "Exception" rule may be the activation of a new CIG, while the result of the execution of an "Interaction" rule may usually be a modification of some CIGs. It also shows that, when multiple CIGs are active, the *Interaction Analysis* module may be used to analyse possible interactions, and the *Management Definition* module can support the management of such interactions. The output of the *Management Definition* module is a new ⟨Condition, Manag⟩ rule, dynamically stored in the Interaction KB.

While Fig. 1 graphically illustrates the behaviour of our framework, in Section 3 we discuss the architecture we have identified to achieve it, and considering the fact that the management of exceptions and interactions may involve forms of synchronizations between the different CIGs (possible synchronizations between CIGs have been omitted, for the sake of clarity, from Fig. 1).

binatorial explosion of the computation and of the number of the identified interactions.

[3]As an example, a possible undesired interaction between the actions Act1 in CIG$_A$ and Act2 in CIG$_B$ can be detected and physician can choose to manage it via the substitution of Act2 with a set of actions achieving the goal of Act2, but non-interacting with Act1. However, such a substitution must be performed only in case the execution of the two CIGs enforces the execution of both Act1 and Act2 (at times such that their effects may overlap in time). Indeed, if in CIG$_A$ a path of actions not including Act1 has been selected for execution, there is no need to substitute Act2.

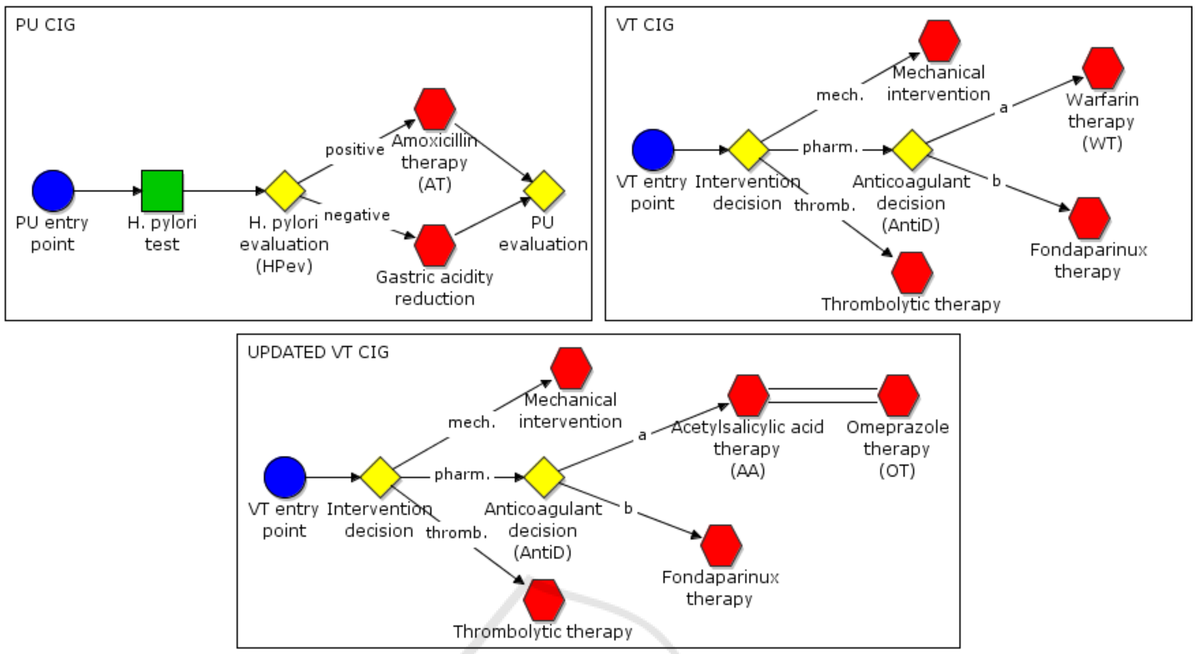

Figure 2: Part of PU and VT original CIGs (above) and the updated version of VT after the managing of the interaction (below).

2.1 Case Study

In this section, we present a “synthesized” case study, which has been created with the help of some physicians of Azienda Ospedaliera “San Giovanni Battista” in Turin in order to be able to exemplify the main features of our approach. We consider a comorbid patient, who is treated for Peptic Ulcer (PU) and for deep Venous Thrombosis (VT) and has a heart failure (i.e., an “exception” arises) during the execution of these CIGs. The two diseases are managed by two specific CIGs (the upper part of Fig. 2 shows simplified parts of the CIGs). Besides the CIGs, additional medical knowledge is available, including the trigger for exceptions. In our example, among them, we consider the exception for heart failure (notably, in this context, heart failure can be considered an exception: it is not statistically recurrent in PU and VT, thus its treatment is not contained into the original CIGs).

In our example, the CIGs for PU and VT are executed concurrently by two different physicians: Physician$_1$ manages PU, and Physician$_2$ manages VT. We consider a sample working section articulated as follows.

Step 1. We suppose that Physician$_1$, at a certain point of execution of PU and VT (e.g., at the beginning), decides to analyze the possible interactions between the two CIGs. To do so, Physician$_1$ exploits the Interaction Manager module (see Fig. 1) focusing on relevant parts of PU and of VT, and the Interaction Manager module detects an interaction between warfarin therapy (WT) in VT and amoxicillin therapy (AT) in PU.

Step 2. Physician$_1$ chooses to manage such an interaction replacing the action WT with an alternative plan, having the same goal. For instance, the new therapeutic plan may be the combination of acetylsalicylic acid (AA) therapy and omeprazole (OT) therapy. As we will see in the Section 4, the Interaction Manager module creates a trigger rule to implement such a management (if/when required).

Step 3. Physician$_1$ and Physician$_2$ go on with the independent executions of the CIGs. We suppose that in PU “PU start”, “H.Pylori test”, “HPev” with exit “positive” has been executed; in the meanwhile, in VT “VT start”, “intervention decision”, with exit “pharm”, and “AntiD” with exit “a” has been executed.

Step 4. At this point, the chosen management of the interaction is required, and is executed (i.e. the trigger rule created at step 2 above is executed, thus modifying the CIG as shown in the lower part of Fig. 2).

Step 5. The execution continues on the modified CIGs.

Step 6. To exemplify all the main features of our approach, we further suppose that at this point the patient has a heart failure, and we show how our framework supports its treatment.

3 ARCHITECTURE FOR THE CONCURRENT EXECUTION OF MULTIPLE CIGS

3.1 The Architecture

Our approach is based on the client-server model. This choice is motivated by the need (i) to support a distributed execution of the patient treatments, since different CIGs can be managed by different physicians (i.e. each physician needs a client to manage her CIGs) and (ii) to have a global vision of patient treatments, and to "synchronize" them (such a vision will be stored and managed in the server). Notably, from an abstract point of view our approach can be described as an agent based system (i.e. each module can be seen as an agent).

For sake of simplicity, in this paper we assume that all the CIGs are related to the same patient. The extension to cope with more than one patient is obvious. We propose a ***server*** model (see Fig. 3) composed by the following modules:

- a ***"General Manager"*** (in the middle of the "Server" in Fig.3): it maintains the global vision of the patient and of her treatments (i.e. global data structures). It interacts with the other modules to update such a vision and to synchronize them (the functionalities of such a module are described in more detail in subsection 3.2);
- *the "Executor Modules"*("**Exec CIG1**", "**Exec CIG2**", and "**Exec CIG3**", in the left part of the "Server" in Fig.3; notably, there is one "Exec" module for each CIG under execution for the patient). Each Executor manages the execution of a CIG for a specific patient (Bottrighi et al., 2015) (see subsection 3.3);
- ***the "Interaction Manager"*** (top right part of the "Server" in Fig.3): it supports the study of the interactions between CIGs and defines how they should be managed (see subsection 3.4);
- the ***"Trigger Manager"*** (bottom right part of the "Server" in Fig.3)***:*** it manages the triggers in KBs (see subsection 3.5).

Notably, the architecture of our framework is open. It is possible to add the new modules to provide new facilities, by specifying their communication API (i.e. how they communicate with the other modules, the patient's DB and the client).

The **client** provides physicians with a GUI to support the execution of one or more CIGs (e.g. in Fig. 3 Client$_1$ allows to manage the execution of CIG$_1$ and CIG$_2$, while Client$_2$ supports the execution of CIG$_3$). Each *client* sends/receives messages to/from the *executor* module to manage the execution of the CIGs. Moreover, physicians can activate the *interaction* module to study possible interactions between two or more CIGs (e.g. in Fig. 3, Client$_1$ activates it).

3.2 The General Manager Module

The ***General Manager*** is the core of the system, since it supports the concurrent execution of CIGs on a given patient. To achieve such a goal, it manages the interplay between the other modules in the server by (i) sending/receiving messages, and (ii) maintaining two data structures to provide a "global vision" of the execution of the CIGs: (i) the ***graph of CIG dependencies*** and (ii) the ***yellow pages of CIGs***. Such data structures work as a *shared memory*, where all the modules have the read permissions, while the *General Manager* has also the write permission.

The ***graph of CIG dependencies*** has two components: *nodes* and *arcs*. Each *node* represents one CIG under execution (for the given patient). The *arcs* represent the dependencies between such CIGs. An arc starting from a node A and ending into a node B means that B must be *suspended* by the execution of A. Thus, the *graph* represents the synchronization between CIGs: CIGs without entering arcs are *active*, while CIGs reached by an arc are temporarily suspended. We provide a set of primitives to update the *graph*: creation/deletion of a node, creation/deletion of an arc.

The ***yellow pages of CIGs*** store all the instances of CIGs currently in execution. The operations provided to update the *yellow pages* are: add a CIG, remove a CIG, update a pharmacological dosage, update a temporal constraint, add a node to a CIG, add an arc to a CIG (assuming that CIGs are represented as a Task-Network Model).

The updates to the data structures are triggered by messages sent by the other modules in the server. A message represents a list of instructions expressed using the primitives described above. The ***General Manager*** manages messages as transactions, i.e. units of work performed in an atomic way. It performs all the updates required and then it notifies such updates to the modules to maintain the synchronization. The General Manager manages the message of updates using a FIFO policy.

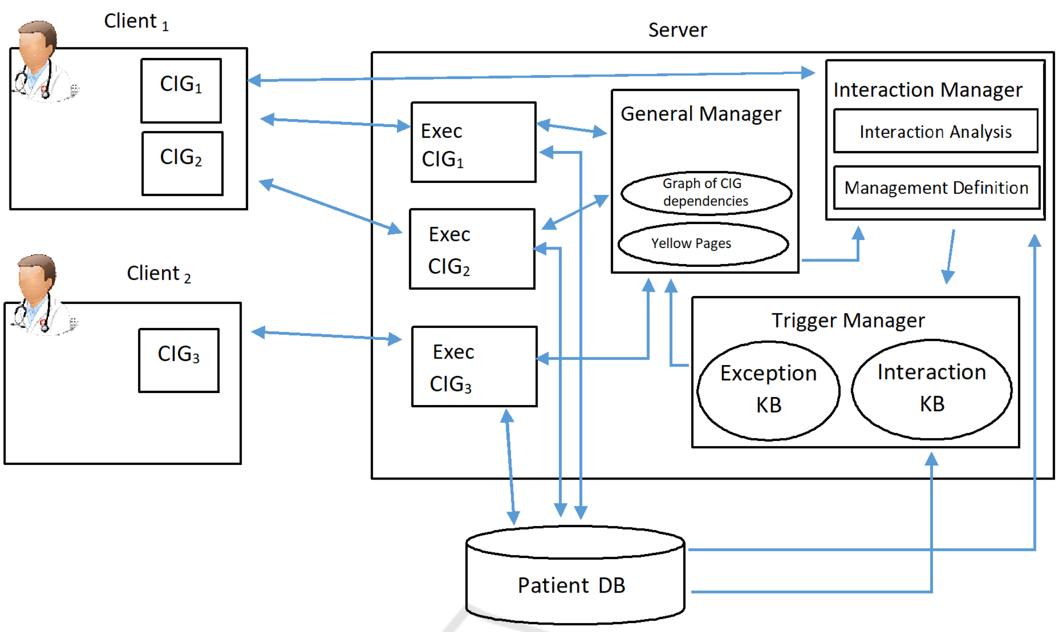

Figure 3: General Architecture of our framework. The arrows show the flow of information between the different modules.

3.3 The Executor Module

The ***Executor*** module manages the execution of a CIG instance for a specific patient. In our approach, there is an instance of Executor for each CIG under execution. The Executor of a CIG can be active or suspended depending on the current state (i.e. active/suspended) of the CIG represented in the *graph of dependencies*. Each instance of Executor reads (i) the instance of the CIG that it has to execute from the yellow pages, and (ii) the patient data from the *Patient DB*. The *Executor* interacts with a specific client to execute the current actions in the CIG. In case the CIG is terminated, the *Executor* sends a message to the *General Manager*, to remove the node (representing the CIG) from the graph, and remove the CIG from the *yellow pages*. Specifically, we use the executor of META-GLARE (Bottrighi and Terenziani, 2016), but our methodology is mostly system-independent, and it can be adapted for use any CIG executor (such as, e.g., (Fox et al., 1998)) or Asbru (Shahar et al., 1998)).

3.4 The Interaction Manager Module

The ***Interaction Manager module*** supports the detection and the definition of management for CIG interactions. It is composed by two modules (see Fig. 3): the ***Interaction Analysis*** and the ***Management Definition***. The *Interaction Analysis* module (see (Piovesan et al., 2014)) operates in two steps. First, it provides physicians with a navigation tool (operating at the different abstraction levels supported by the given CIGs) supporting the choice of a specific part (called "focus") of the CIGs, the part currently of interest for the treatment of the current patient. Second, it provides a knowledge-based tool that automatically detects all the possible interactions between the actions in the "focus". Moreover, this module has been recently extended with a set of facilities to *temporally* analyze interactions (Anselma et al., 2017), distinguishing among temporally certain, possible or impossible interactions and performing hypothetical reasoning. Once detected an interaction, the *Management Definition* module (Piovesan and Terenziani, 2015) supports physicians in the selection of a management, choosing among different *modalities* (i.e., the *management options* (Piovesan and Terenziani, 2015)). Notably, in our approach, managements are not applied immediately to CIGs, but through the creation of dynamic trigger rules (see the discussion in Section 2). The triggers have the form ⟨Condition, Manag⟩ where "Condition" indicates a Boolean condition on the execution of specific CIG action(s) or decision result(s), and "Manag" represents the actions to cope with such a situation. These actions can be described using a subset of primitives to operate on the global data structures (see Section 3.2). Such trigger rules are automatically generated by the by a specific component of the *Management Definition* module (the "Trigger Generator", not detailed in Fig.3 for the sake of brevity and clarity). The "Trigger Generator" takes as input from the other modules the detected interacting actions, and the management options chosen to manage such an interaction, plus additional parameters. The Trigger Generator consists of a set of parametric procedures, one for each management option, to automatically generate a trigger, on the basis of the input parameters. Then, the trigger is sent as a message to the *Trigger Manager*.

For instance in Section 4 we show the trigger created by the Interaction Manager module to manage the interaction between WT and AT (see section 2.1).

3.5 The Trigger Manager

The ***Trigger Manager*** module manages the triggers. To achieve such a goal, it has (i) to check whether the triggers stored in the *KB*s fire and then to notify that the management had to be applied and (ii) to maintain up-to-date the *Interaction KB*, since it is a dynamic KB (see details in Section 2).

To cope with (i), the *Trigger Manager* evaluates whether a rule in the KBs had be executed. The form of rules is ⟨*Condition, Manag*⟩ (see Section 2) and the *Trigger Manager* checks whether Condition is true or not (i.e. the patient status retrieved in the Patient DB or the execution status of the CIGs retrieved in the yellow pages satisfy Condition). If Condition is true, the *Trigger Manager* sends a message to the *General Manager*. Such a message contains *Manag* (i.e. the set of instructions to cope with the situation described in Condition).

To cope with (ii), the **Trigger Manager** adds a trigger to the *Interaction KB*, when it receives a message from the *Interaction Manager* module. Each message contains a trigger that has to be added. The *Trigger Manager* manages the messages using a FIFO policy.

The triggers in the *Interaction* KB are not permanent, since they are context and patient dependent. Thus, the *Interaction Manager* removes a trigger: (i) when it is used (in the case that it is not reusable, e.g. in the case it is applied to a repeatable part of the CIGs, it is removed when the repetitions of such a part is ended) or (ii) when one of the CIGs in its Condition ends.

4 OUR SYSTEM IN ACTION: MANAGING THE CASE STUDY

We describe how our framework works on the case study described in subsection 2.1. The patient is affected by both Peptic Ulcer (PU) and deep Venous Thrombosis (VT) and two CIGs are executed to treat such diseases. Two physicians are involved: Physician$_1$, managing PU, and Physician$_2$, managing VT. In our framework, each physician interacts with the system via a client: (1) Physician$_1$ uses Client$_{PU}$ to execute the CIG PU via the executor instance Executor$_{PU}$, and (2) Physician$_1$ uses Client$_{VT}$, to execute the CIG VT via the executor instance Executor$_{VT}$. Suppose that both physicians are managing the first action in the CIG (but this is not restrictive at all). In such a context, *the graph of dependencies* contains two independent nodes (one for PU and one for VT), while the *yellow pages* contain the current instances of the CIGs. The Interaction KB is empty and Exception KB contains the triggers to manage the exceptions. In our example, among the others, the trigger TR-HF (i.e. the trigger for heart failure) is stored in the Exception KB:

```
TR-HF:
(1) ⟨(Heart Failure = TRUE),
(2) (ADD_NODE HF-PLAN TO GRAPH;
(3) ADD HF-PLAN TO YELLOW PAGES;
(4) ADD_ARC from HF-PLAN to VT;
(5) ADD_ARC from HF-PLAN to PU;)⟩
```

In TR-HF, the Condition (line 1) captures that the patient has a heart failure, the Manag (lines 2-5) describes the instructions that must be executed to manage it. In short, (2)-(5) encode the commands to activate a new CIG "HF-PLAN" suspending the execution of VT and PU.

To analyze the possible interactions between the two CIGs, Physician$_1$ (through Client$_{PU}$) calls the *Interaction Manager* module and selects the relevant part of CIGs that the module has to analyse (i.e., the "focus"). The *Interaction Manager* identifies all the interaction between the actions in the "focus" via the *Interaction Analysis* module. In this specific example, the *Interaction Manager* module finds an interaction between warfarin therapy (WT) and amoxicillin therapy (AT). Such an interaction increases the anticoagulant effect of warfarin and raises the risk of bleedings. As described in subsection 2.1, Physician$_1$ decides to apply the *replanning management option* (Piovesan and Terenziani, 2015), substituting WT with an alternative new plan. Such a new plan is automatically generated by the *Management Definition* module (as described in (Bottrighi et al., 2016)). In our example, the new therapeutic plan is the combination of acetylsalicylic acid (AA) therapy and omeprazole (OT) therapy.

Then *the Trigger Generator* is invoked. It takes as input the PU and VT CIGs, the management option chosen *by* Physician$_1$ (i.e., the *replanning* option) and the new alternative plan, and (automatically) produces as output the trigger rule TR-WTAT described below. The Condition part of TR-WTAT represents the conditions under which the interaction can occur. In particular, in our example, AT and WT interacts in case (line 1): (i) the decision AntiD has been taken, having as result to execute the path "a" which contains WT, and (ii) either the decision HPev has been taken with result "positive" (i.e. choosing the path containing AT), or HPev has not been already executed (in this last situation, the system preventively applies the management, avoiding the cases in which decision HPev is taken with "positive" result only after WT has been executed, impeding the application of the management). Notably, such a condition is automatically built by the *Trigger Generator*, through a navigation throughout the PU and VT CIGs. The Manag part of TR-WTAT is automatically built by the *Trigger Generator* on the basis of the management option chosen by Physician$_1$ and the new alternative plan. Specifically, the Manag part of TR-WTAT prescribes to (line 2) remove WT, and (lines 3-4) to add AA, OT and (lines 5-6) the corresponding arcs in the CIG VT (the result of the execution of TR-WTAT is shown in Fig. 2).

```
TR-WTAT:
(1) ((Exec(AntiD)=a AND (Exec(HPev)=positive
    OR NOT Exec(HPev)),
(2) (remove action WT in VT;
(3) ADD_ACTION AA to VT;
(4) ADD_ACTION OT to VT;
(5) ADD_ARC in VT from AntiD to AA;
(6) ADD_ARC in VT from AA to OT;))
```

Then, the *Interaction Manager* module sends a message containing the TR-WTAT rule to the *Trigger Manager*.

As a consequence, the *Trigger Manager* adds it to the *Interaction KB* (see Fig. 2). Then, the two physicians can independently go on with the execution of the CIGs.

For instance, suppose that Physician$_1$ (through Client$_{PU}$) has executed the actions "PU start", "H.Pylori test", and "HPev", which results positive; in the meanwhile, Physician$_2$ (through client Client$_{VT}$) has executed "VT start", "intervention decision", with exit "pharm", and "AntiD" with exit "a". This situation triggers TR-WTAT (i.e. Condition in TR-WTAT is satisfied). Thus, the *Trigger Manager* sends a message to the *General Manager* containing the instruction to manage such an interaction (i.e. the Manag component in TR-WTAT, i.e. lines 2-6) and removes TR-WTAT from the *Interaction KB*, since it is not reusable during the patient treatment.

Then, the *General Manager* executes as a unique transaction the instructions in the message, updating the global vision. In our example, the instance of VT in the *yellow pages* is updated by replacing WT with the alternative plan (see lines 2-6 in TR-WTAT), as shown in the lower part of Fig. 2. Thus, the *General Manager* notifies to Executor$_{VT}$ that the instance of VT in the *yellow pages* has been updated. As consequence, Executor$_{VT}$ sends a message to Client$_{VT}$ to refresh the visualization of VT, and let Physician$_2$ go on with the execution of the updated CIG.

Moreover, let us suppose that, during the execution of such CIGs, the patient has a heart failure. As a consequence TR-HF is triggered by the *Trigger Manager*. Then the Trigger Manager sends a message to *General Manager* with the instructions to manage the heart failure (lines 2-5 in TR-HF). The *General Manager* executes these instructions. The first two instructions (lines 2-3 in TR-HF) generate (both in the graph of CIG dependencies and in the yellow pages of CIGs) the node corresponding to the CIG to treat heart failure. As a result of such a generation, our framework supports the search for a physician accepting the responsibility of executing the new CIG (following the approach in (Bottrighi et al., 2013)), and generates a new instance of Executor module to manage the Heart Failure CIG. The selected physician can manage the execution of the CIG trough a client. In case s/he is already executing a CIG for the specific patient, the Heart Failure CIG is added to its client, otherwise a new client is initialized for her/him. Moreover, the interpretation of lines 4-5 in TR-HF adds two (suspension) arcs in the graph of CIGs dependencies, then the *General Manager* notifies the suspension to Executor$_{VT}$ and to Executor$_{PU}$. Consequently, the two executors notify the suspension to the corresponding clients.

5 CONCLUSIONS

Traditional CIG execution engines provide physicians with consolidated support for the execution of a single CIG on a single patient. However, the treatment of "exceptions" and of comorbidities demands for more extended supports. Indeed, the management of such phenomena requires also a support for the concurrent execution of multiple CIGs on the same patient. The approaches proposed in the literature manage either "exceptions" or comorbidities, and do not provide facilities to cope with the coordination between such a concurrent executions, which is an essential issue.

In this paper, we provide the first homogeneous framework for the management of both "exceptions" and interactions dealing with the concurrent and coordinate execution of multiple CIGs. Our approach is modular, in that it adds a further layer building upon "traditional" execution engines for a single CIG. Though our framework is being built on top of META-GLARE, our methodology is general, and can be adapted for similar CIG systems (such as, e.g., PROforma (Fox et al., 1998) or Asbru (Shahar et al., 1998)).

We are currently implementing our approach using Java (Java-based prototypes of META-GLARE and its extensions to cope with comorbid patients are available). As soon as the implementation will be completed, we plan to develop an extensive experimentation of our framework, especially in the context of comorbidity treatment. Moreover, we plan to extend our approach to provide a more comprehensive support for distributed execution of CIGs to grant treatment continuity, contextualization, and responsibility assignment and delegation.

ACKNOWLEDGMENTS

This research is original and has a financial support of the Università del Piemonte Orientale.

REFERENCES

Anselma, L., Bottrighi, A., Hommersom, A., Terenziani, P., and Hunter, A. (2015). Supporting physicians and patients through recommendation: Guidelines and beyond. In Hommersom, A. and Lucas, P. J. F., editors, *Foundations of Biomedical Knowledge Representation - Methods and Applications*, volume 9521 of *Lecture Notes in Computer Science*, pages 281–286. Springer.

Anselma, L., Piovesan, L., and Terenziani, P. (2017). Temporal detection and analysis of guideline interactions. *Artificial Intelligence in Medicine*, 76:40–62.

Barnett, K., Mercer, S. W., Norbury, M., Watt, G., Wyke, S., and Guthrie, B. (2012). Epidemiology of multimorbidity and implications for health care, research, and medical education: a cross-sectional study. *The Lancet*, 380(9836):37–43.

Bottrighi, A., Chesani, F., Mello, P., Montali, M., Montani, S., Storari, S., and Terenziani, P. (2009). Analysis of the GLARE and GPROVE Approaches to Clinical Guidelines. In *KR4HC*, volume 5943 of *Lecture Notes in Computer Science*, pages 76–87. Springer.

Bottrighi, A., Leonardi, G., Piovesan, L., and Terenziani, P. (2016). Knowledge-Based Support to the Treatment of Exceptions in Computer Interpretable Clinical Guidelines:. *International Journal of Knowledge-Based Organizations*, 6(3):1–27.

Bottrighi, A., Molino, G., Montani, S., Terenziani, P., and Torchio, M. (2013). Supporting a distributed execution of clinical guidelines. *Computer Methods and Programs in Biomedicine*, 112(1):200–210.

Bottrighi, A., Rubrichi, S., and Terenziani, P. (2015). META-GLARE: A meta-engine for executing computer interpretable guidelines. In Riaño, D., Lenz, R., Miksch, S., Peleg, M., Reichert, M., and ten Teije, A., editors, *Knowledge Representation for Health Care - AIME 2015 International Joint Workshop, KR4HC/ProHealth 2015, Pavia, Italy, June 20, 2015, Revised Selected Papers*, volume 9485 of *Lecture Notes in Computer Science*, pages 37–50. Springer.

Bottrighi, A. and Terenziani, P. (2016). META-GLARE: A meta-system for defining your own computer interpretable guideline system—Architecture and acquisition. *Artificial Intelligence in Medicine*, 72:22–41.

Fox, J., Johns, N., and Rahmanzadeh, A. (1998). Disseminating medical knowledge: the PROforma approach. *Artificial Intelligence in Medicine*, 14(1-2):157–181.

Fraccaro, P., Arguello Castelerio, M., Ainsworth, J., and Buchan, I. (2015). Adoption of Clinical Decision Support in Multimorbidity: A Systematic Review. *JMIR Medical Informatics*, 3(1):e4.

Grando, A., Peleg, M., and Glasspool, D. (2010). A goal-oriented framework for specifying clinical guidelines and handling medical errors. *Journal of Biomedical Informatics*, 43(2):287–299.

Leonardi, G., Bottrighi, A., Galliani, G., Terenziani, P., Messina, A., and Corte, F. D. (2012). Exceptions Handling within GLARE Clinical Guideline Framework. In *AMIA*.

López-Vallverdú, J. A., Riaño, D., and Collado, A. (2013). Rule-Based Combination of Comorbid Treatments for Chronic Diseases Applied to Hypertension, Diabetes Mellitus and Heart Failure. In *Process Support and Knowledge Representation in Health Care*, volume 7738, pages 30–41. Springer Berlin Heidelberg, Berlin, Heidelberg.

Merhej, E., Schockaert, S., McKelvey, T. G., and De Cock, M. (2016). Generating conflict-free treatments for patients with comorbidity using ASP. In *KR4HC 2016*, Lecture Notes in Computer Science, pages 93–100.

Michalowski, M., Wilk, S., Michalowski, W., Lin, D., Farion, K., and Mohapatra, S. (2013). Using Constraint Logic Programming to Implement Iterative Actions and Numerical Measures during Mitigation of Concurrently Applied Clinical Practice Guidelines. In *Proceedings of AIME*, number 7885 in Lecture Notes in Computer Science, pages 17–22. Springer Berlin Heidelberg.

Peleg, M. (2013). Computer-interpretable clinical guidelines: A methodological review. *Journal of Biomedical Informatics*, 46(4):744–763.

Peleg, M., Somekh, J., and Dori, D. (2009). A methodology for eliciting and modeling exceptions. *Journal of Biomedical Informatics*, 42(4):736–747.

Peleg, M., Tu, S. W., Bury, J., Ciccarese, P., Fox, J., Greenes, R. A., Hall, R. W., Johnson, P. D., Jones, N., Kumar, A., Miksch, S., Quaglini, S., Seyfang, A., Shortliffe, E. H., and Stefanelli, M. (2003). Comparing Computer-interpretable Guideline Models: A Case-study Approach. *JAMIA*, 10(1):52–68.

Piovesan, L., Molino, G., and Terenziani, P. (2014). Supporting Physicians in the Detection of the Interactions between Treatments of Co-Morbid Patients. In *Healthcare Informatics and Analytics: Emerging Issues and Trends*, pages 165–193. IGI Global.

Piovesan, L., Molino, G., and Terenziani, P. (2015). Supporting Multi-Level User-Driven Detection of Guideline Interactions. In *Proceedings of the International Conference on Health Informatics (HEALTHINF-2015)*, pages 413–422. Scitepress.

Piovesan, L. and Terenziani, P. (2015). A Mixed-Initiative approach to the conciliation of Clinical Guidelines for comorbid patients. In *KR4HC 2015*, volume 9485 of *Lecture Notes in Artificial Intelligence*, pages 95–108. Springer International Publishing, Pavia.

Quaglini, S., Stefanelli, M., Lanzola, G., Caporusso, V., and Panzarasa, S. (2001). Flexible guideline-based patient careflow systems. *Artificial Intelligence in Medicine*, 22(1):65–80.

Shahar, Y., Miksch, S., and Johnson, P. (1998). The Asgaard project: a task-specific framework for the application and critiquing of time-oriented clinical guidelines. *Artificial Intelligence in Medicine*, 14(1-2):29–51.

Sánchez-Garzón, I., Fernández-Olivares, J., Onaindia, E., Milla, G., Jordán, J., and Castejón, P. (2013). A Multi-agent Planning Approach for the Generation of Personalized Treatment Plans of Comorbid Patients. In *AIME 2013*, pages 23–27.

Terenziani, P., Bottrighi, A., and Rubrichi, S. (2014). META-GLARE: a meta-system for defining your own CIG system: Architecture and Acquisition. In *KR4HC*, pages 92–107.

Tu, S. W. and Musen, M. A. (1999). A flexible approach to guideline modeling. *Proceedings of the AMIA Symposium*, pages 420–424.

van den Akker, M., Buntinx, F., Metsemakers, J. F., Roos, S., and Knottnerus, J. A. (1998). Multimorbidity in general practice: prevalence, incidence, and determinants of co-occurring chronic and recurrent diseases. *Journal of Clinical Epidemiology*, 51(5):367–375.

Wilk, S., Michalowski, M., Michalowski, W., Rosu, D., Carrier, M., and Kezadri-Hamiaz, M. (2017). Comprehensive mitigation framework for concurrent application of multiple clinical practice guidelines. *Journal of Biomedical Informatics*, 66:52–71.