Exceptions Handling within GLARE Clinical Guideline Framework

Giorgio Leonardi[1], Alessio Bottrighi[1], Gabriele Galliani[1], Paolo Terenziani, Antonio Messina[2], and Francesco Della Corte[3]
[1]DiSIT, Univ. Piemonte Orientale "A. Avogadro", Alessandria, Italy
[2] Azienda Ospedaliera "Maggiore della Carità", Novara, Italy
[3]Department of Translational Medicine, Univ. Piemonte Orientale "A. Avogadro", Novara, Italy

Abstract. *Computerized clinical guidelines (CIGs) are widely adopted in order to assist practitioner and patient decision making. However, a main problem in their adoption is the fact that, during guidelines executions on specific patients, unpredictable facts and conditions (henceforth called exceptions) may occur. A proper and immediate treatment of such exception is necessary, but most current software systems coping with CIGs do not support it. In this paper, we describe how the GLARE system has been extended to deal with exceptions in CIGs.*

Introduction

Clinical Guidelines (CGs) are, in the definition of the MeSH dictionary, "work consisting of a set of directions or principles to assist the health care practitioners with patient care decisions about appropriate diagnostic, therapeutic, or other clinical procedures for specific clinical circumstances". One of the main goals of CGs is to capture medical evidence and to put it into practice. However, from one side, evidence is essentially a form of statistical knowledge, and 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 CG 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 CG: (i) ideal patients, i.e., patients that have "just the single" disease considered in the CG (thus excluding the concurrent application of more than one CG), and are "statistically relevant" (they model the typical patient affected by the given disease), not presenting rare peculiarities/side-effects; (ii) ideal context of execution, so that all necessary resources are available.

On the other hand, when a specific physician applies a given CG to a specific patient, unexpected conditions may show up. For instance, some laboratory instrument (recommended by the CG) may be missing, and/or the patient may show specific conditions (including the sudden appearance of life-threatening complications) not foreseen in the general CG. Such situations are unexpected, and, as such, cannot be specified a-priory in the CGs. However, especially in case of unexpected life threatening problems, the physician has to stop the "standard" execution of the CG, and to start to cope with such new problems. Henceforth, such problems will be called "exceptions", since they are exception to the "standard" execution of a CG.

Studies have shown that CG implementations can best affect clinician behaviour if they deliver patient-specific advise during the patient treatment [1,2]. Computer interpretable guideline (CIG) systems have been developed to this purpose (see, e.g., the survey in [3] and the overview of the status of the art in the recent book [4]). However, although such system supports the execution of CIGs on specific patient, they do not support the treatment of exceptions. This is a major limitation, which severely limit their practical applicability. The goal of this paper is to describe an approach to extend current CIG systems in order to support the definition and the treatment of exceptions. Although the methodology we propose is mostly general, in this paper we will focus specifically on the extensions we made to the GLARE system.

The GLARE System

GLARE (Guideline Acquisition, Representation and Execution) [5,6] is a software system to manage CGs. It has been built in a long-term cooperation between our Computer Science Department and Azienda Ospedaliera San Giovanni Battista in Turin, the 3rd largest hospital in Italy.

Representation formalism. GLARE relies on a limited but clear representation formalism [5,6], in which the basic primitives are *atomic* and *composite actions* (*plans*). Atomic actions are used to model elementary steps in a guideline, while composite actions represent more complex procedures, which can be defined in terms of their components via the *part-of* relation. A guideline itself is a composite action, which can be progressively refined by following the part-of chain, until atomic actions are reached. Three main types of atomic actions have been introduced in GLARE: *work actions, query actions,* and *decisions*. *Work actions* represent operative steps which must be executed at a given point of the guideline. *Query actions* are requests of information from the outside world (physicians, databases, knowledge bases). *Decision actions* are the means for selecting among alternative paths. Decision actions can be further subdivided into diagnostic decisions, used to make explicit the identification of the disease the patient is suffering from, and therapeutic decisions, used to represent the choice of a path, containing the implementation of a particular therapeutic process. Actions are described in terms of their attributes.

The order of execution of actions is established by means of a set of *control relations*: *sequence, controlled, alternative* and *repetition*. In particular, repetitions state that an action has to be repeated several times (maybe a number of times which is not known a priori, until a certain exit condition becomes true). On the other hand, controlled relations are used in order to represent temporally constrained actions, such as "start of A at least 1 hour after the beginning of B", and so on. (Possibly imprecise) action durations and temporal delays among actions can be specified.

Acquisition and Execution tools. GLARE's architecture is composed by two main modules: an *acquisition tool*, meant to be adopted –e.g., by a committee of expert physicians- to introduce a new guideline in the system, and an *execution tool*, exploited by the user physician to apply a guideline to a specific patient [5,6]. The tools strictly interact with a set of databases. In particular, the acquisition tool provides a user-friendly graphical interface to acquire guideline components. The acquisition tool also provides physicians with different forms of consistency checking, such as *name* and *range checking, logical design criteria fulfillment* (for example, alternative arcs may only exit from a decision action), and *temporal consistency checking*. The latter issue has required the adoption and the extension of complex AI techniques, whose description is outside the scope of this paper, but that was extensively treated in [7].

GLARE's execution module executes an acquired guideline for a given patient, retrieving data about the patient from the Patient Database. Temporal reasoning techniques are used in order to identify the next action (or the next set of actions) to be executed. Such actions are managed through the *agenda technique* [5]. The Agenda is a data structure containing the next actions to be executed for the given patient, with an indication of their execution time. In particular, for each action it contains, the agenda store its *earliest* and *latest allowed (execution) time*. The action in the Agenda must be executed in the scheduled time. After the execution of an action, it is deleted from the Agenda. The CIG is checked, to look for the next action(s). Such action(s) are then inserted in the Agenda, and its (their) earliest and latest allowed execution times are evaluated by the execution tool, on the basis of the current time and of the delays suggested in the guideline. In case the control relations in the CIG specify that not just one action, but a set of actions must be executed next, all such actions are inserted in the Agenda, and the execution engine support their concurrent execution.

Advanced facilities. GLARE architecture is modular and open. In the last years, GLARE has been extended with new modules, providing users with advanced facilities for CIGs (i) semi-automatic adaptation to specific execution contexts (e.g., a given hospital; [8]), (ii) the automatic treatment of temporal constraints in CGs [7], (iii) decision making, based on decision theory [9], (iv) model-based verification of CGs [10], and (v) execution in a distributed environment [11].

Coping with Exceptions

The management of exceptions within a software system managing CIGs arises several difficult challenges. For instance, in general, plans to cope with exceptions cannot be simply represented as new actions in CIGs, for the simple reason that exceptions are unpredictable, so that they can occur at any moment during the execution (or may not occur at all). Our approach consists of three tasks. First, we identify the "ontology" of exceptions. Second, we propose a formalism to represent the exceptions, and the plans to treat them. Third, we specify and devise a mechanism to integrate the treatment of exceptions with the "standard" execution of CIGs.

Ontology. At least two different types of exceptions can be distinguished: exceptions related to the status of the execution of the actions in a CIG, and exceptions related to the status of the patient. The first type (*CIG-exception* henceforth) arises whenever, for any contextual reason, the execution of the current action fails. For instance, the failure may be due to the absence of a resource needed to carry on the action (typically, an instrument). Another typical failure occurs when the deadline for the execution of the current action expires. The second type of exceptions (*patient-exception* henceforth) arises when the standard execution of a CIG cannot be carried on, due to an unexpected condition in the status of the patient. Indeed, we further distinguish into two subtypes of patient-exceptions: *CIG-dependent* and *CIG-independent*. CIG-dependent patient-exceptions are those exceptions that, although are not frequent enough to be explicitly managed in the CIG, can be foreseen to occur at a specific point in the execution of the CIG. For instance, an action of drug administration can raise an exception, in case a patient is allergic to such a drug. On the other hand, CIG-independent patient-exceptions are totally unpredictable, and may occur at any time during the execution of CIGs. For instance, an ischemic stroke may occur, requiring an immediate treatment, regardless of what is the CIG and the specific action in the CIG currently in execution.

In this work, we envision the development of a domain/guideline-independent library of exceptions, containing the representation of all types of exceptions.

Representation. In our approach, the description of the actions in the CIGs can be augmented, to include the list of its CIG-exceptions and of its CIG-dependent patient-exceptions. On the other hand, CIG-independent patient-exceptions are only stored in the library of exceptions. Each exception is represented through a set of attributes: <Name> and <Description> contain the name and a textual description of the exceptions (<Description> is optional). <Type> contains the type of the exceptions (see Ontology above). <Plan> contains a link to the plan to manage the exception. It is worth noticing that, thanks to the expressiveness of GLARE's formalism, exceptions can be represented using the same formalism we adopt in order to specify guidelines. As we will see, this solution allows us to deal in a homogeneous way (through the same execution engine) with both the execution of CIGs and the execution of plans coping with exceptions. Finally, the attribute <Modality> specifies what must be the interplay between the execution of the exception and the execution of the current CIG. For the sake of generality, we envision five different types of interplay

(i) *Concurrent*. The plan for the exception must be executed concurrently with respect to the current CIG;

(ii) *Suspend*. The CIG execution is suspended, and will be resumed after the end of the execution of the plan for the exception;

(iii) *Abort*. The current CIG is aborted. Only the plan for the exception is executed;

(iv) *Abort&Goto(X,CIG$_i$)*. The execution of the current CIG is aborted. The plan for the exception is executed. After this execution, the execution engine executes the action X of the guideline CIG$_i$.

(v) *Ignore*. The Exception is ignored; so that the standard execution of the current CIG continues (this modality is useful in cases when the execution of the CIG is crucial for the patient health, while the treatment of the exception is somehow not relevant, given the current CIG).

Our approach to the treatment of exceptions is characterized by the fact that we support the possibility of specifying different Plans of treatment and different modalities of execution for the same exception, depending on the context when it occurs (i.e., depending on which action of which guideline is being executed when the exception arise). This attention to the context is a major departure with respect to the treatment of exceptions in object oriented programming, as well as in the current approaches to exceptions in the clinical guideline literature. However, this step is extremely important, to cope with practical medical applications. As a specific example, let us consider an episode of sudden systolic blood pressure decrease (i.e. the value of systolic pressure falls under 90 mmHg). The procedure to cope with this exception during a hemodialysis session suggests a nurse to administer osmotic drugs (e.g. mannitol) to re-establish proper blood pressure levels, which allow the patient to terminate the session. In the severe trauma guideline, instead, different treatments are specified, which are furthermore highly dependent on the current action in the guideline. See section "Implementation" for further details.

To cope with such cases, we allow the possibility of specifying context-dependent plans and modalities for exceptions. Context-dependent plans and modalities are represented in a dedicated data structure stating, for each exception, for each CIG and (possibly) for each action in the CIG, what is the chosen plan and modality Of course, default can be specified, in case the same treatment (and or modality) is applicable to an entire guideline, or to all the guidelines. Notice that, additionally, in case the plan for the exception has alternative paths leading to alternative terminations, if the Abort&Goto modality is chosen, different "goto" actions can be specified, one for each possible termination. In such a way, we reconcile the generality of the plan to cope with an exception (which is the same independently of the current CIG) with the fact that its modality of execution may be, in certain cases, highly dependent on the current action/CIG being in execution.

Integration of Exception and CIG Executions. The execution engine of CIGs (called *Executor* henceforth) must be widely extended in order to cope with exceptions.

First, the basic *Executor* must be modified, in such a way that, (i) before the execution of each CIG action, the conditions triggering the CIG-dependent patient-exceptions associated to the current action (if any) are checked, and (ii) as a result of the execution of the action, the presence of CIG-exceptions is checked.
Second, a new engine, that we call *Exception-monitor*, must be devised, in order to continuously monitor the clinical record of the patient, to check whether the patient status triggers one of the conditions of the CIG-independent patient-exceptions. Of course, the *Exception-monitor* and the *Executor* run concurrently.

Each time an exception is triggered, the executor loads the related exception plan and its modality of execution (as explained in Section "Representation", the modality may be unique for the given exception, or may depend on the current CIG/action). Then, depending on the modality, the following actions are performed:

(i) in the case of "*Concurrent*" modality, the Executor inserts in the agenda also the first action in the plan for the exception, and re-start its standard agenda-based execution (it is worth remembering that (1) GLARE's original *Executor* supports the concurrent execution of the actions in the Agenda, and (2) since the plan for the exceptions are represented using the guideline formalism, the CIG Executor itself can be used in order to execute such plans).

(ii) In the case of "Suspend" modality, the first action of the exception plan is put on top of the Agenda. Thus, the Executor executes this plan first, and then returns to the other actions in the Agenda, recovering the execution of the suspended CIG.

(iii) In the case of "Abort" modality, the Agenda is first made empty (i.e., all the actions in it are deleted). Then, the first action of the exception plan is put on top of the Agenda. Thus, the Executor executes this plan first, and then terminates (since the Agenda becomes empty).

(iv) The "Abort&Goto(X,CIG_i)" modality is similar to "Abort", except that, after the completion of the exception plan, the action X of the guideline CIG_i is inserted in the Agenda, forcing the *Executor* to jump to that specific point of the CIG.

(v) In the case of "Ignore" modality, the standard execution of the current CIG is simply carried on.

Before concluding this section, it is worth mentioning two features of the approach we propose.
First of all, the methodology above easily supports CIG-dependent exception nesting. As a matter of fact, plans coping with exceptions are modelled like guidelines themselves, so that, in turn, their actions may specify CIG-dependent exceptions. Depending on the chosen modality, such nested exceptions may be run concurrently, or may be suspended, or may abort the execution of the exception plan in which they are nested.
Second, we also support cases in which two GIG-independent exceptions, defined in the exceptions library, are triggered together. Unfortunately, there seems to be no general rule to identify a unique correct behaviour, covering all possible situations. We thus allow the possibility to specify, for chosen pairs $<E_i,E_j>$ of exceptions, whether (1) E_i overrides E_j (so that only E_i is executed) or vice-versa, or (2) E_i must be executed before E_j (or vice-versa), or (3) E_i and E_j must be executed concurrently. Since not all the co-occurrences are known a-priori for each CIG, it is possible to define a minimum set of known general rules as the default behaviour, leaving guideline authors free to override these rules, or to add new ones, to provide different selection strategies for specific CIG. Furthermore, these rules can

be maintained incrementally: after defining a first set, new rules can be added when new exceptions are added to the repository, or each time a new co-occurrence has been experienced by the CIG users and defined as a new rule.

Implementation

This section presents a brief description of the exception handling system we developed and its integration with the Glare system. Moreover, real world examples, implemented in Glare using the functionalities of this new exception manager, are provided in the context of severe trauma management, to show the applicability of our strategy in a real context.

System components implementation. We implemented the concepts exposed in an exception management system, which has been integrated and tested in Glare. The exception management system is implemented in Java and is composed by two main components: (1) a graphical editor for user-friendly definition and maintenance of the exceptions library, and (2) proper extensions to the Glare execution engine to allow the CIG-independent patient-exceptions handling. Each exception of this type is associated to a proposition, which defines a particular (problematic) patient's state. Exceptions are triggered when the associated propositions become true, independently from the status of execution of the actions in the CIG. To access the patient-related information (e.g. the blood pressure or the body temperature) needed to periodically verify these propositions, the system relies to SQL queries to retrieve the most up-to-date information from the patient's data base.

Graphical editor and exception repository maintenance. The exception management system offers a user-friendly graphical editor, which permits the CIG author to: (1) build new propositions combining operators, operands and constant values; (2) create new exceptions and eventually customize the modality of execution of the exception plan for a particular CIG plan (see "Representation" above), and (3) define the rules to be applied by the execution engine when two or more exceptions are triggered at the same time (see "Integration of Exception and CIG Executions" above).

In particular, the engine helps the user in performing these three task providing the following features: considering the task (1), the editor permits the user to define new propositions (for example the proposition " 'systolic pressure' <= '90 mmHg' AND 'cardiac frequency' < '45 bpm') or to edit an existing one by combining operators (i.e. arithmetic, logic and comparison), operands (such as 'systolic pressure') and constants (e.g. '90 mmHg') by means of a guided procedure. For each operand (see Figure 1), the editor can automatically build the SQL query which will be used run-time to retrieve the most up-to-date values associated to the considered operand from the patients database, each time this proposition must be evaluated. Once defined, the proposition is associated with a unique identifier (proposition_ID) and can be saved in the propositions library.
Task (2) helps the user in defining new exceptions or in modifying existing ones. An exception is defined as a tuple: <exception_ID, name, description, plan_ID, proposition_ID, exceptionPlan_ID, modality>, where:

- exception_ID: unique identifier of the exception
- name, description: name and natural language description of the exception
- plan_ID: identifier of the CIG plan for which this exception must be checked
- proposition_ID: identifier of the proposition to be verified to raise this exception
- exceptionPlan_ID: identifier of the exception plan to be executed to manage this exception
- modality: modality of execution of the exception plan (Concurrent, Suspend, Abort, Abort & Goto, Ignore)

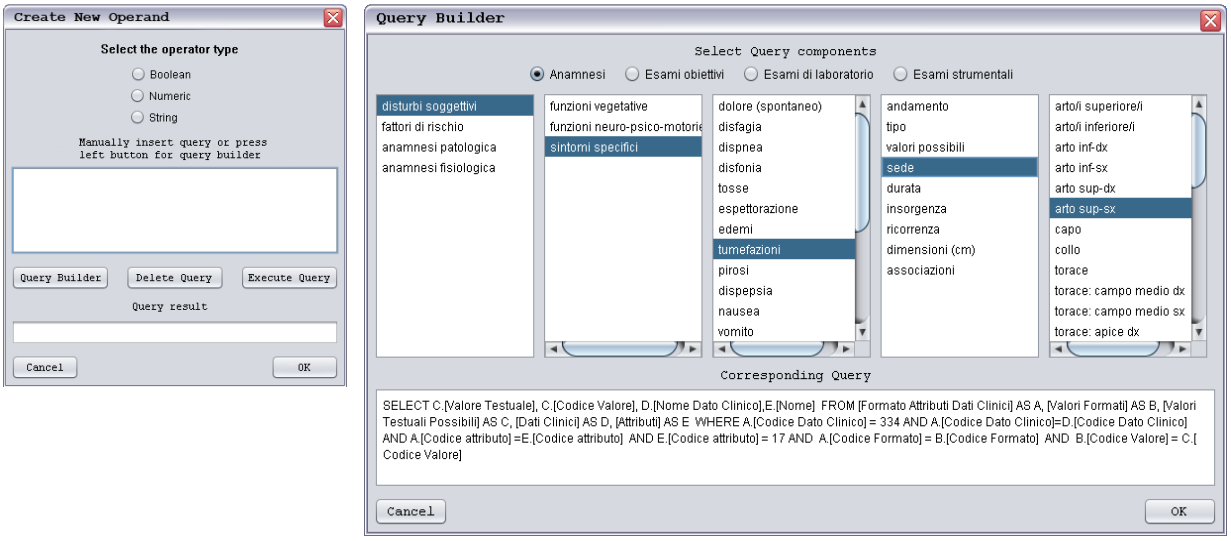

Figure. 1. Creation of new operands and associated SQL query using the graphical editor. The names in the right figure refer to real patient's data defined in the RoPHS project.

Once the user has indicated the exception_ID (or accepted an auto-generated identifier), she/he must: (1) enter name and description of the associated exception, (2) choose a proposition from the propositions library and an exception plan from the exception plans library. The standard execution modality (if defined) for the chosen plan is automatically loaded, but the user has the possibility to override it, by changing the modality or by customizing it to indicate, for example, to go to a particular action of the CIG plan, in case of Abort & Goto.
Finally, task (3) permits to define, for each pair $<E_i, E_j>$ of exceptions defined for the plan_ID, the rules to be applied (i.e. one of them overrides or must be executed before or after the other one, or both of them must be executed concurrently) when they happen at the same time.

All the exceptions defined by the user are permanently stored in the exception library, in order to be automatically loaded and used by the execution engine, when a new instance of a CIG plan is created and run. This repository can be updated at any time using this graphical editor, adding, deleting or modifying propositions, exceptions and co-occurrence rules.

CIG execution engine modifications. The CIG execution engine of Glare has been modified and extended to cope with exceptions at run-time. The new version of the engine now loads and exploits the exceptions saved in the exceptions library when a new instance of a CIG plan is generated and started. In particular, when a new instance of CIG plan is created, on one side an exception monitor, responsible for checking and raising the exceptions defined for the CIG plan itself, runs concurrently and asynchronously with the execution engine. On the other side, the engine can interrogate the monitor at any time, during the CIG plan execution, to know if exception(s) raised and, if the answer is positive, can apply run-time the exception plan(s) to manage it/them.

The exception monitor has been implemented as a Java thread (subclass of the *java.lang.Thread* class), and its constructor permits to receive:

(1) the set of exceptions to be checked (the set E of tuples <exception_ID, name, description, plan_ID, proposition_ID, exceptionPlan_ID, modality>, where plan_ID matches the CIG plan which is being executed);
(2) the time interval T which must elapse between two successive exceptions check, and
(3) the information about what must be done when two or more exceptions raises at the same time (defined in task (3) of the graphical editor).

After the engine has created the new instance of the CIG plan to be executed, and has instantiated and properly loaded the monitor, both the guideline and the monitor are started. In particular, the monitor executes the following operations (implemented by overriding the method *run()* of the *Thread* class):

1. Wait for the period T to elapse
2. For each exception *exc* ∈ E:
    a. Let P the proposition identified by: *exc.propositionID*, associated to the exception *exc.exceptionID*
    b. For each operand *op* ∈ P, execute the query *op.query* on the patients database to retrieve the most up-to-date values
    c. Solve the proposition P and store its result R (true/false) as a pair <exception_ID, R>
3. Jump to point 1

The exceptions to be raised are defined as all the exception_IDs for which the corresponding value of R is set to true when the monitor is interrogated. The operations from point 1 to point 3 form an infinite cycle, which will be interrupted by the execution engine (by interrupting the thread implementing the monitor) only when the instance of the CIG reaches its end.

After having instantiated and run the CIG and the monitor, and while the latter performs autonomously its operations, the engine behaves like the following: each time the next CIG activity must be chosen, the execution engine asks the monitor if exceptions raised meanwhile. If the answer is negative, the CIG execution continues normally, otherwise the CIG execution engine receives from the monitor the set of exceptions to be managed and, if the set contains more than one exception, it also receives information about how to manage the multiple exceptions. Afterwards, the exception plan(s) are loaded and the execution stack is updated accordingly to the policy to be applied to manage the eventual multiple exceptions and to the modality of execution associated to the plan(s). After having performed all these operations, the engine continues as usual, since the exception plan(s), dynamically loaded and properly put on the execution stack, are now seen by the engine as part of the (augmented) CIG. The process described continues until the CIG reaches its end.

The extensions described are totally transparent to the user, since this important additional service allows the system to execute CIGs in a flexible and more true to life way in a completely automatic fashion. In the case where no exceptions are defined for a particular CIG, the latter will be executed such as no CIG-independent patient-exceptions management system exists.

CIG implementation. In this section we describe the application of our approach with real examples in the context of severe trauma treatment. The severe trauma guideline, briefly described in this section, is developed under the RoPHS (Report on the Piedmont Health System) project, funded by the Regione Piemonte.

This guideline is based on the evidence of the need of a synergistic organization to reduce the time of intervention and the excessive delay before hospital admission. In particular, the guideline is based on the following principles:

(i) an appropriate clinical out-of-hospital evaluation of trauma patients for a prompt identification of lesions and therapeutic priorities;
(ii) the ability to perform some effective life-treating actions to support cardiac and pulmonary functions;
(iii) the correct clinical decision must be made to refer the patient to an hospital offering a competent treatment of the trauma, not necessarily the nearest, and
(iv) the optimum operative linking between the services for the acute management and the rehabilitation hospital.

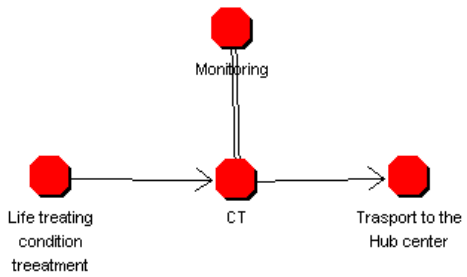

Figure 2. Fragment of the severe trauma guideline acquired in GLARE

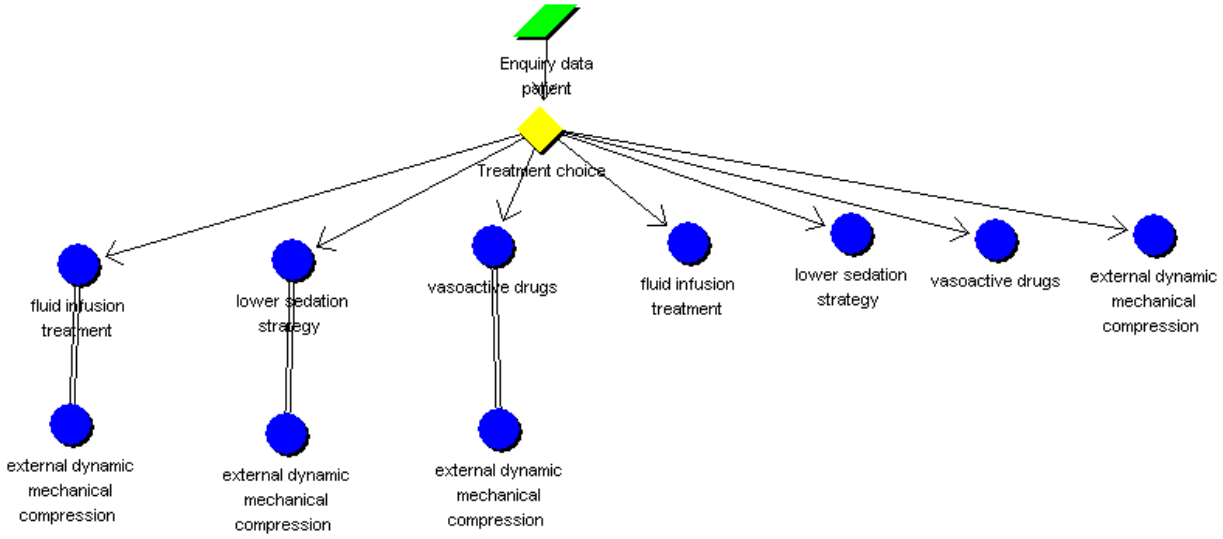

Figure 3. The treatment plan for systolic blood pressure exception meanwhile "Life treating condition treatment" plan execution.

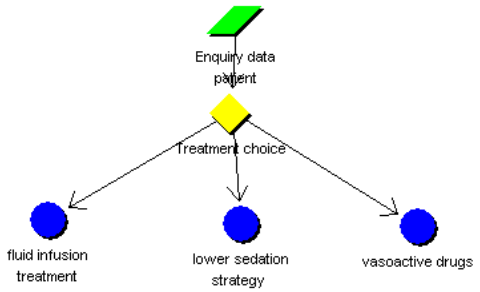

Figure 4. The treatment plan for systolic blood pressure exception after "Life treating condition treatment" has been executed.

Moreover, the guideline exploits the "Hub and Spoke" strategy, in order to emphasize the integration of services throughout the various service areas to ensure efficacy of treatment of the trauma patients. The "Hub and Spoke" model relies on a leader center (i.e. the hub), where the trauma patients may be treated for severe neurologic, thoracic, abdominal and vascular traumas. Spoke centers are strategically placed in the territory to ensure an appropriate treatment in the geographical area and are closely linked with their hubs through the emergency system (e.g. the 118 service, in Italy). In the context of the RoPHS project, the guideline is developed and applied on Eastern Piedmont. In this area, the Hub center is located in Novara, and the Spoke centers are hospitals located in Borgosesia, Borgomanero, Domodossola, Verbania, Vercelli and Biella.

Different exceptions are defined for this guideline: exceptions which can occur during the execution of the entire guideline, and exceptions which can occur only in specific sections of the guideline. For example, an exception applied on the whole guideline regards the heart failure. In the case that the patient has a heart failure, the current action and the guideline are immediately aborted, (since the <MODALITY> of this exception is set to Abort), and the treatment plan for heart failure exception is executed.
Considering exceptions occurring in a specific section of the guideline, shown in Figure 2, the "Life treating condition treatment" plan is followed by two concurrent plans: (1) "CT" plan (defining all the actions concerning the choice and the execution of the proper kind of Computed Tomography), and (2) "Monitoring" plan, where the patient is monitored with a multi-parametric monitor to gather information about his/her heart rate, blood pressure, pulse oximetry and saturation. Finally, the patient will be transported in a specialized hospital, called Hub center. During this part of guideline, exceptions on monitored data can happen. For example, an exception to cope with the problem of sudden systolic blood pressure decrease is defined. This exception captures episodes where the systolic pressure falls under 90 mmHg. If the exception monitor, while checking the patient's systolic pressure, verifies that it is lower than 90 mmHg, it raises this exception. The execution engine will finally capture the exception and will start the

proper treatment (exception) plan. The treatment plan is dependent from the action executed at the time of the exception, since its management in the severe trauma guideline is context dependent:

- If the current action(s) belong(s) to the "Life treating condition treatment" plan, this plan will be executed concurrently with the exception (i.e. <MODALITY> is set to Concurrent). This exception is managed by choosing one or more of the following actions: fluid infusion treatment, vasoactive drugs, and lower sedation strategy. These actions can be performed as unique treatments, or can be associated to external dynamic mechanical compression, in order to manage an eventual arterious external bleeding. Moreover, also the external dynamic mechanical compression can be performed as a unique treatment (see Fig.3).
- Otherwise, if the exception is captured after the conclusion of the "Life treating condition treatment" plan, the exception plan, even if executed concurrently with the guideline such as the previous one, relies on a different strategy. The actions to be performed in this plan are: fluid infusion treatment, vasoactive drugs, and lower sedation strategy. Note that there is not external dynamic mechanical compression between them (neither unique treatment nor in combination with one other treatment) since, after the "Life treating condition treatment", the patient's conditions are more stable and arterious external bleeding is not possible (see Fig. 4).

The system GLARE, extended as described above, allowed to cope with all the exceptions needed to properly manage the severe trauma guideline, representing and managing them during the execution of this CIG.

Discussion and conclusions

In many practical cases, unexpected conditions (exceptions) may arise during the execution of a CIG. In this paper, we proposed a general approach to cope with exceptions, showing how software systems coping with CIGs can be extended (in the representation formalism, as well as in the execution module). Moreover, we have demonstrated the feasibility of our approach extending the GLARE system [6] and applying it the severe trauma guideline developed under the RoPHS (Report on the Piedmont Health System) project, funded by the Regione Piemonte.
Considering the generality and relevance of the phenomenon, in the following we discuss the more relevant approaches in the literature, covering exceptions representation and management.
PROforma language [12] allows one to specify exceptions, while the system can set an Exception flag if an abnormal event has occurred in the processing of a guideline operation. However, in our approach, exceptions are hierarchically organized and we distinguish among three different types of exceptions. Moreover, for every exception, it is possible to define a specific treatment plan to manage it.
Tu and Musen [13] differentiate between normal flow and exceptional flow and contemplate the specification of scenario-based exceptions handlers. Tu and Musen only consider exceptions regarding the execution of actions in the guideline. In their approach only the current action(s) can generate exceptions (such as our CIGs dependent exception). Our approach is more general, since we also allow to define exception on the basis of the patient's data. Moreover, [13] provides only two modalities of interactions between CIGs and exceptions: (1) exceptions leading a patient back to a scenario covered by the guideline (as our Abort&Goto modality), and (2) exceptions where the patient is managed outside the guideline (Abort modality in our approach).
Hierarchies of exceptions are managed by [14]. They distinguish between "hazards" (concept somehow close to our CIG-independent exceptions) and "obstacles" (as our CIG-dependent exceptions), mainly focusing of what we call CIG-exceptions, allowing hierarchical refinement. They take into account some of the modalities of interactions between CIGs and exceptions we also pointed out in our approach: parallelExecuting (close to our Concurrent modality), suspending (Suspend in our approach) and discarding (Abort in our approach). However, in their approach, exception treatment is goal oriented: corrective actions are chosen accordingly to the substitution of the default goal expressed in the CIG. Our approach is context-dependent, since corrective actions are automatically chosen when the patient status reaches an undesired state.
Quaglini et al [15] managed the flexibility of their guideline-based careflow system introducing a classification of exceptions which is similar to our ontology. Exceptions can be expected or unexpected, synchronous or asynchronous and GL-related or unrelated. Our approach differs in the exception handling, since in [15] the triggers are requested by the physician who, at a certain point of the CIG execution, asks the system to delay the current action, to substitute the current action with another one or to add a new non-planned action. The system will then manage this trigger to cope with the physician's request. Our system, instead, triggers exceptions when an abnormal patient's state is detected, and automatically loads from the repository a plan to cope with the triggered exception, without the need of human intervention. This difference is justified, because [15] manages exceptions in a careflow management system, where the physician must have the experience (and must be free) to introduce the proper

deviations under her/his responsibility. Our system is a CIG-based decision support system, therefore it is responsibility of the system to suggest not only the proper actions to be performed in a normal situation, but also to recognize abnormal states of the patient and to suggest what must be done to treat the patient accordingly.
Peleg et al. [16] propose a methodology for eliciting and modelling exceptions. They describe a conceptual model in which exceptions can be expected or unexpected, and are triggered as asynchronous events. Triggers start a synchronous exception management branch during the CIG execution. [16] defines two types of exception causes: (1) HumanCause, e.g. human errors, malicious actions or non-compliance, and (2) Non-humanCause, e.g. organizational causes, work item failures, deadline expirations. However, they do not distinguish between CIG-dependent and CIG-independent exceptions, and the exception management, briefly sketched, describes an approach similar to our Suspend modality, but goal driven: after the execution of the corrective actions to manage an exception, a set of goals should fulfilled to resume the CIG normal execution otherwise the CIG is aborted. On the other hand, our approach defines a complete exception management system, where different modalities of execution can be defined for each exception plan.
Future research will focus on an extensive evaluation of the implemented system for educational and training purposes and on an extensive study of the impact of exception co-occurrences to define alternative rules for selecting the more appropriate exception plan, eventually involving medical knowledge for this selection.

ACKNOWLEDGEMENTS.
The work reported in this paper has been partially supported by Regione Piemonte, in the RoPHS (Report on the Piedmont Health System) project.

REFERENCES

[1] JM. Overhage, WM Tierney, XH. Zhou, CJ. McDonald. A randomized trial of Corollary Orders to Prevent Omissions. JAMIA 4(5), 364-375, 1997.
[2] S. Shea, W. DuMouchel, L. Bahamonde. A meta-analysis of 16 randomized controlled trials to evaluate computer-based clinical reminder systems for preventing care in the ambulatory setting. JAMIA 3(6), 399-409, 1996.
[3] M. Peleg, SW Tu, P. Ciccarese, J. Fox, RA. Greenes, et al. Comparing Computer Interpretable Guideline Models: a Case Study Approach. JAMIA 10(1), 52-68, 2003.
[4] A. Ten Teije, S. Miksch and P. Lucas. Computer-based Medical Guidelines and Protocols: A Primer and Current Trends, Volume 139 Studies in Health Technology and Informatics, July 2008.
[5] P. Terenziani, G. Molino, M. Torchio. A Modular Approach for Representing and Executing Clinical Guidelines. Artificial Intelligence in Medicine 23, 249-276, 2001.
[6] P. Terenziani, S. Montani, A. Bottrighi, G. Molino, M. Torchio, Applying Artificial Intelligence to Clinical Guidelines: the GLARE Approach, in [4], 273-282.
[7] L. Anselma, P. Terenziani, S. Montani, A. Bottrighi. Towards a Comprehensive Treatment of Repetitions, Periodicity and Temporal Constraints in Clinical Guidelines. Artificial Intelligence in Medicine Journal 38, Elsevier, 171-195, 2006.
[8] A. Bottrighi, P. Terenziani, S. Montani, M. Torchio, G. Molino. Clinical Guidelines Contextualization in GLARE, Proc. AMIA'06, Washington, November 2006.
[9] S. Montani , A. Bottrighi, P. Terenziani. Supporting Therapy Selection in Computerized Clinical Guidelines by Means of Decision Theory. Proc. Medinfo'07, Brisbane, Australia, 2007.
[10] A. Bottrighi, L.Giodano, G. Molino, S. Montani, P. Terenziani, M. Torchio. Adopting model checking techniques for clinical guidelines verification. Artificial Intelligence in Medicine 48(1), 1-19, 2010.
[11] A. Bottrighi, M. Torchio, S. Montani, G. Molino, P. Terenziani. Supporting human interaction and human resources coordination in distributed clinical guidelines, Proc. Medinfo (World Congress for Health Informatics) 2010, Cape Town, September 2010, 319-323.
[12] DR. Sutton, J. Fox. The syntax and semantics of the PROforma guideline modeling language. JAMIA 10(5), 433-443, 2003.
[13] S. W. Tu, M. A. Musen , A flexible approach to guideline modeling. Proc AMIA Symp. 1999: 420–424.
[14] A. Grando, M. Peleg, D. Glasspool. A goal-oriented framework for specifying clinical guidelines and handling medical errors. Journal of Biomedical Informatics 43, 287-299 (2010).
[15] Quaglini, S, Stefanelli, M, Lanzola, G, Caporusso, V, Panzarasa, S. Flexible guideline-based patient careflow systems. Artif Intell Med 2001 Apr;22(1):65-80.
[16] Peleg, M, Somekh, J, Dori, D. A methodology for eliciting and modeling exceptions. J Biomed Inform 2009 Aug;42(4):736-47.