Artificial Intelligence in Medicine (2006) **38**, 171–195

ELSEVIER

ARTIFICIAL INTELLIGENCE IN MEDICINE

http://www.intl.elsevierhealth.com/journals/aiim

Towards a comprehensive treatment of repetitions, periodicity and temporal constraints in clinical guidelines

Luca Anselma [a,*], **Paolo Terenziani** [b,1], **Stefania Montani** [b,1], **Alessio Bottrighi** [b,1]

[a] *DI, Università di Torino, Corso Svizzera 185, 10149 Torino, Italy*
[b] *DI, Università del Piemonte Orientale "Amedeo Avogadro" , Via Bellini 25, 15100 Alessandria, Italy*

Received 17 December 2004; received in revised form 17 March 2006; accepted 21 March 2006

KEYWORDS
Clinical guidelines;
Temporal constraint representation;
Temporal reasoning;
Consistency-checking;
Repeated/periodic actions

Summary

Objective: In this paper, we define a principled approach to represent temporal constraints in clinical guidelines and to reason (i.e., perform inferences in the form of constraint propagation) on them. We consider different types of constraints, including composite and repeated actions, and propose different types of temporal functionalities (e.g., temporal consistency checking).
Background: Constraints about actions, durations, delays and periodic repetitions of actions are an intrinsic part of most clinical guidelines. Although several approaches provide expressive temporal formalisms, only few of them deal with the related temporal reasoning issues.
Methodology: We first propose a temporal representation formalism and two temporal reasoning algorithms. Then, we consider the trade-off between the expressiveness of the formalism and the computational complexity of the algorithms, in order to devise a correct, complete and tractable approach. Finally, we show how the algorithms can be exploited to provide clinical guideline systems with different types of temporal facilities.
Results: Our approach offers several advantages. During the guideline acquisition phase, it enables to represent temporal constraints, and to check their consistency. In the execution phase, it checks the consistency between the execution times of the actions and the constraints in the guidelines, and provides query answering and simulation facilities.

* Corresponding author. Tel.: +39 011 6706821; fax: +39 011 751603.
E-mail addresses: anselma@di.unito.it (L. Anselma), terenz@mfn.unipmn.it (P. Terenziani), stefania@mfn.unipmn.it (S. Montani), alessio.bottrighi@mfn.unipmn.it (A. Bottrighi).
[1] Tel.: +39 0131 360174; fax: +39 0131 360198.

0933-3657/$ – see front matter
doi:10.1016/j.artmed.2006.03.007

1. Introduction

Clinical guidelines are a means for specifying the ''best'' clinical procedures and for standardizing them. Over the last few years, the medical community has started to recognize that computer-based systems dealing with clinical guidelines provide relevant advantages, since, e.g., they can be used to support physicians in the diagnosis and treatment of diseases, or for education, critical review and evaluation aims [1]. Thus, many different approaches and projects have been developed in recent years to create domain-independent computer-assisted tools for managing clinical guidelines (see e.g., Asbru [2], DILEMMA and PRESTIGE [3], EON [4], GEM [5], GLIF [6,7], GUIDE [8,9], ONCOCIN [10], PROforma [11], T-HELPER [12], and also [1,13–15]). Most of these approaches distinguish between an *acquisition phase*, in which expert-physicians (usually in cooperation with knowledge engineers) introduce clinical guidelines into the computer-based system, and an *execution phase*, when user-physicians execute a given guideline on a specific patient (i.e., on the basis of the patient's data). Moreover, recently, several approaches have started to focus also on the treatment of temporal aspects [16–19]. 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. Furthermore, it is also necessary to carefully take into account the (implicit) temporal constraints derived from the hierarchical decomposition of actions into their components and from the control-flow of actions in the guideline.

Within the Artificial Intelligence (AI) community, a lot of work has been devoted to the treatment of time-related phenomena; roughly speaking, one could identify two different mainstreams [20].

The first mainstream is mainly devoted to the definition of general-purpose formalisms, covering a wide range of temporal (and, possibly, non-temporal) phenomena. It aims at coping with the evolving world, by describing the internal structure of events, and at modelling how the world changes in response to such events. In particular, a lot of different logical approaches (including first order, modal, or nonmonotonic logical formalisms) have been developed to achieve such a goal.

The second mainstream is mainly focused on the definition of representation formalisms and of reasoning techniques to deal specifically with temporal constraints between temporal entities per se, without regard to the internal description of such entities (henceforth, we will call ''constraint-based'' the approaches in this mainstream). In particular, since the 1980s, several domain-independent constraint-based temporal managers have been developed within the AI community (see, e.g., [21] for a comparison between some of them, and the general survey in [22]); such managers are mainly conceived as knowledge servers to which temporal constraint propagation problems can be delegated, and which possibly have to be coupled with other modules (e.g., a planner) to solve complex problems. By focusing on a more restricted problem, one could obtain higher efficiency than general-purpose approaches. In fact, it could be possible to define specialised reasoning techniques that make inferences in a more efficient way than, e.g., a standard theorem prover for the first-order logic. However, the aim towards specialization led many of these approaches to focus on specific classes of constraints, so that there still seems to be a gap between the range of phenomena they cover and the types of temporal constraints emerging from the domain of clinical guidelines. In this paper, we propose a constraint-based approach aiming at filling (at least in part) such a gap.

Developing a constraint-based manager of temporal constraints for clinical guidelines involves both the design of an expressive representation formalism, and the development of suitable temporal reasoning algorithms operating on them. However, subtle issues such as the *trade-off* between the *expressiveness* of the representation formalism and the *tractability* of correct and complete temporal reasoning algorithms have to be faced in order to deal with temporal constraints in a well-founded way; few works in the area of computerized guidelines have deeply analysed this topic so far.

In Section 2 we discuss the advantages of a well-founded approach, provide an overview of the state of the art, and sketch the basic principles and motivations underlying our approach. In Section 3 we introduce our formalism, and in Section 4 we provide two tractable, correct and complete algorithms to perform temporal reasoning in the acquisition and in the execution phase, respectively. In Section 5 we describe how to exploit our formalism and algorithms to provide clinical guidelines systems with temporal reasoning facilities. Finally, we address comparisons and conclusions.

2. Representing and reasoning with temporal constraints in clinical guidelines

Representing and reasoning with temporal constraints is an essential feature for computer-based approaches to clinical guidelines. In particular, a temporal manager coping with time-related issues can be exploited in different ways in the management of clinical guidelines. For instance, during the acquisition of a new guideline, the consistency of the temporal constraints it contains can be automatically checked; during the execution of a guideline on a specific patient, the temporal manager can be used to check whether the specific actions have been executed in such a way that the constraints in the guideline have been respected, or to determine the times when the next actions need to be executed (see Section 5 for a more detailed and exhaustive treatment of these issues). Therefore, desiderata such as correctness and completeness of temporal reasoning play a crucial role also within clinical guideline applications (see Section 2.1). However, although many domain-independent temporal managers have been devised within the AI literature (see, e.g., [22]), and several approaches to time-related issues have been faced within the clinical guideline literature (Section 2.2), several new challenges have to be addressed when dealing with temporal representation and temporal reasoning about clinical guidelines (Section 2.3).

2.1. Motivating the desiderata

As in most AI approaches to the treatment of time (see, e.g., the survey in [22]), in designing our formalism we had to carefully take into account the fundamental *trade-off* between the *expressiveness* of (temporal) formalisms and the temporal *complexity* of the *correct* and *complete* (temporal) reasoning algorithms operating on them.

While expressiveness is an obvious desideratum, we will now briefly motivate the second term of the above trade-off. First, it is important to stress the point that a formalism for temporal constraints is not very useful if it is not paired with algorithms for temporal reasoning, performing temporal inferences on a set of constraints (expressed in the given formalism) and/or checking their consistency. Consider, for instance, a Knowledge Base (KB) containing the temporal constraints (i) and (ii) between three events A, B and C.

$$\mathrm{KB} = \{(\mathrm{i})\ \mathrm{A\ before\ B};\ (\mathrm{ii})\ \mathrm{B\ before\ C}\}$$

The constraint (iii) *A before C* can be inferred (i.e., it is logically implied by (i) and (ii)), so that, given KB, one can correctly assert (iii), but not (iv) *A after C*, which is actually *inconsistent* with KB (in other words, the set of constraints KB′ = {(i), (ii), (iv)} cannot be satisfied). Temporal reasoning is necessary in order to support such an intended semantics. With no temporal reasoning, a user can represent any set of constraints, even an inconsistent one (e.g., KB′ above) with no reaction by the system.

Of course, temporal reasoning algorithms are computationally expensive. An important desideratum is *tractability*, i.e. the fact that the running time of the algorithms grows as a fixed power of the number of the actions and/or constraints in the knowledge base (i.e., polynomial time). Any faster-growing (e.g., exponential time) algorithms are not tractable and much more computationally expensive.

However, temporal reasoning algorithms should also be *correct*, i.e., such that they only infer constraints that are logically implied by the initial set of constraints (correctness grants that no wrong inference is made). *Completeness* (i.e., the fact that all logically implied constraints are actually inferred) is a fundamental desideratum as well, since it is essential in order to grant that the system's answers are fully reliable (e.g., if (iii) is not inferred from {(i), (ii)}, the answer to the question "Is (iv) consistent with {(i), (ii)}?" may be yes).

In particular, as in most AI approaches, the main task of our temporal reasoning algorithms is that of *checking the consistency of temporal constraints* in a guideline. In fact, real-world guidelines usually consist of hundreds of actions, often related by temporal constraints. This means that: (i) the fact that hundreds of constraints are mutually consistent cannot be taken for granted and (ii) consistency checking cannot be directly performed by physicians (and/or by a knowledge engineer), since making explicit all the possible implications of such a large number of constraints is an overwhelming and too complex task.

2.2. State of the art

Many AI approaches focused their attention to the definition of suitable formalisms to represent time-related phenomena and to reason with them. Besides "logical" approaches (e.g., temporal or non-monotonic logics), starting from the early 80s, many constraint-based approaches have been developed in AI [22]. Such approaches are mostly concerned to define domain-independent knowledge servers to which temporal reasoning, in the

form of *propagation of temporal constraints*, can be delegated, and which can be coupled with other modules (e.g., a planner, or a system which manages guidelines) to solve complex problems.

The aim towards specialization led these approaches to focus on specific classes of constraints (e.g., *qualitative* constraints such as "A before B", *quantitative* constraints such as *dates*, *delays* and *durations*) [22], or to devote great attention to granularities and/or periodic/repeated constraints [23–25] or to the integration of different sorts of constraints (e.g., qualitative and quantitative constraints [26]).

In the area of clinical guidelines several interesting approaches have been devised to represent temporal constraints. For instance, GLIF [6,7] deals both with temporal constraints on patient data elements and with duration constraints on actions and decisions. In PROforma [11], 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 [3] model temporal constraints within conditions. EON [4] 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 [27].

A rich ontology to deal with temporal information in clinical trial protocols has been proposed in [28], considering also relative and indeterminate temporal information and cyclical event patterns.

Despite the large amount of work devoted to the representation of temporal constraints, and the very rich and expressive formalisms being identified, little attention has been paid to temporal reasoning. Notable exceptions are represented by the approaches by Shahar [18] and by Duftschmid et al. [19].

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 of 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 [2,19]. In particular, in [19], 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 [29]. Temporal constraint propagation is used in order to (1) detect inconsistencies, and to (2) provide the minimal constraints between actions. In [19], 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 [30].

2.3. Dealing with temporal constraints in clinical guidelines: new challenges and open problems

Despite the large amount of valuable works, there still seems to be a gap between the range of phenomena covered by current AI constraint-based approaches and the needs arising from clinical guidelines management. The first important issue concerns the *expressiveness* of temporal formalisms. Consider, e.g., Example 1 (which is a simplified part of a guideline about multiple mieloma).

Example 1. The therapy for multiple mieloma is made by six cycles of 5-day treatment, 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.

While Example 1 above is a real example taken from the clinical domain, in the following we introduce a fictitious example, which includes, besides the main features of Example 1, other interesting features emerging from the clinical domain (e.g., "conditioned" repetitions). We thus use Example 2 as the leading example in our paper, to exemplify the expressiveness of our approach and its reasoning capabilities.

Example 2. The guideline G is composed by two repeated actions, a and b, where b must start at least 10 days after the end of a. Action a is repeated once every week for 2 weeks, until condition c_w does not hold anymore. Action b is repeated twice a week and each repetition must be at most 1 day after the previous one.

In its turn, a is composed by the repeated action $a1$ and the atomic action $a2$. $a1$ is repeated once a day for 3 days and each repetition is performed if the condition c_i holds. Action $a2$ lasts exactly 2 days and must start after the end of the repeated action $a1$ but no more than 1 day after.

Finally, $a1$ is composed by two atomic actions, $a11$ and $a12$ that must be performed in parallel (i.e., at the same time).

Of course, in the clinical practice, any guideline can be executed on different patients. For instance, let us suppose that the (temporal constraints involved by the) execution of the guideline in Example 2 on patient $P1$ is described in Example 2A.

Example 2A. Since, for patient $P1$, condition c_w held only for the first week, action a has been repeated only once. The subaction $a1$ has been attempted on the second, third and fourth day of the first week; however, since c_i only held on the second and fourth day, the second execution of $a1$ has been skipped. These facts led to the observation of the following instances of atomic actions:

- $a11_1$ (which is an instance of $a11$, and, precisely, of the first execution of the first repetition of $a1$ composing the first repetition of a), executed between 8 and 9 a.m. of the second day;
- $a12_1$ at the same time of $a11_1$;
- $a11_3$ executed during the fourth day;
- $a12_3$ at the same time of $a11_3$;
- $a2_1$ starting between 3 and 4 hours after the end of $a12_3$;
- $b1_1$ starting 20 days after the end of $a2_1$.

While the constraints in Example 2A can be easily modelled in most of the constraint-based formalisms developed within the AI literature (in the following, a representation in STP [29] of such constraints will be provided—see Fig. 9), temporal constraints in Example 2 are more challenging. Roughly speaking, almost all AI constraint-based temporal approaches are ''extensional'', since they operate on a knowledge base containing an explicit representation of all the events, and all the constraints between them. Thus, for instance, the treatment of Example 2 in the STP (or in the TCSP) framework [29] would involve an explicit representation of all the actions in all the repetitions (and of the corresponding constraints). Such a solution seems to us neither commonsense (since plenty of events and constraints need to be unnecessarily elicited) nor efficient (since it involves an unnecessary growth of the number of modelled events). Even worst, it is not practically feasible if one wants to model also ''conditioned repetitions'' (such as in the case of a and $a1$ in Example 2), in which the number of repetitions cannot be known at the time the guideline is built (and which are, realistically, quite common within clinical guidelines). We thus aim at devising, as far as possible, an ''intensional'' approach, in which repetitions are not ''expanded'' unless needed.

A further desideratum of our approach is that of describing a ''high-level'' representation formalism, providing, as far as possible, explicit constructs to model the different temporal issues to be captured in the guideline domain. This means that we want to provide a direct support to model:

(1) qualitative (such as ''at the same time'') and quantitative (e.g., ''at least 10 days after'') constraints, as well as repeated/periodic events (e.g., actions a, $a1$ in Example 2); all types of constraints may be *imprecise* and/or *partially defined*;
(2) a structured representation of complex events (in terms of part-of relations), to deal with structured descriptions of the domain knowledge (consider, e.g., the composite action a in Example 2);
(3) the distinction between *classes* of actions (e.g. an action in a general guideline) and *instances* of such actions (e.g., the specific execution of an action in a guideline) (the actions in Example 2 are ''classes''; $a11_1$—$b1_1$ in Example 2A are instantiations of such actions on the specific patient $P1$).

Of course, our temporal reasoning algorithms have to operate on a knowledge base of constraints in the high-level formalism, checking their consistency (and performing other time-related operations) in a correct, complete and tractable way. In particular:

(4) the consistency of the temporal constraints between classes and instances must be supported. This involves dealing with the *inheritance* of constraints (from classes to instances) and with the *predictive* role of constraints between classes.[1]

Obviously, the interplay between issues (1)—(4) needs to be dealt with, too. For example, the

[1] For example, given an execution of the composite action a in the guideline G of Example 2, one expects to have an instance of b at least 10 days after (an analogous problem has been faced in [46] concerning workflow systems).

interaction between composite and periodic events might be complex to represent and manage. In fact, in the case of a composite periodic event, the temporal pattern regards the components, which may, recursively, be composite and/or periodic events. For instance, consider Example 1. In Example 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 mieloma.

While some of the above issues have been treated in an ad-hoc way in the literature, in this paper we aim at devising a general module coping in an integrated way with all of them, providing a temporal knowledge server which acts as an independent module to which temporal problems in different clinical guidelines may be delegated.

The strategy we chose to adopt in order to achieve our goal is that of devising a *two-layer* approach:

(1) the high-level layer provides a high-level language to represent the above-mentioned temporal phenomena and to offer several temporal reasoning facilities;
(2) the low-level layer consists of an internal representation of the temporal constraints, on which temporal constraint propagation algorithms operate.

We designed our high-level language with specific attention to the modelling of repeated actions, and in such a way that tractable temporal reasoning can be supported. At the low-level layer, we chose to exploit as much as possible STP (Simple Temporal Problem [29]), a standard AI temporal reasoning framework. In a certain sense, our approach uses STP as an 'assembly language' and builds an expressive 'high-level temporal reasoning framework' on top of it.[2] Obviously, the gap between our high-level language and STP is very large. Filling such a gap is the main contribution of our approach, and has involved the design of suitable temporal reasoning algorithms to cope with issues (1)–(4) above, as well as an extension of the STP framework itself (to consider labelled trees of STPs—see Section 4.2).

[2] We thus believe that our contribution is analogous, in the temporal reasoning area, to the design of a new language in the programming area. New languages are finally compiled/interpreted into some 'standard' assembly language, but they are nevertheless introduced in order to provide high-level tools to make easier the work of programmers.

3. Representing temporal constraints in clinical guidelines

3.1. Representation formalism

Our high-level language allows one to express temporal constraints about durations and delays between actions. More specifically, we list below some of the primitives that our language supports.

Dates can be expressed by the predicate **date**(**A**, **L1**, **U1**, **L2**, **U2**), stating that the action A must start between dates $L1$ and $U1$ and end between dates $L2$ and $U2$. Precise dates can be expressed imposing $L1 = U1$ or $L2 = U2$. Please note that also unknown dates are allowed by imposing that the extremes assume value $-\infty$ or $+\infty$. Other constructs include the predicate **duration**(**A**, **L**, **U**), stating that the duration of action A must be included between L and U, **delay**(**P1**, **P2**, **L**, **U**), stating that the delay between $P1$ and $P2$ must be between L and U, where $P1$ and $P2$ are time points (i.e. starting or ending points of actions). Also qualitative temporal constraints such as ''before'', ''after'', ''during'' are supported by our language: in fact, it supports all and only the qualitative constraints that can be mapped to conjunctions of STP constraints [22] (further details on STP can be found in Section 4).

For example, the duration of $a2$ in Example 2 (''Action $a2$ lasts exactly 2 days'') can be expressed by the predicate $duration(a2, 2d, 2d)$; the relation between a and b in Example 2 (''b must start at least 10 days after the end of a'') can be represented by the constraint $delay(End(a), Start(b), 10d, +\infty)$; the relation between $a11$ and $a12$ (''$a11$ and $a12$ must be performed in parallel'') can be represented by the qualitative temporal constraint $equal(a11, a12)$.

For representing composite actions, we support the predicate **partOf**(**A′**, **A**), stating that the action A' is part of the composite action A. Please note that the *partOf* relation induces a temporal constraint between the actions: i.e., action A' must be *during* action A.

For example, in Example 2 we have: partOf($a1$, a), partOf($a2$, a), partOf($a11$, $a1$), partOf($a12$, $a1$), partOf($b1$, b).

The predicates described above can be also used for representing temporal constraints between instances of actions.

In order to describe the relation between instances and classes, we need to introduce a further predicate, **instanceOf**(**I**, **A**, **p**) to represent the fact that the instance of action I is an instance of the class of actions A. If A is a repeated action, then p represents the fact that I is an instance

```
Rep ::= repetition(A, RSpec)
A ::= class of action
RSpec ::= [Nrep, ITime, {RConstr}, {Cond}] {, Rspec}
Nrep ::= natural number
ITime ::= exact duration
RConstr ::= {fromStart(m,m)} {Bet} {toEnd(m,m)}
Bet ::= inBetweenAll(m,m) | inBetween((m,m), …,(m,m))
m ::= duration
Cond ::= while(C) | onlyIf(C)
C ::= boolean expression
```

Figure 1 Syntax in BNF of the *repetition* primitive. Curly brackets represent optionality.

of the *p*th repetition of *A* (if *A* is not a repeated action, $p = 0$). From Example 2A, we have *instanceOf*($a11_1$, $a11$, 1), *instanceOf*($a12_1$, $a12$, 1), *instanceOf* ($a11_3$, $a11$, 3), *instanceOf*($a12_3$, $a12$, 3), *instanceOf* ($a2_1$, $a2$, 1), *instanceOf*($b1_1$, $b1$, 1).

Regarding repetition of actions, we provide the predicate **repetition**(**A**, **RSpec**), to state that the (possibly composite) class of action *A* is repeated according to the parameter *RSpec*. Since this primitive represents an innovative feature of our language, we deal in detail with the predicate. Please note that the predicate *repetition* regards classes of actions only, since the instances of actions are observed separately and independently of each others at execution time.

The syntax of the predicate repetition is described in BNF in Fig. 1.

RSpec is a recursive structure of arbitrary (finite) depth, where each level R_i states that the actions described in the next level (i.e., R_{i+1}, or — by convention — the action *A*, if $i = n$) must be repeated a certain number of times in a certain time span. To be more specific, any basic element R_i consists of a quadruple [*Nrep*, *ITime*, {*RConstr*}, {*Cond*}], where the first term represents the number of times that R_{i+1} must be repeated, the second one represents the time span in which the repetitions must be included, the third one (which is optional) may impose a pattern that the repetitions must follow, and the last one (which is optional) allows one to express conditions that must hold so that the repetitions can take place. Informally, we can roughly describe the semantics of a quadruple R_i as the natural language sentence ''repeat R_{i+1} *Nrep* times in exactly *ITime*, if *Cond* holds''.

RConstr 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 ITime 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 ITime;
- **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**$_{\mathbf{Nrep-1}}$, **max**$_{\mathbf{Nrep-1}}$)) representing the (possibly imprecise) delays between each repetition and the subsequent one. Note that any couple ($\min_j$, $\max_j$) may be missing, to indicate that we do not impose any temporal constraint between the *j*th repetition and the $(j + 1)$th one.

Cond is a (possibly empty) set of conditions that influence the repetitions. The conditions may be of type:

- **while**(**C**), where *C* is a Boolean predicate. It states that, when *C* becomes false, the repetition ends;
- **onlyIf**(**C**), where *C* is a Boolean predicate. It states that, if *C* is true, the repetition may be performed and, if *C* is false, the repetition must not be performed and we can pass to the next repetition. This construct allows to skip single repetitions.

In order to make our approach to temporal constraints more user-friendly, a (possibly graphical) user interface could be used to acquire and represent temporal constraints (concerning both (i) dates, durations, delays and qualitative relations between non-repeated events and (ii) repetition/periodicity constraints) [30—33].

Before describing the semantics of the above primitives, we show how the periodicity constraints in Example 2 can be expressed by using the *repetition* construct. Action *b* (''Action *b* is repeated twice a week and each repetition must be at most 1 day after the previous one'') exemplifies the *RConstr* component: *repetition*(*b*, [2, 7*d*, *inBetweenAll*(0*d*,1*d*),]).

For representing action *a* (''Action *a* is repeated once every week for 2 weeks, until condition c_w does not hold anymore''), we can use a two-level specification: one level for representing the ''once

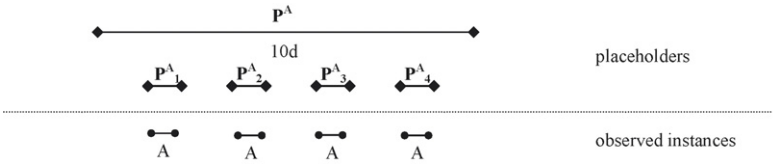

Figure 2 Extensional semantics for repetition(A, [4,10d],,). Diamond-shaped arrows represent placeholders and circle-shaped arrows represent instances of actions.

every week'' part and one for representing the ''for 2 weeks'' part; furthermore, the ''until condition c_w does not hold anymore'' part can be represented by a *while* condition as follows, *repetition*(a, [2, 14d,, *while*(c_w)], [1,7d,,]).

The repetition of action $a1$ (''$a1$ is repeated once a day for 3 days and each repetition is performed if the condition c_i holds'') can be represented by the following constraint: *repetition*($a1$, [3, 3d,, *onlyIf*(c_i)], [1,1d,,]).

In the following, we describe the semantics of the *repetition* construct by introducing progressively the various components of the construct. However, a logical semantics of our formalism, which might be used in order to logically prove the correctness and completeness of our approach (as we did in [34] for a fragment of our language about repetitions), is outside the scope of this paper.

For each component, a figure is given to intuitively describe the extensional semantics of the construct.

- **repetition**(**A**, [**Nrep**, **ITime**,,]) (see Fig. 2)
  (1) there exists a temporal interval (placeholder) P^A lasting ITime; there exist Nrep temporal intervals (placeholders) $P^A_1, \ldots, P^A_{Nrep}$ such that they are not overlapping and they are (not-strictly) during P^A;
  (2) for each P^A_i there exists an instance of the action A (more precisely, for each P^A_i before NOW).

Here and in the following, in case action A is not atomic, all the instances of its atomic components must be considered (from the algorithmic point of view, we manage the possible nesting of composite/repeated actions through the generation of placeholders; see Fig. 12).

- **repetition**(**A**, [**Nrep**, **ITime**, **RConstr**,]) (see Fig. 3)
  (1) (same as above);
  (2) (same as above);
  (3) (a) if fromStart(m, M) $\in$ RConstr, then the delay between the start of P^A and the start of P^A_1 must be between m and M;
  (b) if toEnd(m, M) $\in$ RConstr, then the delay between the end of P^A_{Nrep} and the end of P^A must be between m and M;
  (c) if inBetween((m_1, M_1), ..., (m_{Nrep-1}, M_{Nrep-1})) $\in$ RConstr, then the delay between the end of P^A_i and the start of P^A_{i+1} must be between m_i and M_i, $i = 1, \ldots,$ Nrep-1;
  (d) if inBetweenAll(m, M) $\in$ RConstr, then the delay between the end of P^A_i and the start of P^A_i must be between m and M, $i = 1, \ldots,$ Nrep-1.
- **repetition**(**A**, [**Nrep**, **ITime**,, **Cond** = **while C**]) (see Fig. 4)
  (1) (same as above);
  (2) let j be the maximum value such that $\forall k$, $1 \leq k \leq j \leq$ Nrep $C(k)$ holds. For each P^A_i, $1 \leq i \leq j$, there exists an instance of action A (where we indicate by $C(i)$ the fact that condition C holds at the time of P^A_i).

Here, placeholders are used to represent the frequency of the checks of conditions; e.g., in Fig. 4, we introduce the placeholders to state that the condition C has to be checked at most four times in 10 days. In general, we assume that the temporal location of placeholders is always specified (both for while and onlyIf) in the given and in the next level of the repetition specification. For example, with ''repetition(A, [4,10d, fromStart(1d,1d) toEnd(2d,2d), while C], [1,2d,,])'' we state that the while condition must be checked before each

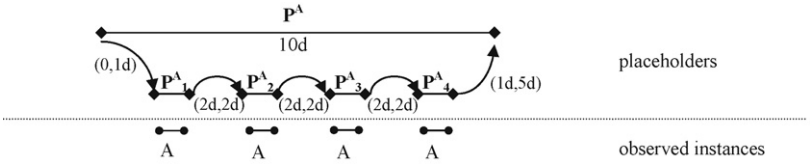

Figure 3 Extensional semantics for repetition(A, [4,10d, fromStart(0,1d) inBetweenAll(2d,2d) toEnd(1d,5d),]).

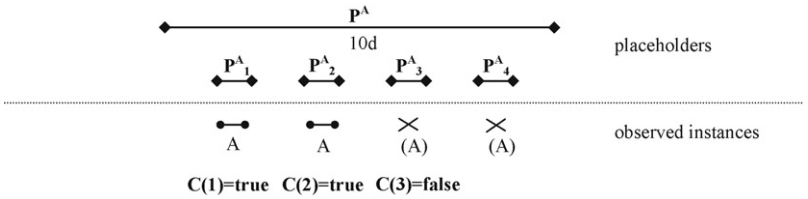

Figure 4 Extensional semantics for repetition(A, [4,10d,, while C]). A cross indicates a missing (and not observed) instance.

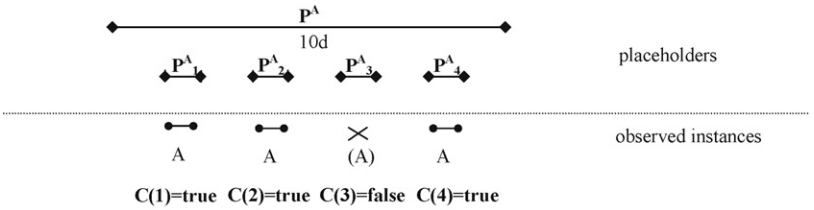

Figure 5 Extensional semantics for repetition(A, [4,10d,, onlyIf C]).

repetition, that is every 2 days for 10 days, i.e., on days 2, 4, 6 and 8.

Notice also that it is part of the intended semantics of our "while" construct the fact that the temporal constraints in the repetition must be such that, potentially (i.e., in the case the condition always holds), all the *Nrep* repetitions must be executable (see also footnote 3 in the following). In some sense, placeholders can be seen as a technical device we had to introduce in the semantics (and in the reasoning algorithms; see Section 4) in order to enforce such an intended semantics.

- **repetition**(**A**, [**Nrep**, **ITime**,, **Cond** = **onlyIf C**]) (see Fig. 5)
  (1) (same as above);
  (2) $\forall i$, $1 \leq i \leq$ Nrep if $C(i)$ holds, then there exists an instance of action A.

The semantics of the construct that has both the *RConstr* component and the *Cond* component can be compositionally obtained by combining the semantics of *RConstr* and *Cond*.

On the other hand, the semantics of an $(n+1)$-level repetition ($n \geq 1$) of the form *repetition*(A, $RSpec^n$, ..., $RSpec^1$) can be provided on the basis of the semantics of its nested n-level repetition as follows.

- **repetition**(**A**, **RSpec**^{n}, ..., **RSpec**^{1})
  (1) at level $n+1$, there exist a temporal interval (placeholder) P^A lasting ITime^{n+1} and Nrep^{n+1} temporal intervals (placeholders) $P^A_1, \ldots, P^A_{\text{Nrep}^{n+1}}$ such that they are not overlapping and they are (not strictly) during P^A;
  (2) the placeholders P^A at level n coincide with the placeholders P^A_i at level $n+1$;
  (3) if the $(n+1)$th level contains an *onlyIf* or *while* condition, its semantics is the same as above, apart of the fact that only "legal" placeholders at levels from n to 1 are used, i.e. those placeholders that have not been excluded by conditions at the previous (from n down to 1) levels, if any.
  (4) the instances have to be placed in the "legal" placeholders at level $n+1$ corresponding to the "legal" placeholders at level n.

Notice that placeholders are of use for assuring that, even with conditioned repetitions, the temporal constraints allow to execute all the repetitions.[3]

As an example, in Fig. 6 we show the extensional semantics of the constraints in Example 2. The figure shows all the details, including the "$\equiv$" symbol to denote the identity of placeholders at different nesting levels (see the semantics of repetition(A, $RSpec^n$, ..., $RSpec^1$) at point (2)). To make the example concrete, we suppose that, as in Example 2A, the condition of while (i.e., C_w) holds in the first iteration of a and does not hold in the second iteration; furthermore, we suppose that the condition of onlyIf (i.e., C_i) does not hold in the second iteration of $a1$. Moreover, we also suppose that NOW occurs after the first repetition of b.

[3] For example, the periodicity constraint repetition(A, [10, 30d, , while(C)], [1, 100d, ,]), where the higher level lasts 30 days and the lower nesting level lasts 100 days, is not allowed. In fact, even if it would be possible that it has a consistent execution (e.g., if the condition C is false before the first iteration), it is more intuitive for the user that we signal this constraint as inconsistent, because it is not possible to execute all the 30 repetitions.

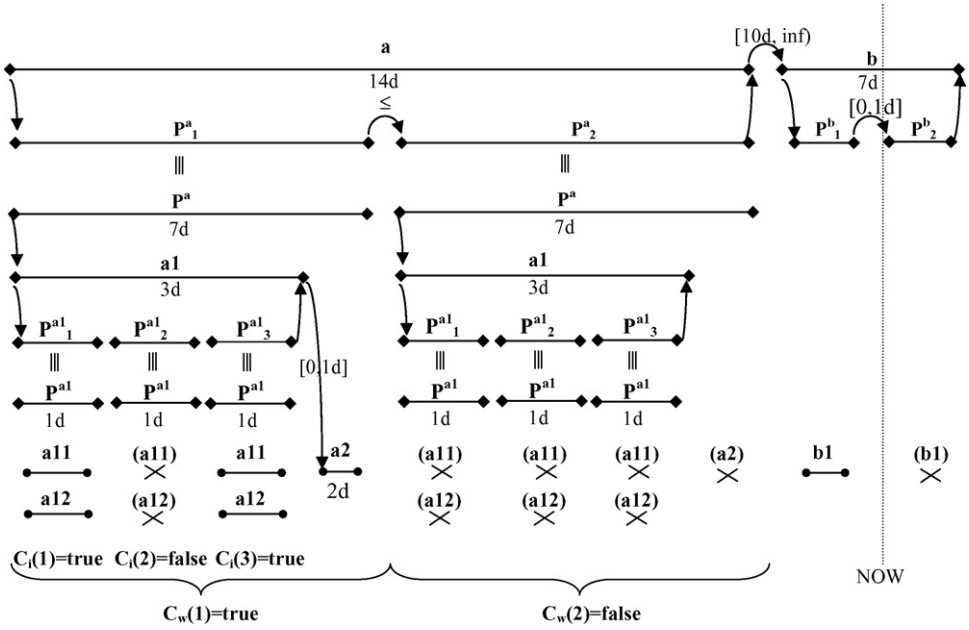

Figure 6 Extensional semantics for Example 2 (given the assumptions in Example 2A). The second iteration of *b*1 is not yet observed because it will possibly occur in the future (i.e., after NOW).

3.2. Expressiveness

In this subsection we analyse the expressiveness of our language about repetitions. We consider as a reference the classification criteria provided in [35] and in [36]. Before starting the analysis, it is worth remembering that both the above approaches are devoted to deal mainly with infinite periodical events (called *periodicities* in [35] and *periodical granularities* in [36]), while in our approach we deal with repeated events that are always finite (actually, infinite repetitions are not useful/possible in the clinical practice). In the following, we thus classify our formalism considering the ''patterns of repetitions'' it can model, and ignoring the fact that such patterns are finite.

Egidi and Terenziani [35] have defined the basic class of calendars (e.g., minutes) as elementary partitions of the timeline, and, on the basis of the expressiveness of many approaches in the literature, have identified five additional *orthogonal* properties of periodicities (none of which holds on calendars):

- Non-adjacency, i.e., intuitively, the fact that there are gaps *between* the denoted time intervals (e.g., Mondays).
- Gap, i.e. the fact that there are gaps *within* the denoted time intervals (e.g., Mondays work shifts; for example, from 9 to 12 and from 14 to 18, considered as a unique interval with a gap in it).
- Overlaps, i.e., the fact that the denoted time interval may overlap in time (e.g., the union of Tom's and Mary's working shifts—supposing they overlap in time).
- Eventual periodicity, i.e., the fact that the repetition cannot be represented as a purely periodic pattern, but also contains an aperiodic part (e.g., Monday 3 Jan 2005, plus Wednesday 5 Jan 2005, plus Mondays work shifts).
- Structure, i.e., the fact that the intervals can be grouped into periodic structures (e.g., work-shifts grouped by months and years).

In [37] a formal account, in terms of Presburger Arithmetics, of the expressiveness of calendars and of each property has also been provided. Our language can express calendars (or, more precisely, finite portions of calendars), and, more generally, repetition patterns for which the following properties hold:

- Non-adjacency; in particular, we can also state explicit metric constraints about the distance between different repetitions.
- Structure; in particular, we admit nested repetitions (i.e., repeated patterns that are subpart of higher-level periodic patterns).

On the other hand, the property of admitting eventually periodic pattern is not applicable to our approach (since our patterns are always finite) and we provide no construct to deal with patterns having ''Gaps''. As concerns ''Overlaps'', we admit

that different actions in the same repetition pattern overlap in time, but we do not provide any construct to define a single (possibly nested) repeating pattern with overlaps.

Recently, Bettini [36,38] has proposed a mathematical characterisation of granularity, aiming at providing a standard reference in the area of Temporal Databases. Since Bettini's granularities are a proper superset of *periodic granularities*, they constitute a natural reference to evaluate the expressiveness of our language. Bettini et al. [38] defined granularities as mappings from integers to subsets of the time domains. They defined *periodical granularities* (which are those granularities which are periodical with respect to the basic granularity) and classified them on the basis of two main parameters: boundedness of the extensions and the "Gap" property [36]. As discussed above, our formalism does not deal with intervals with gaps, and can only cope with finite (bounded) extensions.

4. Reasoning with temporal constraints in clinical guidelines

Regarding the instances of actions, we designed the high-level language in such a way that all constraints can be mapped onto bounds on differences and, thus, internally represented as a 'standard' STP framework (see Section 4.1).

However, regarding the classes of events, while dates, delays, durations and qualitative temporal constraints might be represented with an STP about classes, it is not possible to represent in such a basic way also the temporal constraints about repeated/periodic and/or composite actions.

In fact, for the reasons given in Section 2, an extensional representation is not feasible. On the other hand, an intensional approach cannot be directly implemented in "classical" temporal constraint propagation approaches (such as, e.g., STP), which need an explicit (i.e., "extensional") representation of all the temporal entities (time points and/or time intervals). For instance, the temporal constraint in Example 2 stating that the start of the repeated action b must be at least 10 days after the end of the repeated action a is actually a constraint between the start of the first repetition of b and the end of the last repetition of a. Furthermore, the components of the specification of the periodicity of repeated events (e.g., *inBetween*, *inBetweenAll*) are actually constraints on the single repetitions (rather than on the repeated event as a whole). But, in the intensional approach, one does not want an explicit representation of each single repetition.

Moreover, as discussed in Section 2.3, when we check the consistency of the instances of events, we wish to check not only that they are consistent with each other, but also that they respect the temporal constraints in the guideline. The problem consists in checking whether the instances are located in time in such a way that they respect the temporal constraints imposed by their related classes. Therefore, the temporal constraints in the guideline must be "inherited" on the instances, and we have to check that all the temporal constraints (the ones about instances and the inherited ones) are consistent.

The reasoning mechanisms have also to take into account the predictive role of the classes. For instance, if we are executing the guideline of Example 2 and we observe an instance of the action $a1$, then we expect to have an instance of action $a2$ in at most 1 day. If we do not observe such an instance, we must signal to the user that an instance is missing, because this may indicate an inconsistency. The problem is more complex if we take into account that only events occurring before the time when the temporal reasoner is activated (i.e., NOW) can actually be observed. Thus, if the temporal constraints impose that the missing instance could start after NOW, not having observed it does not raise an inconsistency, because it could be observed in the future.

Dealing with composite actions involves some side effects. For example, given the guideline in Example 1 in Section 2.3, it may happen that we could not observe any instance of the action "mieloma treatment" in the clinical record of a patient, but just instances of the atomic actions "administering melphalan" and "administering prednisone". In a similar way, in Example 2 one could observe just the instances of the atomic actions $a11$, $a12$, $a2$ and $b1$ and not instances of the composite actions a, $a1$ and b. Nevertheless, temporal constraints between composite actions must be taken into account by the temporal reasoner. For example, in Example 2, the constraint that action b must start at least 10 days after action a is actually a constraint between the composite actions and not between their atomic actions. Therefore, the reasoning mechanisms must be able to map the temporal constraints on composite actions onto their composing actions.

4.1. STP

As mentioned above, we have chosen to model the temporal constraints concerning "standard" (i.e., non-repeated) actions in the guidelines, using the well-known and widely used AI framework, STP [29].

In STP, a set of constraints is modelled 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 can provide the *minimal network* of the constraints as output (i.e., the minimum and maximum distance between each pair of points) [29]. The minimal network can be computed by an "all-to-all shortest paths" algorithm such as the Floyd-Warshall's one, and it can also be used for efficient query answering [39].

The STP framework can 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 [26,40]). Moreover, also the constraint that the temporal extent of a composite action contains the extents of its components can be trivially modelled in STP (so that, it can easily deal with the temporal constraints involved by the hierarchical representation of guideline actions—i.e., issue (2) in Section 2.3; see also [19]).

4.2. Data structures for temporal constraints about repeated actions

In order to devise tractable, correct and complete temporal reasoning algorithms, an important step is the definition of suitable data structures to model a set of temporal constraints. STP provides suitable data structures for bounds on differences, which can be modelled as graphs on which the well-known Floyd-Warshall's algorithm operates to check consistency. However, as discussed at the beginning of this section, the STP framework is not expressive enough to cope with repeated/periodic actions. Thus, we have chosen to model the constraints regarding repeated actions into separate STPs, one for each repeated action. In our approach, the overall set of constraints between actions in the guideline is represented by a tree of STPs (STP-tree henceforth). The root of the tree is the STP which represents the constraints between all the actions in the guideline, except the components of repeated actions. For example, in Fig. 7, we show the STP-tree representing the temporal constraints involved by Example 2 in Section 2.3. In the figure, the root of the STP-tree, $N1$, contains the action representing the entire guideline.

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 is labelled with the list of properties describing the temporal constraints on the repetitions (i.e., *RSpec*). In Fig. 7, for example, the repeated actions composing the guideline (i.e., the actions composing a and b) are represented in separated STPs, children nodes of the actions a and b.

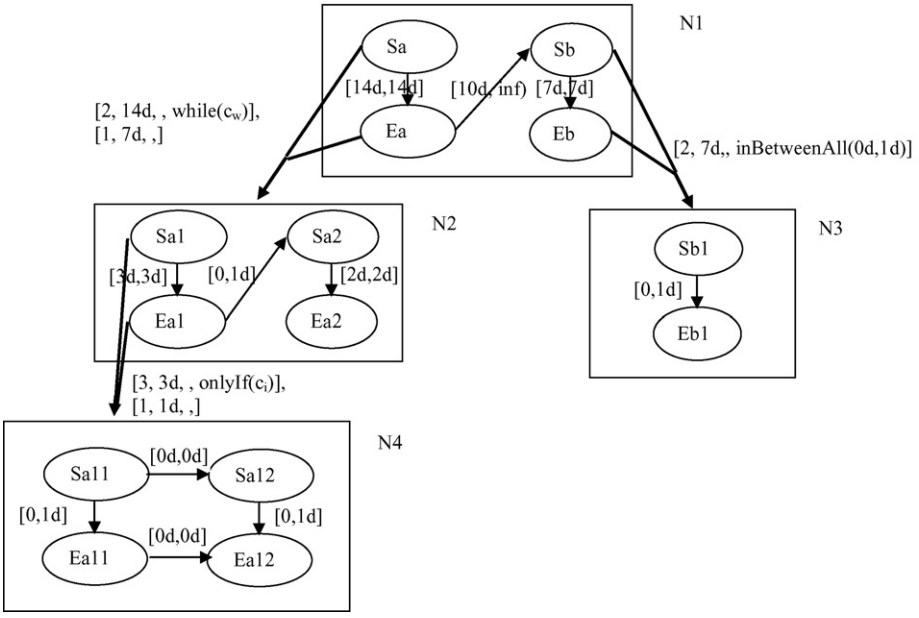

Figure 7 STP-tree for Example 2. Sx and Ex stand for the starting and ending points of action x, respectively.

```
procedure generateSTPTree(GL : Guideline) : STP-tree
    1. T ← generate_root(GL)
    2. for each repeated action A
    3.   generateRepetition(T, A, GL)
    4. return T

procedure generateRoot(GL : Guideline) : STP-tree
    1. generate an empty STP-tree T
    2. generate N, the root STP-node of T
    3. put in N all the atomic and composite actions and, transitively, all their components except the components of repeated
       actions
    4. put in N the temporal constraints between the actions it contains
    5. put in N the temporal constraints relative to part-of relations
    6. return T

procedure generateRepetition(T : STP-tree, A : Action, GL : Guideline)
    1. generate N', STP-node child of A
    2. put on the edge from A to N' the repetition specification RSpec
    3. put in N' all the atomic actions and the composite actions A such that partOf(A', A) and, transitively, all their parts
       except the components of repeated actions
    4. put in N' the temporal constraints between the actions it contains
    5. put in N' the temporal constraints relative to partOf relations
    6. for each repeated action A'
    7.   generateRepetition(T, A', GL)
```

Figure 8 Algorithm for generating the STP-tree of temporal constraints in a guideline.

4.2.1. Mapping high-level temporal constraints onto an STP-tree

The STP-tree corresponding to a guideline can be automatically constructed on the basis of the temporal constraints in the guideline (expressed using the high-level language in Section 3) by executing an algorithm such as the one sketched in Fig. 8.

The algorithm recursively constructs the STP-tree, from the root to the leaves, by putting in each STP-node all the actions except the components of repeated actions, which are represented in separate STP-nodes. On the other hand, the partOf relations that do not involve repeated actions are represented in the same STP as the composite action by adding to such an STP-node the constraints that all the components are contained into the corresponding composite action. To summarize, in the STP-tree there are as many STP-nodes as the number of repeated actions, and in each STP-node there are as many actions as the number of actions in the guideline that are parts of the repeated action that the STP-node represents. Specifically, each action is represented in the STP-node as a pair of time points, while constraints between (not repeated) actions are represented by arcs connecting them (see Fig. 7).

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 executions of the guidelines on specific patients. In Fig. 9 the STP for the instances of Example 2A is represented. The temporal point $X0$ is the reference point and in this case it represents the midnight of the day when the execution of the guideline started. For example, the two edges labelled "$[32h,33h]$" between $X0$ and the two endpoints of $a11_1$ represent the fact that $a11_1$ is (entirely) executed between 8 and 9 a.m. of the second day; the two edges labelled "[0,0]" between the endpoints of $a11_1$ and $a12_1$ represent the fact that the two events are performed at the same time.

4.3. Checking the consistency of a guideline

Given an STP-tree, it is possible to check its consistency in an intensional way, i.e., without gener-

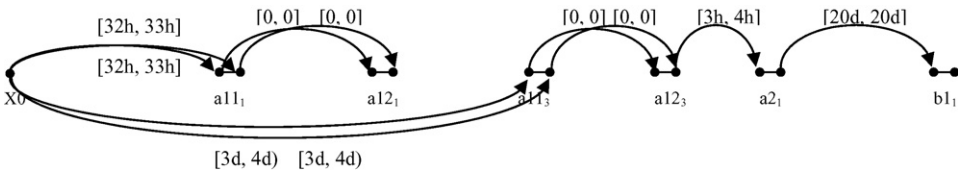

Figure 9 STP of the instances for Example 2A.

```
procedure STP_tree_consistency(X: STP-node, RSpec)
    1.  if tail(RSpec) = NULL then
    2.    return checkSTPContainment(X, head(RSpec))
    3.  else
    4.    checkLevelContainment(head(RSpec), tail(RSpec))
    5.    return STP_tree_consistency(X, tail(RSpec))

procedure checkLevelContainment(currR = [Nrep, ITime, RConstr, Cond], tailRSpec)
    1.  nextR = [nextNrep, nextITime, nextRConstr, nextcond] ← head(tailRSpec)
    2.  minLevelDuration ← NRep * nextITime + RConstr.fromStart.min + Σ_i RConstr.inBetween[j].min + RConstr.toEnd.min
    3.  if ITime < minLevelDuration then return INCONSISTENT

procedure checkSTPContainment(X: STPNode, currR = <Nrep, ITime, RConstr, Cond>)
    1.  FloydWarshall(X)
    2.  if X = INCONSISTENT then return INCONSISTENT
    3.  let minDurationSingleRepetition the maximum among the minimum distances between all pairs of temporal points in X
    4.  maxDurationSingleRepetition ← ITime – ((NRep – 1) * minDurationSingleRepetition + RConstr.fromStart.min +
        Σ_i RConstr.inBetween[j].min + RConstr.toEnd.min)
    5.  impose in X that the maximum distance between each pair of points is the minimum between the current maximum
        distance and maxDurationSingleRepetition
    6.  FloydWarshall(X)
    7.  if X = INCONSISTENT then return INCONSISTENT else return X
```

Figure 10 Algorithm for checking the consistency of a guideline (represented as an STP-tree).

ating every repetition of repeated actions. However, it is not sufficient to check the consistency of each STP contained in the STP-nodes 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 towards its leaves. Basically, the root contains a ''standard'' STP, so that the Floyd-Warshall's algorithm can be applied to check its consistency. Thereafter, for each node *X* in the STP-tree (except the root), we proceed as shown in the algorithm *STP_tree_consistency* in Fig. 10.

STP_tree_consistency takes in input the STP-node that must be checked (i.e. *X*) and the repetition/periodicity specification (i.e., the label of the arc of the STP-tree entering STP-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 constraint. The procedure recursively calls itself (step 5) on each level of the repetition specification. In the algorithm, we represent a recursive *n*-level specification *RSpec* as a list of n elements and *head* and *tail* operators select the first element (i.e., the outmost level) and the rest of the elements, respectively. For each level, it checks whether the next level can be contained in the current one (step 4 and procedure *CheckLevelContainment*) until it reaches the last level (step 1), where it checks whether the STP-node can be repeated as many times as imposed by the constraint itself (step 2 and procedure *checkSTPContainment*).

In procedure *CheckLevelContainment*, we compute the minimum possible duration of all the repetitions (steps 1—2) and we check whether it is consistent with the *ITime* of the level (step 3). Please note that, since we impose that *ITime*s are exact durations, they cannot change and we do not need to propagate new information to higher levels and to parent STP-nodes (see Property 1 below). To compute the minimum duration of the repetitions, we have to take into account that the *ITime* of the next level will be repeated the number of times indicated in the level (i.e., *NRep* times).

In procedure *CheckSTPContainment*, we compute (steps 1—4) the maximum distance between any pair of points that still allows to perform *NRep* repetitions. In order to do so, we consider that *NRep*-1 repetitions will have the minimum possible duration and that all the remaining time (with respect to the *ITime* in the specification) will be assigned to the remaining repetition. Therefore, in step 1 we propagate the STP constraints to obtain the minimal network and the minimum (and maximum) distances between the points in the STP. In step 3 we consider the minimum duration of the STP (i.e., the maximum among the minimum distances among any pairs of points in the STP) and in step 4 we compute the maximum duration of a repetition. Then, in step 5, we add to the STP *X* the new constraint that the STP can last at most *maxDurationSingleRepetition*, and we propagate the constraints (step 6).

Example 3. The execution of the algorithm on the node *N*2 of the STP-tree in Fig. 7 results in the call of

the procedure *STP_tree_consistency*(*N*2, [2,14*d*,, *while*(c_w)], [1,7*d*,,]).

In the first step of recursion on RSpec (the second parameter), procedure *CheckLevelContainment*([2,14*d*,, *while*(c_w)], [1,7*d*,,]) is invoked. *minLevelDuration* is $2 \cdot 7d$, that is 14 days, which is not less than 14*d*, the ITime of the first level.

In the second and last step of recursion, *checkSTPContainment*(*N*2, [1,7*d*,,]) is invoked. After the propagation of the constraints in *N*2, we have, among the other constraints, that *Sa*1 is from 5 to 6 days after *Ea*2. Since "5 days" is the maximum among the minimum distances between the points in *N*2, *minDurationSingleRepetition* is 5 days. *maxDurationSingleRepetition* is $7d-(1-1)\cdot 5d = 7$ days. Adding the additional constraint that the maximum distance between any two points of *N*2 must not be higher than 7 days does not alter *N*2, because the constraints already existing are stricter. Therefore, in this case, no new constraint is inferred by the propagation in step 6.

Complexity. Let *L* be the maximum number of nesting levels in a periodicity constraint and *C* the number of classes in the STP-tree (i.e., the number of actions in the guideline). Since the complexity of the Floyd-Warshalls's algorithm (in steps 1 and 6 of procedure *checkSTPContainment*) is cubic on the number of the points, the complexity of procedure *STP_tree_consistency* is $O(\max\{L, C^3\})$.

Considering that usually the number of nesting levels is less than the number of classes, we have that the complexity of the procedure is $O(C^3)$.

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

Proof (*sketch*). The correctness of our algorithm comes from the fact that there are only two types of constraints that are inferred by the algorithms:

(i) the constraints explicitly added to STP in step 5 of *checkSTPContainment*;
(ii) the constraints inferred through constraint propagation via the Floyd-Warshall's algorithm.

The former are trivially correct, since they simply re-state in terms of bounds on differences the fact that the maximum duration of each single repetition cannot exceed *maxDurationSingleRepetition*, i.e., the maximum duration as derived from the labelled arc.

As regards the latter, Floyd-Warshall's algorithm has been proved to be correct (and complete) on bounds on differences.

Also the proof of completeness relies of the properties of Floyd-Warshall's algorithm (namely, the fact that it is complete on bounds on differences). Therefore, completeness can be proved by showing that:

(i) there is no need to propagate back and forth constraints along the STP-tree, but it is enough to operate locally on each pair $<$constraints on the labelled input arc$-$constraints in the STP$>$;
(ii) all the "intensional" constraints on the labelled arc are mapped onto STP constraints and added into the STP-node before the application of Floyd-Warshall's algorithm.

In turn, the proof of issue (i) can be split into two parts.

First of all, one has to show that the constraints into an STP cannot modify the constraints on the corresponding arc. The syntax of constraints on repetition has been deliberately limited in such a way that such a condition always holds. In fact, constraint propagation in an STP could potentially modify the global duration of a repetition, thus interacting with the constraint (about the global duration) on the arc. However, this is excluded by the fact that we only admit *exact* values for such duration on the arc (the *ITime* component of the specification is just an exact number); thus, the duration, as inferred from the STP, may be either consistent or inconsistent with it, but can never modify it (e.g., restrict it, as in the common case of propagation on bounds on differences).

Second, we have to prove that there is no propagation (back and forth) on constraints about the duration of *ITime*s between different levels in the (recursive) specification of repetition on an arc of the STP-tree. This is trivially true, since *ITime*s at each level are exact.

Therefore, since durations on arcs cannot be modified, each STP-node can be treated independently of the others.

Moreover, notice that the algorithm is complete in checking possible inconsistencies between the different levels: steps 2 and 3 in *checkLevelContainment* achieve such a goal, by evaluating the minimal duration of a level of specification, and check that it can be contained into the *ITime* of the upper level.

As regards point (ii), it should be noticed that the constraints on the arc impact those in the STP just as regards the maximum duration of each repetition (remember that the STP-node "intensionally" represents the "prototype" of each repetition). Step 5 in *checkSTPContainment* adds such a constraint (notice that, since, in general, the starting

```
procedure integratedConsistency(T : STP-tree, E : executionSTP, NOW)
    1. visitSTPTree(root(T), E, NULL)
    2. futureInstances ← placeAndCheckInstances(T, E, NOW)
    3. inherit(T, E)
    4. propagate(E)
    5. checkFuture(E, NOW, futureInstances)
```

Figure 11 Algorithm for checking the consistency of an execution of a guideline.

and ending actions represented in an STP might be unknown, the algorithm simply adds the maximum duration constraint on each pair of actions). □

Notice, however, that the above reasoning mechanism does not provide the minimal network between all the actions in the STP-tree. Finally, even if not explicitly stated, the algorithm takes advantage also of a conversion table mapping each input granularity provided to the user (currently, we do not admit user-defined granularities [24,25]) into the basic granularity.

4.4. Reasoning with the executions of the guideline

In Figs. 11, 12, 14, 16–18 it is reported an algorithm for checking the consistency of the execution of a guideline with respect to its related guideline. In our work, as in most approaches to clinical guidelines, we suppose that one has *full observability* of instances (i.e., all the instances of actions which have been executed have been observed and inserted into the knowledge base), and that, for each instance, one knows the corresponding class of actions and/or repetition in the guidelines.

```
// Inherit the periodicity/repetition constraints by visiting the STP-tree
procedure visitSTPTree(X: STP-node, E : executionSTP, Parent : placeholder)
    1. let p the number of times that the STP-node X is repeated
    2. for each repeated action A in X do
    3.    add provisionally to E the placeholders P^Parent_1(A), …, P^Parent_p(A) representing the repetitions of action A
    4.    for each P^Parent_i do
    5.       add to E the constraints that P^Parent_i is (non-strictly) during Parent (only if Parent ≠ NULL)
    6.       let RSpec the repetition specification of A in T
    7.       inheritPeriodicityConstraints(E, RSpec, P^Parent_i)
    8.       visitSTPTree(child(X, A), E, P^Parent_i); od; od

// Recursively inherit the constraints relative to each level of the periodicity/repetition constraint
procedure inheritPeriodicityConstraints(E : executionSTP, RSpec, Parent: placeholder)
    1. if RSpec ≠ NULL then
    2.    R = [Nrep, ITime, RConstr, Cond] ← head(RSpec)
    3.    add to E the constraint that Parent lasts exactly ITime
    4.    add provisionally to E the placeholders P^Parent_1, …, P^Parent_Nrep
    5.    for each P^Parent_k do
    6.       inheritPeriodicityConstraints(E, tail(RSpec), P^Parent_k)
    7.       add to E the constraints that P^Parent_k is (non-strictly) during Parent
    8.       add to E the constraint that P^Parent_k is after the previous repetition P^Parent_k-1 ; od
    9.    inheritRConstr(E, RConstr, Parent, <P^Parent_1, …, P^Parent_Nrep>)

// Add the constraints related with RConstr
procedure inheritRConstr(E : executionSTP, RConstr, Parent, <P^Parent_1, …, P^Parent_Nrep>)
    1. if fromStart(m, M) ∈ RConstr then
    2.    add to E the constraint that the delay between the start of Parent and P^Parent_1 is between m and M
    3. if toEnd(m, M) ∈ RConstr then
    4.    add to E the constraint that the delay between the end of P^Parent_Nrep and Parent is between m and M
    5. if inBetweenAll(m, M) ∈ RConstr then
    6.    add to E the constraints that the delay between the end of P^Parent_k and the start of P^Parent_k+1 is between m and
       M
    7. if inBetween((m_1, M_1), …, (m_Nrep-1, M_Nrep-1)) ∈ RConstr then
    8.    add to E the constraints that the delay between the end of P^Parent_k and the start of P^Parent_k+1 is between m_k
       and M_k
```

Figure 12 Algorithm for visiting the STP-tree and for generating the placeholders.

The procedure *integratedConsistency* (see Fig. 11) accepts three parameters: *T* is the STP-tree that describes the constraints about classes of actions in the guideline, *E* the STP that describes the temporal constraints between the instances of actions (i.e., the actions that have been executed on specific patients), and *NOW*, that corresponds to the time of the present. The basic idea is to:

1. visit the STP-tree and place the placeholders representing the repetitions (see the semantics of periodicity constraints in Section 3.1) (step 1);
2. place the instances in the proper placeholders and check whether the observed instances correspond with the expected instances (according to the STP-tree) (step 2). At this point, expected but not observed instances are hypothesized to occur in the future. Please note that steps 1 and 2 jointly inherit the periodicity constraints;
3. inherit the non-periodic temporal constraints from classes to instances (step 3);
4. propagate the constraints and check whether they are consistent (step 4);
5. check whether the missing instances may actually have not been observed because they will start in the future (step 5).

Let us go into details of each point.

4.4.1. Visit the STP-tree

Procedures *visitSTPTree*, *inheritPeriodicityConstraints* and *inheritRConstr* (see Fig. 12) jointly visit the STP-tree and make the periodicity constraints explicit by adding the placeholders representing the repetitions (as stated in the semantics of periodicity constraints—see Section 3.1).

Procedure *visitSTPTree* accepts as parameters an STP-node (*X*), the STP of the instances (*E*) and the placeholder that represents the possible repetition which contains *X* (Parent) (if *X* is the root, Parent is NULL). In step 1 it computes the number of times that the STP-node is repeated (given the repetition specification labelling its entering arc). Then, for each repeated action *A* in *X* (step 2), it adds the placeholders representing the actions (step 3), the constraints of containment in the parent placeholder (step 5), then it invokes the procedure *inheritPeriodicityConstraint*, which adds to *E* the placeholders required by the periodicity constraint on the arc entering *A* (step 7). Finally, step 7 performs the recursive call on the child STP-node. Procedure *inheritPeriodicityConstraints* accepts as parameters *E*, the STP that contains the instances, *RSpec*, the repetition specification of a repeated action, and *Parent*, the placeholder that contains the entire repetition. It recursively adds the proper placeholders of each level of *RSpec* (see the description of the periodicity constraints in Section 3.1). More specifically, in step 1 we check whether we have not reached the base case; in this case, we extract from *RSpec* the top level (i.e., the quadruple [*Nrep*, *ITime*, *Rconstr*, *Cond*]). In step 3 we add the temporal constraint that *Parent* must last exactly *ITime* (see point (1) of the semantics of periodicity constraints in Section 3.1), in step 4 we add the *Nrep* placeholders (see, again, point (1)) in the semantics and, for each one (step 5), we call recursively the procedure on the remaining levels (step 6). Then, we add the constraints that the *Nrep* placeholders are contained in *Parent* (step 7) and that they must not overlap (step 8) (see again, point (1) in the semantics). Finally, we call procedure *inheritRConstr* that adds the temporal constraints implied by *RConstr* on the placeholders (see point (3) in the semantics of "repetition(*A*, [*Nrep*, *ITime*, *Rconstr*,])" in Section 3.1).

If we execute step 1 of procedure *integratedConsistency* on Example 2, we obtain the placeholders represented in the higher part of Fig. 6 (reported again here in Fig. 13 for the sake of clarity) and the related temporal constraints.

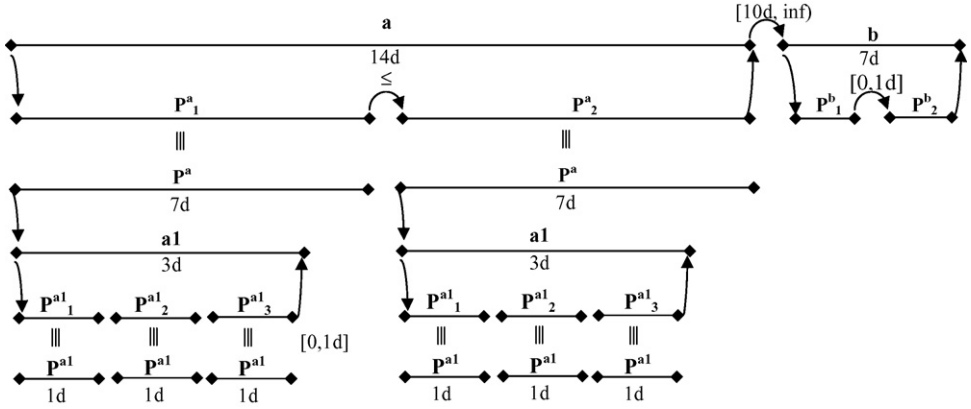

Figure 13 Placeholders obtained after step 1 of procedure *integratedConsistency* on Example 2.

```
// Place the instances in the proper placeholder and check whether the observed instances in E correspond to the
expected instances (according to T)
procedure placeAndCheckInstances(T : STP-tree, E : executionSTP, NOW) : SetOfFutureInstances
    1.  forall instances i ∈ E such that instanceOf(i, A, n) holds do
    2.    add to E the constraint that i is contained into the placeholder corresponding to the n^th repetition of action
          A
    3.    let P^Parent_i(i) such a placeholder (if n=0, i does not belong to a repeated action and P^Parent_i(i) = NULL) ; od
    4.  futureInstances ← ∅
    5.  for each instance i expected according to T but missing from E do
    6.    if checkCond(T, E, P^Parent_i(i)) = FAIL then
    7.       hypothesize i by provisionally inserting it in E
    8.       futureInstances ← futureInstances ∪ {i} ; od
    9.  forall instances i ∈ E – futureInstances do
    10.   add to E the constraint that i starts before NOW
    11. return futureInstances

// Check whether a missing instance of a repeated action is consistent with regards to the conditions in the repetition
specifications
procedure checkCond(T : STP-tree, E : executionSTP, P^Parent_i : placeholder)
    1.  if P^Parent_i = NULL then return FAIL
    2.  let R = <Nrep, ITime, RConstr, Cond> the level of the repetition specification relative to P^Parent_i
    3.  if Cond = onlyIf(C) then
    4.    if there are no instances in E belonging to P^Parent_i then return SUCCESS
    5.  if Cond = while(C) then
    6.    if there is no instance in E belonging to P^Parent_i and there is no instance in E belonging to a following
          repetition P^Parent_i, j>i then return SUCCESS
    7.  return checkCond(T, E, Parent)
```

Figure 14 Algorithm for placing the instances in the placeholders and for checking whether the missing instances are consistent with the conditioned repetitions.

4.4.2. Place and check instances

At this point, we have added to the STP containing the instances of the actions all the placeholders implied by the semantics of the periodicity constraints. The following step is performed by procedure *placeAndCheckInstances*.

In procedure *placeAndCheckInstances* (see Fig. 14), for each instance of a repeated action, we add the constraint that the instance is contained in the proper placeholder (located by using the instanceOf relation). Then (steps 4—11), we check whether all the instances that the STP-tree provides to exist have actually been observed. To perform this task, for each missing instance (step 5), we check whether its absence is justified by some conditioned repetition (step 6). If it is not the case, we hypothesize the missing instance by provisionally inserting it in the STP about instances and in the set *futureInstances*; afterwards (in procedure *checkFuture*), we will check if the instance is missing because it might occur in the future. In steps 9 and 10, we add the constraints that all the observed instances have been started before NOW.

Let us go into details of step 5. This check is performed by procedure *checkCond*. It accepts as parameters the STP-tree, the STP about instances and the placeholder P_i^{Parent} that has no corresponding instance, and returns *SUCCESS* if the absence of the instance if justifiable by a conditioned repetition, *FAIL* otherwise. The procedure basically goes back up on the levels of the repetition specifications and on the STP-nodes of the STP-tree until it reaches the root (see the recursive call on *Parent* in step 7 and the base case in step 1). More specifically, in the algorithm we suppose that each placeholder is labelled with the quadruple [*Nrep*, *ITime*, *RConstr*, *Cond*] from the level of the repetition specification corresponding to the placeholder (see the level *R* in step 2 of procedure *inheritPeriodicityConstraints* and the placeholders generated in step 4 of the same procedure). (We have chosen to not explicitly label the placeholders with the quadruple in order to avoid a cumbersome notation.) If in the *Cond* parameter of the quadruple there is an onlyIf(*C*) condition (step 3), then we check (step 4) that there are no instances belonging to the same placeholder. In fact, if the condition *C* is false, then there cannot be any instance belonging to the repetition (see the semantics of onlyIf in Section 3.1). On the other hand, if in the *Cond* parameter there is a while(*C*) condition (step 5), then we check (step 6) that there are no instances belonging to the same placeholder and, moreover, no instance belonging to a following repetition (see the semantics of while in Section 3.1).

If we execute step 2 of procedure *integratedConsistency* on Example 2, we obtain the STP represented in Fig. 15. Regarding the missing instances, we detect that: the first missing occurrence of *a*11 and of *a*12 is compatible with the onlyIf condition in the periodicity constraint of *a*1, in fact both components of *a*1 are missing; the following missing occurrences of *a*11, *a*12 and *a*2 are compatible with

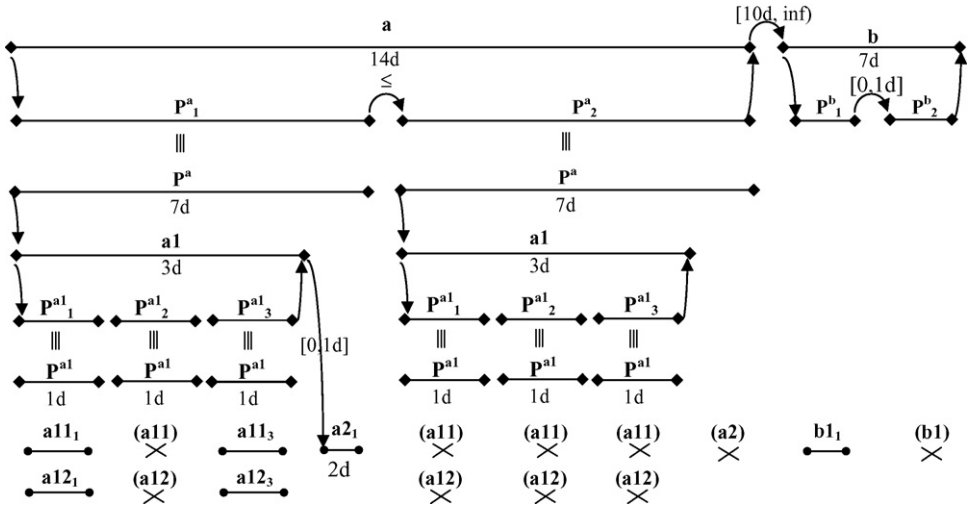

Figure 15 Placeholders obtained after step 2 of procedure *integratedConsistency* on Example 2. Note that, for the sake of clarity, not all temporal constraints are shown.

// Inherit the temporal non-periodic constraints
procedure inherit(T : STP-tree, E : executionSTP)
1. **forall** pairs of instances i, i'∈ E belonging to the same repetition **do**
2. let c, c' the classes corresponding to i and i' i.e., such that instanceOf(i, c, n) and instanceOf(i', c', n) hold
3. instantiate on i and i' in E the constraints in T between c and c' ; **od**
4. **forall** pairs of placeholders P^{Parent}, P'^{Parent} ∈ E belonging to the same repetition and representing the repetition of two repeated actions A and A' (see step 3 of procedure visitSTPTree) **do**
5. instantiate on P^{Parent} and P'^{Parent} the constraints in T between A and A'

Figure 16 Algorithm for inheriting the non-periodic temporal constraints.

the while condition in the periodicity constraint of *a*, in fact all the components of *a* are missing from the second repetition and there are no following repetitions; finally, the missing occurrence of *b*1 is not justifiable by any condition in the periodicity constraint of *b*, so we add *b*1 to *futureInstances*.

4.4.3. Inherit non-periodic constraints

The inheritance of non-periodic constraints is performed by procedure *inherit* (see Fig. 16). For each pair of instances (step 1), we instantiate the relative temporal constraints on classes (expressed as bounds on differences) (steps 2 and 3). Then (steps 4 and 5), we do the same for the placeholders, in order to instantiate also the temporal constraints from non-atomic actions.

For instance, in the example, the constraint that $a2_1$ can last at most 2 days (deriving from the class *a*2) is added to the instance.

4.4.4. Propagate constraints

In procedure *propagate* (see Fig. 17), we propagate the temporal constraints by using the well-known all-pairs shortest paths algorithm by Floyd and Warshall.

4.4.5. Check future instances

In procedure *checkFuture* (see Fig. 18), after the propagation of the constraints, we check whether the missing instances that were hypothesized in step 8 of procedure *placeAndCheckInstances* must necessarily start before NOW. In fact, in this case, we report an inconsistency because we have not observed a required instance.

In the example, we can detect that, if NOW has a value comprised between the end of the placeholder P^b_1 and the end of the placeholder representing the action *b* (as in Fig. 6), then the second occurrence of *b*1 can consistently start in the future.

// Propagate the constraints
procedure propagate(E : executionSTP)
1. FloydWarshall(E)
2. **if** E = INCONSISTENT **then return** INCONSISTENT

Figure 17 Algorithm for propagating the temporal constraints.

```
// Check whether the instances in futureInstances may actually start in the future
procedure checkFuture(E : executionSTP, NOW, futureInstances)
    1.  for each i ∈ futureInstances do
    2.    if NECESSARY(Start(i) before NOW) then return INCONSISTENT
```

Figure 18 Algorithm for checking whether the missing and hypothesized instances may actually start in the future.

Optimizations. It is not necessary to hypothesize all the missing instances, or to add all the placeholders of a repeated action. In fact, it is possible to exploit two optimizations:

(a) we hypothesize only the first missing repetition of an action (hypothesizing the following ones does not affect the consistency of the executionSTP [41]); and
(b) we generate the placeholders of the repetitions only as far as there is a related instance; when we reach the last instance in a repetition, we may stop generating placeholders.

For the sake of clarity and brevity, such optimizations are not explicitly shown in the algorithm.

Complexity. Let us denote with C the number of classes in the STP-tree, with I the number of instances in the executionSTP, with R the maximum number of times that an action is repeated and with L the maximum number of levels in periodicity specification.[4]

Since procedure *visitSTPTree* in Fig. 12 visits the tree and generates the placeholders by "unfolding" the repetitions, step 1 of procedure *integratedConsistency* in Fig. 11 is performable in a time $O(\max\{R \cdot C, I\})$. Exploiting optimization (b), we can reduce the time to $O(\max\{C, I\})$, because we have at most one repetition for each class without a corresponding instance. After this step, the executionSTP will include also $O(C)$ hypothesized instances and $O(I)$ placeholders, that is $O(I + C) = O(\max\{I, C\})$ instances all together.

Regarding step 2, steps 1—3 of procedure *placeAndCheckInstances* in Fig. 14 can be performed in a time $O(\max\{I, C\})$. Steps 5—8 are iterated $O(R \cdot C)$ times, but, exploiting optimization (b), we may reduce to $O(C)$ times. The call of *checkCond* takes $O(\max\{L, C\})$; therefore steps 5—8 take $O(\max\{L \cdot C, C^2\})$. Steps 9—11 of *placeAndCheckInstances* take $O(I)$ time, so that the entire procedure is performed in $O(\max\{I, L \cdot C, C^2\})$. Considering that usually the number of nesting levels is less than the number of classes, we have that the complexity of the procedure is $O(\max\{I, C^2\})$.

Step 3 of procedure *integratedConsistency* can be performed in a time $O(\max\{I^2, C^2\})$, because it iterates for each pair of instances (that are $O(I + C)$) and, then, for each pair of placeholders (that are $O(I)$).

Step 4 of procedure *integratedConsistency*, since Floyd-Warshall's algorithm is cubic on the number of points, can be performed in a time $O(\max\{I^3, C^3\})$.

Finally, step 5 of procedure *integratedConsistency*, exploiting the locality properties of STP constraints proved in [39], can be done in a time $O(C)$, since there are $O(C)$ instances in *futureInstances* and each check requires constant time.

Thus, the complexity of *integratedConsistency* procedure is $O(\max\{C, I\} + \max\{I, C^2\} + \max\{I^2, C^2\} + \max\{I^3, C^3\} + C) = O(\max\{C^3, I^3\})$.

Property 2. *The integratedConsistency procedure is correct and complete as regards consistency checking of the constraints in the executionSTP and in the STP-tree.*

Proof (*sketch*). The proof is based on the fact that all and only the temporal constraints specified in the STP-tree are inherited on the instances, that the semantics of the temporal constraints (including the repetition/periodicity constraints) is met, and that correct and complete temporal constraint propagation is performed via the all-pairs shortest paths algorithm. □

5. Exploiting temporal reasoning within Clinical Guidelines Systems

In Sections 3 and 4 we have proposed a principled approach coping with issues (1)—(4) in Section 2.3. In fact, it provides support for: qualitative and quantitative temporal constraints and repetition/periodicity constraints; composite actions; classes and instances of actions considering also the inheritance of constraints and the predictive role of the classes. The adoption of our approach can provide computer-based guideline systems with crucial advances. In the following, we discuss several facilities that can be designed on the basis of our representation formalism (see Section 3) and constraint

[4] Please note that we assume a suitable implementation of the data in STP-tree and in executionSTP. In particular, we assume that, given a class, it is possible to access in constant time to the list of its instances and that, given an instance, it is possible to access in constant time to its class.

procedure Next_Action()
1. retrieve the set of candidate next actions through a navigation of the control-flow relations in the guideline;
2. apply the algorithm in Fig. 11 to obtain the minimal network of temporal constraints;
3. retrieve the actions' possible execution-times from the minimal network (in the form of distances from the last-executed action, or from the origin of time).

Figure 19 Next-action facility.

procedure ask_GL()
1. provisionally add the constraints in Q to the set of constraints in the guideline
2. check consistency (via the algorithm in Fig. 10)

Figure 20 Yes/no query facility (only constraints about classes).

propagation algorithms (see Section 4), both during guideline acquisition and execution. Although the approach we propose is system-independent, in some cases we will exemplify it by sketching how we are planning to implement it in GLARE (Guide-Line Acquisition, Representation and Execution) [42—44]. GLARE is a prototypical system to acquire and execute clinical guidelines, developed by the Computer Science Department of the Università del Piemonte Orientale of Alessandria (Italy) in cooperation with Azienda Ospedaliera San Giovanni Battista of Torino (the third hospital in Italy).

5.1. Temporal facilities

On the basis of above approach, the following facilities can be provided by a computer-based manager of clinical guidelines:

1. the ***consistency-checking-guideline*** facility: this facility can be used in order to check the temporal consistency of the guideline in a principled way. Such a facility can be provided through an invocation to the algorithm in Fig. 10, which can be advocated at any stage during the acquisition of a clinical guideline, so that incremental consistency checking is also possible. By default, consistency checking can also be executed at the end of each acquisition working session.
2. the ***consistency-checking-instance*** facility: this facility can be used in order to check whether the temporal constraints in the guideline have been respected or not by the instances of actions that have been executed on the specific patients (considering also partial—i.e., ongoing executions). Such a facility is directly provided by our algorithm in Fig. 11.
3. the ***query*** facilities: during the execution of a given guideline (e.g. the guideline for multiple mieloma), the *query* facilities provide the user-physicians with a tool to obtain temporal information. This temporal information have often a crucial role when the user-physicians must take a decision. The set of the query facilities is:

3.1 the ***next-action*** facility: for scheduling purposes, it is important to provide a facility to assess when the next actions have to be performed, given the constraints in the whole guideline and given the time when the last actions in the guideline have been executed. The execution time of the next action(s) can be obtained through the algorithm in Fig. 19.

3.2 the ***yes/no query*** facility: given the set KB of temporal constraints in the guideline (and possibly the constraints KB′ on the instances of actions), one may ask whether a given set Q of temporal constraints is possible given KB (i.e., if it is consistent with KB). For instance, one may ask whether, given the constraints in Example 2, the first repetition of action b can be performed 16 days after the first repetition of $a11$. Moreover, one may ask the same question given the constraints in Example 2 plus those (regarding the specific instances of actions) in Example 2A. If the query Q only involves constraints in the guideline, the algorithm in Fig. 10 can be used, along the lines shown in Fig. 20. On the other hand, in case the query Q also involves constraints on the instances of actions, the algorithm in Fig. 11 must be used to check consistence (see Fig. 21)

3.3 the ***extract*** facility: this facility outputs the temporal constraints between a given set of actions. Such a facility can be efficiently implemented on the basis of the minimal network provided by our algorithm in Fig. 11 (along the lines discussed in [40], and using the *locality* properties proven in [39] to enhance efficiency), e.g., to have in output the minimal and maximal distance between the pairs of actions;

procedure Ask_instance()
1. provisionally add the constraints in Q to the set of constraints in the guideline
2. check consistency (via the algorithm in Fig.11).

Figure 21 Yes/no query facility (constraints about classes and instances).

3.4 the ***hypothetical query*** facility. In order to enhance the decision-making facilities, hypothetical temporal queries can be provided, to ask queries in the hypothesis that a new set TC of temporal constraints is assumed (in addition to the constraints in the guideline and those about instances). Hypothetical queries can be expressed in the abstract form

$$Q? \text{ If TC}$$

where Q is any type of query in our approach (i.e., a next-action, yes/no, or extract query). For example, given a pattern A_1, ..., A_n of actions in a guideline, temporal reasoning can be used in order to answer queries such as "If I perform action $A1$ today at 12 o'clock, when will I have to perform A_2, ..., A_n?", or "Is it OK if I perform A_1 today at 12, A_2 at 18 and A_3 at 20, and, if so, when will I have to perform A_4?".

Hypothetical queries can be answered in two steps (see Fig. 22):

4. the ***temporal-simulation*** facility: still considering decision making, temporal reasoning can be profitably coupled with "simulation" computer-based facilities to see the temporal consequences of choosing among different alternative paths in a guideline. In particular, GLARE provides the "what if?" facility allowing physicians to discriminate among different alternatives of a decision by simulating the consequences of each choice, i.e., by visiting the paths in the guideline stemming from each of the alternatives (see, e.g., [43]). Taking advantage of the algorithm in Fig. 11, such a facility can be extended in order to provide physicians with a way of comparing paths from the temporal point of view (i.e., in order to find the maximal and minimal temporal duration of each path). This facility can be provided as in Fig. 23.[5]

[5] A much more complex extension should be needed in order to deal with the fact that physicians might choose to execute actions which are not present in the given guideline. However, such an extension, which would involve an explicit treatment of action intentions [2], is outside the goals of this paper, which focuses on temporal issues.

5.2. A modular architecture

In our opinion, in order to enhance the generality of the temporal reasoning approach, such facilities can be better provided by a modular approach, in which a layered Temporal Server (TS) is loosely coupled with a guideline system (see Fig. 24). The TS can be seen as an object (e.g. a Java object) that interacts with the Guideline System using a Graphical User Interface (GUI) and that provides a set of methods implementing the facilities described above. The clinical guideline system delegates temporal-related problems to the TS module. The core of TS is the *temporal reasoner* (TR), that consists of the implementation of the two temporal reasoning algorithms in Figs. 10 and 11, and of the related data structures. The *facilities layer* uses the two consistency-checking algorithms in order to provide the facilities 1—4 along the lines described above. Moreover, for acquiring and representing temporal information the *interface layer* may make use of advanced visualization techniques such as the ones described in [30—33].

6. Comparisons and conclusions

In this paper, we propose a *principled* domain- and system-independent approach to the treatment of temporal constraints in clinical guidelines. We first motivate the introduction of a new and principled approach to the different types of temporal constraints involved in clinical guideline management. We then propose a new representation formalism, coping with both qualitative and quantitative temporal constraints, and constraints about (possibly periodic) repeated events. We also introduce two correct, complete and tractable algorithms to perform temporal reasoning on our formalism. Finally, we show how they can be used to implement different types of temporal reasoning facilities in a clinical guideline system. The formalism and the algorithms in this paper are an integration and an extension of our preliminary work in [41,44]. In particular, the treatment of conditioned repetitions (see Sections 3 and 4) and the design of the temporal facilities and of the architecture (see Section 5) are entirely new contributions of this paper.

procedure Hypothesize()
1. provisionally add the constraints in TC to the set of constraints
2. answer the query Q (in the new set of constraints)

Figure 22 Hypothetical queries.

procedure Temporal_Simulation()
1. for each path P_i to be compared:
2. hypothesize the existence of an instance of each action in P_i which has not been executed yet
3. apply the algorithm in Fig. 11 to the (executed and hypothesized) actions in P_i, to determine the minimal network MN_i
4. retrieve the minimal and maximal duration of P_i from MN_i

Figure 23 Temporal simulation.

The approach by Miksch et al. [19] is the closest one to ours in the literature. With respect to such an approach, we propose an extended language to deal with repetitions (e.g., we cope with conditioned repetitions, through the 'while' and 'onlyIf' constructs). Moreover, in order to grant the completeness of the constraint-based temporal reasoning process, we had to extend the basic STP framework, via the definition of the STP-tree and of the related constraint propagation algorithms. Finally, from the point of view of end-users, we also provide, besides the facilities in Miksch's approach (except temporal

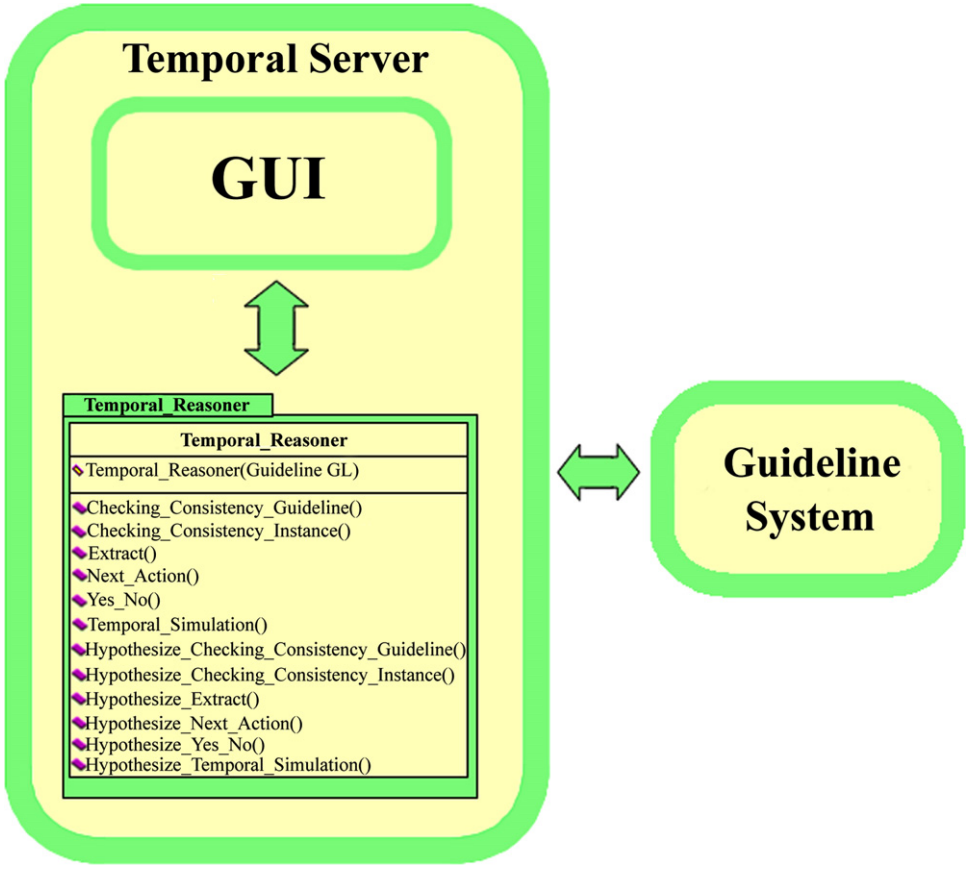

Figure 24 A modular architecture for our temporal knowledge server.

constraint visualization), the facilities 2—4 discussed in Section 5. In particular, the treatment of the ''a posteriori'' consistency between the temporal constraints in the guideline and of the execution times (i.e., facility 2) requires several extensions both from the representation point of view (since a separate STP needs to be used) and from the algorithmic point of view (since new constraint-propagation-based temporal reasoning algorithms have to be devised—see Section 4.4).

Temporal reasoning and query-answering facilities considering only qualitative constraints about atomic and not repeated actions in the medical domain have been also provided in [45]. Among related approaches in other areas, it is worth mentioning also the work in [46], dealing with facility 1 (and other facilities) in the closely related field of Workflow management, and considering also multiple granularities and disjunctive temporal problems.

Before concluding, it is worth discussing one of the assumptions of our approach, which relies on the full observability of instances of actions (i.e., the fact that we suppose that whenever an action is performed on a patient, it is observed and inserted in the set of instances of actions managed by our temporal reasoner). This assumption is reasonable in several practical applications: it holds for fully monitored patients (e.g., intensive care patients), and should (hopefully) hold also for hospitalized patients. In case such an assumption does not hold, either an exponential complexity extension of our algorithms must be devised, or the representation formalism must be restricted to preserve computational tractability, as we did in the general-purpose domain-independent temporal reasoner we proposed in [47].

Finally, it is worth mentioning that we have identified and tested our formalism considering also, besides the examples from the literature, two sets of temporal constraints arising from clinical protocols and guidelines, provided us by Prof. John Fox, Advanced Computation Laboratory, Cancer Research UK, London, UK and by Prof. Gianpaolo Molino, Azienda Ospedaliera San Giovanni Battista, Turin, Italy. We have a coverage of most of the temporal constraints contained in such sets. However, having to mediate between the expressiveness of the representation formalism and the goal of providing computationally tractable complete forms of temporal reasoning, we had to leave out some forms of disjunctive constraints, the most frequent of which is the ''non-overlapping'' constraint, stating that two actions *A* and *B* can be performed in any order, but at different times (i.e., Before(*A*,*B*) OR After(*A*,*B*), in the terms of Allen's Algebra [48]); as well known in the AI literature, such kind of constraints makes complete temporal reasoning intractable). In our future work, we will try to investigate whether ad hoc extensions of our approach can be devised to deal efficiently with such constraints, exploiting the fact that they are relatively rare in most practical clinical guidelines (a more general solution would involve the extensive use of TCSP [29]; a drawback of TCSP is the exponential running times of the temporal reasoning algorithms.

Also, as future work, we plan to implement a Java version of the approach presented in this paper, and to loosely pair it with GLARE, according to the architecture discussed in Section 5.2.

Acknowledgements

We are very grateful to Prof. John Fox, Advanced Computation Laboratory, Cancer Research UK, London, UK and to Prof. Gianpaolo Molino, Azienda Ospedaliera San Giovanni Battista, Turin, Italy, who provided us examples of temporal constraints taken from clinical guidelines. We are also very grateful to Prof. Gianpaolo Molino and his group for their continuous advise and cooperation in developing and testing the GLARE system, and, specifically, the temporal approach described in this paper.

Finally, we would also like to thank the anonymous referees for their inspiring and constructive criticism.

References

[1] Gordon C, Christensen JP, editors. Health telematics for clinical guidelines and protocols. Amsterdam, The Netherlands: IOS Press; 1995.

[2] 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 1998;14:29—51.

[3] Gordon C, Herbert SI, Johnson P. Knowledge representation and clinical practice guidelines: the DILEMMA and PRESTIGE projects. In: Proceedings of Medical Informatics Europe '96; 1996. p. 511—5.

[4] Musen MA, Tu SW, Das AK, Shahar Y. EON: a component-based approach to automation of protocol-directed therapy. J Am Med Inform Assoc 1996;3(6):367—88.

[5] Shiffman RN, Karras BT, Agrawal A, Chen R, Menco L, Nath S. GEM: a proposal for a more comprehensive guideline document model using XML. J Am Med Inform Assoc 2000; 7(5):488—98.

[6] Ohno-Machado L, Gennari JH, Murphy S, Jain NL, Tu SW, Oliver DE, et al. The guideline interchange format: a model for representing guidelines. J Am Med Inform Assoc 1998; 5(4):357—72.

[7] Peleg M, Boxwala A, Ogunyemi O, Zeng Q, Tu S, Lacson R, et al. GLIF3: the evolution of a guideline representation format. In: Proceedings of American Medical Informatics Association Annual Fall Symposium; 2000. p. 645—9.
[8] Quaglini S, Dazzi L, Gatti L, Stefanelli M, Fassino C, Tondini C. Supporting tools for guideline development and dissemination. Artif Intell Med 1998;14(2):119—37.
[9] Quaglini S, Stefanelli M, Cavallini A, Miceli G, Fassino C, Mossa C. Guideline-based careflow systems. Artif Intell Med 2000;20(1):5—22.
[10] Tu SW, Kahn MG, Musen MG, Ferguson JK, Shortliffe EH, Fagan LM. Episodic skeletal-plan refinement on temporal data. Commun ACM 1989;32:1439—55.
[11] Fox J, Johns N, Rahmanzadeh A, Thomson R. Disseminating medical knowledge: the PROforma approach. Artif Intell Med 1998;14:157—81.
[12] Musen MA, Carlson CW, Fagan LM, Deresinski SC, Shortliffe EH. T-HELPER: automated support for community-based clinical research. In: Proceedings of 16th annual symposium on computer applications in medical care; 1992. p. 719—23.
[13] Focus on Clinical Guidelines and Patient Preferences. Special Issue of J Am Med Inform Assoc 1998;15(3).
[14] Tu SW, Mark MS, Musen A. A flexible approach to guideline modeling. In: Proceedings of 1999 American Medical Informatics Association Annual Symposium; 1999. p. 420—4.
[15] Fridsma DB, editor. Special issue workflow management and clinical guidelines. J Am Med Inform Assoc 2001;22(1):1—80.
[16] Keravnou ET, editor. Special issue: temporal reasoning in medicine. Artif Intell Med 1996;8(3).
[17] Shahar Y, Musen MA. Knowledge-based temporal abstraction in clinical domains. Artif Intell Med 1996;8(3):267—98.
[18] Shahar Y. A framework for knowledge-based temporal abstraction. Artif Intell 1997;90(1/2):79—133.
[19] Duftschmid G, Miksch S, Gall W. Verification of temporal scheduling constraints in clinical practice guidelines. Artif Intell Med 2002;25(2):93—121.
[20] Terenziani P. Reasoning about time. Encyclopedia of cognitive science Macmillan Reference Ltd.; 2003 . p. 869—874.
[21] Yampratoom E, Allen J. Performance of temporal reasoning systems. ACM SIGART Bull 1993;4(3):26—9.
[22] Vila L. A survey on temporal reasoning in artificial intelligence. AI Commun 1994;7(1):4—28.
[23] Terenziani P. Integrating calendar-dates and qualitative temporal constraints in the treatment of periodic events. IEEE Trans Knowledge Data Eng 1997;9(5):763—83.
[24] Bettini C, Jajodia S, Wang X. Solving multi-granularity constraint networks. Artif Intell 2002;140(1/2):107—52.
[25] Combi C, Franceschet M, Peron A. Representing and reasoning about temporal granularities. J Logic Comput 2004;14(1):51—77.
[26] Meiri I. Combining qualitative and quantitative constraints in temporal reasoning. Artif Intell 1996;87(1/2):343—85.
[27] Sherman E, Hripcsak G, Starren J, Jender R, Clayton P. Using intermediate states to improve the ability of the Arden Syntax to implement care plans and reuse knowledge. In: Proceedings of annual symposium on computer applications in medical care; 1995. p. 238—42.
[28] Weng C, Kahn M, Gennari JH. Temporal knowledge representation for scheduling tasks in clinical trial protocols. In: Proceedings of American Medical Informatics Association Annual Symposium; 2002. p. 879—83.
[29] Dechter R, Meiri I, Pearl J. Temporal constraint networks. Artif Intell 1991;49:61—95.
[30] Kosara R, Miksch S. Metaphors of movement: a visualization and user interface for time-oriented, skeletal plans, artificial intelligence in medicine. Special Issue: Inform Visual Med 2001;22(2):111—31.
[31] Combi C, Portoni L, Pinciroli F. Visualizing temporal clinical data on the www. In: Proceedings of the joint european conference on artificial intelligence in medicine and medical decision making (AIMDM '99); 1999. p. 301—11.
[32] Chittaro L. Information visualization in medicine, special issue. Artif Intell Med 22(2) 2001, 81-L 191.
[33] Kosara R, Miksch S. Visualization methods for data analysis and planning in medical applications. Int J Med Inform 2002;68(1/3):141—53.
[34] Terenziani P. Toward a unifying ontology dealing with both user-defined periodicity and temporal constraints about repeated events. Comput Intell 2002;18(3):336—85.
[35] Egidi L, Terenziani P. A lattice of classes of user-defined symbolic periodicities. In: Proceedings of 11th international symposium on temporal representation and reasoning (TIME 2004); 2004. p. 13—20.
[36] Bettini C, De Sibi R. Symbolic representation of user-defined time granularities. In: Proceedings of sixth international workshop on temporal representation and reasoning (TIME99); 1999. p. 17—28.
[37] Egidi L, Terenziani P.In: Proceedings of the 11th international symposium on temporal representation and reasoning (TIME 2004). A mathematical framework for the semantics of symbolic languages representing periodic time 2004;21—7.
[38] Bettini C, Dyreson C, Evans W, Snodgrass R, Wang X, A glossary of time granularity concepts, in temporal databases: research and practice, Lecture Notes in Computer Science 1399; 1998.
[39] Brusoni V, Console L, Terenziani P. On the computational complexity of querying bounds on differences constraints. Artif Intell 1995;74(2):367—79.
[40] Brusoni V, Console L, Pernici B, Terenziani P. LaTeR: managing temporal information efficiently. IEEE Expert 1997;12(4):56—64.
[41] Anselma L. Recursive representation of periodicity and temporal reasoning. In: Proceedings of 11th international symposium on temporal representation and reasoning (TIME 2004); 2004. p. 52—9.
[42] Terenziani P, Molino G, Torchio M. A modular approach for representing and executing clinical guidelines. Artif Intell Med 2001;23:249—76.
[43] Terenziani P, Montani S, Bottrighi A, Molino G, Torchio M. Supporting physicians in taking decisions in Clinical Guidelines: the GLARE's "what if" facility. In: Proceedings of American Medical Informatics Association Annual Symposium; 2002. p. 772—6.
[44] Terenziani P, Montani S, Torchio M, Molino G, Anselma L. Temporal consistency checking in clinical guidelines acquisition and execution: the GLARE's approach. In: Proceedings of American Medical Informatics Association Annual Symposium; 2003. p. 659—63.
[45] van Beek P. Temporal query processing with indefinite information. Artif Intell Med 1991;3:325—39.
[46] Bettini C, Wang X, Jajodia S. Temporal reasoning in workflow systems. Distributed Parallel Databases 2002;11(3):269—306.
[47] Terenziani P, Anselma L. A knowledge server for reasoning about temporal constraints between classes and instances of events. Int J Intell Syst 2004;19(10):919—47.
[48] Allen JF. Maintaining knowledge about temporal intervals. Commun ACM 1983;26(11):832—43.