Towards the Flexibility in Clinical Guideline Modelling Languages

Nataliya Mulyar[1], Maja Pesic[1], Wil M.P. van der Aalst[1] and Mor Peleg[2]

[1] Eindhoven University of Technology
GPO Box 513, NL5600 MB Eindhoven, The Netherlands
{n.mulyar, m.pesic, w.m.p.v.d.aalst}@tue.nl
[2] Department of Information Systems, University of Haifa
Mount Carmel, 31905, Israel
{morpeleg}@mis.hevra.haifa.ac.il

Abstract. Recent analysis of clinical Computer-Interpretable Guideline (CIG) modelling languages from the perspective of the control-flow patterns has revealed limited capabilities of these languages to provide flexibility for encoding and executing clinical guidelines [15]. The concept of flexibility is of major importance in the medical-care domain since no guarantee can be given on predicting the state of patients at the point of care. In this paper, we illustrate how the flexibility of CIG modelling languages can be improved by describing clinical guidelines using a *declarative* approach. We propose a CIGDec language for modelling and enacting clinical guidelines.

Keywords: Clinical guidelines, Computer-interpretable guidelines, flexibility, modelling languages, declarative model specification,temporal logic.

1 Introduction

Clinical practice guidelines are "systematically developed statements to assist practitioner decisions about appropriate health actions for specific clinical circumstances" [7].The main intent of clinical guidelines is to improve the quality of patient care and reduce costs. To provide patient-specific advice at the point of care the medical community has taken initial steps towards the computerization of clinical knowledge contained in clinical guidelines. Computer-interpretable guidelines were extensively used for developing decision-support systems [17]. Creating computer-interpretable representations of the clinical knowledge contained in clinical guidelines is crucial for developing decision-support systems that can provide patient-specific advice at the point of care. These types of systems have been shown to affect clinicians' behavior more than paper-based guidelines [17]. Unfortunately, due to the absence of a single standard for developing CIG modelling languages, the functionality of decision-support systems employing such modelling languages from the perspective of the control-flow differs to a great extent.

We analyzed the suitability of four modelling languages Asbru, PRO*forma*, GLIF and EON for expressing control-flow patterns [2] and revealed that these languages do not offer more control-flow flexibility than process modelling languages employed by the Workflow Management Systems (WFMS) [15]. This is remarkable since one would

expect CIG modelling-languages to offer dedicated constructs allowing for more flexibility. Accommodating *flexibility* into guidelines means that the CIG would be sensitive to the characteristics of specific patients and specific health care organizations [31].

The modelling languages we analyzed explicitly model a care process by specifying the steps and the order in which these steps are to be executed. Although process languages allow for some flexibility by means of modelling alternative paths, any of which could be taken depending on some a-priori available data, they are incapable of handling exceptional or unpredicted situations. Exceptional situations have to be modelled explicitly. However, modelling of all possible scenarios results in complex models and is not feasible since exceptional situations and emergencies may arise at any point in time. This makes it difficult or even impossible to oversee what activity should be performed next. To overcome these problems, i.e. reduce the complexity of models, and to allow for more flexibility in selecting an execution path, *in this paper* we propose CIGDec as a declarative language for modelling clinical guidelines. Unlike imperative languages, declarative languages specify the "what" task should be performed without determining of the "how" to perform it. CIGDec specifies by means of constraints the rules that should be adhered to by a user during a process execution while leaving a lot of freedom to the user in selecting tasks and defining the order in which they can be executed. CIGDec should be considered as a variant of ConDec [18] and DecSerFlow [4].

The reminder of this paper is organized as follows. In Section 2 we introduce CIG modelling languages Asrbu, GLIF, EON and PRO*forma* using a patient-diagnosis scenario. We also briefly describe the similarities and differences between the considered languages from the perspective of the control-flow patterns. In Section 3 we introduce CIGDec and illustrate a CIGDec model of the patient-diagnosis scenario. We discuss the drawbacks and advantages of the proposed language in Section 4. Related work is presented in Section 5. Section 6 concludes the paper.

2 Computer-Interpretable Guidelines

This section describes the main concepts of four well-known CIG modelling languages: Asbru, EON, GLIF, and PRO*forma*. These have been evaluated from the control-flow perspective using the workflow patterns [3]. We introduce the main concepts of these languages by modelling the following patient diagnosis scenario in the tools AbruView, Protege-2000 (for EON and GLIF) and Tallis respectively. A patient is registered at a hospital, after which he is consulted by a doctor. The doctor directs the patient to pass a blood test and urine test. When the results of both tests become available, the doctor sets the diagnosis and defines the treatment strategy.

While specifying the behavior of the scenario, we immediately reflect on the possibilities to deviate from this scenario which might be necessary for example in an emergency case. In particular, we indicate whether it is possible to skip a patient registration step and immediately start with the diagnosis; whether it is possible to perform multiple tests of the same kind or perform only one of them; whether it is possible to perform the consultancy by the doctor after performing one of the tests again.

While describing the models of the patient-diagnosis scenario we also indicate the degree of support of the control-flow patterns by the analyzed modelling languages. Ta-

ble 1 summarizes the comparison of the CIG modelling languages from the perspective of the control-flow patterns [3]. The complete description of the patterns and how they are supported by the analyzed languages can be found in [23, 15].

Basic Control–flow	1	2	3	4	New Patterns	1	2	3	4
1. Sequence	+	+	+	+	21. Structured Loop	+	+	+	+
2. Parallel Split	+	+	+	+	22. Recursion	+	-	-	-
3. Synchronization	+	+	+	+	23. Transient Trigger	-	-	-	+
4. Exclusive Choice	+	+	+	+	24. Persistent Trigger	-	-	+	+
5. Simple Merge	+	+	+	+	25. Cancel Region	-	-	-	-
Advanced Branching and Synchronization					26. Cancel Multiple Instance Activity	+	-	+	+
6. Multi-choice	+	+	+	+	27. Complete Multiple Instance Activity	+	-	-	+
7. Structured Synchronizing Merge	+/-	-	-	+	28. Blocking Discriminator	-	-	-	-
8. Multi-merge	-	-	-	-	29. Cancelling Discriminator	+	-	-	+
9. Structured Discriminator	+	+	+	+	30. Structured N-out-of-M Join	+	-	+	+
Structural Patterns					31. Blocking N-out-of-M Join	-	-	-	-
10. Arbitrary Cycles	-	+	+	-	32. Cancelling N-out-of-M Join	-	-	-	+
11. Implicit Termination	+	+	+	+	33. Generalized AND-Join	-	-	-	-
Multiple Instances Patterns					34. Static N-out-of-M Join for MIs	-	-	-	-
12. MI without Synchronization	-	-	-	-	35. Static N-out-of-M Join for MIs with Cancellation	-	-	-	-
13. MI with a priori Design Time Knowledge	+/-	+/-	+/-	+/-	36. Dynamic N-out-of-M Join for MIs	-	-	-	-
14. MI with a priori Run-Time Knowledge	-	-	-	-	37. Acyclic Synchronizing Merge	-	-	-	+
15. MI without a priori Run-Time Knowledge	-	-	-	-	38. General Synchronizing Merge	-	-	-	-
State-Based Patterns					39. Critical Section	+	-	+	-
16. Deferred Choice	+	-	+	+	40. Interleaved Routing	+	-	+	-
17. Interleaved Parallel Routing	+	-	-	-	41. Thread Merge	-	-	-	-
18. Milestone	-	-	-	+	42. Thread Split	-	-	-	-
Cancellation Patterns					43. Explicit Termination	-	-	-	-
19. Cancel Activity	+	+	+	+					
20. Cancel Case	+	-	+/-	+					

Table 1. Support for the Control–flow Patterns in (1)Asbru, (2)EON, (3)GLIF, and (4)PRO*forma*

Figure 1 presents the scenario modelled in AsbruView [1], which is a markup tool developed to support authoring of guidelines in Asbru [26]. A process model in Asbru [25] is represented by means of a time-oriented skeletal plan. The root plan (marked as Plan A) is composed of a set of other plans. The plans are represented as 3-dimensional objects, where the width represents the time axis, the depth represents plans on the same level of the decomposition (i.e. which are performed in parallel), and the height represents the decomposition of plans into sub-plans. Parent plans are considered to be completed when all mandatory sub-plans completed. Enabling, completion, resumption and abortion conditions can be specified for each plan if necessary.

As the time axe shows, plans *Register patient*, *Consult with doctor*, *Test phase* and *Define the treatment* are executed sequentially. The *Test phase* plan is a parallel plan consisting of two activities *ask for urine test* and *ask for blood test*. The parallel plan

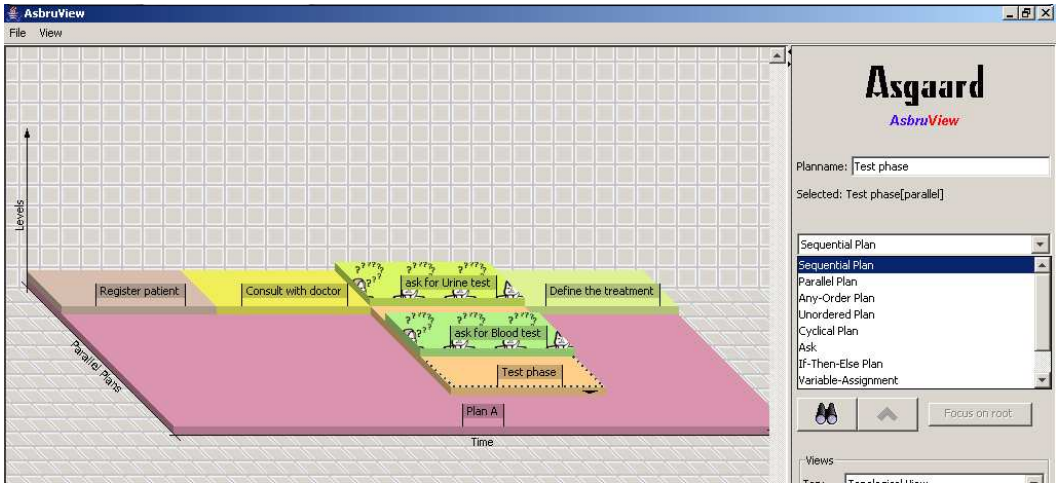


Fig. 1. The patient-diagnosis scenario modelled in AsbruView

requires all enclosed activities to be completed in order to pass the flow of control to the next plan. In this model, we used only two types of plans: sequential (root plan) and parallel plan (Test phase plan). AbsruView allows to visualize also Any-order Plan, Unordered Plan, Cyclical Plan, and If-then-else Plan, and two types of actions: Ask and Variable Assignment.

Deviations from the modelled scenario are not possible in AsbruView, since the all plans are structured and their order is strictly defined. It would be possible to adjust the model and implicitly incorporate all required execution paths. In particular, the Cyclical Plan should be used in order to iterate the execution of a certain task. In order to relax the parallel order of the blood- and urine-tests' tasks, an Any-order Plan could be used. However, the behavior of the model would be still deterministic and not allow for much flexibility. In Asbru there is a concept of plan activation mode. It allows conditions for aborting, suspending and resuming a plan. This can be relevant for the case of registering a patient and not having all the needed data initially: a plan is suspended and later resumed. As the pattern-based analysis showed [15], Asbru is able to support 20 out of 43 control-flow patterns. Asbru uniquely supports the recursive calls and interleaved parallel routing, which are the features not directly supported by other analyzed languages.

An EON model of the patient-diagnosis scenario created in *Protege-2000* environment is illustrated in Figure 2. *Protege-2000* is an ontology-editor and knowledge-base framework (cf. `http://protege.stanford.edu`). Main modelling entities in EON [28] are scenarios, action steps, branching, decisions, and synchronization [29, 27]. A scenario is used to characterize the state of a patient. There are two types of Decision steps in EON, i.e. a Case step (select precisely one branch) and a Choice step (select at least one branch). An Action step is used to specify a set of action specifications or a sub-guideline that are to be carried out. Branch and Synchronization steps are used to specify parallel execution. In Figure 2 these steps are used to do the two tests in parallel.

The following features offered by EON can be used in order to make the model of the patient-diagnosis scenario more flexible. A Scenario can be used to model different

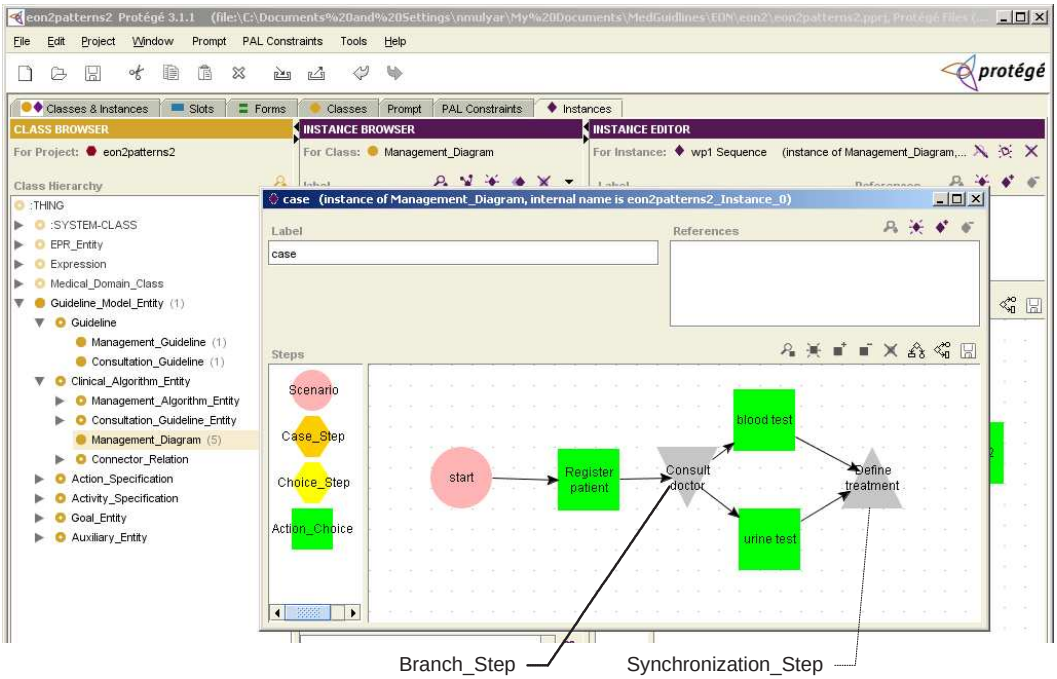
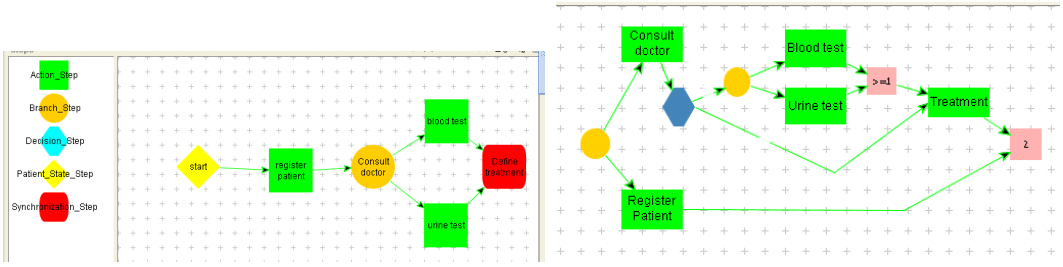


Fig. 2. The patient-diagnosis scenario modelled in EON/*Protege*

entry points to the model. This allows to ”jump” into the middle of the model and to start execution from that point. This feature is useful for emergency cases where for example a registration step has to be skipped and immediate treatment procedure has to be started. Unfortunately, EON offers not much flexibility with respect to synchronization of multiple branches, i.e. it allows the *define treatment* task to be executed only if a single or all branches have been executed. However, it is incapable of predicting how many branches were selected and performing a partial synchronization after all selected branches were executed. From all analyzed modelling languages, EON supports the lowest number of the control-flow patterns, i.e. only 11 out of 43.

GLIF3.5 [5] is a specification method for structured representation of guidelines. To create a model in GLIF, an ontology schema and a graph widget have to be loaded into the *Protege-2000* environment. Figure 3(a) visualizes the GLIF model of the basic patient-diagnosis scenario. In GLIF3.5 five main modelling entities are used for process modelling, i.e. an Action Step, a Branch Step, a Decision Step, a Patient-State Step, and a Synchronization Step. An Action Step is a block used to specify a set of tasks to be performed, without constraints set on the execution order. It allows for including sub-guidelines into the model. Decision steps are used for conditional and unconditional routing of the flow to one out of multiple steps. Branch and Synchronization steps are used for modelling concurrent steps. A Patient-State Step is a guideline step used for describing a patient state and for specifying an entry point(s) to a guideline.

In order to allow the behavior of the basic patient-diagnosis scenario to deviate, all possible pathes have to be explicitly modelled. Figure 3(b) represents a scenario, in which *Register patient* step can be done in parallel to any other step, but it has to be exactly once to complete the process (if more than once is desired, an iteration condition for *Register patient* step can be added which resembles a while loop: while not all patient data has been entered, repeat Register Patient. In this scenario, a decision can be taken to order tests or to proceed to treatment without tests. However, treatment or ordering of tests cannot be done before consulting with a doctor. One or two tests can be ordered before proceeding to treatment. Figure 3(b) shows how complex the model has become after we introduced several deviations from the basic scenario. Thus, this specification needs to model graphically all the possible paths of execution, and it is not very scalable.



(a) (b)

Fig. 3. The patient-diagnosis scenario modelled in GLIF3.5/*Protege*

Similar to EON, GLIF allows multiple entry points into the model to be specified by means of the Patient-State step. This allows the execution to start from any point where a patient enters a scenario model while skipping tasks-predecessors. GLIF offers more variants for synchronizing parallel branches, i.e. to synchronize after one, several or all tasks have been completed. However, GLIF is incapable of synchronizing branches in the conditions when it is unknown which branches and how many of them will be chosen. This explains why the number of control-flow patterns supported by GLIF (17 out of 43) is bigger than in EON but still smaller than Asbru.

PRO*forma* [8] is a formal knowledge representation language for authoring, publishing and executing clinical guidelines. It deliberately supports a minimal set of modelling constructs: actions, compound plans, decisions, and enquiries that can be used as tasks in a task network. In addition, a keystone may be used to denote a generic task in a task network. All tasks share attributes describing goals, control flow, preconditions, and post-conditions. A model of the basic patient-diagnosis scenario created in Tallis is shown in Figure 4(a). Note that in PRO*forma* control-flow behavior is captured by modelling constructs in combination with the scheduling constraints. Scheduling constraints are visualized as arrows connecting two tasks, meaning that the task at the tail of the arrow may become enabled only after the task at the head of the arrow has completed. To deviate from the basic scenario, some of the scheduling constraints should be removed as it is shown in Figure 4(b).

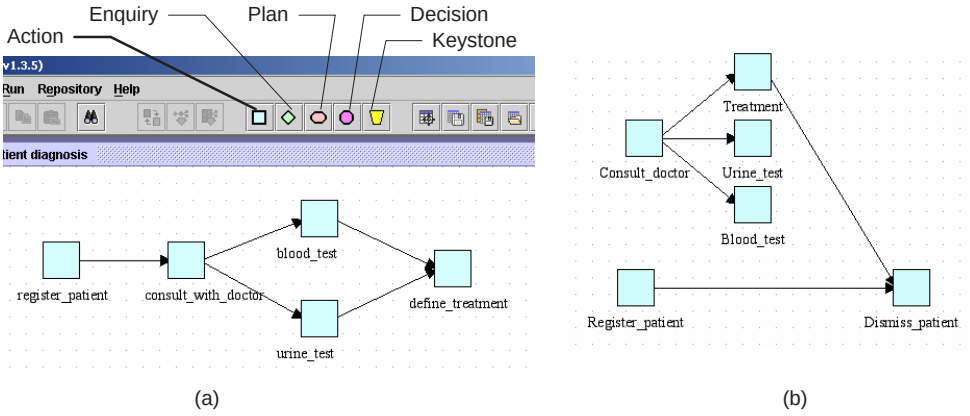


Fig. 4. The patient-diagnosis scenario modelled in PRO*forma*/Tallis

In contrast to all examined languages, PRO*forma* allows for late modelling, i.e. if it is not clear in advance what steps exactly should be performed, these steps are modelled by means of keystones, which are substituted by a desired type of the task before the model is deployed. Furthermore, by means of triggers it is possible to specify that a task has to be performed even if the task's preconditions were not satisfied. PRO*forma* also allows for more flexibility during the synchronization of multiple paths, thus it is able to predict how many paths from the available ones were selected and to merge them when they have completed. Furthermore, scheduling constraints in PRO*forma* are not obligatory. This means that stand-alone tasks may be activated upon the fulfillment of a pre-condition. Such tasks do not depend on the imperatively specified flow of other activities. PRO*forma* has the highest degree of pattern-support from all analyzed languages, i.e. it supports 22 out of 43 patterns.

Table 2 shows the terms used in the CIG modelling languages and our preferred terms. These terns will be used in the remainder of this paper.

Terms	Asbru	EON	GLIF	PRO*forma*
Process model	Plan	Guideline	Guideline	Plan
Task/ activity	Plan	Action	Action	Action, Enquiry
Parallel branching	Plan type	Branch and Synchronization	Branch and Synchronization	Action or Enquiry
Exclusive branching	Plan precondition, Plan type	Decision	Decision	Decision, Enquiry and scheduling constraints

Table 2. Terms used by Asbru, EON, GLIF, and PRO*forma*

The medical community has always emphasized that it is impossible to use workflow formalisms because of specific requirements such as flexibility. However, when we examined guideline modelling languages we didn't find more flexibility than in todays workflow and BPM products.

The analysis of CIG modelling languages [15] showed that these languages are very similar to BPM languages. Given a large variety of process modelling languages nowadays it makes no sense to develop more complicated language which would support more control-flow patterns. Instead, we take a completely new approach and propose a CIGDec language for encoding clinical guidelines.

3 Declarative description of clinical guidelines

In this section we present the CIGDec declarative language and show benefits of applying it for modeling clinical guidelines.

Modelers who use traditional CIG modelling languages have to represent all possible scenarios (normal and exceptional) that can occur during the execution. Such a model has to include all possible scenarios that can occur during the execution. This means that CIG modelers have to predict in detail all possible execution paths in advance for the guideline they are modelling. The model itself tends to be very complex and strictly predefines all relationships between all steps in the guideline. Such a model not only prescribes to users what to do, but it also contains a detailed specification about how to do it. Hence, traditional CIG modelling languages are of an imperative nature.

CIGDec is a declarative language, i.e., its models specify what to do and leave it up to the user to decide how to work depending on the case. CIGDec models do not require all possible scenarios to be predicted in advance. On the contrary, the model consists of a set of tasks and some dependencies (relationships) between these tasks. Dependencies between tasks can be seen as some general rules that should always hold in the guideline. Any task in the model can be performed by a user if and only if none of the specified rules is violated. As an extreme example, a CIGDec model that consists only of a set of tasks without dependencies would represent a completely free guideline, where a user can execute any task in any desired order. As more rules in the model as less possibilities to deviate from a certain execution order is given to the user. Therefore, rules constrain the model. Hence, we refer to dependencies between tasks (rules) as to *constraints*.

Any CIG model consists of a set of tasks and some relationships between them specifying the exact order of tasks. Typically, traditional languages use a predefined set of constructs that can be used to define relations between tasks: 1) sequence, 2) choice, 3) parallelism, and 4) iteration. These constructs are used to define the exact control-flow (order of tasks) in the guideline. In CIGDec, this set of constructs is unlimited, i.e., constructs can be added, changed and removed, depending on the requirements of the application, domain, users, etc. We refer to constructs used for defining possible types of dependencies between tasks in CIGDec as to *constraint templates*. Each template has its semantics, which is formally represented by one Linear Temporal Logic (LTL) formula. This semantics is used for the computerized enactment of the guideline [19]. LTL is a logic extended with special temporal operators - 'always' ($\Box$), 'eventually' ($\Diamond$), 'until' $\sqcup$, and 'next time' $\bigcirc$. This logic is extensively used in the field of model checking, where the target model is verified against properties specified in LTL [11, 10]. For computerized enactment of CIGDec model we use algorithms for translating LTL expressions into automata developed in the model checking field [9, 4, 19]. Since LTL formulas can be very complex and hard to understand, each template also has unique graphical representation

for users. In this way, we ensure that CIGDec users do not have to be LTL experts in order to work with models [19]. Although the set of templates is 'open', we propose a starting collection of templates in Section 3.1.

3.1 CIGDec templates

When looking at a traditional CIG model, one usually tries to find the starting point and then follows the control-flow until the end point is reached. This cannot be applied to CIGDec models. Constructs (lines) do not necessarily describe the order of tasks, but rather various dependencies between them. In our starting set of constraint templates we distinguish between two types of templates: 'existence' (unary) templates, and binary templates that can represent a 'relation' or 'negative relation'. Figure **??** shows examples of templates.

'Existence' templates are unary templates because they involve only one task. Generally, they define the cardinality (possible number of executions) of the task. The top four examples in Figure 5(a, b, c and d) are unary 'existence' templates.

- *Existence: at most N times* template specifies the upper upper and/or lower bounds for the numbers of executions of the task, e.g. a cardinality of the type '0..N'. In the example in Figure 5(a) the task *announce death* can be executed at most once.
- *Existence: exactly N times* template specifies exact number of executions of the task, e.g. a cardinality 'N'. In the example presented in Figure 5(b) task *close file* has to be executed exactly once in the process.
- *Existence: at least N times* template specifies at least how many times a given task at task has to be executed. In the example in Figure 5(c) task *register data* has to be executed at least once.
- *Conditional never* template specifies that a given task should not be performed if an associated with the task condition is fulfilled. For example, Figure 5(d) shows that *X ray* is performed only is the *pregnancy* is false.

Binary templates involving two tasks are listed below:

- *Responded existence* template specifies that if one task is performed then the other task is performed before or after the first task. The example in Figure 5(e) specifies that if *surgery* is performed then the *family is informed* before or after the operation.
- *Response* template considers the order of activities, thus specifying that one task has to be executed at least once after the other task has completed. The example in Figure 5(f) shows that the *surgery report* has to be filled in at least once after the *surgery*. Note that in all these examples it was possible to have an arbitrary execution of other tasks between the two related tasks. For example, execution sequence [*surgery*, *inform family*, *fill operation report*] fulfills both constraints 'responded existence' and 'response'.
- *Choice* template specifies that either one of two tasks can be performed. The example in Figure 5(g) shows that either a prostate or gynecological check are possible. It is not obligatory to perform any of them, but once one of them is performed the other one cannot be performed anymore. Also, it is possible to execute the selected task multiple times.

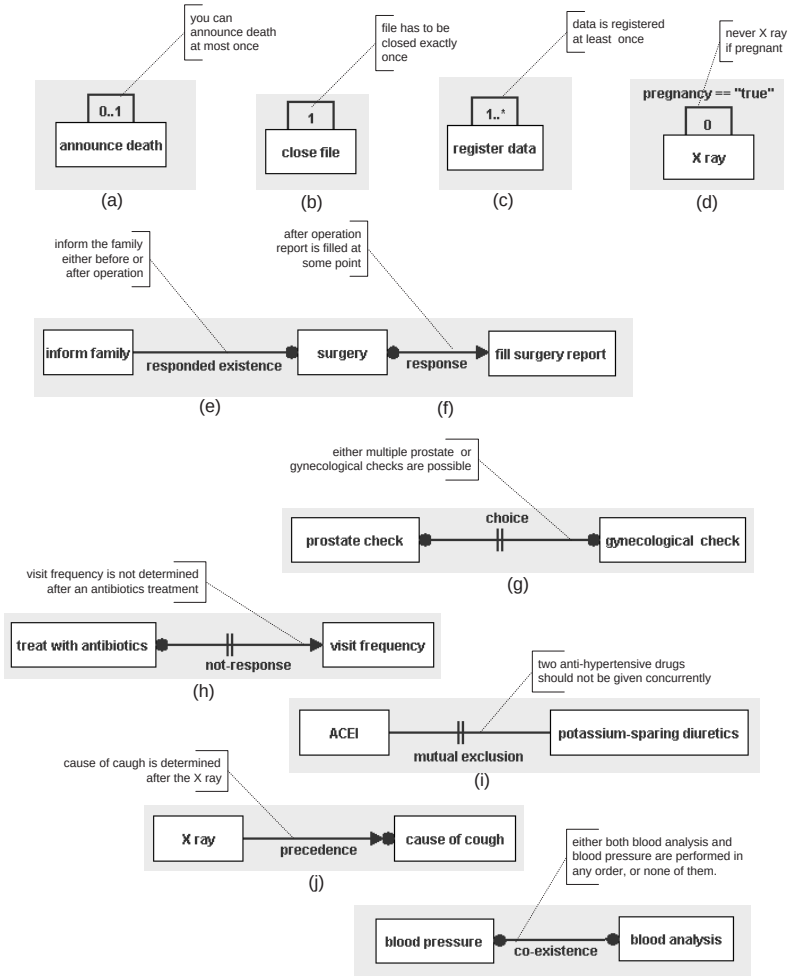


Fig. 5. Examples of CIGDec constraints.

- *No-response* template specifies that a task can not be executed after the other task has completed. An example in Figure 5(h) shows that if an infection is 'treated with antibiotics' (once or multiple times), the 'visit frequency' is not determined.

- *Mutual exclusion* template between two tasks prevents them to execute concurrently. For example, Figure 5(i) depicts that two anti-hypertensive drugs should not be given simultaneously (ACEI and potassium-sparing diuretics).

- *Precedence* template. As shown in Figure 5(j), there is a precedence relation between tasks *X ray* and *cause cough*, and therefore, the cause of cough can only be determined after at least one *X ray*.

- *Co-existence* template between tasks *A* and *B* specifies that if *A* happens then *B* happens and vice versa, without specifying in which order. For example, the blood pressure and the analysis of blood of a patient can be performed concurrently or in any order.

Table 4 in the Appendix shows CIGDec templates and its corresponding LTL formulas.

3.2 CIGDec model for the diagnosis scenario

Figure 6 depicts a CIGDec model of our patient-diagnosis scenario. It consists of five tasks. In an extreme case, it would be possible to make and use the model consisting only out of these tasks and without any constraints. This would be a unrestricted model allowing for maximum flexibility, where tasks could be executed an arbitrary number of times ('0..*') and in any order. This model would have an infinite number of execution possibilities (different process instances). However, to develop a model that provides guidance, we add five constraints derived from three constraint templates.

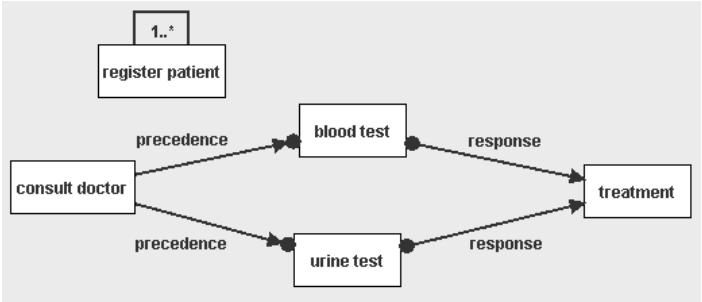


Fig. 6. CIGDec model for the diagnosis scenario.

First, there is one unary (involving one task) constraint created from the template 'existence' - constraint presented as cardinality *1..** above the task *register patient*. This constraint specifies that the task *register patient* has to be executed at least once within one process (guideline) enactment.

Second, there are two constraints created from the template 'precedence' as shown in Figure 6: one between tasks *consult doctor* and *blood test* and one between tasks *consult doctor* and *urine test*. Precedence is a binary template, i.e., it defines a dependency between two tasks. A 'precedence' between two tasks *A* and *B* means that task *B* can only be executed after task *A* was executed at least once [4]. It is possible that other tasks are executed between *A* and *B*. Hence, if we want to execute task *blood test* we can do so only after we have executed task *consult doctor*. Note that other tasks from the model can be executed between *consult doctor* and *blood test*. Task *test urine* also has a 'precedence' relation with task *consult doctor* and it can be executed only after task *consult*

doctor. Similarly, there could be other tasks between them. Moreover, the doctor may be consulted multiple times before and after doing the tests.

Third, we use a binary template 'response' to create two constraints: one between tasks *blood test* and *treatment* and one between tasks *urine test* and *treatment*. Template 'response' between tasks *A* and *B* defines that after every execution of task *A* task *B* has to be executed at least once while it is possible that other tasks are executed between *A* and *B*. Thus, after every *blood test* at least one *treatment* should follow, and there could be other taks from the model executed between them. The same holds for tasks *urine test* and *treatment*.

The possibilities given to a user during execution of the model depicted in Figure 6 are defined as a combination of all five constraints in the model. When looking at the models designed by means of the analyzed language Asbru, the execution always had to start with the task *register patient*. This may cause problems in cases of emergency, when there is no time for the registration requiring the procedure with doctor (task *consult doctor*) to start immediately. While in EON and GLIF allow multiple entry-points to a scenario, these entrance steps have to be modelled explicitly. In PRO*forma* a task can be modelled without use of scheduling constraints which allows this task to be executed at any moment. Note however, that the CIG languages assume that a task can be executed once during the model execution or *iteratively* a specified number of times. In CIGDec model a patient-registration step can be performed at any moment during the CIGDec process. Furthermore, CIGDec model allows to perform *register patient* multiple times in case the required data is not available on time.

If we look at the traditional models Figures 1, 2, 3 and 4 (i.e. mode using Absru, EON, GLIF and PRO*forma*), task *consult doctor* was executed exactly once. CIGDec model allows this task not to be executed at all, but it also allows it to be executed multiple times. For example, some patients use medication periodically. For them only the *treatment* task has to be performed either before or after the *register patient* has been executed. On the other hand, in some complex cases, task *consult doctor* can be performed more than once at various points during the CIGDec execution.

If necessary, a doctor can order a *blood test* many times or not at all during the CIGDec process. However, constraint 'precedence' between this task and *consult doctor* makes sure that *blood test* can not be done for a patient that has not seen the doctor before. Note that his holds only for the first *blood test*. Sometimes, the results can be unexpected and doctor can order a different type of *blood test* without having to see the patient again. After every *blood test*, task *treatment* is performed. It is possible that during *treatment* no medication is prescribed due to the good test results. However, it is also possible to wait and to perform several *blood tests* in order to make a good decision before the task *treatment* is performed.

Since task *urine test* has the same relationships as task *blood test* ('precedence' with *consult doctor* and 'response' with *treatment*), the same variants of execution paths hold like for the task *blood test*. However, note that none of the tasks 'blood test' and 'urine test' do not have to execute at all, or each of them can be executed one or more times, or only one of them can be executed one or more times.

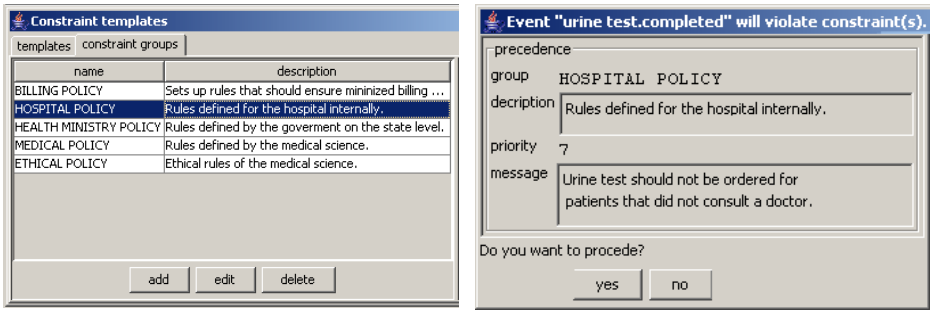
Table 3 shows three (out of many) examples (cases) of possible usages of the CIGDec model from Figure 6. First, in the 'case A' a periodical medication is prescribed to a

chronic patient: only *register patient* and *treatment* tasks are executed. In the 'case B' an urgent vist starts directly with *consult doctor* and only afterwards the task *register patient* is executed. The *urine test* was not necessary. The results of the *blood test* were unclear so the *treatment* is executed only after the results of the second *blood test* became available and an additional *consult doctor* task. In the 'case C', the situation was not urgent, so task *register patient* was performed before the task *consult doctor*. Both *urine test* and *blood test* are performed. However, due to alarming results of the *urine test* an immediate *treatment* was executed to prescribe appropriate medication. The results of *blood test* arrived later, and an additional *treatment* task was executed to handle the *blood test* results as well.

case A	case B	case C
register patient	consult doctor	register patient
treatment	blood test	consult doctor
	register patient	urine test
	blood test	treatment
	consult doctor	blood test
	treatment	treatment

Table 3. Some examples of possible enactments of CIGDec model in Figure 6

During the execution, users are guided to follow the constraints from the model. At any point in time a constraint can be fulfilled, temporarily violated or permanently violated. The state of each of the constraints is indicated in the worklist tool by different colors: green for fulfilled, orange for temporarily violated, and red for permanently violated. One example of a temporarily violated constraint is our 'response' constraint between *blood test* and *treatment*: the moment after the task *blood test* is executed and before the task *treatment* is executed it is temporarily violated, but it becomes fulfilled as soon as the task *treatment* is executed. Because all constraints in the model in Figure 6 are *mandatory* (i.e., they have to be followed) the enactment system will make sure they do not reach the state when they are permanently violated. CIGDec constraints can also be *optional*. Optional constraints are showed as dashed lines and are used as a warning system and users can permanently violate them. CIGDec has a developed warning system for optional constraints. Let us assume that it is possible to execute *urine test* without a previous *consult doctor* task, i.e, it is allowed to permanently violate constraint 'precedence' between these two tasks. However, if a user is about to violate this constraint, (s)he should be warned by the system. In this case, we can set the constraint to be optional. In order to generate an informed warning, for each optional constraint we specify: (1) to which group (policy) it belongs; (2) priority level and (3) context related message. Figure 7 (a) shows how policies are defined on the system level. Figure 7 (b) shows the full warning presented to the user when (s)he is about to violate this constraint.



(a) Groups of Constraints

(b) Violation warning

Fig. 7. Specification of optional constraints.

4 Discussion

We have shown that CIGDec can be used to define the degree of flexibility given to a user during the process execution. We have also indicated that a degree of the absolute flexibility can be reached by leaving out all constraints resulting in the freedom given to a user to select any task and execute tasks in any desired order. Since the degree of flexibility has to be controlled in the context of medical care in order to adhere to strict and desirable recommendations, the mandatory and optional constraints have to specified for a modelled guideline. To control the adherence to the specified constraints, the execution engine CIGDec prohibits the violation of the mandatory constraints while allowing the optional constraints to be neglected. All user steps that might result in the violation of constraints are communicated to a user by means of warnings.

The advantages of the proposed CIGDec-based approach over the analyzed modelling languages that employ the imperative approach are as follows:

- CIGDec enables the *flexibility in selection*, meaning that a user executing a model specified in CIGDec gets a freedom in choosing an execution sequence, without requiring this sequence to be thought of in advance and explicitly modelled during the design-time.
- CIGDec enables *late binding*, meaning that it allows to choose an appropriate task at the point of care. This feature is particular important in modelling of CIG since it is not always possible to predict what steps will need to be executed, thus the task selection is case-dependent.
- CIGDec ensures the *absence of change*, meaning that it prohibits choices of users that would violate mandatory constraints.
- CIGDec allows for extendability and allows new LTL formulas to be introduced, thus applicability of CIGDec could be tailored to a specific situation.

The disadvantages of using CIGDec are as follows:

- If a process to be modelled has to be very strict and should allow for flexibility, then the use of CIGDec may result in a complex model.

- CIGDec aims at the modelling of rather small processes, since the description of large processes (containing approximately several thousands of tasks) becomes difficult to understand.

Since both imperative and declarative languages have disadvantages, in order to improve the flexibility of the CIG modelling languages we recommend to augment the CIG languages with the features offered by CIGDec.

5 Related Work

The recent *Workflow Patterns initiative* [2] has taken an empirical approach to identifying the most common control constructs inherent to modelling languages adopted by workflow systems. In particular, a broad survey of modelling languages resulted in 20 workflow patterns being identified [12]. The collection of patterns was originally limited to the control-flow perspective, thus the data, organizational and application perspectives were missing. In addition, the set of control-flow patterns was not complete since the patterns were gathered non-systematically: they were obtained as a result of an empirical analysis of the modelling facilities offered by selected workflow systems.

The first shortcoming has been addressed by means of the systematic analysis of data and resource perspectives and resulted in the extension of the collection of the control-flow patterns by 40 data patterns and 43 resource patterns [22, 24]. The issue of the incompleteness of the control-flow patterns we have resolved by means of the systematic analysis of the classical control-flow patterns against Workflow Pattern Specification Language [14]. Furthermore, we revised the current set of the control-flow patterns and extended it with new patterns. The revised set of the control-flow patterns [23] we have used in this paper to evaluate CIS's modelling languages.

Many workflow systems and standards such as XPDL, UML, BPEL, XLANG, WSFL, BPML, and WSCI were evaluated from the perspective of the control-flow patterns, a summary of which is available at [2].

There have been many attempts to enrich the flexibility of workflow (process) management systems. Case-handling systems are systems that offer more flexility by focusing on the whole case (process instance), instead of individual tasks [21]. An example of such a system is FLOWer [16], where users can 'move up and down' the process by opening, sipping and re-doing tasks, rather than just executing tasks. Although users have a major influence on execution in FLOWer, their actions are seen as going backwards or forward in a traditional process model. Moreover, this might some unwanted side-effects. For example, if the user wishes to execute again (re-do) an earlier task, s(he) will also have to execute again (re-do) all tasks that followed it. Unlike in FLOWer, deviations are not seen as an exception in CIGDec but as 'normal' behavior while the process instance unfolds further according to the choices of users.

Flexibility of process enactment tools is greatly increased by their adaptivity. ADEPT is an example os an adaptive system where users can change the process model during the enactment [20]. ADEPT is a powerful tool which enables users to insert, move and delete tasks form the process instance they are currently working on. However, the user has to be a process modelling expert in order to change the model. Moreover, in medical domain cases may have many differences and adaptations would be too frequent and time

consuming. CIGDec does not see deviations as changes in the model and a good-designed CIGDec model can cover a wide variety of cases.

One of a promising ways to introduce flexibility is to replace imperative by declarative. Various declarative languages "describe the dependency relationships between tasks, rather than procedurally describing sequences of action" [6]. Generally, declarative languages propose modeling constraints that drive the model enactment [6, 13, 30]. Constraints describe dependencies between model elements. Constraints are specified using pre and post conditions for target task [30], dependencies between states of tasks (enabled, active, ready, etc.) [6] or various model-related concepts [13].

6 Conclusions

In this paper, we have proposed a declarative approach which could be applied to overcome problems experienced by the imperative languages used for modelling clinical guidelines. In particular, we have shown how by means of applying the CIGDec language more flexibility in selection can be achieved than the considered CIG modelling languages offer. Furthermore, we showed how the model declared in CIGDec can be enacted. In addition, we discussed differences between the proposed declarative and analyzed imperative languages, their advantages and disadvantages, and made a proposition to combine the features of imperative and declarative approaches in order to increase their applicability and usability.

A CIGDec Constraint Templates

	template	type	LTL formula
a	at most once (0..1)	unary	$\neg(\Diamond(A.completed \wedge \bigcirc(\Diamond A.completed)))$
b	exactly once (1)	unary	$(\Diamond A.completed) \wedge (\neg\Diamond(A.completed \wedge \bigcirc(\Diamond A.completed)))$
c	at least once (1..*)	unary	$\Diamond A.completed$
d	absence (0)	unary	$\neg(\Diamond A.completed)$
e	responded existence	binary	$\Diamond A.completed \Rightarrow \Diamond B.completed$
f	response	binary	$\Box(A.completed \Rightarrow \Diamond B.completed)$
g	choice	binary	$(\Diamond A.completed \Rightarrow \neg(\Diamond B.completed))$ $\wedge \Diamond B.completed \Rightarrow \neg(\Diamond A.completed)$
h	not response	binary	$\Box(A.completed \Rightarrow \neg(\Diamond B.completed))$
i	mutual exclusion	binary	$\Box((A.started \Rightarrow \bigcirc(\Diamond B.started$
			$\Rightarrow (\neg B.started \sqcup (A.completed \vee A.canceled)))))\wedge$ $\Box((B.started \Rightarrow \bigcirc(\Box A.started$ $\Rightarrow (\neg A.started \sqcup (B.completed \vee B.canceled)))))$
j	precedence	binary	$\Diamond B.completed \Rightarrow ((\neg B.completed) \sqcup A.completed)$

Table 4. Some examples of possible enactments of CIGDec model in Figure 6

References

1. AsbruView. http://www.ifs.tuwien.ac.at/asgaard/asbru/tools.html.
2. W.M.P. van der Aalst, A.H.M. ter Hofstede, B. Kiepuszewski, and A.P. Barros. Workflow Patterns Home Page. http://www.workflowpatterns.com.
3. W.M.P. van der Aalst, A.H.M. ter Hofstede, B. Kiepuszewski, and A.P. Barros. Workflow Patterns. *Distributed and Parallel Databases*, 14(1):5–51, 2003.
4. W.M.P. van der Aalst and M. Pesic. Specifying, discovering, and monitoring service flows: Making web services process-aware. BPM Center Report BPM-06-09, BPM Center, BPMcenter.org, 2006. http://is.tm.tue.nl/staff/wvdaalst/BPMcenter/reports/2006/BPM-06-09.pdf.
5. A.A. Boxwala, M.Peleg, S. Tu, O.Oqunyemi, Q. Zeng, D. Wang, and et al. GLIF3: a representation format for sharable computer-interpretable clinical practive guidelines. *Biomedical Informatics*, 37(3):147–161, 2004.
6. P. Dourish, J. Holmes, A. MacLean, P. Marqvardsen, and A. Zbyslaw. Freeflow: mediating between representation and action in workflow systems. In *CSCW '96: Proceedings of the 1996 ACM conference on Computer supported cooperative work*, pages 190–198, New York, NY, USA, 1996. ACM Press.
7. M.J. Field and K.N. Lohr. *Guidelines for clinical practice: from development to use*. Institute of Medicine, Washington,D.C: National Academy Press, 1992.
8. J. Fox, N. Johns, and A. Rahmanzadeh. Disseminating Medical Knowledge: The PROforma Approach. *Artificial Intelligence in Medicine*, 14(1):157–182, 1998.
9. D. Giannakopoulou and K. Havelund. Automata-based verification of temporal properties on running programs. In *ASE '01: Proceedings of the 16th IEEE international conference on Automated software engineering*, page 412, Washington, DC, USA, 2001. IEEE Computer Society.
10. G.J. Holzmann. *The SPIN Model Checker: Primer and Reference Manual.* Addison-Wesley, Boston, Massachusetts, USA, 2003.
11. E.M. Clarke Jr., O. Grumberg, and D.A. Peled. *Model Checking*. The MIT Press, Cambridge, Massachusetts and London, UK, 1999.
12. B. Kiepuszewski. *Expressiveness and Suitability of Languages for Control Flow Modelling in Workflows*. PhD thesis, Queensland University of Technology, Brisbane, Australia, 2003. Available via http://www.workflowpatterns.com.
13. P. Mangan and S. Sadiq. On building workflow models for flexible processes. In *ADC '02: Proceedings of the 13th Australasian database conference*, pages 103–109, Darlinghurst, Australia, Australia, 2002. Australian Computer Society, Inc.
14. N. Mulyar, W.M.P. van der Aalst, A.H.M. ter Hofstede, and N. Russell. Towards a WPSL: A Critical Analysis of the 20 Classical Workflow Control-flow Patterns. Technical report, Center Report BPM-06-18, BPMcenter.org, 2006.
15. N. Mulyar, W.M.P. van der Aalst, and institution = Center Report BPM-06-29, BPMcenter.org year = 2006 annote = http://is.tm.tue.nl/staff/wvdaalst/BPMcenter/reports/2006/BPM-06-29.pdf M. Peleg, title = A Pattern-based Analysis of Clinical Computer-Interpretable Guideline Modelling Languages. Technical report.
16. Pallas Athena. *Flower User Manual*. Pallas Athena BV, Apeldoorn, The Netherlands, 2002.
17. M. Peleg. Chapter 4-2: Guideline and Workflow Models. In Greenes R.A., editor, *Medical Decision Support: Computer-Based Approaches to Improving Healthcare Quality and Safety*. Elsevier, 2006.
18. M Pesic and W.M.P. van der Aalst. A declarative approach for flexible business processes management. In *Business Process Management Workshops*, pages 169–180, 2006.
19. M Pesic and W.M.P. van der Aalst. A declarative approach for flexible business processes management. In *Business Process Management Workshops*, pages 169–180, 2006.

20. M. Reichert and P. Dadam. ADEPTflex: Supporting Dynamic Changes of Workflow without Loosing Control. *Journal of Intelligent Information Systems*, 10(2):93–129, 1998.
21. H. Reijers, J. Rigter, and W.M.P. van der Aalst. The Case Handling Case. *International Journal of Cooperative Information Systems*, 12(3):365–391, 2003.
22. N. Russell, W.M.P.van der Aalst, A.H.M. ter Hofstede, and D. Edmond. Workflow Resource Patterns: Identification, Representation and Tool Support. In O. Pastor and J. Falcao e Cunha, editors, *Proceedings of the 17th Conference on Advanced Information Systems Engineering (CAiSE'05)*, volume 3520 of *Lecture Notes in Computer Science*, pages 216–232. Springer-Verlag, Berlin, 2005.
23. N. Russell, A.H.M. ter Hofstede, W.M.P. van der Aalst, and N. Mulyar. Workflow Control-Flow Patterns: A Revised View (Draft version; request for comments) . BPM Center Report BPM-06-22, BPMcenter.org, 2006.
24. N. Russell, A.H.M. ter Hofstede, D. Edmond, and W.M.P. van der Aalst. Workflow Data Patterns. QUT Technical report, FIT-TR-2004-01, Queensland University of Technology, Brisbane, 2004.
25. A. Seyfang, R. Kosara, and S. Miksch. Asbrus Reference Manual, Version 7.3. Technical report, Vienna University of Technology, Institute of Soft-ware Technology, Vienna. Report No.: Asgaard-TR-2002-1, 2002.
26. Y. Shahar, S. Miksch, and P. Johnson. The Asgaard Project: a task-specific Framework for the application and critiquing of time-oriented clinical guidelines. *Artif Intell Med*, (14):29–51, 1998.
27. S.W. Tu. The eon guideline model. Technical report, http://smi.stanford.edu/projects/eon/EONGuidelineModelDocumentation.doc, 2006.
28. S.W. Tu and M.A. Musen. A flexible approach to guideline modeling. In *Proc AMIA Symp*, pages 420–424, 1999.
29. S.W. Tu and M.A. Musen. From guideline Modeling to guideline execution: Defining guideline-based decision-support Services. In *Proc AMIA Annu Symp*, pages 863–867, 2000.
30. J. Wainer and F. de Lima Bezerra. *Groupware: Design, Implementation, and Use*, volume 2806, chapter Constraint-Based Flexible Workflows, pages 151 – 158. Springer Berlin / Heidelberg, 2003.
31. J.S. Wald, L.A. Pedraza, M.E. Murphy, and G.J. Kuperman. Requirements development for a patient computing system. In *Proc. AMIA Symp.*, pages 731–735, 2001.