Journal of Biomedical Informatics 43 (2010) 520–536

Contents lists available at ScienceDirect

Journal of Biomedical Informatics

journal homepage: www.elsevier.com/locate/yjbin

Authoring and verification of clinical guidelines: A model driven approach ☆

Beatriz Pérez [a,*], Ivan Porres [b]

[a] Department of Mathematics and Computer Science, University of La Rioja, La Rioja, Spain
[b] Department of Information Technologies, Åbo Akademi University, Turku, Finland

ARTICLE INFO

Article history:
Received 3 April 2009
Available online 4 March 2010

Keywords:
Clinical guidelines
Formal verification
Computer assisted tools
Model driven development
Model transformation

ABSTRACT

Objectives: The goal of this research is to provide a framework to enable authoring and verification of clinical guidelines. The framework is part of a larger research project aimed at improving the representation, quality and application of clinical guidelines in daily clinical practice.
Methods: The verification process of a guideline is based on (1) model checking techniques to verify guidelines against semantic errors and inconsistencies in their definition, (2) combined with Model Driven Development (MDD) techniques, which enable us to automatically process manually created guideline specifications and temporal-logic statements to be checked and verified regarding these specifications, making the verification process faster and cost-effective. Particularly, we use UML statecharts to represent the dynamics of guidelines and, based on this manually defined guideline specifications, we use a MDD-based tool chain to automatically process them to generate the input model of a model checker. The model checker takes the resulted model together with the specific guideline requirements, and verifies whether the guideline fulfils such properties.
Results: The overall framework has been implemented as an Eclipse plug-in named GBDSSGenerator which, particularly, starting from the UML statechart representing a guideline, allows the verification of the guideline against specific requirements. Additionally, we have established a pattern-based approach for defining commonly occurring types of requirements in guidelines. We have successfully validated our overall approach by verifying properties in different clinical guidelines resulting in the detection of some inconsistencies in their definition.
Conclusions: The proposed framework allows (1) the authoring and (2) the verification of clinical guidelines against specific requirements defined based on a set of property specification patterns, enabling non-experts to easily write formal specifications and thus easing the verification process.

1. Introduction

In the last decade, the health sector has used clinical guidelines and protocols as helpful instruments for decision making[1]. As defined by the Institute of Medicine, clinical guidelines are *systematically developed statements to assist practitioner and patient decisions about appropriate health care for specific clinical circumstances* [1]. In other words, they describe all the decision points and corresponding actions to be carried out depending on a specific patient's state or situation. They identify the clinical tests to be performed in order to confirm or determine the patient's state. Based on the test results, the guideline determines the treatment alternatives.

Among the most important potential advantages of documenting and using clinical guidelines are assessing and improving the quality of care, providing support for medical decision-making, controlling health care costs and reducing both practice variability and the inappropriate use of resources [1,2].

As a long term research goal we are working on the development of an overall framework aimed at improving the representation, quality and application of clinical guidelines in daily clinical practice. Particularly, we propose to represent clinical guidelines using a visual computer language (UML 2.0 statecharts [3]). This model is taken as the starting point for the verification and tool development processes we propose, and which are aimed at improving the quality of clinical guidelines and developing clinical guideline-based decision support systems (GBDSS) to facilitate patient care at the point of need respectively.

In this article we focus on our approach to improve the representation and verification of clinical guidelines being the main contributions threefold.

☆ This work has been partially supported by the Spanish Ministry of Science and Innovation, project TIN2009-13584, by the Spanish Ministry of Industry, Tourism and Commerce, project LISBioBank (TSI-020302-2008-8), by the Government of Aragon, project LIS (PI108/08) and by Programa Europa CAI-Gobierno de Aragon.
* Corresponding author. Fax: +34 941 299 460.
E-mail address: beatriz.perez@unirioja.es (B. Pérez).
[1] Although clinical guidelines and protocols are different tools (protocols are used for more specific clinical problems than guidelines), in this paper we use the term *clinical guideline* or simply *guideline* to refer to both terms.

1532-0464/$ - see front matter © 2010 Elsevier Inc. All rights reserved.
doi:10.1016/j.jbi.2010.02.009

Firstly, we present an approach to represent clinical guidelines using a visual computer language (UML 2.0 statecharts [3]). The objective is to improve the representation of such guidelines and thus simplify quality improvement and the development of new computer tools for their processing. Based on our experience with several real-life guidelines, we provide a set of representation patterns in order to assist in the modelization process of each clinical guideline as a UML statechart.

Secondly, we have established a pattern-based approach for defining commonly occurring types of requirements in guidelines in order to help non experts in their formal specification.

Thirdly, we have developed a framework based on Model Driven Development (MDD) techniques [4], and in particular in a Model Driven Architecture approach (MDA), to verify specific requirements in guidelines in order to be checked against semantic errors and inconsistencies in their definition.

This paper is based on previous works published by the authors of this paper [5] and with Domínguez et al. [6,7]. In this paper we provide a revised and extended version of these works presenting the first proposal which includes a complete view of our overall framework focusing on the authoring and verification of clinical guidelines.

The paper is structured as follows. Next we outline the background and related work of this research. In Section 2, we present an overview of our overall approach and introduce the IRC guideline as the case study used throughout the paper. In order to make our paper self-contained in Section 3 we describe our approach for formally representing clinical guidelines as UML statecharts. In Section 4, we describe in detail the overview of the verification framework we propose for clinical guidelines. Section 5 discusses the advantages and limitations of our approach. Finally, conclusions and further work are set out in Section 6.

1.1. Background and related work

1.1.1. Formal representation

There is a large number of published guidelines since each guideline is focused on a desired health outcome. Furthermore, guidelines may vary from hospital to hospital since they reflect variations in resources, as well as in the working philosophy of the hospital in question. Because of the vast amount of clinical guidelines, several organizations have undertaken efforts to publish them (using text formats such as HTML or PDF) in the literature and on the internet to make them more accessible and to enable evidence-based knowledge to be reused (see for example [8,9]). During the past thirty years there have been several efforts to develop various computer-interpretable models and tools for the management of guidelines (see GLIF [10,11], Asbru [12–14], EON [15], PROforma [16,17], GEM [18,19], GLARE [20,21] or [22,23] for a review) which are mainly aimed at providing guided support to the physician during the application of the guideline. They are designed to provide different services such as specification and visualization of guidelines, and a close interconnection with clinical data which would simplify its storage, updating and tuning. In particular, these proposals address the problem of a formal representation of guidelines by providing a set of representation primitives that capture their structure [24]. Although the degree of formality of the representation format varies among these approaches, the majority of them represent the guidelines in a format which is precise enough to allow the enactment of the guideline in a (semi)-automatic fashion. Nevertheless, the modeling of clinical guidelines using some of these methods can entail a significant effort [25], and in some cases even understanding the rules governing their semantics involves a difficult task for the user [12,26], which can make the modeling process very difficult and even error-prone.

Taking this into account, in [6,7,27] the authors of this paper together with Domínguez et al. proposed to represent the dynamics of each clinical guideline by using UML 2.0 statecharts [3]. Statecharts have been successfully used to represent the behavior of different and varied kinds of systems including air traffic control systems [28], embedded systems [29] or even biological [30,31] and perioperative processes [32]. In particular, it is worth highlighting that in [32] it is shown that statecharts capture successfully the behavioral aspects of surgical care delivery. We believe that representing clinical guidelines by our approach contributes to making the representation of clinical guidelines a guided and supported process.

1.1.2. Validation and verification

However, the work done on developing, disseminating and computerizing guidelines far exceeds the efforts in improving their quality [25,33]. The fact is that although guidelines have been based on the development of consensus among experts taking into account evidence-based medicine and daily medical practice, this process has limitations and can lead to flawed conclusions [34]. Additionally, the fact that clinical guidelines are commonly represented in natural language makes them accessible to practitioners, but they can also contain ambiguities possibly leading to their inappropriate use. As a consequence, most clinical guidelines are lacking in quality because of the inconsistency and poverty of the methodological rigor used to define them. In fact, many works in the literature have assessed the quality of clinical guidelines finding that most guidelines housed by authoritative institutions are lacking in quality [2]. They therefore require a reevaluation and improvement [2,35] both to make sure that they do not lead to undesired situations (for example, regarding the treatment of a disorder, that the guideline precludes the prescription of redundant drugs) and to verify that they hold expected properties (such as that the application of the guideline finally leads to the purpose for which it was developed).

Given this situation, recent efforts have been made to stimulate the improvement of clinical guidelines using different verification techniques such as theorem proving [12,25,36–41], model checking [33,42–45] or knowledge–based verification [46,47]. In these approaches, a guideline is modeled in some predefined guideline representation language (Asbru, GLARE, etc.) resulting in a first model which is translated into the input specification language of the chosen technique (such as PROMELA [48]), obtaining a second model. Then, properties specified in a formal language (such as LTL [49] (Linear Temporal Logic) [33,43–45], CTL [50] (Computation Tree Logic) [43,44], ACTL (Action Computation Tree Logic) [42], or a variant of ITL [51] (Interval Temporal Logic) [12,25,36–41]) are checked in this second model in order to verify the guideline.

In our particular case, in order to develop decision support systems for guidelines [6,7], we decided to extend our approach by applying verification techniques to guidelines in order to generate GBDSSs of guidelines which have been previously verified against quality requirements. Specifically, we have chosen the SPIN [48] model chequer which uses PROMELA (PROcess MEta LAnguage) [48] as the input specification language and LTL (Linear Temporal Logic) formulas [49] to specify the properties to be verified in the model. Starting from the statechart that represents a guideline, we use a MDA tool which allows us to automatically translate the statechart into the PROMELA specification. The resulted model is taken as input of the SPIN tool which, together with a specific requirement specified in LTL, verifies it in the guideline.

Comparing our approach with different verification techniques that have been used in the literature to improve the quality of guidelines, in particular in [25,41] a theorem proving approach is proposed. Firstly, the guideline is represented using the Asbru language [23] and then it is translated into a formal representation in the KIV theorem prover [52]. The authors use a variant of Interval

Temporal Logic (ITL) to formulate properties. In contrast to this verification technique, we have decided to use a model checker based on the fact that, as is shown in [53], if a property fails to hold, the result returned by a theorem prover is not normally useful for users, who must try to determine whether the fault lies with the system and property being verified, or with the failed proof. So, the verification process can be tedious and time-consuming, requiring substantial human guidance [54].

Other works such as [33,42] use, as we propose, a model checking approach to deal with the problem of improving the quality of guidelines. In particular, in [33] the GLARE agent-based representation language is used to model guidelines. This language is based on a three layered architecture whose intermediate layer consists of a set of XML documents used to represent guidelines in order to facilitate their dissemination. Later, the SPIN model checker is used to prove several properties in the guideline, which are defined by using LTL formulas. As described in [33], the translation of guidelines to the PROMELA language is implemented in Java taking as input model the XML specification of the guideline. Our proposal takes advantage of the well-known MDA approach making the transformation of the statechart representing a guideline into the PROMELA language an automatic and more agile process, saving on human resource costs. Additionally, we want to highlight that, to our knowledge, this is the first work to use the MDA approach for using model checking techniques, thus opening a new field of application.

In [42] guidelines are modeled in the Asbru language and this model is translated into the input language of the Cadence SMV model checker [55]. Properties are formulated by using the Action Computation Tree Logic (ACTL) language. We wish to emphasize that although our approach is similar to this one from several points of view, the use of different model checkers with their corresponding property specification languages allows the verification of different kinds of properties because of the difference in expressiveness and theoretical complexity of the language used.

1.1.3. Requirements specification

Previous works on the verification of clinical guidelines cover a wide range of properties devoted to detecting both semantic errors in the definition of guidelines and coherence properties concerning the chosen guideline representation language. Although in most of these approaches the work done on the formalization and verification of guidelines has yielded good results, often the time and effort involved has been significant [25]. One of the reasons is that professionals with specific skills are required during the verification process. In particular, formal verification processes require properties to be specified in mathematical formalisms such as temporal logics. This constitutes a big challenge since it requires significant expertise from practitioners who usually do not have solid mathematical backgrounds [56–58]. Additionally, accurately representing a property can also be surprisingly difficult because of all the details that must be taken into account [56]. Hence the importance of providing patterns for commonly occurring types of requirements which enable non-experts in the specification language of the tool to easily write formal specifications and thus easing the verification process.

Taking this into account, we have defined a set of property specification patterns for commonly occurring types of requirements in guidelines. Specifically, we have defined a patterns hierarchy which even could be used to specify requirements in other contexts other than the clinical one.

2. Overview: from a guideline in natural language to an information and decision support system

In this section, we present an overview of our overall framework, represented diagrammatically in Fig. 1.

2.1. Representation of clinical guidelines

Although clinical guidelines are usually expressed in natural language, they conform to a specific structure and can be represented in a computer language. In [6,7,27] the authors of this paper together with Domínguez et al. proposed the representation of the dynamics of each guideline by using UML 2.0 statecharts [3] (see step number 1 in Fig. 1). In order to assist in such a long and complex process as the specification of a clinical guideline could be turn out to, we have also provided several patterns to assist in the modelization process in order to have a better and more understandable representation of every clinical guideline. In particular, these patterns take into account the specific elements and semantics of statecharts and provide representation rules of guidelines by using statecharts in the medical context.

This step is carried out through the actual collaboration between medical domain experts and knowledge engineers. In this phase, knowledge engineers participate in as many face-to-face meetings as necessary with medical domain experts to get familiarized with the domain, terminology, and recommendations of the guideline. Also, medical domain experts learn the specification language from the knowledge engineers, which are the ones familiarized with both the UML statechart language and the representation patterns. Then, knowledge engineers can start to translate the guideline to the UML statechart model by following the representation patterns. During the translation process, any doubt arisen, problem encountered or error identified concerning the understandability of the guideline specification is directly discussed with the medical domain experts. The collaboration between knowledge engineers and domain experts goes on until a final decision regarding the guideline statechart model is reached.

2.2. Verification of clinical guidelines

Based on the increasing importance of improving the quality of clinical guidelines, we apply model checking techniques [59] to the verification of guidelines. In particular, in order to formally verify each guideline against quality requirements, we carry out a second step consisting of three sub steps.

In the first sub step, requirements expressed in natural language are formalized in temporal logic. In order to overcome the difficulty perceived in specifying formal properties expected in guidelines, we have defined a set of property specification patterns for commonly occurring types of requirements (see sub step number 2.1 in Fig. 1). These patterns enable people who are not experts in the specification language of the tool to write formal specifications easing the verification process. Thus, firstly medical domain experts establish in natural language the requirements to be verified in the guideline. The formalization of such requirements in the LTL language is easily carried out by following our property specification patterns, task performed by knowledge engineers in a close collaboration with domain experts who provide support for the understandability of such requirements.

Having formally specified the properties to be verified in the guideline, the second sub step consists of formalizing the guideline into the input specification language of the SPIN verification tool. In order to carry out such a process automatically, in [5] we proposed a statechart to PROMELA transformation approach which we have implemented in the model to text tool MOFScript [60] by defining specific transformations in the MOFScript language (see sub step number 2.2 in Fig. 1). Thus, having automated the statechart to PROMELA model transformation, this second sub step is easily carried out by any professional without specific skills in informatics. Then, in the third sub step, the chosen model checking tool accepts the PROMELA model and the formally specified property. The tool can check if the given model satisfies the property

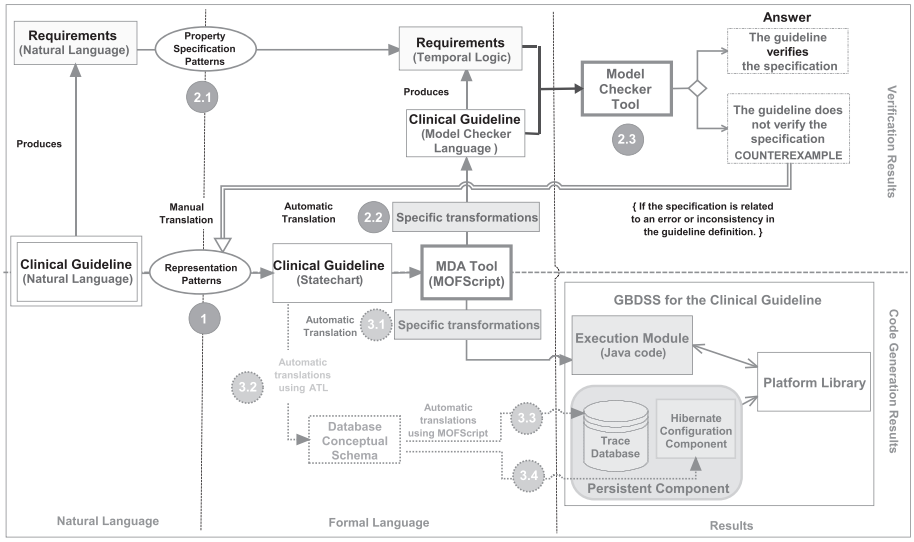

Fig. 1. Proposed workflow.

and it may generate a counterexample otherwise (see sub step number 2.3 in Fig. 1). In this latter case, if the property verified is related to an error or inconsistency in the definition of the guideline, the process will go back to step 1 in which the inconsistency detected will be taken into account for the redefinition of the statechart that represents the guideline.

Using our approach, in order to carry out the verification process of each guideline it is only necessary (1) to manually design the statechart modeling it (from which the PROMELA model is automatically generated) and (2) to formalize the properties to be verified using our property specification patterns.

2.3. Generation of an information and decision support system

Having verified the guideline against quality requirements, the third main step consists in the generation of actual computer applications that help practitioners to follow a guideline in practice.

In [6,7,27] the authors of this paper together with Domínguez et al. proposed the development of a clinical guideline-based decision support system (GBDSS) for the clinical guideline. We defined such a system as a computer-based system which helps health professionals in their decision making concerning the application of a particular clinical guideline to specific patients. In order to achieve this goal, this system has to take into account the indications of the guideline, the current circumstances of the patient and should provide the necessary information to help the physician in her decisions about what to do next.

Following our proposal, the main modules that constitute a GBDSS are the *execution module* and the *persistent component*, which are guideline-dependent, and the *platform library* (see bottom right part of Fig. 1) which have been developed following the Model-View-Controller (MVC) pattern [61]. In particular, the *execution module* corresponds with the Java implementation of the statechart and constitutes the mechanism that allows the physician to apply the GBDSS to patients. The *persistent component*, in the data layer, constitutes the main module which guarantees the persistence of the guideline application. Specifically, it is composed of the *hibernate configuration component*, which is defined to achieve the persistent task and is developed following the Hibernate O/R mapping technology [62], and the *trace database*, which physically stores the data generated during the guideline application and whose instances come from the running of the execution module. Finally, the *platform library* whose definition is independent of the guideline provides standard services of the system related to the implementation of the presentation and the data layers. The main characteristic is that the programming interfaces of the library methods are defined independently of the platform chosen. In this way, the code generated for the execution module has calls to these methods so that, in order to change some characteristics related to the presentation or the data layer, we only have to modify the implementation of the methods in the platform library (not the execution module). As a consequence, having manually developed a platform library under specific criteria (user interface, data storage, etc.), it can be reused in different GBDSS which will share those implementation aspects. And the other way round, different implementations of this library will allow us to run the GBDSS of a guideline with different characteristics.

In [6,7,63,64] we proposed the use of several MDA–based tools to automatically obtain the GBDSS guideline-dependent modules from the manually created guideline specifications represented as a statechart. Particularly, based on this model, we have defined several sets of transformations in two different MDA-based tools in order to automatically generate those modules. As we explained in [6,7], we have defined a set of transformation rules in the previously used MOFScript tool to automatically generate the Java code that constitutes the execution module (see sub step number 3.1 in Fig. 1). On the other hand, the development of the persistent component requires the performance of several database schema mappings as an intermediate step before obtaining the SQL and XML files which generate the two modules of the persistent component [63,64]. For this reason, we have defined a set of transformations in a tool with support for customizable model to model transformations (the ATL MDA-based tool [65]) in order to carry out such schema mappings (see sub step number 3.2 in Fig. 1). Then, we have defined another two sets of transformations in the MOFScript tool to generate respectively (1) the final SQL code which implements the trace database (see sub step number 3.3 in Fig. 1) and (2) the XML files that constitute the hibernate configuration component (see sub step number 3.4 in Fig. 1). Finally, these two guideline-dependent modules generated from the statechart are combined with the *platform library* resulting in the GBDSS for the clinical guideline.

We would like to emphasize that the platform library and the transformations defined in both MDA tools to generate the guideline-dependent modules are independent of the guideline used, so

neither the set of transformations nor the platform library has to be modified to develop the GBDSS for different clinical guidelines. We do not delve into the development of the GBDSSs since it is beyond the scope of this paper but for more information the reader is referred to [6,7,63,64].

We should remark that since the generation of guideline-dependent modules is automatic, sub steps number from 3.1 to 3.4 can be carried out by any professional without specific skills in informatics. On the other hand, the platform library is developed, conforming to specific criteria (user interface, data storage, etc.), by knowledge engineers with programming skills. So, for the performance of this step, collaboration between clinical experts and knowledge engineers is not necessary.

Therefore, following our approach, for the development of the GBDSS for a guideline it is only necessary (1) to design manually the statechart modeling it and develop the platform library under specific criteria (user interface, data storage, etc.), and (2) taking the statechart as source model to automatically generate the GBDSS guideline-dependent modules which, together with the platform library, constitute the GBDSS for the guideline.

Overall, we propose a complete framework aimed at improving the formal representation, quality and computer aided application of clinical guidelines.

2.4. Tool chain

The challenge is to perform these steps in a viable and cost-effective way. The vast amount of clinical guidelines (there are thousands of guidelines in existence), the possibility of changes over time and their dependence on the hospital which uses them, make both the manual verification process and the implementation of every guideline into a software system a long, cumbersome and costly endeavour.

To overcome these challenges, we propose the use of a Model Driven Development (MDD) approach [4], in particular a Model Driven Architecture approach (MDA) [66]. The general idea behind MDD is to focus on models rather than on computer programs, so that the code programs are generated in an automatic way by means of a refinement process [4]. So, starting from the statechart that represents a clinical guideline, we use a Model Driven Architecture tool chain to perform the verification and the implementation processes automatically which means a lower cost than that required for manual verification and implementation.

We wish to remark that the overall verification and tool implementation processes have been implemented as an Eclipse plug-in called *GBDSSGenerator* attending at Guideline-Based Decision Support Systems generator. This plug-in uses and integrates the defined ATL and MOFScript transformations allowing the processes to be carry out automatically by only selecting different menu options the plug-in provides. Thus, using the developed plug-in, we can proceed as follows: A paper-format guideline is represented in a computer interpretable format as a UML 2.0 statechart guided by our representation patterns and using the Borland Together Modelling tool [67]. This tool is a UML 2.0 compliant tool which allows us to graphically represent the statechart and to import it in XMI format obtaining a `.uml2` extension file . At this point, the required properties to be proven in the guideline are specified in the LTL language by using our property specification patterns resulting in several `.ltl` extension files. Then, the GBDSSGenerator takes the statechart `.uml2` extension file as a starting point of the verification process and automatically generates the associated PROMELA model by only selecting a menu option which the plug-in provides. After this, the SPIN model checker takes both the generated PROMELA model and a property of those previously specified in the LTL language and checks whether the given model satisfies the property or not (see sub step number 2.3 in Fig. 1). As described previously, the process requires returning to step 1 in which the inconsistencies detected will be taken into account for the redefinition of the statechart that represents the guideline obtaining a different `.uml2` extension file. Regarding the generation of the GBDSS for the guideline, the GBDSSGenerator takes the final `.uml2` extension file as input model and, by only selecting different menu options which the plug-in provides, automatically generates the guideline-dependent modules of the GBDSS. Then, the generated guideline-dependent modules together with a platform library (previously developed under specific criteria regarding the user interface, the data storage, etc.) will constitute the GBDSS for the guideline.

It should be emphasized that, following our proposal based on MDA, for each guideline the resources (such as time, effort or costs) required to both carry out the verification process and generate the GBDSS are lower than those required for a manual implementation. Overall, it is worth noting that we propose a new application of MDA tools and techniques to support model driven business processes and their mapping to MDA computation models.

2.5. Overview of the case study

In order that the reader can have a better understanding of our approach, we will use as a case study a particular clinical guideline presented in [7]. This guideline is used for the management of infections related to intravenous catheters (IRC) and provides evidence-based recommendations for preventing such infections. In particular, it is used in a Spanish hospital and has been developed on the basis of a guideline published by the US Agency for Health Care Research and Quality (AHRQ) National Guideline Clearinghouse (NGC) [8]. From now on we will refer to this guideline as the IRC guideline.

The use of intravenous catheters is indispensable in modern-day medical practice, especially in intensive care units (ICU), for patients in critical conditions suffering from a wide range of serious illnesses and injuries. The problem is that catheter-related infections have been found to constitute the most common cause of hospital-acquired nosocomial bacteremia and to increase morbidity and mortality [68,69]. Hence the importance of establishing intervention guidelines for the prevention, diagnosis and treatment of infections caused by central catheters. In particular, the IRC guideline establishes the criterion that the physician must take into account to decide whether or not to remove the catheter, as well as what antibiotic treatment must be given to the patient or what clinical actions or tests must be carried out regarding her clinical condition.

Regarding its representation, this guideline is presented as a text document of 10 pages, written in natural language. It also comprises several tables of the treatment options for patients to whom the guideline is applied, and flowcharts which partially explain the behavior of the application of the guideline.

Next, we explain in more detail our approach for the representation of guidelines as UML statecharts in order to make our paper self-contained, and we move on to focus on the validation and verification issue as the main contribution of our approach.

3. Representation of clinical guidelines using UML

Clinical guidelines are normally represented in free text using other related or overlapping approaches corresponding to graphic representations of algorithms, such as flowcharts, decision tables, and protocol charts [1,70]. Although natural language is easy to understand, its potential for ambiguity and the difficulty of accessing long text documents in urgent situations makes the free text

form rather inconvenient for representing guidelines. Furthermore, this form of representation means that in most cases the same information is distributed amongst such different representation methods (flowcharts, decision tables, etc.) that the practitioner is forced to consult all of them in order to gather the information needed. Some clinical researchers, moreover, argue that flowcharts are ineffective for representing clinical algorithms and propose moving towards the use of a *meta-language* that allows the use of computer-based decision aids [1].

In addition, a clinical guideline can be considered to be a kind of medical process. Medical processes have been modeled in many different ways, using both formal and informal methods [23,71]. Formal methods overcome the problem of ambiguity [72], but may be more difficult to understand for health care personnel. It is claimed in [71] that the modeling of medical processes must be done by means of methods that overcome the problems of both formal and informal approaches: methods that, on the one hand, have an easily understandable notation, and on the other hand can be formally analyzed.

3.1. Using UML statecharts to represent clinical guidelines

Taking this into account, we propose the use of UML 2.0 statecharts [3] as a method to represent the dynamics of clinical guidelines. Particularly, statecharts consist essentially of states, which denote a situation of objects during which some condition holds, and transitions, which are the means by which the objects change of state. States can be simple (without substates) or composite, distinguishing between orthogonal composite states, to model concurrent behaviors where several states are active simultaneously, and simple composite states, to specify that only one of its substates must be active.

The reasons for this choice are multiple. UML statecharts allow the user to properly model concurrent behaviors, as well as to represent state hierarchy, increasing both scalability and legibility. As we shown in [7], these model elements of statecharts will play an important role in the representation of clinical guidelines, since guidelines frequently include many situations which have a hierarchical or a concurrent nature. Additionally, UML statecharts are considered to be an easy to communicate and an understandable visual formalism [30] which, as we have described previously, has been applied over a wide range of different and varied kinds of systems.

Another important advantage of using this formalism is that many modeling tool vendors have adopted and supported UML, so there are many support tools available for this modeling language both commercial (such as Enterprise Architect [73], Poseidon [74] or Borland Together Modeling tool [67]) and open source (such as ArgoUML [75] or Eclipse UML2 tool [76]), some of them better than some commercial alternatives (see [77–79] for a review). In particular, many such UML tools conform to the Model Driven Development approach, providing facilities for generating any kind of text from models, in particular, for code generation (see [77,78,80] for a review).

One possible drawback of UML as a whole is the lack of a precise definition. However, the UML subset defining statecharts has been studied and formalized by many researchers [81,82], who have also developed analysis tools as described in [83,84].

To summarize, we consider that using UML statecharts will enable us not only to represent clinical guidelines in a better way than with other informal methods of representation, but also to make available several modeling and code generator tools from which, based on those representations, we can generate code in an automatic way.

3.2. Representation patterns

There is no single criterion to follow to create a statechart representing a clinical guideline, especially taking into account that existing guidelines can be represented in multiple formats. However, based on our experience with several real-life clinical guidelines, we have described several patterns to assist in the modelization process. These patterns take into account the specific elements and semantics of statecharts and provide representation rules of guidelines by using statecharts in the medical context [6,7]. So that this paper may be self-contained, we give a summary of these representation patterns. We emphasize that we do not intend to give an explanation of the elements and semantics of statecharts, but to present the representation patterns by using statecharts in the medical context. These patterns determine, among other things, that the situation of a patient with respect to the application of a guideline is represented by means of states. Usually, these situations can be refined into more substates refering to different discrete steps necessary to treat the patient or to improve the judgment of the physician, which are represented by means of hierarchical states (simple composite states) or concurrent states (orthogonal composite states) which are two of the main components of the expressivity richness of statecharts. On the other hand, the occurrences that cause a patient to change her state are represented by a transition. We do not delve into more detail of the description of the representation patterns, but refer to [7] for a more complete description.

We wish to remark that, although UML seems a suitable notation for representing the guidelines, we have nevertheless found some drawbacks concerning its use in our work. UML is considered to be the de-facto standard for modeling software systems but it is also true that UML does not have one single and precise action semantics [85]. In addition, UML proposes a wide variety of specialized actions. We want to avoid this, due to our intention of representing clinical guidelines as easily as possible. Therefore, as part of our representation patterns, we have had to define our own action language for representing clinical guidelines, and which complies with the UML semantic [3]. In particular, this action semantics includes different actions and events defined based on several identified situations which commonly take place in everyday clinical practice and, in particular, in clinical guidelines' application [7] (for example, we have defined an event to represent the arrival of the results of a specific clinical test testResultsArrive(testName)). On the other hand, when dealing with guidelines representation models and, in particular, with the verification and validation of clinical guidelines, the use of ontologies comes into play. Regarding this issue, we would like to note that such events and actions have to be defined within an ontology, which constitutes the link between our work and previous defined ontologies. In other words, our approach does indeed need to use a formally defined ontology of medical terms that contains all the different actions and events that appear in a clinical guideline. However, we consider that our approach is independent of the actual ontology and it can be used with different ontologies.

In order that the reader can have a better understanding of the defined patterns, next we briefly explain them by describing an excerpt of the statechart defined from the IRC guideline.

3.3. Example: the IRC guideline as a statechart

As mentioned above, the IRC guideline consists of a 10 page text document written in free text form together with other representations such as flowcharts or treatment tables. As would be expected, for the representation of the guideline as a UML statechart a physician with specialized skills has been required in order to help us to understand the specific medical background

surrounding the guideline. Finally, it has been proved that the defined statechart provides a unified, more understandable and efficient way of representing the source IRC guideline.

A fragment of the statechart defined from the IRC guideline is shown in Fig. 2. Firstly, the patient is in the state *Patient is not suspected to have IRC*. If the patient shows specific symptoms (such as signs of local infection or high fever without any demonstrable infectious cause), which is represented by the triggered of the event *symptomsAppear*(), the IRC guideline is applied and the state changes to *Guideline is being applied to the patient*. During the course of the application of the guideline, the patient can be suspected to have (represented by the state *Patient is suspected to have IRC*) or can definitely have IRC (depicted by state *Patient with diagnosis of IRC*). While the patient is in the former state, several clinical test are carried out (for example, maki and hemoccult tests). The clinical test results will determine whether the patient has an infection related to the intravenous catheter. In the affirmative answer, the patient changes her state to that of being diagnosed with IRC (*Patient with diagnosis of IRC*) in which case she is treated with appropriate antibiotics in accordance with the specific bacterial infection until she recovers from it. For both the negative answer and when the patient is treated and finally recovers from the infection, the patient changes her state again to that of being not suspected to have IRC (*Patient is not suspected to have IRC*).

4. Formal verification of clinical guidelines

The objective of the formal verification process is to ensure that a given guideline exhibits a number of desired properties. At this point, two issues come into play. On the one hand, we have to establish and formally define the requirements to be verified in the guideline. On the other hand, we have to choose a formal verification technique which provides an effective and efficient way to verify guidelines. We address these issues in the following subsections by giving our verification approach. We conclude this section by showing our experience of using this approach with the IRC guideline.

4.1. Verification requirements for clinical guidelines

Verification requirements in a guideline have been studied in the past by several authors [12,25,33,36–47,86–89]. Based on the classification introduced in [42] and in [41], in [5] the authors of this paper considered two main kinds of properties: *medical properties* or properties at the conceptual level, and *structural properties* or properties at the implementation level. Medical properties refer to aspects such as clinical parameters, physicians' actions or overall intentions of the guideline. Structural properties, on the other hand, specify the general correctness requirements related to the formal representation chosen to model the guideline. In [5] we gave a first approximation for a subclassification of medical and structural properties based mainly on [33] and on [46] respectively.

In the present paper we focus on medical properties since we consider them to be the most interesting and generalizable ones. We redefine and extend the previously established classification developing a pattern–based framework for overcoming the perceived difficulty of specifying formal properties expected in guidelines [56–58]. In order to establish such a framework, we have analyzed numerous works in the literature: those dealing with general medical practice and which focus on clinical guidelines development and content [1,2,34], and those whose goal is the formal proving of guidelines [12,25,33,36–47,86–89]. We have drawn two main conclusions from these comparisons: (1) guideline requirements are determined or established from different sources which guideline properties are expected to hold and (2) to the best of our knowledge, there are no approaches devoted to the definition of property patterns to enable non experts to define guideline specifications. Only [38] provides reasoning patterns to specify guidelines control structures and in [38,39,86] to determine the behavior of treatment selection, but these are provided in an inaccurate fashion. Taking this into account, firstly we have determined different requirement sources on which guideline properties are expected to hold, distinguishing among the following: *good medical practice*, the *particularities of the hospital*, the *guideline goal* and the *patient specific clinical condition*. Secondly, we have identified the properties the analyzed works consider useful to be verified in guidelines and abstracted these properties from particularities. Based on these requirements, the idea is to establish a set of property specification patterns which provides complete support for the formal specification of commonly occurring guideline properties. Then, each property formulated in natural language in a specific requirement source background conforms to one of the defined patterns which, by providing the mappings of the property to formal specification languages enables developers to easily formulate the requirement to be checked against the guideline. Next, we explain in detail both the different requirement sources and the property specification patterns we have considered.

4.1.1. Sources of verification properties

We consider that the verification of clinical guidelines against expected requirements can be done within the scope of different requirement sources depending on the aim for which these requirements are checked on the guideline. In order to establish these sources, we have based them not only on our experience but also on the previous cited works which deal with both general medical practice and the formal proving of clinical guidelines. We consider that the verification of guidelines against expected properties can be done taking into account at least four sources described below:

- *Good medical practice.* Clinical guidelines must conform to the corresponding medical standard, which is defined by scientific knowledge, practical experience and professional acceptance. For example, one would expect that a good-quality medical guideline regarding treatment of a disorder would preclude the prescription of redundant drugs, or advise against the

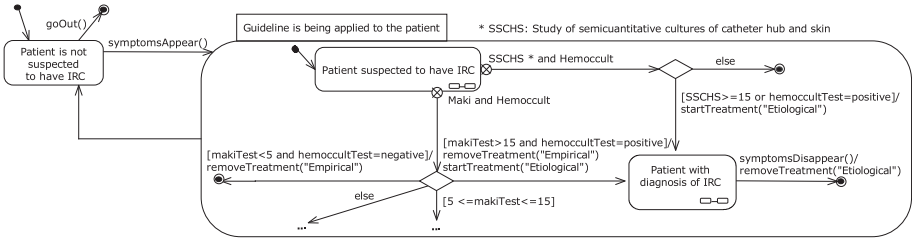

Fig. 2. Part of the statechart defined from the IRC guideline.

prescription of a treatment that is less effective than some alternative. Requirements defined within the scope of this source aim at verifying whether the guideline conforms to good medical practice. Checking such properties could be valuable, in particular during the process of designing and adapting medical guidelines to the possible changes in their definition that could take place over time.

Since principles of best care practice become a medical standard for optimal care, the properties defined in this requirement source can be checked in all guidelines, as long as those properties are related with the indications of the guideline (as far as prescribed treatments, medical actions to be carried out, etc. are concerned). In particular, in [38] the authors consider the knowledge surrounding this source to be the *metaknowledge*, a term that we have also adopted to refer to the knowledge about good medical practice.

- *Particularities of the hospital*. Each hospital has different treatment facilities and resource levels which can lead to them producing their own versions of clinical guidelines. Additionally, the particularities of the hospital in which a guideline is applied may impose several limitations on the regular application of guidelines related, for example, to the reduction in the use of certain pharmacological treatments or the lack of specific hospital resources (such as laboratory instruments). The consequences that may result from such limitations have to be identified and thoroughly analyzed in order to make sure that in the case of a specific set of resources being available (or not available) there is a therapy for a patient to which a guideline is being applied. To sum up, this source concerns the contextualization of the guideline to a given hospital.

  Regarding the scope of the requirements defined in this source for a specific hospital, this can be checked in any clinical guideline used in that hospital, since optimization of the hospital resources may affect the regular use of the guidelines in daily practice.
- *Guideline goal*. This source refers to the aspects or factors intrinsic to the guideline by itself, such as the conditions and clinical problems it covers or the desired outcomes. Properties defined in this requirement source can be used, for example, to verify whether the guideline contains a path requiring specific support services needed for a given treatment. Additionally, as is claimed in [34], guideline documentation should be assessed to determine whether the guideline conforms to the principles outlined in it. So, verifying that the application of the guideline finally leads to the purpose for which the guideline was developed constitutes an interesting property to be checked. Taking this into account, some of these defined requirements could be reused to be checked against both other versions of the guideline and guidelines developed for the same purpose.
- *Patient specific clinical condition*. A clinical guideline describes a set of alternative paths the physicians can choose during the diagnostic process. This source refers to the possible paths the guideline proposes to be carried out given a specific patient clinical state. The properties defined in the scope of this source can be used to check the feasibility of a given action, or path of actions on the patient or to prove whether there is a therapy for a patient under these clinical conditions. So, the feasibility of future physician actions can be proven before carrying out the action. In this case, the defined requirements are, in most cases, specific to the guideline.

4.1.2. Property specification patterns

Given the lack of previous works published on the subject of specification patterns in the medical context, we have looked at the literature on property specification patterns in general [56–58,90,91]. Of particular interest is the work of Dwyer et al. [57] whose specification patterns have been increasingly used in the literature in a wide range of contexts (for example, in web service applications [58], cash management systems [91] even in the medical domain [56]) to specify commonly occurring types of properties. For this reason, we have taken this approach as a starting point for the definition of our property specification patterns. First, we briefly present Dwyer et al.'s approach and then move on to describe our proposal built on it.

In this section we assume that the reader is familiar with both Computational Tree Logic (CTL) and Linear Temporal Logic (LTL). Otherwise, the reader is referred to [50] and [49] respectively.

Dwyer et al. patterns. Dwyer's approach for property specification patterns consists of the hierarchy of patterns shown in the Fig. 3. As is shown in the figure, patterns are classified into occurrence and order. *Occurrence patterns* represent requirements related to the existence or absence of certain states/events during a defined interval of time. On the other hand, *order patterns* are used to represent a certain ordering of states/events during a defined interval of time. Additionally, each pattern has a *scope* which is used to specify the part of the program execution over which the pattern must hold. Examples of these scopes are *Global*, *Before* and *After*. In particular, in the *Global* scope the entire program execution is considered. In the *Before* scope the execution is considered up to a given state or event. The *After* scope means that the property must hold after the execution of a given state or event. So, in this approach, patterns specify *what* must occur while the scope specifies *when* the patterns must hold [58].

For each specification pattern, Dwyer et al. provide mappings to several formal specification languages (such as Computational Tree Logic (CTL), first proposed in [50], or Linear Temporal Logic (LTL) [49]) presented as temporal logic formulas. These formulas contain one or more variables or predicates that the user has to substitute with valid values from the model in order to obtain the specific property to be verified later in the model. Therefore, the instantiation of patterns to construct specific properties consists of choosing the suitable pattern and filling in the pattern's variables of the formula. More details of these patterns can be found in [57] or in [92] where the complete list of specification patterns is shown.

In order to make sure that this approach is complete enough for representing the widest possible spectrum of guideline properties, firstly we have considered the analyzed works which deal with the formal proving of guidelines. We have identified the properties these works consider useful to be verified in guidelines and ab-

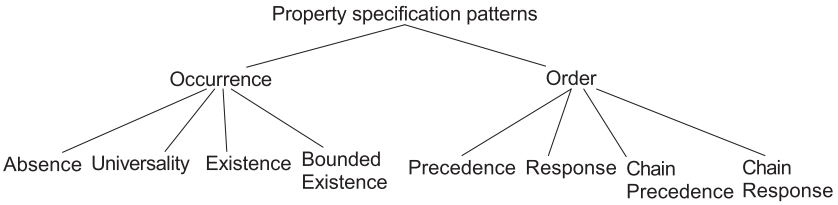

Fig. 3. Dwyer's property specification patterns hierarchy.

stracted these properties from particularities. Starting from the collected properties, we have based on Dwyer et al.'s property specification patterns and manually determined whether each property matches a Dwyer pattern following a documentation process. In particular, as is proposed in [92] (pp. 416), for each property we have recorded, when it has been possible, the following information: (1) the description of the property in natural language (*Requirement*), (2) the pattern of which the property is an instance (*Pattern*), (3) the scope of the pattern (*Scope*), (4) the parameters provided to the template (*Parameters*), (5) the property specification in the formal specification language chosen (CTL, LTL, etc.), (6) the source of the property such as the authors and citation of the paper (*Source*), (7) the application domain which in our case has been presented by indicating the specific clinical guideline for which the property is defined (*Domain*) and (8) any additional information needed (*Note*). We would note that the documentation process followed has required a significant effort since it has been necessary to thoroughly read each paper, understand the description in natural language of the clinical guideline used (in most cases part of the definition of the guideline was not provided) and check and identify the formal specification of the properties given in the paper. As a result, we have collected 54 requirements from the 19 analyzed papers. Due to space reasons, we do not include the complete documentation of these properties, but we show in Fig. 4 the number of properties that the most significant analyzed papers define matching each pattern. In particular, from this analysis we have concluded that, while the vast majority of the properties match one or other of Dwyer et al.'s patterns, some of them have not been identified as being instances of any of these patterns. These latter properties have the particularity of being related to the existence of at least one path in the system (the guideline application in our specific case) in which some conditions must hold (from now on we will refer to this type of property as *properties of existential nature*). By way of example, in Fig. 5 we show four properties from the 54 collected which conform to this type. The fact that this property type is not matched in Dwyer's patterns (confirmed with Mr. Dwyer via email) has led us to slightly tailor the original pattern hierarchy to give support to its specification. It must be said that, although their proposal has an existence pattern, in principle this pattern is defined in order to describe a portion in all executions of a system's execution that contains an instance of certain events or states [92], which differs from what we want to specify.

Our approach for the property specification patterns. In order to have an extension of the Dwyer et al.'s patterns which provides support for the specification of properties of an existential nature, we have searched the literature for works proposing patterns to specify properties of this type. We have mainly focused on those works which have used the approach of Dwyer et al. As a result of this analysis, we have found that the work of Ryndina et al. [90,91,93] provides a proposal for the extension of Dwyer et al.'s *Existence* pattern. These authors have developed a tool named the *SUM Analyser* which allows users to define UML use cases, validate different types of properties (generic and specific, defined using CTL) in these use cases and obtain an interpretation of the results in terms of those use cases. In particular, for the definition of specific properties this tool uses a tailored version of Dwyer et al.'s approach which consists of excluding those patterns that are not commonly used and including others to suit the requirements of the use case model analysis. The patterns hierarchy finally proposed by Ryndina et al. is shown in Fig. 6. As can be seen in this figure, their approach for the extension of the *Existence* pattern consists of considering four new subcategories:

- *Everywhere eventually.* In order to represent that something will always eventually happen, no matter what execution path is taken. The associated CTL formula is `AFp`. We note that this subpattern matches with the *Existence Global* pattern of Dwyer et al.
- *Possible existence.* It is possible for something to happen, that is, a property may hold in some paths but not all the paths of the execution. The corresponding CTL formula is `EFp`. This pattern is strongly related to Dwyer et al.'s *Absence* pattern.
- *Always eventually.* No matter where in the system execution we are, something will always eventually happen. The CTL formula is `AG(AFp))`. This pattern is a stronger variation of the *Everywhere eventually* pattern.
- *Liveness.* At any time during the execution of the system, something will eventually become possible. The corresponding CTL formula is `AG(EFp)`. This pattern is a stronger variation of the *Possible existence* pattern.

It is worth pointing out that these authors only consider the Global scope [90] since, as is claimed in [57], this scope is the most used in the property specifications.

Taking into account this approach (1) we have checked whether the properties of an existential nature that we have collected from the literature can be considered as instances of the new patterns proposed by Ryndina et al., and (2) we have given an extension proposal where necessary. In particular, of the four properties of existential nature presented in Fig. 5, those labelled (a), (c) and (d) could be mapped to the *Possible existence* pattern. In these properties, the variable or predicate `p` has been substituted by another composed formula which refers, at the same time, to the existence of another path of the guideline representing a certain ordering of actions. Following Dwyer et al.'s methodology for defining chain patterns, we have defined a subpattern of *Existence* named *Chain Possible Existence* in order to give support to this type of property. The formal definition of this subpattern is the following:

Dwyer et al.'s patterns		References	Teije, Marcos et al. [25, 41]	Terenziani, Giordano et al. [33, 45]	Balser et al. [37]	Hommersom et al. [38]	Lucas [39]	Groot et al. [43]	Hommersom et al. [44]
Occurrence	Universal	Global	1		1	9	2		6
		After				6	8		
	Absence	Global							2
		Before							1
		After		1					5
	Existence	Global		1		1		1	
Order	Response	Global							14
		After	1						

Fig. 4. Number of properties matching each pattern.

Requirement: There exists a path in which, eventually, for a patient diagnosed with Ductal Carcinoma In Situ (DCIS) a possible treatment could be an axillary staging by sentinel node (asbSN) followed by a breast conserving therapy (bct).
Pattern: Unknown
Scope: Unknown
Parameters: Propositional
CTL: EF(DCIS & EXEF(asbSN & EXEF bct))
Source: P. Groot et al. [43] (pp.7)
Domain: A Dutch breast cancer guideline.

(a)

Requirement: There exist some execution on which the angiography is not required.
Pattern: Unknown
Scope: Unknown
Parameters: The formal specification is not provided in the paper.
Source: P. Giordano et al. [33] (pp.4) and P. Terenziani et al. [45] (pp.4)
Domain: Medical guideline for "stroke", developed in collaboration with Azienda Ospedaliera San Giovanni Battista in Turin.

(b)

Requirement: The following treatment path for surgery of operable invasive breast cancer occurs in the protocol: There exists a treatment path in which provided that there are contra indications neither for BCT nor for SN, the sentinel node is carried out with negative results. It continues with the excision of the tumour with the particularity that the tumour margins that have been resected are tumour free. In this situation neither a total resection of the breast (mastectomy) nor a dissection of the axillary nodes (AD) should be carried out.
Pattern: Unknown
Scope: Unknown
Parameters: Propositional
CTL: EX(!CI-BCT & !CI-SN & EF(sentinel-node & SN = neg & EF(tumour-excision & TF & AG(!mastectomy & !AD))))
Source: A. Hommersom et al. [44] (pp.4)
Domain: The protocol of the Dutch Integral Cancer Centre East (IKO in Dutch), which was based on the CBO guideline.
Note: The specification of the requirement is not explicitly provided in the paper.

(c)

Requirement: The following treatment path for surgery of operable invasive breast cancer occurs in the protocol: Provided that there are contra indications neither for BCT nor for SN, the sentinel node is carried out with negative results. It continues with the excision of the tumour with the particularity that the tumour margins that have been resected are tumour free. Then, a total resection of the breast (mastectomy) is recommended, so a dissection of the axillary nodes (AD) should not be carried out.
Pattern: Unknown
Scope: Unknown
Parameters: Propositional
CTL: EX(!CI-BCT & ! CI-SN & EF(sentinel-node & SN = neg & EF(tumour-excision & !TF & EF(mastectomy) & AG(!AD))))
Source: A. Hommersom et al. [44] (pp.4)
Domain: The protocol of the Dutch Integral Cancer Centre East (IKO in Dutch), which was based on the CBO guideline.
Note: The specification of the requirement is not explicitly provided in the paper.

(d)

Fig. 5. Existential properties which do not match with Dwyer et al. patterns.

Chain Possible Existence. There exists at least one path in the system execution in which a specific set of states/events p1...pn takes place. This pattern is the generalization of the Possible Existence pattern.

Although the properties labelled (a), (c) and (d) in Fig. 5 specify the ordering occurrence of actions during the application of the guideline, we have also considered the definition of properties related to the existence of paths in the guideline in which some clinical actions must take place in any order. For this reason, we propose the specialization of the previous pattern *Chain Possible Existence* by considering the two possibilities (with and without order), giving rise to two subpatterns: *Chain Possible Existence with order* and *Chain Possible Existence without order* respectively, whose definition and associated formulas in CTL and LTL are the following:

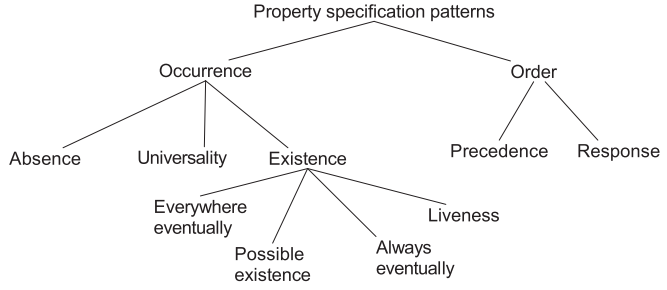

Fig. 6. Ryndina's property specification patterns hierarchy.

Chain Possible Existence with order. There exists at least one path in the system execution in which a certain ordering of states/events p1, p2, p3 takes place.
CTL: EF(p1 & EF(p2 & EFp3))
LTL: Do not supported

Chain Possible Existence without order. There exists at least one path in the system execution in which certain states/events p1, p2, p3 take place no matter what the order.
CTL: EF(p1 & EF(p2 & EFp3)) | EF(p1 & EF(p3 & EF p2)) | EF(p2 & EF(p1 & EF p3)) | EF(p2 & EF(p3 & EF p1)) | EF(p3 & EF(p1 & EF p2)) | EF(p3 & EF(p2 & EF p1))
LTL: Do not supported.

On the other hand, property (b) in Fig. 5 shows that there exists at least one path in the guideline application in which a certain clinical test (in particular the angiography technique) is not re-

quired to treat the patient. This property can be represented formally in CTL as EG!p, where *p* represents the application of the clinical test. This property does not match with any pattern either of Dwyer et al. or of Ryndina et al. We therefore suggest the inclusion of another pattern in our proposal which is a subpattern of *Existence*. This pattern is named *Possible absence* and represents the fact that it is possible for something not to happen. As with the possible existence, this pattern is strongly related to Dwyer et al.'s *Absence* pattern. The formal specification of this pattern is the following:

Possible Absence. There exists at least one path in the runtime guideline application in which a certain state/event does not happen.
CTL: $EGp \equiv AFp$
LTL: Do not supported.

Regarding the expressivity of the LTL and CTL temporal logic languages for specifying the patterns considered in our proposal, we would like to highlight as an important issue in our work that the properties of existential nature (*Possible Existence*, *Chain Possible Existence with and without order* and *Possible Absence*) can not be directly represented by the LTL language [94–98]. Nevertheless, by using specific verification techniques, such as model checkers, the verification of such properties with LTL is possible by means of the verification of the negation of the property. We would also stress that the *Liveness* pattern in Dwyer's et al. and Ryndina's et al. approaches is not supported by LTL [94–98]. In particular, the formula associated in CTL is AG(EFp) and neither this formula nor its negation are represented in LTL. This is why model checking techniques can not be used to decide whether this kind of formula is true or not [99].

As for the instantiation of patterns to construct specific properties, since we represent each guideline as a statechart, in the LTL or CTL formulas obtained by using the requirement patterns, each predicate in these formulas models the fact that (1) a state in the statechart is active, (2) an event has been triggered (3) an action has been carried out or (4) the patient has a specific clinical condition.

Our final property specification pattern hierarchy built upon Dwyer and Ryndina et al.'s proposals can be seen in Fig. 7 in which the new proposed subpatterns are depicted on a square. To sum up, we emphasise that since the proposed patterns hierarchy gives support for representing the complete list of 54 requirements collected from the 19 analyzed papers, we think that this approach is sufficiently complete for representing a wide spectrum of guideline properties.

4.2. Verification process

There are many formal verification techniques, two of the best known being theorem proving [53] and model checking [59]. In [5], we decided to use a model checker for the verification of clinical guidelines, based mainly on the three comparison aspects for these two approaches presented in [100]. The reasons for taking such a decision are that (i) following our approach each guideline is represented by a statechart, so the state space is finite, (ii) model checking is completely automatic [54], and (iii) using a model checker, counterexamples are automatically generated. Additionally, we have also considered two other comparison aspects. Since, unlike theorem proving, model checking cannot be applied to systems with an arbitrary large number of processes, the fourth comparison aspect is related to the number of processes of the system. In our particular case, only one statechart is defined for each guideline, so there is no problem with the number of processes in the model. The fifth comparison aspect is related to the fact that, in the verification process of model checking, the behavior of the system is modeled as a finite state machine called a Kripke structure [101]. A Kripke structure is basically a graph whose nodes represent the reachable states of the system and the edges represent state transitions. Therefore, for its state-based nature, UML statecharts have many similarities with Kripke structures and they can be easily converted to these structures. Taking all this into account, the use of a model checker seems to be a natural choice.

Among the different model checkers in the literature, we have chosen the SPIN model checker [48] based on several criteria. SPIN can be used in three basic modes: as a simulator, as a verifier and as a proof approximation system [48]. During simulation and verification the tool checks for the absence of deadlocks, unspecified receptions, and executable code. In addition, the use of SPIN can be particularly useful for the purpose of refutation [54], which we consider especially interesting in the verification of guidelines. As we have specified in Section 2, as a model checker the verification process of SPIN consists of the following steps. Firstly, the system is modeled in the specification language of the tool. Next, the properties of the system are specified, usually using temporal logic formulas. Then, the model checker accepts the model and a property that the system is expected to satisfy. The tool outputs *yes* if the given model satisfies the property and generates a counterexample otherwise. In particular, as we have commented previously, in SPIN the input specification language is PROMELA [48], while the properties to be verified are represented by LTL formulas

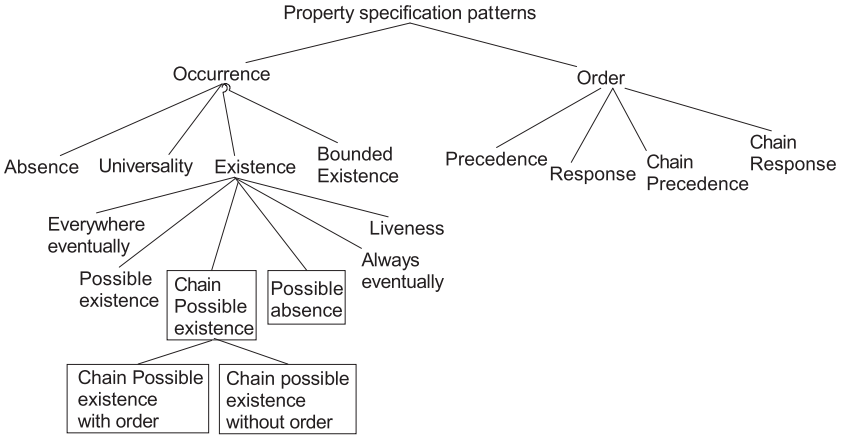

Fig. 7. Our property specification patterns hierarchy.

[49]. We now discuss our approach for translating a clinical guideline, represented by a UML statechart, into the PROMELA language.

4.2.1. Encoding guidelines in PROMELA

The UML statechart into PROMELA translation issue has been tackled in the literature in a number of different ways [102–105]. We have based our translation approach on [102], where a proposal to verify UML state machines focusing on a UML subset for protocol models is presented. In particular, the authors describe a method to translate UML state machines (including a specific action language) to the PROMELA language. Although we consider that it is one of the most complete proposals, it does not support some state machine elements such as fork or join pseudostates, which we use in the representation of guidelines. We have therefore introduced several differences to their proposal related to the specific semantics of the statecharts which represent guidelines, and related also to our aim of verifying specific properties in the model. Next, we briefly explain the characteristics of the transformation approach given in [102] showing in each case the changes proposed.

PROMELA Program. In [102] each UML class is translated into a PROMELA process with an argument corresponding to the instance number of the created object. Our model only has a class related to the patient whose behavior is represented by the statechart. Thus, the PROMELA program consists of a process without arguments.

Constants and Variables. As in [102], we define a constant for each state and event signal in the statechart, and local variables encoding the state configuration of each region in the statechart. Nevertheless, since in each LTL formula the names or symbols must be defined using global variables from the model [48], we have declared all variables as being global. Additionally to the approach in [102], we define one variable for each UML action, and constants for event and action parameters in the statechart. We also declare other variables related to the translation of guards and choices in the statechart, the definition of which will be explained in the section devoted to *Data Abstraction*.

Process Structure. Based on UML semantics, [102] divides the main loop of the process encoding the state machine into two parts (identified by PROMELA label statements), `evalcompletions` and `evaltriggers`, to implement completion and signal-triggered transitions respectively. Also, the former part is subdivided into two blocks to control completion transitions (1) from pseudostates (*completion transition from pseudostates*) and (2) from simple and composite states (*completion transition from states*). For the latter block they also make a further distinction: (1) for the consumption of a signal event from the queue or non-deterministic generation of an external signal and (2) for evaluating whether a signal-triggered transition can be fired. In our case, we have adopted this block classification and the content characteristics proposed, but we have introduced some changes related to the translation of pseudostates and to the management of the events queue.

Following our approach, when the patient is in a specific state, only the possible events that can be dispatched from that state are controlled. Thus, we do not consider the management of events queue. As for the translation of pseudostates, we have defined rules for choice, join, fork, entry and exit pseudostates, which are the following:

- *Choice*. A variable `choice` is defined and for each choice an integer value is assigned, in such a way that the value of the `choice` variable will change depending on the corresponding choice. Also, for each guard in the outgoing transitions of the choice, a boolean variable is assigned (see *Data Abstraction* paragraph). Since this kind of pseudostate has outgoing completion transitions, its translation is allocated in the *completion transition from pseudostates* block. Firstly, when a choice is reached, the value of the `choice` variable is fixed to the one associated to that choice. Then, the control flow is transferred to the `asignValueVariables` label statement in order to non-deterministically assign to true the value of the guard of transitions. Later, the control flow is sent to a label statement associated to the choice, in which the guard whose value was assigned to true is controlled (by using `if..fi` statements). Finally, the control flow continues with the specific outgoing transition.
- *Join pseudostate*. Since join pseudostates have a state as the source UML element, their translation is allocated in the *completion transition from states* block. Each join is translated as a completion transition from a completion state but controlled in an `if..fi` statement, if all the simple source states of its incoming transitions are active.
- *Fork pseudostate*. In [102] the code for firing a transition sets the new active state to be the target state of the transition. Then, the translation of a fork pseudostate sets the new active state configuration to be the target state of each of its outgoing transitions.
- *Entry and Exit pseudostates*. For each entryPoint/exitPoint an `entry Point/exitPoint` label is defined. When the translation of the incoming/ outgoing transition is finished, the control flow is sent to the corresponding label in which the outgoing/ incoming transitions are controlled.

Data abstraction. In order to avoid the possible state explosion problem, we have used a data abstraction approach. In particular, we have used data abstraction for the assignment of values in choices. For each guard in the statechart we have defined a boolean variable in the PROMELA specification. In this way, we do not define a PROMELA variable for each variable in each statechart guard, reducing the number of possible states in the PROMELA program.

Currently, our approach for the translation of guidelines represented by statecharts into the PROMELA language does not support several UML statechart elements (such as history pseudostates or do activities). These are advanced modeling elements that could be later incorporated into our transformation proposal if needed.

4.2.2. Automatic translation

In order to manually transform a clinical guideline represented by a UML statechart into the PROMELA language, a professional with both UML and model checking skills may be required. Also, such an encoding process may entail a big effort depending on the guideline used. As we have described previously, we have used a MDA-based tool chain that allows us to customize the transformation strategy from the statechart to the PROMELA specification by defining a set of model to text transformations. So, as we show in Fig. 8 which is an extract of Fig. 1, starting from the manually created guideline specifications represented as a statechart, we use the transformations defined in a MDA tool to automatically generate the PROMELA specification of the model (see step number 2.2). In this way, for each guideline it is only necessary to manually design the statechart modeling it and, based on this model, the PROMELA specification is automatically generated. From this PROMELA specification and a medical property defined using our patterns and expressed by an LTL formula, we use SPIN to check the property in the guideline (see step number 2.3).

Choosing the suitable tool. Among the large amount of MDA-based tools in the literature, we are interested in those with support for customizable model to text transformations. The idea is to define, based on the specific semantics of every statechart which represents a guideline, only one set of transformations for all guidelines by means of which the corresponding PROMELA model is generated. Finally, we have chosen the MOFScript Eclipse plug-in [60,106]. MOFScript is an Eclipse plug-in developed within the MODELWARE Project [107] and is included in the Eclipse Genera-

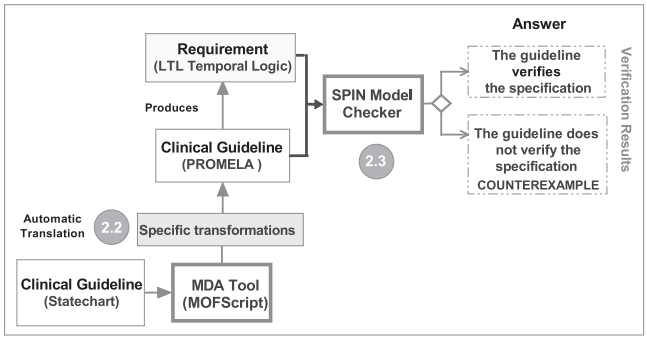

Fig. 8. Architecture of the verification approach.

tive Modeling Technologies (GMT) project [108]. The tool implements the MOFScript language which follows the OMG RFP process on MOF Model-to-Text Transformation [109]. The MOFScript subproject *aims at developing tools and frameworks for supporting model to text transformations* [60]. Among its features we can highlight that it provides control mechanisms (such as loops and conditional statements), collection types (such as list and hashtables), facilities for string manipulation, as well as the possibility of combining, in output files, clear text with expressions referencing model elements [60].

As input models, MOFScript can use any model which complies with the EMF [110] metamodel. From these input models, the tool can generate any arbitrary text (such as Java code or XML) by using a defined set of MOFScript transformations. Each MOFScript transformation contains transformation rules which are basically the same as functions, and which define the behavior of the transformation. The idea is that the transformation rules are defined based on the metamodel and subsequently compiled and executed on the model generating the corresponding text.

Using MOFScript. In our particular case, we use the UML 2.0 metamodel and the statechart which represents the guideline as the model. To create the statechart models, we can use any UML 2.0 compliant tool that can create models in the XMI format supported by EMF (for example, the UML2 Eclipse plug-in [76] or Borland Together Modeling tool [67]). As far as the PROMELA program generation is concerned, we have defined several MOFScript transformation scripts that generate the different PROMELA specification sections (the definition of constants and variables, evalcompletions and evaltriggers blocks, and the translation of pseudostates). We want to highlight that the defined transformation scripts are independent of the guideline used. They do not have to be modified to translate a different clinical guideline.

Regarding the statechart to PROMELA model transformation, we have defined several MOFScript transformation files employed to produce the print statements that generate the various PROMELA specification sections (the definition of constants and variables, evalcompletion and evaltrigger blocks, and the translation of pseudostates). The main transformation (`principal .m2t`) has the main rule that will generate the final PROMELA specification by using specific rules from the rest of the transformation files. We have defined another MOFScript file which is used as a library (`library.m2t`), that is, it contains commonly used rules that are required by other rules during the transformation process. The rest of the transformation files are devoted to the generation of the PROMELA code related to the creation of (1) *constants and variables* (`constantsAndVariables.m2t`), (2) *evalcompletion* (`evalCompletions.m2t`) and (3) *evaltrigger* (`evalTriggers.m2t`) blocks and (4) the implementation of pseudostates (`pseudostates.m2t`). The main rule in `principal.m2t` has calls to specific rules in these transformation files which produce the print statements that finally generate the PROMELA model. In particular, since the transformation `constantsAndVariables.m2t` generates the code corresponding to constants and variables in the PROMELA specification, its rules have to traverse the model looking for several statechart elements (such as states, events and actions), as we explained in section 4.2.1. Then, in order to avoid unnecessary traversals of the model and to be used by other rules during the transformation process, we have defined several collections (such as `hashtables` or `lists`) which collect specific information from the statechart during the execution of the rules in `constantsAndVariables.m2t`. In this way, the statechart is traversed only once, making the translation process faster.

Finally, as we have described in Section 2.4, the defined MOFScript transformations have been integrated in the GBDSSGenerator plug-in.

4.3. Example: verification of the IRC guideline

In this section we briefly present the results and experiences obtained from the verification of the IRC guideline. During the verification process we have not only found several anomalies related to the guideline definition, but also proved different kinds of properties we consider useful in everyday guideline applications. In particular, we have checked in the guideline several properties previously identified in our properties specification patterns which have allowed us to detect some inconsistencies in its definition.

Firstly, as described previously, the statechart model for the IRC guideline has been created using the Borland Together Modeling tool [67] obtaining a `.uml2` extension file. Secondly, we have used the GBDSSGenerator which, taking this file as input model, has automatically generated, by using a menu option the plug-in provides, the PROMELA program resulting a file with more than 900 lines. Thirdly, we have defined several properties to be verified in the guideline, among which we note the following. The first is related to the aspects or factors intrinsic to the guideline (*Guideline goal* source). The property states that *if an empirical treatment has not been ordered at some point of the guideline, later on the guideline does not prescribe the exclusion of the treatment*. This property is identified as (*Absence-After*) in our property specification patterns and a similar version can be found in [33,45]. In particular, based on Dwyer et al. patterns, the LTL formula which represents this property is the following:

$$\Diamond\neg(\text{removeTreatment}==\text{Empirical} \rightarrow \text{beginTreatment}==\text{Empirical}).$$

This property happens to be false and SPIN produces a counterexample which shows that the guideline has an inconsistency in its definition.

By means of the verification process, we can guess whether given a patient with certain clinical conditions there exists a guideline application which leads to a specific patient's state. This kind of property is defined in the *Patient specific clinical condition* scope, for example, starting from some patient's clinical test results, to find out whether with such results there exists a path which leads to a specific patient's state (*Patient isn't suspected to have IRC*). This property is identified as *Possible Existence* in our property specification patterns. In this case, the property to be verified is the following:

$$\Box((\text{MakiTest}==3 \;\&\&\; \text{HemoccultTest}==2)\rightarrow \Box\neg(\text{stateTop\_R}==\text{PatientIsntSuspectedToHaveIRC})).$$

We are checking whether with these test results, the expected state is never reached. This property happens to be false, and we get a counterexample which corresponds to a path in which having such test results the patient reaches the state.

To sum up, thanks to the verification process we have checked in the IRC guideline requirements with different aims, both to ascertain whether the guideline has errors or inconsistences in its definition and to obtain specific information useful for healthcare providers in their day-to-day clinical practice concerning the application of the guideline.

5. Discussion

There are several strengths that we want to highlight. Firstly, the only two manual steps that must be performed in the overall process correspond to (1) the modelization of the clinical guideline as a UML statechart, step guided by the defined representation patterns, and (2) the specification, guided by our property specification patterns, of the temporal-logic requirements to be checked in the guideline. Then, we use MDD techniques that enable us to automatically process manually created guideline specifications and temporal-logic statements to be checked and verified regarding these specifications, making the verification process faster and cost-effective. Secondly, we have defined our property specification patterns in a general fashion build upon Dwyer et al.'s proposal which has been increasingly used in the literature in a wide range of contexts. So, we consider that our patterns could be used to specify requirements in other contexts other than the clinical one. Finally, the overall verification and tool implementation framework has been implemented in the GBDSSGenerator Eclipse plugin by which, in particular, the PROMELA model is automatically generated by only selecting different menu options the plug-in provides. Thus, if the definition of a guideline that was previously verified with our proposal is changed, it will only be necessary to manually modify the statechart which represents the guideline (and therefore the properties defined from it), making it easy to carry out the verification process without having to modify the PROMELA model manually.

As shown in the previous section, the proposed verification approach has been satisfactorily applied to the particular case of the IRC guideline, finding as a result several semantic errors due to ambiguities and inconsistencies in its definition. Besides, we have checked different kinds of properties we consider useful in everyday guideline applications. Additionally, the proposed framework has been applied to several real-life guidelines used in different contexts within the medical care system. Among them, we would like to note its application to a laboratory guideline to carry out the aliquoting process. Other applications include several clinical guidelines published by the National Guideline Clearinghouse (NGC) [111] (for example, guidelines for the management of obesity in primary care and of rubella in pregnancy), obtaining encouraging results along the same line as with the IRC guideline. In particular, following the guideline categorization distinguished by this organization [111], these guidelines cover a wide range including management, diagnosis, treatment or prevention.

On the other hand, it is worth highlighting the importance of the collaboration between medical domain experts and knowledge engineers during both the authoring and verification processes, fact that has been already stressed by previous works in the medical context [112,113]. Particularly, we consider that such a cooperation has constituted a central asset of the representation and verification strategies.

While we have applied our verification approach to different guidelines obtaining encouraging results, we recognize there are certain limitations to the presented work. Here, we touch upon several of these issues, which provide a basis for the extension of this research. Firstly, as we have already mentioned, our approach for the translation of guidelines represented by statecharts into the PROMELA language does not currently support several UML statechart elements. Therefore, in the near future we plan to extend our approach in order to support such elements. Secondly, taking into account the guideline characterization of [111], as described previously, we have applied our approach to guidelines covering certain categories but its application to guidelines covering the other categories (such as counselling or rehabilitation) remains an ongoing task. Thirdly, there are several works which consider an upper level guideline ontology for authoring guidelines easing both the guideline specification, verification and tool support [47,113]. In particular, in [113] authors present a successfully evaluated methodology for collaborative guideline specification and verification which starts with an ontology-specific consensus established by knowledge engineers and expert physicians. In [47] on the other hand, the authors propose a knowledge-based methodology for the identification of anomalies in guidelines by using a knowledge-base component. Based on the temporal nature of clinical guidelines recommending actions and following their results over time, other works propose to use a time oriented language for annotation of a clinical guidelines intentions (Asbru/Asgaard [114], PROforma [115]). In particular, in the Asbru language specific temporal patterns with time annotations have been defined to express conditions in plan state transitions, which proposal has been extended in the context of the PROforma project. As we have described previously, our approach needs to use a formally defined ontology of medical terms that contains all the different actions and events that appear in a clinical guideline. However, we consider that our approach is independent of the actual ontology and it can be used with different ontologies. So, and taking into account these works, we consider that the use of a higher level guideline ontology, with specific temporal components, for guideline statecharts' concepts and semantics could greatly assist in the detailed specification process. So, the extension of our approach including such ontology constitutes a possible line for future work. Finally, regarding previous existing guideline representation languages, and the complex and labor intensive process that entails the modelization of guidelines using most of such languages, we would like to remark the work proposed in [116,117]. In this work, the authors propose an approach to facilitate such a process presenting a new methodology based on information extraction (IE) techniques for semi-automatic information extraction of clinical guidelines. Following their approach, the information extracted can be used in further transformations to finally generate a representation in any guideline representation language. So, the proposal is irrespective to the final guideline representation format. That is why, we consider the application of such a methodology as a complement to our representation format an interesting line of further work.

6. Conclusions

In this paper we present an approach and workflow aimed at improving the authoring and verification of clinical guidelines. The most significant contributions of this research paper are the following. Firstly, we have used UML statecharts as a method to model clinical guidelines which provides some advantages over other informal ways of representation used nowadays, especially when considering text generation. Secondly, we have established a pattern-based approach for defining commonly occurring types of requirements in guidelines in order to help non experts in their formal specification. Thirdly, we have developed a framework based on MDA techniques to verify specific requirements (defined using our property specification patterns) in guidelines in order to be checked against semantic errors and inconsistencies in their definition. Particularly, we have presented a statechart to PROMELA transformation proposal and defined a MDA approach for auto-

matically generating the PROMELA program required by the SPIN model checker to carry out the verification process.

Additionally, the proposed framework is part of a larger research project aimed at improving the authoring, quality and application of clinical guidelines in daily clinical practice. The overall framework has been implemented in the GBDSSGenerator plug-in for the Eclipse platform which allows us both to carry out the verification process and to generate the GBDSS of a guideline in an automatic fashion starting from its manually created statechart model.

References

[1] Institute of Medicine. Guidelines for Clinical Practice: from Development to Use. Washington, DC: National Academy Press; 1992.

[2] Papadopoulos C. The development of Canadian clinical practice guidelines: a literature review and synthesis of findings. JCCA J. Can. Chiropr. Assoc. 2003;47(1):39–57.

[3] OMG, UML 2.0 Superstructure Specification (August 2005), document formal/05-07-04. Available at http://www.omg.org/. Last visited: September 2009.

[4] Selic B. The pragmatics of model-driven development. IEEE Softw. 2003;20(5):19–25.

[5] B. Pérez, I. Porres, Verification of clinical guidelines by model checking, in: Proceedings of the 21th IEEE International Symposium on Computer-Based Medical Systems (CBMS 2008), 2008, pp. 114–119.

[6] I. Porres, E. Domínguez, B. Pérez, A. Rodríguez, M. Zapata, Development of an ubiquitous decision support system for clinical guidelines using MDA, in: Proceedings of the CAiSE'07 Forum, Trondheim, Norway, 2007.

[7] I. Porres, E. Domínguez, B. Pérez, A. Rodríguez, M. Zapata, A model driven approach to automate the implementation of clinical guidelines in decision support systems, in: Proceedings of the 15th Annual IEEE International Conference and Workshop on Engineering of Computer Based Systems (ECBS 2008), 2008, pp. 210–218.

[8] Agency for Healthcare Research and Quality, National guideline clearinghouse, guidelines for the prevention of intravascular catheter-related infections, available at http://www.guideline.gov. Last visited: Sept. 2009.

[9] SEIMC Sociedad Espa nola de Enfermedades Infecciosas y Microbiología Clínica [On-line], Documentos científicos, Available at http://www.seimc.org/documentos/. Last visited: Sept. 2009.

[10] Wang D, Peleg M, Tu S, Boxwala A, Ogunyemi O, Zeng Q, et al. Design and implementation of the GLIF3 guideline execution engine. J. Biomed. Inform. 2004;37(5):305–18.

[11] Ohno-Machado L, Gennari J, Murphy S, et al. The guideline interchange format: a model for representing guidelines. J. Am. Med. Inform. Assoc. 1998;5(4):357–72.

[12] M. Balser, C. Duelli, W. Reif, Formal semantics of Asbru – an overview, in: B.K.H Ehrig, e. A Ertas (Eds.), Proc. of the 6th Biennial World Conference on Integrated Design and Process Technology (IDPT-02), Society for Design and Process Science, USA, 2002, pp. 1–8.

[13] Shahar Y, Young O, Shalom E, Galperin M, Mayaffit A, Moskovitch R, et al. A framework for a distributed, hybrid, multiple-ontology clinical-guideline library, and automated guideline-support tools. J. Biomed. Inform. 2004;37(5):325–44.

[14] Young O, Shahar Y, Liel Y, Lunenfeld E, Bar G, Shalom E, et al. Runtime application of Hybrid–Asbru clinical guidelines. J. Biomed. Inform. 2007;40(5):507–26.

[15] Tu S, Musen M. Modeling data and knowledge in the EON guideline architecture. Medinfo 2001;10(1):280–4.

[16] M. Humber, H. Butterworth, J. Fox, R. Thomson, Medical decision support via the Internet: PROforma and Solo, in: Proceedings of the Tenth World Congress on Health and Medical Informatics (Medinfo2001), Vol. 10, 2001, pp. 464–469.

[17] Sutton D, Fox J. The syntax and semantics of the PROforma guideline modelling language. J. Am. Med. Inform. Assoc. 2003;10(5):433–43.

[18] Shiffman R, Agrawal A, Deshpande A, Gershkovich P. An approach to guideline implementation with GEM. Medinfo 2001;10(Pt 1):271–5.

[19] Shiffman R, Michel G. Toward improved guideline quality: using the COGS statement with GEM. Medinfo 2004;11(Pt 1):159–63.

[20] Terenziani P, Montani S, Bottrighi A, Torchio M, Molino G, Correndo G. A context-adaptable approach to clinical guidelines. Medinfo 2004;11(1):169–73.

[21] Terenziani P, Montani S, Bottrighi A, Torchio M, Molino G, Correndo G. The GLARE approach to clinical guidelines: main features. Stud. Health Technol. Inform. 2004;101:162–6.

[22] Peleg M, Tu S, Bury J, Ciccarese P, Fox J, et al. Comparing computer-interpretable guideline models: a case-study approach. J. Am. Med. Inform. Assoc. 2002;10(1):52–68.

[23] OpenClinical, Methods and tools for representing computerised clinical guidelines, http://www.openclinical.org/gmmsummaries.html. Last visited: September 2009.

[24] Wang D, Peleg M, Tu S, Boxwala A, Greenes R, Patel V, et al. process models and patient data in computer-interpretable clinical practice guidelines: a literature review of guideline representation models. Int. J. Med. Inform. 2002;68(1–3):59–70.

[25] ten Teije A, Marcos M, Balser M, van Croonenborg J, Duelli C, et al. Improving medical protocols by formal methods. Artif. Intell. Med. 2006;36(3):193–209.

[26] M. Moser, S. Miksch, Improving clinical guideline implementation through prototypical design patterns, in: S. Miksch, J. Hunter, E.K. (Eds.), Proceedings of the 10th Conference on Artificial Intelligence in Medicine (AIME 2005), Lecture Notes in Computer Science, vol. 3581, Springer, 2005, pp. 126–130.

[27] Domínguez E, Pérez B, Rodríguez A, Zapata M. Protocolos médicos para la toma de decisiones en un contexto de computación ubicua. Novática 2005;177:38–41.

[28] J. Whittle, J. Schumann, Statechart synthesis from scenarios: an air traffic control: case study, in: Workshop on Scenarios and State Machines at ICSE2002, 2002.

[29] Büssow R, Geisler R, Klar M. Specifying safety-critical embedded systems with statecharts and Z: a case study. In: Astesiano E, editor. Proc. Int. Conf. on Fundamental Approaches to Software Engineering (FASE), vol. 1382. Berlin, Lisbon, Portugal: Springer-Verlag; 1998. p. 71–87.

[30] Efroni S, Harel D, Cohen I. Toward rigorous comprehension of biological complexity: modeling, execution, and visualisation of thymic T-cell maturation. Genome Res. 2003;13(11):2485–97.

[31] N. Kam, I. Cohen, D. Harel, The immune system as a reactive system: modeling T cell activation with statecharts, in: Proc. Visual Languages and Formal Methods (VLFM'01), part of IEEE Symp. on Human-Centric Computing (HCC'01), 2001, pp. 15–22.

[32] Sobolev B, Harel D, Vasilakis C, Levy A. Using the statecharts paradigm for simulation of patient flow in surgical care. Health Care Manag. Sci. 2008;11(1):7986.

[33] L. Giordano, P. Terenziani, A. Bottrighi, S. Montani, L. Donzella, Model checking for clinical guidelines: an agent-based approach, in: AMIA Annu Symp Proc, 2006, pp. 289–293.

[34] N. Health, M.R. Council, A guide to the development, implementation and evaluation of clinical practice guidelines, in: Canberra: NHMRC, 1998, available at http://www.csp.nsw.gov.au/nhmrc/downloads/pdfs/NHMRCClinicalPractice.pdf. Last visited: September 2009.

[35] Watine J, Friedberg B, Nagy E, Onody R, Oosterhuis W, Bunting P, et al. Conflict between guideline methodologic quality and recommendation validity: a potential problem for practitioners. Clin. Chem. 2006;52(1):65–72.

[36] Balser M, Coltell O, van Croonenborg J, Duelli C, van Harmelen F, Jovell A, et al. Protocure: supporting the development of medical protocols through formal methods. In: Proceedings of the Workshop on Computerised Protocols and Guidelines (CGP-04). IOS Press; 2004.

[37] M. Balser, J. Schmitt, W. Reif, Verification of medical guidelines with KIV, in: Proceedings of Workshop on AI Techniques in Healthcare: Evidence-based Guidelines and Protocols (ECAI-2006), 2006.

[38] Hommersom A, Groot P, Lucas P, Balser M, Schmitt J. Verification of medical guidelines using background knowledge in task networks. IEEE Trans. Knowledge Data Eng. 2007;19(6):832–46.

[39] Lucas P. Quality checking of medical guidelines through logical abduction. In: Coenen F, Preece A, Mackintosh A, editors. Proceedings of the 23rd SGAI Int'l Conference Innovative Techniques and Applications of Artificial Intelligence (AI'03). Lecture Notes in Computer Science, vol. XX. Springer; 2003. p. 309–21.

[40] M. Marcos, M. Balser, A. Ten Teije, F. Van Harmelen, From informal knowledge to formal logic: a realistic case study in medical protocols, in: Proceedings of the 13th International Conference on Knowledge Engineering and Knowledge Management (EKAW-2002), Springer-Verlag, 2002, pp. 49–64.

[41] Marcos M, Balser M, ten Teije A, van Harmelen F, Duelli C. Experiences in the formalisation and verification of medical protocols. In: Proceedings of the Ninth European Conference on Artificial Intelligence in Medicine (AIME'03). LNAI, vol. 2780. Springer-Verlag; 2003. p. 132–41.

[42] Bäumler S, Balser M, Dunets A, Reif W, Schmitt J. Verification of medical guidelines by model checking – a case study. In: Model Checking Software, 13th International SPIN Workshop. Lecture Notes in Computer Science, vol. 3925. Vienna, Austria: Springer; 2006. p. 219–33.

[43] Groot P, Hommersom A, Lucas P, Serban R, ten Teije A, van Harmelen F. The role of model checking in critiquing based on clinical guidelines. In: Proceedings of the Eleventh European Conference on Artificial Intelligence in Medicine (AIME'07). Lecture Notes in Computer Science, vol. 4594. Amsterdam, The Netherlands: Springer; 2007. p. 411–20.

[44] A. Hommersom, P. Groot, P. Lucas, M. Marcos, B. Martinez-Salvador, A constraint-based approach to medical guidelines and protocols, in: ECAI 2006 Workshop – AI techniques in healthcare: evidence based guidelines and protocols, 2006, pp. 25–30.

[45] P. Terenziani, L. Anselma, A. Bottrighi, L. Giordano, S. Montani, Automatic checking of the correctness of clinical guidelines in glare, in: Proceedings of the 12th World Congress on Health (Medical) Informatics – Building Sustainable Health SystemsMedInfo (MEDINFO 2007), vol. 129, 2007, pp. 807–811.

[46] G. Duftschmid, Knowledge-based verification of clinical guidelines by detection of anomalies, PhD Dissertation, University of Vienna, 1999.

[47] Duftschmid G, Miksch S. Knowledge-based verification of clinical guidelines by detection of anomalies. Artif. Intell. Med. 2001;22(1):23–41.

[48] SPIN and PROMELA reference manual, Available at http://spinroot.com. Last visited: September 2009.
[49] Pnueli A. The temporal logic of programs. Proceedings of the 18th Annual Symposium on Foundations of Computer Science (FOCS), vol. 0. Los Alamitos, CA, USA: IEEE Computer Society; 1977. p. 46–57.
[50] Clarke E, Emerson E. Design and synthesis of synchronization skeletons using branching-time temporal logic. In: Logic of Programs. Lecture Notes in Computer Science, vol. 131. Springer; 1981. p. 52–71.
[51] Moszkowski B. A temporal logic for multilevel reasoning about hardware. Computer 1985;18(2):10–9.
[52] KIV Theorem Prover, available at http://www.informatik.uni-augsburg.de/lehrstuehle/swt/se/kiv/. Last visited: September 2009.
[53] T. Uribe, Combinations of model checking and theorem proving, in: Proceedings of the Third International Workshop on Frontiers of Combining Systems (FroCoS 2000), Lecture Notes in Computer Science, vol. 1794, Springer, Nancy, France, 2000, pp. 151–170.
[54] V. Pantelic, X. Jin, M. Lawford, D. Parnas, Inspection of concurrent systems: combining tables, theorem proving and model checking, in: H.R. Arabnia, H. Reza (Eds.), Software Engineering Research and Practice, CSREA Press, 2006, pp. 629–635.
[55] The Cadence SMV Model Checker, Available at http://www.kenmcmil.com/smv.html. Last visited: September 2009.
[56] R. Cobleigh, G. Avrunin, L. Clarke, User guidance for creating precise and accessible property specifications, in: SIGSOFT '06/FSE-14: Proceedings of the 14th ACM SIGSOFT International Symposium on Foundations of software engineering, ACM, New York, NY, USA, 2006, pp. 208–218.
[57] M. Dwyer, G. Avrunin, J. Corbett, Patterns in property specifications for finite-state verification, in: Proceedings of the International Conference on Software Engineering (ICSE '99), IEEE, 1999, pp. 411–420.
[58] Yu J, Manh T, Han J, Jin Y, Han Y, Wang J. Pattern based property specification and verification for service composition. In: Aberer K et al., editors. Proc. 7th Int. Conference on Web Information Systems Engineering (WISE 06). Lecture Notes in Computer Science, vol. 4255. Springer; 2006. p. 156–68.
[59] Clarke E, Jr O, Peled D. Model Checking. Masachusetts, Cambridge: The MIT Press; 2001.
[60] MOFScript Eclipse plug in, Available at http://www.eclipse.org/gmt/mofscript. Last visited: September 2009.
[61] E. Gamma, R. Helm, R. Johnson, J. Vlissides, Design patterns: elements of reusable object-oriented software, Addison Wesley, 1995.
[62] Hibernate, Relational Persistence for Java and .NET, http://www.hibernate.org/. Last visited: September 2009.
[63] E. Domínguez, B. Pérez, M. Zapata, Tracing the application of clinical guidelines, in: Proceedings of the International Workshop On Health Data Management, (IWHDM 2008), Lecture Notes in Computer Science, vol. 4977, Springer, 2008, pp. 122–133.
[64] E. Domínguez, B. Pérez, M. Zapata, Towards a traceable clinical guidelines application: a model driven approach, Methods Inform. Med. doi:10.3414/ME09-01-0038.
[65] ATL Eclipse Plug-in, Available at http://www.eclipse.org/m2m/atl/. Last visited: September 2009.
[66] OMG, OMG Model Driven Architecture (June 2003), document omg/2003-06-01. Available at http://www.omg.org/. Last visited: September 2009.
[67] Borland Together 2006 Release 2 for Eclipse, http://www.borland.com/together. Last visited: September 2009.
[68] E. Bouza, J.L. nares, A. Pascual, Diagnóstico microbiológico de las infecciones asociadas a catéteres intravasculares, in: R.C. e. E Cercenado (Ed.), Procedimientos en Microbiología Clínica, segunda edici=n Edition, Sociedad Espa nola de Enfermedades Infecciosas y Microbiología Clínica, 2004, Available at http://www.seimc.org/documentos/protocolos/microbiologia/. Last visited: September 2009.
[69] Leon C, Ariza J. Guías para el tratamiento de las infecciones relacionadas con catéteres intravasculares de corta permanencia en adultos: conferencia de consenso SEIMC-SEMICYUC. Enferm. Infecc. Microbiol. Clin. 2004;22(2):92–101.
[70] Society for Medical Decision Making Committee of Standardization of Clinical Algorithms. Proposal for Clinical Algorithm Standards. Med. Decis. Making 1992;12(2):149–54.
[71] Baresi L, Consorti F, Di Paola M, Gargiulo A, Pezzè M. LEMMA: a language for easy medical model analysis. J. Med. Syst. 1997;21(6):369–88.
[72] E. Domínguez, A. Rubio, M. Zapata, A way of dealing with behaviour of state machines, in: Proceedings of the UML 2000 Workshop: Dynamic Behaviour in UML Models: Semantic Questions, 2000, pp. 32–37.
[73] Enterprise Architect UML Case Tool, Available at http://www.sparxsystems.com/. Last visited: September 2009.
[74] Gentleware, Poseidon for uml case tool, Available at http://www.gentleware.com/. Last visited: September 2009.
[75] ArgoUML modeling tool, Available at http://argouml.tigris.org. Last visited: September 2009.
[76] EMF-based implementation of the Unified Modeling Language (UMLTM) 2.x OMG metamodel for the Eclipse platform, The Eclipse UML2 project website, http://www.eclipse.org/uml2. Last visited: September 2009.
[77] Eclipse Plugin Central (EPIC), Available at http://www.eclipseplugincentral.com/. Last visited: September 2009.
[78] Eclipse plugins, Available at http://www.eclipse-plugins.info/. Last visited: September 2009.
[79] modelbased.net, UML Tools, Available at http://www.modelbased.net/uml_tools.html. Last visited: September 2009.
[80] modelbased.net, MDA Tools, Available at http://www.modelbased.net/mda_tools.html. Last visited: September 2009.
[81] Domínguez E, Rubio A, Zapata M. Dynamic semantics of UML state machines: a metamodeling perspective. J. Database Manag. 2002;13(4):20–38.
[82] Jin Y, Esser R, Janneck J. A method for describing the syntax and semantics of UML statecharts. Softw. Syst. Model. 2004;3(2):150–63.
[83] Gnesi S, Latella D, Massink M. Model checking UML statechart diagrams using JACK. In: Proc. of the Fourth IEEE International Symposium on High Assurance Systems Engineering. IEEE; 1999.
[84] Lilius J, Porres I. vUML: a tool for verifying UML models. In: The 14th IEEE International Conference on Automated Software Engineering. Cocoa Beach, Florida: IEEE Computer Society; 1999.
[85] France RB, Ghosh S, Dinh-Trong T, Solberg A. Model-driven development using UML 2.0: promises and pitfalls. IEEE Comput. 2006;39(2):59–66. doi:10.1109/MC.2006.65.
[86] Hommersom A, Lucas P, Balser M. Meta-level verification of the quality of medical guidelines using interactive theorem proving. In: Proceedings of the 9th European Conference Logics in Artificial Intelligence (JELIA'04). Lecture Notes in Computer Science, vol. 3229. Springer; 2004. p. 654–66.
[87] M. Marcos, H. Roomans, A. ten Teije, F. van Harmelen, Improving medical protocols through formalisation: a case study, in: H. Ehrig, B. Kraemer, A. Ertas (Eds.), Proceedings of the Sixth World Conference on Integrate Design and Process Technology, 2002, pp. 59–64.
[88] M. Marcos, G. Berger, F. van Harmelen, A. ten Teije, H. Roomans, S. Miksch, Using critiquing for improving medical protocols: harder than it seems, in: Proceedings of the 8th Conference on Artificial Intelligence in Medicine (AIME 2001), 2001, pp. 431–441.
[89] M. van Gendt, A. ten Teije, R. Serban, F. van Harmelen, Formalising medical quality indicators to improve guidelines, in: S. Miksch, JHunter, E. Keravnou (Eds.), Proceedings of the 10th Conference on Artificial Intelligence in Medicine (AIME 2005), Lecture Notes in Computer Science, vol. 3581, Springer, 2005, pp. 201–210.
[90] K. Ryndina, Improving Requirements Engineering: An Enhanced Requirements Modelling and Analysis Method, Ph.D. thesis, Department of Computer Science, University of Cape Town, master's thesis, 2005.
[91] Ryndina K, Kritzinger P. Analysis of structured use case models through model checking. S. Afr. Comput. J. 2005;35:84–96.
[92] M. Dwyer, G. Avrunin, J. Corbett., Specification Patterns Website, http://patterns.projects.cis.ksu.edu/. Last visited: September 2009.
[93] O. Ryndina, P. Kritzinger, Improving requirements specification for communication services with formalised use case models, in: Southern African Telecommunication Networks and Applications Conference (SATNAC 2004), Spier Wine Estate, South Africa, 2004.
[94] Clarke E, Draghicescu I. Expressibility results for linear-time and branching-time logics. In: Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency, School/Workshop. London, UK: Springer-Verlag; 1988. p. 428–37.
[95] O. Kupferman, M. Vardi, Relating linear and branching model checking, in: Proceedings of the International Conference on Programming Concepts and Methods (PROCOMET'98), 1998, pp. 304–326.
[96] Laroussinie F, Markey N. Expressiveness of temporal logics. In: Introductory course, 18th European Summer School in Logic, Language and Information (ESSLLI'06). Spain: Malaga; 2006.
[97] M. Vardi, Sometimes and not never re-revisited: on branching versus linear time, in: Proceedings of the 9th International Conference on Concurrency Theory (CONCUR '98), Springer-Verlag, London, UK, 1998, pp. 1–17, invited Presentation.
[98] Vardi M. Branching vs. linear time: final showdown. In: Proceedings of the 7th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS). London, UK: Springer-Verlag; 2001. p. 1–22.
[99] Clarke E, Grumberg O, Hamaguchi K. Another look at LTL model checking. Form. Methods Syst. Des. 1997;10(1):47–71.
[100] Kong W, Seino T, Futatsugi K, Ogata K. A lightweight integration of theorem proving and model checking for system verification. In: Proceedings of the 12th Asia-Pacific Software Engineering Conference (APSEC). Taipei, Taiwan: IEEE Computer Society; 2005. p. 59–66.
[101] Kripke S. Semantic analysis of modal logic. Zeitschrift fur Mathematische Logik und Grundlagen der Mathematik 1963;9:67–96.
[102] T. Jussila, J. Dubrovin, T. Junttila, T. Latvala, I. Porres, Model checking dynamic and hierarchical UML state machines, in: MoDeV^{2}a: Model Development, Validation and Verification; 3rd International Workshop, Genova, Italy, October 2006, 2006, pp. 94–110.
[103] A. Knapp, S. Merz, Model checking and code generation for UML state machines and collaborations, in: G. Schellhorn, W. Reif (Eds.), FM-TOOLS 2002: 5th Workshop on Tools for System Design and Verification, Report 2002-11, Institut für Informatik, Universität Augsburg, Reisensburg, Germany, 2002.
[104] E. Mikk, Semantics and Verification of Statecharts, PhD Dissertation, University of Kiel, 2000.
[105] I. Porres, Modeling and Analyzing Software Behavior in UML, Ph.D. thesis, Åbo Akademi, Finland, 2001.
[106] J. Oldevik, MOFScript eclipse plug-in: metamodel-based code generation, in: Proceedings of the Eclipse Technology eXchange workshop (eTX) at the ECOOP 2006 Conference, Nantes, France, 2006.

[107] MODELWARE project., Modelling solution for software systems, Available at http://www.modelware-ist.org/. Last visited: September 2009.
[108] Eclipse Generative Modeling Tools (GMT), Available at http://www.eclipse.org/gmt. Last visited: September 2009.
[109] OMG, Mofscript Second Revised Submission to the MOF Model to Text Transformation RFP, OMG document ad/2005-11-03, Available at http://www.omg.org/. Last visited: September 2009.
[110] EMF, The Eclipse Modeling Framework website, http://www.eclipse.org/emf. Last visited: September 2009.
[111] Agency for Healthcare Research and Quality, National Guideline Clearinghouse, Available at http://www.guideline.gov. Last visited: September 2009.
[112] Peleg M, Patel V, Snow V, et al. Support for guideline development through error classification and constraint checking. Proc. AMIA Fall Symp. 2002:607–11.
[113] Shalom E, Shahar Y, Taieb-Maimon M, Bar G, Yarkoni A, et al. A quantitative assessment of a methodology for collaborative specification and evaluation of clinical guidelines. J. Biomed. Informat. 2008;41(6):889–903.
[114] 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(1-2):29–51.
[115] Fox J, Johns N, Rahmanzadeh A. Disseminating medical knowledge: the PROforma approach. Artif. Intell. Med. 1998;14:157–81.
[116] Kaiser K, Akkaya C, Miksch S. How can information extraction ease formalizing treatment processes in clinical practice guidelines?: a method and its evaluation. Artif. Intell. Med. 2007;39(2):151–63.
[117] Kaiser K, Miksch S. Versioning computer-interpretable guidelines: semi-automatic modeling of 'Living Guidelines' using an information extraction method. Artif. Intell. Med. 2009;46(1):55–66.