OWL-based Acquisition and Editing of Computer-Interpretable Guidelines with the CompGuide Editor

Tiago Oliveira[1], Filipe Gonçalves[2], Paulo Novais[2], Ken Satoh[1], and José Neves[2]

[1]National Institute of Informatics, Tokyo, Japan,
{*toliveira,ksatoh*}*@nii.ac.jp*
[2]Algoritmi Research Centre/Department of Informatics, University of Minho, Braga, Portugal,
fgoncalves@algoritmi.uminho.pt,{*pjon,jneves*}*@di.uminho.pt*

conflicts of interest: none

Abstract

Computer-Interpretable Guidelines (CIGs) are the dominant medium for the delivery of clinical decision support, given the evidence-based nature of their source material. Therefore, these machine-readable versions have the ability to improve practitioner performance and conformance to standards, with availability at the point and time of care. The formalisation of Clinical Practice Guideline (CPG) knowledge in a machine readable format is a crucial task to make it suitable for the integration in Clinical Decision Support Systems (CDSSs). However, the current tools for this purpose reveal shortcomings with respect to their ease of use and the support offered during CIG acquisition and editing. In this work we characterise the current landscape of CIG acquisition tools based on the properties of guideline visualisation, organisation, simplicity, automation, manipulation of knowledge elements, and guideline storage and dissemination. Additionally, we describe the CompGuide Editor, a tool for the acquisition of CIGs in the CompGuide model for CPGs that also allows the editing of previously encoded guidelines. The Editor guides the users throughout the process of guideline encoding and does not require proficiency in any programming language. The features of the CIG encoding process are revealed through a comparison with already established tools for CIG acquisition.

1 Introduction

Clinical Practice Guidelines (CPGs) are defined as systematically developed statements to assist health care professionals in specific circumstances (Lohr, Field, et al., 1990). They are associated with the reduction of variability in the application of the procedures and also the decrease of medical errors (Silberstein, 2005). The implementation of Clinical Practice Guidelines (CPGs) in Clinical Decision Support Systems (CDSSs) has the potential of improving the acceptance and application of evidence-based clinical recommendations in daily practice given that such systems are able to monitor the actions and observations of health care professionals and provide relevant advice at the point and time of care (de Clercq, Blom, Korsten, & Hasman, 2004; Peleg, 2013). The deployment of these CIG-based CDSSs is not trivial as they have to provide a high level of interactivity with the user, in order to allow the latter to feed the CIG algorithm all the information necessary to produce tailored recommendations. This implies handling multiple data entry points and managing aspects of CIG execution such as the relative order of procedures and temporal constraints, all the while providing interfaces that can convey these recommendations in a clear and objective way (Isern & Moreno, 2008). This operationalization of CIGs has been addressed in previous works which present a system for CIG execution that aims to provide a greater integration of CPG recommendations in the daily practice of health care professionals by mapping the interpreted clinical recommendations, with their respective constraints, onto an agenda of activities (Oliveira, Silva, Neves, & Novais, 2016).

Yet, to feed the execution algorithms embedded in these systems one has to develop methods for the assisted encoding of structured CIGs, in order to assist the transformation of CPGs in their paper versions into executable procedures. The difficulty in this is devising a CIG acquisition work flow that enables the management of complex and intricate instructions regarding CPG parameters. CPGs were not originally conceived to be machine-interpretable and, as such, their text versions are challenging to transpose into a structured model (Martínez-Salvador & Marcos, 2016). This led to the development of different Computer-Interpretable Guideline (CIG) models and tools by different research groups, covering a wide range of clinical situations (Isern & Moreno, 2008; Peleg, 2013). A model (here used as a synonym for language) aims to provide a structure for the correct formalisation of a narrative CPG as a CIG that would be the basis of a CDSS, but by itself it is not sufficient. It is still necessary to guide CIG design, according to a selected model, to ensure a correct syntax and disposition of knowledge elements. Such tools include the Protégé Desktop, the SAGE Workbench, Tallis, GEM Cutter, Asbru View, among others (Isern & Moreno, 2008). Despite providing assistance in the construction of CIGs, these tools have not progressed beyond the level of academic research projects. Additionally, they present limitations in the guidance offered to the user during the CIG acquisition process, the intelligibility of their interface, the way their functionalities are conveyed, the visualisation of guideline knowledge elements and the reuse of this knowledge.

The present work is an extension of the one disclosed in (Gonçalves, Oliveira, Neves, & Novais, 2017). It features a CIG editing tool, the CompGuide Editor, for the CompGuide ontology, the underlying CIG model (Oliveira et al., 2016), based on Web Ontology Language (OWL). The contributions of the work are as follows: a tool for the formalisation of CIGs in a structured format; a step-by-step work flow for encoding clinical knowledge; and a set of mechanisms that enable the reuse of guideline knowledge during CIG encoding. The CompGuide Editor also addresses the limitations identified in existing tools, namely in the set of automatic features made available in order to facilitate CIG acquisition. At the same time, the set of properties listed herein may be further used to assess CIG tools in future works.

The present paper is organised as follows. Section 2 provides related work on CIG acquisition and editing tools. Their most prominent aspects and main limitations are exposed in this section. This assessment is based on a set of properties for CIG creation defined in order to compare existing tools with the one described herein. In Section 3, the CompGuide ontology for CIGs is described by covering the definition of work flows of procedures, clinical constraints, and temporal patterns. Section 4 explains the steps taken in the creation of the CompGuide Editor tool, its main features, and also compares it mainly with the SAGE Workbench, an existing CIG acquisition tool. Finally, section 5 presents conclusions about the work developed so far and future work considerations.

2 Computer-Interpretable Guideline Creation and Editing

As already mentioned, there are several tools that support the acquisition and editing, either manually or semi-automatically, of CIG knowledge elements, taxonomies, and other organisation levels of these machine-readable formats. The following sections describe properties for the assessment of CIG creation and editing and also selected tools in light of these properties.

2.1 Properties for Tool Assessment

This section is dedicated to defining properties for the comparison of CIG acquisition and editing tools. The comparative properties were selected as a means to analyse and evaluate CIG platforms, based on user experience (Bott, 2014). They consist of the following:

- **Guideline Visualisation**: a graphical representation (tree, node-link, network diagrams) of parts or a full CIG work flow. The arrangement of the representations within a diagram helps the user to understand the work flow, identify relevant points of the guideline, and manipulate knowledge elements;
- **Organisation**: this property is related with how easy the tool is to understand, determined by its structure and the way in which its functionalities

are made available and whether they are placed correctly;

- **Simplicity**: this property conveys the ease of access to the functionalities of the tool. Complexity may lead to confusion in the use of the tool, leading users to abandon it;
- **Automation**: when creating or editing new instances, the user should only implement the most relevant knowledge elements, with the rest being automatically completed by the platform;
- **Manipulation of Knowledge Elements**: the ability to arrange instances in the guideline view and filter the workflow of the CIG with the help of graphical type links;
- **Web/Local Repository**: the possibility to save or load CIGs either locally or onto a cloud repository;

Although organisation and simplicity are related, they convey different meanings. It is possible for an application to provide a wide variety of well organised features, placed in a logical and coherent way in the interface, but the access to be far from simple, requiring the user to navigate through numerous menus until he reaches the desired functionality.

2.2 Assessment of Existing Tools

In terms of related work, the tools selected for analysis are linked to the most prominent CIG languages. All of the underlying languages follow a Task Network Model (TNM), in which every recommendation is viewed as a clinical task.

The first tool is the Protégé Desktop, an open source ontology development and knowledge acquisition environment developed by Stanford Medical Informatics (Noy et al., 2003). It is a graphical Java tool, which provides an extensible architecture for the creation of customised knowledge-based tools and assist users in the construction of large electronic knowledge bases. It provides a platform which can be extended with graphical widgets for tables, diagrams, and animation components. The tool is used to author guidelines in various models, including the Guideline Interchange Format (GLIF) (Peleg et al., 2000) and PRODIGY (Purves, Sugden, Booth, & Sowerby, 1999).

The SAGE Workbench is a complete, self-contained environment that uses the Shareable Active Guideline Environment (SAGE) model (Beard et al., 2002). This model encodes guideline knowledge needed to provide situation-specific decision support and uses standardised components for interoperability. The SAGE Workbench provides a knowledge authoring tool based on Protégé. The user interface for the SAGE Workbench is organised as a number of tabs for: the navigation of frames that are directly and indirectly referenced from a selected instance in a tree structure, expression of integrity constraints about a knowledge base in Protégé axiom language, search of terms in a medical terminology service, and so forth.

The Tallis tool is a relatively recent (when compared to the other tools) Java implementation of PROforma-based authoring and execution developed by the Cancer Research UK (Sutton & Fox, 2003). Tallis is based on a later version of the PROforma model (Steele & Primer, 2002). It is, in fact, a suite of tools in which the main component is a desktop graphical editor for authoring guidelines. This editor also includes a test application for simulating interactions with the encoded CIGs. Workflows are displayed in Tallis both in a network view and in a tree view, by taking advantage of the hierarchical structure of plans in the respective CIG language.

GEM Cutter is a simple XML editor that facilitates the markup of CPG texts, and therefore supports the conversion of a guideline document into the GEM format and publication in a cross-platform manner (Shiffman, Agrawal, Deshpande, & Gershkovich, 2001). The main window of the tool comprises three panels, a menu bar, and a button bar. The CPG text is loaded onto the leftmost panel. The middle panel contains an expandable tree view of the GEM hierarchy. The rightmost panel shows information regarding the knowledge element selected at the time. GEM is intended to be used throughout the entire guideline life-cycle to model information pertaining to guideline development, dissemination, implementation, and maintenance. Information at both high and low levels of abstraction can be accommodated

Finally, Asbru View is a graphical user interface for viewing, creating and modifying Asbru plans. It is based on different views of different aspects of the plans (Votruba, 2003). The topological view displays mainly the relationships between plans, without a precise time scale. The temporal view is focused on the temporal dimension of plans and conditions. In addition to the topological information, physicians need to be able to see the details of the temporal extensions of plans. It consists of a display that represents each plan with a graphical object whose features change with the values they depict.

Table 1 shows a comparison between these tools, by using the comparative properties explained earlier. Important to say that this comparison does not include plug-ins that can be applied to these platforms in order to add new features. We can conclude that some properties are absent from the existing tools and should be developed in the CompGuide Editor tool to improve the user experience. Although all the tools possess the basic requirement of allowing the instantiation and editing of CIG elements, they are mainly focused on the proper functioning of the tool and less on the appearance or ease of management.

Most tools (the exception is GEM Cutter) present some form of graphical view of the workflow. Whether it is through a network structure or a tree structure, it is possible to have a general view of the instances that build a CIG and then focus on particular elements. Although the analysed tools are clearly well organised, they lack simplicity. Taking Protégé Desktop and SAGE Workbench as examples, despite their having many useful features available, the amount of menus they display is significant, which makes the user lose considerable time trying to understand the different functionalities. As such, simplicity is a property that should be prioritised. As for the manipulation of knowledge, namely the ability to dynamically move and arrange knowledge

elements, it is absent from existing tools, but an important element in modern applications, enabling the management of a CIG visual layout. One of the features that new tools should present is the ability to automatically fill in less significant data, allowing the user to focus on the information relevant to the instance he is currently managing, leaving less important details to be handled by the system. This property includes the automatic creation and filling of mandatory data fields for an instance, such as the date of creation of a CPG or its version, the relationships expressing the connection of a clinical task to a subsequent task, and so forth. This kind of automation is not present in none of the analysed CIG tools. Another important property is the ability to import or export CIGs stored locally or in a cloud. Only Protégé Desktop and SAGE Workbench show this property.

Table 1: Comparison of tools for the acquisition and editing of CIGs. The "✓" signifies that the tool possesses the property.

Feature/Platform	*Protégé Desktop*	*SAGE Workbench*	*Tallis*	*GEM Cutter*	*Asbru View*
Guideline Visualisation	✓	✓	✓		✓
Organisation	✓	✓	✓	✓	✓
Simplicity			✓	✓	✓
Automation					
Manipulation of Knowledge Elements					
Local Repository	✓	✓	✓	✓	✓
Web Repository	✓	✓			

3 Ontology for Computer-Interpretable Guidelines

The CIG model used in this work is the CompGuide ontology (Oliveira, Novais, & Neves, 2013). It provides representation primitives for clinical recommendations based on Web Ontology Language (OWL) by following TNM, in which each recommendation assumes the form of a task. In this sense, a CPG is an instance of the class *ClinicalPracticeGuideline* in the ontology. In order to reflect the TNM, a set of key OWL classes were defined as subclasses of *ClinicalTask*. They include the following:

- *Action*: a task that should be performed by a health care professional such as an observation, procedure, exam, or treatment application;
- *Question*: a task to get information about the clinical parameters that build the state of the patient;

- *Decision*: a task that encodes a decision regarding the state of a patient, featuring various options and respective conditions;

- *Plan*: a composed task containing instances of the other tasks defined to achieve a specific goal. A *Plan* is the top clinical task and a *ClinicalPracticeGuideline* always consists of a *Plan* within which other tasks are defined.

In CompGuide there are object properties that connect instances of the classes as mentioned above in order to define the relative order between tasks. In this regard, it is possible to define: sequential tasks, in which a task should directly follow another (with the *nextTask* property); parallel tasks which should be executed simultaneously (with the *parallelTask* property); and alternative tasks from which one is automatically selected for execution (with the *alternativeTask* property). In this sense, a guideline in CompGuide resembles a linked list of recommendations. Additionally, it is possible to define different types of conditions that constrain task execution. They are specified with the *Condition* class and include: trigger conditions to select one amongst alternative tasks, pre-conditions which must be verified before executing a task, conditions for options in *Decision* tasks, and expected outcomes for clinical tasks. The *Condition* has specific properties for clinical parameters and their values.

The classes that enable the representation of temporal restrictions are all subclasses of *TemporalElement* (Oliveira et al., 2016). The relationship between these temporal classes and the classes in *ClinicalTask* are shown in Figure 1, along with the properties used to connect them. One of the subclasses of *TemporalElement* is *TemporalUnit* which represents the different units in which a temporal constraint may be expressed. It is an enumerated class consisting of the instances *second*, *minute*, *hour*, *day*, *week*, *month*, and *year*. The main classes that enable the definition of temporal restrictions about the execution of tasks are:

- *Duration*: definition of how long *Actions* and *Plans* should last;

- *WaitingTime*: definition of a delay in the start of a clinical task;

- *Periodicity*: definition of the frequency of a clinical task. This temporal pattern can be defined for any type of task. An instance of *Periodicity* can also be connected to an instance of Duration through the *hasDuration* object property, thus determining for how long a periodic task should take place;

- *CyclePartDefinition*: definition of a nested temporal pattern. Within it, is possible to define a new duration for each instance of the periodic task or a new periodicity through the *CyclePartPeriodicity* class;

- *CyclePartPeriodicity*: a nested temporal pattern for the definition of a periodicity within a periodicity.

In regards to *Periodicity*, if one wants to state the number of times the clinical task should take place, i.e. the number of cycles of the periodic task), it is necessary to formulate a repetition constraint, which is possible with the *repetitionValue* data property, with a range of integer numerical values. It could also be the case that periodic task should only occur until a condition about the state of a patient is verified. To express this, one uses the *hasStopCondition* object property to connect an instance of *Periodicity* to instances of the class *Condition*.

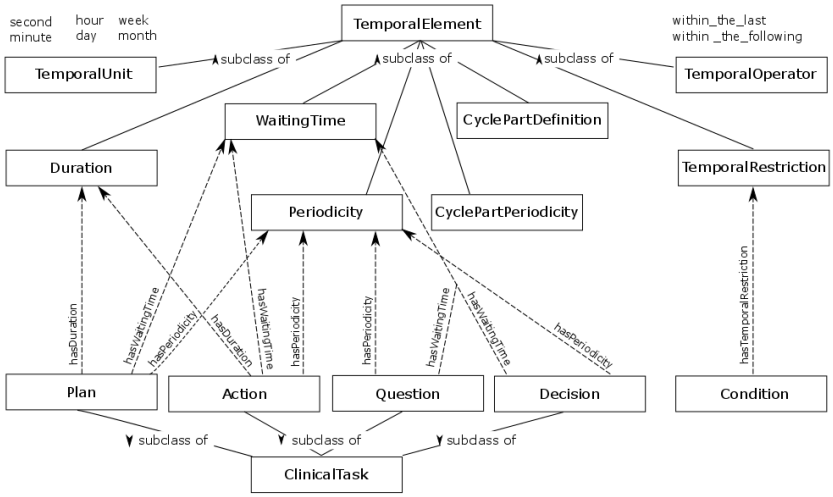

Figure 1: Representation of the CompGuide ontology with clinical tasks and respective temporal elements.

Temporal reasoning about the state of a patient is enabled by the *Temporal-Restriction* class, whose instances can be associated with a *Condition* through the *hasTemporalRestriction* property. With the *hasTemporalOperator* property a *TemporalOperator* is specified for the restriction. *TemporalOperator* consists of two instances, *within_the_last* and *within_the_following*. The operator *within_the_last* is used when one aims to express that a condition about the patient state must have held true at least once, within a period of time just before execution time. It is used in trigger conditions, pre-conditions and conditions of rules in *Decision* instances. This operator is interpreted by checking if, in the state of the patient, there is a record regarding the parameter in the condition, registered within the specified time frame, whose value validates the condition. As for the *within_the_following* operator, it expresses a condition about the future, in which one aims to observe the effect a clinical task has after being applied to a patient. Such conditions are used in task outcomes. Within the context of a CPG for the diagnosis and treatment of colon cancer, an exam-

ple of a temporal restriction would be an *Action* that advised chemotherapy with an outcome stating that the tumour should become operable within six months. In this case, there is a condition with a temporal restriction featuring a *within_the_following* operator.

A distinctive feature of CompGuide, when compared to other CIG models, is that it does not require any proficiency in a programming language in order to define constraints. Instead, all is done with the basic elements of an ontology: classes, instances, and properties. It was showed in (Oliveira, Novais, & Neves, 2015) that this CIG ontology was able to provide sufficiently expressive constructors for CPGs from different medical specialities and categories.

The work flow of tasks, their clinical constraints, and their temporal constraints demand a formalisation tool that is capable of providing instructions for the definition of the intricate and complex patterns of CPG recommendations.

4 CompGuide Editor for the Management of Guidelines

The CompGuide Editor was developed as a Protégé Desktop plug-in, given the need to create software capable of implementing all the features offered by this application, more specifically the functionality of managing the data of an ontology through the use of a graphical interface, along with the development of new features capable of solving the limitations in existing projects. Another advantage that came from using Protégé was the ability to implement extra features from other plug-ins in a simple and intuitive way. Since OWL is one of the ontology languages supported by Protégé and, at the same time, the underlying language of the CompGuide ontology, using this application as the basis for the editor was the logical decision to make.

4.1 Methods for the Development of the Plug-in

The development process of the CompGuide Editor for the management of CIGs was designed based on the main goal of drastically reducing the learning curves for the creation of ontology-enabled representations and the time required to become acquainted with its concepts. Other aims, such as the creation of an open source software based solution supported by a big community platform and the capacity to import extra features from different tools, were also taken into account in this project.

Currently, the most used versions of Protégé vary between versions 3, 4 and 5, being 3 the oldest and 5 the newest. Since version 3 is the oldest, it is also the version with the greater number of plug-ins created for the application, which are incompatible with versions 4 and 5. As such, the selected version of the Protégé Desktop application was version 4, since the CompGuide ontology was developed using this release. Protégé Desktop 3 uses OWL API 1 while Protégé Desktop 4 uses OWL API 2. The development of the plug-in required the use

of various OWL and Protégé extension APIs, such as the the Jena API (provides services of parsing, database persistence, and querying), the Protégé/OWL API (ontological base for creating, manipulating and serialising OWL ontologies) and the Protégé OWL API (employs graphical user interfaces and delivers classes/methods to manage OWL files). Figure 2 displays the Protégé application system and summarises the required APIs. Although there are other APIs used in the Protégé Desktop application, the list presented shows only the required APIs used in the creation of the functionalities of the CompGuide Editor.

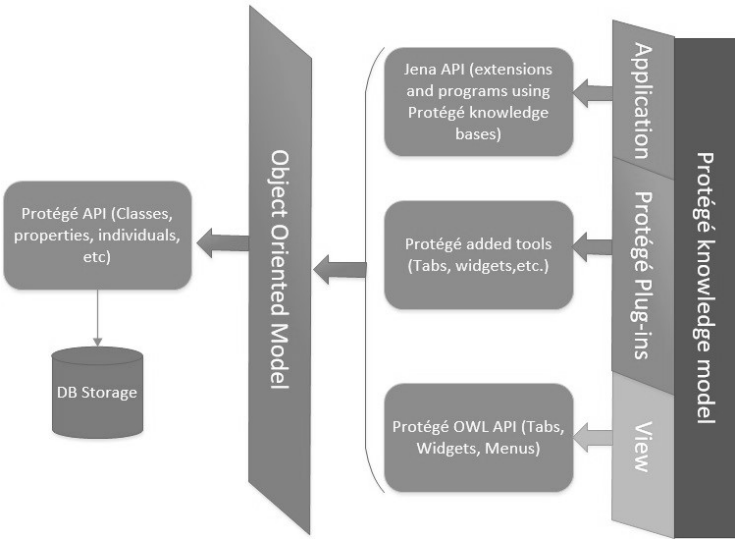

Figure 2: Protégé Ontology Editor and Knowledge Acquisition System.

The support for interactively navigating the relationships in OWL was provided by the OntoGraf plugin. It supports various layouts for automatically organising the structure of the ontology. Different relationships are supported: subclass, individual, domain/range of object properties, and equivalence. OntoGraf makes it possible to filter relationships and node types to help create the desired view (da Silva & Freitas, 2011).

Figure 3 shows a UML class diagram of two important classes in the domain: *ClinicalPracticeGuideline* and *Plan*. It provides an overview of the representation of a CPG within the system, with additional elements that complement the description of the ontology in Section 3. *ClinicalPracticeGuideline* is the starter class of a CPG and it is associated with a *Scope* and a *Plan*. The *Scope* manages information about the purpose and range of the CPG such as the clinical specialty, the guideline category or conditions for the application of the CPG. Also, *ClinicalPracticeGuideline* has a set of attributes that convey administrative information, namely authorship, date of creation, date of last update,

version number, and so forth. The *Plan* class is a subclass of *ClinicalTask* and manages the set of clinical tasks to be executed in the CPG. At least one task must be implemented in the *hasFirstTask* relationship and it is mandatory for a *ClinicalPracticeGuideline* to have a *Plan*. The figure also shows the *Duration* and the *Periodicity* classes, with their respective attributes that allow the definition of maximum, minimum and average values for these temporal patterns, as well as temporal units. There is also the possibility to implement a stop condition, related with a clinical parameter, to the task within the *Periodicity* class, under which its execution is immediately halted. The purpose of the *Condition* class is to define the constraints about the state of the patient, whether they are expressed through numerical or qualitative values, for different clinical situations. *TriggerConditionSet*, *PreConditionSet*, and *OutcomeConditionSet* represent the different possible situations, with a direct correspondence to the description in Section 3. The *Condition* can be further enhanced with a *TemporalRestriction* in order to express a temporal horizon over which the condition should be verified. The classes depicted in Figure 3 and the remaining classes in the system correspond to a direct and complete mapping of the classes and properties defined in the CompGuide ontology.

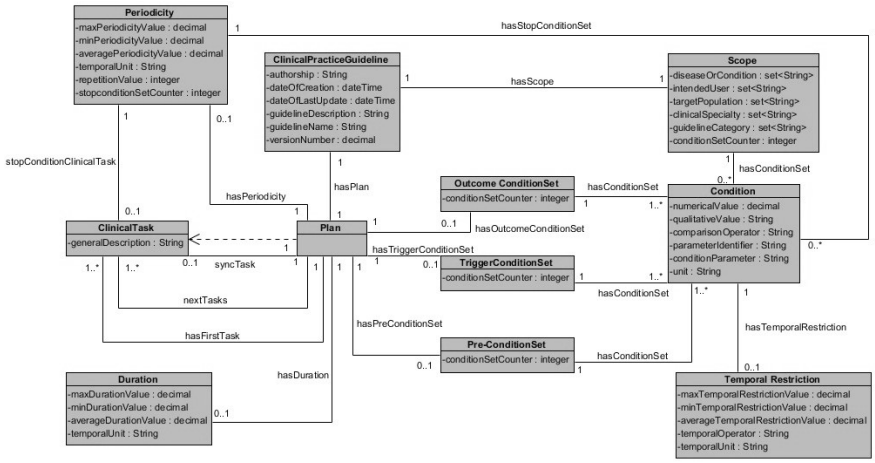

Figure 3: Class diagram of *ClinicalPracticeGuideline* and *Plan* and respective dependency relationships.

4.2 Main Features

The most significant contribution of the CompGuide Editor is the employment of software assistants. Based on a set of rules, these assistants guide the user step-by-step in the task they want to perform, through the use of wizards. Furthermore, the software assistants automatically handle the skeleton of the managed instances (according to the CompGuide ontological data structure),

link the data of the managed instances, ensure the correctness of data types, automatically implements less relevant data (dates, version handling, etc.) and carry out a smart management of digital resources. These wizards guarantee the intended straightforwardness as they ensure that the users have an easy access to the functionalities demanded by the task to be carried out. Other existing Protégé-based tools, namely Protégé Desktop and the SAGE Workbench, possess complex functionalities but do not provide simple access to them, which is an improvement with regards to existing tools.

In the CompGuide Editor, there are three different wizards, i.e. the *Creation Wizard* (dedicated to adding new clinical knowledge to the CompGuide ontology), the *Edit Wizard* (used to update the existing clinical guidelines) and the *Delete Wizard* (to either delete full clinical guidelines or erase parts of existing guidelines). Furthermore, these wizards share an interface similarity, offering an organised, simple, and matching design, providing the means for a fast-learning and superior user experience. Figure 4 shows images of the editor regarding the automated management process of a CIG, through the use of the *Creation Wizard*. In this case, the editor is guiding the guideline encoder through the process of creating a new CPG instance and filling in administrative and scope information.

Focusing on the main interface of the CompGuide Editor, Figure 5 shows its simplicity. The tool is a single tab in Protégé, consisting of two main views: CGuide Wizard Options and OntoGraf. The CGuide Wizard Options view (bottom view) has the set of features to manage the CIG plus the options to download/upload the CompGuide ontology file, while the OntoGraf view provides a 2D dynamic graphical representation of the ontology. In addition to the graphical view of the CIG, it is possible to see CIG elements in a list, organised by the class to which they belong, through the Individuals by Type view. The number shown in front of the the name of the class represents the number of individuals that exist in that OWL class. When accessing the CIG management features, a new window will appear, in which a uncomplicated step-by-step procedure will start. In order to complete this process, the user must follow the instructions of the corresponding wizard and insert all the required data. In the end, the CompGuide Editor tool will insert all changes in the ontology, where the wizards process all the data structures and automatically fill in the names of instances, dates, versions and establishes the connections between instances. An example of this is that a CPG instance is always associated with an instance of *Scope*, which specifies the range of the CPG. When the CPG instance is created, the skeleton for the *Scope* instance is immediately and automatically generated. In the case of CPG instance deletion, both CPG instances and corresponding *Scope* will be eliminated.

As for the OntoGraf view, its aim is to promote a better understanding of CompGuide ontology concepts and support for interactively navigate the relationships of the CompGuide knowledge by manipulating the graphical representation of a CPG, through the selective expansion of nodes and drag-and-drop features. Additionally, various layouts are supported for automatically organising the graphical structure of the ontology, to filter data links and node types to

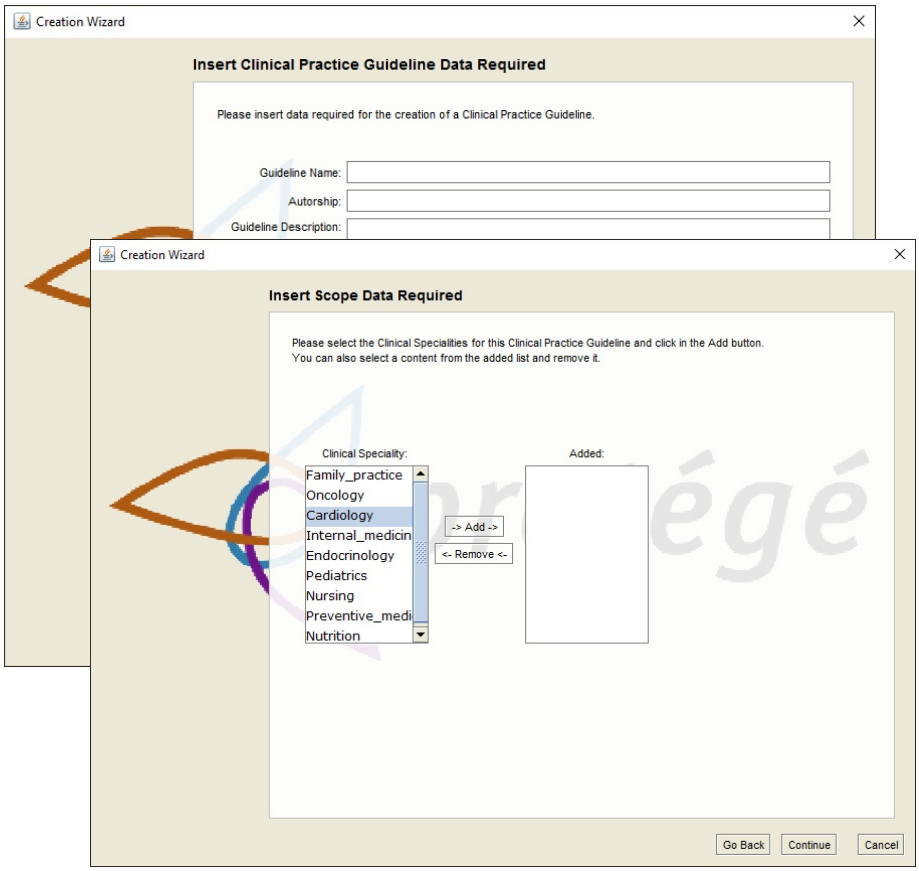

Figure 4: Management of the creation of a CPG with the Creation Wizard.

help create the desired view, and to glance at a node information by placing the mouse on top of it. A small example pertaining to a CPG for the management of coronary heart disease is shown in Figure 6, using this plug-in's view. More details can be explored on the OntoGraf's website [1]. Other plug-ins can be added into the CompGuide Editor Tab as a way of enriching the content and functionalities available for the user, in a simple and easy way.

The interaction of users with this tool is represented in the architecture of Figure 7. In it, the main users of the system are health professionals acting as CIG encoders.

They are responsible for creating, modifying or deleting clinical steps or aspects in a CIG file, with the help of the software assistants. These users can also download the latest available version of the CompGuide ontology, or share their modified CIG file to the CompGuide development team, wich, in turn, validates it. If the file is considered to be valid, it is stored in the Git repository

[1] https://protegewiki.stanford.edu/wiki/OntoGraf

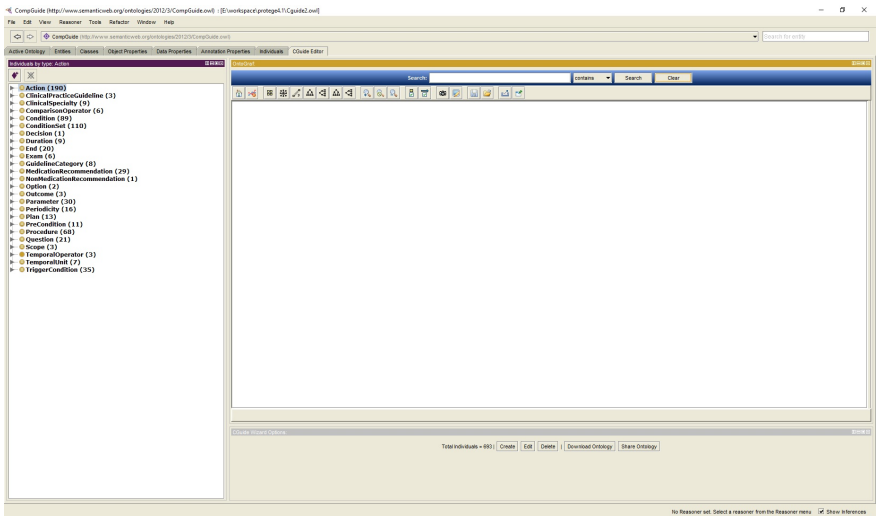

Figure 5: CompGuide Editor main interface.

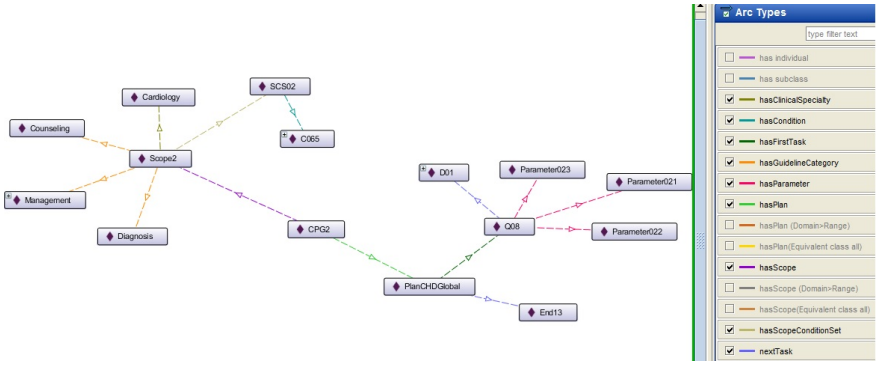

Figure 6: Graph containing the instances of a *Plan* for the management of coronary heart disease.

for further use in the editor. Based on the user's needs and feedback, this group is responsible for applying changes (if required) to the CompGuide Editor, that can positively influence its use.

Both the ontological model and the plug-in can be accessed through the use of the CompGuide GitHub repository[2] or the ontology and plug-in library of the Protégé wiki page[3].

[2] https://github.com/CompGuideRepository/CompGuide-Editor

[3] https://protegewiki.stanford.edu/wiki

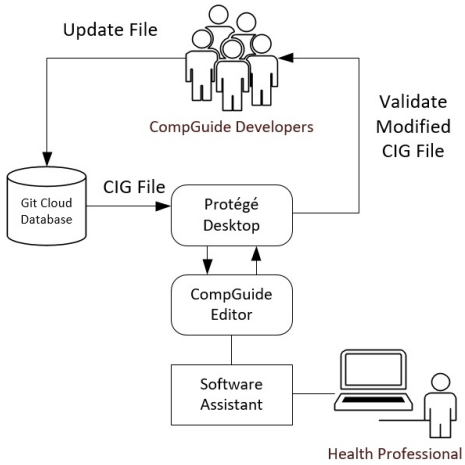

Figure 7: The CompGuide Editor System.

4.3 Assessment remarks

To verify the usability of the CompGuide Editor, a CPG from the National Comprehensive Cancer Network (NCCN) for the Diagnosis and Management of Colon Cancer (Benson et al., 2013) was fully represented in the CompGuide ontology. The CPG contains numerous clinical tasks with complex relationships. The process of representing the guideline resulted in an *owl* file with 680 instances, out of which 223 were task instances. Among the clinical tasks, 95 of them had temporal constraints, of which the most common was the *Periodicity*, featured in 79 tasks. Figures 8 shows an example of the creation process of the NCCN *Plan* and the association with the corresponding first task. Since this is a lengthy CPG, the CompGuide facilitated its acquisition by providing information step-by-step on which fields are required for the definition of each task and associated constraints. This was particularly useful in the definition of the procedural logic of the CPG, the sequence of clinical tasks, and splitting points in the CPG workflow, where it is necessary to choose one from multiple alternative tasks. The graphical view was particularly useful in the visualisation of this last aspect, allowing a rapid comprehension of the CPG workflow by selectively expanding and shrinking parts of the graph. These were the aspects in which CompGuide proved to be more useful.

Another important aspect is the capacity to re-utilise the knowledge contained in the ontology. This feature is shown in Figure 8 where the user selects a set of condition instances which are applied to the new CPG. While proceeding the creation/editing process, the wizard verifies if all the required data is correctly inserted. In case this verification process fails the system notifies which information is to be revised. Figure 9 shows the output of an unsuccessful verification case, where the *Periodicity* restriction of an *Action* clinical has missing

data. Given the complex and sensitive nature of treatments for cancer, namely in the definition of chemotherapy regimens, the clinical tasks reflecting these medical recommendations are bounded by numerous constraints. An example of is the statement: "CapeOx (the name of a chemotherapy regimen) should be applied every 3 months, with the administration of capecitabine every 12 hours for 14 days" (Benson et al., 2013). In this case there is a nested periodicity of 12 hours within the initial periodicity of 3 months. There is also a duration (14 days) bounding the nested periodicity. This kind of complex task has to be accurately defined and, thus, having it checked by the wizard upon knowledge acquisition is necessary.

As described earlier, when the management process is complete the wizard manages the structure of the instances. While this is happening, the tool also handles less relevant data, such as dates and versions. Since the tool has automation features, it allows the user to focus on the guideline's most crucial information, i.e. the actual clinical tasks. Figure 10 confirms the automatic management of the newly created CPG and *Plan* instances (as seen in the Protégé Entities tab) with the version and dates managed by the wizard.

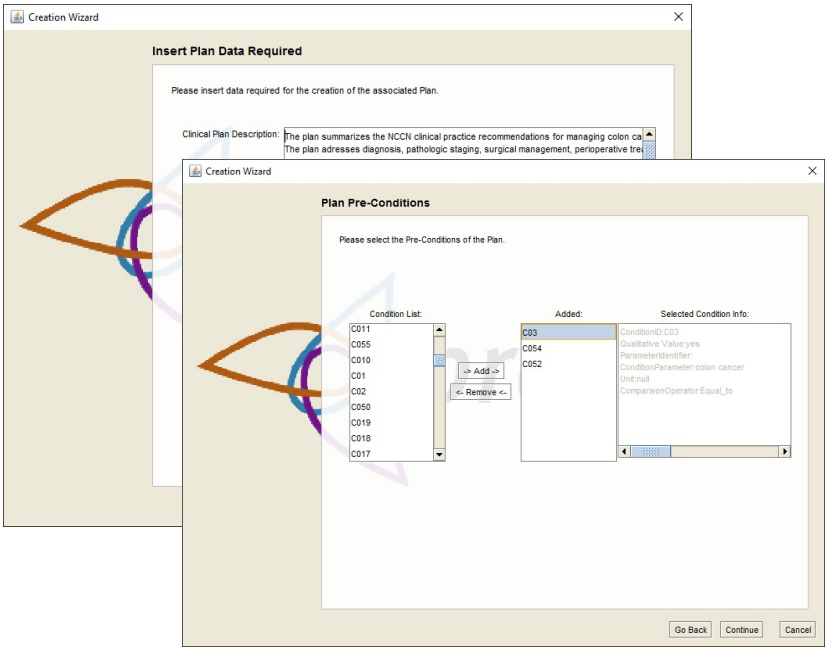

Figure 8: Creation of the Global Colon Cancer Plan.

Taking as an example the SAGE Workbench, which is the closest knowledge acquisition tool to the one presented herein, there are a number of aspects over which the CompGuide Editor can be considered an improvement. The SAGE Workbench spawns over eight knowledge tabs for the management of elements (SAGE Project, 2006), which increases the complexity of knowledge acquisi-

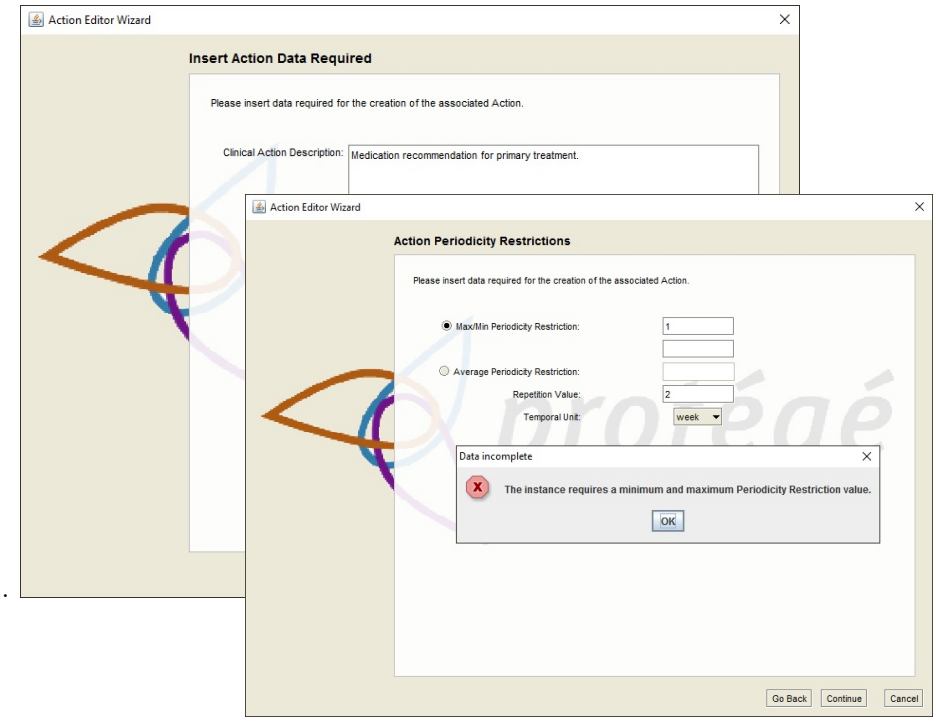

Figure 9: Failed verification based on incomplete inserted data.

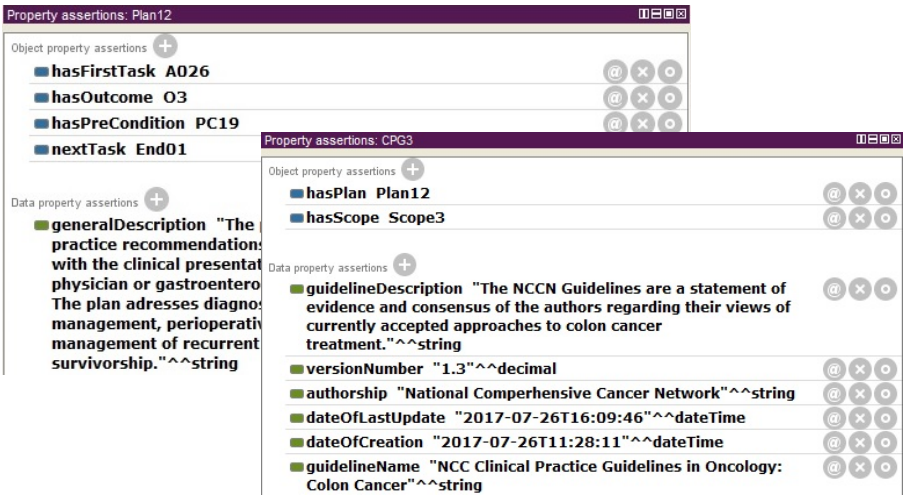

Figure 10: CPG and Plan creation result.

tion and the ability to establish or see the relationships between the different instances. By condensing the management of all elements in one view, the

CompGuide Editor allows one to keep, at all times, a sense of all the connections. The layered acquisition of CIGs and the focus on each kind of knowledge element is supported by the assistants in a guided way. Such guidance is absent from the SAGE Workbench, which demands a deeper understanding of the CIG model at the start of guideline encoding, thus implying a longer adaption time to the acquisition tool. Furthermore, the SAGE Workbench requires proficiency in Protégé Axiom Language, whilst in the CompGuide Editor the definition of clinical and temporal constraints is completely self-contained, in the sense that these tasks are fulfilled by using the basic elements of the ontology. In terms of the manipulation of knowledge elements, the graphical view offered by the SAGE Workbench is static and does not allow the dynamic visualisation and disposition of CIG knowledge elements. Notwithstanding the issues regarding automation, simplicity, and knowledge manipulation, there are features that the CompGuide Editor does not yet possess and are present in the SAGE Workbench. The first is the connection to a medical terminology service that ensures that the medical terms used throughout the CIG are correct, unambiguous, and homogeneous. The second is the automatic validation of CIGs. Currently in the CompGuide Editor, the verification performed is related with the data types of the attributes and presence/absence of necessary attributes upon acquisition, but no form of semantic validation of CIG content is performed. For this purpose, the functionalities of Protégé reasoners should be explored to their fullest extent. Their goal is to determine ontology consistency, identify subsumption relationships between classes, and so forth.

5 Conclusions and Future Work

The CompGuide ontology for CIGs aims to be self-contained and provide all the constructors for the definition of a work flow of procedures, clinical constraints, and temporal constraints using the basic elements of OWL, namely classes, properties, and instances. Following this intention, the CompGuide Editor was developed to acquire CPG knowledge in this ontology. Based on the properties of guideline visualisation, organisation, simplicity, automation, manipulation of knowledge elements, and the existence of a web or local repository, we have assessed existing CIG acquisition tools and identified aspects to improve. Such aspects were mainly related with the lack of simplicity and ease of comprehension showed by the current tools, allied to a poor guidance offered during CIG acquisition and the absence of automation in dealing with less important but needed information. Such features are implemented in the CompGuide Editor, largely through the software assistants for the three main CIG acquisition tasks: create, edit, and delete. Through a comparative analysis, we were able to establish herein in what way the contributions of the presented plug-in for Protégé are improvements over the current state of the art.

As future work, the integration of the editor with a medical terminology service is a necessity for ensuring the correctness of medical terms and will play an important role in the procedures for the semantic validation of the

CIG. The Unified Medical Language System (UMLS) (U.S. National Library of Medicine, 2016) is the open source medical terminology service chosen for integration, given the numerous source vocabularies it uses. At the same time, a usability study for the tool is necessary. Such a study has not been conducted yet given the difficulty in creating the necessary conditions for it. The current proposal for assessment is to gather volunteers to encode the same CPG using the CompGuide Editor and then having them answer a user experience questionnaire. Another group of volunteers would also encode the CPG, but using another tool from the state of the art, in order to have additional information to compare with the CompGuide Editor group. The challenge is that it is necessary to gather volunteers who are familiar with medical terms and they have to receive training in both the ontology and the tool beforehand, making this a time consuming process. Currently, the possibility of resorting to a crowdsourcing platform to perform this task is being studied.

Acknowledgements

This work has been supported by COMPETE: POCI-01-0145-FEDER-007043 and FCT – Fundação para a Ciência e Tecnologia within the Project Scope UID/CEC/ 00319/2013.

References

Beard, N., Campbell, J. R., Huff, S. M., Leon, M., Mansfield, J. G., Mays, E., ... others (2002). Standards-based sharable active guideline environment (sage). In *Amia annu symp proc.*

Benson, A., Bekaii-Saab, T., Chan, E., Chen, Y.-J., Choti, M., Cooper, H., & Engstrom, P. (2013). *NCCN Clinical Practice Guideline in Oncology Colon Cancer* (Tech. Rep.). National Comprehensive Cancer Network. Retrieved from `http://www.nccn.org/professionals/physician{_}gls/f{_}guidelines.asp`

Bott, R. (2014). Summary of the Guideline Workbenches Evaluation. *Igarss 2014* (1), 1–5. doi: 10.1007/s13398-014-0173-7.2

da Silva, I., & Freitas, C. (2011). Using visualization for exploring relationships between concepts in ontologies. In *2011 15th international conference on information visualisation* (p. 317-322). doi: 10.1109/IV.2011.40

de Clercq, P. A., Blom, J. A., Korsten, H. H. M., & Hasman, A. (2004). Approaches for creating computer-interpretable guidelines that facilitate decision support. *Artificial Intelligence in Medicine*, *31*, 1–27. doi: 10.1016/j.artmed.2004.02.003

Gonçalves, F., Oliveira, T., Neves, J., & Novais, P. (2017). CompGuide: Acquisition and Editing of Computer-Interpretable Guidelines. In Á. Rocha, A. M. Correia, H. Adeli, L. P. Reis, & S. Costanzo (Eds.), *Recent advances in information systems and technologies: Volume 1* (pp. 257–266). Cham:

Springer International Publishing. Retrieved from http://dx.doi.org/10.1007/978-3-319-56535-4{_}26 doi: 10.1007/978-3-319-56535-4_26

Isern, D., & Moreno, A. (2008). Computer-based execution of clinical guidelines: a review. *International journal of medical informatics*, *77*(12), 787–808. doi: 10.1016/j.ijmedinf.2008.05.010

Lohr, K. N., Field, M. J., et al. (1990). *Clinical practice guidelines: directions for a new program* (Vol. 90) (No. 8). National Academies Press.

Martínez-Salvador, B., & Marcos, M. (2016). Supporting the refinement of clinical process models to computer-interpretable guideline models. *Business & Information Systems Engineering*, *58*(5), 355–366. doi: 10.1007/s12599-016-0443-3

Noy, N. F., Crubézy, M., Fergerson, R. W., Knublauch, H., Tu, S. W., Vendetti, J., ... others (2003). Protege-2000: an open-source ontology-development and knowledge-acquisition environment. In *Amia annu symp proc* (Vol. 953, p. 953).

Oliveira, T., Novais, P., & Neves, J. (2013). Representation of clinical practice Guideline components in OWL. In J. B. Pérez et al. (Eds.), *Advances in intelligent systems and computing* (Vol. 221). Springer International Publishing. Retrieved from http://dx.doi.org/10.1007/978-3-319-00563-8{_}10 doi: 10.1007/978-3-319-00563-8_10

Oliveira, T., Novais, P., & Neves, J. (2015). Assessing an Ontology for the Representation of Clinical Protocols in Decision Support Systems. In J. Bajo et al. (Eds.), *Advances in intelligent systems and computing* (Vol. 372, pp. 47–54). Springer International Publishing. doi: 10.1007/978-3-319-19629-9_6

Oliveira, T., Silva, A., Neves, J., & Novais, P. (2016). Decision Support Provided by a Temporally Oriented Health Care Assistant. *Journal of Medical Systems*, *41*(1), 13. doi: 10.1007/s10916-016-0655-6

Peleg, M. (2013). Computer-interpretable clinical guidelines: A methodological review. *Journal of Biomedical Informatics*, *46*(4), 744–763. doi: 10.1016/j.jbi.2013.06.009

Peleg, M., Boxwala, a. a., Ogunyemi, O., Zeng, Q., Tu, S., Lacson, R., ... Greenes, R. a. (2000). GLIF3: the evolution of a guideline representation format. In *Proceedings / amia ... annual symposium. amia symposium* (pp. 645–649). doi: D200640[pii]

Purves, I. N., Sugden, B., Booth, N., & Sowerby, M. (1999, jan). The PRODIGY project–the iterative development of the release one model. In (pp. 359–63).

SAGE Project. (2006). *SAGE Guideline Encoding Tools.* Retrieved 28/07/2017, from http://sage.wherever.org/encoding/encoding{_}tools.html

Shiffman, R. N., Agrawal, A., Deshpande, A. M., & Gershkovich, P. (2001). An approach to guideline implementation with gem. *Studies in health technology and informatics*(1), 271–275. doi: 10.3233/978-1-60750-928-8-271

Silberstein, S. (2005). Clinical practice guidelines. *Journal of Neurosurgery Pediatrics*, *25*(10), 765–766. doi: 10.1111/j.1468-2982.2005.01014.x

Steele, R., & Primer, F. J. T. P. (2002). *Introduction to proforma language and software with worked examples* (Tech. Rep.). Technical report. London, UK: Advanced Computation Laboratory, Cancer Research.

Sutton, D. R., & Fox, J. (2003). The syntax and semantics of the proforma guideline modeling language. *Journal of the American Medical Informatics Association*, *10*(5), 433–443. doi: doi.org/10.1197/jamia.M1264

U.S. National Library of Medicine. (2016). *Unified Medical Language System (UMLS).* Retrieved 28/07/2017, from `https://www.nlm.nih.gov/research/umls/`

Votruba, P. (2003). *Structured knowledge acquisition for asbru.* Viena University of Technology.