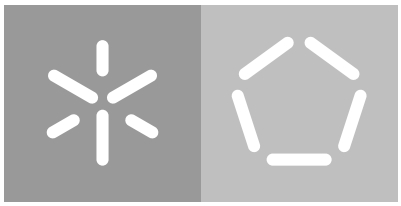


Universidade do Minho
Escola de Engenharia
Departamento de Informática

Filipe Manuel Gonçalves
pg25309

Computer-Interpretable Guidelines in Decision Support Systems

Creation and Editing of Clinical Protocols for Automatic Interpretation

Master dissertation
Master Degree in Computer Science

Dissertation supervised by
Paulo Jorge Novais

December 2016

ACKNOWLEDGEMENTS

The completion of a dissertation thesis involved not only the study of one researcher, but always the contribution of several researchers who have devoted much of their time to contribute their studies to the scientific community. As such, I thank you all to those who contributed to the development of this project.

I would like to thank my advisor, Professor Paulo Jorge Freitas de Oliveira Novais, who demonstrated the availability, the trust in my work, for the good advices, and especially the motivation conveyed.

Special thanks to my co-advisor Tiago José Martins Oliveira for the attention he gave, the clarifications provided, for the opportunity to work in this great project, for the confidence placed in me and for his friendship and patience that has been shown throughout the whole process.

I thank all my friends who always supported me in good and bad moments.

And last but not least, a big thanks to the most important people in my life, my father Fernando Gonçalves and my grandmother Laura Albuquerque, who always provided me with the means, guidance and support to achieve my dreams.

As such, I dedicate this work to those I love the most.

ABSTRACT

Currently in the health sector there is a growing need to standardize and promote the improvement of clinical practice in order to reduce costs, which requires a solution that will allow these goals to be more easily achieved. To this end, the solution that gathers the current interest is the use of clinical protocols and promoting conformity with practices contained in them.

Clinical protocols aim to improve the quality of the clinical process, reducing variations in clinical practice and reducing health care costs. In order to be effective, these parameters must be integrated into the care flow and provide specific advice to a patient, regardless of time or place. Thus, their formalization as Computer-Interpretable Guidelines (CIG) makes possible the development of decision support systems based on CIGs, which may have a greater impact on the behavior of health professionals.

However, the absence of a general pattern in terms of CIG often hinders progress in the development of these systems. Currently available tools for creating and editing clinical protocols for automatic interpretation are not functional or user-friendly. Most of them are academic projects developed in obsolete languages.

As a means to solve this issue, this dissertation project presents an user-friendly tool that manages the creation and editing of CIGs, without requiring the user to have programming knowledge, and through the use of interfaces that are simple and intuitive.

Keywords - Artificial Intelligence in Medicine, Clinical Decision Support System, Clinical Protocols, Computer-Interpretable Guidelines.

RESUMO

Atualmente no setor da saúde há uma crescente necessidade de padronizar e promover a melhoria das práticas clínicas com o intuito de reduzir custos, o que exige uma solução que permita que estes objetivos sejam mais facilmente atingidos. Para o efeito, a solução que mais desperta o interesse atualmente é a utilização de protocolos clínicos e reforço da conformidade com as práticas que neles são recomendadas.

Os protocolos clínicos visam melhorar a qualidade do processo clínico, reduzindo as variações da prática clínica e reduzindo os custos de saúde. De forma a serem eficazes, devem ser integrados no fluxo de atendimento e prestar aconselhamento específico para um paciente, independentemente do tempo ou local onde se encontram. Assim, a sua formalização como Computer-Interpretable Guidelines (CIGs) torna possível o desenvolvimento de sistemas de apoio à decisão baseados em CIGs, que apresentam uma maior capacidade de afetar o comportamento dos profissionais de saúde.

Contudo, a inexistência de um padrão generalizado a nível das CIGs dificulta muitas vezes o progresso no desenvolvimento destes sistemas. As ferramentas atualmente disponíveis para a criação e edição de protocolos clínicos para interpretação automática não são funcionais ou de fácil utilização. Como meio de resolver esta questão, neste projeto de dissertação propõe-se o desenvolvimento de uma ferramenta *user-friendly* capaz de gerir a criação e edição de CIGs, sem a necessidade do utilizador apresentar conhecimentos de programação, e através do uso de interfaces que sejam simples e intuitivas.

Keywords - Inteligência Artificial em Medicina, Sistema de Apoio à Decisão Clínica, Protocolos Clínicos, Guiões Computer-Interpretable.

CONTENTS

NOTATION AND TERMINOLOGY

NOTATION

Throughout the document acronyms related to the representation of names of models and clinical protocols are used. In order to best understand them, this chapter was created in order for the reader to understand their interpretation.

ACRONYMS

IT - Informatics Technology
AI - Artificial Intelligence
AIM - AI in Medicine
CDSS - Clinical Decision Support System
CIG - Computer-Interpretable Guidelines
CP - Clinical Protocol
CPG - Clinical Protocol Guideline
CDS - Clinical Decision Support
JDK - Java Development Kit
JVM - Java Virtual Machine
API - Application Programming Interface
IDE - Integrated Development Environment
IOM - Institute of Medicine
OS - Operational System
OWL - Web Ontology Language
IRI - Internationalized Resource Identifier
XML - eXtensible Markup Language
RDF - Resource Description Framework
W3C - World Wide Web Consortium
SHOE - Simple HTML Ontology Extensions
OIL - Ontology Inference Layer
DAML - DARPA Agent Markup Language
TNM - Task Network Model
SAGE - Standards-based Shareable Active Guideline Environment
Protégé - Ontology Editor and Knowledge Acquisition System

GEM - Guideline Elements Model
DTS - Distributed Terminology System
OKBC - Open Knowledge Base Connectivity Protocol
UML - Unified Modelling Language
TCP - Transmission Control Protocol
IP - Internet Protocol

LIST OF FIGURES

LIST OF TABLES

1

INTRODUCTION

This document was developed as a Master's dissertation in Computer Science at the University of Minho. It is entitled *Computer-Interpretable Guidelines in Decision Support Systems: Creation and Edition of Clinical Protocols for Automatic Interpretation in Clinical Decision Support Systems*.

The work covers areas of computer science such as Artificial Intelligence (AI), e-Health, Medical Informatics and Clinical Decision Support Systems (CDSS).

In this chapter the motivation and background of this work will be presented, followed by a short explanation of the theme and objectives to accomplish, the research method used, and the structure of the document.

1.1 MOTIVATION

The possibility of developing programs that simulate intelligent behavior, took the form of AI, a term coined by John McCarthy (McCarthy, 2001). According to Eysenck and Keane (1994), the human being can be seen as an information processor. When making a connection between mind and computer, through AI, it became possible to develop models based on neural systems, trying to mimic the human being in its complexity, teaching the computer to think (Eysenck and Keane, 2005). Since then, researchers have been dreaming of creating an electronic brain. Of all the modern technological missions, this research to create AI in computer systems has been one of the most ambitious and controversial.

From the early developments in AI, researchers and doctors were interested in the potential that technology can have in medicine. With intelligent computers able to store and process vast amounts of knowledge, the hope was that they would become perfect doctors, assisting or surpassing health professionals in tasks such as diagnosis.

Clancey and Shortliffe (1984) provided the following definition: "AI in Medicine mainly deals with the construction of AI programs that perform diagnoses and therapy recommendations" (Clancey and Shortliffe, 1984).

1.2 CLINICAL PROTOCOLS

Clinical Protocols (CPs) are decision tools that allow to shorten the distance between the actual clinical practice and the optimal clinical practice. However, they are also described as documents developed in a systematic way to improve the quality of care, reduce unjustified variations in medical practice and reduce health care costs . In order to be effective, clinical protocols should be integrated into the flow of care and provide specific advice for each patient, when and where needed. The main objectives to be achieved by CPs are improving the delivery of health care to patients, decreasing costs and reducing the variability of medical practice (ten Teije et al., 2008).

The development of guidelines reflects a drive towards evidence-based medicine, and is designed to achieve a reduction in practice variation, and some degree of standardization of clinical practice for the benefit of patients. This standardization is agreed to be the best way to reduce medical error, which is acknowledged to be a problem in medicine ((Leape et al., 1993) and (Gopher et al., 1989)).

Although medicine is a complex and hazardous business, Leape (1993) points out that an error rate of 1%, as found in Gopher's study (1989) of an intensive care unit, would be unacceptable in other high hazard industries. Moreover, recent high visibility cases of error have brought the issue to the attention of the public, and there is growing concern about the accountability of health professionals. One reason why the use of protocols may be particularly difficult to manage in medicine concerns the professional independence fostered by the culture of medicine. In other industries one of the main methods for implementing protocols involves monitoring and enforcement. The fact that the practice of health professionals is largely self-regulated may explain why much of the medical research on CPs to date, which concentrates on compliance rates, has shown that compliance appears to be low (Grilli and Lomas, 1994).

All of these factors mean that there is a growing interest in getting health professionals, and especially doctors, to follow newly introduced CPs. Given that the number of CPs is set to increase, it is important to understand the attitudes of health professionals to their use.

1.3 CLINICAL DECISION SUPPORT SYSTEM

The general idea of CDSSs is that of computer programs that help doctors make diagnoses. Although computers play a number of important clinical functions, people have recognized since the early days of computing that computers can support health professionals, helping them to filter out the vast collections of possible diseases, findings and treatments (Khalifa, 2014).

CDSS are experts in the clinical area designed to help doctors and other health professionals in the clinical process , in tasks such as determining the diagnosis based on patient data (Berner, 2007).

However, they are viewed with some scepticism by health professionals. The reason is that these tools can promote other types of errors related to the entry and retrieval of information, and communication failure. The goal of treatment is to manage the trajectory of a patient in order to produce an improvement in his medical condition, which implies a high level of detail of the information used to make decisions.

1.4 Computer-Interpretable Guidelines

The implementation of CPs in CDSSs has the potential of improving the acceptance and application of CPs in daily practice, because the systems are able to monitor the actions and observations of health professionals and provide advice at the point of care (de Clercq et al., 2004).

CIGs are increasingly applied in various fields. According to the Institute of Medicine (IOM), these decision support systems are in fact crucial elements in long-term strategies for promoting the use of guidelines (Field et al., 1992).

The conversion of computer algorithms from their text versions is not an easy task , as these versions were not originally designed to be interpretable by computers and in some cases contain complex instructions , which handle too many variables and it is difficult to translate into efficient algorithms (Chim, JCS and Cheung, NT and Fung, H and Wong, 2003).

Sometimes the vocabulary used in the documents is evasive, featuring words to quantify measures rather than numerical limits, and the criteria in the decision points are not always explicit and indicate what to do. The lack of precision of concepts gives rise to ambiguity and gaps in knowledge, in which computers can't handle (Chim, JCS and Cheung, NT and Fung, H and Wong, 2003). The greater simplicity and assertiveness of a protocol, the easier it is to adapt to the CIG format.

This led to the development of different CIG models tools by different research groups, covering a wide range of clinical situations (Isern and Moreno, 2008).

According to the Agency for Health care Research and Quality in the USA, the characteristics to be fulfilled by clinical guidelines are validity, reproducibility, reliability, clinical flexibility, clarity and scheduled review. However, most clinical guidelines do not necessarily fully satisfy these factors.

Problems in development of guidelines are as follows: The first is the lack of high level evidence such as randomized controlled trials, which influences recommendation grade.

The second is the ease of clinical application. Evidence based clinical guidelines are not likely to be easy to use if sufficient high-level scientific evidence is not available.

Despite these efforts, only a few systems have progressed beyond the prototype status and research project. Build systems that are accepted by professionals in this area proved to be a difficult task.

1.5 ADVANTAGES OF STRUCTURED FORMATS OF CIGS

There are four areas of great importance in the design of CIGs, which should be considered in the development of CIG format to be used (de Clercq et al., 2004). These areas are:

- Modelling and representation of CPs;
- Acquisition of CPs;
- Verification and test of CPs;
- Implementation of CPs.

Each of them has a number of aspects which serve analysis parameters of different approaches to modelling CIGs. Among the main models of CIGs, the ones to be highlighted are Arden Syntax (developed in University of Columbia (Hripcsak, 1994)), PROforma (developed in the Imperial Cancer Research Fund in England (Fox et al., 1997)), GLIF (developed by InterMed Collaboratory (Patel et al., 1998)), and *Asbru* (originally developed by University of Standford and currently developed in the Vienna University of Technology and Ben-Gurion University (Rospocher et al., 2010)).

The representation format of a CIG is an important component for its implementation. However, in the development of CDSSs, there are other steps and technologies that should be highlighted.

A first level to be addressed is the modelling technologies. The object orientation is a technology that lets you specify the domain CDSS. It is flexible to change, lets you create and implement fully reusable software components (Adratt, Eduardo and LIMA, L and BARRA, 2004).

Other important aspect in the implementation of CIGs is the ability to communicate. Health information should befall technological changes and the data must be shared and reusable, not being constricted by hardware, product or operational system limitations (Adratt, Eduardo and LIMA, L and BARRA, 2004).

The development of formalisms for representing CIGs is an area of great interest, however there is no dominant CIG platform and no system has a widespread use outside the institution where they were developed, demonstrating often the inability of these tools in creating and editing guidelines (und Naturwissenschaften, 2015).

By studying the main application properties of these CIGs , the motivation of this project lies in implementing a user-friendly tool to represent clinical guidelines in a specific model, able to fill the limitations of the existing applications.

1.6 SCOPE

1.6.1 *e-Health*

E-Health is an emerging field in the intersection of medical informatics, public health and business, referring to health services and information delivered or enhanced through the Internet and related technologies. In a broader sense, the term characterizes not only a technical development, but also a state-of-mind, a way of thinking, an attitude, and a commitment for networked, global thinking, to improve health care locally, regionally, and worldwide by using information and communication technology (Peleg et al., 2003).

The "e" in e-Health does not only stand for "electronic," but implies a number of other "e's," which together perhaps best characterize what e-Health is all about (or what it should be, namely (Eysenbach, 2001)).

1. **Efficiency** - one of the promises of e-Health is to increase efficiency in health care, thereby decreasing costs. One possible way of decreasing costs would be by avoiding duplicative or unnecessary diagnostic or therapeutic interventions, through enhanced communication possibilities between health care establishments, and through patient involvement.

2. **Enhancing** quality of care - increasing efficiency involves not only reducing costs, but at the same time improving quality. E-Health may enhance the quality of health care for example by allowing comparisons between different providers, involving consumers as additional power for quality assurance, and directing patient streams to the best quality providers.

3. **Evidence** based - e-Health interventions should be evidence-based in a sense that their effectiveness and efficiency should not be assumed but proven by rigorous scientific evaluation. Much work still has to be done in this area.

4. **Empowerment** of consumers and patients - by making the knowledge bases of medicine and personal electronic records accessible to consumers over the Internet, e-Health opens new avenues for patient-centered medicine, and enables evidence-based patient choice.

5. **Encouragement** of a new relationship between the patient and health professional, towards a true partnership, where decisions are made in a shared manner.

6. **Education** of physicians through online sources (continuing medical education) and consumers (health education, tailored preventive information for consumers).

7. **Enabling** information exchange and communication in a standardized way between health care establishments.

8. **Extending** the scope of health care beyond its conventional boundaries. This is meant in both a geographical sense as well as in a conceptual sense. e-Health enables consumers to easily obtain health services online from global providers. These services can range from simple advice to more complex interventions or products such a pharmaceuticals.

9. **Ethics** - e-Health involves new forms of patient-physician interaction and poses new challenges and threats to ethical issues such as online professional practice, informed consent, privacy and equity issues.

10. **Equity** - to make health care more equitable is one of the promises of e-Health, but at the same time there is a considerable threat that e-Health may deepen the gap between the "haves" and "have-nots". People, who do not have the money, skills, and access to computers and networks, cannot use computers effectively. As a result, these patient populations (which would actually benefit the most from health information) are those who are the least likely to benefit from advances in information technology, unless political measures ensure equitable access for all. The digital divide currently runs between rural vs urban populations, rich vs poor, young vs old, male vs female people, and between neglected/rare vs common diseases.

However , despite all the benefits that e-Health can provide in clinical areas, through the use of AI is possible to improve their implementations using CDSS, and CIGs may have an active role in this improvement, provided that their availability is increased.

1.6.2 *Artificial Intelligence in Medicine*

AI is the study of ideas which enable computers to do the things that make people seem intelligent. The central goals of AI are to make computers more useful and to understand the principles which make intelligence possible (Saridis, 2001). AIM is AI specialized to medical applications.

The earliest work in AIM dates to the early 1970s, when the field of AI was about 15 years old (Benko and Sik Lányi, 2009)). Early AIM researchers had discovered the applicability of AI methods to life sciences that demonstrated the ability to represent and utilize expert knowledge in symbolic form. The general AI research community was fascinated by the applications being developed in the medical world, noting that significant new AI methods

were emerging as AIM researchers struggled with challenging biomedical problems. Over the next decade, the community continued to grow, and with the formation of the American Association for Artificial Intelligence in 1980, a special subgroup on medical applications was created (Kononenko, 2001).

Though the introduction of personal computers and high-performance workstations it was possible to develop new types of AIM research and new models for technology dissemination. If the computer is a useful manager of billing records, it should also maintain medical records, laboratory data, data from clinical trials, etc. And if the computer is useful to store data, it should also help to analyse, organize, and retrieve it.

Often, health professionals are skeptical regarding the use of these technologies, because they are afraid of losing their jobs. However, this concern is not justified. AI no longer aims the substitution of professionals by computer artifacts. AI aims to improve the usability of programs for assisting physicians in figuring out what is wrong with the patients and provide new solutions to help making better decisions (Horn, 2001).

Information technology, in general, can help improving human health and longevity. To achieve this goal innovative and intelligent software can be deployed in order to improve medical research, disease prevention, and health care service delivery.

With this work, it's intended to create a tool that aids in the management of a knowledge base for automatic execution engine of CIGs. In other words, the creation of the substrate under which the decision support system works.

1.7 THEME AND OBJECTIVES

The theme of this work is the Computer-Interpretable Guidelines in Decision Support Systems: Creation and Editing of Clinical Protocols for Automatic Interpretation. Taking as starting point CDSSs that use CIGs as support for their knowledge base, the objective of this work lies in the study of the main aspects of the creation and editing tools of CIGs for automatic interpretation that are currently being used. For this we identify aspects that could be improved based on the comparison of existing tools, and develop a CIG tool that incorporates these improvements.

The research questions that guided the execution of work were:

- What creation and editing of CIG tools are being used?
- What aspects could be improved or applied in these tools? (easy-to-use? reliable? too complex?)
- How should a new CIG tool be planned? (platform to use? performance? interface?)
- Should a new CIG tool allow for the users to publish their modified CIG files to a development team? (communication?)

The research questions previously specified allowed us to state the following objectives to be achieved:

1. Identify key aspects for representing medical knowledge in CDSSs;
2. Identify tools used to create and edit CIGs and their key issues;
3. Identify aspects that could be improved or applied in these tools;
4. Formalize the creation and editing of CPs in the OWL language;
5. Design a tool capable of managing the stored set of parameters in CPs, without the need for advanced programming skills;
6. Design features capable of generating graphical representations of CPs;
7. Design features capable of downloading the latest CIG files (when the platform is connected to the web);
8. Design features capable of sending their modified CIG files to the development team (when the platform is connected to the web).

1.8 RESEARCH METHODOLOGY

Regarding the research methodology we adopted the action-research methodology (Somekh and Bridget, 2005). Initially a crucial collection of the information was gathered for the construction of a solution design process. Then the research of relevant concepts and designs for the job began. The assimilation of concepts and projects were subject to constant renewal, as new ideas and information arose. The last part of the work was the development of a functional model and prototype that allowed the achievement of the set of goals.

This research methodology has five iterated identifiable phases:

1. **Diagnosing** - Definition of the problem and its characteristics;
2. **Action planning** - Constant updating of state of the art and objectives of the work;
3. **Action taking** - Development of a prototype in order to achieve the defined objectives;
4. **Evaluating** - Analysis and prototype correction based on the results obtained;
5. **Specifying learning** - The diffusion of knowledge and results obtained in the scientific community.

As for the development of software solutions the methodology used will be adapted from SCRUM. As such, all previously explained steps will be applied in software development. The first steps are diagnosing the problem and updating the state of art and objectives of the work. Next is the software development of the proposed objectives. With these tasks completed, an evaluation of the work will be done, whose results are reported in the paper. Through this results, new problems arise which leads to a new cycle.

Scrum development is a simple methodology intended to solve long product development which allows the developer to focus in the set of goals proposed. This methodology also solves the mismatch problem between a product's business requirement and the actual resulting implementation (which normally occurs when developing big products).

1.9 DOCUMENT STRUCTURE

This work was structured in five chapters, organized as follows:

1. **Introduction** - In the first chapter there is a brief description of the current situation, an introduction to key concepts and a presentation of motivation, theme, objectives and research methodology. Also a brief description of the document is performed;

2. **State of the art in Computer-Interpretable Guidelines Tools** - The second chapter deals with creation and editing tools of CIGs, referring its importance and the benefits and disadvantages of their use. These aspects are subsequently used to point out the main features of some models such as *SAGE Workbench*, *Protégé Desktop*, *Tallis* (which uses PROForma model), *GEM Cutter* and *Asbru View*. At the end of the chapter an analysis of the key aspects of these tools is performed and their main limitations are identified;

3. **Clinical Protocols in *CompGuide*** - The third chapter analyzes a number of factors related to the *CompGuide* OWL structure, such as the data structure of the OWL classes used in the *CompGuide* ontology, its domain model, who will be using this system (system actors), the plug-in functional and non-functional requirements and its use case diagrams. In the end, a conclusive discussion is made, defining the main conclusions in this chapter;

4. **Implementation in Protégé Desktop** - The fourth chapter addresses the system according to a point of view of implementation and software development based on the analysis of the problem and according to the studied artifacts in software engineering. Therefore, it is pertinent to address the software architecture, technologies and tools used, class diagrams, sequence diagrams, a small view of the application interfaces, among others;

5. **Conclusion** - The last chapter summarizes the work done so far and the main conclusions to be drawn;

2

STATE OF THE ART IN COMPUTER-INTERPRETABLE GUIDELINES TOOLS

This chapter intends to describe the features of existing creation and editing tools of CPs, and discuss and compare their key aspects and main deficiencies.

In computer science, several languages and tools exist for helping users and system developers in creating good and effective CIGs. In particular, various tools help people create, either manually or semi-automatically categories, partonomies, taxonomies, and other organization levels of CIGs (Cristani, Matteo and Cuel, 2005). Some of the most important modelling editors and CIG managers are:

- *Protégé Desktop*;
- *SAGE Workbench*;
- *Tallis*;
- *GEM Cutter*;
- *Asbru View*;

In the next sub-chapters, all these tools will be explained with more detail.

2.1 PROTÉGÉ DESKTOP

Protégé (Musen and Protégé Team, 2015) is 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 customized knowledge-based tools and assists users in the construction of large electronic knowledge bases. *Protégé* provides two main ways of modelling ontologies:

- **Protégé-Frames editor** - the knowledge model is compatible with the Open Knowledge Base Connectivity protocol (OKBC). Therefore, all entities (i.e., instances, classes, slots, facets, and constraints) are frames. Instances represent objects in the domain of

interest. Classes are either named collections of instances or abstract conceptual entities in the domain (e.g., the concept of a drug ingredient). Slots are binary relations describing properties of classes (e.g., the indications of a drug). Facets describe properties of slots (e.g., the data type of a slot's value). Constraints specify additional relationships that must hold among instances;

- **Protégé-OWL editor** - enabling users to build ontologies in OWL;

Protégé supports the construction of a domain ontology, the design of customized knowledge acquisition forms, and entering domain knowledge that can be adapted to enable conceptual modelling with new and evolving Semantic Web languages. *Protégé* lets us think about domain models at a conceptual level without having to know the syntax of the language ultimately used on the Web. We can concentrate on the concepts and relationships in the domain and the facts about them that we need to express (Noy, Natalya F and Sintek, Michael and Decker, Stefan and Crubézy, Monica and Fergerson, Ray W and Musen, 2001).

It provides a platform which can be extended with graphical widgets for tables, diagrams, and animation components to access other knowledge-based systems embedded applications. *Protégé* is a library, which other applications can use to access and display knowledge bases. It can be extended by way of a plug-in architecture and a Java-based API for building knowledge-based tools and applications. *Protégé* is used to author guidelines in various models. Part of the modelling can be accomplished using predefined graphical symbols. These symbols are arranged in a diagram and linked by graphs. The underlying data is entered by forms (Leong et al., 2007).

Protégé ontologies can be exported into a variety of formats including RDF(S), OWL, and XML Schema (Rubin et al., 2007).

Figure 1 shows the interface of the *Protégé Desktop* application.

1 https://github.com/protegeproject/owlviz

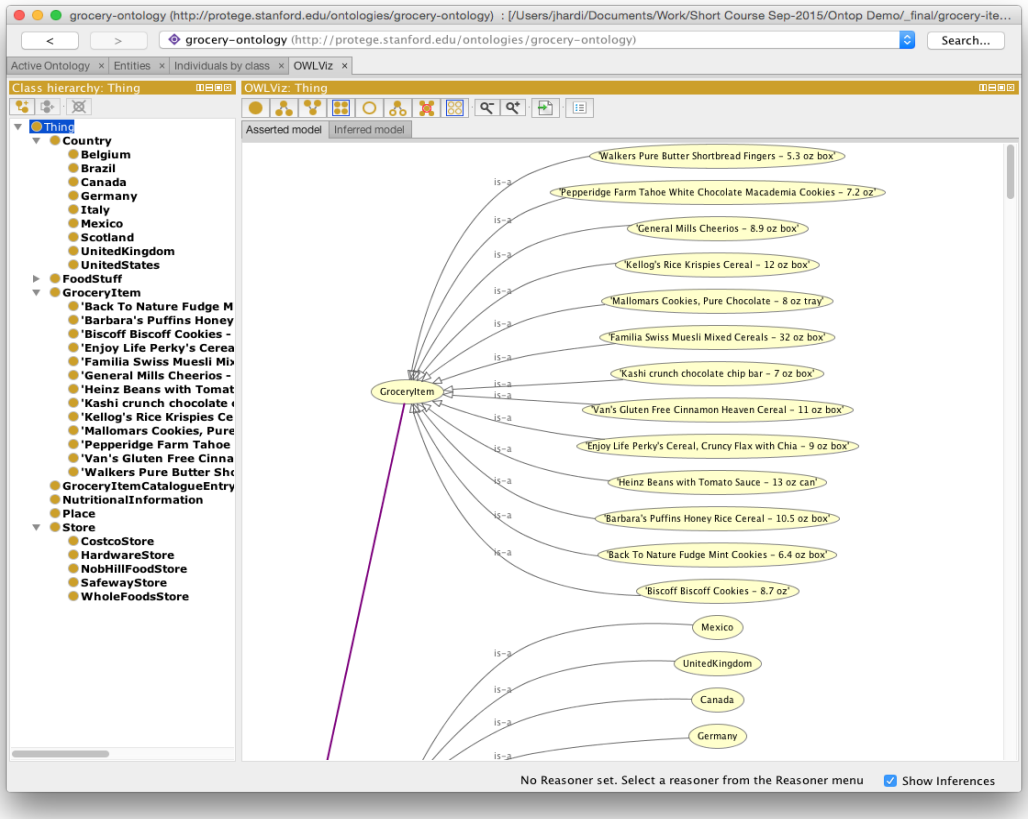


Figure 1.: *Protégé Desktop* Interface illustrating a guideline workflow (extracted from OWLViz Git Project [1]).

2.2 SAGE WORKBENCH

The *SAGE* Workbench is a complete, self-contained environment that uses *SAGE* guideline model. This model encodes guideline knowledge needed to provide situation-specific decision support and use standardized components for interoperability. *SAGE Workbench* provides a knowledge authoring tool based on *Protégé*. Also *SAGE* defines the knowledge deployment process and knowledge execution architecture (Beard et al., 2002).

SAGE Guideline Workbench includes a suite of tools that project members use to create, view, edit, and validate *SAGE* guidelines that conform to the format of the *SAGE* Guideline Model and that are executable by the *SAGE* Execution Engine to provide decision support for guideline-based care. The project has a number of requirements for the guideline workbench. It should be a tool that:

- Supports encoding process;

- Provides connection to terminology services to be used during encoding;
- Allows debugging/validation of guidelines;
- Provides a document-oriented view of the guideline knowledge base so that clinicians and knowledge engineers can easily review the content of the knowledge base.

SAGE Workbench also includes a terminology plug-in (the *SAGE* DTS tab) which accesses via the Internet the Apelon DTS (Distributed Terminology System) terminology service (developed by Apelon, Inc., USA). This plug-in allows users to view standard and *SAGE*-based terminologies, do concept queries, and view complex logical concept expressions. The Apelon DTS utilizes client-server technology and access to this functionality requires the user to log in to the DTS server. Note that it is not necessary to access Apelon's terminology service in order to use the workbench or run the demonstration guidelines.

One of its key approaches is to integrate guideline based decision support with the workflow of care process. In addition, *SAGE* is recognized as one of the improved Clinical Decision Support (CDS) architectures with its large coverage of knowledge base. *SAGE* includes a knowledge authoring tool based on *Protégé*.

Therefore, *SAGE* can be a strong and concrete knowledge representation model for clinicians (Kim, J and Shim, BinGu and Kim, SunTae and Lee, JaeHoon and Cho, InSook and Kim, 2009);

The *SAGE* website [2] provides a description of the method for encoding *SAGE* guideline applications as well as full documentation on all the software modules making up the package. *SAGE* Workbench is designed for use on MS Windows computers.

The user interface for the *SAGE* Workbench is organized as a number of tabs as shown in figure 2.

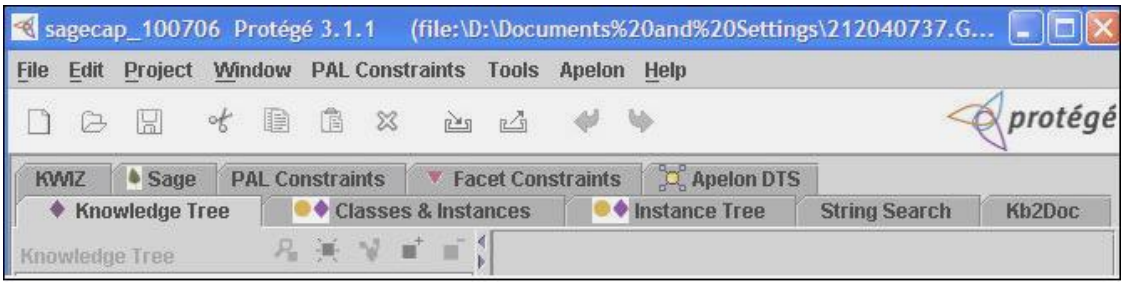


Figure 2.: Tabs shown in the *Protégé Desktop* platform using *SAGE* Workbench Plug-in (extracted from *Sage* Website [3]).

2 http://sage.wherever.org/
3 http://sage.wherever.org/encoding/encoding_tools.html

- KnowledgeTree Tab allows navigation of frames that are directly and indirectly referenced from a selected instance in a tree structure. It allows you to browse and edit, in a single window, all the frames reachable from the top-level instance (usually the top-level Guideline instance);

- Facet Constraints Tab allows a user to identify and fix all instances in a knowledge base, or instances of selected classes, that have slots whose values violate constraints associated with the slot. If a user wanted to find all instances that had constraint violating facets in his/her ontology, they had to manually iterate through all the instances in their ontology looking for red boxes that identified slots that had constraint violating facets. The Facet Constraints Tab allows a user to accomplish this task with the click of one button;

- PAL (*Protégé* Axiom Language) is a subset of first-order logic which can be used to express integrity constraints about a knowledge base. The PAL Constraints Tab is a front-end for this constraint system. This Tab allows a user to create, browse, and modify constraints in the knowledge base, and to evaluate constraints (either as a group or individually);

- *Apelon* terminology Tab and plug-in (i.e. and integration between *Apelon* software and *Protégé*) works with *Apelon* DTS 3.0 server across Internet. This plug-in allows a search of terms from several terminologies and creation of a reference in the *Protégé* guideline Workbench to a term in a standard terminology;

- Kwiz Tab allows to customize high-level views of the knowledge base, constrained navigation, reuse of existing knowledge bases, context-sensitive search and help;

- *SAGE* Tab provides a self-contained testing environment within *Protégé* for an encoded guideline. After validating the knowledge base, a user can select data from a test case and run the guideline by simulating the arrival of triggering events. After evaluating possible immunizations that may be due, it requests information on immunization consent and serious illnesses that may render immunization inadvisable. Once responses to the questions are submitted, the *SAGE* Execution Engine will generate its final recommendations for the immunizations that should be given;

- Kb-to-doc Document-generation Tab allows to generate a document-oriented view of the contents of the encoded guideline;

2.3 TALLIS

Tallis is a new Java implementation of PROforma-based authoring and execution tools developed by the Cancer Research UK (Sutton and Fox, 2003). *Tallis* is based on a later version of the PROforma language model (Steele and Primer, 2002).

Tallis consists of a suite of applications:

- Composer - a desktop graphical editor for authoring PROforma guidelines. Composer includes a test application for simulating interaction with your PROforma guidelines, and contains default settings for enacting your application in OpenClinical saving your application in the OpenClinical repository;
- Tester - a desktop application for testing PROforma guidelines. The tester comes bundled with Composer, but may be run as a standalone application if required;
- Web Enactment - a web application for enacting your application on the web, used in OpenClinical;

Each of these applications will run on any platform and integrate with other components, including third party applications. Yet it requires the *Tallis* engine and core plugins (Martínez-Salvador and Marcos, 2016).

PROforma is a language for describing the activities that are to be carried out by some agent to achieve particular objectives in some situation, possibly under various kinds of practical constraints (e.g. timing, resources or information constraints). It combines features of a specification language as developed in software engineering, and a knowledge representation language as developed in AI (Steele, Rory and Primer, 2002).

The user interface for the *Tallis* composer is shown in figure 3.

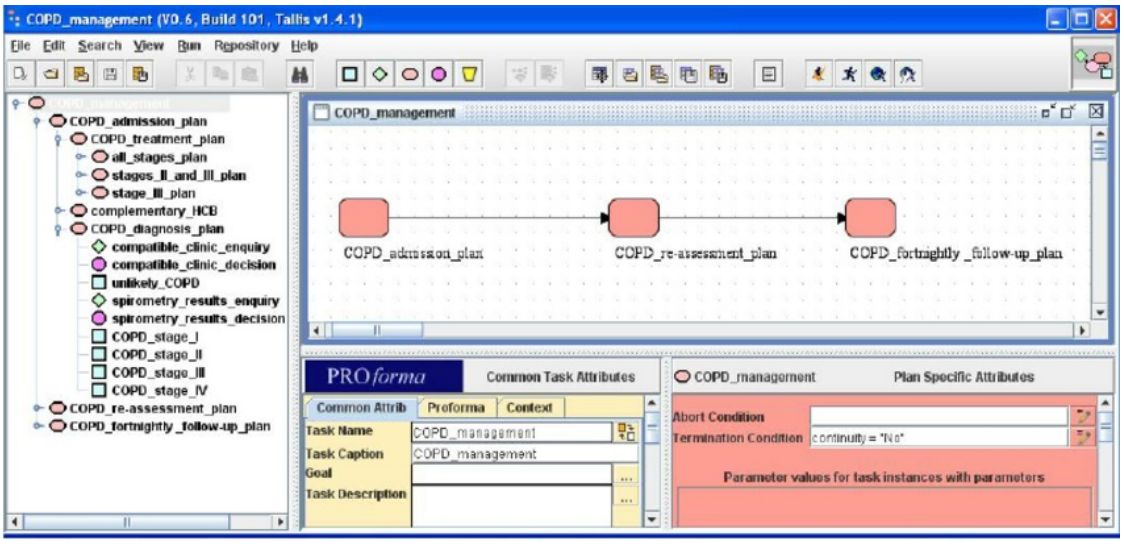


Figure 3.: *Tallis* Composer Interface (extracted from (Lozano et al., 2009)).

Process-descriptions are displayed in *Tallis* both in a network view and in a tree view:

- The tree view displays the process-descriptions hierarchical structure;
- The network view displays task ordering according to scheduling constraints;

The hierarchy of a process-description is based on plans: each plan defines a new level in the hierarchy.

- All the plans can be viewed and their contents at a glance in the tree view;
- The network view is more suited for viewing the contents of one plan at a time;

Task properties can be viewed or edited in the Task Properties window by clicking on the task in the tree or in the network area.

More details can be found in the user-guide found in their website (Oettinger, 2005).

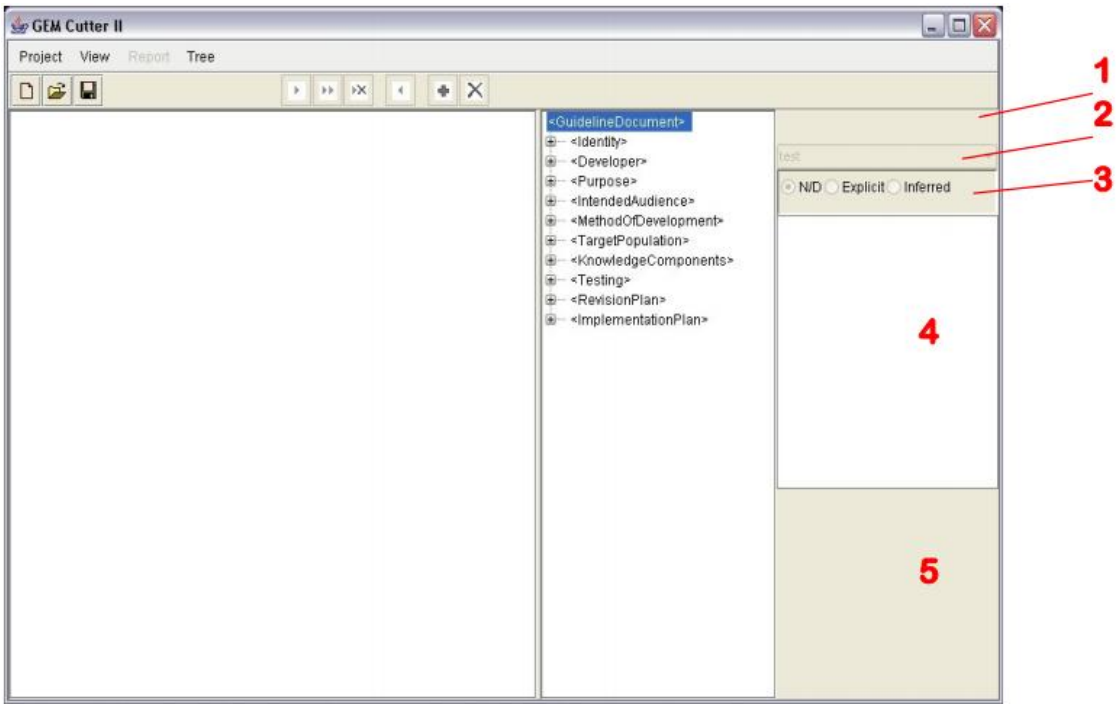
2.4 GEM Cutter

GEM Cutter is an XML editor that facilitates the markup of clinical practice guideline text, and thereby supports the conversion of a guideline document into the *GEM* format and publication in a cross-platform manner (Shiffman et al., 2001).

The Guideline Elements Model (*GEM*) is an XML- based guideline document model that can store and organize the heterogeneous information contained in practice guidelines. It is intended to facilitate translation of natural language guideline documents into a format that can be processed by computers.

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. Use of XML facilitates computer processing of the guideline information (Karras et al., 2000).

The user interface for the *GEM Cutter* is shown in figure 4.



1 - Element Name; 2 - Action Type; 3 - Element Source; 4 - Element Text; 5 - Element Definitions.

Figure 4.: *GEM Cutter* Interface (extracted from (Michel and Shiffman, 2009)).

The *GEM Cutter* main window comprises three panels, a menu bar, and button bar. Guideline text is loaded into the leftmost panel. The middle panel contains an expandable tree view of the *GEM* hierarchy. The rightmost panel has 5 areas as shown in the figure:

1. Element Name - Contains the name of the Element that is currently selected in the tree view;

2. Action Type – Drop-down list of action types;

3. Element Source - Displays the source of the Element that is currently selected in the tree view. The Source value is derived from the manner in which the data was supplied to the *GEM* document;

4. Element Text - Contains the complete text of the element that is currently selected in the tree view. Text can be input directly or edited in this window;

5. Element Definitions - Contains the definition of the element that is currently selected in the tree view;

Next is shown the user interface of *GEM Cutter* after opening a Guideline in figure 5.

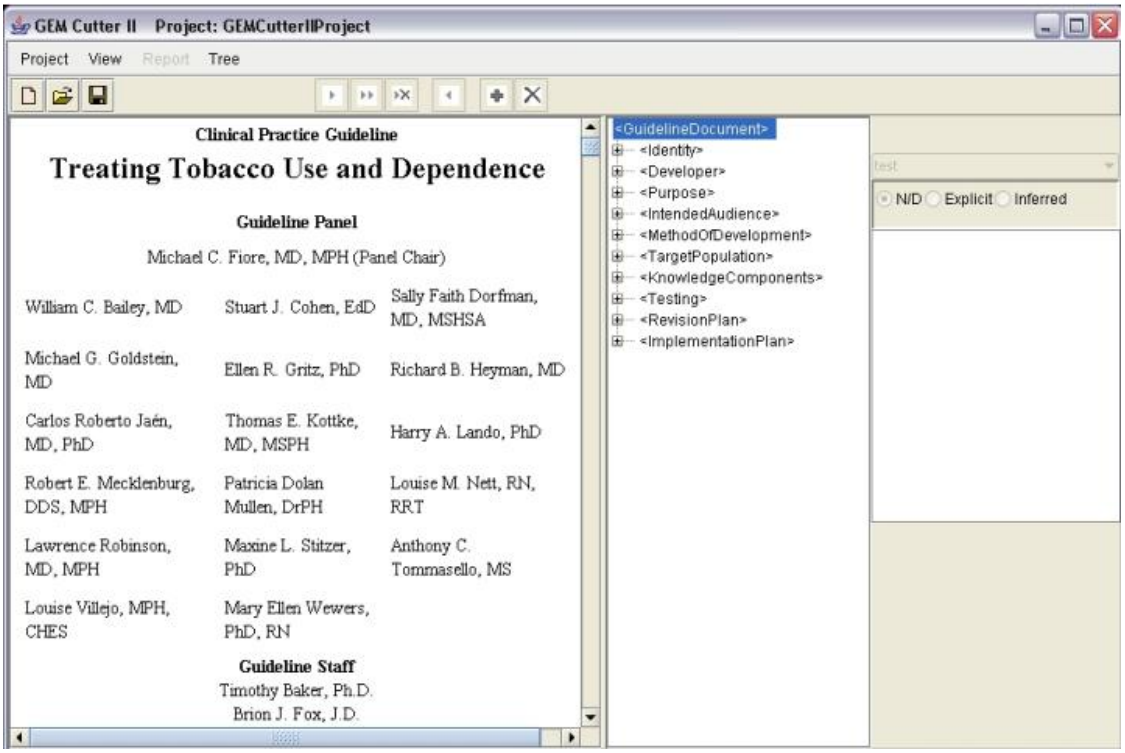


Figure 5.: Treating Tobacco Use and Dependence Guideline Example in *GEM Cutter* (extracted from (Michel and Shiffman, 2009)).

More details can be found in the user-guide found in their website (Polvani et al., 2000).

2.5 ASBRU VIEW

Asbru is a complex language which cannot be understood by physicians with no or little training in formal methods. *Asbru View* is a tool to make *Asbru* accessible to physicians, and to give any user an overview over a plan hierarchy. *Asbru View* is based on visual metaphors to make the underlying concepts easier to grasp. This was done because not only is the notation foreign to physicians, but also the underlying concepts. In other words *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).

Asbru View consists of two main views: Topological View (TopoView) and Temporal View (TempView):

- The Topological View mainly displays the relationships between plans, without a precise time scale. The basic metaphor in this view is the running track;
- The Temporal View concentrates 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. For this purpose, the Temporal View is used. It consists of a display that represents each plan with a graphical object whose features change with the values they depict;

The user interface for the *Asbru View* is shown in figure 6.

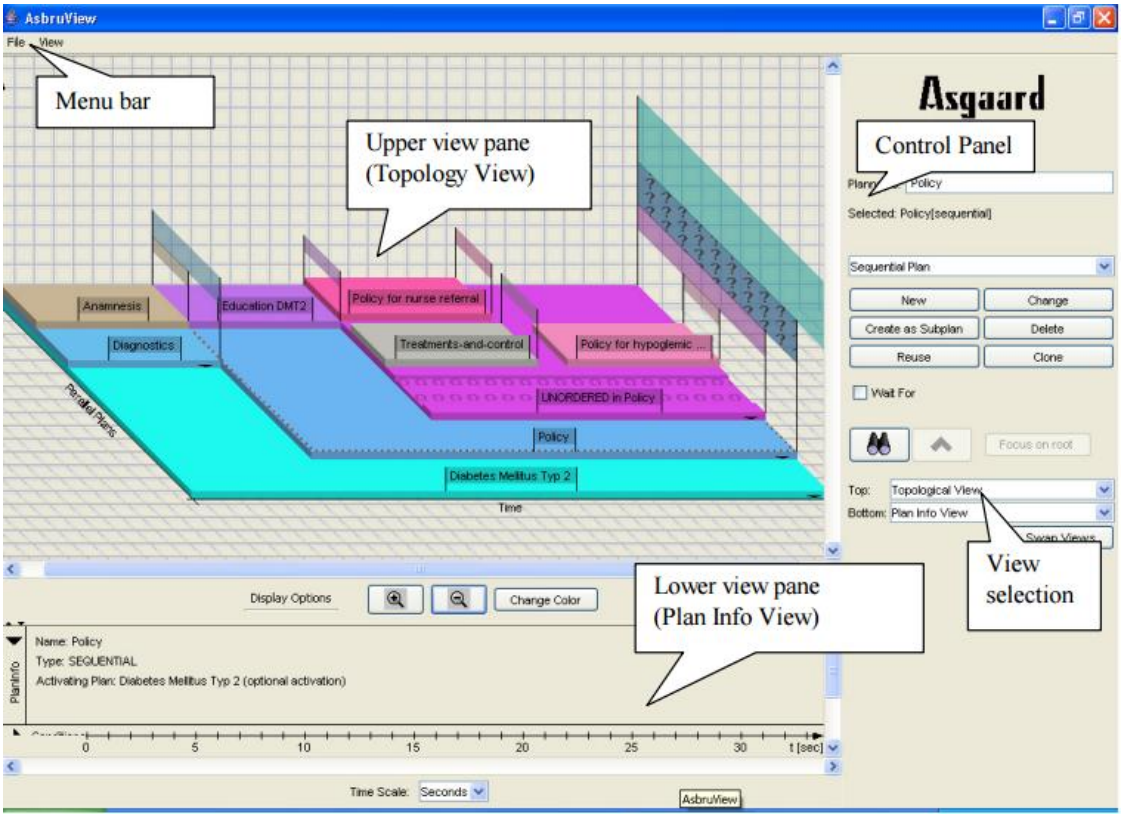


Figure 6.: *Asbru View* Interface (extracted from (Huber, 2005)).

Asbru View main window consists of four panels, a Menu Bar, a Control Panel, an Upper view pane and a Lower view pane. Each of this panels consist of a set of controls explained briefly in the following list:

- **Menu Bar** - Global commands such as New, Quit, etc.;

- **Control Panel** - Controls to create and modify plans, to focus on plans and to select the views shown on the right side of the control panel;
- **Upper view pane** - The upper view on the plans. The screen-shot shows the Topology View;
- **Lower view pane** - Lower view. Can be hidden;

2.6 DISCUSSION

Now that the highlighted platforms were properly studied, a small discussion will be made comparing the key aspects and main deficiencies of each application and determine what features could be included in the new model.

As such a set of comparative features were selected as a means to analyse and evaluate these platforms. These features are some of the most important when analysing the user-experience (Bott, 2014). This set consists of:

1. **Graphical Guidelines View** - Graphical representation (tree, node-link, network diagrams) of parts of or a full CIG workflow. The arrangement of the representations within a drawing helps the user user to understand the workflow, identify relevant points of the guideline, and manipulate knowledge elements;
2. **Organization** - In all platforms organization is a must for the understanding of the platform. Organization is related with how easy the tool is to understand, determined by its structure and the way in which its functionalities are made available, whether they are placed correctly, under the right menu. This feature allows the user a better understanding of the application structure, optimizing its use;
3. **Simplicity** - Another feature that is crucial in the creation of any platform. This feature conveys the ease of access to the functionalities of the tool. Complexity leads to confusion in the use of the platform, leading many users to abandon it;
4. **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;
5. **Drag-and-Drop** - The ability to drag-and-drop instances in the Graphical Guideline View and filter the workflow of the CIG with the help of graphical type links;
6. **Web/Local Repository** - The possibility to save or load CIGs either locally or in a cloud repository;

Next a comparison table of the platforms studied earlier will be displayed, using the comparative features explained. Important to say that this comparison doesn't include plug-ins that can be applied to these platforms in order to add new features.

Feature/Platform	*Protégé Desktop*	*SAGE Work-bench*	*Tallis*	*GEM Cutter*	*Asbru View*
Graphical Guidelines View	x	x	x		x
Organization	x	x	x	x	x
Simplicity			x	x	x
Automation					
Drag-and-Drop					
Local Repository	x	x	x	x	x
Web Repository	x	x			

Table 1.: Comparison Table of Managing Tools for CIGs.

As we can see in this comparison table (Table 1), we can conclude that some features could be included in *CompGuide Editor* to improve the user-experience.

Despite all the tools displaying the main feature (the modelling of ontologies), they begin to exhibit characteristics of past applications, focusing only on the proper functioning and less on appearance or ease of management.

One of the features that new platforms should present is the capacity to deploy Automation data, wherein the user implements only the data relevant to the instance, leaving the less important details to the responsibility of the system. Users also want all data managed by the platform to be seen in a simple and organized graphical format (a good layout encourages understanding of managed data, especially for users with little informatics knowledge). The same is true for displayed menus.

Let's analyze the *Protégé Desktop* and *SAGE Workshop* cases. Despite having many useful features available, the amount of menus they display is significant, which makes the user lose a lot of time in understanding the different functionalities. As for the Drag-and-Drop feature, it is an important element these days, enabling the management of a CIG visual layout and more comprehensive user form.

Another important feature is the ability to import or export CIGs stored locally or in a cloud. We must give relevance to this feature, as most of the information is currently held in clouds, giving the possibility for the user to access all this data anywhere, any time.

With this study completed, we end here the state of the art chapter, giving relevance to some important aspects to add in the creation of *CompGuide Editor* platform.

3

CLINICAL PROTOCOLS IN COMPGUIDE

In order to better understand the created plug-in and its features, it's crucial to understand first the essential points of the *CompGuide* ontology used in this project. Having that in mind, it's necessary to approach this matter by analyzing the problems that may appear when executing the CPs, according to its classes, individuals, and restrictions associated. As such, this chapter is intended to analyze a set of elements that describe the ontology domain and the features and actors of the system.

Sub-chapter Web Ontology Language gives a brief description of the OWL languages and its uses. Sub-chapter *CompGuide* Ontology explains what is a *CompGuide* ontology, what is used for, and the advantages of it's uses. Sub-chapter OWL Structure explains what this structure represents and it's uses. In sub-chapter Domain Model, it's shown and explained the domain in which the project is inserted. The next sub-chapter (System Actors) is presented with details about the users that will use both the *CompGuide* ontology and the project features. Posteriorly is presented the sub-chapter Requirements Analysis and Gathering where are shown the functional and non-functional requirements necessary to obtain the desired solution. After this, the sub-chapter Use Cases reveals a graphical diagram that allows the reader to better interpret the structure explained in this chapter.

In the last sub-chapter (Discussion and Analysis) is elaborated a discussion and analysis of all spoken themes in this chapter.

3.1 WEB ONTOLOGY LANGUAGE

Web Ontology Language (OWL) is a formal language for representing ontologies in the Semantic Web developed by W3C Web ontology Working Group. OWL was primarily designed to represent information about categories of objects and how objects are interrelated (the sort of information that is often called an ontology). OWL can also represent information about the objects themselves (the sort of information that is often thought of as data) (Horrocks et al., 2003). As OWL is supposed to be an ontology language, it had to be able to represent a useful group of ontology features. As there were already several ontology languages designed for use in the Web, OWL had to maintain as much compatibility as

possible with the existing languages (SHOE (Heflin et al., 1999), OIL (Patel-Schneider et al., 2001) and DAML+OIL (Mcguinness et al., 2002)).

The multiple influences on OWL resulted in some difficult trade-offs. Somewhat surprisingly, considerable technical work had to be performed to devise OWL in such a way that it could be shown to have various desirable features, while still retaining sufficient compatibility with its roots (McGuinness et al., 2004).

An OWL ontology describes a domain in terms of classes, properties and individuals and may include rich descriptions of the characteristics of those objects. OWL ontologies can be used to describe the properties of Web resources. Where earlier representation languages have been used to develop tools and ontologies for specific user-communities in areas such as sciences, health and e-commerce, they were not necessarily designed to be compatible with the World Wide Web, or more specifically the Semantic Web, as is the case with OWL (Liu and Özsu, 2009).

OWL can express which objects belong to which classes, and what the property values are of specific individuals. Equivalence statements can be made on classes and on properties, disjointness statements can be made on classes, and equality and inequality can be asserted between individuals. OWL has the ability to provide restrictions on how properties behave that are local to a class. OWL can define classes where a particular property is restricted so that all the values for the property in instances of the class must belong to a certain class (or datatype); at least one value must come from a certain class; there must be at least certain specific values and there must be at least or at most a certain number of distinct values (Horrocks et al., 2003).

The advantages of OWL reside in the manner a system uses the information. Machines do not grasp yet human language and, occasionally, there is content that escapes their understanding. For instance, a human being may comprehend that in some situations there are words that are unquestionably related, although not being their replacements. A machine does not recognize these relationships, but semantics are essential. The advantage lies in the creation of a better management of the information and its descriptions. If the system is internal to an organization, there is no need to use OWL. However, if it is something that must be released into the world, OWL will probably be a better choice in the long term (Oliveira, Tiago and Novais, Paulo and Neves, 2013).

3.2 CompGuide Ontology

CompGuide is a CIG model developed under OWL that offers support for administrative information concerning a guideline, workflow procedures, and the definition of clinical and temporal constraints which allows an advanced reasoning and the sharing of a standard representation. However, the representation of clinical information requires an inherent

flexibility, given the variability of decision making processes that one may find in different medical domains. When compared to other models of the same type, besides having a comprehensive task network model, it introduces new temporal representations and the possibility of reusing preexisting knowledge and integrating it in a guideline (Oliveira et al., 2013a). The creation of guidelines were made using the ontology development tool *Protégé Desktop* 4.

Complex pieces of information are represented as instances of classes with various properties, and simple information is represented as property data. However, simple information that is reusable and that will probably be needed in many parts of the CP is represented in the form of specific instances of classes. The entire representation is similar to a linked list of procedures (Oliveira et al., 2013b).

Since this ontology is the startup point for the work presented here, a brief description of its characteristics will follow.

In this ontology, a CP is represented as an individual of the class *ClinicalPracticeGuideline*, which has a set of data properties to express administrative information and object properties to connect it to individuals of other classes. A Task Network Model (TNM) is implemented in the form of four classes of tasks:

- *Plan* - a task container. In other words a collection of tasks containing any number of other tasks, including other plans. This ensures the possibility of nesting plans and work at different levels of implementation;
- *Action* - a task performed by an health care agent, namely a clinical procedure, a clinical exam, a medication recommendation or a non-medication recommendation;
- *Question* - an inquiry task to obtain information about the state of a patient. It is also used to record the observations of the medical information, and to save the results of clinical tests. This type of task gathers all the information necessary for implementing the CP algorithm;
- *Decision* - a reasoning task about the state of a patient which implies the choice between two or more options, yielding a conclusion which is then used to update the state of the patient. The most obvious example of such a task is the clinical diagnosis;

Just like the linked lists structures, there have to be control structures to define the relative order between tasks in the workflow. A guideline has a main Plan which contains all the tasks. The individual corresponding to this Plan has, in turn, an object property that points to its first task. Then, the previous tasks always indicates those which follow. It is possible to define sequential tasks, tasks which should be executed at the same time (parallel tasks) and alternatives in the guideline workflow (alternative tasks). In order to define a synchronization point for execution paths that are generated from the situation, the task

which precedes the parallel tasks is also connected to a synchronization task by *syncTask* property. This synchronization task is the point in the execution flow where the various execution paths converge. Regarding implementation of alternative tasks, if the execution engine must select automatically a task to perform from among a set of alternatives, based on conditions of the patient's state, it should choose the task with the property *alternativeTask*. Otherwise, if it should be the health professional to select the alternative task, the property to be used should be *preferenceAlternativeTask*.

This ontology features different types of clinical constraints expressed as conditions of the patient's condition. This constraints are:

- *TriggerConditions* - conditions on patient status parameters expressed in quantitative or qualitative terms that are associated with alternative tasks and dictate their choice. An alternative task is only selected if its *TriggerConditions* are validated;
- *PreConditions* - conditions on patient status parameters expressed in quantitative or qualitative terms that define the cases where a task can be executed;
- *Outcomes* - conditions on patient status parameters expressed in quantitative or qualitative terms that define the objectives of a *Plan* or *Action*;

Temporal restrictions are also an important element of medical algorithms. Thus, *CompGuide* provides *Periodicity* and *Duration* classes. The former may be used to express from when to when a task should be executed and/or its number of repetitions. Through *Periodicity* it is also possible to define stop conditions for a cyclic task and, in the event of these stop conditions holding true, the task the guideline execution should move to, which is a stop condition task. The *Duration* indicates how long a task should last (Oliveira et al., 2014).

Figure 7 represents a small workflow showing the initial formalization of a *ClinicalPracticeGuideline*.

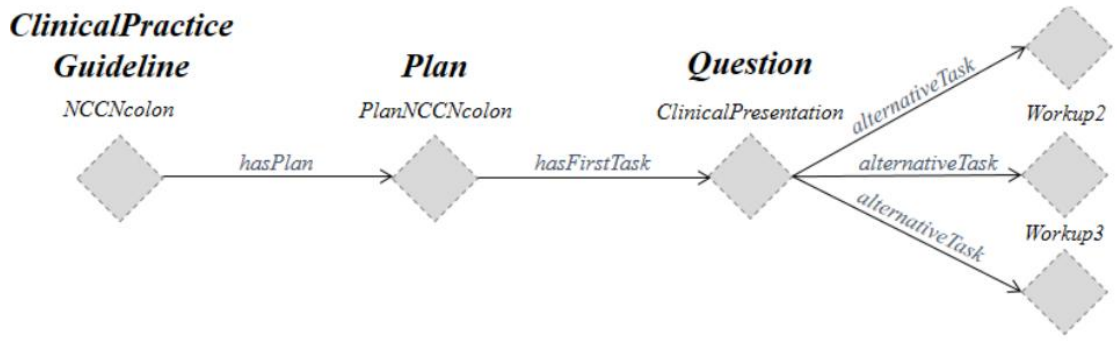


Figure 7.: Initial formalization of a *ClinicalPracticeGuideline* for Colon Cancer (extracted from (Oliveira et al., 2014)).

In comparison with other CIG models, *CompGuide* does not require any expertise in programming languages to define these conditions, unlike existing approaches. It also provides greater expressiveness in the definition of tasks and controlling relationships.

3.3 OWL STRUCTURE

The OWL (Web Ontology Language) is designed for use by applications that need to process the content of information instead of just presenting information to humans. OWL facilitates greater machine interpretability of Web content than that supported by XML, RDF, and RDF Schema (RDF-S) by providing additional vocabulary along with a formal semantics (McGuinness et al., 2004). In other words, OWL is a language for defining ontologies on the Web. The Semantic Web is a vision for the future of the Web in which information is given explicit meaning, making it easier for machines to automatically process and integrate information available on the Web. The Semantic Web will build on XML's ability to define customized tagging schemes and RDF's flexible approach to representing data. The first level above RDF required for the Semantic Web is an ontology language what can formally describe the meaning of terminology used in Web documents. If machines are expected to perform useful reasoning tasks on these documents, the language must go beyond the basic semantics of RDF Schema (Wang et al., 2004).

An OWL ontology describes a domain in terms of classes, properties and individuals and may include rich descriptions of the characteristics of those objects. OWL ontologies can be used to describe the properties of Web resources. Where earlier representation languages have been used to develop tools and ontologies for specific user-communities in areas such as sciences, health and e-commerce, they were not necessarily designed to be compatible with the World Wide Web, or more specifically the Semantic Web, as is the case with OWL. Features of OWL are a collection of expressive operators for concept description including boolean operators (intersection, union and complement), plus explicit quantifiers for properties and relationships; the ability to specify characteristics of properties, such as transitivity or domains and ranges (Bechhofer et al., 2004).

OWL can declare classes, and organize these classes in a subsumption (subclass) hierarchy. OWL classes can be specified as logical combinations (intersections, unions, or complements) of other classes, or as enumerations of specified objects. OWL can also declare properties, organize these properties into a sub-property hierarchy, and provide domains and ranges for these properties. The domains of OWL properties are OWL classes, and ranges can be either OWL classes or externally-defined datatypes such as string or integer. OWL can state that a property is transitive, symmetric, functional, or is the inverse of another property. OWL can express which objects (also called individuals) belong to which classes, and what the property values are of specific individuals. Equivalence statements

can be made on classes and on properties, disjointness statements can be made on classes, and equality and inequality can be asserted between individuals (Knublauch et al., 2004).

Also OWL provides restrictions on how properties behave that are local to a class. OWL can define classes where a particular property is restricted so that all the values for the property in instances of the class must belong to a certain class (or datatype); at least one value must come from a certain class (or datatype); there must be at least certain specific values; and there must be at least or at most a certain number of distinct values (Knublauch et al., 2004).

Figure 8 shows the modeling of class *ClinicalPracticeGuideline* of *CompGuide* ontology presented in the *Protégé Desktop* application.

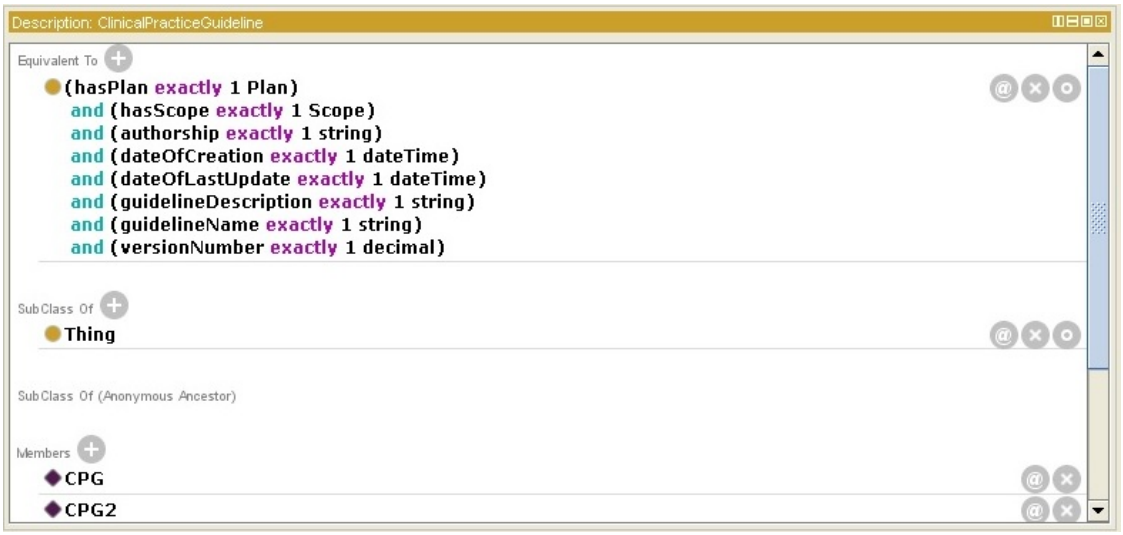


Figure 8.: Example of ClinicalPracticeGuideline Model presented in *Protégé Desktop* Application

The OWL class *ClinicalPracticeGuideline* presented in figure 8 shows that it has the following set of OWL data properties: exactly one dateOfcreation and one dateOfLastUpdate (both in dateTime datatype [temporal date plus hour]), exactly one authorship, one guidelineDescription and one guidelineName (all in string datatype), and exactly one versionNumber (in decimal datatype). Also, the OWL class *ClinicalPracticeGuideline* has two OWL object properties: exactly one OWL class *Plan* (linked by the object property hierarchy hasPlan) and exactly one OWL class *Scope* (linked by the object property hierarchy hasScope). Also, we can see that this OWL class has two OWL individuals (CPG and CPG2).

3.4 DOMAIN MODEL

Domain modeling is a representation of meaningful real-world concepts pertinent to the domain that needs to be modeled in software (Gomaa, 2001). Derived from an understanding

of system-level requirements, identifying domain entities and their relationships provides an effective basis for understanding and helps practitioners design systems for maintainability, testability, and incremental development. As a result, domain modeling envisions the solution as a set of domain objects that collaborate to fulfill system-level scenarios (Wei and Hong, 2003).

As such, a domain model is necessary to understand the concepts associated to the *CompGuide* ontology. Figure 9 shows the domain model of this ontology and respective explanation of each entity. Since the domain features a wide complexity, the following figure will only show the most important entities of *CompGuide*. The concepts have the following meaning:

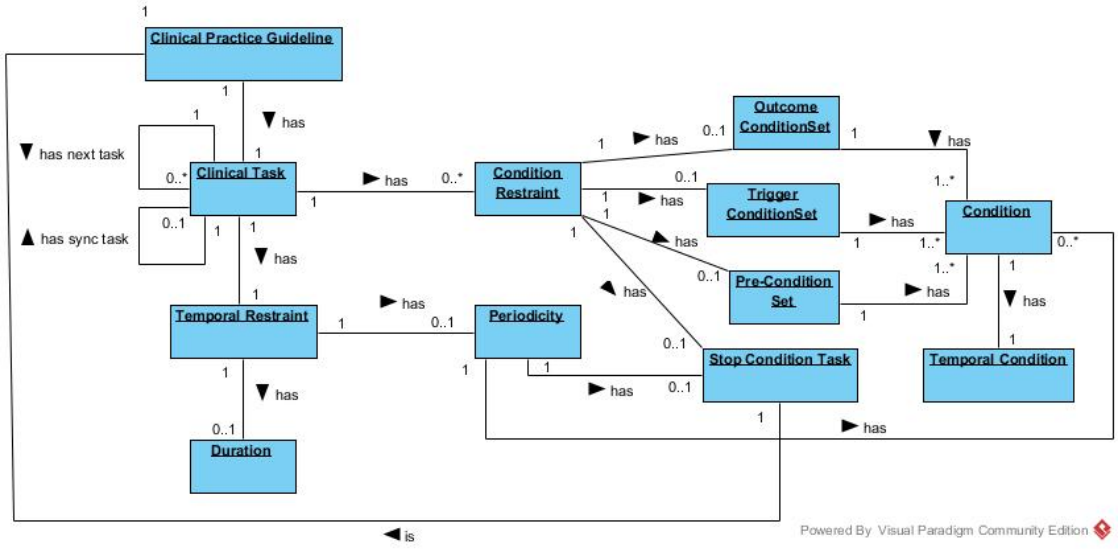


Figure 9.: *CompGuide* CIG Domain Model

- **Clinical Practice Guideline -** it is an important concept since this entity represents a CP. Each CP is started by this entity, and has several clinical tasks that must be managed by health professionals. Also this entity has a set of restrictions that let the health professionals know if the health conditions of the patient are in accordance with the restrictions that the clinical protocol has, improving the search system and thereby reducing the necessary costs;

- **Clinical Task -** this entity represents the tasks that health professionals should perform over time. clinical tasks may have clinical/time constraints that affect their execution logic. A clinical task has four sub-tasks (Plan, Action, Question, Decision) and each task is linked to another clinical task (creating a sequence of clinical tasks managed by the health professionals). Also, multiple clinical tasks can be executed in parallel, synchronized or with some preference alternative (proposed based on the

clinical constraints and the patient health condition). Each clinical task can have multiple stop conditions (based on health conditions or timed tasks), allowing a better control/performance in the execution of the task;

- **Condition Restraint -** this entity specifies a set of selected conditions which verify the health conditions of a patient and the CIG scenario. The scenarios allow a physician to synchronize the management of a patient with the corresponding parts of a guide and are normally used as information entry points;
- **Outcome ConditionSet -** this entity specifies the set of health conditions a patient will have after the clinical task is completed;
- **Trigger ConditionSet -** this entity specifies the set of health conditions a patient must have in order to apply the current clinical task;
- **Pre-ConditionSet -** this entity specifies the set of health conditions a patient must have before going through this clinical task;
- **Stop Condition Task -** this entity allows for the application of a synchronized entry, where the current clinical task is paused when the defined Stop Condition Task is being applied;
- **Temporal Restraint -** this entity specifies the set of structures related to time constraints implemented into clinical tasks. The use of this entity allows the temporal synchronization of clinical tasks to be performed in a CP;
- **Periodicity -** represents the execution cycle of a clinical task, allowing a repetition pattern to be executed. Also, stop conditions can be applied to the periodicity in order to halt this cycle and follow to the next clinical task. Periodicity is an entity that can be applied in the Plan and Action clinical tasks;
- **Duration -** this entity is used to limit the duration/time that a clinical task is expected to last. In other words, it is a time restriction that can be applied to a clinical task. Duration is an entity that can be applied in the Plan and Action clinical tasks;
- **Condition -** this entity allows to compare the pre-defined values stated in the CIG and the values examined in the patient. The results of this comparison will assist in choosing the next clinical task;
- **Temporal Condition -** this entity may be implemented within a condition, allowing to add temporal restrictions into conditions;

Other important entities which need to be described but not shown in this figure:

- **Clinical Actions -** this entity models tasks that must be performed in an Action clinical task. In other words, it allows to identify the action that must be executed in an Action clinical task. Three types of tasks are defined: medical actions, activity targeted actions (such as sending messages or obtaining patient data) and control actions (invoking structures as sub-tabs or macros that allow recursion);

- **Parameter -** this entity is used in the Question clinical tasks, and it aims to define the characteristics of the issue inspected in the patient health status. As such, it allows to identify the health parameters of the patient, which are needed to select the next viable clinical task. This is a very important entity, since it's used to identify/select the next Clinical step in the AIM system;

- **Option -** this entity is used in the Decision clinical tasks, and sets the number of options that a clinical professional can select to make a decision. This entity is needed since justified changes occur due to differences in health systems, differences in population characteristics, or due to patient preferences or professional, in case there are more than a scientifically valid option;

3.5 SYSTEM ACTORS

An actor is a behaviored classifier which specifies a role played by an external entity that interacts with the subject (for example, by exchanging signals and data), a human user of the designed system, some other system or hardware using services of the subject.

The term role is used informally as some type, group or particular facet of users that requires specific services from the subject modeled with associated use cases. When an external entity interacts with the subject, it plays the role of a specific actor. That single physical entity may play several different roles, and a specific role may be played by single or multiple different instances (Boggs and Boggs, 2002).

In this project there are two types of users: Health Professionals and Administrator. Generally speaking, the actors are the elements that will interact with the system. These actors are represented as important elements in the system operating mechanism.

3.5.1 *Administrator*

This user is responsible for managing all the files saved in the repository. The administrator (or admin) has the responsibility to maintain the latest CIG version in the repository (downloaded and used by the health professionals), check the modified versions sent by health professionals, and check the feedback from users, creating, if necessary, changes that will positively influence the use of this plug-in.

3.5.2 *Health Professionals*

These users are responsible for managing the set of CIGs, allowing the creation, modification or deletion of clinical steps or aspects in the *CompGuide* ontology file. Through the use of a simplified step-by-step process (Wizard method inserted into the plug-in), health professionals are able to approach CP medical knowledge into the CIG data.

Also, health professionals have access to graphical display features of the *CompGuide* ontology, which can be used to more easily understand the data structure of the CIG.

Since the plug-in is connected to a repository, health professionals can download the latest version of the *CompGuide* ontology, or share their modified CIG version to the *CompGuide* development team. A requirement for these functionalities is a stable connection to the Internet for them to work. The remaining functionalities are managed locally.

3.6 REQUIREMENT ANALYSIS

The process of gathering software requirements from the client, analyzing and documenting them is known as requirement analysis (Van Lamsweerde, 2009). These processes encompass those tasks that go into determining the needs or conditions to meet for a new or altered product, taking into account possibly conflicting requirements of the various stakeholders, such as beneficiaries or users. Requirement analysis is critical to the success of a development project. Requirements must be actionable, measurable, testable, related to identified business needs or opportunities, and defined to a level of detail sufficient for system design (Pohl, 2010).

The goal of requirement engineering is to develop and maintain a sophisticated and descriptive system requirements specification document. Requirements can be functional and non-functional.

3.6.1 *Functional Requirements*

Functional requirements describe the features the system has available, explained in a complete and consistent manner. In other words, functional requirements should include functions performed by specific screens, outlines of workflows performed by the system, and other business or compliance requirements the system must meet (Wiegers, 2003).

The next list represents the set of features that are available when using the *CompGuide Editor* plug-in:

- Allow the user to access all functionalities of *Protégé Desktop* 4.X application;

- Allow the user to include features of other plug-ins (restricted to application *Protégé Desktop* 4.X) in the *CompGuide Editor* plug-in, in a simple and quick way, and without the need for programming knowledge;
- Allow access to all the reasoner features of *Protégé Desktop* 4.X. *Protégé* reasoners are responsible for the verification and validation of the created individuals associated with the structure of the ontology classes and restrictions;
- Allow the user to create, edit, or delete individuals in *CompGuide* ontology through a step-by-step Wizard system while respecting the basic structures of classes and restrictions;
- Allow the user to download the latest versions of the OWL file *CompGuide* ontology, through the *CompGuide Editor* plug-in or by visiting the Git *CompGuide* repository;
- Allow the sharing of versions modified by users to *CompGuide* development team, in a safe and simple way;
- Allow the display of information (represented by a list and dynamic 2D graphic) of any part of the *CompGuide* ontology;
- In the graphical representation, allow the user to filter structures (classes, individuals, data property stored in individual, associations, etc.);
- In the graphical representation, allow the user to drag the nodes (representing classes/individuals) to the positions that the user wants, allowing to store this representation in image files (PNG, GIF or JPEG format);
- Allow the administrator to manage *CompGuide* ontology versions available in the repository (via Git);
- Allow the administrator to receive the OWL files (restricted only to the CIG *CompGuide* ontological file) associated with a short description of the changes made by the user;

3.6.2 *Non-Functional Requirements*

Non-functional requirements (or system qualities) describe terms related to the security, reliability, maintainability, scalability, usability and technologies involved in the system (Chung et al., 2012).

These requirements are persistent qualities and constraints that ensure the usability and efficacy of the entire system. Failing to meet any one of them result in systems that do not meet internal business, user, or market needs (Young, 2001).

The next list represents the set of system qualities that are available when using the *CompGuide Editor* plug-in:

- **Adaptation:** provide a plug-in for *Protégé Desktop* 4.X application that can be integrated with its features, along with the features of other plug-ins;
- **Free Access:** development and implementation of the plug-in by only using free technologies (*Protégé Desktop* 4.X, Java Development Kit 1.8 and Ant Compiler);
- **Interface Organization:** *Protégé Desktop* 4.X application should allow the re-organization of the models of the interface without the need for additional programming knowledge, in a quick and easy way;
- **Presentation:** *CompGuide Editor* plug-in provides an appealing, interactive, simple and easy handling;
- **Learning Time:** users should know how to use the system in a short time;
- **Scalability:** the system should be able to maintain the same performance (response time) when there is an increase of information (more stored individuals in ontological file) and / or when multiple simultaneous requests are requested to the cloud repository;
- **Standardization:** the Wizard presentation should follow a standard step-by-step process (making it less confusing for the user to interpret the features of the plug-in);
- **Performance:** The system should process any event in a given time;
- **Security:** all data communication features involve only the CIG files associated with the *CompGuide* ontology, making it impossible to send any other files that are not associated with the plug-in solution;
- **Allocated Space Management:** all data communication features keep a care in the file space allocation, compressing all data sent/ decompressing all data received by the plug-in, reducing the connectivity and file management requirements;
- **Portability:** the solution should run on most computer platforms (provided there is a *Protégé* 4.X version available to be installed on that platform);
- **Communication:** the solution must include the ability to share or access *CompGuide* CIGs available in the repository;
- **Ethical Requirements:** the system should not use the user information for external commercial purposes;
- **Organization:** keep all managed files by the plug-in in an organized and easy manner to interpret (easier access);

3.7 USE CASES

Use cases is a list of actions or event steps, typically defining the interactions between a role (known in the Unified Modeling Language as an actor) and a system, to achieve a goal. The actor can be a human or other external system (Jacobson et al., 2011). In other words, a use case is a series of related interactions between a user (or more generally, an actor) and a system that enables the user to achieve a goal (Bittner, 2002). Yet use case diagrams never describes how they are implemented. These can be imagined as a black box where only the input, output and the function of the black box is known.

The purposes of use case diagrams can be as follows:

- Used to gather requirements of a system;
- Used to get an outside view of a system;
- Identify external and internal factors influencing the system;
- Show the interacting among the requirements are actors;

Use case analysis is an important and valuable requirement analysis technique that has been widely used in modern software engineering, making it one of the best ways to capture functional requirements of a system. (Cockburn, 2008).

Next the different interactions between the users and the system will be shown.

3.7.1 *Use Cases Diagram*

There are five types of relationships in a use case diagram (Bittner, 2002). They are:

- Association between an actor and a use case;
- Generalization of an actor;
- Extend relationship between two use cases;
- Include relationship between two use cases;
- Generalization of a use case;

As such, figure 10 show the use cases diagram of this system.

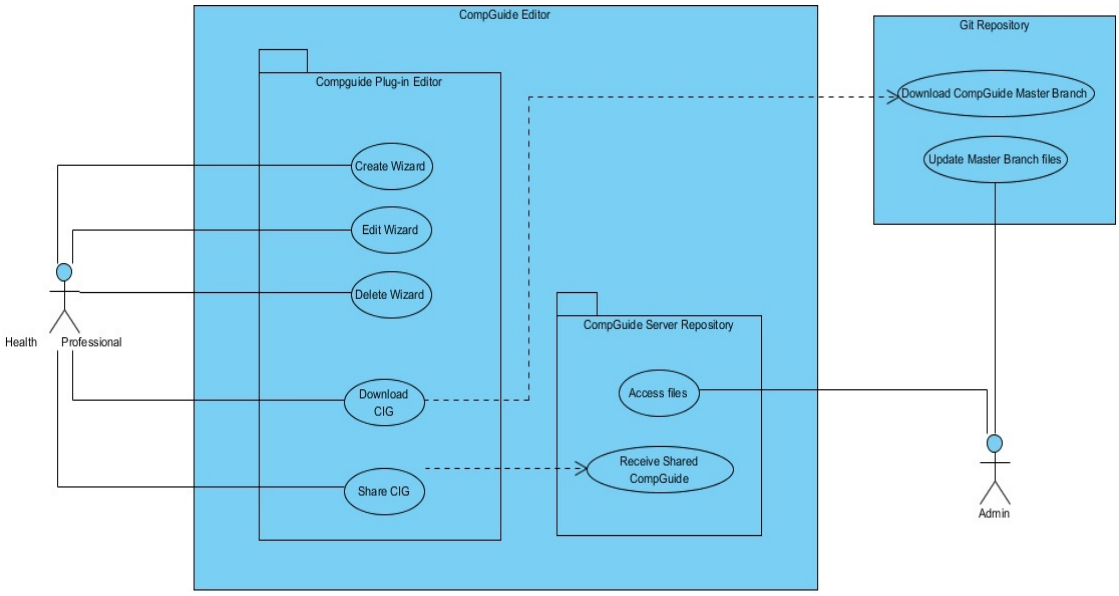


Figure 10.: *CompGuide Editor* Use Case Diagram

In this diagram, there are two systems represented: the *CompGuide Editor* system (with sub-systems *CompGuide Editor* plug-in and *CompGuide* server repository) and the Git repository.

The Git repository is the repository where the latest *CompGuide* CIG versions will be uploaded (by the admin) and ready to be used by the health professionals when downloading either through the GitHub Web repository or by using the download feature of the *CompGuide Editor* plug-in in *Protégé Desktop* 4 application.

The use of the Git tool is one of the most used platform by software development teams. GitHub is a web-based Git repository hosting service. It offers all of the distributed version control and source code management functionality of Git as well as adding its own features. It provides access control and several collaboration features such as bug tracking, feature requests, task management, and wikis for every project (Loeliger and McCullough, 2012). Git is mostly used for allocation and sharing of data to a private or public group of people. This is a tool with easy access and low complexity when it comes to data management, keeping active 24/7. Since the *CompGuide* CIG files are allocated by the Git repository master branch, only the admin may update these files, so that new CIG versions can be easily implemented in the system.

The *CompGuide* server repository is a Java Transmission Control Protocol (TCP) server. Its main objective is to receive and organize *CompGuide* CIGs altered and shared by the health professionals that use the *CompGuide Editor* plug-in. This system organizes all received files based on the Internet Protocol (IP) address and the temporal data and hour sent. Only the admin has access to these files.

CompGuide Editor plug-in was made to work in *Protégé Desktop* 4 application. In the figure, the download CIG feature is dependent on the Git repository server status and share CIG feature is dependent on the *CompGuide* server repository status. As such, if these servers are offline, a message will appear, informing the user that the connection couldn't be established. The Create Wizard, Edit Wizard and Delete Wizard are the features designed to manage the *CompGuide* CIG individuals. Since these are made locally, these features are only enabled if the *CompGuide* ontology is loaded in the *Protégé Desktop* 4 application.

3.7.2 *Description of Use Cases*

The Description of Use Cases allows to know what the end user wants to achieve by first identifying their problem. Once the problems are found, solutions can start to be looked. This process allows to outline the interactions between actor and system in solving a problem as described under a system situation. This tool can be used to find out the desired system behavior with its user.

The Description of Use Cases constitutes an high level user-and-system conversation, which aims to find out the intents or actions of actors and how the system reacts to those actor inputs. The description should be concise when deciding what to include in the events flow. This analysis is aimed to identify requirements from an end user point of view. Implementation details can, however, be modeled with Unified Modelling Language (UML) sequence diagram in form of sub-diagram of user stories.

Tables 2, 3, 4, 5, 6, 7, 8 and 9 in Annex A show the description of the use cases diagram.

3.8 DISCUSSION AND ANALYSIS

Studying the entities defined in the domain model, by investigating and analyzing the functional and non-functional requirements of the system along with the verification of the use case diagrams, it was possible to have a working basis, on which the problem in question can be identified, and the set of features that the system must provide to solve them. Through this chapter, we can see that a CP is a representation of a set of interrelated tasks, which apply restrictions/conditions related to the patient status or to temporal restrictions. The use of the CIG *CompGuide* is one of the CP forms that allow the automatic execution of a linked tasks network model, used by clinical artificial intelligent systems.

Therefore, the use of *CompGuide* ontology allows the display of information on the different clinical tasks to be performed, becoming a personal assistant to enable health professionals manage the clinical status of the patients in a faster, simpler, and more cost effective way.

The creation of *CompGuide Editor* plug-in will keep this ontology updated with the latest clinical protocol details, specifically the procedures/clinical tasks applied in current medicine. These details are implemented in a wizard / step-by-step model, eliminating the need for health professionals to have any knowledge about the *Protégé Desktop* features, and greatly reducing the amount of time required to implement changes in the CIG. The included data sharing features are an added value to easy access to the CIG files, both by health professionals and by *CompGuide* development team.

4

IMPLEMENTATION IN PROTÉGÉ DESKTOP

This chapter is intended to describe the set of planned steps taken in developing a tool able to create/edit CIGs *CompGuide*, in a quick and simple way. As such, an explanation is needed about how the idea was born, the platform used, its programming language, its advantages and limitations, the external plug-ins and Java API used, its features, and a concise presentation.

The following sub-chapters show the data structure of the application, its main features and restrictions, some diagrams and models explaining how the application features are executed and relating the data structure and the functions used, the appearance of the CIG tool plug-in.

In the end, there will be a solution analysis, discussing about the key aspects and difficulties found in the development of this tool.

4.1 TECHNOLOGIES AND TOOLS USED

The idea of creating a *Protégé Desktop* plug-in emerged from the need to create software capable of implementing all the features offered by *Protégé Desktop* application (specifically the functionality of managing the data of an ontology through the use of a graphical interface) along with the creation of new features capable of resolving the problems of a project. Another advantage that came from using this application was the ability to implement extra features from other plug-ins without the need of any programming knowledge required, in a quick, easy and simple way. In addition, *Protégé Desktop* application presents an enormous active community that is constantly updating its functionalities. All these points lead to the conclusion that the creation of a plug-in for the *Protégé Desktop* application would be an asset in solving the proposed problem.

Since *Protégé Desktop* application is a tool programmed in Java, it was taken into account the advantages of creating a plug-in in this programming language. The following list shows the set of advantages in using this platform (Gosling and Mcgilton, 1996):

- Java is currently one of the most used programming languages around the world, and still growing in enterprises, through new adoptions;
- In addition to being a programming language, it's also a development platform;
- **Portability:** It is possible to develop desktop, mobile, card, web, digital TV, and other applications. In other words, this language tends to be widely developed mainly for mobile, web and cloud computing;
- **Community:** Java user groups are very strong in the whole world. By exploring the internet, anyone has easy access to programming material, ways to participate in regional meetings, lectures and even short courses. There is also an ease to solve programming problems, thanks to the exchange of ideas in forums or chats;
- **Frameworks:** Thanks to the investment of this community and some enterprises, there is now a variety of frameworks to facilitate the work of the developer;
- **Programming Adaptability:** The Java virtual machine currently runs about 350 languages, such as Groovy dynamic language, JPython, Python, JRuby, Ruby, etc.;
- **Operational Systems:** When a Java file is compiled, it generates a new file that can be interpreted in any Java Virtual Machine (JVM). As long as the operating system has a JVM, the Java project can be run on any of these systems (Windows, GNU / Linux or Mac);
- **Evolution:** Java tends to be developed for technologies that integrate HTML5. This occurs because new Java applications have the need to focus on Web and mobile architectures;

4.2 SOFTWARE ARCHITECTURE

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[1]. As such, the selected version of the *Protégé Desktop* application was version 4, since *CompGuide* ontology was created using this release (*Protégé Desktop* 3 uses OWL API 1 while *Protégé Desktop* 4 uses OWL API 2) (Rubin et al., 2007). Because of compatibility issues, the plug-in should be created for *Protégé Desktop* 4 application. However, through some studies in the *Protégé Desktop* support forums[2], we can conclude that the compatibility issues only arise between older versions of the application (for example, an ontology created

1 http://protegewiki.stanford.edu/wiki/Protege_Plugin_Library
2 http://protege-project.136.n4.nabble.com/Protege-OWL-4-x-Support-f21363.html

in *Protégé Desktop* 4 application will be compatible with *Protégé Desktop* 5, but incompatible with *Protégé Desktop* 3).

In order to begin the process of creating a plug-in for *Protégé Desktop* 4 application, the study of some tutorials[3] and the execution of some tests were needed. Since the programming language would be Java, the IDE used was the Eclipse application. It was also necessary to create a script to run in the Ant platform. This script allows the compilation and creation of the plug-in in JAR format (used by *Protégé Desktop* 4 application). As such, the compiled plug-in is placed inside the *Protégé Desktop* plug-ins folder. After the generation of this file, its features are ready to be used, by simply restarting the application.

The organization of the Java APIs used in this system are shown in the next figure (11).

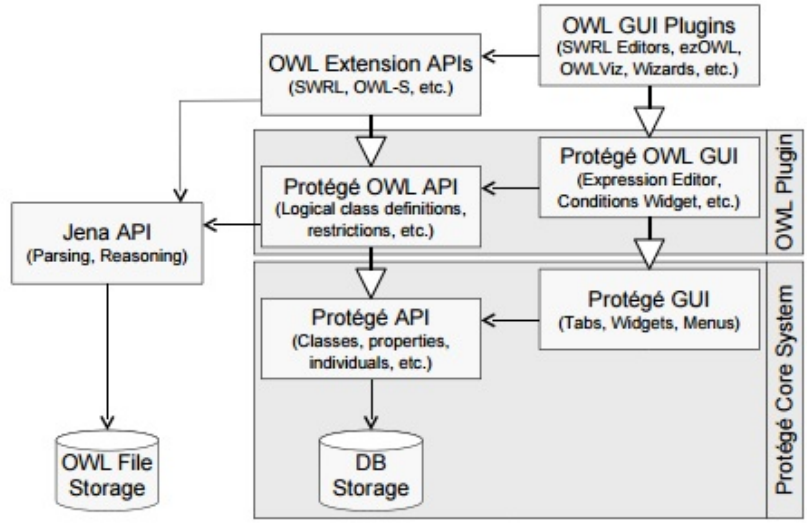


Figure 11.: The OWL plug-in in the *Protégé Desktop* core system (extracted from (Knublauch et al., 2004))

The features of the most important system components are as follows:

- The *Protégé* API (or OWL API) is a Java API and reference implementation for creating, manipulating and serialising OWL ontologies (Horridge and Bechhofer, 2009);
- The *Protégé*-OWL API is an open-source Java library for the OWL and RDF. The API provides classes and methods to load and save OWL files, to query and manipulate OWL data models, and to perform reasoning based on description logic engines. Furthermore, the API is optimized for the implementation of graphical user interfaces. The API is designed to be used in two contexts: the development of components that are executed inside of the *Protégé*-OWL editor's user interface and the develop-

3 http://protegewiki.stanford.edu/wiki/Protege4DevDocs

ment of standalone applications (e.g., Swing applications, Servlets, or Eclipse plug-ins) (Knublauch et al., 2005);

- The plug-in APIs are used and presented in the graphical interface used in the *Protégé*-OWL API. *Protégé Desktop* can implement several plug-ins running at the same time, making it possible to access several functionalities in a single window. Also, *Protégé*-OWL API has features that facilitate the window organization and management of how the different plug-ins are displayed to the user. *CompGuide Editor* plug-in is inserted in this category. Most of *Protégé Desktop* application plug-ins can be downloaded through the *Protégé* Wiki url link[4];

Although there are other APIs used in *Protégé Desktop* application, the list presented shows only the required APIs used in the creation of the functionalities of the *CompGuide Editor* plug-in. In other words, figure 11 shows that for the proper function of the created features, it's mandatory the use of these APIs (OWL API and *Protege*-OWL API). *CompGuide Editor* plug-in also has dependencies features of another *Protégé Desktop* plug-in (plug-in OntoGraf), which is automatically installed when using the *Protégé Desktop* 4.1+ application. However, this plug-in is compatible and can be installed on any version of the *Protégé Desktop* 4.X (4.0-4.3). Again, it is important to refer that to install a plug-in in the application, the user must place the respective jar file in the application board plug-ins folder and reset the application.

OntoGraf plug-in gives support for interactively navigating the relationships of OWL ontologies. Various layouts are supported for automatically organizing the structure of the ontology. Different relationships are supported: subclass, individual, domain/range object properties, and equivalence. Relationships and node types can be filtered to help create the desired view (da Silva and Freitas, 2011). Figure 12 shows a small demonstration of the OntoGraf View.

4 http://protegewiki.stanford.edu/wiki/Protege_Plugin_Library

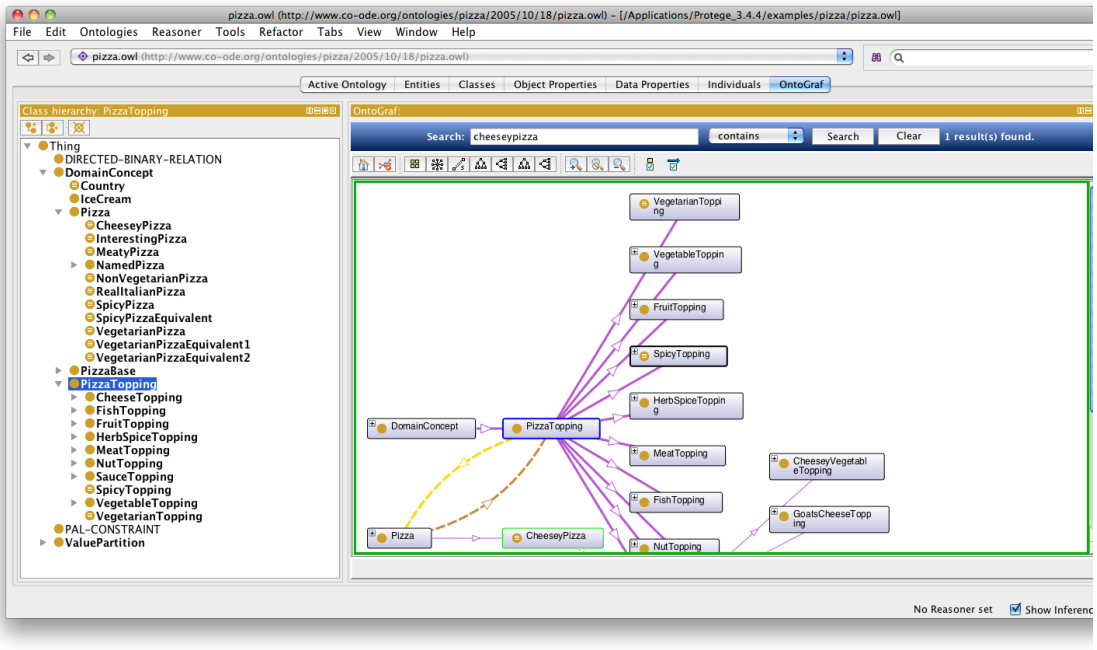


Figure 12.: OntoGraf plug-in in *Protégé Desktop* application(extracted from OntoGraf Wiki url[5]).

Later in the plug-in interface features, all *CompGuide Editor* plug-in features will be properly shown and explained.

So far we have explained the methodologies used in solving the ontological data management problem. Now we will make the point analysis on the data management via cloud (one of the other objectives in the creation of this plug-in). One of the ideas that came to solve this problem would be the sharing data by using one of the most popular platforms in group development: the Git platform.

Git is an open source project that is by far, the most widely used modern version control system in the world today. Git is a mature, actively maintained project that is an example of a Distributed Version Control System(DVCS). Rather than have only one single place for the full version history of the software as is common in once-popular version control systems like CVS or Subversion (also known as SVN), in Git, every developer's working copy of the code is also a repository that can contain the full history of all changes. In addition to being distributed, Git has been designed with performance, security and flexibility in mind (Gandrud, 2013).

Git proposes a solution in the contribution of solving an open source project. Let's say, for example, if you wanted to contribute to an open source project you had to manually download the project's source code, make your changes locally, create a list of changes called a patch and then e-mail the patch to the project's maintainer. The maintainer would

5 http://protegewiki.stanford.edu/wiki/OntoGraf

then have to evaluate this patch, possibly sent by a total stranger, and decide whether to merge the changes. This example shows there may occur some problems in the development (in group) of any open source project between unknown members. That's where Git enters. Like Subversion, the centralized workflow uses a central repository to serve as the single point-of-entry for all changes to the project. Instead of trunk, the default development branch is called master and all changes are committed into this branch. This workflow doesn't require any other branches besides master (Lee et al., 2013). Developers start by cloning the central repository. In their own local copies of the project, they edit files and commit changes as they would with SVN; however, these new commits are stored locally—they're completely isolated from the central repository. This lets developers defer synchronizing upstream until they are at a convenient break point. To publish changes to the official project, developers push their local master branch to the central repository (Loeliger, 2006). This is the equivalent of SVN commit, except that it adds all of the local commits that aren't already in the central master branch. If a developer's local commits diverge from the central repository, Git will refuse to push their changes because this would overwrite official commits, giving the user a chance to manually resolve the conflicts (Loeliger, 2006).

The use of the Git platform would provide advantages in data management via cloud repository. Through some investigation, it was found that these features could be implemented in the Java language by implementing the features of JGit. JGit is a pure Java library implementing the Git version control system. As such, it allows the use of most of Git commands in the creation of new Java tools[6].

However, some difficulties were found in implementing APIs not supported by *Protégé Desktop* application. In order to use any API not dependent on the *Protégé Desktop* application, it would be required to import not only the source API code that the developer wanted to use, but also all the APIs source code that were dependent of that API. Therefore, it was necessary to change the programming perspective and use only APIs required by the *Protégé Desktop* application. For example, instead of copying the repository to the local machine by using the Git-clone command (available in JGit library), the system would download the http link associated to the *CompGuide* repository master branch. For sharing the ontology modified files (altered by health professionals), since the commit commands could not be implemented, a Java TCP Socket server was created to receive and organize all this data. This system can be exemplified with the image shown in figure 13. In the next sub-chapters all these features will be properly explained.

6 https://eclipse.org/jgit/

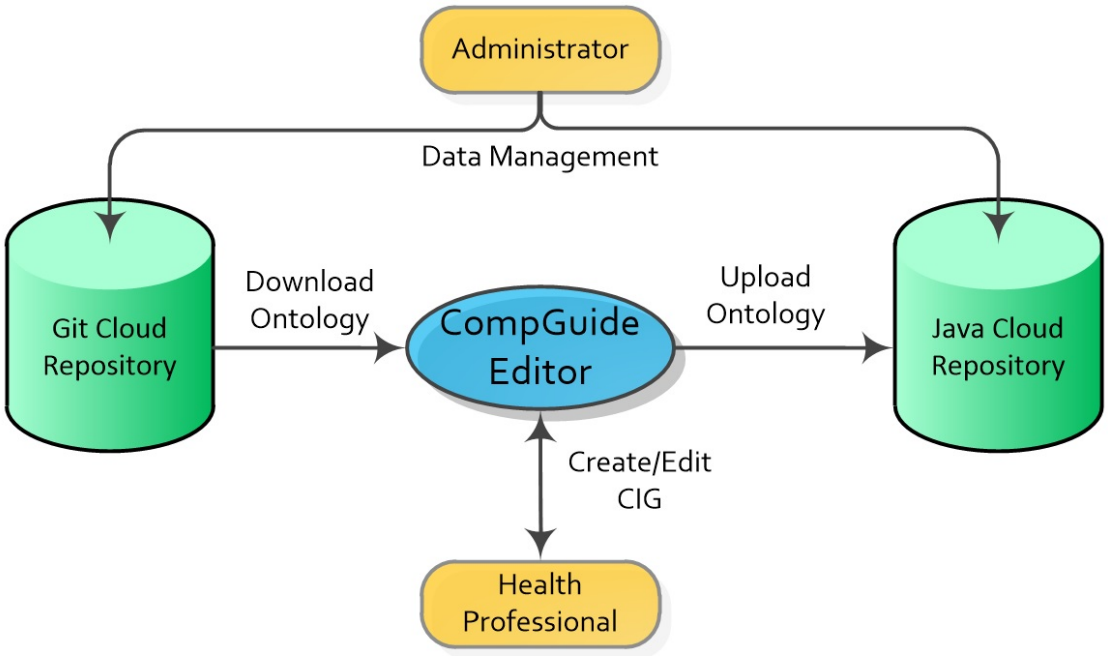


Figure 13.: *CompGuide* System.

4.3 CLASS DIAGRAMS

The class diagram is a static diagram that is used to visualize, describe and document different aspects of a system. Also, it is fundamental in the constructing of executable code for the software application. This diagram provides an overview of the target system by describing the objects and classes inside the system and the relationships between them (Purchase et al.). The class diagrams are widely used in the modeling of object oriented systems because they are the only UML diagrams which can be mapped directly with object oriented languages. It is also known as a structural diagram (Purchase et al., 2001).

Figures 14, 15, 16, 17 and 18 describe the OWL classes data structure of the *CompGuide* ontology. All the objects shown in this diagram belong to the *CompGuide* ontology data structures. In order to better interpret these objects, the class diagram was divided into five figures, representing the different types of clinical tasks and clinical actions in the OWL ontology. Since ClinicalPracticeGuideline (CPG) OWL class presents the hasPlan relation (and can be described as the starter of the CP), Plan clinical task is shown in the CPG class Diagram figure. An explanation of each figure will be made by detailing the most important features of the structure.

The different classes illustrated allow the management of information related to the CP of the *CompGuide* ontology, including its execution, completed tasks, the observations of the patient's condition, and time constraints. These classes are essential for the system to

function properly and calculate the tasks that the health professional should perform or complete.

Note that this diagram focuses on the structure of the OWL classes presented in the *CompGuide* ontology. As such, since this ontology has a high level of complexity, a simplified approach was adopted to make it easier to study the diagram.

Figure 14 shows the OWL classes ClinicalPracticeGuideline and Plan. ClinicalPracticeGuideline is the starter class of a CP, and has associated to it a Scope and a Plan class. The Scope manages information regarding the ClinicalPracticeGuideline (for example clinicalSpeciality, guidelineCategory, or condition restraints of the CP). The Plan class is a sub-class of the ClinicalTask class (as are the Action, Decision and Procedure clinical tasks), and manages the set of clinical tasks to be executed in the CP (at least one task must be implemented into the hasFirstTask relation). The Duration class manages the total time a CP needs to be concluded, while the Periodicity class manages the time interval between the execution of a particular clinical task. The Condition class manages the restrictions/conditions of any task, and the Temporal Restriction class adds the possibility of applying temporal restrictions. Also, there is the possibility of implementing Stop Conditions into the Periodicity cycle (by using the Condition OWL class).

The relationship between various tasks (hasFirstTask, syncTask, nextTask, etc.) is made just as a structure of a simple linked list (item navigation is forward only) used in programming languages. A linked-list is a sequence of data structures which are connected together via links. In other words, it is a sequence of links containing items. Each link contains a connection to another link (Timnat et al., 2012).

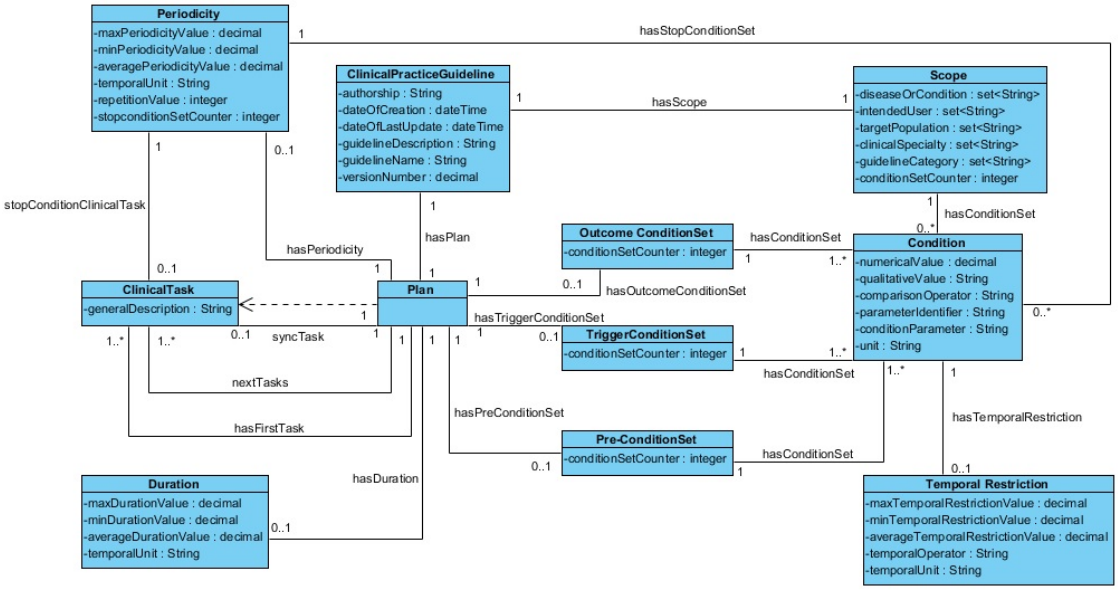


Figure 14.: Class Diagram - CPG and Plan Clinical Task.

Figure 15 shows the OWL class Action. It manages the clinical actions to be executed in the CP (like Exams, Procedures, Formulas, Medication and Non-Medication Recommendations, shown in figure 16). Most of this structure is similar to the Plan class structure. The big differences are the removal of the hasFirstTask relation (exists only in the Plan class) and the insertion of the ClinicalActionType individuals (in hasClinicalActionType relation), where the clinical actions are inserted (at least one clinical action must be implemented into the Action).

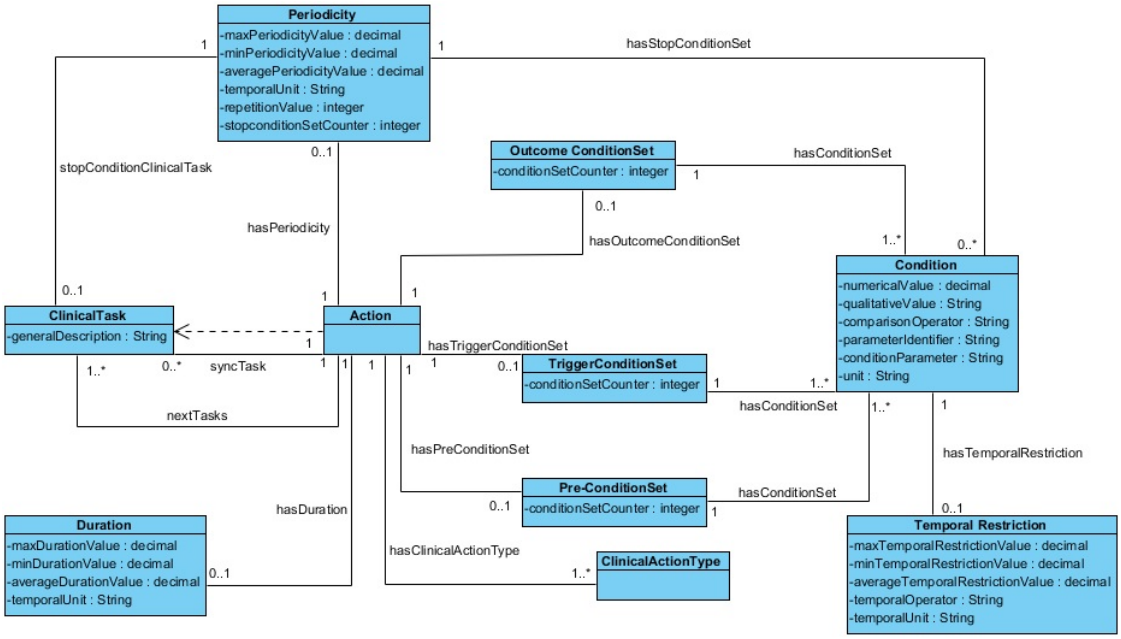


Figure 15.: Class Diagram - Action Clinical Task.

Figure 16 shows all the sub-classes of the ClinicalActionType class already mentioned. Parameter class is used in the Formula and Question classes to manage a health feature required for the successful execution of the CP.

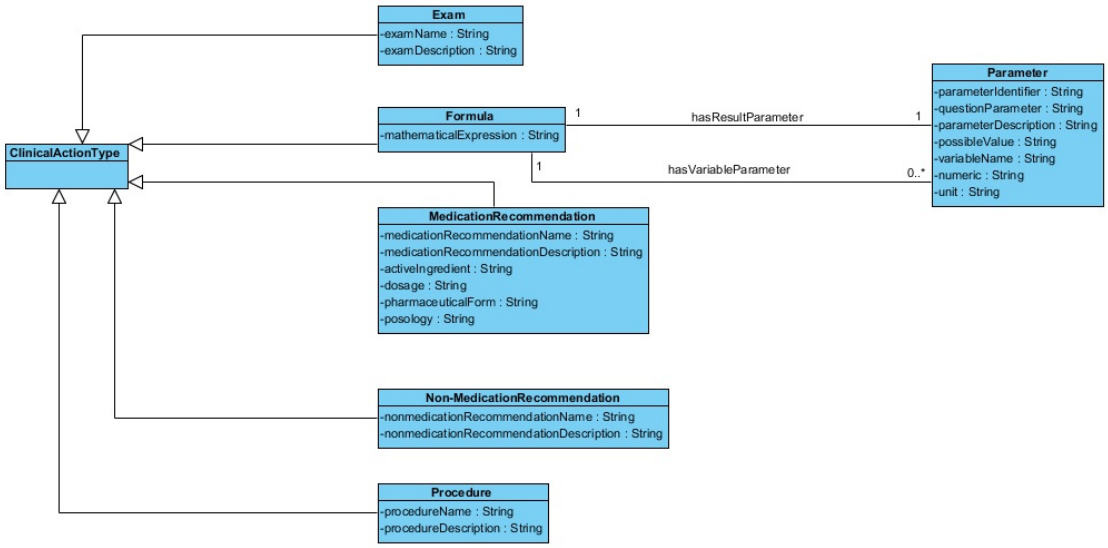


Figure 16.: Class Diagram - Clinical Action Types.

Figure 17 shows the OWL class Question. This class manages the set of health features required (by using a set of Parameters) to the CP. At least one Parameter must be implemented in the Question class (through the use of hasParameter relation).

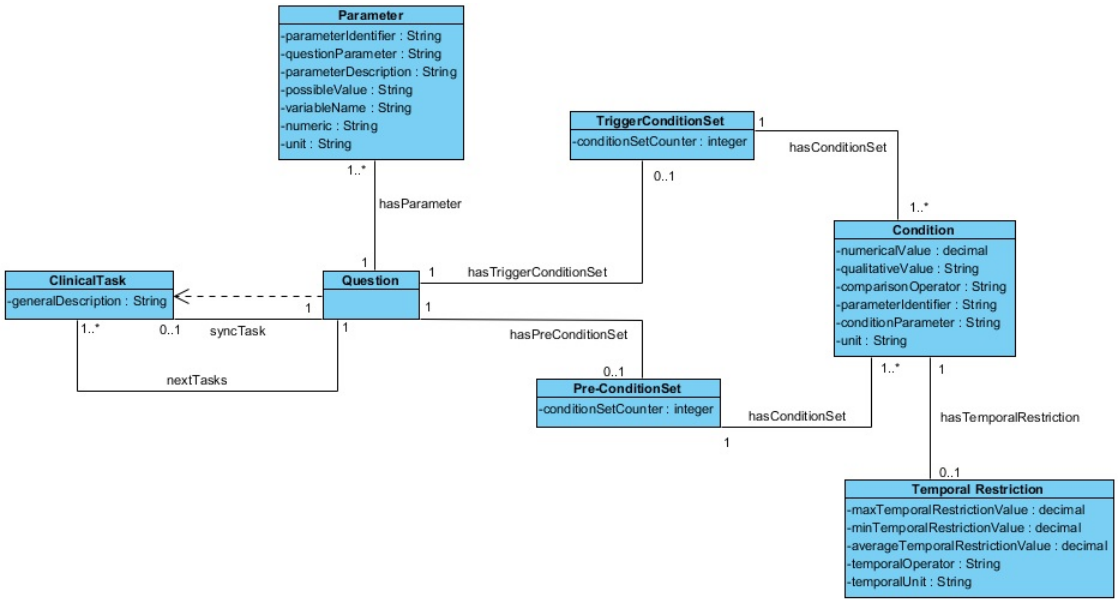


Figure 17.: Class Diagram - Question Clinical Task.

Figure 18 shows the OWL class Decision. This class manages the set of clinical Options that can be selected to execute a CP. As such, two or more Options must be available for this Decision clinical task (hasOption relation) where each clinical Option is managed by the OWL class Option. Also, Condition restraints can be implemented into each Option.

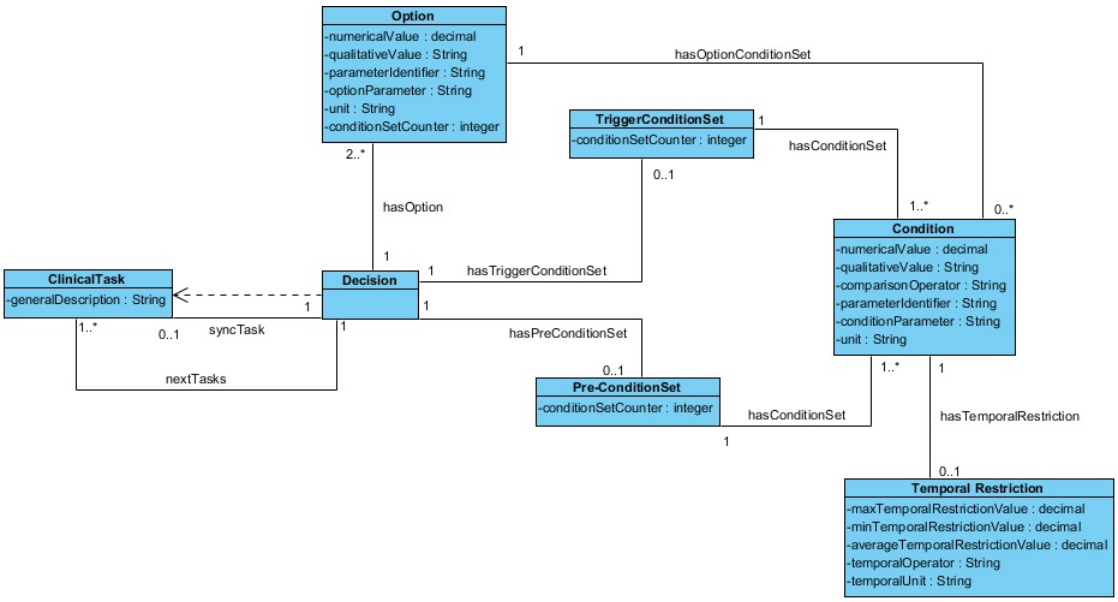


Figure 18.: Class Diagram - Decision Clinical Task.

4.4 SEQUENCE DIAGRAMS

The sequence diagram is used primarily to show the interactions between objects in the sequential order that those interactions occur. In other words, it is a form of interaction diagram which shows objects as lifelines running down the page, with their interactions over time represented as messages drawn as arrows from the source lifeline to the target lifeline. Sequence diagrams are good at showing which objects communicate with which other objects and what messages trigger those communications (Douglass, 2003).

These diagrams are useful in documenting how a future system should behave. During the design phase, architects and developers can use the diagram to force out the object interactions in the system, thus demonstrating overall system design. One of the primary uses of sequence diagrams is in the transition from requirements expressed as use cases to the next and more formal level of refinement. Another use is to document how objects in an existing system currently interact (Briand et al., 2003).

All sequence diagrams are presented in figures 39 - 53 in Annex B. Figure 19 shows the sequence diagram representation of the CPG edition feature.

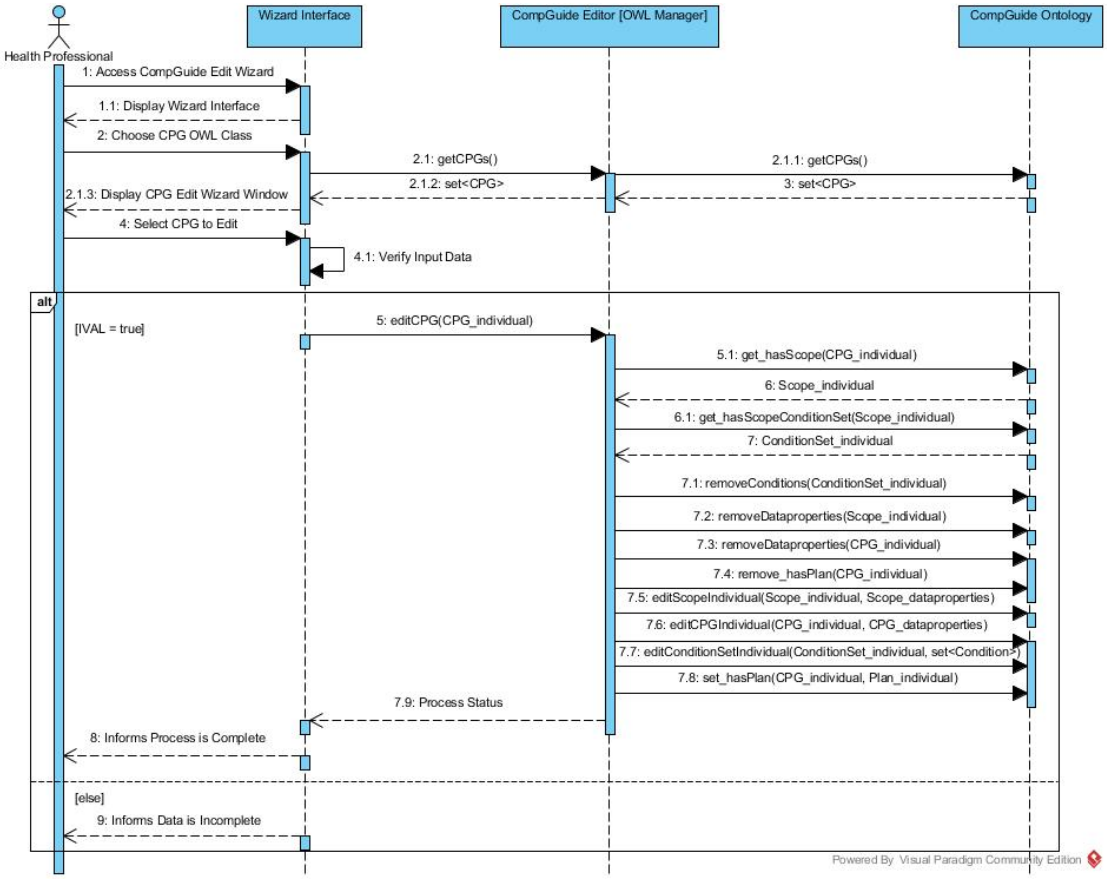


Figure 19.: Edit CPG Sequence Diagram.

These diagrams can be categorized into four groups (creation, edition, deletion and share). These group features can be explained in the following way:

- Creation features follow the set of steps: after validation of health professional-input data, the system checks the fields filled; creates the individuals related to the fields filled (an individual of periodicity class will only be created if the fields for the periodicity (in the Wizard) have been inserted by the health professional); individual data properties are created and filled with data from the Wizard; in the end, these individuals are associated (data objects);

- Edition features follow the set of steps: after the health professional select the individual do edit, the system gets its respective data properties and its data objects; fills all the Wizard fields with the information obtained from the individuals; health professional changes the necessary fields; when concluding this process, the system verifies the fields filled and determines the individuals it has to create; if the individuals already exist, the data properties of the individual are removed and new data properties

are inserted, else new individuals are created with the necessary data properties. If there are individuals which shouldn't exist (based on the filled Wizard fields), these individuals are deleted;

- In delete features, after the health professional selects the individuals to delete, a loop is executed, deleting the respective individuals and it's data objects associated. For example, if the CPG creation feature handles the creation of CPG and Scope individuals, the CPG delete feature will remove the CPG and Scope individuals. All other delete features follow the same example;

- Download and share ontology features are dependent of the status of the connection between the system and the respective cloud repositories. Since it is intended that the amount of data to send / receive is as low as possible, all data managed in these features are compressed. When concluding the download, the system handles the decompression of the respective files;

4.5 PLUG-IN INTERFACE

This sub-chapter presents the plug-in interface and clarifies the features implemented.

Figure 20 shows the initial interface when opening the *Protégé Desktop* application. This interface displays a set of menus at the top of the window, along with a set of Tabs. Through these Tabs, the user has the possibility of positioning and managing the set of plug-ins (and corresponding functionalities). Another important element shown in this figure is the ontology IRI. Ontologies and their elements are identified using Internationalized Resource Identifiers (IRIs). Each ontology may have an ontology IRI, which is used to identify an ontology. If an ontology has an ontology IRI, the ontology may additionally have a version IRI, which is used to identify the version of the ontology. The version IRI may be, but need not be, equal to the ontology IRI. An ontology without an ontology IRI must not contain a version IRI (Allen and Unicode Consortium., 2007).

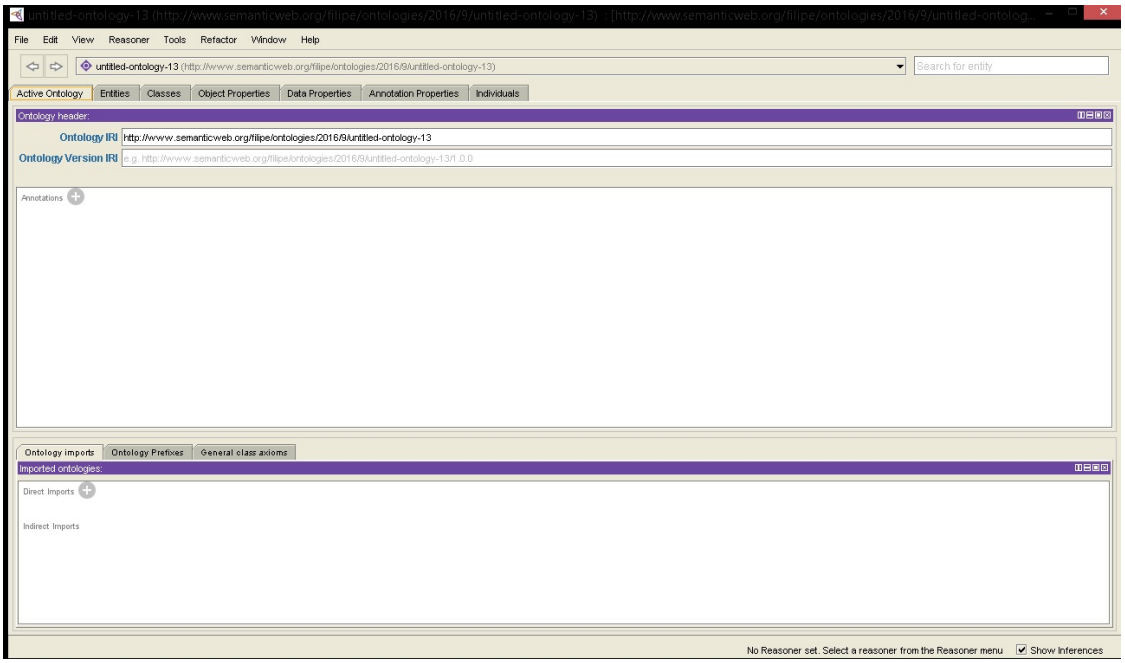


Figure 20.: Home Interface of *Protégé Desktop* application

In order to enable the *CompGuide Editor* plug-in Tab, the health professional must access Window -> Tabs -> CGuide Editor in the top menu, just like is shown in figure 21.

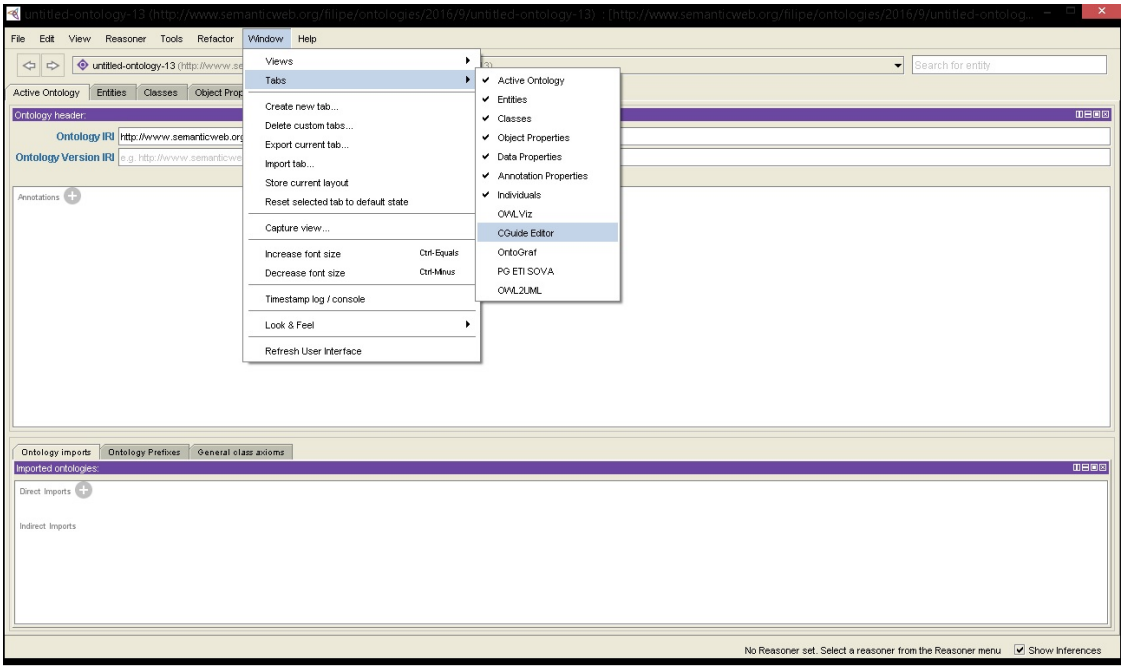


Figure 21.: Enabling *CompGuide Editor* plug-in in *Protégé Desktop* application

Figure 22 shows the new *CompGuide Editor* Tab, along with its features. This Tab consists of three Views: OntoGraf View (allows the graphical representation of the ontology), individuals by type View (shows all the individuals (sorted by the OWL classes) saved in the ontology.), and *CompGuide* Wizard Options View (has the set of features to manage the individuals and their relations plus download/upload the *CompGuide* ontology file. This View also shows the total number of individuals saved in the loaded ontology). Most of the features in *CompGuide* Wizard Options View are disabled. This happens since the loaded ontology in the *Protégé Desktop* application is not the same as the *CompGuide* ontology file. In order to activate its features, the loaded active ontology IRI is compared with the *CompGuide* ontology IRI. If both are equal, the *CompGuide Editor* features are enabled to be used by the health professional. This comparison process is absolutely necessary, since it's intended that the created plug-in is only able to manage the individuals of the *CompGuide* ontology file. Only the download ontology feature is always enabled.

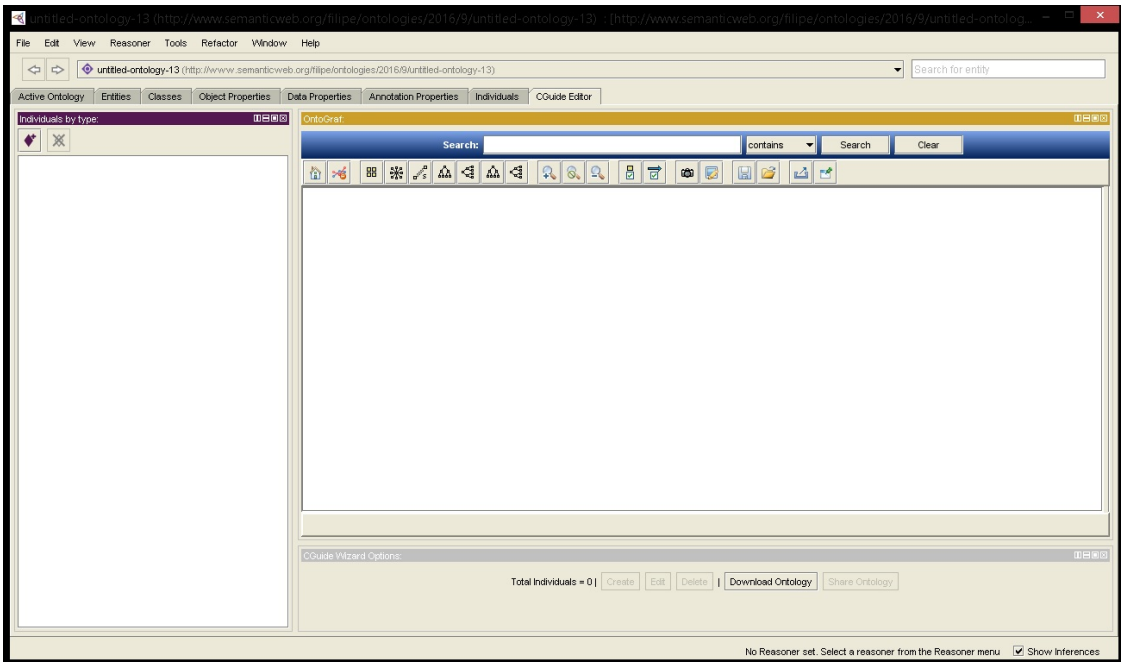


Figure 22.: *CompGuide Editor* plug-in interface in *Protégé Desktop* application

After the *CompGuide* ontology is loaded into the application, all features in the *CompGuide Editor* get enabled, like is shown in figure 23. Also the individuals by type View are updated with the *CompGuide* ontology individuals.

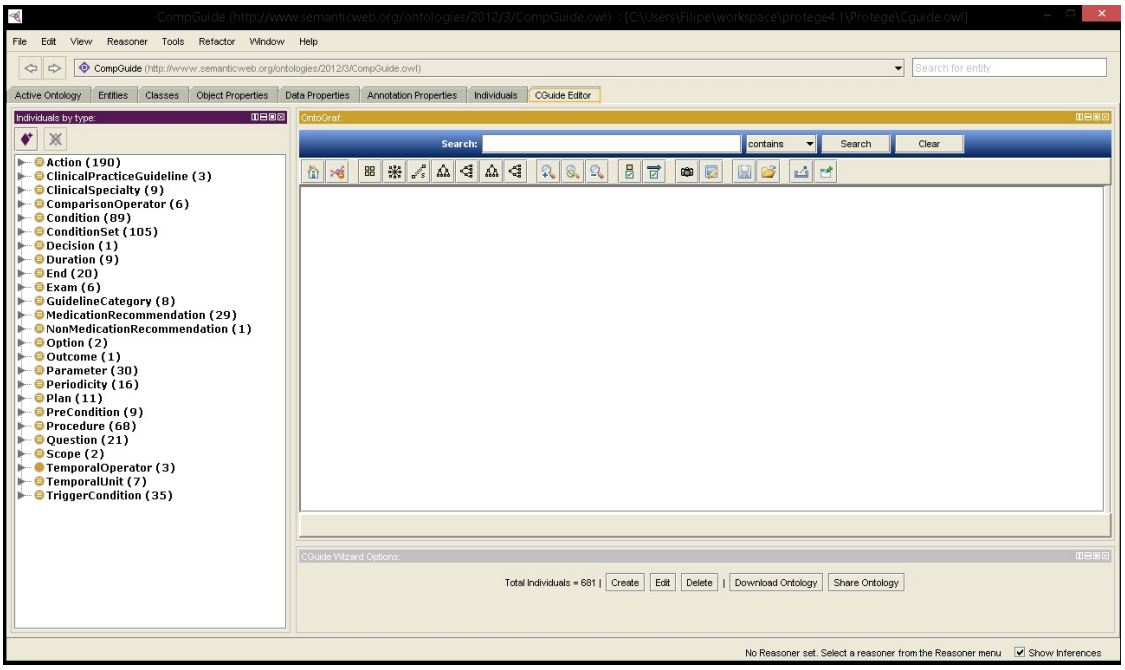


Figure 23.: *CompGuide Editor* plug-in interface in *Protégé Desktop* application after *CompGuide* ontology is loaded.

4.5.1 *Individuals by type and OntoGraf Interface*

The individuals by type View shows the set of individuals saved in the ontology, divided by its OWL classes. The number shown in front of the class name represents the number of individuals that exist in that OWL class. When clicking in one class, the OntoGraf View will update its View, showing a graphical representation of all the individuals of the selected OWL class, and the related object properties. This process also happens when double clicking in any of the graphical nodes shown in the OntoGraf View. Also, when placing the cursor upon one of the shown nodes, the health professional can see its data properties assertions, object properties assertions, super-classes and sub-classes. Each data/object property assertion presents a different color, as does the individuals/classes, and its caption can be seen either by placing the cursor upon the node/arc, or by opening the arc filter. All this can be seen in figures 24 and 25.

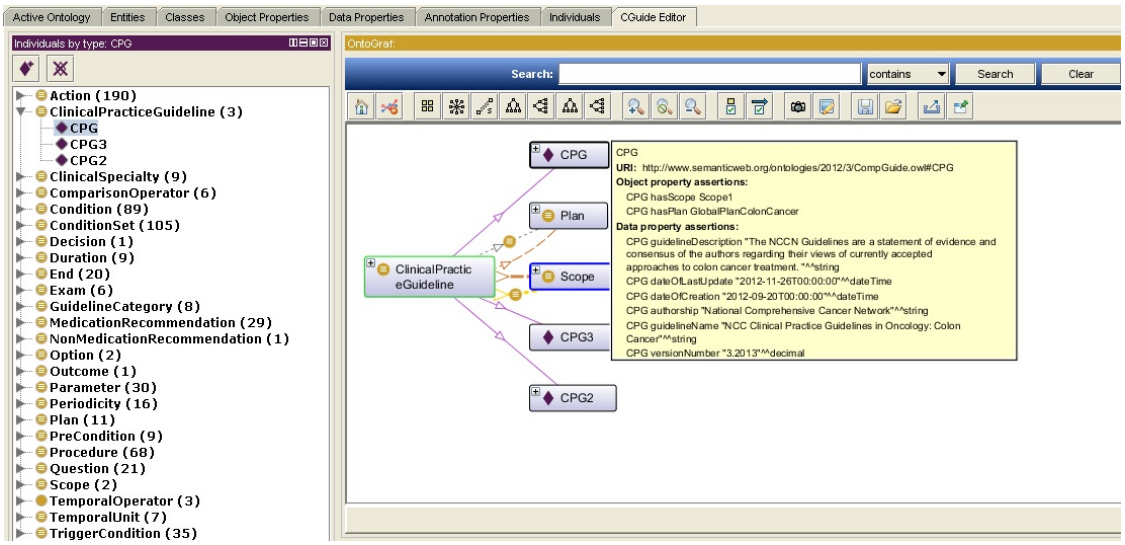


Figure 24.: OntoGraf View in *CompGuide Editor* plug-in Tab.

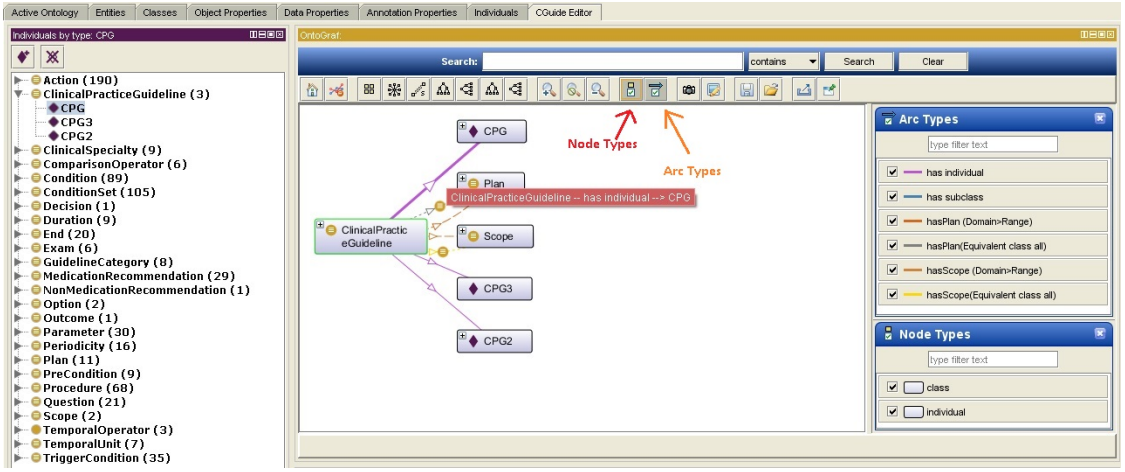


Figure 25.: Filtering Node/Arc in OntoGraf View.

When right clicking in a OntoGraf node (for example in the CPG individual), a few options appear. The Show neighborhood option allows to see all entities linked to this node. Set as focus removes all the entities except the selected entity and centers it. Expand option is similar to Show neighborhood option and shows all related classes and individuals related to the selected entity. The Collapse option does the opposite of the Expand option, removing the nodes related to the selected node. Expand on option allows to expand one of the shown arcs in the list. The list of options explained can be seen in figure 26.

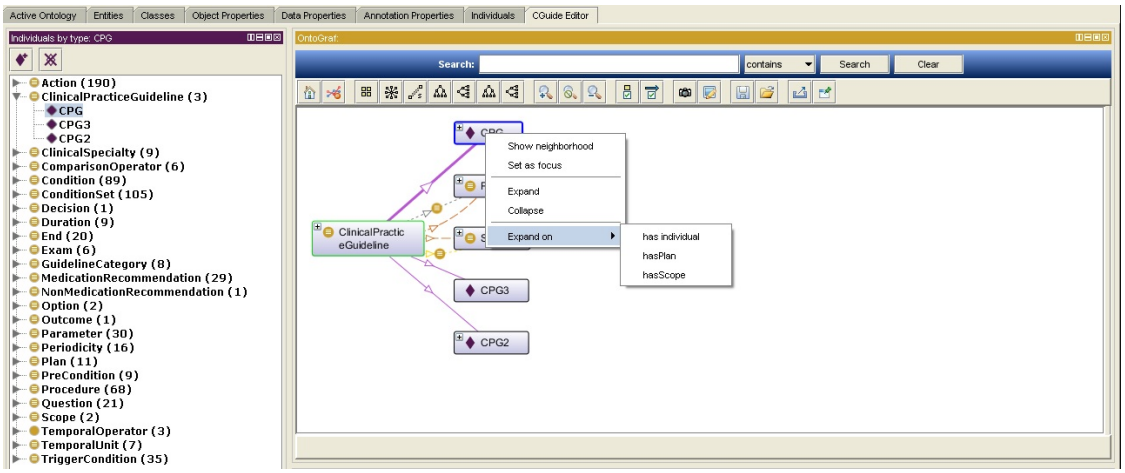


Figure 26.: Options shown when right clicking a node in OntoGraf View.

Other two important features in the OntoGraf top menu are the Configure Node Tooltips and the Export Graph as Image options. The first option (Configure Node Tooltips) allows the health professional to configure the type of data that is shown in the OntoGraf View, both node or arc data type. With this, certain data that may not be important to the health professional may be removed from the graphical representation of the ontology. The second option (Export Graph as Image) allows the health professional to save the ontology representation into an image file format (PNG, GIF or JPEG files). These features are shown in figure 27.

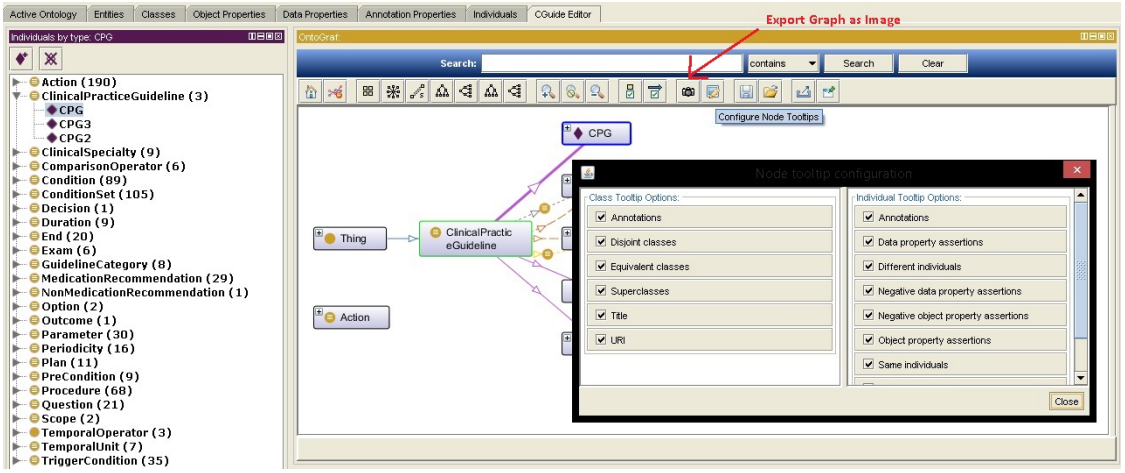


Figure 27.: Configure Node Tooltips and the Export Graph as Image options in OntoGraf View.

On top of these options (figure 27), there's a Search filter and a Clear option, where the health professional may insert the name of an OWL class or individual to automatically be shown in the graphical representation.

Briefly, the OntoGraf View features allow to automatically display the entities of an ontology, represented in a dynamic 2D graphs, in an organized way (tree - Vertical/Horizontal, Vertical/Horizontal Directed, Radial, Spring, or Grid), Zoom in/out the displayed entities, filter the node/arc types, save the graphical representation into an image file or configuring the data shown in the View, among other features. All these features can be accessed by selecting the icons in the top menu (da Silva and Freitas, 2011). A demonstration video of OntoGraf features can be shown in the Youtube OntoGraf link[7].

4.5.2 *CompGuide Wizard Options Interface*

In this view (figure 28) there are five buttons, three associated to the local management of the *CompGuide* ontology (Create, Edit or Delete *CompGuide* individuals) and two associated to receiving (download ontology)/sending (share ontology) *CompGuide* ontology from/to the *CompGuide* repositories.

Figure 28.: *CompGuide* Wizard Options View.

When clicking on any of the Create, Edit or Delete buttons, a new window is opened with a set of options. In here, the health professional can choose which individual class he intends to create/edit/delete. This window is shown in figure 29. For the creation of this wizards, the OWL *Protégé* API Wizard features were used.

7 https://www.youtube.com/watch?v=JRNxvIZ5LBg

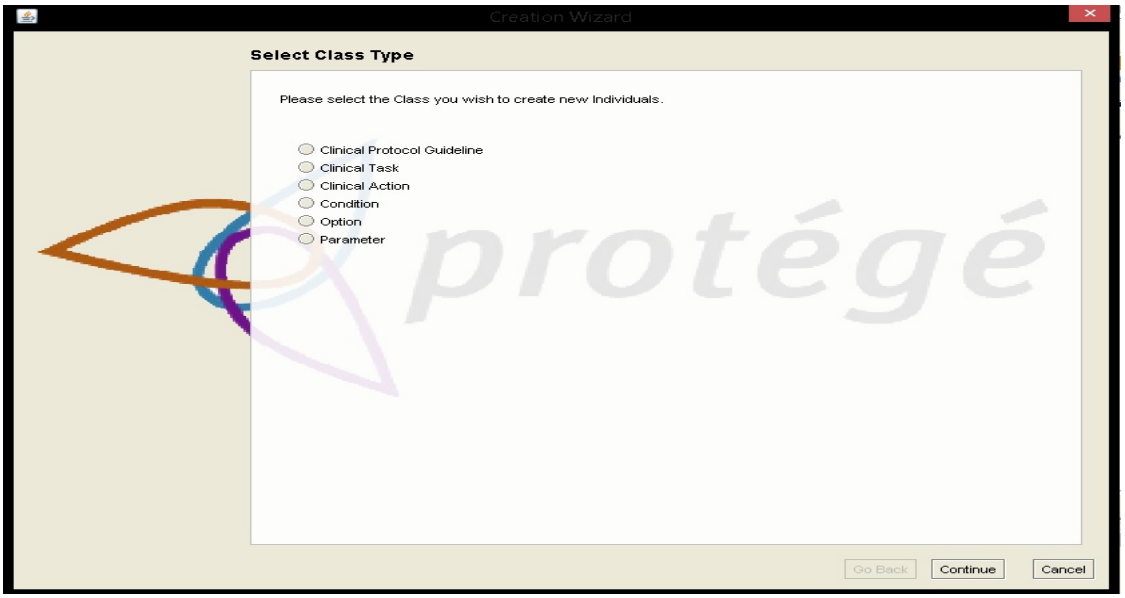


Figure 29.: *CompGuide* Wizard Class Selection Window.

Figure 29 shows the set of options that appears in the first Wizard window. The first option (Clinical Protocol Guidelines) allows the health professional to manage the CPGs. The second option (Clinical Tasks) opens a new set of options, allowing the health professional to select the clinical task type he intends to manage (Plan, Action, Decision, Question, End). This set of options can be seen in figure 30. The third option (Clinical Action) opens a new set of options, allowing the health professional to select the clinical action type he intends to manage (Exam, Formula, Procedure, Medication Recommendation, Non-Medication Recommendation). This set of options can be seen in figure 31. The fourth option (Condition) allows to manage the condition individuals which allow for the implementation of medical and temporal restrictions to the clinical tasks. The fifth option (Option) allows to manage the set of options used in Decision clinical tasks. The last option (Parameter) allows to manage the data requested when executing a Question clinical task.

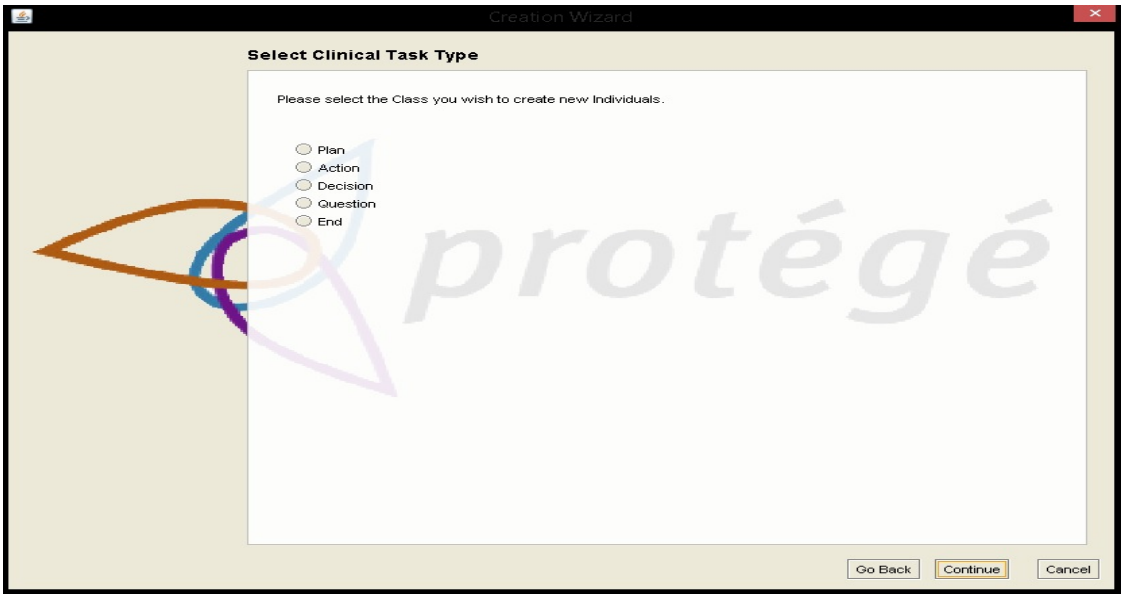


Figure 30.: *CompGuide* Wizard Clinical Task Selection Window.

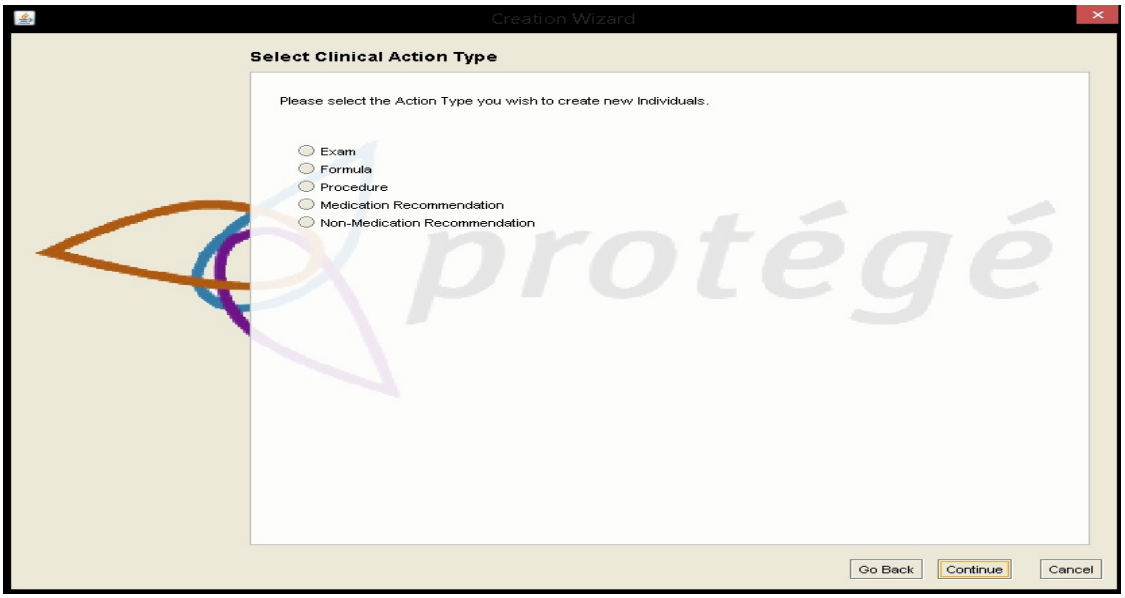


Figure 31.: *CompGuide* Wizard Clinical Action Selection Window.

Now the interfaces creating/editing *CompGuide* individuals will be explained. Both this processes are similar. There are only two differences between these two Wizards: when editing, besides selecting the individual class, the health professional must also select the respective individual to edit. Although both wizards present the same interfaces, the system gets all related data to the selected individual and fills them in the Edit Wizard. When selecting the first option (Clinical Protocol Guidelines), a step-by-step process begins, asking the health professional data related to the creation/edition of CPG. Figures 54, 55 and 56 in Annex C shows the most important windows in this step-by-step creation/edition process. Each window has a small description (top side) asking the health professional the type of data required. Also, there was a caution to create a low complexity and automatic process (low number of steps where the health professional has the need to write any information, having only to select the desired features into the selected individual). The following figures ask for the following data:

- Figure 54 requests the set of clinical specialties of the CPG. The health professional must select from the list of available specialties;
- Figure 55 requests the set of the patient conditions that should be applied in this CPG;
- Figure 56 requests the selection of the Plan associated to this CPG. As explained before, it is in the Plan individual that is implemented the set of clinical tasks of the CP;

When selecting any Condition or Plan (shown in figures 55 and 56), a small description of the selected individual is shown next to the list. After all data is correctly inserted, a message informs of the creation/edition process of the CPG.

As for the clinical tasks, the Wizard allows the health professional to create/edit individuals of these OWL classes. The End option is an additional clinical task, used to indicate the ending of a set of clinical tasks. As such, the individuals of this OWL class have a set of Trigger Conditions that determine the conditions necessary to end the CP. Since the clinical tasks have some similar (but not all) data structures, the creation/edition interfaces of these individuals are mostly the same. Therefore, a general explanation of the most important windows will be made for the management of these individuals:

- Figure 57 requests a small description of the clinical task;
- Figures 58 and 59 requests the Periodicity values and associated stop conditions restrictions. Periodicity allows to determine the repetitive/cycle time required to apply a clinical process in a particular clinical task. Conditions can be applied into this restriction in order to interrupt this process;

- Figure 61 requests how the next tasks should be executed. Parallel task allows a set of tasks to be executed at the same time. Alternative task allows to define a set of alternatives to follow in the execution of a CPG. Preference alternative task is similar to the alternative task option, but adds the possibility to give a preference to one of these choices. The next task option allows to select one clinical task to be executed after the current one is completed;

Until now, all interfaces shown enabled the health professional to select all conditions/clinical tasks (existing in the ontology) to enable automatization and fast management of individuals. Yet, this Wizard also allows the management of clinical actions (Exams, Formula, Procedures, Medical and Non-medical Recommendations), Conditions (health or temporal restrictions in a clinical task), Options (used to define choices to be selected in a Decision clinical task) and Parameters (used to request information in a Question clinical task).

Figures 62 and 63 shows the management of a Condition individual, where are requested the health and temporal data restrictions.

As for the *CompGuide* Delete Wizard, these interfaces are very similar between all the classes. First, the health professional must select the individual class type (like in figure 29). After that, a list of individuals belonging to the selected OWL class will be shown (figure 64), where the health professional must choose the set of individuals to be deleted of the *CompGuide* ontology. In the end, a message appears showing the status of the deletion process.

When accessing the share ontology feature, the following figure (figure 32) is shown. In here, the health professional inserts the username (identification of the *CompGuide* ontology files manager/group) and a description of the ontology modifications made. After completing this process, the ontology will be sent to the *CompGuide* ontology repositories.

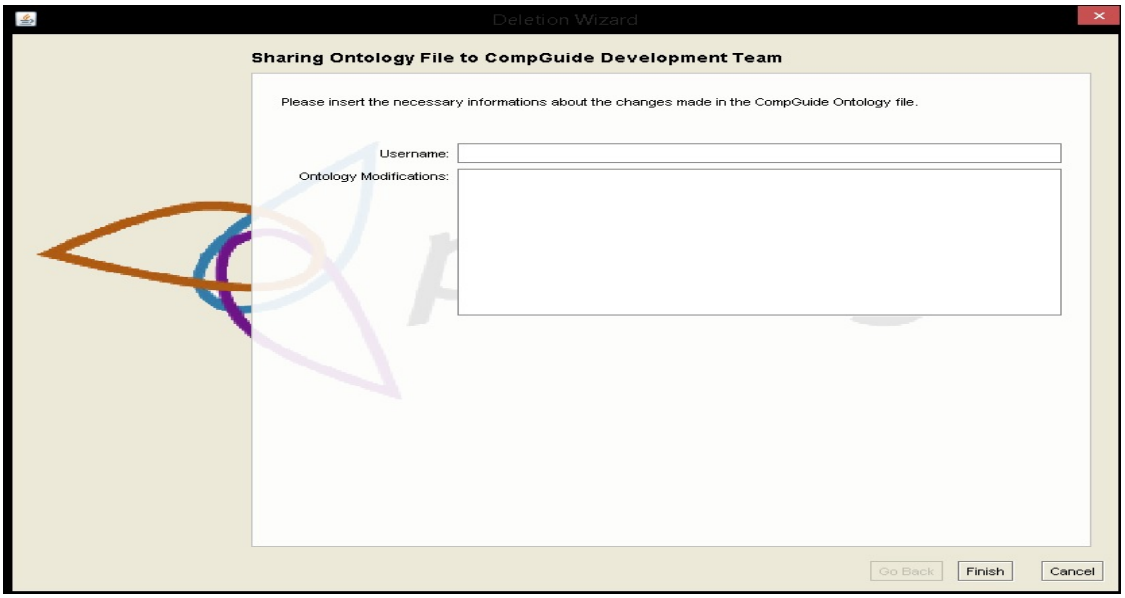


Figure 32.: *CompGuide* Share ontology Window.

4.5.3 *CompGuide Git ontology Repository*

When a health professional clicks the download ontology button (in the *CompGuide Editor* plug-in), the system will download the latest updated files (compacted in a zip file) located in the *CompGuide* master branch Git repository. In this sub-chapter, some figures will be shown illustrating how the admin may update these files, without interfering in the *CompGuide Editor* plug-in code. Figure 33 presents the *CompGuide* master branch Git repository when accessed through GitHub website (while Git is a command line tool, GitHub provides a Web-based graphical interface (Dabbish et al., 2012)).

8 https://github.com/filipebravo123/CompGuide_Plugin

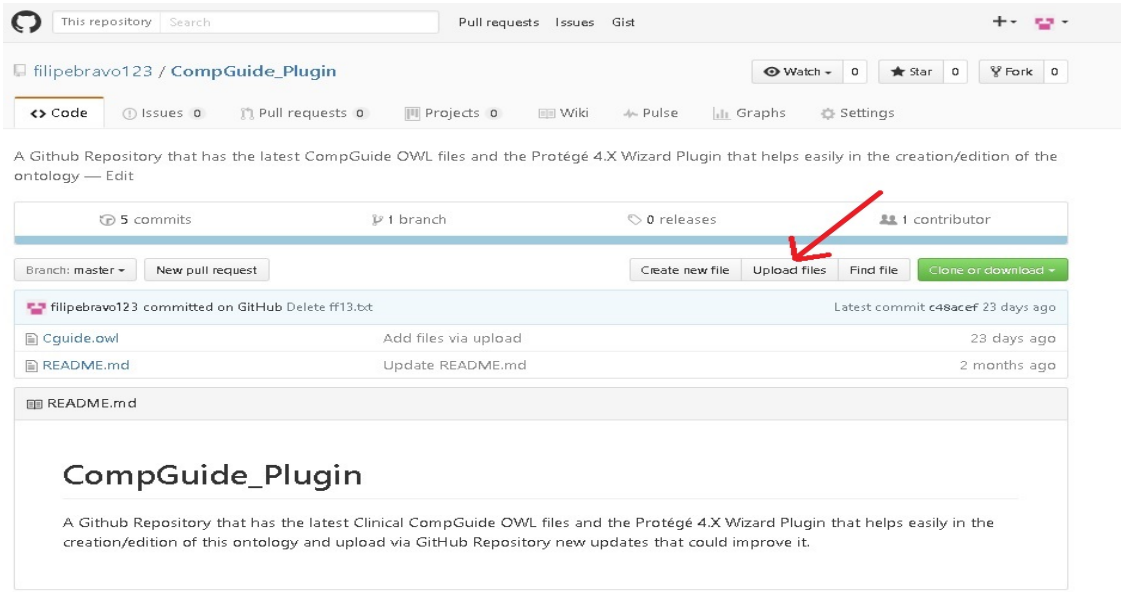


Figure 33.: *CompGuide* Git Master Branch Repository (exported from GitHub Website[8]).

To have access to this web page, the admin must first insert the login credentials and after the validation is complete, all GitHub features will be available. In order to update/upload files to the master branch, the admin must click in the upload files button, opening the web page presented in figure 34. After that, the admin can drag/choose the files to be uploaded/updated into the repository and write a small description of the changes implemented in the newest version. *CompGuide* Git ontology repository can also be updated through the use of Git command line.

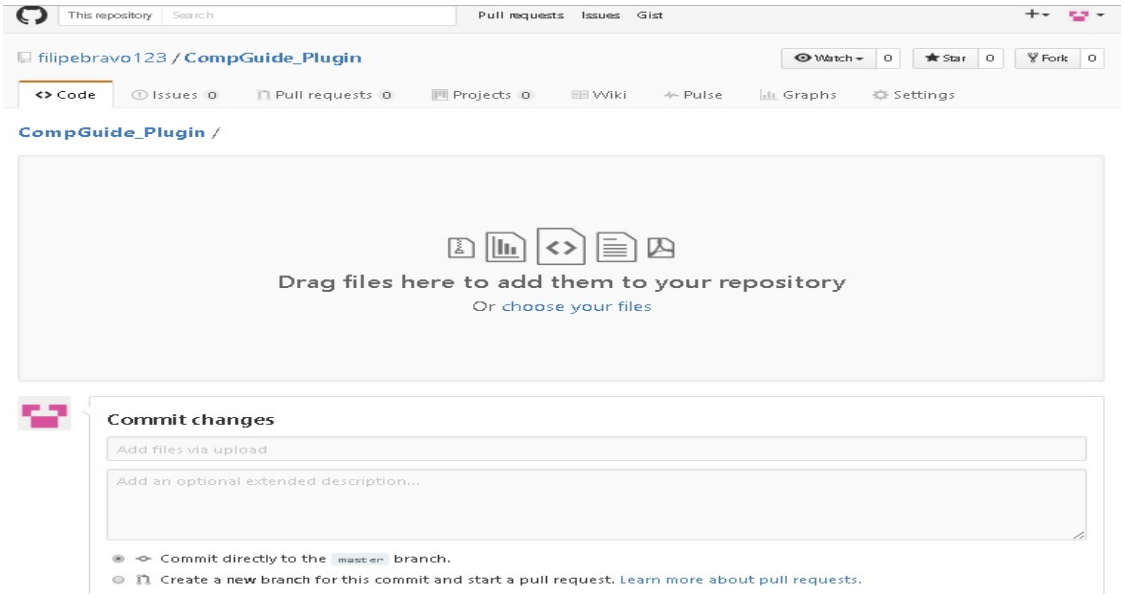


Figure 34.: Upload/Update *CompGuide* Git Master Branch Repository files.

4.5.4 *CompGuide Java Server Repository*

When a health professional clicks the share ontology button (in *CompGuide Editor* plug-in), the system will create a .txt format file and write the username and description input, and will compact this file and the active ontology file (loaded in the *Protégé Desktop* application) into a zip file. After this, the zip file is sent to *CompGuide* Java server repository, through a TCP connection. Figure 35 shows the output messages with a small explanation of each line, when starting this server.

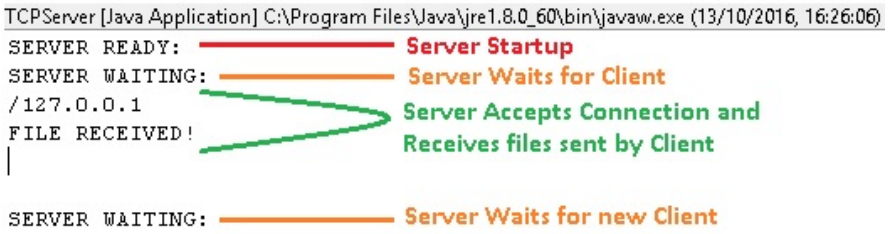


Figure 35.: *CompGuide* Java Server Repository - System Output.

This Java server repository will maintain the service active, and its main objective is to receive all the files sent (through the *CompGuide Editor* plug-in) and organize them by IP address and by temporal date and hour. Figure 36 shows this file organization.

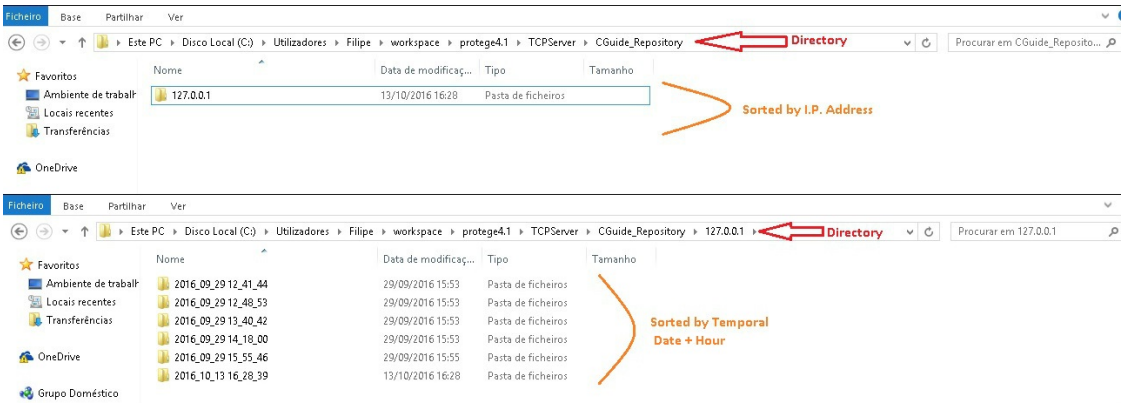


Figure 36.: *CompGuide* Java Server Repository - Files Management.

Figure 37 shows the received compacted file and its contents. As mentioned before, the logfile.txt file shows the username and description input by the health professional.

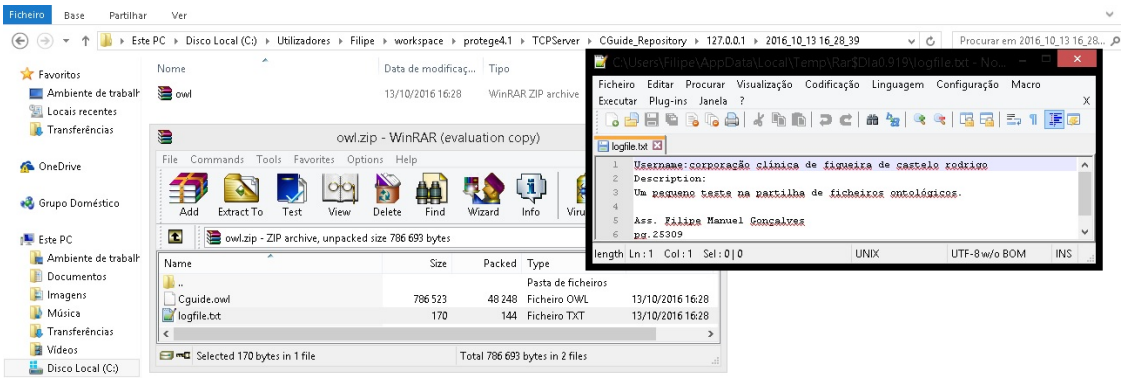


Figure 37.: *CompGuide* Java Server Repository - Unziping Received Files.

4.5.5 *Adding CompGuide Editor View to other Protégé Desktop plug-ins*

A small and easy process must be done in order to implement *CompGuide Editor* features in any *Protégé Desktop* 4 application. The user must first go to the top menu, select Window -> Views -> Ontology Views -> CGuide Wizard Options and drag the plug-in to the desired location in the Tab. The same way can be done to implement other plug-ins features into the *CompGuide Editor* Tab. This process is shown in figure 38.

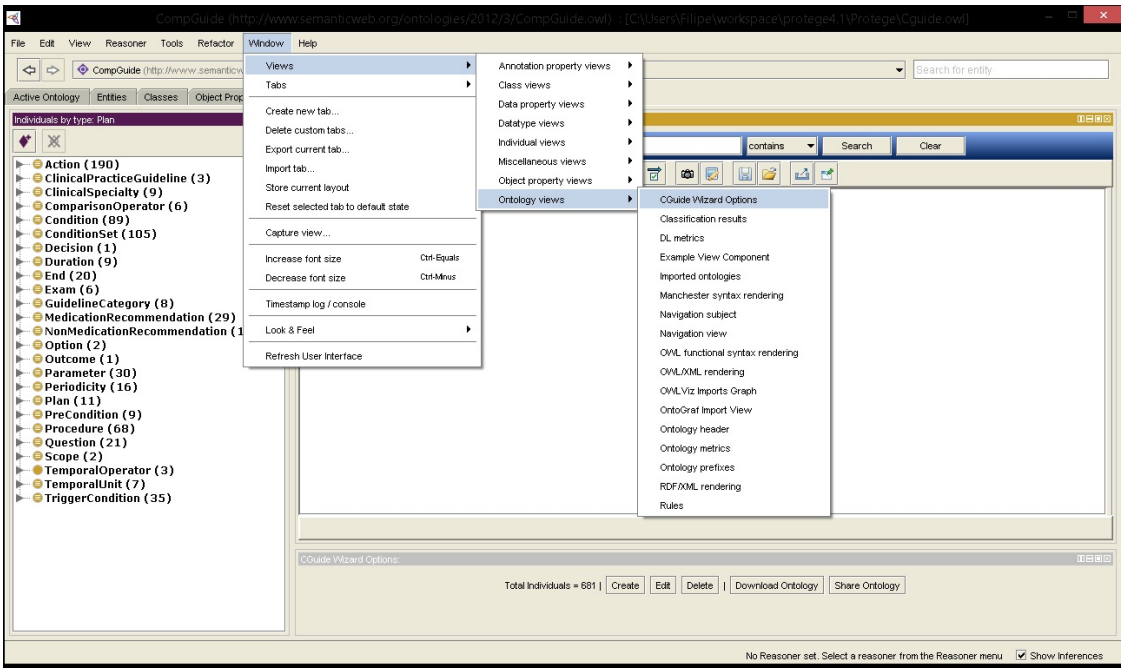


Figure 38.: *Protégé Desktop* plug-ins - Adding *CompGuide Editor* View into *Protégé Desktop* Tab.

4.6 DISCUSSION AND ANALYSIS OF THE SOLUTION

In this chapter, it is explained the steps done in analyzing and adapting the most important elements in a CP (CPGs, Clinical Task, Clinical Actions, etc.) into a digital platform, by study of a class diagram. This process allowed to plan how the *CompGuide Editor* tool should manage the group of OWL individuals in the *CompGuide* ontology file, its advantages and disadvantages, and ways to improve its features.

In accordance with the set of system features, and taking into account all the initial objectives defined, all the goals were acquired positively, allowing the access to features, such as creation, editing and deletion of data concerning the CPGs, done by health professionals. However, some complications were found in the implementation of Git features (as explained in this chapter), leading to a re-planning of these functionalities (related to the transmission of files into cloud repositories). This features were based on the creation of functionalities found in Java server-client file exchanging code, using only the dependent Java libraries applied in the *Protégé Desktop* application. In order to receive the modified *CompGuide* ontologies (edited by the health professionals), a Java TCP Socket server was created, which receives and manages all data files sent from the *CompGuide Editor* tool. As for the *CompGuide* ontologies downloaded from the *CompGuide Editor* tool, these are saved in the Git cloud repository, which only the admin can manage its files.

In this system, the admin can access and change the files stored in both the repositories. All transmitted files are compressed, reducing the required amount of data to send. In addition, newer versions of *CompGuide* ontology can be updated in the Git repository without the need to change the features of *CompGuide Editor* tool.

With this step finished, all objectives were acquired with positive results.

5

CONCLUSION

In this chapter we will briefly describe the conclusions obtained in this work, report the set of objectives completed, limitations of the proposed solution, and introduce perspectives for future work.

This work focuses on the necessity of platforms for the creation and edition of CIGs, in facilitating their use by people who do not possess programming skills. CIGs hold large amounts of clinical knowledge which are used in CDSSs. As such it is one of the key components in promoting improvements in the quality of clinical processes, reducing variations in clinical practice and reducing health care costs.

Another important aspect is the support that these systems can give to health professionals, which are subject to stressful situations, responsible for medical errors, variations in clinical practice, and practice of defensive medicine. This shows that it is necessary to approach health professionals with good clinical practice and evidence-based medicine, by giving some assistance in the decision making with the help of computer science.

In computer science, several languages and tools exist for helping final users and system developers in creating good and effective CIGs, such as *Protégé Desktop*, *SAGE Workbench*, *Tallis*, *GEM Cutter*, *Asbru View* and many others. Yet most of these platforms lack some important features, leaving place for improvements.

By studying these tools, the creation of a plug-in for the *Protégé Desktop* application was planned, with the capacity to manage and share the *CompGuide* ontologies, through the use of cloud repositories. The study and analysis of the structure of existing classes in the *CompGuide* ontology, as well as reading various tutorials (regarding the plug-in creation process) were essential in the progress of this project. The use of UML diagrams helped to keep the focus on the set of objectives to be accomplished.

5.1 ACCOMPLISHMENT OF THE OBJECTIVES AND CONTRIBUTIONS

Regarding the objectives defined in sub-chapter 1.9, it can be considered that they have been satisfactorily achieved. Taking into consideration that it was prepared an extensive analysis of the problem. The implementation of a tool with this level of complexity became

a difficult task. Yet, in the end all project requirements have been implemented, and as a result, the final solution meets the objectives imposed in the beginning.

The first objective, identify key aspects for representing medical knowledge in CDSSs, regarded the collection of information of CIGs, necessary to detect the main challenges that may arise, and describe its characteristics. These aspects are shown in chapter 1 and it shows that the main requirement to express this medical knowledge is by creating clinical tasks linked in a clinical workflow.

The second and third objectives, identify tools used to create and edit CIGs and their key issues and identify aspects that could be improved or applied in these tools, involved analysing the set of solutions or tools that can solve the problems previously surveyed. Its conclusion is shown in chapter 2. Although existing tools provide a set of functionalities that meet many needs for the user community, there are still some unmet challenges. The main challenge is the little knowledge users may have about the terminologies and ontologies used in the tools, which implies a longer learning curve in becoming acquainted with the concepts and learning how to create terminology-enabled representations. This was the main challenge to be taken into account in the creation of the *CompGuide Editor* tool.

The fourth objective, formalize the creation and editing of CPs in the OWL language, was achieved with the characterization and identification of the most important aspects in the CPs of the *CompGuide* ontology. Its conclusion is shown in chapter 3 and it shows the set of requirements that the *CompGuide Editor* tool must meet, in accordance with the needs of its users.

Objectives five - eight (design a tool capable of managing the stored set of parameters in CPs, without the need for advanced programming skills; design features capable of generating graphical representations of CPs; design features capable of downloading the latest CIG files; design features capable of sending their modified CIG files to the development team), were achieved as shown in chapter 4, in which it is explained the organization and execution of the features in the *CompGuide Editor* tool and the design of an interface that allows a health professional to view the management of a *CompGuide* ontology. Additionally, the tool incorporates features for accessing cloud repositories. As for the sharing features, some limitations appeared, which are explained in the sub-chapter 5.2.

The value proposition of this project lies in the development of a user-friendly tool (*CompGuide Editor*) able to create, edit and manage CIGs that use the *CompGuide* ontology in a simple and intuitive way and without requiring the user to have programming knowledge. Through the method of Wizard it allows to smooth the learning curve necessary to handle ontologies. The *CompGuide Editor* tool is also able to visualize CIG elements in a list or in a dynamic 2D graphic, and has features that enable sharing ontologies with a guideline development team. This need comes since different approaches that exist do not present some crucial features.

5.2 LIMITATIONS AND PERSPECTIVES FOR FUTURE WORK

A major limitation in building the *CompGuide Editor* plug-in took place in the creation of features related to the download/upload of ontological files. The initial idea was to use the features of the free Git platform, in order to manage all file sharing within a cloud repository, without any financial cost. However, since the *CompGuide Editor* tool is a plug-in that runs dependent of *Protégé Desktop* application APIs, the use of external APIs became very difficult. The way to solve this problem was to create sharing features using only the APIs used by the *Protégé Desktop* application. Despite this limitation, these features have been successfully created, as it is shown in chapter 4.

Although the *CompGuide Editor* plug-in displays sharing functionalities of the *CompGuide* ontology, this feature only sends/receives the corresponding ontological file to a cloud repository. To access the data presented in this file, the user must open it with an OWL file manager. Mapping all this information to a database would be an asset in the management of the *CompGuide* ontologies. With the ontology imported into a database, the user would simply need to do queries from the *Protégé Desktop* interface, where all changes would be saved directly into the cloud repository. This is a relevant aspect for future developments of the tool.

SPARQL *Protégé* plug-in [1] provides support for composing and executing database queries from within the *Protégé Desktop* interface, and could be one of the solutions for this future work.

However, these features are dependent of a tool that would import the *CompGuide* ontology into the database, providing quick updates to the data stored in the cloud.

1 http://protegewiki.stanford.edu/wiki/SPARQL_Query

BIBLIOGRAPHY

Adratt, Eduardo and LIMA, L and BARRA, C. (2004). Guidelines: Fundamentos Teóricos e Evolução Tecnológica dentro da Medicina. In *IX Congresso Brasiliero Informática em Saúde (CBIS)*, pages 07–10.

Allen, J. D. and Unicode Consortium. (2007). *The Unicode standard 5.0*. Addison-Wesley.

Beard, N., Campbell, J. R., Huff, S. M., Leon, M., Mansfield, J. G., Mays, E., McClay, J. C., Mohr, D. N., Musen, M. A., O'Brien, D., et al. (2002). Standards-based sharable active guideline environment (sage): A project to develop a universal framework for encoding and disseminating electronic clinical practice guidelines. In *AMIA*.

Bechhofer, S., Hendler, J., Patel-Schneider, P. F., Stein, L. A., and Olin, F. W. (2004). OWL Web Ontology Language Reference.

Benko, A. and Sik Lányi, C. (2009). *Encyclopedia of Information Science and Technology, Second Edition*. IGI Global.

Berner, E. S. (2007). *Clinical decision support systems*. Springer.

Bittner, K. (2002). *Use case modeling*. Addison-Wesley Longman Publishing Co., Inc.

Boggs, W. and Boggs, M. (2002). *Mastering UML with rational rose 2002*. Sybex.

Bott, R. (2014). Summary of the Guideline Workbenches Evaluation. *Igarss 2014*, (1):1–5.

Briand, L. C., Labiche, Y., and Miao, Y. (2003). Towards the Reverse Engineering of UML Sequence Diagrams.

Chim, JCS and Cheung, NT and Fung, H and Wong, K. (2003). Electronic clinical practice guidelines: current status and future prospects in Hong Kong. *Hong Kong Medical Journal*, 9(4):299—-301.

Chung, L., Nixon, B. A., Yu, E., and Mylopoulos, J. (2012). *Non-functional requirements in software engineering*, volume 5. Springer Science & Business Media.

Clancey, W. J. and Shortliffe, E. H. (1984). Readings in medical artificial intelligence: the first decade.

Cockburn, A. (2008). Why I still use use cases. *alistair.cockburn.us*.

Cristani, Matteo and Cuel, R. (2005). A Survey on Ontology Creation Methodologies. *Int. J. Semantic Web Inf. Syst.*, 1(2):49—-69.

da Silva, I. C. S. and Freitas, C. M. D. S. (2011). Using visualization for exploring relationships between concepts in ontologies. In *2011 15th International Conference on Information Visualisation*, pages 317–322. IEEE.

Dabbish, L., Stuart, C., Tsay, J., and Herbsleb, J. (2012). Social coding in github: transparency and collaboration in an open software repository. In *Proceedings of the ACM 2012 conference on Computer Supported Cooperative Work*, pages 1277–1286. ACM.

de Clercq, P. A., Blom, J. A., Korsten, H. H. M., and Hasman, A. (2004). Approaches for creating computer-interpretable guidelines that facilitate decision support. *Artificial intelligence in medicine*, 31(1):1–27.

Douglass, B. P. (2003). Real time uml. In *Formal Techniques in Real-Time and Fault-Tolerant Systems: 7th International Symposium, FTRTFT 2002, Co-sponsored by IFIP WG 2.2, Oldenburg, Germany, September 9-12, 2002. Proceedings*, volume 2469, page 53. Springer.

Eysenbach, G. (2001). What is e-health? *Journal of medical Internet research*, 3(2):e20.

Eysenck, M. and Keane, M. T. (2005). *Cognitive Psychology: A Student's Handbook: A Student's Handbook 5th Edition*. Psychology Press.

Field, M. J., Lohr, K. N., et al. (1992). *Guidelines for clinical practice: from development to use*. National Academies Press.

Fox, J., Johns, N., Lyons, C., Rahmanzadeh, A., Thomson, R., and Wilson, P. (1997). Proforma: a general technology for clinical decision support systems. *Computer methods and programs in biomedicine*, 54(1):59–67.

Gandrud, C. (2013). Github: A tool for social data set development and verification in the cloud. *Available at SSRN 2199367*.

Gomaa, H. (2001). Designing concurrent, distributed, and real-time applications with uml. In *Proceedings of the 23rd international conference on software engineering*, pages 737–738. IEEE Computer Society.

Gopher, D., Olin, M., Badihi, Y., Cohen, G., Donchin, Y., Bieski, M., and Cotev, S. (1989). The Nature and Causes of Human Errors in a Medical Intensive Care Unit. *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, 33(15):956–960.

Gosling, J. and Mcgilton, H. (1996). The Java^{TM} Language Environment.

Grilli, R. and Lomas, J. (1994). Evaluating the message: the relationship between compliance rate and the subject of a practice guideline. *Medical care*, 32(3):202–13.

Heflin, J., Hendler, J., and Luke, S. (1999). SHOE: A Knowledge Representation Language for Internet Applications.

Horn, W. (2001). AI in medicine on its way from knowledge-intensive to data-intensive systems. *Artificial Intelligence in Medicine*, 23(1):5–12.

Horridge, M. and Bechhofer, S. (2009). The owl api: a java api for working with owl 2 ontologies. In *Proceedings of the 6th International Conference on OWL: Experiences and Directions-Volume 529*, pages 49–58. CEUR-WS. org.

Horrocks, I., Patel-Schneider, P. F., and van Harmelen, F. (2003). From SHIQ and RDF to OWL: the making of a Web Ontology Language. *Web Semantics: Science, Services and Agents on the World Wide Web*, 1(1):7–26.

Hripcsak, G. (1994). Writing arden syntax medical logic modules. *Computers in biology and medicine*, 24(5):331–363.

Huber, B. (2005). Asbruview 2.0 User Guide.

Isern, D. and Moreno, A. (2008). Computer-based execution of clinical guidelines: a review. *International journal of medical informatics*, 77(12):787–808.

Jacobson, I., Bittner, K., and Spence, I. (2011). *Use Case 2.0: The Guide to Succeeding with Use Cases*. Ivar Jacobson International.

Karras, B. T., Nath, S., and Shiffman, R. N. (2000). A preliminary evaluation of guideline content mark-up using gem–an xml guideline elements model. In *Proceedings of the AMIA Symposium*, page 413. American Medical Informatics Association.

Khalifa, M. (2014). Clinical Decision Support: Strategies for Success. *Procedia Computer Science*, 37:422–427.

Kim, J and Shim, BinGu and Kim, SunTae and Lee, JaeHoon and Cho, InSook and Kim, Y. (2009). Translation Protégé knowledge for executing clinical guidelines. In *Proceedings of Conference Protege*.

Knublauch, H., Fergerson, R. W., Noy, N. F., and Musen, M. A. (2004). The protégé owl plugin: An open development environment for semantic web applications. In *International Semantic Web Conference*, pages 229–243. Springer.

Knublauch, H., Horridge, M., Musen, M. A., Rector, A. L., Stevens, R., Drummond, N., Lord, P. W., Noy, N. F., Seidenberg, J., and Wang, H. (2005). The protege owl experience. In *OWLED*.

Kononenko, I. (2001). Machine learning for medical diagnosis: history, state of the art and perspective. *Artificial Intelligence in Medicine*, 23(1):89–109.

Leape, L. L., Lawthers, A. G., Brennan, T. A., and Johnson, W. G. (1993). Preventing medical injury. *QRB. Quality review bulletin*, 19(5):144–9.

Lee, H., Seo, B.-K., and Seo, E. (2013). A git source repository analysis tool based on a novel branch-oriented approach. In *2013 International Conference on Information Science and Applications (ICISA)*, pages 1–4. IEEE.

Leong, T. Y., Kaiser, K., and Miksch, S. (2007). Free and open source enabling technologies for patient-centric, guideline-based clinical decision support: a survey. *Yearbook of medical informatics*, pages 74–86.

Liu, L. and Özsu, M. T. (2009). *Encyclopedia of database systems*, volume 6. Springer Berlin, Heidelberg, Germany.

Loeliger, J. (2006). Collaborating with git. *Linux Magazine, June*.

Loeliger, J. and McCullough, M. (2012). *Version Control with Git: Powerful tools and techniques for collaborative software development*. " O'Reilly Media, Inc.".

Lozano, E., Marcos, M., Martínez-Salvador, B., Alonso, A., and Alonso, J. R. (2009). Experiences in the development of electronic care plans for the management of comorbidities. In *International Workshop on Knowledge Representation for Health Care*, pages 113–123. Springer.

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

McCarthy, J. (2001). What is Artificial Intelligence? 0(sep 28):14.

Mcguinness, D., Fikes, R., Hendler, J., and Stein, L. (2002). DAML+OIL: an ontology language for the Semantic Web. *IEEE Intelligent Systems*, 17(5):72–80.

McGuinness, D. L., Van Harmelen, F., et al. (2004). Owl web ontology language overview. *W3C recommendation*, 10(10):2004.

Michel, G. and Shiffman, R. (2009). GEM Cutter 2.5 User Guide.

Musen, M. A. and Protégé Team (2015). The Protégé Project: A Look Back and a Look Forward. *AI matters*, 1(4):4–12.

Noy, N. F., Crubézy, M., Fergerson, R. W., Knublauch, H., Tu, S. W., Vendetti, J., Musen, M. A., et al. (2003). Protege-2000: an open-source ontology-development and knowledge-acquisition environment. In *AMIA Annu Symp Proc*, volume 953, page 953.

Noy, Natalya F and Sintek, Michael and Decker, Stefan and Crubézy, Monica and Fergerson, Ray W and Musen, M. A. (2001). Creating semantic web contents with protege-2000. *IEEE intelligent systems*, (2):60—-71.

Oettinger, A. (2005). The Tallis Composer User Interface.

Oliveira, T., Leão, P., Novais, P., and Neves, J. (2014). Webifying the Computerized Execution of Clinical Practice Guidelines. *Trends in Practical Applications of Heterogeneous Multi-Agent Systems. The PAAMS Collection SE - 18*, 293:149–156.

Oliveira, T., Neves, J., Barbosa, E., and Novais, P. (2013a). Clinical Careflows Aided by Uncertainty Representation Models. *Hybrid Artificial Intelligent Systems*, 8073(i):71–80.

Oliveira, T., Novais, P., and Neves, J. (2013b). Representation of clinical practice guideline components in owl. In *Trends in Practical Applications of Agents and Multiagent Systems*, pages 77–85. Springer.

Oliveira, Tiago and Novais, Paulo and Neves, J. (2013). Representation of Clinical Practice Guideline Components in OWL. In *Trends in Practical Applications of Agents and Multiagent Systems*, pages 77—-85. Springer.

Patel, V. L., Allen, V. G., Arocha, J. F., and Shortliffe, E. H. (1998). Representing clinical guidelines in glif. *Journal of the American Medical Informatics Association*, 5(5):467–483.

Patel-Schneider, D. F., van Harmelen, F., Horrocks, I., McGuinness, D. L., and F., P. (2001). OIL: An Ontology Infrastructure for the Semantic Web. *IEEE Intelligent Systems*, 16(1541-1672):38–45.

Peleg, M., Tu, S., Bury, J., Ciccarese, P., Fox, J., Greenes, R. A., Hall, R., Johnson, P. D., Jones, N., Kumar, A., et al. (2003). Comparing computer-interpretable guideline models: a case-study approach. *Journal of the American Medical Informatics Association*, 10(1):52–68.

Pohl, K. (2010). *Requirements engineering: fundamentals, principles, and techniques*. Springer Publishing Company, Incorporated.

Polvani, K.-A., Agrawal, A., Karras, B., Deshpande, A., and Shiffman, R. (2000). Gem cutter manual. *Yale Center for Medical Informatics*, pages 122–131.

Purchase, H. C., Colpoys, L., Mcgill, M., Carrington, D., and Britton, C. UML class diagram syntax: an empirical study of comprehension.

Purchase, H. C., Colpoys, L., McGill, M., Carrington, D., and Britton, C. (2001). Uml class diagram syntax: an empirical study of comprehension. In *Proceedings of the 2001 Asia-Pacific symposium on Information visualisation-Volume 9*, pages 113–120. Australian Computer Society, Inc.

Rospocher, M., Eccher, C., Ghidini, C., Hasan, R., Seyfang, A., Ferro, A., and Miksch, S. (2010). Collaborative encoding of asbru clinical protocols. In *International Conference on Electronic Healthcare*, pages 135–143. Springer.

Rubin, D. L., Noy, N. F., and Musen, M. A. (2007). Protege: a tool for managing and using terminology in radiology applications. *Journal of Digital Imaging*, 20(1):34–46.

Saridis, G. N. (2001). *Hierarchically Intelligent Machines*. World Scientific.

Shiffman, R. N., Agrawal, A., Deshpande, A. M., and Gershkovich, P. (2001). An approach to guideline implementation with gem. *Studies in health technology and informatics*, (1):271–275.

Somekh and Bridget (2005). *Action Research: A Methodology For Change And Development: A Methodology for Change and Development*.

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

Steele, Rory and Primer, F. J. T. P. (2002). Introduction to PROforma language and software with worked examples. Technical report, Technical report. London, UK: Advanced Computation Laboratory, Cancer Research.

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

ten Teije, A., Miksch, S., and Lucas, P. (2008). *Computer-based Medical Guidelines and Protocols: A Primer and Current Trends*. IOS Press.

Timnat, S., Braginsky, A., Kogan, A., and Petrank, E. (2012). Wait-Free Linked-Lists.

und Naturwissenschaften, S. G. V. f. M. (2015). Archive. *IMIA Yearbook*, pages 145–158.

Van Lamsweerde, A. (2009). *Requirements engineering: from system goals to UML models to software specifications*. Wiley Publishing.

Votruba, P. (2003). *Structured knowledge acquisition for asbru*. na.

Wang, X. H., Zhang, D. Q., Gu, T., and Pung, H. K. (2004). Ontology based context modeling and reasoning using owl. In *Pervasive Computing and Communications Workshops, 2004. Proceedings of the Second IEEE Annual Conference on*, pages 18–22. Ieee.

Wei, Z. and Hong, M. (2003). A feature-oriented domain model and its modeling process. *Journal of Software*, 14(8):1345–1356.

Wiegers, K. E. (2003). *Software requirements : practical techniques for gathering and managing requirements throughout the product development cycle*. Microsoft Press, Redmond, 2nd edition.

Young, R. R. (2001). *Effective requirements practices*. Addison-Wesley.

A

USE CASE TEXT DESCRIPTIONS

This Annex provides a description of the different use cases running in the *CompGuide Editor* system, in the form of tables 2-9. These use case descriptions are properly explained in sub-chapter 4.4. Create Wizard Use Case Text Description is related with the creation features, Edit Wizard Use Case Text Description is related with the edition features and Delete Wizard Use Case Text Description is related with the delete features, which are all managed locally. The rest of the use case descriptions are associated with the download and share of ontology features (from both client and server side), which are dependent of the internet connectivity, as it is shown in the pre-condition.

States Caption:

- **UD** - User Defined (feature accessed by the user);
- **OUT** - Output (data output by the system);
- **INP** - Input (data input by the user);
- **VAL** - Validation (system validation of a condition);
- **IVAL** - Input Validation (system validation of the input data by the user);
- **ALT** - Alternative (shows an alternative from the current process);

Table 2.: Create Wizard Use Case Text Description

Name			**Create new individuals in the *CompGuide* CIG**
Purpose			Automatically creates and inserts *CompGuide* class individuals into the *CompGuide* CIG, and adds data property based on the input data.
Pré-Condition			*CompGuide* ontology OWL file must be loaded in *Protégé Desktop* 4 application.
Pós-Condition			None.
Super Use Case			None.
Event Flow			**Normal behavior.**
	UD	1.	Health Professional access the Create Wizard feature;
	OUT	2.	System displays a set of options, asking the Health Professional what individual class does he want to create;
	INP	3.	Health Professional selects desired class;
	OUT	4.	System asks multiple questions to the Health Professional, requiring the necessary data to create the individual;
	INP	5.	Health Professional answers all the questions in the system;
	OUT	6.	System reaches the end of the Wizard;
	IVAL	7.	System verifies if all data were input correctly;
		8.	System creates all the necessary individuals, links them to the associated classes, and integrates all input data into the created individuals;
	OUT	9.	System informs the Health Professional that the changes in the ontology were completed.
	ALT		**Alternative 7a [Invalid Data]:**
	OUT	1.	System informs the Health Professional that the changes in the ontology failed;
		2.	Returns to 2;

Table 3.: Edit Wizard Use Case Text Description

Name			**Edit individuals in the *CompGuide* CIG**
Purpose			Automatically edits the selected *CompGuide* class individuals into the *CompGuide* CIG, by editing data property based on the input data.
Pré-Condition			*CompGuide* ontology OWL file must be loaded in *Protégé Desktop* 4 application.
Pós-Condition			None.
Super Use Case			None.
Event Flow			**Normal behavior.**
	UD	1.	Health Professional access the Edit Wizard feature;
	OUT	2.	System displays a set of options, asking the Health Professional what individual class does he want to edit;
	INP	3.	Health Professional selects desired class;
	OUT	4.	System ask Health Professional which individual does he want to edit;
	INP	5.	Health Professional selects individual;
	OUT	6.	System asks multiple questions to the Health Professional, requiring the necessary data to edit the selected individual. Shown textfields are filled with the selected individual data;
	INP	7.	Health Professional answers all the questions in the system;
	OUT	8.	System reaches the end of the Wizard;
	IVAL	9.	System verifies if all data were input correctly;
		10.	System deletes old data and creates the necessary individuals, associates them to selected classes, and integrates and links all input data into the edited individuals;
	OUT	11.	System informs the Health Professional that the changes in the ontology were completed.
	ALT		**Alternative 9a [Invalid Data]:**
	OUT	1.	System informs the Health Professional that the changes in the ontology failed;
		2.	Returns to 2;

Table 4.: Delete Wizard Use Case Text Description

Name			**Delete selected individuals in the *CompGuide* CIG**
Purpose			Automatically deletes all selected and related *CompGuide* class individuals in the *CompGuide* CIG.
Pré-Condition			*CompGuide* ontology OWL file must be loaded in *Protégé* application.
Pós-Condition			None.
Super Use Case			None.
Event Flow			**Normal behavior.**
	UD	1.	Health Professional access the Delete Wizard feature;
	OUT	2.	System displays a set of options, asking the Health Professional what individual class does he want to delete;
	INP	3.	Health Professional selects desired class;
	OUT	4.	System ask Health Professional which individual does he want to delete;
	INP	5.	Health Professional selects individuals;
	OUT	6.	System reaches the end of the Wizard;
		7.	System deletes all selected and linked individuals.
	OUT	8.	System informs the User that the selected individuals were eliminated.

Table 5.: Download *CompGuide* CIG Use Case Text Description

Name			**Download *CompGuide* CIG.**
Purpose			Download the latest OWL files related to the *CompGuide* CIG.
Pré-Condition			Access to a Internet connection.
Pós-Condition			None.
Super Use Case			None.
Event Flow			**Normal behavior.**
	UD	1.	Health Professional access the Download CIG feature;
	VAL	2.	System verifies *CompGuide* Git master branch http connection link;
		3.	Download zip file from http link;
	VAL	4.	Verify existence of *CompGuide* repository folder in *Protégé* application directory.
		5.	Create *CompGuide* repository folder;
		6.	Create folder in *CompGuide* repository local directory with temporal date plus hour name;
		7.	Unzip zip file in the latest created folder directory;
		8.	Delete zip file;
	OUT	9.	System informs the Health Professional the download was completed successfully.
	ALT		**Alternative 2a [*CompGuide* Git repository not accessible]**
	OUT	1.	System informs the Health Professional there is a problem in the connection with Git repository;
		2.	End

Table 6.: Share *CompGuide* CIG Use Case Text Description

Name			**Upload Shared *CompGuide* CIG.**
Purpose			Send *CompGuide* CIG OWL files modified by Health Professionals to *CompGuide* server repository.
Pré-Condition			Access to a Internet connection.
Pós-Condition			None.
Super Use Case			None.
Event Flow			**Normal behavior.**
	UD	1.	Health Professional access the Share CIG feature;
	IVAL	2.	System verifies *CompGuide* server repository connection status;
	OUT	3.	System asks the Health Professional for an username/identification and description of the modified details made in *CompGuide* CIG;
	UD	4.	User inserts required data;
		5.	System creates .txt file and writes all input data into the created file;
		6.	Create zip file with created .txt file and *CompGuide* CIG OWL file;
		7.	Send zip file to *CompGuide* server repository;
		8.	System sending process completed;
		9.	System deletes .txt and zip files;
	OUT	10.	System informs the Health Professional that the uploaded process was completed successfully.
	ALT		**Alternative 2a [*CompGuide* server repository not accessible]**
	OUT	1.	System informs the Health Professional there is a problem in the connection with the *CompGuide* server repository;
		2.	End
	ALT		**Alternative 8a [File unsuccessfuly sent]**
	OUT	1.	System informs the Health Professional that the file couldn't be sent;
		2.	End

Table 7.: Download Shared *CompGuide* CIG Use Case Text Description

Name			**Download Shared *CompGuide* CIG.**
Purpose			Receive *CompGuide* CIG OWL files modified by Health Professionals into *CompGuide* server repository.
Pré-Condition			Access to a Internet connection.
Pós-Condition			None.
Super Use Case			None.
Event Flow			**Normal behavior.**
		1.	System opens TCP server connection;
		2.	System awaits for Health Professionals requests;
		3.	A request arrives to the System and is accepted;
	VAL	4.	System verifies if folder name with requested IP Address exists;
		5.	System creates folder in requested IP Address folder with temporal date plus hour name;
		6.	System establishes a connection with the Health Professional and downloads the zip file to the newly created folder;
		7.	System creates .txt file and writes all input data into the created file;
		8.	System completes the transfer process;
		9.	Return to 2;
	ALT		**Alternative 4a [Folder doesn't exist]**
		1.	Create folder with requested IP Address name;
		2.	Return to 5.

Table 8.: Access files Use Case Text Description

Name			**Access *CompGuide* server repository files.**
Purpose			Manage and access CIG *CompGuide* files shared by Health Professionals.
Pré-Condition			Have access to *CompGuide* server repository.
Pós-Condition			None.
Super Use Case			None.
Event Flow			**Normal behavior.**
	UD	1.	Admin explores *CompGuide* server repository directory;
	UD	2.	Admin unzip the received file;
	UD	3.	Admin manages the received files by verifying log-file and analysing the changes made in *CompGuide* CIG file or by opening the received OWL file in the *Protégé Desktop* application;

Table 9.: Update *CompGuide* files Use Case Text Description

Name			**Update Git files in master branch repository.**
Purpose			Admin updates the files allocated in the *CompGuide* master branch repository in Git platform.
Pré-Condition			Have a stable connection with Git platform.
Pós-Condition			None.
Super Use Case			None.
Event Flow			**Normal behavior.**
	UD	1.	Admin enters into the Git platform;
	OUT	2.	System asks for login credentials;
	INP	3.	Admin inputs the required credentials;
	IVAL	4.	System validates login;
	OUT	5.	System gives access to all repositories linked into the account;
	UD	6.	Admin enters in Git *CompGuide* repository;
	OUT	7.	System shows available management options;
	UD	8.	Admin access master branch and uploads the OWL files into the repository;
		9.	System begins upload process;
	OUT	10.	System informs the Admin that the upload process is completed;
	ALT		**Alternative 4a[Failed login]**
	OUT	1.	System informs Admin login credentials are not correct;
		2.	Return to 2;
	ALT		**Alternative 9a [Upload files isn't completed]**
	OUT	1.	System informs Admin that upload process wasn't completed successfully;
		2.	Return to 8;

B

SEQUENCE DIAGRAMS

This Annex contains the sequence diagrams of the system features. The system features shown in these diagrams can be categorized in four categories: creation of individuals, edition of individuals, deletion of individuals and sharing of the *CompGuide* ontology. Only the sequence diagrams most important to the application are shown in this list of figures (figures 39-53). All these sequence diagrams are properly explained in sub-chapter 4.4.

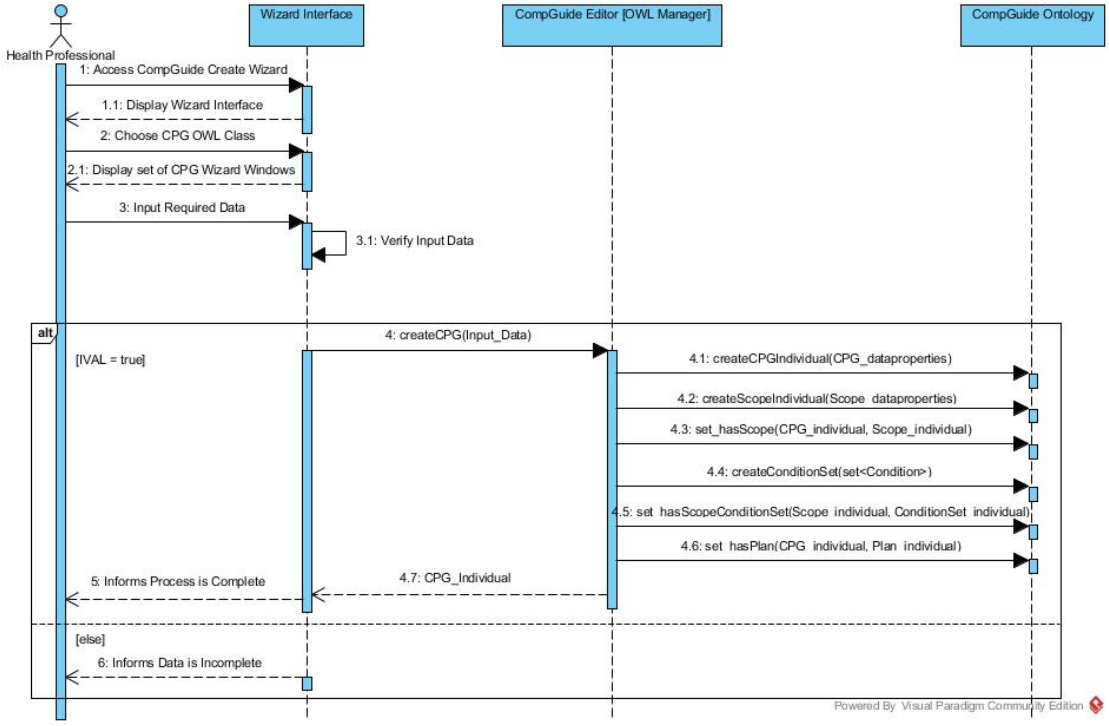


Figure 39.: Create CPG Sequence Diagram.

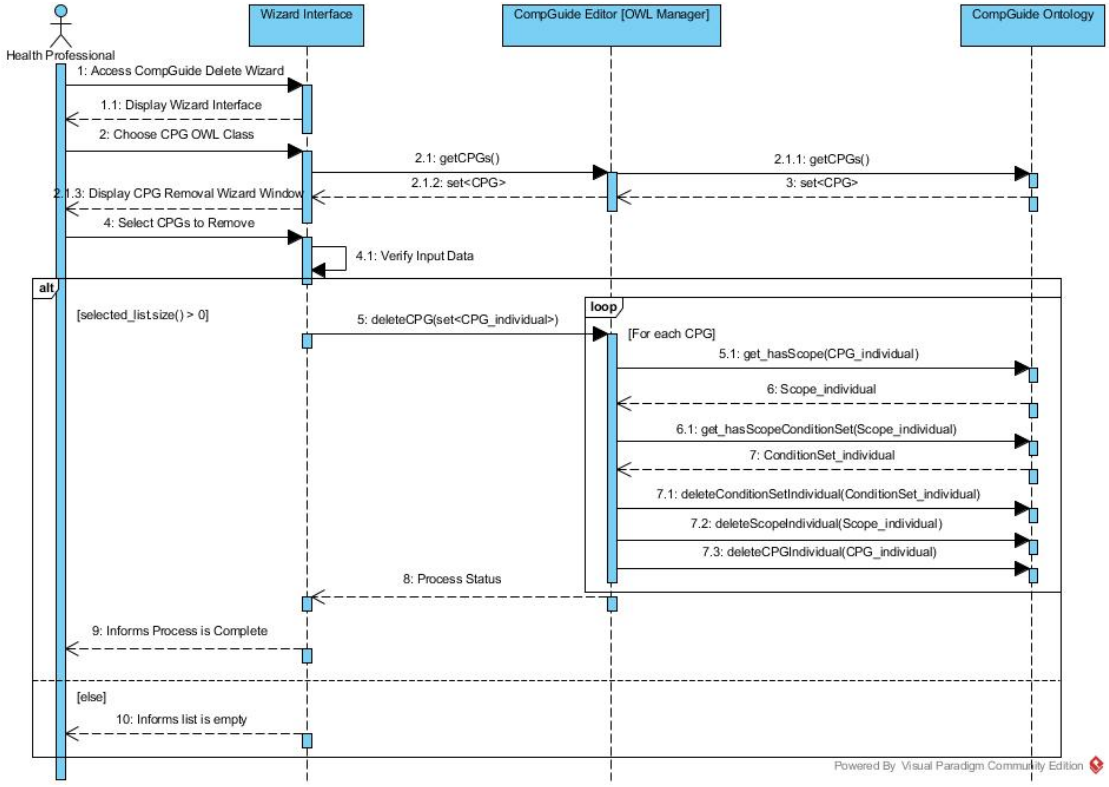


Figure 40.: Delete CPG Sequence Diagram.

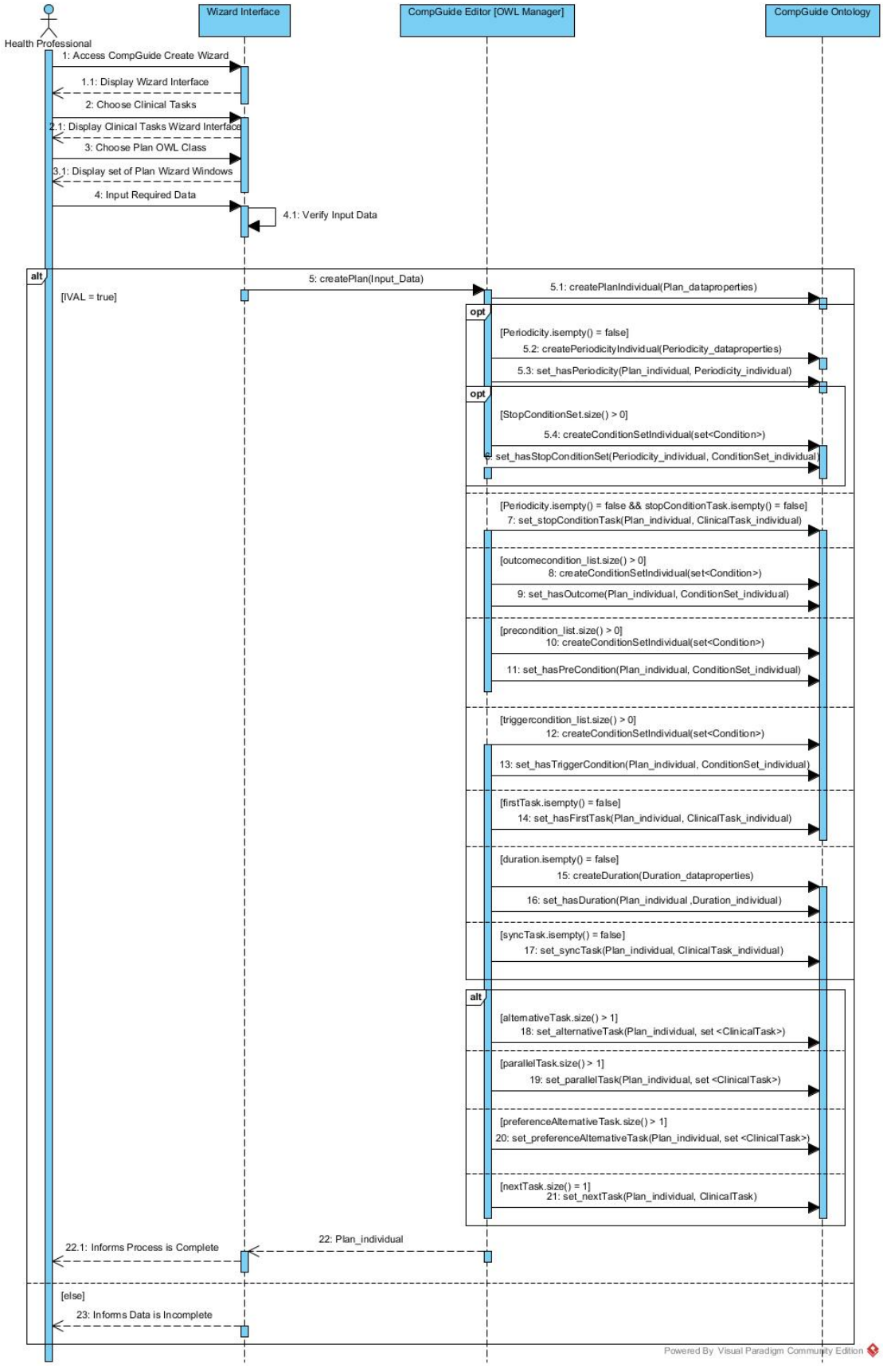


Figure 41.: Create Plan Clinical Task Sequence Diagram.

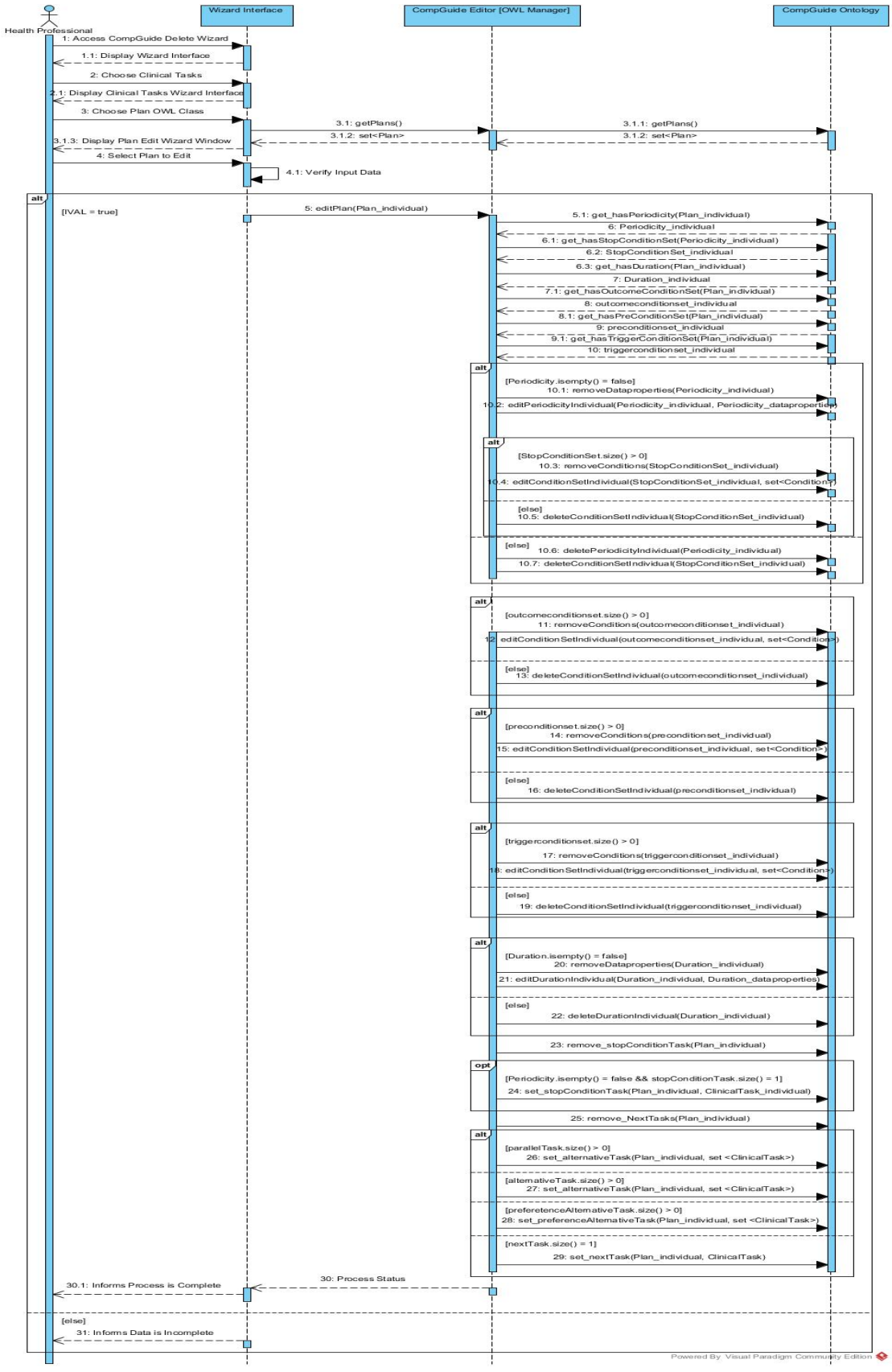


Figure 42.: Edit Plan Clinical Task Sequence Diagram.

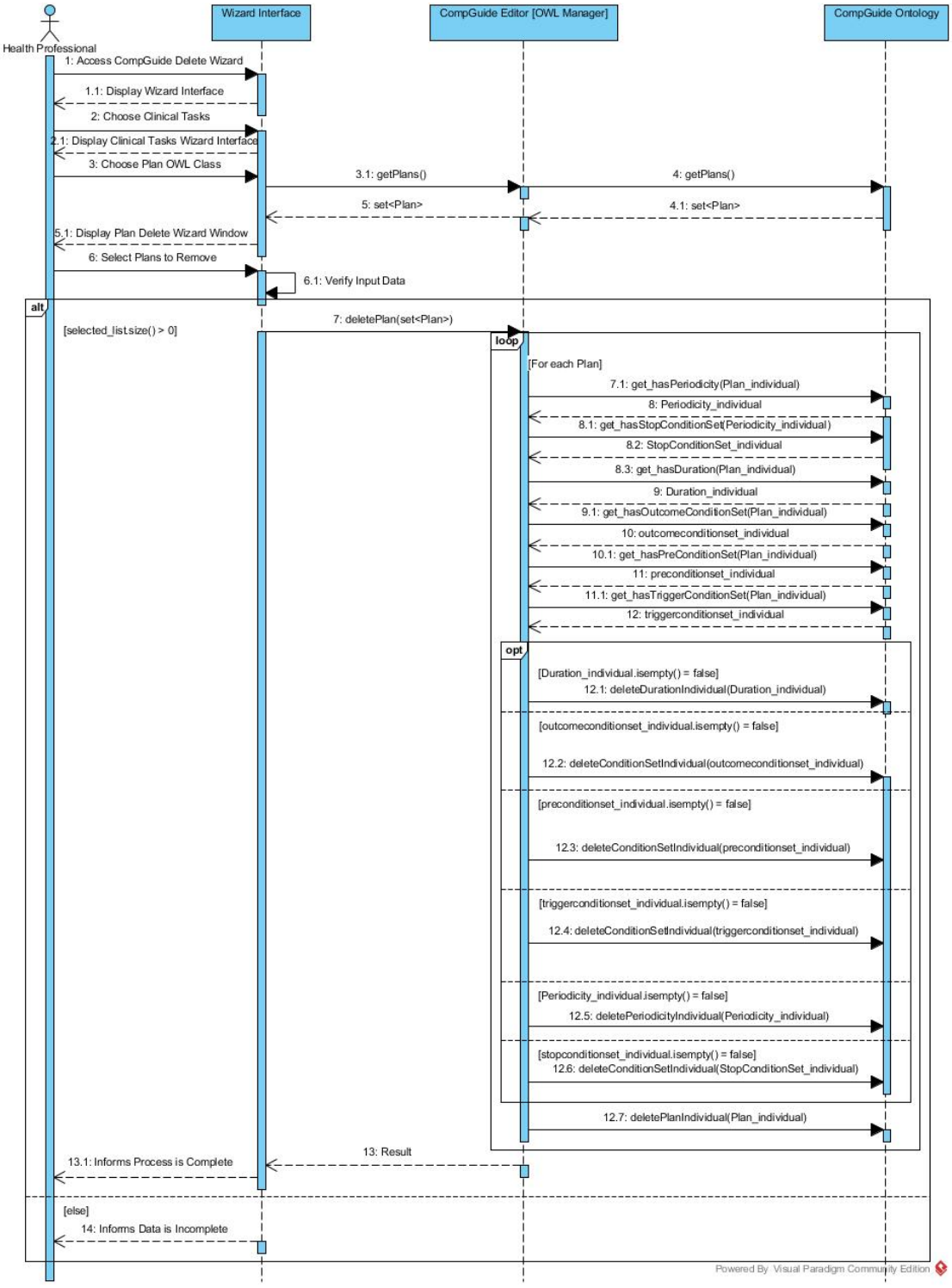


Figure 43.: Delete Plan Clinical Task Sequence Diagram.

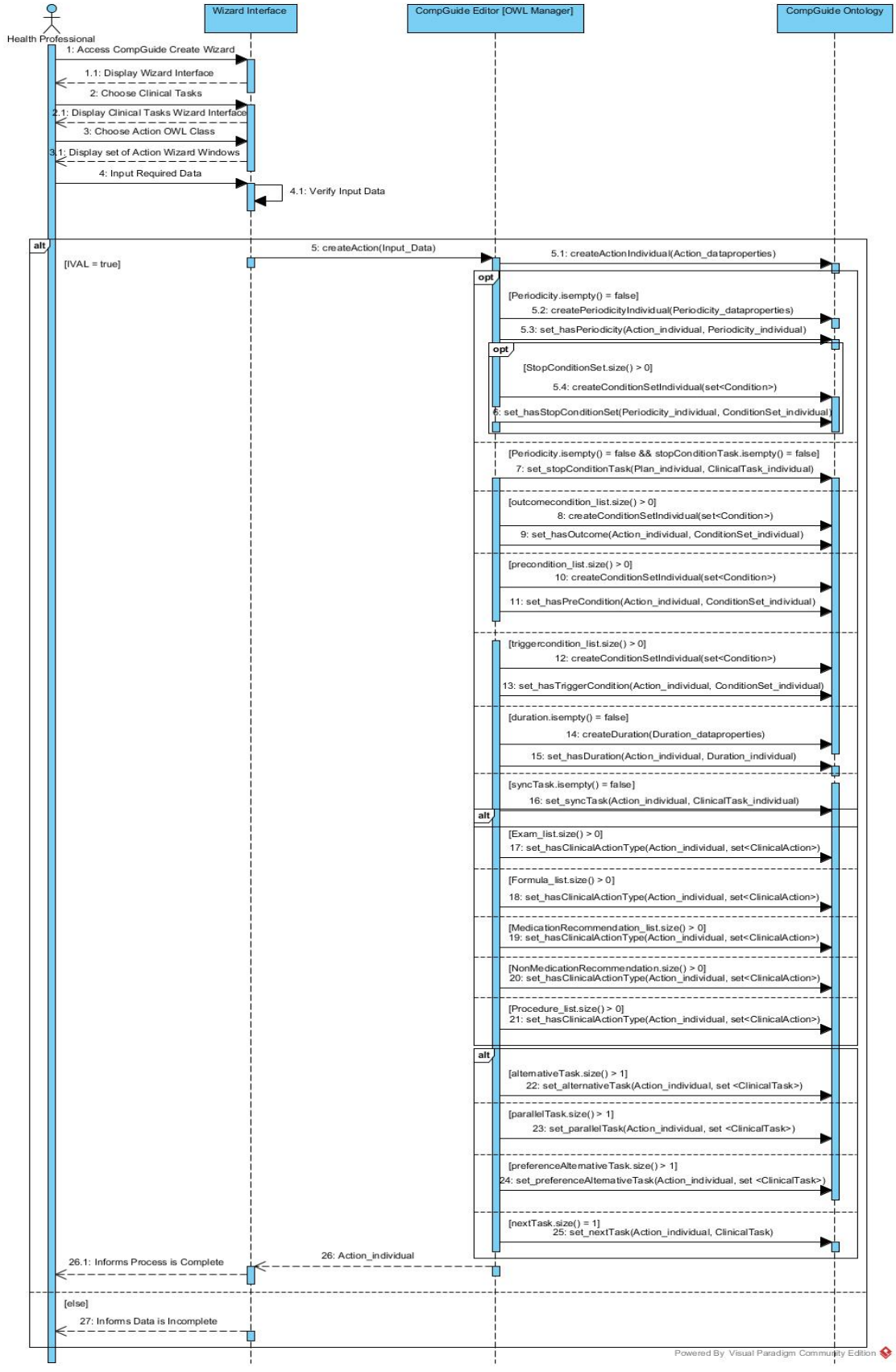


Figure 44.: Create Action Clinical Task Sequence Diagram.

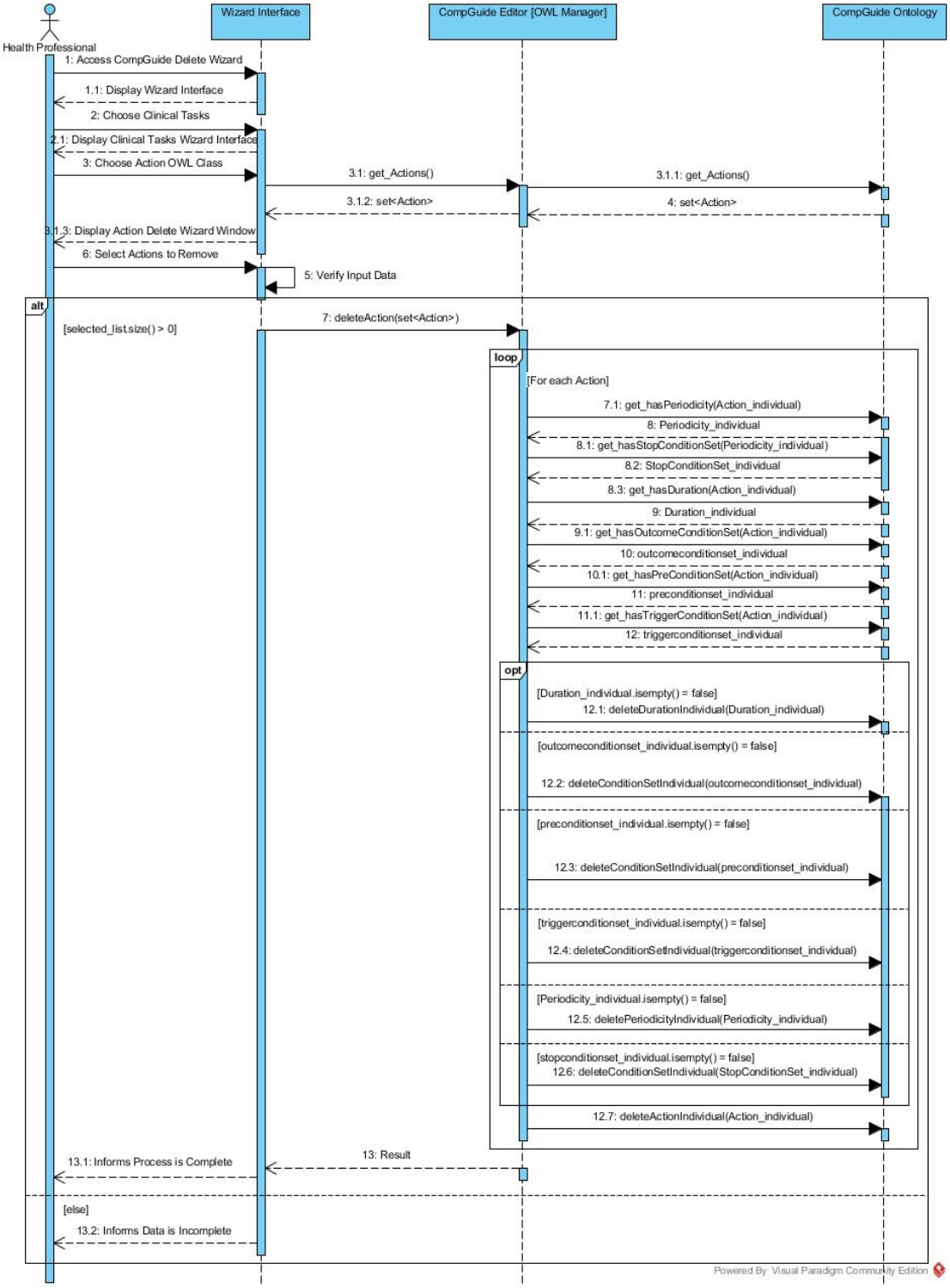


Figure 45.: Delete Action Clinical Task Sequence Diagram.

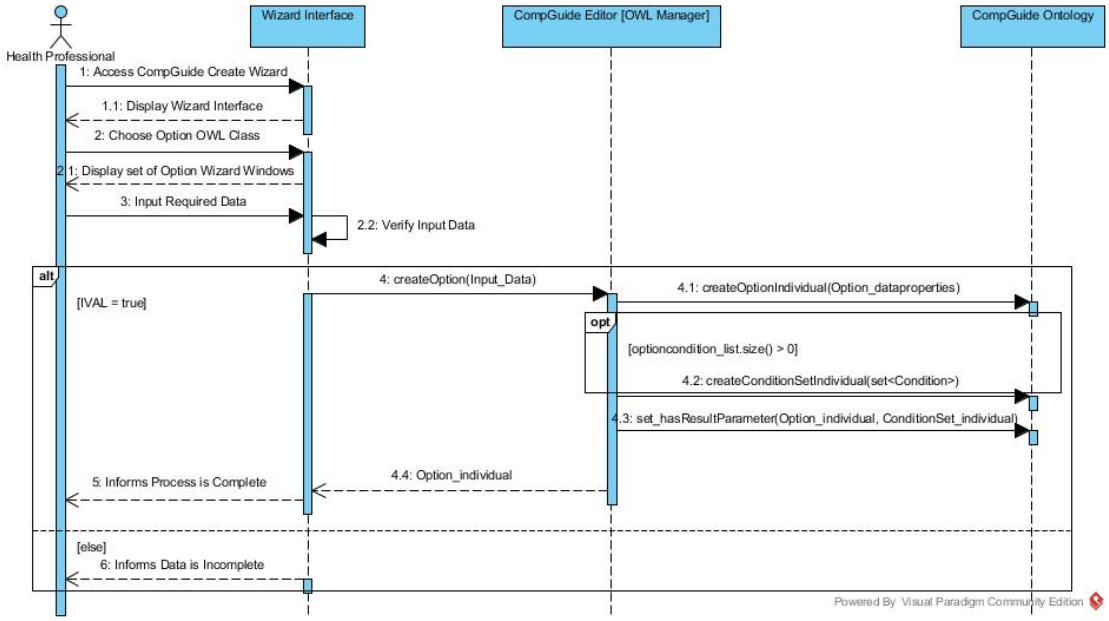


Figure 46.: Create Option Sequence Diagram.

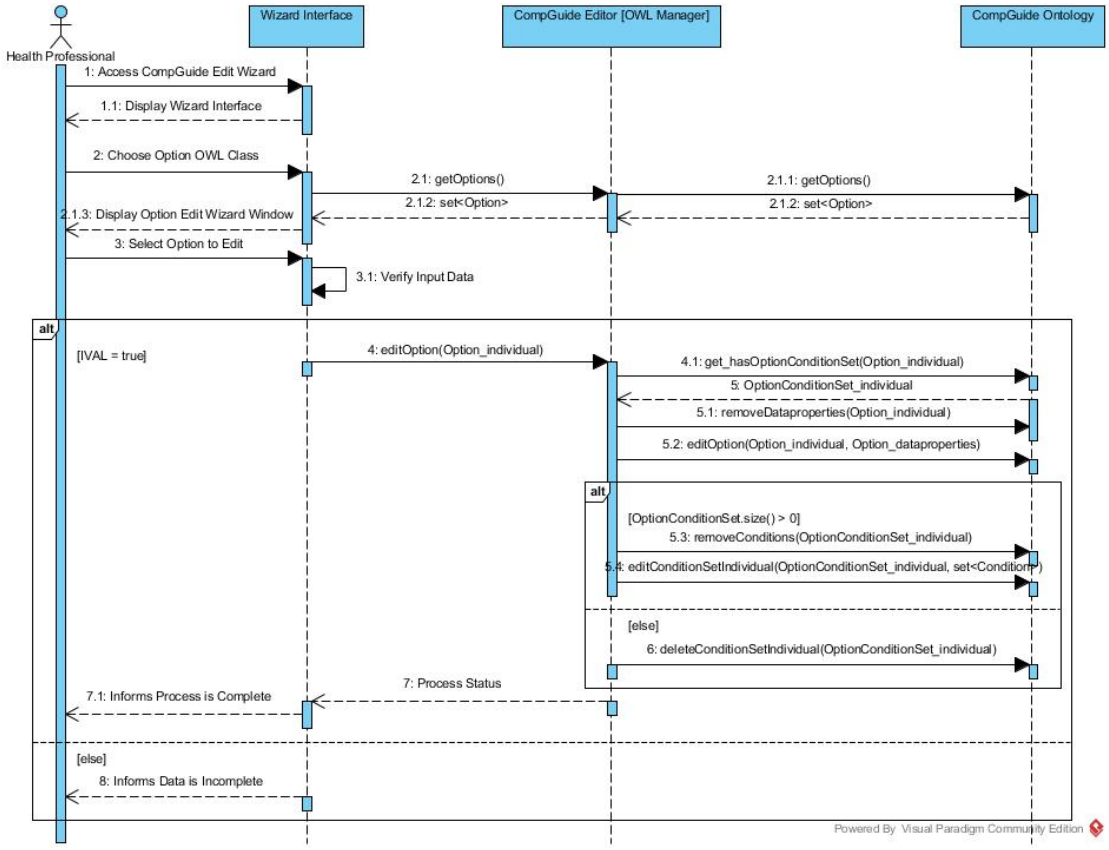


Figure 47.: Edit Option Sequence Diagram.

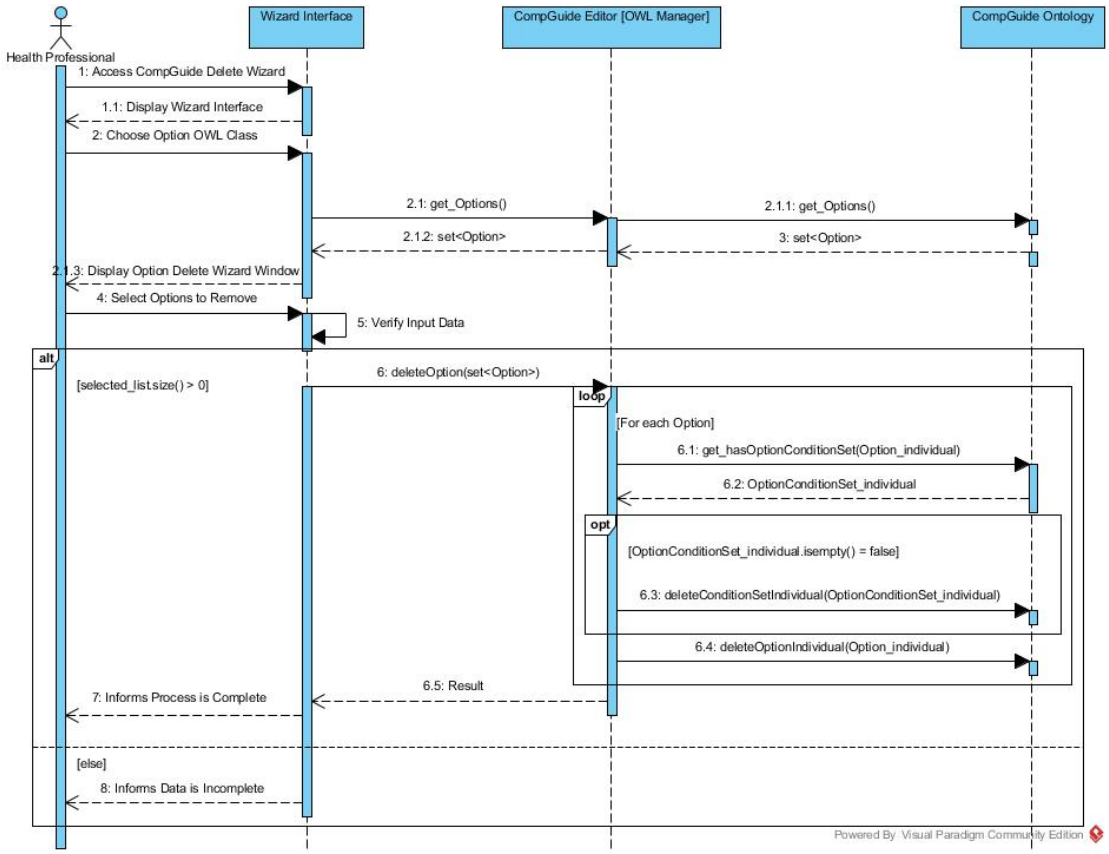


Figure 48.: Delete Option Sequence Diagram.

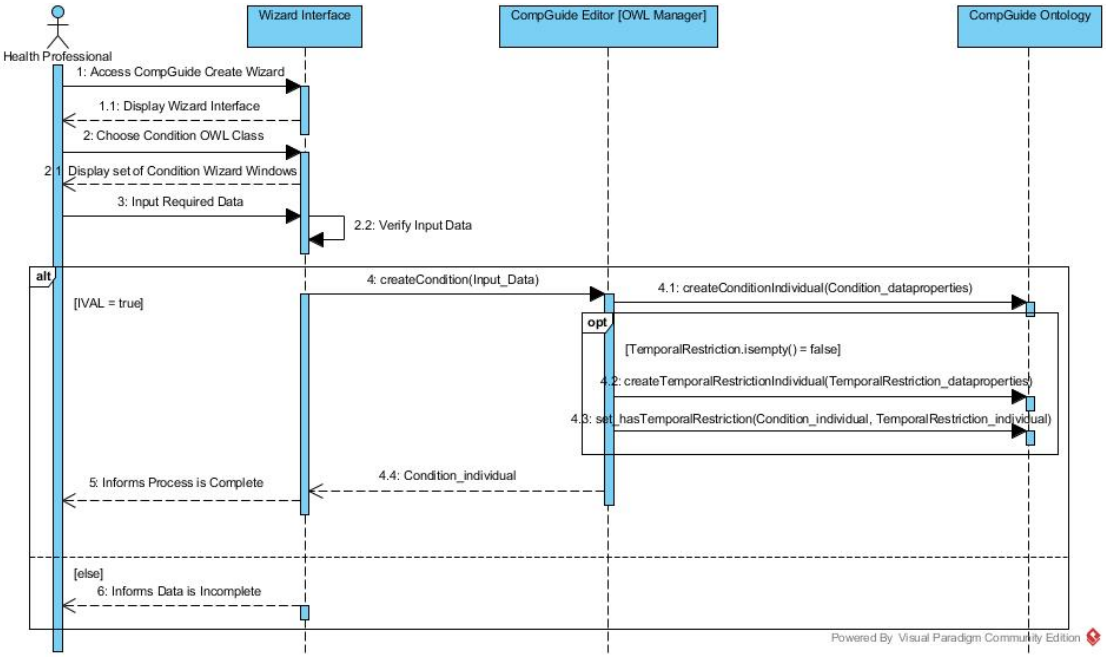


Figure 49.: Create Condition Sequence Diagram.

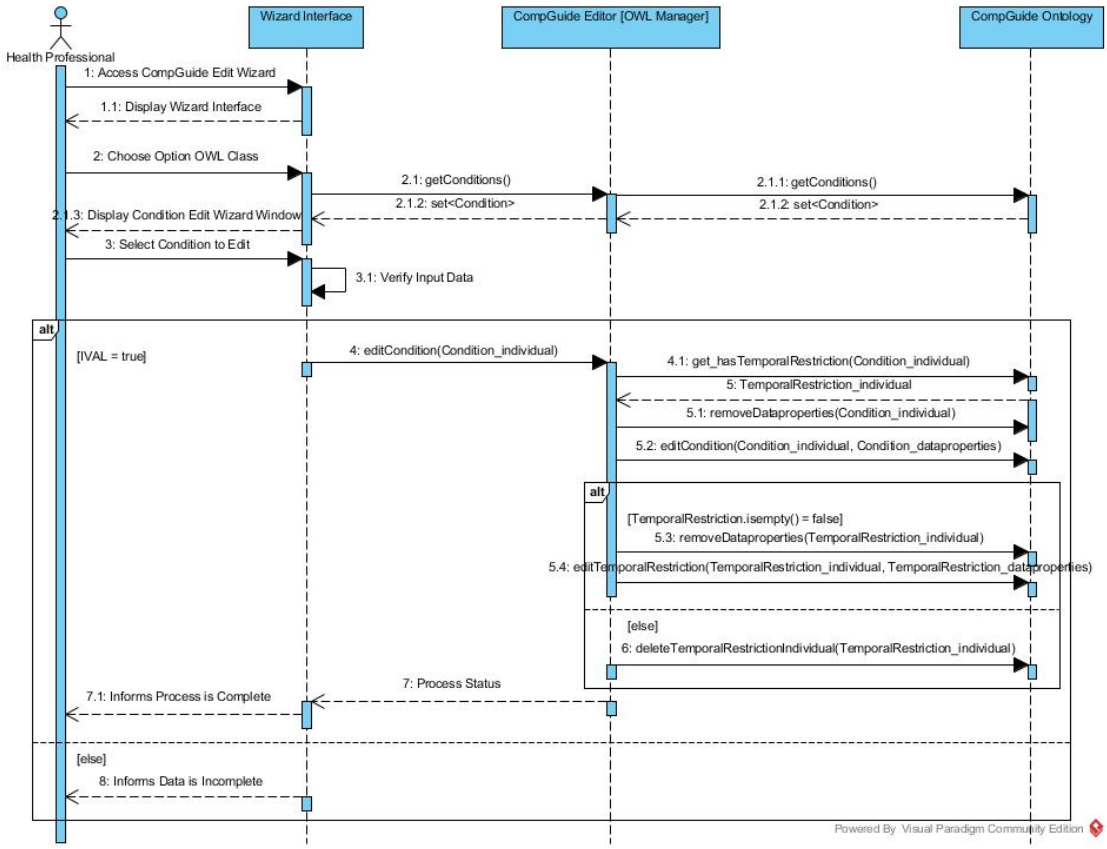


Figure 50.: Edit Condition Sequence Diagram.

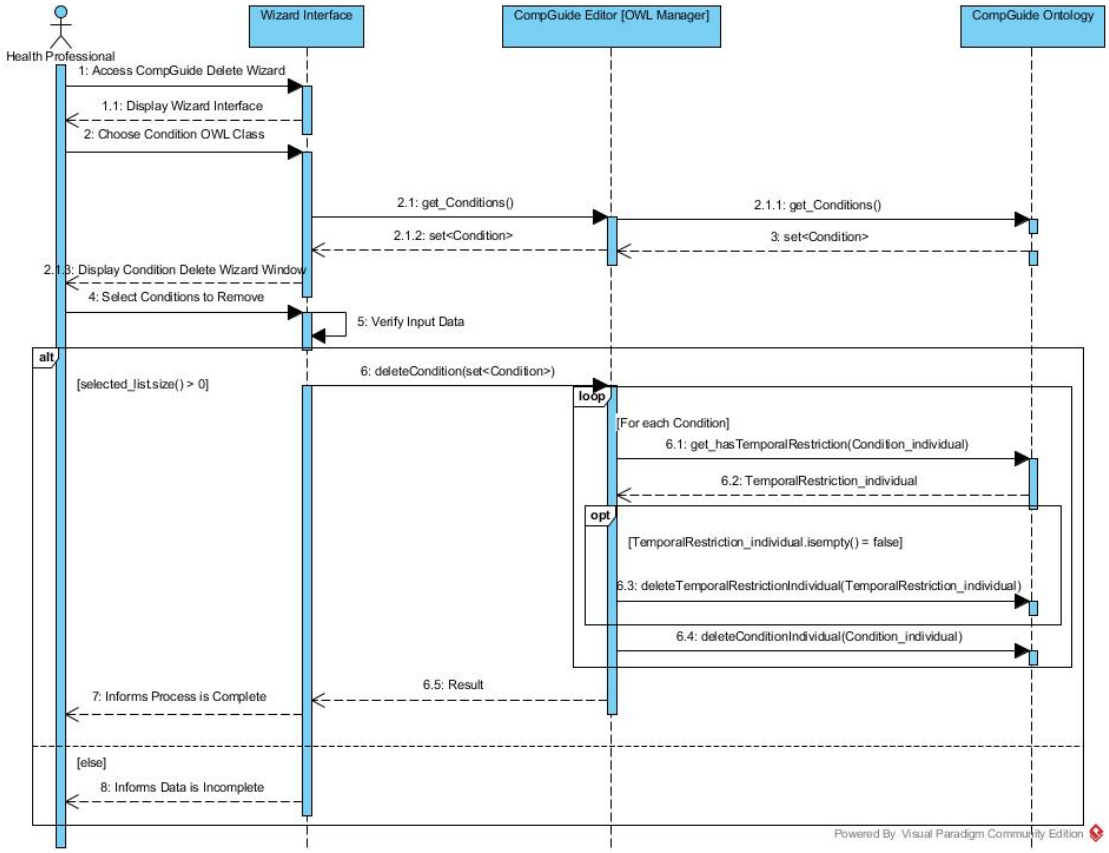


Figure 51.: Delete Condition Sequence Diagram.

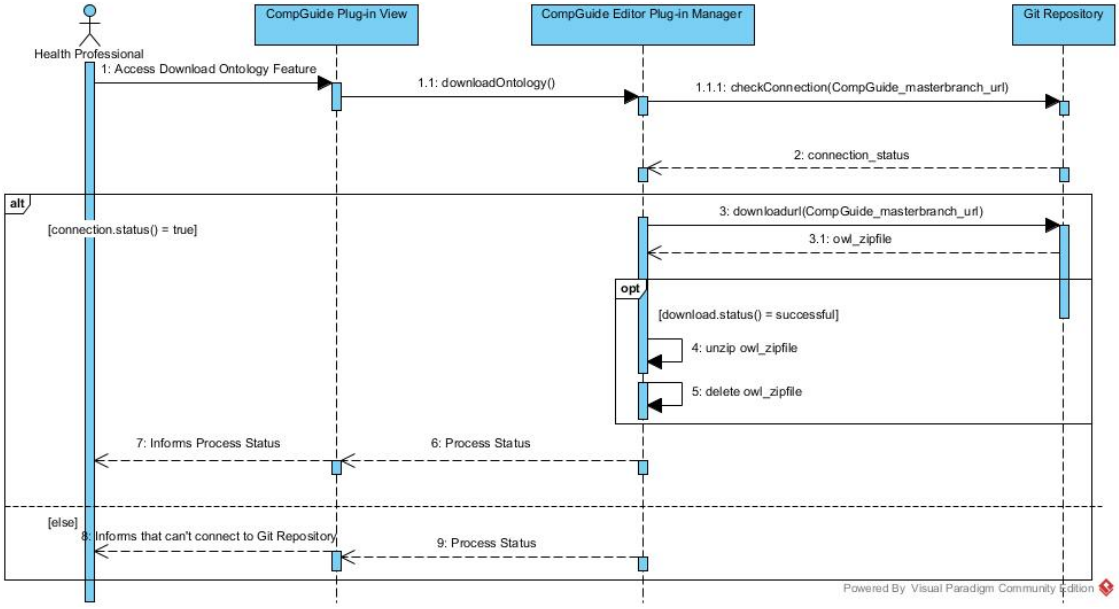


Figure 52.: Download *CompGuide* ontology Sequence Diagram.

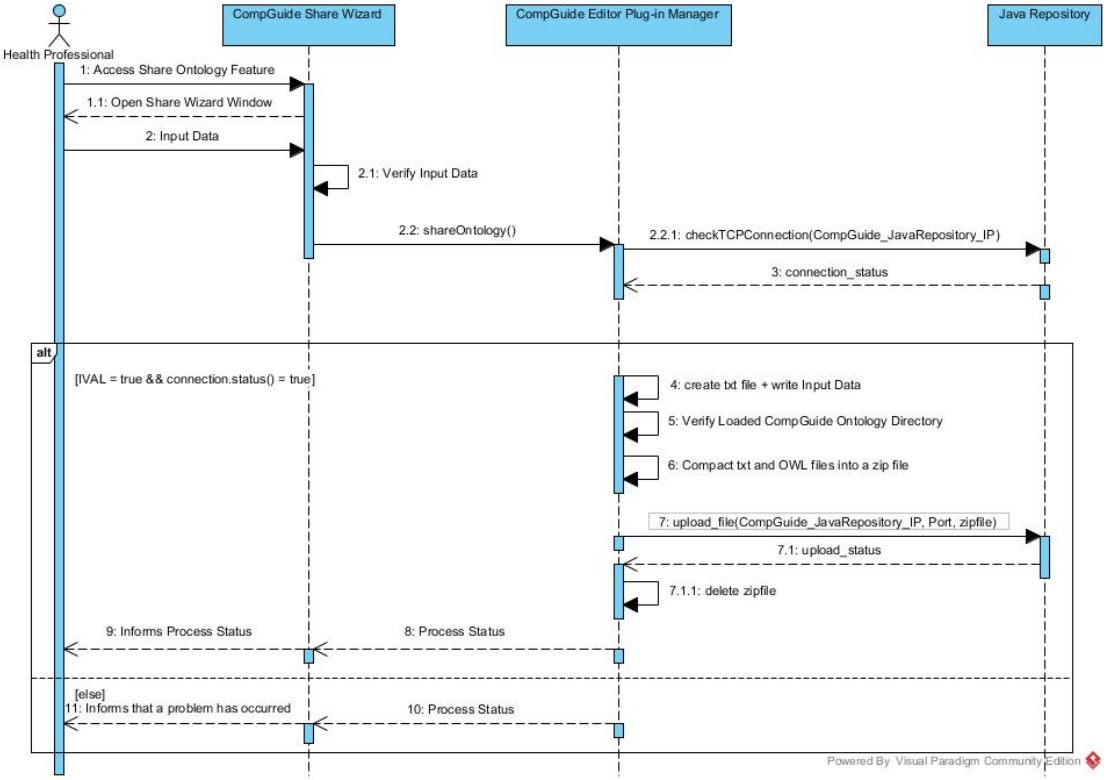


Figure 53.: Share *CompGuide* ontology Sequence Diagram.

C

INTERFACE FIGURES

This Annex contains the figures of the system interfaces. The list of figures (figures 54-64) are properly explained in sub-chapter 4.5.

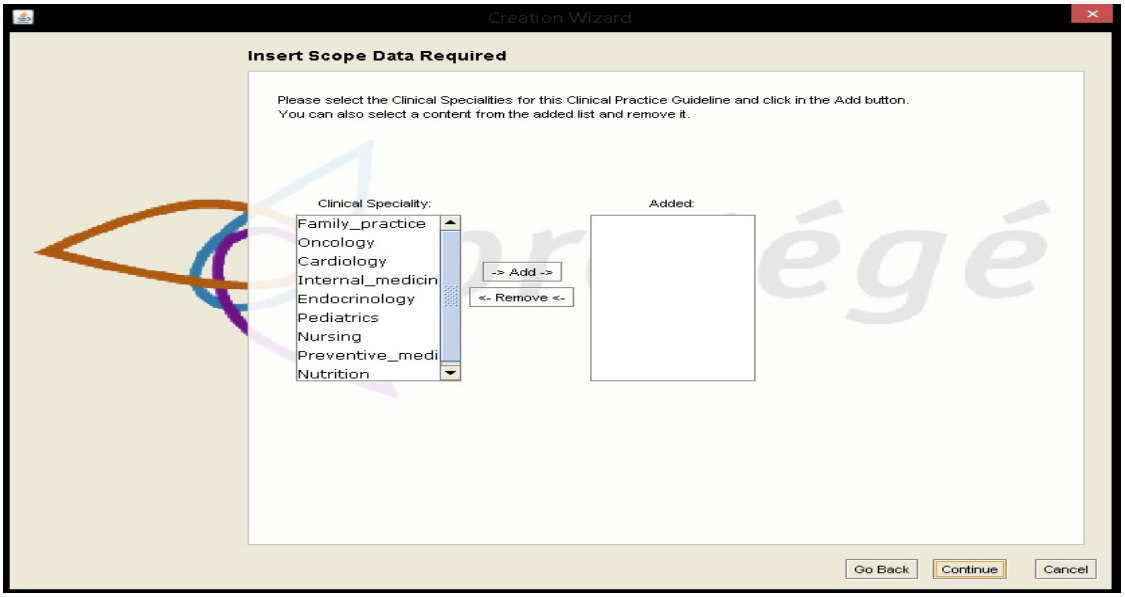


Figure 54.: CPG Scope - Clinical Specialties Selection Window.

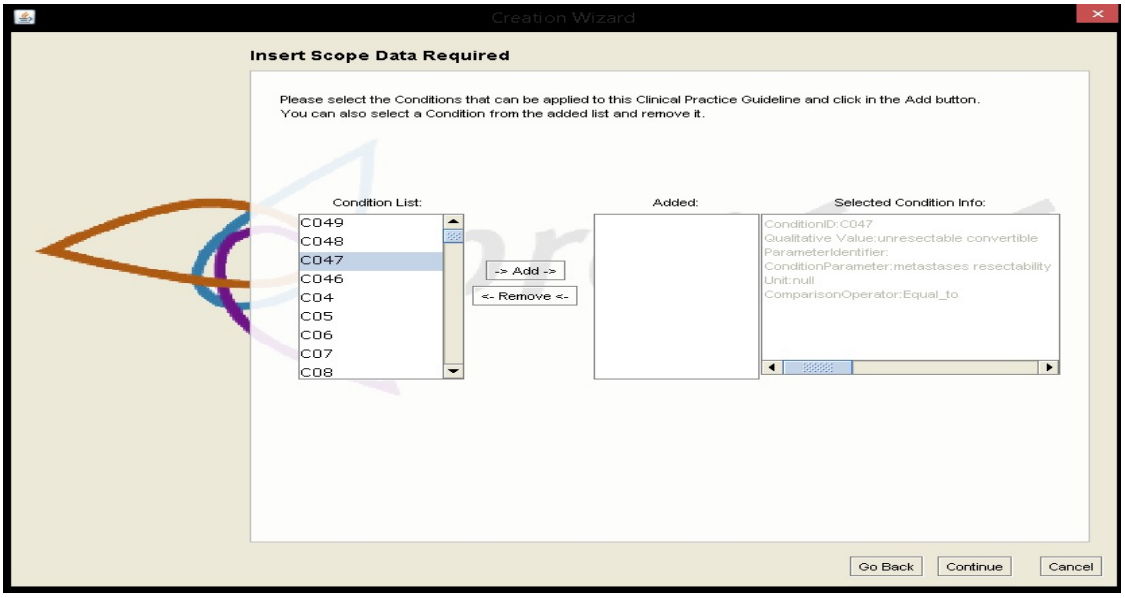


Figure 55.: CPG Scope - Conditions applied in CPG Selection Window.

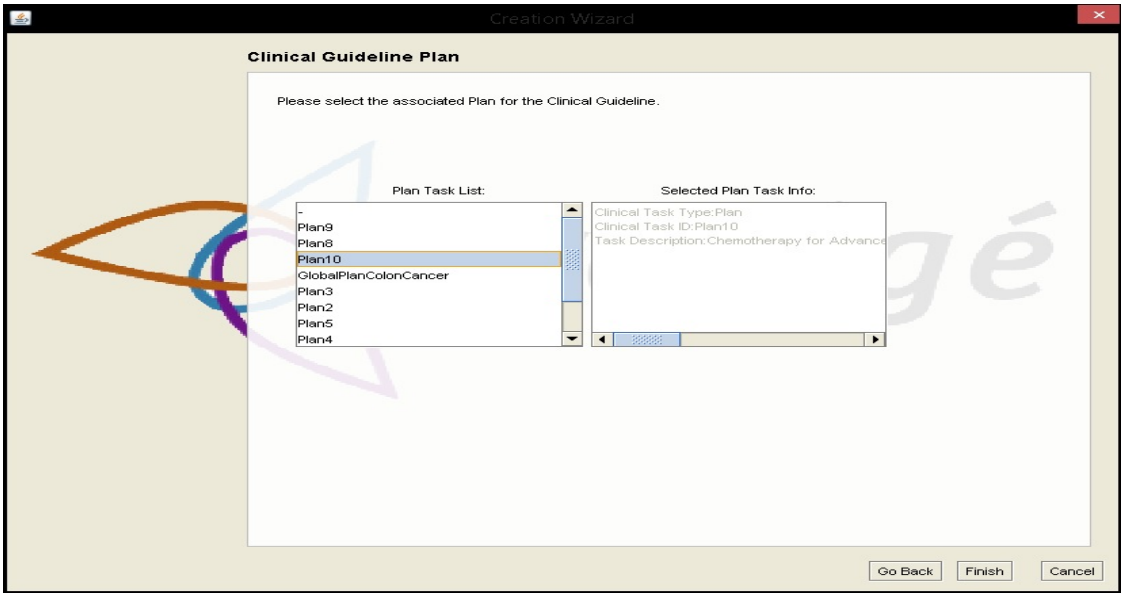


Figure 56.: CPG Plan - Plan Selection Window.

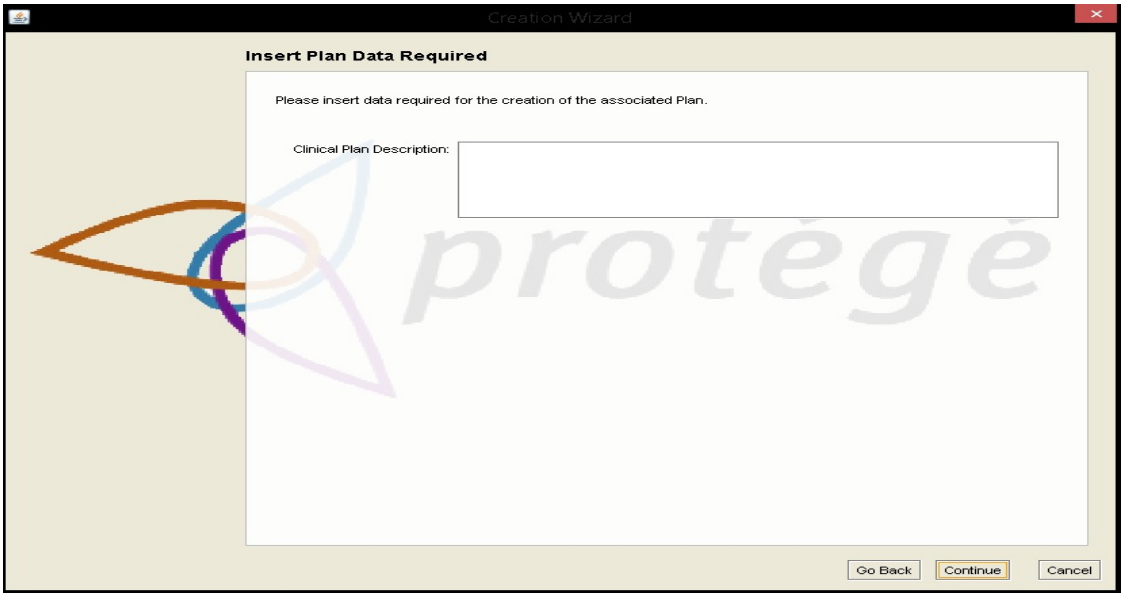


Figure 57.: Clinical Tasks - Description Window.

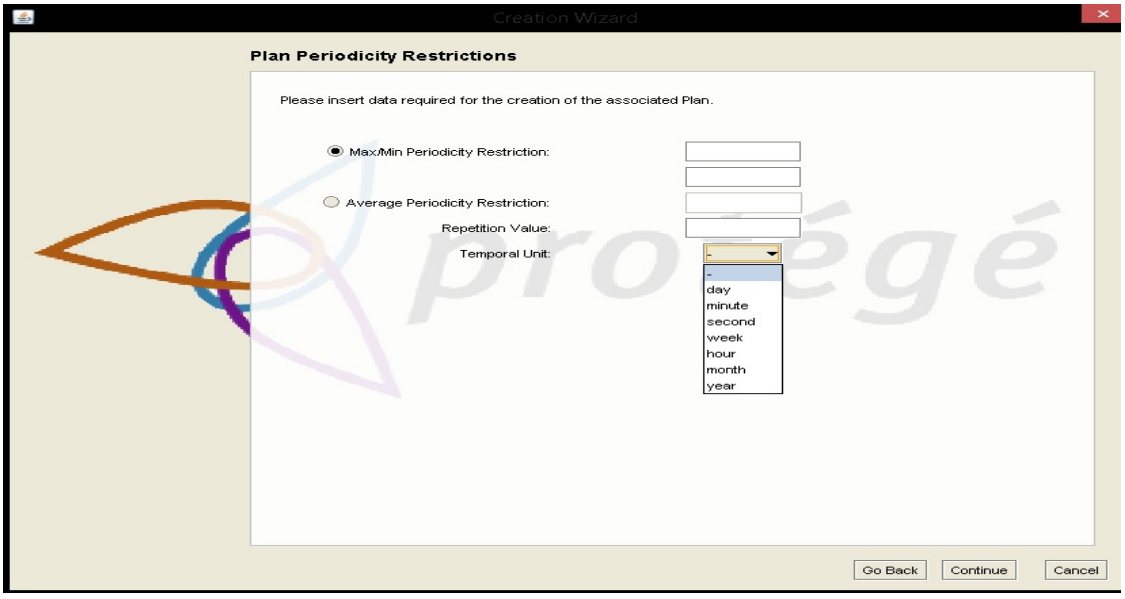


Figure 58.: Action/Plan Periodicity - Periodicity Restriction Values Window.

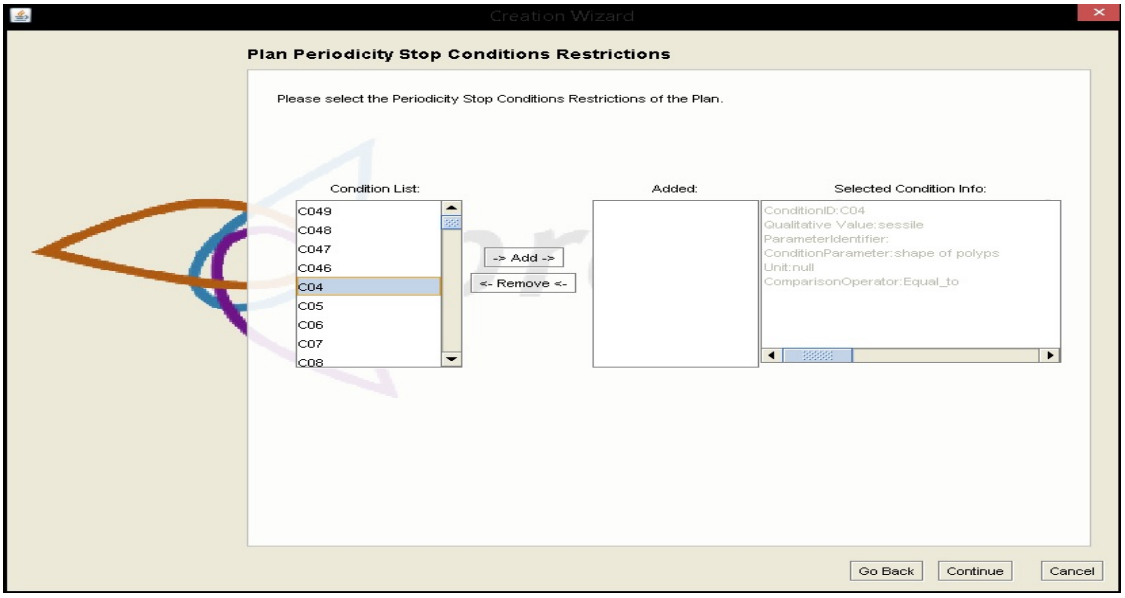


Figure 59.: Action/Plan Periodicity - Periodicity Restriction Stop Conditions Window.

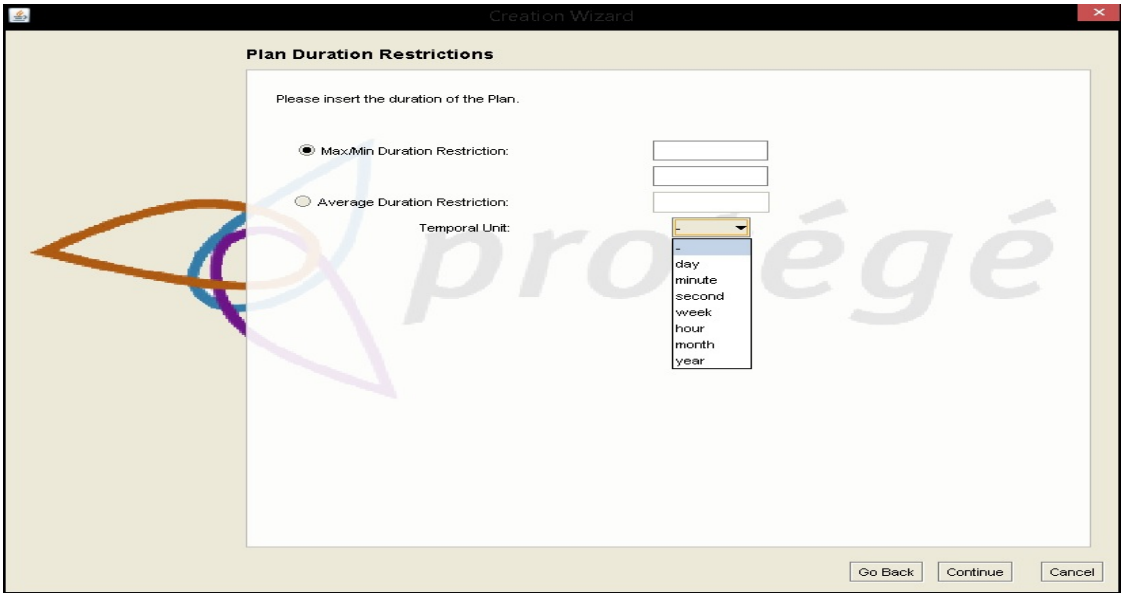


Figure 60.: Action/Plan Duration - Duration Restrictions Values Window.

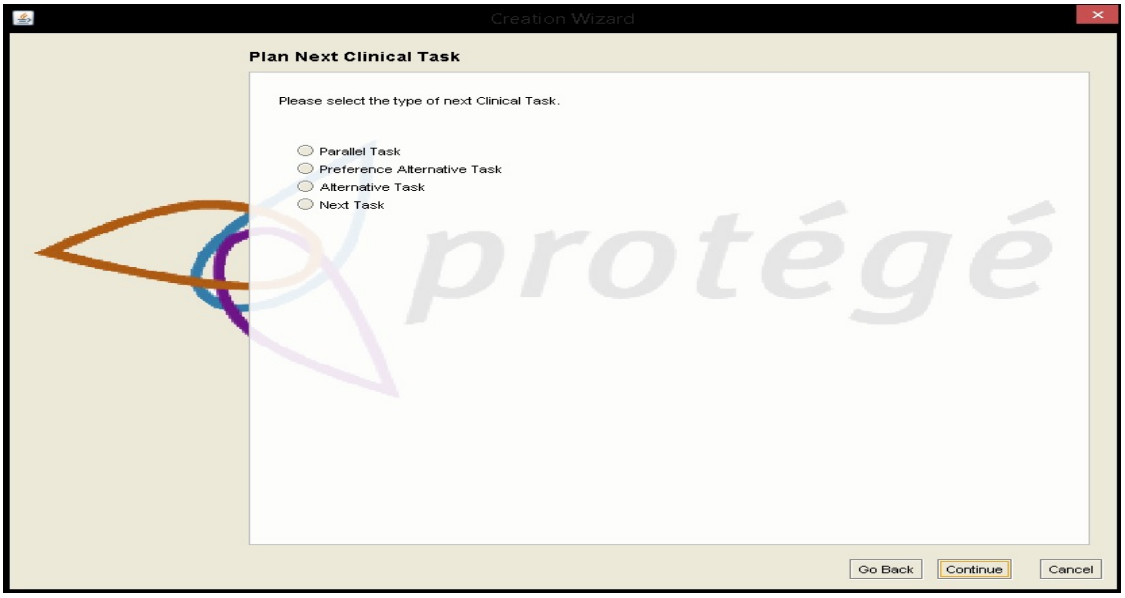


Figure 61.: Clinical Task - Next Clinical Task Type Selection Window.

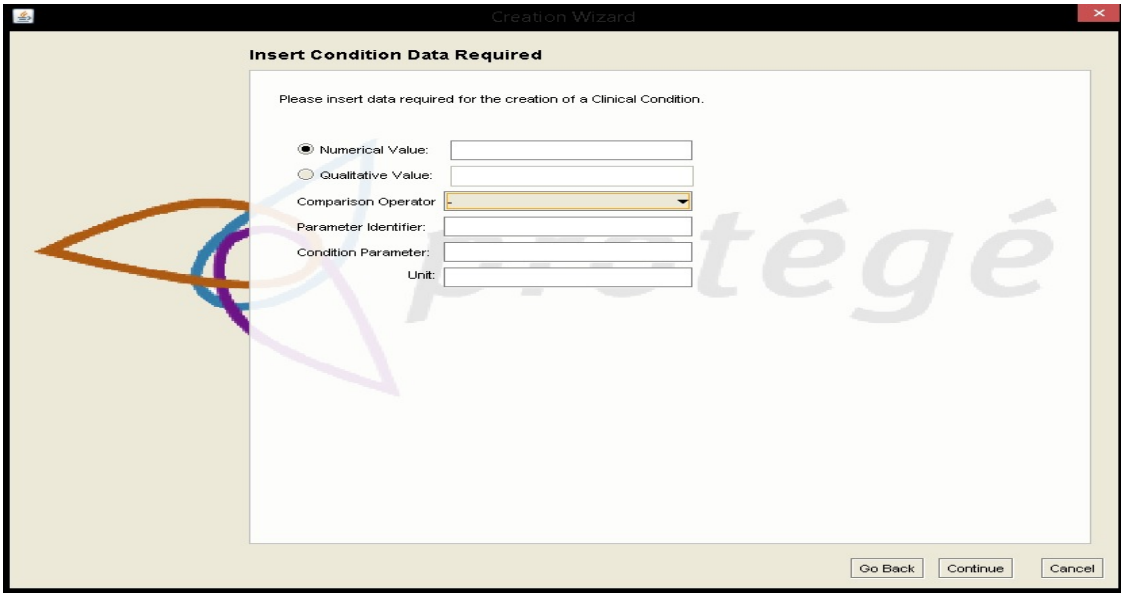


Figure 62.: Clinical Condition - Clinical Restriction Values Window.

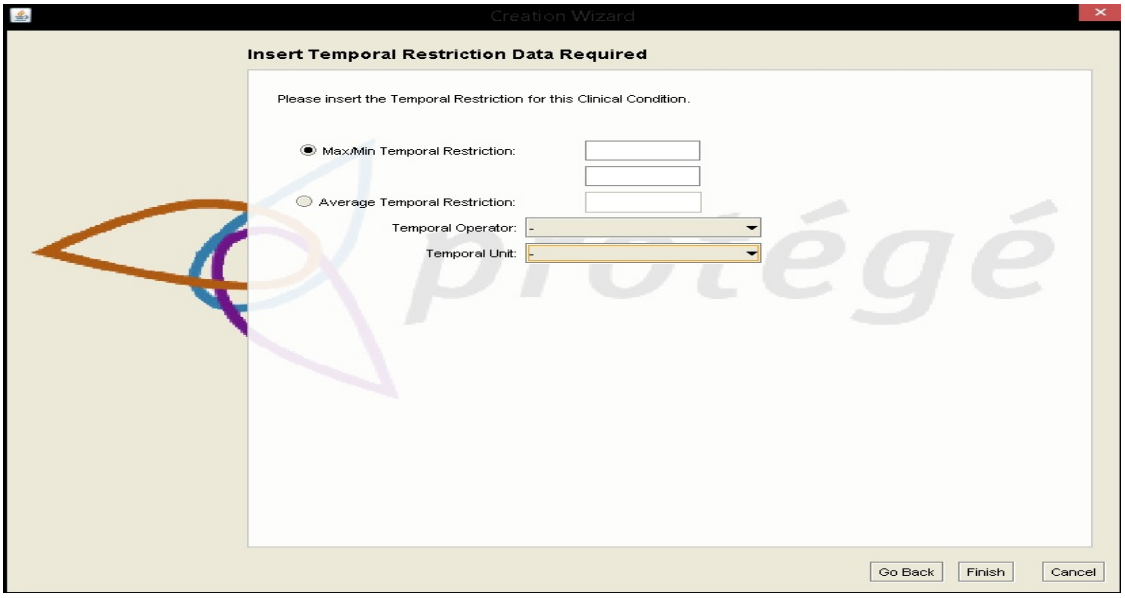


Figure 63.: Clinical Condition - Clinical Temporal Restriction Values Window.

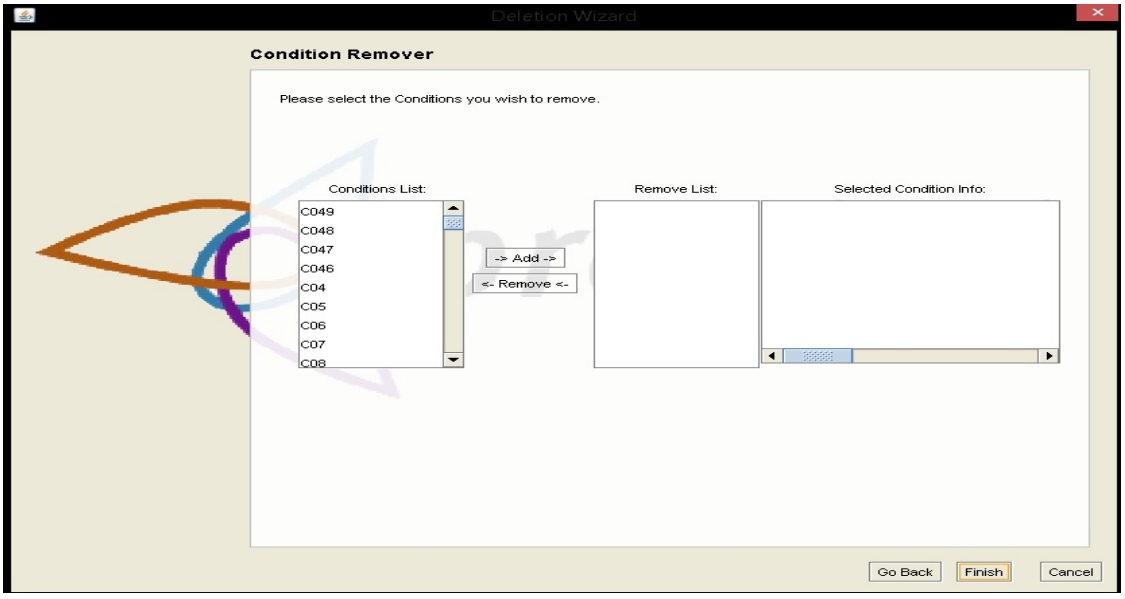


Figure 64.: Clinical Condition - Deletion of Condition individuals Window.