POLITECNICO DI MILANO

Scuola di Ingegneria Industriale e dell'Informazione

Corso di Laurea Magistrale in Ingegneria Biomedica

Towards a Triplet Extraction Tool for Next Generation Biomedical Ontologies

Relatore: Prof. Stefano BONACINA

Tesi di Laurea di:

Andrea CONDORELLI Matr. 786630

Anno Accademico 2012 - 2013

Table of contents

Figure List

Table List

Sommario

Introduzione, Background e Scopi

Lo scopo del lavoro è stato quello di sviluppare un applicativo software per creare automaticamente ontologie in ambito biomedico a partire da un testo - di ambito biomedico - in linguaggio naturale. Le ontologie in ambito biomedico sono fondamentali per favorire la condivisione delle conoscenza tra gruppi di ricerca differenti.

In ambito informatico, **un'ontologia** è una rappresentazione formale di conoscenza attraverso un elenco di "statement" o triplette [18]: ogni tripletta è formata da due concetti, detti Domino e Range, e dalla Relazione che li lega (e.g. "Il Rene", Dominio, "è un", Relazione, "Organo", Range). L'insieme delle triplette di un'ontologia costituisce l'universo dell'ontologia, che deve essere coerente con se stesso (pena la creazione di ontologie inconsistenti), ma potrebbe contenere informazioni non condivisibili da un osservatore esterno [18]. Bisogna sottolineare questo aspetto, che è abbastanza contro intuitivo: ogni tripletta è considerata vera in quanto tale, anche se le informazioni che contiene non sono corrette (non è "compito" dell'ontologia verificare se le triplette inserite sono corrette o meno, ma dello sviluppatore dell'ontologia) [18]. Ad esempio in un'ontologia potrebbe essere presente lo statement "Il cuore" "è formato da" "quattro ventricoli": nell'universo di questa ontologia, questo statement è "grammaticalmente" corretto, palesemente contro la realtà; qualora fosse presente però un secondo statement che afferma che "Il cuore" "è formato da" "due ventricoli", dovendo essere entrambi veri, avremmo un'ontologia inconsistente. Abbiamo precedentemente usato la parola "formale", relativamente alla definizione di ontologia, proprio perché ogni ontologia è basata su una **logica formale** [15]. Una delle caratteristiche principali delle ontologie, infatti, è quella di permettere azioni di "reasoning" (ragionamento): ogni ontologia è in grado (quasi) sempre di indicare se a partire dall'universo dell'ontologia sia possibile implicare una determinata conclusione [15]. Il "quasi" tra parentesi è legato alla specifica logica usata in un'ontologia: esistono infatti logiche "decidable", ossia che sono sempre in grado di dire se una conclusione sia implicabile o meno a partire da una serie di statement in un numero finito di operazioni, o "semi-decidable", ossia in grado di implicare sempre una conclusione solamente quando implicabile in un numero finito di operazioni [14]. Logiche

che non sono "decidable" non assicurano la convergenza della risposta in un numero finito di iterazioni: viene di solito usata come logica la Logica Descrittiva in quanto essa è "decidable". Questo problema è noto come "**decidability problem**" [14]. Le ontologie possono essere usate in molteplici modi: uno di questi è aggiungere le capacità espressive e computazionali di una logica formale a una base strutturata di informazioni [16]. In questo modo, l'ontologia è in grado di "creare conoscenza", nel senso che è in grado di creare nuove triplette che nessuno ha esplicitamente aggiunto nel suo "universo" (e.g. a partire dagli statement "Il cuore" "fa parte" "del corpo" e "I ventricoli" "fanno parte" "del cuore", qualora "fare parte" sia una relazione "transitiva", l'ontologia è in grado di affermare che "I ventricoli" "fanno parte" "del corpo") [31]. Un altro modo molto comune di utilizzare le ontologie è quello di usarle come **classificazione di domini**: ad esempio, al posto di usare un'etichetta arbitraria per indicare una specifica malattia, un medico può considerare l'uso del "International Classification of Diseases" (ICD9) [22]. Inoltre, è possibile associare i concetti ai relativi sintagmi (i.e. un sintagma è un gruppo di parole riferito a un solo concetto principale) [14]: per permettere la cosiddetta "**ricerca semantica**". Un'ultima possibilità offerta dalle ontologie è la **rappresentazione grafica** del loro universo: ogni ontologia può essere considerata un grafo e per questo, i suoi concetti possono essere rappresentati come nodi collegati da frecce che rappresentano le relazioni tra di essi.

Fino ad oggi, la quasi totalità dei motori di ricerca su internet esegue "**ricerche a identità**": digitando il termine da cercare, il motore non fa altro che estrapolare tutti i risultati che contengono esattamente quella stringa (con approcci più o meno raffinati). Per cui, se volessi cercare tutte le informazioni relative a una malattia, dovrei semplicemente inserire il nome della stessa in un motore di ricerca. In questo modo, però, avrei la certezza di non trovare tutte le informazioni disponibili: altri ricercatori potrebbero riferirsi alla stessa malattia con diciture (stringhe) diverse. Anche usando la lista dei possibili sinonimi perderei molti articoli: un qualsiasi scrittore può riferirsi a un concetto senza citarlo esplicitamente. Se però fossi in grado di cercare tutti gli articoli relativi al "concetto" in una base di dati "annotata" (i.e. una base di dati in cui a ogni testo sono associati i vari concetti), questa ricerca andrebbe a buon fine: esistono vari annotatori biomedici che eseguono il compito molto bene [1]. E' proprio per questo motivo che in campo biomedico sta aumentando la popolarità delle ontologie (e.g. la "Gene Ontology" [20] è un'ontologia comunemente usata dai genetisti per trovare tutte le informazione a disposizione relative a uno specifico gene). Per lo stesso motivo, possiamo affermare che le ontologie facilitino la "**interoperabilità**" tra basi di conoscenza eterogenee: non importa quale etichetta venga usata da due centri di studio per riferirsi alla stessa patologia finché la stessa etichetta è associata allo stesso concetto, che è intrinsecamente univoco. Qualora questi due centri usassero due ontologie diversi non sarebbe comunque un problema [81]: è infatti possibile **"mappare" ontologie diverse** (i.e. con "mappare" ci si riferisce all'operazione di riconoscimento di concetti equivalenti in ontologie diverse) [12]; l'operazione di "mapping" viene eseguita da una serie di strumenti semi-automatici: per quanto attualmente questi strumenti mostrino evidenti limiti (molto spesso sono più semi-manuali che semi-automatici), le relazioni di equivalenza tra ontologie diverse sono spesso disponibili sul web (e.g. lo Unified Medical Language System UMLS [7] contiene un metatesauro basato su più di 160 dizionari e sorgenti già mappate tra di loro).

Un punto di debolezza delle ontologie è costituito dalla loro **creazione**. La creazione di ontologie è un lavoro lungo e complesso, che richiede sia conoscenze relative al dominio da modellare che abilità informatiche e conoscenza della Logica Descrittiva [35]. Inoltre, non esiste nessun approccio universale alla costruzione di ontologie: esso viene ancora eseguito manualmente in base alle abilità dei singoli sviluppatori [82]. Non esistono ancora strumenti di supporto consolidati e standard per il design di un'ontologia, come potrebbe essere il diagramma entità-relazione usato nel mondo delle basi di dati.

Il nostro **scopo** è stato quello di progettare e implementare uno strumento che, sfruttando l'elaborazione del linguaggio naturale, fosse in grado di estrarre automaticamente triplette da testi in lingua inglese.

Metodi

Per progettare il sistema abbiamo approfondito lo studio della **linguistica Inglese** per comprendere in che modo venissero formate frasi grammaticalmente corrette e per comprendere quali tipologie di informazioni fosse possibile ottenere studiando la struttura della frase [54]. Abbiamo quindi approfondito le tematiche legate allo studio del "**Natural Language Processing**" (elaborazione del linguaggio naturale, NLP) [53]: abbiamo cercato di capire quali strumenti avessimo a disposizione per poter estrapolare tutta l'informazione presente in una frase scritta in lingua inglese. Abbiamo inoltre approfondito le tematiche legate alle **ontologie** per comprendere come strutturare correttamente l'informazione estratta [27].

Il sistema è stato progettato e realizzato secondo il **modello a cascata**, ma con ritorni, tipico dell'Ingegneria del Software [61]. Il nostro strumento è stato sviluppato con il linguaggio di programmazione Java e usando alcune "Application Program Interface" (API) disponibili: per implementare alcuni strumenti per l'elaborazione del linguaggio naturale è stato usato OpenNLP [64], per disegnare i grafi Jung2 [65], per salvare le ontologie in file di formato corretto è stato usato Jena [66], per implementare l'interfaccia grafica Jswing.

Oltre a fornire lo strumento per l'estrazione di tripletta a partire da un testo biomedico, abbiamo anche progettato **un'Interfaccia Grafica** che supporti l'utente durante le operazioni di inserimento del testo, di estrazione di triplette, di visualizzazione del contenuto e di salvataggio del contenuto su file. Lo strumento è stato pensato per due diversi tipi di utenti: il primo è lo specialista che ha bisogno di costruire ontologie per rappresentare la conoscenza degli ambiti di suo interesse. Il secondo è l'utente non medico che vuole sfruttare lo strumento per visualizzare le informazioni relative a un argomento biomedico attraverso il grafo dell'ontologia estratta dal testo. Per questo motivo, abbiamo disegnato dei test per valutare le performance in differenti situazioni. Per valutare il funzionamento da utente specialista, abbiamo testato il nostro applicativo software con 10 abstract presi da PubMed [32]. Per valutare il funzionamento da utente comune, abbiamo scelto 9 porzioni di testo scaricato da Wikipedia [67] relativo a argomenti biomedicali (e.g. "The kidneys"). L'elenco dei testi è visibile in Tabella 1. Abbiamo deciso di calcolare la precisione del nostro strumento, definita come il rapporto tra le triplette corrette rispetto al numero totale di triplette estratte.

Abbiamo chiamato questo strumento "**Deterministic Triplet Extraction Tool**" (DeTET). La parola "Deterministic" indica che l'approccio è "rule-based" (i.e. creare una serie di regole da seguire). E' stata scelta la lingua inglese per una ragione specifica: l'inglese è una lingua con una struttura della frase estremamente rigida e caratterizzata da relativamente poche eccezioni [54]. L'unità linguistica di base è la parola: gruppi di parole creano **sintagmi** ("**phrase**" in inglese), gruppi di sintagmi creano una struttura che non ha un corrispettivo in italiano detta "clause". Gruppi di "clause" creano frasi, gruppi di frasi creano testi (o più in generale "discourse") [54].

Tabella 1. Elenco dei testi da usare per valutare le performance del sistema.

#	Core concept	Title	Reference	Number of sentences
1	Trauma	Introduction	http://en.wikipedia.org/wiki/Trauma_(medicine)	30
2	Radiography	Introduction	http://en.wikipedia.org/wiki/Radiography	11
3	Pneumonia	Signs and Symptoms	http://en.wikipedia.org/wiki/Pneumonia	12
4	The human body	Introduction	http://en.wikipedia.org/wiki/Human_anatomy	10
5	Thrombotic Event	Risk of a Thrombotic Event after the 6-Week Postpartum Period	The New England Journal Of Medicine [68]	8
6	Test NP Lists			4
7	The kidneys	Introduction	http://en.wikipedia.org/wiki/Kidney	17
8	Pneumonia	Introduction	http://en.wikipedia.org/wiki/Pneumonia	11
9	Deep Brain Stimulation	Introduction	http://en.wikipedia.org/wiki/Deep_Brain_Stimulation	23
10	Kidney abstract	Determination of relative Notch1 and gamma-secretase-related gene expression in puromycin-treated microdissected rat kidneys.	PubMed [69]	9
11	Kidney abstract	Chronic Kidney Disease and the Risks of Death, Cardiovascular Events, and Hospitalization	PubMed [70]	9
12	Kidney abstract	Association of chronic kidney graft failure with recipient blood pressure.	PubMed [71]	12
13	Kidney abstract	PKD1 gene and its protein	PubMed [72]	7
14	Kidney abstract	Acute Kidney Injury, Mortality, Length of Stay, and Costs in Hospitalized Patients	PubMed [73]	8
15	Deep Brain Stimulation	Deep Brain Stimulation	PubMed [74]	8
16	Deep Brain Stimulation	Deep brain stimulation	PubMed [75]	12
17	Deep Brain Stimulation	Deep brain stimulation for intractable chronic cluster headache: proposals for patient selection.	PubMed [76]	6
18	Deep Brain Stimulation	Deep brain stimulation and cluster headache	PubMed [77]	8
19	Deep Brain Stimulation	Asymmetric pallidal neuronal activity in patients with cervical dystonia.	PubMed [78]	14

L’aspetto più interessante è che le “clause” hanno una struttura simile alle triplette, detta struttura **Soggetto-Verbo-Oggetto** (SVO) [57]. La struttura SVO può variare o arricchirsi in base al tipo di verbo (e.g. i verbi “ditransitive” come “to give” hanno bisogno di due oggetti per formare “clause” grammaticamente); esistono solamente sei tipi diversi di verbo [57]: è quindi possibile creare regole specifiche per affrontare ogni situazione. Poiché la “clause” è composta da sintagmi e non esistendo ancora “analizzatori” di “clause” implementati, il nostro lavoro è incentrato sul sintagma inglese: un sintagma è un gruppo di parole costituito da una “head word” e da una serie di “pre-modification” e “post-modification”. Mentre la “head word” è obbligatoria, le “modification” possono essere arbitrariamente tolte o aggiunte senza che la frase diventi grammaticamente non corretta (e.g. “The injured chest” è un sintagma, la cui parola principale è “chest”: togliendo “injured” il sintagma mantiene comunque una sua completezza, togliendo “chest” no); esistono cinque tipi di sintagmi che prendono il nome dal tipo (in inglese “Part Of Speech tag”) della head word: “Noun Phrase”, “Verb Phrase”, “Prepositional Phrase”, “Conjunction Phrase”, “Adverbial Phrase” e “Adjective Phrase” [56]. Esistono vari strument per raggruppare le parole di un testo in sintagmi: essi sono chiamati “chunker” [80]. Poichè una “clause” è composto da sintagmi, dovremo sviluppare uno strumento che associ uno o più sintagmi al ruolo di soggetto di una “clause”, uno o più sintagmi al ruolo di verbo e uno o più sintagmi al ruolo di oggetto. Facendo questo, lo strumento ha di fatto estratto la tripletta relativa a quella “clause”. Usando un chunker però ci troviamo ad affrontare un problema: alcuni elementi della frase vengono ignorati dal chunker, come le virgole e le congiunzioni.

Il nostro applicativo software non considera solo i sintagmi, ma considera anche virgole e congiunzioni, elementi che facilitano l’estrazione di statement dal testo perché aiutano il sistema a ricostruire la struttura della frase. Abbiamo deciso di chiamare questa struttura “Sintagma Arricchito” e lo strumento che serve per estrarlo “Enriched Chunker”. Il sistema considera quindi come “sintagmi” anche le virgole (creando dei “Sintagmi Arricchiti” di tipo “,-type”) e le congiunzioni (creando dei “Sintagmi Arricchiti” di tipo “AndOr-type”).

Risultati

I risultati del presente lavoro sono sia implementativi che di prove di collaudi di quanto realizzato.

In Figura 1 viene mostrato l'intero processo da seguire per estrarre triplette a partire da un testo medico inserito. In Figura 2 è rappresentato il diagramma delle classi relativo al software implementato. La classe "Phrase" è la classe che corrisponde ai Sintagmi Arricchiti presentati prima. La classe "Triplet" è una classe usata per contenere le triplette estratte: un'ontologia nel nostro sistema è una ArrayList di "Triplet". La classe "GUI" è quella che permette, tramite interfaccia grafica, l'interazione utente sistema. La classe LogAnaliz è la classe che esegue tutto il lavoro: essa infatti crea e popola le varie strutture (ArrayList<Triplet> e ArrayList<Phrase>), viene chiamata dalla classe GUI per elaborare il testo, permette il salvataggio e il caricamento di ontologie. In Figura 3 viene mostrato il diagramma delle sequenze dal punto di vista dell'utente che sta utilizzando l'applicativo software. Nella Figura 3 sono indicati tutti i passi per l'estrazione di triplette a partire da un qualsiasi testo. In Figura 4 è mostrato il grafo estratto dal testo "*The kidneys are essential in the urinary system and also serve homeostatic functions such as the regulation of electrolytes, maintenance of acid–base balance, and regulation of blood pressure (via maintaining salt and water balance). They serve the body as a natural filter of the blood, and remove wastes, which are diverted to the urinary bladder.*". In Figura 5 sono mostrate le triplette estratte dallo stesso testo della Figura 4.

Le performance sono state valutate sia su testi medici generici (i.e. presi da Wikipedia [19]) che su testi medici specialistici (i.e. presi da PubMed [18]). In Tabella 2 sono mostrati alcuni risultati sintetici. Nella Tabella 3 sono riportati i risultati relativi al tempo di calcolo.

Il sistema implementato permette l'estrazione automatica di un'ontologia a partire da un testo (Figura 4).

Tabella 2. Performance dell'applicativo software sviluppato.

Metric	Value
Mean extracted triplets for sentence	2,237
Mean precision	0,885
Total extracted triplets	478,000
Total input sentences	219,000
Total correct triplets	428,000
Total mean (Total correct triplets / Total extracted triplets)	0,895
Total computation time	56,631 s

Tabella 3. Tempi di calcolo dell'applicativo software sviluppato.

# Sentences	# Triplets extracted	Time elapsed (s)
161	308	3,425
401	770	4,718
710	1.340	5,717
4686	8.844	23,957
9372	17.688	44,619

Discussione

Seguendo le fasi di sviluppo di un sistema software [61], il nostro sistema è stato progettato e sviluppato.

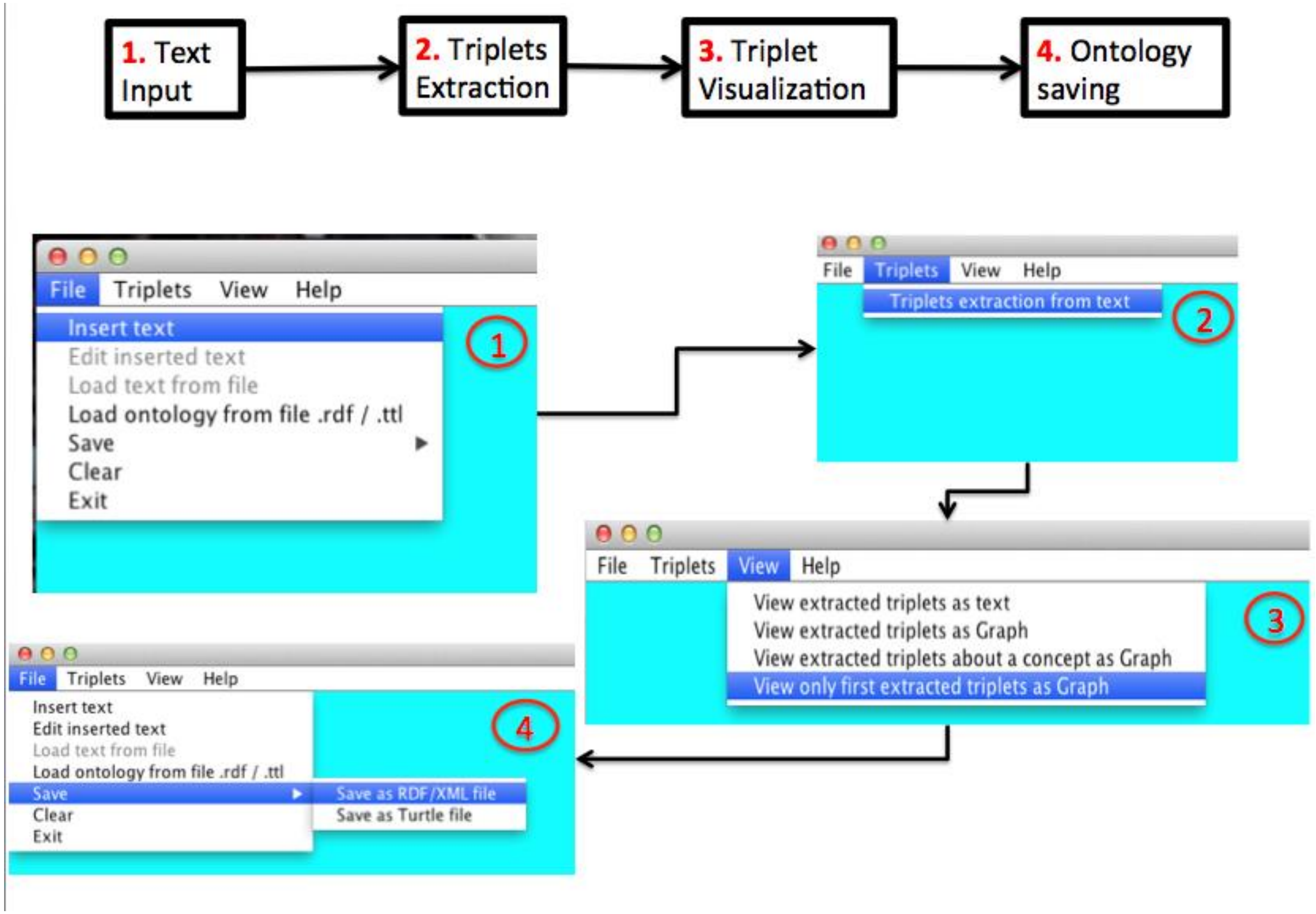
I **test condotti** hanno dimostrato che lo strumento ottiene buoni risultati sia con testi specialistici medici (i.e. scaricati da PubMed) che con testi non specialistici medici (i.e. scaricati da Wikipedia). Il tempo di calcolo richiesto è incoraggiante: un lavoro simile [80] ha richiesto circa 30 secondi per estrarre 160 triplette da 100 frasi (non è stato possibile elaborare le stesse frasi, in quanto non fornite dagli autori). Il sistema permette all'utente di memorizzare le ontologie create su file con due formati standard "RDF/XML" [1] o "Turtle"[1]. Le performance del sistema però sono fortemente influenzate dalla qualità del testo fornito dall'utente: se esso contiene errori grammaticali o concettuali, il sistema commetterà degli errori inevitabili e non correggibili. Oltretutto alcuni strumenti che il "Deterministic Triplet Extraction Tool" (DeTET) utilizza, come il "chunker", commettono a volte errori che portano a commetterne altri in cascata. La gestione delle frasi complesse rimane un punto debole dell'applicativo software: frasi con particolari strutture potrebbero portare ad errori di estrazione. Un'altra situazione che causa problemi è l'utilizzo di formulazioni grammaticalmente corrette ma non frequenti: ad esempio il pronome “which” solitamente si riferisce alla “Noun Phrase” più vicina ma non c’è nessuna regola grammaticale che renda questo uso obbligatorio.

Il sistema è stato pensato e progettato per essere rivolto a varie tipologie di utenti. Tuttavia una fase di valutazione della usabilità non è ancora stata svolta.

Conclusioni

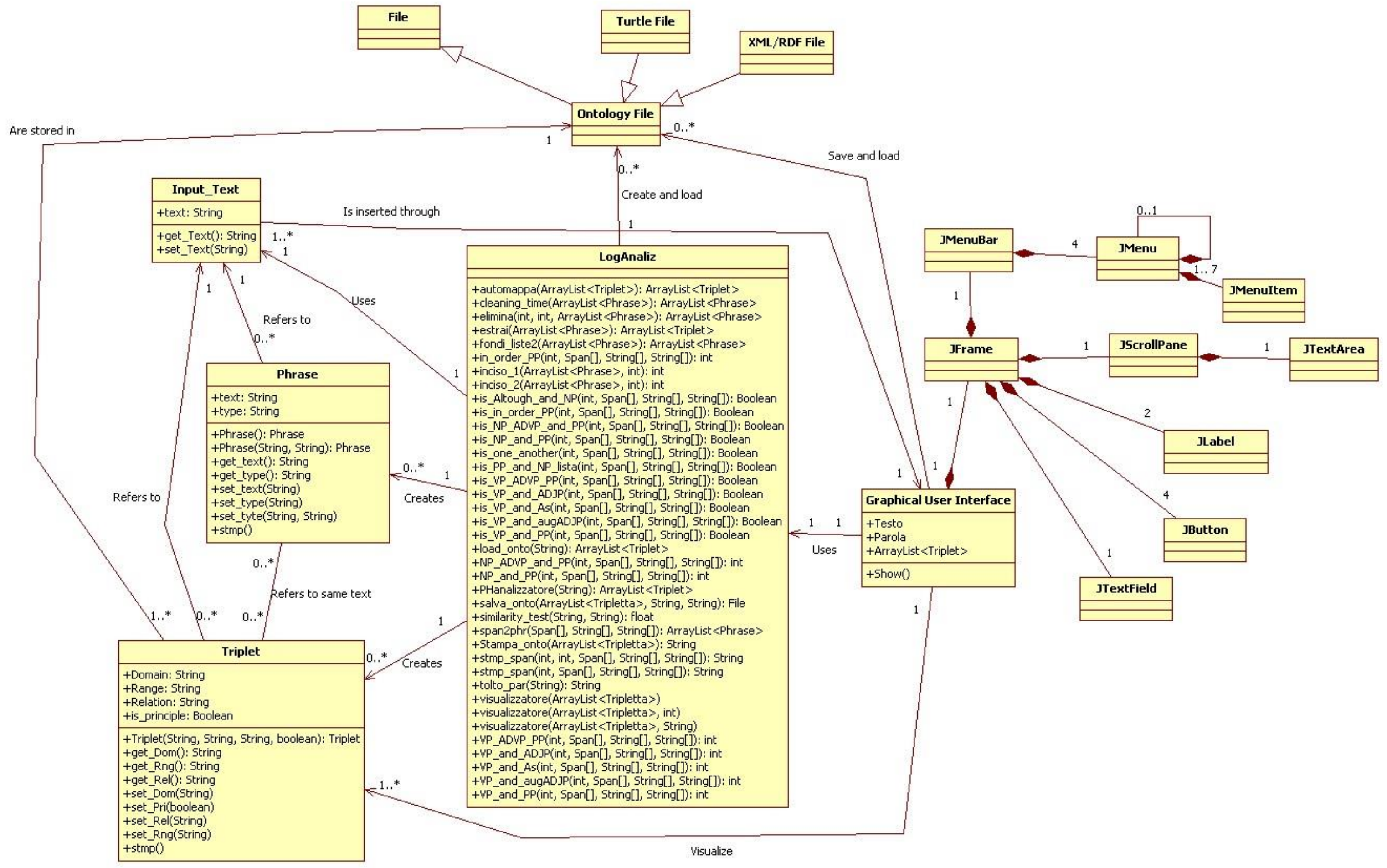
In questo lavoro di laurea abbiamo modellato, progettato e implementato uno strumento per estrarre automaticamente triplette da testo fruttando l’elaborazione del linguaggio naturale: lo abbiamo chiamato Deterministic Triplet Extraction Tool (DeTET). E’ stato creato un contenitore per organizzare le informazioni grammaticali estraibili da una frase inglese: lo abbiamo chiamato Sintagma Arricchito. A partire da questo, è stato sviluppato un applicato software e un’interfaccia grafica per permettere il suo utilizzo. In questo modo, abbiamo contribuito alla soluzione del problema della generazione di ontologie biomediche in modo automatico.

Nonostante i risultati ottenuti siano incoraggianti, rimangono problematiche aperte di miglioramento delle performance e di usabilità del sistema. Tali problematiche potranno essere affrontate in futuri sviluppi della ricerca.



1. Text Input
2. Triplets Extraction
3. Triplet Visualization
4. Ontology saving
File
Triplets
View
Help
Insert text
Edit inserted text
Load text from file
Load ontology from file .rdf / .ttl
Save
Clear
Exit
1
Triplets extraction from text
2
View extracted triplets as text
View extracted triplets as Graph
View extracted triplets about a concept as Graph
View only first extracted triplets as Graph
3
Save as RDF/XML file
Save as Turtle file
4

Figura 1. Interfaccia Grafica implementata: passi per estrarre triplette a partire da un testo.



File
Turtle File
XML/RDF File
Ontology File
Are stored in
Save and load
Create and load
Is inserted through
Input_Text
+text: String
+get_Text(): String
+set_Text(String)
Uses
Refers to
Phrase
+text: String
+type: String
+Phrase(): Phrase
+Phrase(String, String): Phrase
+get_text(): String
+get_type(): String
+set_text(String)
+set_type(String)
+set_tyte(String, String)
+stmp()
Creates
Refers to
Refers to same text
Triplet
+Domain: String
+Range: String
+Relation: String
+is_principle: Boolean
+Triplet(String, String, String, boolean): Triplet
+get_Dom(): String
+get_Rng(): String
+get_Rel(): String
+set_Dom(String)
+set_Pri(boolean)
+set_Rel(String)
+set_Rng(String)
+stmp()
Creates
LogAnaliz
+automappa(ArrayList<Triplet>): ArrayList<Triplet>
+cleaning_time(ArrayList<Phrase>): ArrayList<Phrase>
+elimina(int, int, ArrayList<Phrase>): ArrayList<Phrase>
+estrai(ArrayList<Phrase>): ArrayList<Triplet>
+fondi_liste2(ArrayList<Phrase>): ArrayList<Phrase>
+in_order_PP(int, Span[], String[], String[]): int
+inciso_1(ArrayList<Phrase>, int): int
+inciso_2(ArrayList<Phrase>, int): int
+is_Altough_and_NP(int, Span[], String[], String[]): Boolean
+is_in_order_PP(int, Span[], String[], String[]): Boolean
+is_NP_ADVP_and_PP(int, Span[], String[], String[]): Boolean
+is_NP_and_PP(int, Span[], String[], String[]): Boolean
+is_one_another(int, Span[], String[], String[]): Boolean
+is_PP_and_NP_lista(int, Span[], String[], String[]): Boolean
+is_VP_ADVP_PP(int, Span[], String[], String[]): Boolean
+is_VP_and_ADJP(int, Span[], String[], String[]): Boolean
+is_VP_and_As(int, Span[], String[], String[]): Boolean
+is_VP_and_augADJP(int, Span[], String[], String[]): Boolean
+is_VP_and_PP(int, Span[], String[], String[]): Boolean
+load_onto(String): ArrayList<Triplet>
+NP_ADVP_and_PP(int, Span[], String[], String[]): int
+NP_and_PP(int, Span[], String[], String[]): int
+PHanalizzatore(String): ArrayList<Triplet>
+salva_onto(ArrayList<Tripletta>, String, String): File
+similarity_test(String, String): float
+span2phr(Span[], String[], String[]): ArrayList<Phrase>
+Stampa_onto(ArrayList<Tripletta>): String
+stmp_span(int, int, Span[], String[], String[]): String
+stmp_span(int, Span[], String[], String[]): String
+tolto_par(String): String
+visualizzatore(ArrayList<Tripletta>)
+visualizzatore(ArrayList<Tripletta>, int)
+visualizzatore(ArrayList<Tripletta>, String)
+VP_ADVP_PP(int, Span[], String[], String[]): int
+VP_and_ADJP(int, Span[], String[], String[]): int
+VP_and_As(int, Span[], String[], String[]): int
+VP_and_augADJP(int, Span[], String[], String[]): int
+VP_and_PP(int, Span[], String[], String[]): int
Uses
Visualize
Graphical User Interface
+Testo
+Parola
+ArrayList<Triplet>
+Show()
JMenuBar
JMenu
JMenuItem
JFrame
JScrollPane
JTextArea
JLabel
JButton
JTextField
0..1
1.. 7
0..*
1..*
1
2
4

Figura 2. Diagramma delle classi del programma software implementato. Mostra come è fatto il sistema.

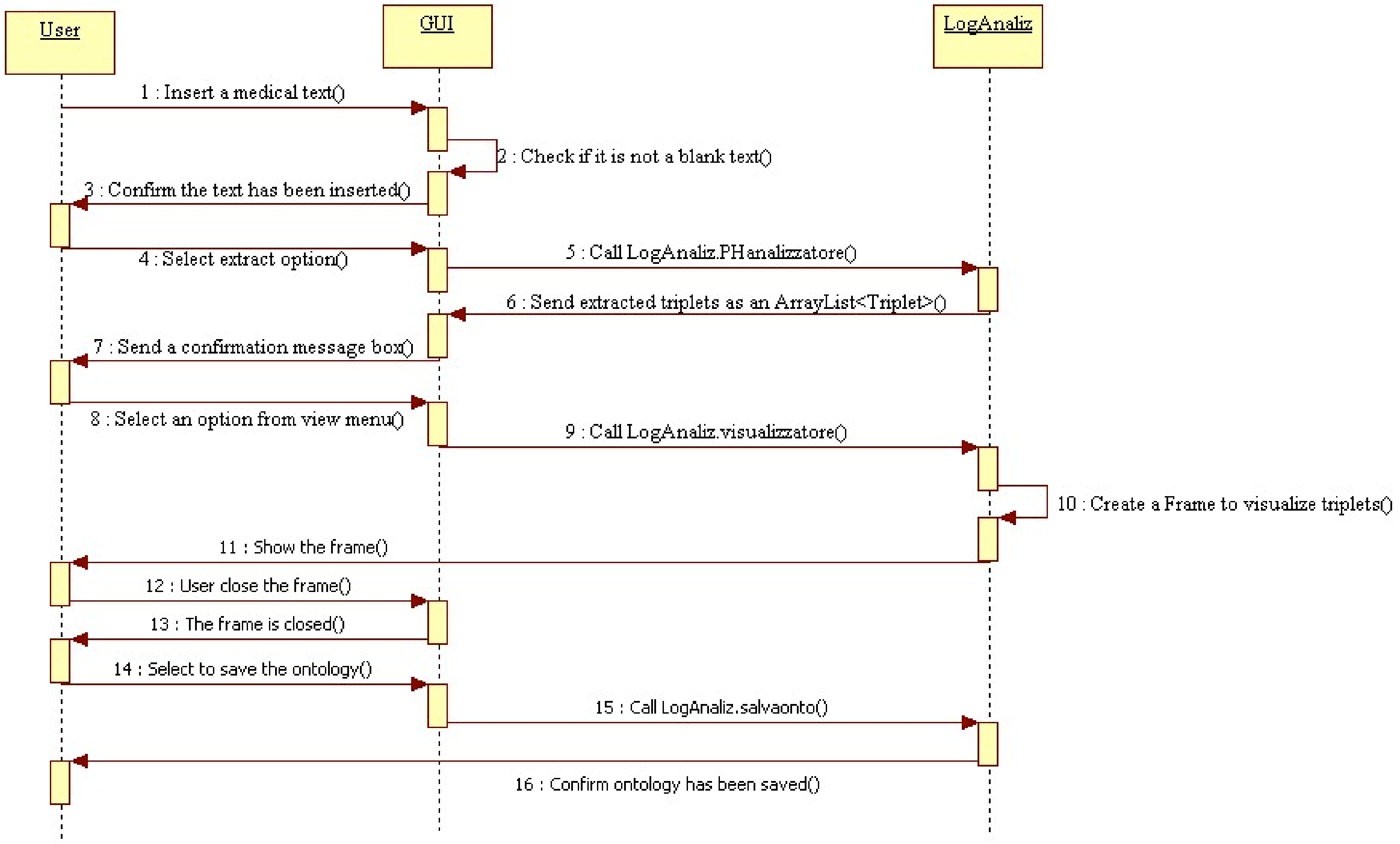

User
GUI
LogAnaliz
1 : Insert a medical text()
2 : Check if it is not a blank text()
3 : Confirm the text has been inserted()
4 : Select extract option()
5 : Call LogAnaliz.PHanalizzatore()
6 : Send extracted triplets as an ArrayList<Triplet>()
7 : Send a confirmation message box()
8 : Select an option from view menu()
9 : Call LogAnaliz.visualizzatore()
10 : Create a Frame to visualize triplets()
11 : Show the frame()
12 : User close the frame()
13 : The frame is closed()
14 : Select to save the ontology()
15 : Call LogAnaliz.salvaonto()
16 : Confirm ontology has been saved()

Figura 3. Diagramma delle sequenze dell'applicativo software sviluppato. Mostra come funzioni il software.

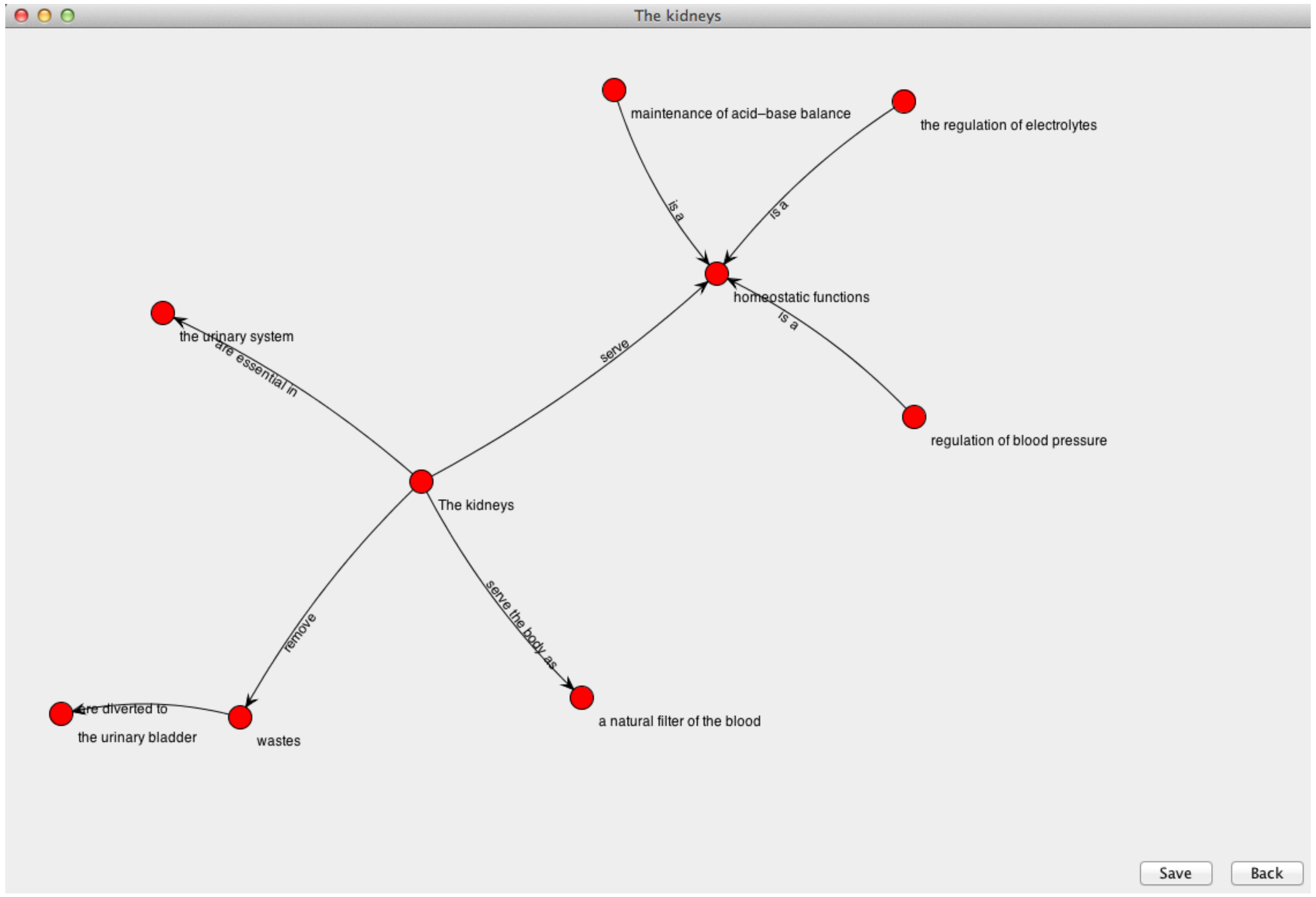
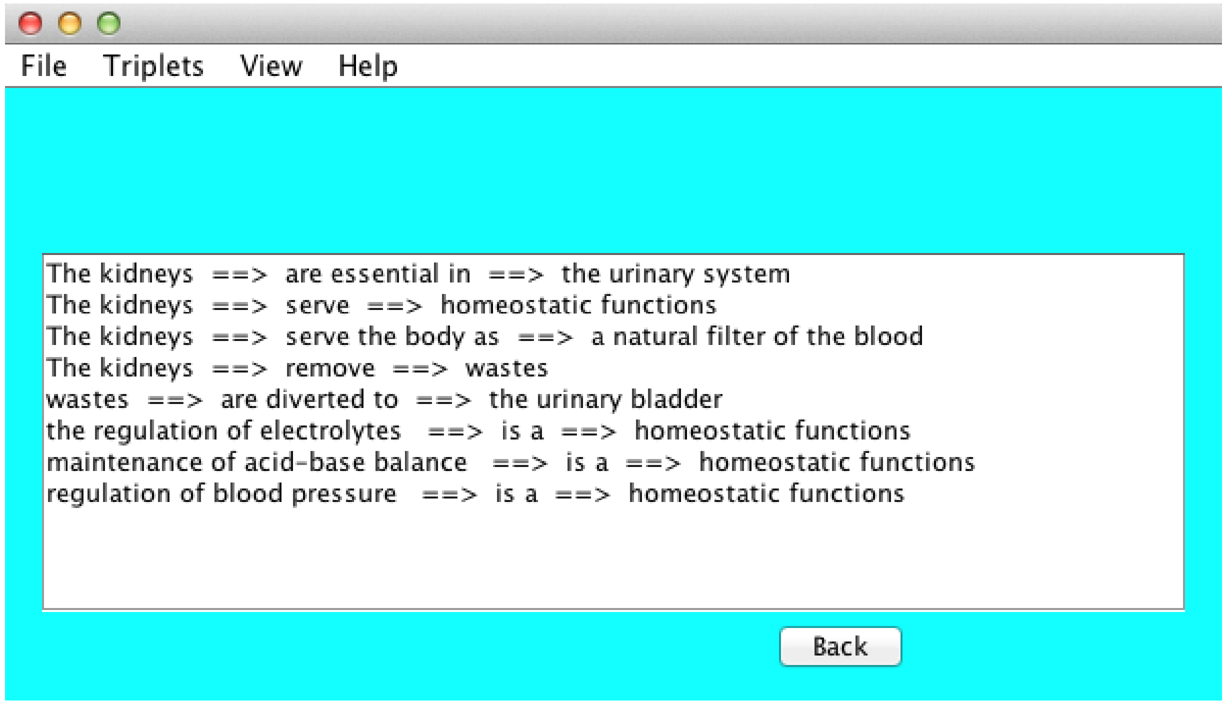


Figura 4. Come il sistema mostra il grafo estratto dal testo "*The kidneys are essential in the urinary system and also serve homeostatic functions such as the regulation of electrolytes, maintenance of acid–base balance, and regulation of blood pressure (via maintaining salt and water balance). They serve the body as a natural filter of the blood, and remove wastes, which are diverted to the urinary bladder.*".



```
The kidneys  ==>  are essential in  ==>  the urinary system
The kidneys  ==>  serve  ==>  homeostatic functions
The kidneys  ==>  serve the body as  ==>  a natural filter of the blood
The kidneys  ==>  remove  ==>  wastes
wastes  ==>  are diverted to  ==>  the urinary bladder
the regulation of electrolytes  ==>  is a  ==>  homeostatic functions
maintenance of acid-base balance  ==>  is a  ==>  homeostatic functions
regulation of blood pressure  ==>  is a  ==>  homeostatic functions
```

Figura 5. Come il sistema mostra le triplette estratte a partire dal testo "*The kidneys are essential in the urinary system and also serve homeostatic functions such as the regulation of electrolytes, maintenance of acid–base balance, and regulation of blood pressure (via maintaining salt and water balance). They serve the body as a natural filter of the blood, and remove wastes, which are diverted to the urinary bladder.*". Il simbolo "==>" serve per dividere il Dominio dalla Relazione e la Relazione dal Range (e.g. nella tripletta "The Kidney ==> are essential in ==> the urinary system", "The Kidney" è il Dominio, "are essential in" è la Relazione e "the urinary system" è il Range).

Abstract

Introduction, Background and objectives

The object of this work is **the design**, **the model and the implementation of a software system for creating biomedical ontologies automatically** starting from texts in Natural Language. Furthermore, we designed and developed a Graphical User Interface to facilitate the use of the system. Ontologies in biomedical domains are used for organizing biological concepts and representing relationships among them [3].

In Computer Science, **an ontology is a formal representation** of knowledge as a set of concepts connected by relations and properties to denote things [2]. When a property links two different concepts, we call this link "Relation". Two concepts linked together by a relation form a triplets (a.k.a. statement): first concept is called Domain and last concept is called Range (we will use "Domain + Relation + Range" layout to write statements). When the knowledge of a domain is represented with this declarative formalism, the set of objects that can be represented is called the universe [15]: the ontology does not consider concepts outside its universe, and every statement belonging to the universe is to considered true by the ontology. The point is an ontology could contain wrong pieces of information from an external point of view (e.g. the statement "Men + are not + mortal" is wrong), but it would consider them true a priori: its task is to decide if a conclusion is impliable from previously stored statements or not; its task is not to decide if a stored statement is true or not. The only check ontology does on stored statements is the consistency check (i.e. it checks if there are not contradictory statements). We have used the word "formal" to describe what is an ontology because every ontology is based on some kind of **formal logic** like the First Order Logic (FOL) [23] or some sub-Logic derived from it (e.g. Description Logic DL) [23]. Every logic has a "resoner" tool: a "reasoner" is used to establish if a specific conclusion is impliable from the knowledge stored in the ontology. Thus, the question our logic system would like to answer is "Does some selected axioms imply a specific theorem?": this kind of question is linked to the "Decidability problem" [25]. An axiom is a valid statement (i.e. it is considered true). The system could replies in three different ways: "Yes" , "No" , or just not terminate (it means the system does not know the answer). The point we do not know is how many loops the system needs to give an output: until we do not see the output, we can not establish if the process will terminate or not. If the system always replies to every answer within a finite number of loop, it is called "decidable". If the system always replies within a finite number of loop to every answer when the answer is "Yes",

it is called “semi-decidable”. Otherwise is called “undecidable”. Ontologies use the Description Logic because it is a decidable logic.

Biomedical ontologies can be used to serve integration of clinical data [11]: biomedical researchers have a growing need to integrate data deriving from different disciplines at different granularity levels [11]. This implies that medical terminology becomes affected by an increasing concern with matters of consistency (i.e. no contradictions) [11]. By providing a common structure and terminology, the use of biomedical ontology aims at providing a single data source for review: “*furthermore, the use of such a common vocabulary promises benefits of less redundant data and easier opportunities for longitudinal studies and meta-analyses and for ensuring consistency of data across the lifetime of the patient and from one healthcare institution to the next.”* [11]. The use of biomedical ontologies promises significant rewards, for example for clinical decision support, because they improve the information retrieval process [11]. Furthermore, biomedical ontologies allow the user to semantic search [29] and process [31] biomedical contents based on the meaning that this contents have to humans. **Semantic search** is defined as a information retrieval process that exploits somehow domain knowledge [29]. Domain knowledge can be formalized through an ontology; thus, if we store the knowledge through an ontology, lot of problems of traditional search scenario, such as synonymy (i.e. same concept can be expressed with several words) and word acceptance (i.e. same word can have several meanings), do not exist: in a single ontology, every concept must be unique and have only one “mean” (the concept itself is its meaning). One example of **knowledge processing** is the work of Rubin et al. to demonstrate the capabilities of logical deduction in a clinical scenario [31]: they created a system to infer about the consequences of penetrating injuries. They created a knowledge model of the chest, and the heart anatomy and physiology. Then they used a domain-independent classifier to infer ischemic regions of the heart as well as anatomic spaces containing ectopic blood after a penetrating injury. Given a set of anatomic structures that are directly injured by a projectile, they wanted to create a reasoning application that deduces secondary injuries (regions of myocardium that will be ischemic if a coronary artery is injured, and propagation of injury as bleeding occurs into damaged anatomic compartments that surround the heart). They applied this reasoning service to try to infer the effects of some injuries and the results were confirmed by a physician: “Our results suggest that inferring the consequences of penetrating injury can be formalized as a classification task. There are benefits in using OWL as a representation language.” [31]. Ontologies offer to enhance extracting information service: if a user want to extract some tags from a text, he/she must read and find out best concepts to describe what he/she just reads [16]. This is avoidable thanks to an “annotator”: it is a tool which can automatically assigns concept from an external ontology to a text the open biomedical annotator [1]).

Several different approaches to **ontologies development** have been carried out [35]. All these approaches follow three steps:

1. Searching for a exhaustive set of terms
2. Organizing the terms into a taxonomy of classes
3. Choose which of the taxonomy display in the ontology

Ontology creators usually use two possible approaches based upon the direction of ontology construction; in bottom-up approach they start with some descriptions of the domain and obtain a classification [35], while in top-down one, they start with an a priori abstract view of the domain itself [35]. The **real problem is every ontology is an handmade product**, created from a team of experts (domain experts and IT experts): *"Although ontologies have been proposed as an important and natural means of representing real world knowledge for the development of database designs, most ontology creation is not carried out systematically."* [36]. In addition, nevertheless the large number of existing ontologies, there is no state-of-art methodology for building them. We can make a comparison with database development field, where the conceptual modeling of every database follows standardized steps, such as the creation of an Entity-Relationship (E–R) model. Once every step has been followed, the creation of a database is almost automatic. In the ontologies subject matter does not exist any similar tool to E-R model or a standardized workflow: thus any team of experts will use their own approach. However, some best-practice tutorials exist ([37] or [38]).

The aim of this work has been to design and develop a tool to exploit natural language texts to automatically extract triplets.

Methods

To develop our system, we have analyzed English Linguistic to increase our understanding on how sentences are built and to understand which pieces of information can be mined exploiting linguistic knowledge. Furthermore, we have deepened the Natural Language Processing to establish what tools are the best fit for our system. In the end, we have studied the ontologies subject matter to understand how to store extracted pieces of information.

Our system has been developed using the Software Engineering waterfall model with returns [61]. Our system has been developed using Java programming language and uses some available Application Program Interface (API): OpenNLP [64] has been used to implement Natural Language Processing tools, Jung2 has been used to draw graphs, Jena [66] has been used to save ontologies on external files and to load these. The Graphical User Interface has been developed using Jswing.

We envisage our system will be used by two different kind of users: biomedical researchers and consumers, as a graph representation (with not more than 15-20 nodes) is clearer and better understandable than a similar length text. Researchers could also use our system to create an ontology because creating an ontology by hand is pretty hard. It could provide a easy way to create ontologies from a selected text: the user is "responsible" for the correctness of the text, because our system itself cannot be aware of incorrect knowledge inside an input text (as a first implementation).

To text the performance of the system, we have selected some texts from two main sources: Wikipedia texts [67] and PubMed/Medline [32] abstracts.

First source is compatible with a non technical user who would like to increase the knowledge about a medical topic. Thus, his/her source will not be a technical one: probably, he/she will use Wikipedia [67]. We have selected the introduction of six common and less common topics: Trauma, Radiography, Pneumonia, The human body, The kidneys and Deep Brain Stimulation. We have added also the "Signs and Symptoms" paragraph of Pneumonia topics, also found on Wikipedia. Every text has been download at 02-19-2014. **Second source** is composed by abstracts of some papers: this source try to simulate the general practitioner who would like to increase his/her knowledge about a specialist topic to improve his/her care quality. Ten abstracts has been downloaded from PubMed/Medline [32] and one from New England Journal of Medicine. PubMed's ones are splitted this way: five of them are related to "Kidney" while other five are related to "Deep Brain Stimulation". We have chosen the "Deep Brain Stimulation" topic because it is a specialist topic which could be looked for by non specialist clinician. The NEJM's one is related to Thrombotic Event (see the appendix A for more accurate references). A last text has been created to **test Noun Phrase lists**: it contains some artificially created sentences to evaluate the performances with lists. In Table_a 1 there is the resume of test texts: the test corpus contains 219 sentences.

Our approach is based on the fact that English Linguistic follows precise rules: a clause minimal structure is de facto fixed. For this reason we called our tool "Deterministic": it follows a rule-based approach. Thus, the system will extract the clause structures and then it will convert them in ontology statements. We based our approach on some free available Natural Language Processing (NLP) tools. We chose English language because the English linguistic is very rigid and because the sentence has a Subject Verb Object (SVO) structure: the SVO structure is quite similar to an ontology triplet. In English, the word is the smallest unit; words are grouped in phrases; phrases are grouped in clauses which are grouped in sentences; a group of sentences composes the discourse. Phrases and clauses are related with our work. A phrase is a sequence of at least one word. Every phrase has only one main word, called *head word*: it is a lexical item which is central to the phrase in the sense that some crucial information would be missing without it. The Part-Of-Speech tag of the head word implies the phrase type, but not every word class can originate a phrase: only five different kinds of phrase exist Noun Phrase (NP), Verb Phrase (VP), Adjective Phrase (AdvP), Adverbial Phrase (AdvP), and Prepositional Phrase (PP). **Clauses** are unit of syntactic construction formed by phrases" [57]. They contain always at least one Verb Phrase (VP) normally preceded by a subject element and followed by any elements needed to make the clause grammatically complete (the aforementioned SVO structure). Thus we have to design and develop a tool to automatically find every Subject, Verb and Object of a given clause: this way it would also have extracted the ontology triplet. Because of the clause is composed by phrases, we have to work at the phrase level: the Natural Language Processing tool to extract phrases from sentences is called **chunker**. A chunker only extract phrases from sentences: this way it discards some elements, such as commas and conjunction. For this reason, we decided to design a new structure to store grammatical information: commas become ","-type Phrase and conjunctions become "AndOr"-type Phrase. This way, the phrase punctuation is included into the grammatical structure. Furthermore some Phrases are merged together and other are erased, edited or moved. We called this structure "**Enriched Phrases**" and we called the tool to extract them "**Enriched Chunker**".

Table_a 1. Texts for testing DeTET performances.

#	Core concept	Title	Reference	Number of sentences
1	Trauma	Introduction	http://en.wikipedia.org/wiki/Trauma_(medicine)	30
2	Radiography	Introduction	http://en.wikipedia.org/wiki/Radiography	11
3	Pneumonia	Signs and Symptoms	http://en.wikipedia.org/wiki/Pneumonia	12
4	The human body	Introduction	http://en.wikipedia.org/wiki/Human_anatomy	10
5	Thrombotic Event	Risk of a Thrombotic Event after the 6-Week Postpartum Period	The New England Journal Of Medicine [68]	8
6	Test NP Lists			4
7	The kidneys	Introduction	http://en.wikipedia.org/wiki/Kidney	17
8	Pneumonia	Introduction	http://en.wikipedia.org/wiki/Pneumonia	11
9	Deep Brain Stimulation	Introduction	http://en.wikipedia.org/wiki/Deep_Brain_Stimulation	23
10	Kidney abstract	Determination of relative Notch1 and gamma-secretase-related gene expression in puromycin-treated microdissected rat kidneys.	PubMed [69]	9
11	Kidney abstract	Chronic Kidney Disease and the Risks of Death, Cardiovascular Events, and Hospitalization	PubMed [70]	9
12	Kidney abstract	Association of chronic kidney graft failure with recipient blood pressure.	PubMed [71]	12
13	Kidney abstract	PKD1 gene and its protein	PubMed [72]	7
14	Kidney abstract	Acute Kidney Injury, Mortality, Length of Stay, and Costs in Hospitalized Patients	PubMed [73]	8
15	Deep Brain Stimulation	Deep Brain Stimulation	PubMed [74]	8
16	Deep Brain Stimulation	Deep brain stimulation	PubMed [75]	12
17	Deep Brain Stimulation	Deep brain stimulation for intractable chronic cluster headache: proposals for patient selection.	PubMed [76]	6
18	Deep Brain Stimulation	Deep brain stimulation and cluster headache	PubMed [77]	8
19	Deep Brain Stimulation	Asymmetric pallidal neuronal activity in patients with cervical dystonia.	PubMed [78]	14

Results

We have implemented our approach using Java programming language; we have also developed a User Graphic Interface (GUI) to allow the user to use the implemented tool as a standalone software; the GUI has been implemented using Java programming language, too.

Figure 1_a shows how to extract triplets from a text using our application software:

1. The user selects “File” menu and “Insert text” option to insert the text inside the system.
2. The user selects “Triplets” menu and “Triplet extraction from text” option to extract triplets from the previously inserted text.
3. User selects one view option from the “View” menu to display extracted triplets
4. User selects “Save” option from “File” menu to save the ontology. The user has two choices, “Save as RDF/XML file” and “Save as Turtle file”: this way he/she can choose the format of the output ontology.

Figure_a 2 shows the class diagram of the implemented system. Figure_a 3 shows the every step to extract triplets from text. Figure_a 4 shows the graph extracted from the text: “*The kidneys are essential in the urinary system and also serve homeostatic functions such as the regulation of electrolytes, maintenance of acid–base balance, and regulation of blood pressure (via maintaining salt and water balance). They serve the body as a natural filter of the blood, and remove wastes, which are diverted to the urinary bladder*.”. Figure_a 5 shows the triplets extracted to build the Figure 4 graph.

Performances have been evaluated on Table_a 1 texts. Table_a 2 shows a summary of global performance. Table_a 3 shows the evaluation of DeTET speed, using texts composed by random sentences from Table_a 1 texts.

Table_a 2. Summary of Deterministic Triplet Extraction Tool performances.

Metric	Value
Mean extracted triplets for sentence	2,237
Mean precision	0,885
Total extracted triplets	478
Total input sentences	219
Total correct triplets	428
Total mean (Total correct triplets / Total extracted triplets)	0,895
Total computation time	56,631 s

Table_a 3. Results for tool speed tests.

# Sentences	# Triplets extracted	Time elapsed (s)
161	308	3,425
401	770	4,718
710	1.340	5,717
4686	8.844	23,957
9372	17.688	44,619

Discussion

We have developed the Deterministic Triplet Extraction Tool (DeTET) following every waterfall model step [61].

The results in extracting triplets from testing texts (Table_a 1) are encouraging (Table_a 2): the system scores a mean precision of almost 90%. DeTET seems to be a very fast tool: the processing time was calculated on the command line version of DeTET, not on the GUI-version. It is quicker than similar systems: for example Rusu et al. [80], performed an extraction of 168 triplets from 100 sentences in 29,5 seconds. Authors did not specify the sentences, thus a run with same inputs in our system is not possible. However, in almost the same time (23,95 seconds) DeTET extracts 8.844 triplets from 4686 sentences (Table 8).

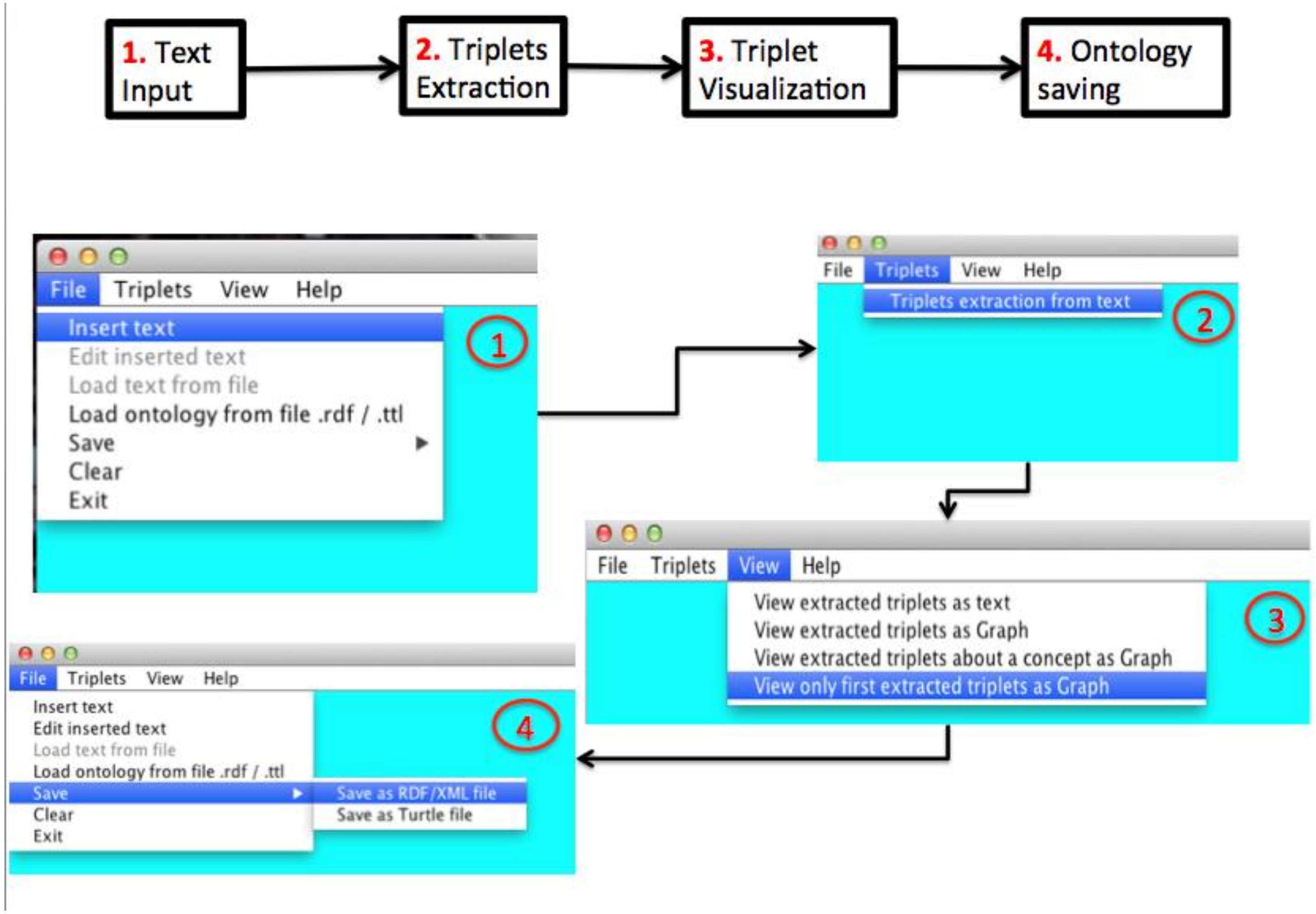
DeTET has some **weakness**. First, if the sentence contains any grammatical or linguistic error, DeTET will make mistakes: this kind of issue is unavoidable. Second, if the text contains any kind of conceptual mistake, DeTET will not notice it and will make mistakes. Furthermore, DeTET has some problem in processing complex structure sentences and phrases such as the embedding phrases, which means some phrases that contain other phrases (e.g. the NP “the kidney damaged by alcohol” contains also a VP, “damaged”). This kind of issue is very hard to manage, because NLP tools often make mistakes in processing embedding phrase: for example, often an embedded verb phrase is considered as a standalone phrase.

There are some open issues we have not completely deal with. For example, String Similarity issue. We have proposed an easy approach to the problem because a deeper approach lead to another project. Also we have avoided statement with same Domain and Range (e.g. "Cells generate theirself"): this kind of situation could lead not only to more precise results, but also requires to much time to be implemented. Also an improved pronoun management approach could lead to enhanced results.

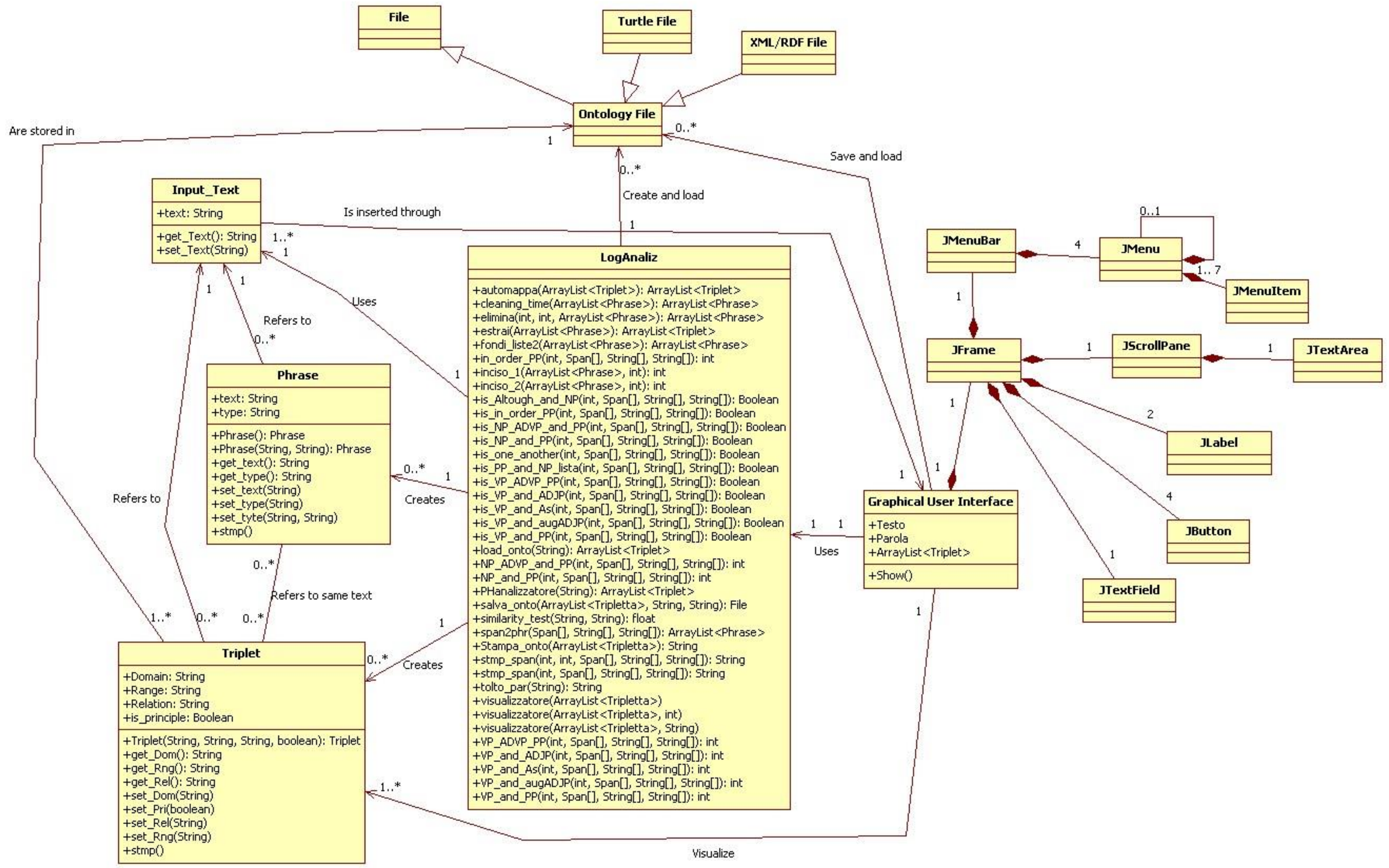
As part of the future work we plan to enhance graph visualization too, allowing user to interact with graph nodes.

Conclusions

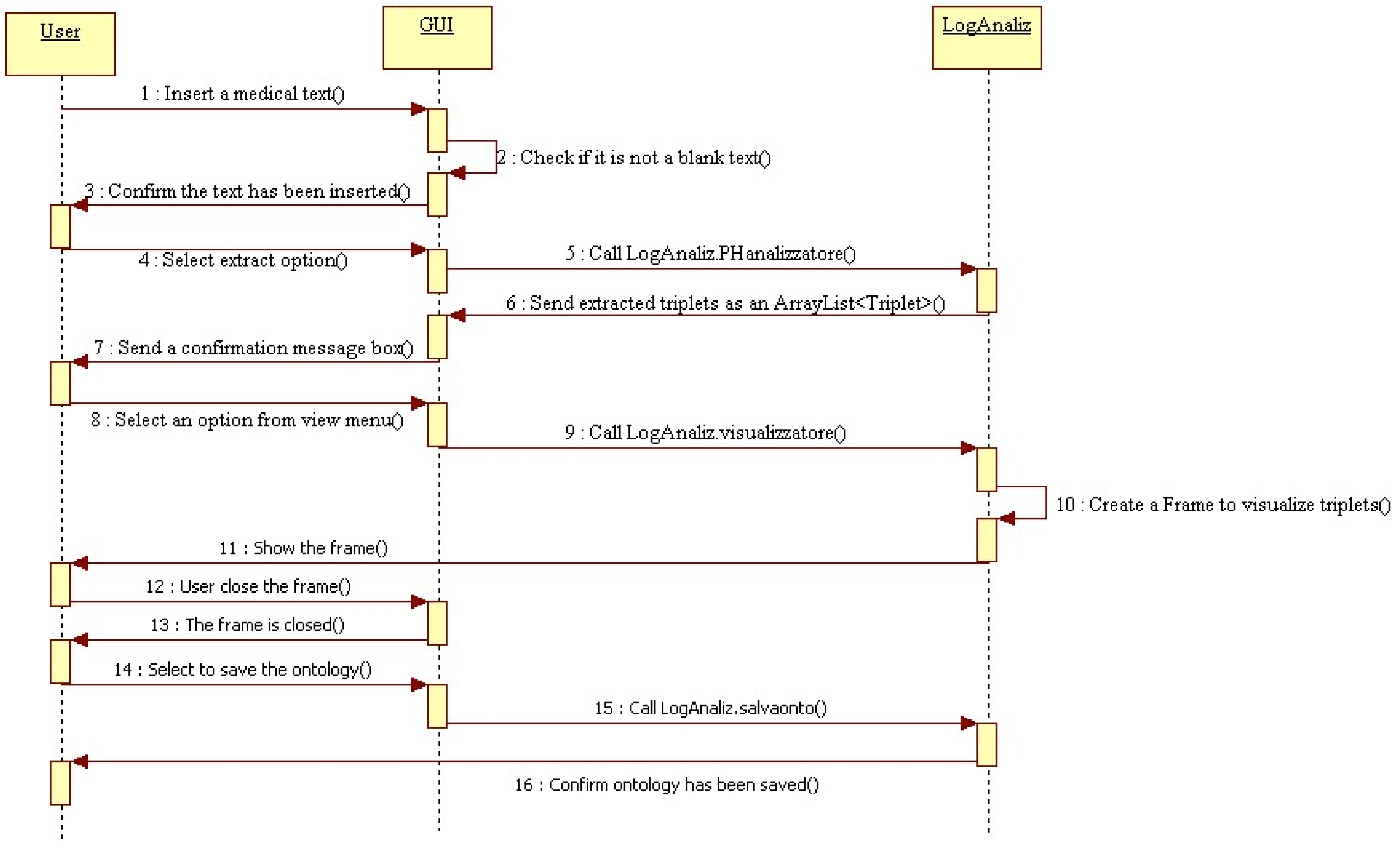
We presented an approach to Next Generation of Biomedical Ontologies, and the implementation of a software tool for extracting triplets from biomedical texts, we called Deterministic Triplet Extraction Tool (DeTET). It exploits Natural Language Processing techniques to extract structured pieces of information from natural language digital texts. We have developed new structure to store grammatical information from a sentence we called Enriched Phrase. It has been developed as a standalone Java tool with a clear and user-friendly Graphic User Interface. We performed some investigation tests, obtaining around 90% correct extracted triplets and low computation time (less than a couple of seconds for a page-length text). Thanks to that, Deterministic Triplet Extraction Tool seems to be the essential step to solve ontology automatic generation issue.


1. Text Input
2. Triplets Extraction
3. Triplet Visualization
4. Ontology saving
File
Triplets
View
Help
Insert text
Edit inserted text
Load text from file
Load ontology from file .rdf / .ttl
Save
Clear
Exit
1
Triplets extraction from text
2
View extracted triplets as text
View extracted triplets as Graph
View extracted triplets about a concept as Graph
View only first extracted triplets as Graph
3
Save as RDF/XML file
Save as Turtle file
4

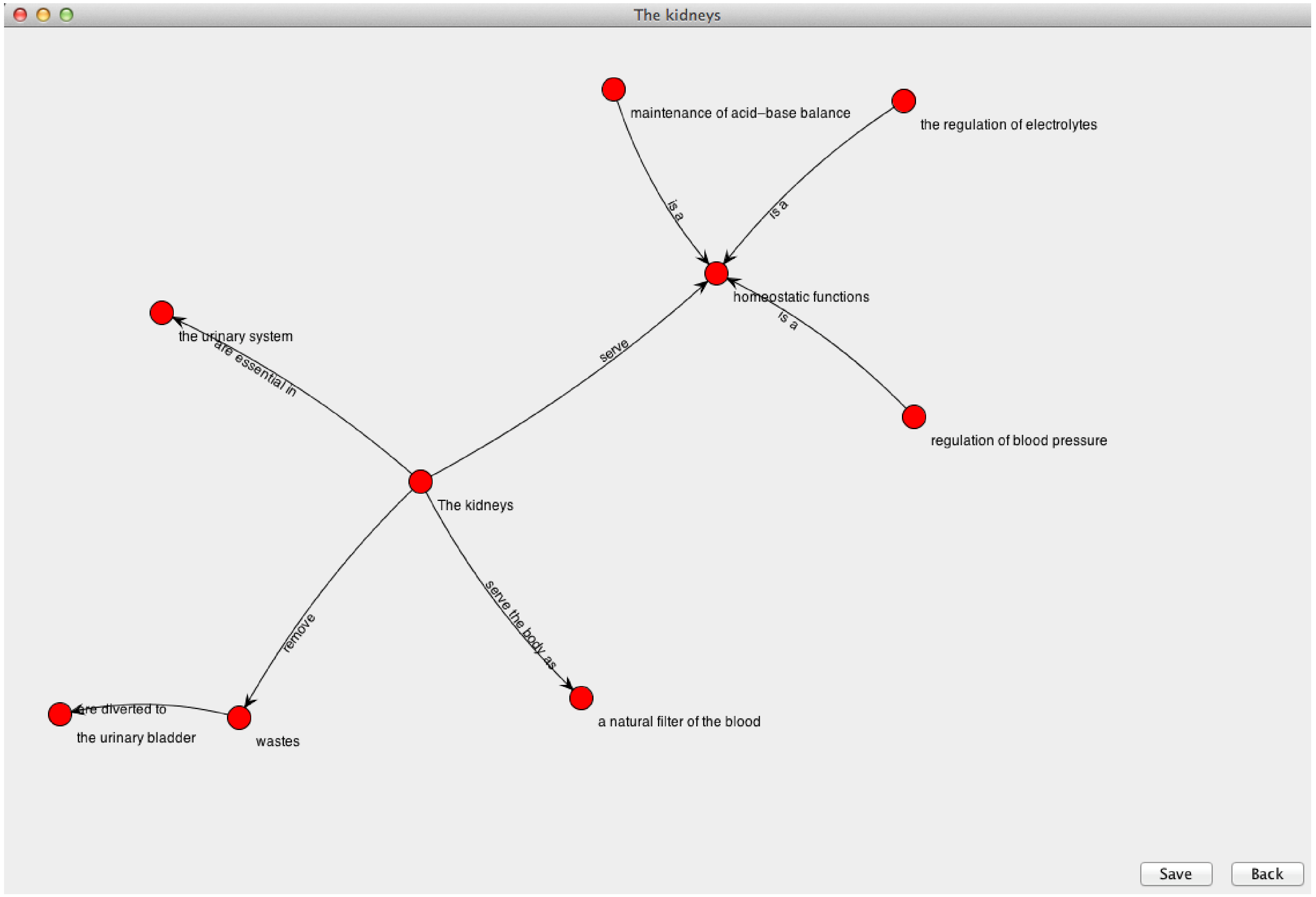
Figure_a 1. The workflow to extract triplets from a text using the Graphical User Interface.


File
Turtle File
XML/RDF File
Ontology File
Are stored in
1
0..*
Save and load
0..*
Create and load
1
Input_Text
+text: String
+get_Text(): String
+set_Text(String)
Is inserted through
1..*
1
Uses
1
1
Refers to
0..*
1
Phrase
+text: String
+type: String
+Phrase(): Phrase
+Phrase(String, String): Phrase
+get_text(): String
+get_type(): String
+set_text(String)
+set_type(String)
+set_tyte(String, String)
+stmp()
0..*
1
Creates
Refers to
0..*
Refers to same text
1..*
0..*
0..*
Triplet
+Domain: String
+Range: String
+Relation: String
+is_principle: Boolean
+Triplet(String, String, String, boolean): Triplet
+get_Dom(): String
+get_Rng(): String
+get_Rel(): String
+set_Dom(String)
+set_Pri(boolean)
+set_Rel(String)
+set_Rng(String)
+stmp()
1
0..*
Creates
1..*
Visualize
LogAnaliz
+automappa(ArrayList<Triplet>): ArrayList<Triplet>
+cleaning_time(ArrayList<Phrase>): ArrayList<Phrase>
+elimina(int, int, ArrayList<Phrase>): ArrayList<Phrase>
+estrai(ArrayList<Phrase>): ArrayList<Triplet>
+fondi_liste2(ArrayList<Phrase>): ArrayList<Phrase>
+in_order_PP(int, Span[], String[], String[]): int
+inciso_1(ArrayList<Phrase>, int): int
+inciso_2(ArrayList<Phrase>, int): int
+is_Altough_and_NP(int, Span[], String[], String[]): Boolean
+is_in_order_PP(int, Span[], String[], String[]): Boolean
+is_NP_ADVP_and_PP(int, Span[], String[], String[]): Boolean
+is_NP_and_PP(int, Span[], String[], String[]): Boolean
+is_one_another(int, Span[], String[], String[]): Boolean
+is_PP_and_NP_lista(int, Span[], String[], String[]): Boolean
+is_VP_ADVP_PP(int, Span[], String[], String[]): Boolean
+is_VP_and_ADJP(int, Span[], String[], String[]): Boolean
+is_VP_and_As(int, Span[], String[], String[]): Boolean
+is_VP_and_augADJP(int, Span[], String[], String[]): Boolean
+is_VP_and_PP(int, Span[], String[], String[]): Boolean
+load_onto(String): ArrayList<Triplet>
+NP_ADVP_and_PP(int, Span[], String[], String[]): int
+NP_and_PP(int, Span[], String[], String[]): int
+PHanalizzatore(String): ArrayList<Triplet>
+salva_onto(ArrayList<Tripletta>, String, String): File
+similarity_test(String, String): float
+span2phr(Span[], String[], String[]): ArrayList<Phrase>
+Stampa_onto(ArrayList<Tripletta>): String
+stmp_span(int, int, Span[], String[], String[]): String
+stmp_span(int, Span[], String[], String[]): String
+tolto_par(String): String
+visualizzatore(ArrayList<Tripletta>)
+visualizzatore(ArrayList<Tripletta>, int)
+visualizzatore(ArrayList<Tripletta>, String)
+VP_ADVP_PP(int, Span[], String[], String[]): int
+VP_and_ADJP(int, Span[], String[], String[]): int
+VP_and_As(int, Span[], String[], String[]): int
+VP_and_augADJP(int, Span[], String[], String[]): int
+VP_and_PP(int, Span[], String[], String[]): int
1
1
Uses
1
1
1
Graphical User Interface
+Testo
+Parola
+ArrayList<Triplet>
+Show()
1
JMenuBar
4
JMenu
0..1
1.. 7
JMenuItem
1
JFrame
1
JScrollPane
1
JTextArea
1
2
JLabel
4
JButton
1
JTextField

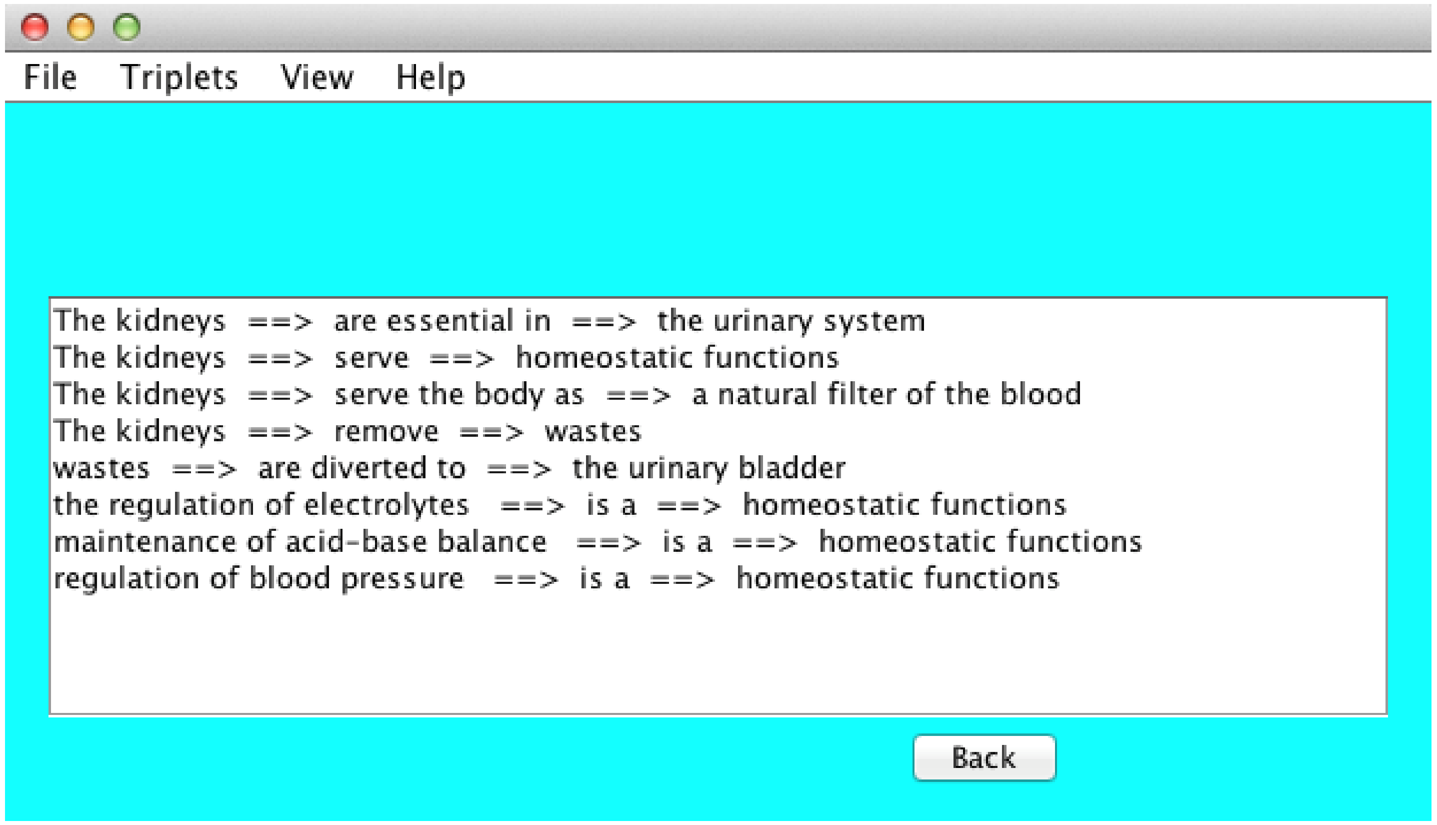
Figure_a 2. Class diagram of implemented system.


User
GUI
LogAnaliz
1 : Insert a medical text()
2 : Check if it is not a blank text()
3 : Confirm the text has been inserted()
4 : Select extract option()
5 : Call LogAnaliz.PHanalizzatore()
6 : Send extracted triplets as an ArrayList<Triplet>()
7 : Send a confirmation message box()
8 : Select an option from view menu()
9 : Call LogAnaliz.visualizzatore()
10 : Create a Frame to visualize triplets()
11 : Show the frame()
12 : User close the frame()
13 : The frame is closed()
14 : Select to save the ontology()
15 : Call LogAnaliz.salvaonto()
16 : Confirm ontology has been saved()

Figure_a 3. Sequence diagram of implemented system. "User" label means the user is using the application software, "GUI" is the Graphical User Interface class and LogAnaliz is the class which processes the text.



Figure_a 4. Graph output from the text "*The kidneys are essential in the urinary system and also serve homeostatic functions such as the regulation of electrolytes, maintenance of acid–base balance, and regulation of blood pressure (via maintaining salt and water balance). They serve the body as a natural filter of the blood, and remove wastes, which are diverted to the urinary bladder.*".



```
The kidneys  ==>  are essential in  ==>  the urinary system
The kidneys  ==>  serve  ==>  homeostatic functions
The kidneys  ==>  serve the body as  ==>  a natural filter of the blood
The kidneys  ==>  remove  ==>  wastes
wastes  ==>  are diverted to  ==>  the urinary bladder
the regulation of electrolytes  ==>  is a  ==>  homeostatic functions
maintenance of acid-base balance  ==>  is a  ==>  homeostatic functions
regulation of blood pressure  ==>  is a  ==>  homeostatic functions
```

Figure_a 5. Output when the "View extracted triplets as text" option is selected. The "==>" symbol is used to separate elements of a triplet (e.g. "The kidneys ==> remove ==> wastes" is the same as the triplet "The kidneys + remove + waste").

1. Introduction

Nowadays, the **range of available biomedical data is huge** and it is expanding quickly. This spread means that researchers have to face the effective extraction of the needed data from the large amounts of biomedical data available. To this end, biomedical researchers have started to use ontologies and terminologies to organize and to append meta-tag to their data for better searching and retrieving [1]. In Computer Science, **an ontology is a formal representation** of knowledge as a set of concepts connected by relations and properties to denote things [2]. Thus, ontologies in biomedical domains are used for organizing biological concepts and representing relationships among them [3]. Major results include the Gene Ontology (GO) [4], the Foundational Model of Anatomy (FMA) [5] and the Unified Medical Language System (UMLS) [6]. The GO [4] is an ontology used by geneticists to find every piece of available information about a specific gene. FMA [5] is an upper level ontology, which means it contains general concepts to be used by other ontologies: its main topic is the anatomy and thus every ontology about the human body could use FMA as the main source of knowledge about anatomy. UMLS is a "set of files and software that brings together many health and biomedical vocabularies and standards to enable interoperability between computer systems." [7] .

The main issue researchers are facing is the **semantic interoperability**: what does happen if one researcher uses concepts from an ontology to append tags to his research and another researcher uses similar concepts from different ontologies to search the work of the first researcher? Ontology-based applications, in order to achieve semantic interoperability, need to ‘‘harmonize” the ontology they use [8]. In literature this problem is referred to as the ontology mapping (or alignment) problem and concerns the discovering of equivalent concepts belonging to different ontologies [9]: this way researchers can reach a consistent semantic interoperability.

An ontology is for **giving semantic** meaning to stored digital data: this way information is not only the label used to describe it, but also becomes a concept which can be semantically retrieved; thus if two hospitals have two different names for the same kind of information (e.g. two different names for the same disease), but this piece of information is also linked with its meaning this two units can share the same knowledge once semantic interoperability has been reached. Semantic information can add artificial intelligence features to biomedical knowledge base (e.g. the skill to imply new statements from previous stored statements, thus to create new pieces of knowledge from available one).

An ontology can be viewed as a **Tree-like** structure where the concepts are the nodes and the relations between them are labeled edges. The graph visualization of an ontology is one of the clearer way to visualize some pieces of information about a subject. For example Figure 1 shows "*a structured description of core biomedical knowledge consisting of well defined semantic types and relationships between them*"[10] . It is from UMLS [6] and called the **UMLS Semantic Network**, we will come back on this topic in the Background. The Figure 1 has to be read from top to down: the nodes represent concepts, the lines represent "is a" relationships from the lower node to the higher (e.g. "Anatomical Abnormality" "is a" "Anatomical Structure"), the arrows represent non-hierarchical relationship (e.g. "Finding" "evaluation of" "Biologic Function").

Biomedical ontologies can be used to serve integration of clinical data [11]: biomedical researchers have a growing need to integrate data deriving from different disciplines at different granularity levels [11]. This implies that medical terminology becomes affected by an increasing concern with matters of consistency (i.e. no contradictions) [11]. By providing a common structure and terminology, the use of biomedical ontology aims at providing a single data source for review: "*furthermore, the use of such a common vocabulary promises benefits of less redundant data and easier opportunities for longitudinal studies and meta-analyses and for ensuring consistency of data across the lifetime of the patient and from one healthcare institution to the next.*" [11]. The use of biomedical ontologies promises significant rewards, for example for clinical decision support, because they improve the information retrieval process [11].
Nowadays, almost every biomedical ontology is **created by hand**: domain experts build an handmade ontology for their own purposes. Ontology creation is an hard and time consuming task, which requires lot of human intensive work: it is still easier to use some already created ontologies than create an ontology for a specific purpose. This way, we run into the biggest issue of ontology subject matter: if the pieces of knowledge we need are split in two ontologies, it is possible to merge them but it is quite hard.

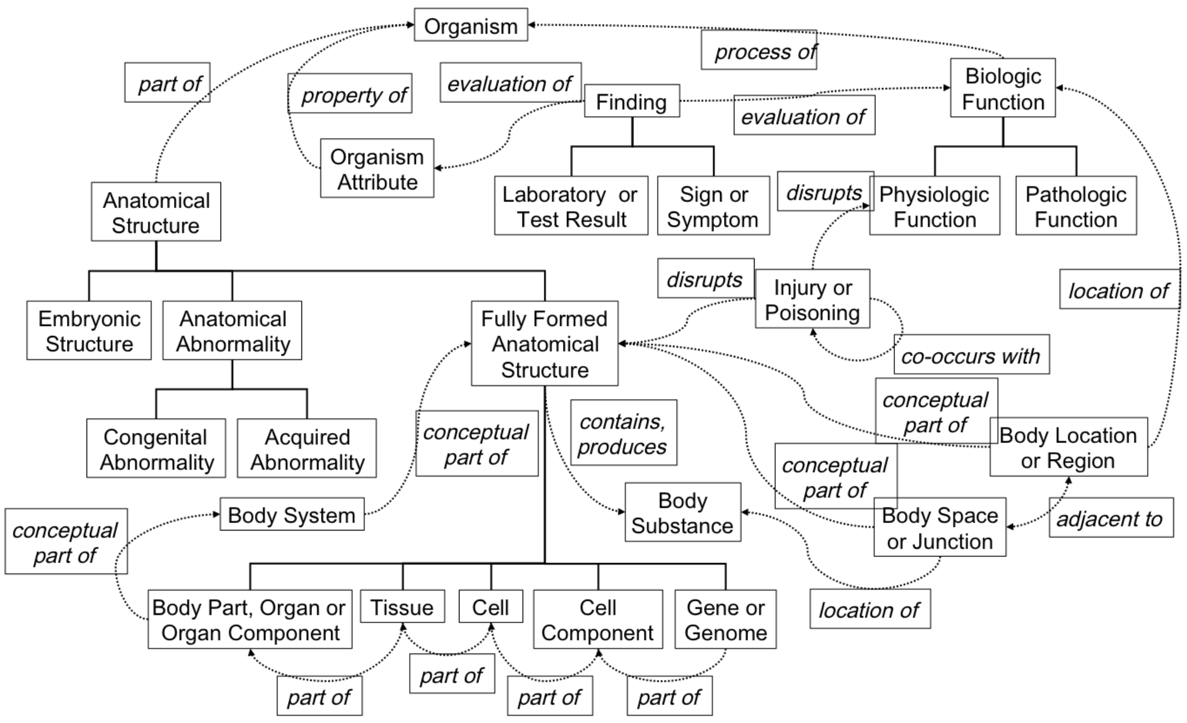


Fig. 1. Unified Medical Language System semantic network [6]. **Every line means an "is_a" relation from the bottom of the Figure to the top (e.g. "Pathologic Function" is a "Biologic Function").**

At present, an algorithm for ontologies mapping which perform well in every medical domain and which is really automatic has not been created yet: every algorithm scores good results with some ontologies but has very bad one with others (e.g. an algorithm could work well mapping human-anatomy-specific ontologies and not so well in mapping mouse-anatomy-specific ontologies); for example Khan and Keet [12] tested some algorithms in mapping foundational ontologies (also known as upper level ontologies, a foundational ontology is an ontology which describes very general and not-domain-specific concepts): these algorithms found less than a third of all the available alignments among them.

The object of this work is **the design**, **the model and the implementation of a software system for creating biomedical ontologies automatically** starting from text in Natural Language. It is based on extraction of triplets from texts; a triplet is composed by two different concepts (Domain and Range) linked together by a Relation: “Domain + Relation + Range”. In a sentence the Domain could be the Subject, the Relation could be the Verb and the Range could be the Object. Furthermore, we designed and developed a Graphical User Interface to facilitate the use of the system.

2. Background

In this paragraph, ontologies subject matter and natural language processing are described.

2.1 Ontology subject matter

A natural model is a simplified description of some aspects of reality: it is used to understand, to structure or to predict parts of the real world. An ontology in Information Technology is the attempt to create a natural model of a part of the real world. It is a machine-processable specification of the topic with a formally defined meaning. Thus the word “ontology” refers to a formal representation of knowledge. In the following paragraphs, we present ontologies subject matter and their formal logic. We are also expand some concepts view in the Introduction chapter, such as what is an ontology, what does mean adding semantic to data, how an ontology is usually created and what is the ontologies mapping. Last subparagraph presents some remarkable biomedical ontologies (GO [4], FMA[5] and Unified Medical Language System UMLS [13]).

2.1.1 Introduction to ontologies

The word “ontology” derives from ancient Greek and means the study of being [14]. Parmenides was among the first to propose an ontological characterization of the fundamental nature of reality [14]. For Plato, the ontologies should be derived from some observations of reality [14]; Aristotle developed ten categories to classify all things that may exist, with some subcategories to further specify each of them [14] .Thus ontologies can be used to describe, classify and structure pieces of knowledge about a main topic.

One of the fittest definition of ontology is the one proposed by Gruber in 1993: ontologies are “explicit specification of a shared conceptualization” [15]. The phrase “explicit specification” refers to the logic layer under the ontologies subject matter; the “conceptualization” describes the main role of an ontology, to create an abstract model of some phenomenon of the world; the word “shared” point out the aim of ontologies: they must be a way to share knowledge; an ontology captures consensual knowledge: it is not restricted to some individual, but accepted by a large group [16].

As previously reported in Information Technology (IT) the word “ontology” refers to a formal representation of knowledge constituted by a set of concepts and their properties. Properties could link couples of concepts together or could be referred to a single concept (e.g. Apoptosis is the process of programmed cell death [17]: every cell destroy itself physiologically; thus a cell can be modeled as an item which has the relation “destroy” with itself). When a property links two different concepts, we call this link “Relation”. Two concepts linked together by a relation form a triplets (a.k.a. statement): first concept is called Domain and last concept is called Range (we will use “Domain + Relation + Range” layout to write statements). Thus an ontology is a set of rules to store and structure knowledge about a topic: it contains concepts and properties organized in statements and individuals; an individual is a instantiation of a concept (e.g. “Andrea” is a “person”, is not a concept) [18]. Since the knowledge has been represented and stored, it can be processed through logical operations such as logical deductions [18]. Logical deduction relies on a set of domain-independent rules to create new statements starting from stored ones: they are domain-independent in the sense they provided template-like ways for inferring knowledge in which the placeholders could be substituted by domain concepts. For example, a property is transitive if: if A is related to B through a transitive property and B is related to C trough same transitive property, then A is always related to C trough same transitive property, where A, B and C are concepts. Transitivity could be encoded as this set of rules:

```
If
    $A\ R_1\ B$
    $B\ R_1\ C$
and if
    $R_1$ is transitive
then
    $A\ R_1\ C$
where
    A,B,C are concepts;
    $R_1$ is a property (or Relation).
```

An easy example is a user who wants to extract knowledge both from FMA [5] and GO [4]: he/she wants to study the Craniofacial dysostosis, a syndrome caused by the gene FGFR2 and characterized by early fusion of the bones of the skull and face; he/she would like to extract the name of every bone next to a malformed one. In FMA [5] exists spatial relations such as “next to” or “close to”: to allow the user to extract the whole information, the name of the bones from GO [4] has to be mapped to the name of the same bones inside FMA: this way

if the bone “A” is next to bone “B” (information from FMA [5])

if the bonce “C” is malformed (information from GO [4])

if bone “A” is the same bone as bone “C” (mapping between FMA [5] and GO [4])

then the bone “B” is next to a malformed bone (implied statement from statements of both ontologies).

Thus, using both ontologies the system can imply the bone “B” is “next to a malformed bone”: this statement has not been manually added in any of the two ontologies.

When the knowledge of a domain is represented with this declarative formalism, the set of objects that can be represented is called the universe [15]: the ontology does not consider concepts outside its universe, and every statement belonging to the universe is to considered true by the ontology. The point is an ontology could contain wrong pieces of information from an external point of view (e.g. the statement “Men + are not + mortal” is wrong), but it would consider them true a priori: its task is to decide if a conclusion is impliable from previously stored statements or not; its task is not to decide if a stored statement is true or not. The only check ontology does on stored statements is the consistency check (i.e. it checks if there are contradictory statements).

An ontology could be viewed as a graph where the nodes are concepts and the edges are relations. For example in Figure 2 is shown the ontology created from the text: *”The kidneys are essential in the urinary system and also serve homeostatic functions such as the regulation of electrolytes, maintenance of acid–base balance, and regulation of blood pressure. They serve the body as a natural filter of the blood, and remove wastes, which are diverted to the urinary bladder.”*. “The Kidneys” and “wastes” are concepts linked by the relation “remove”.

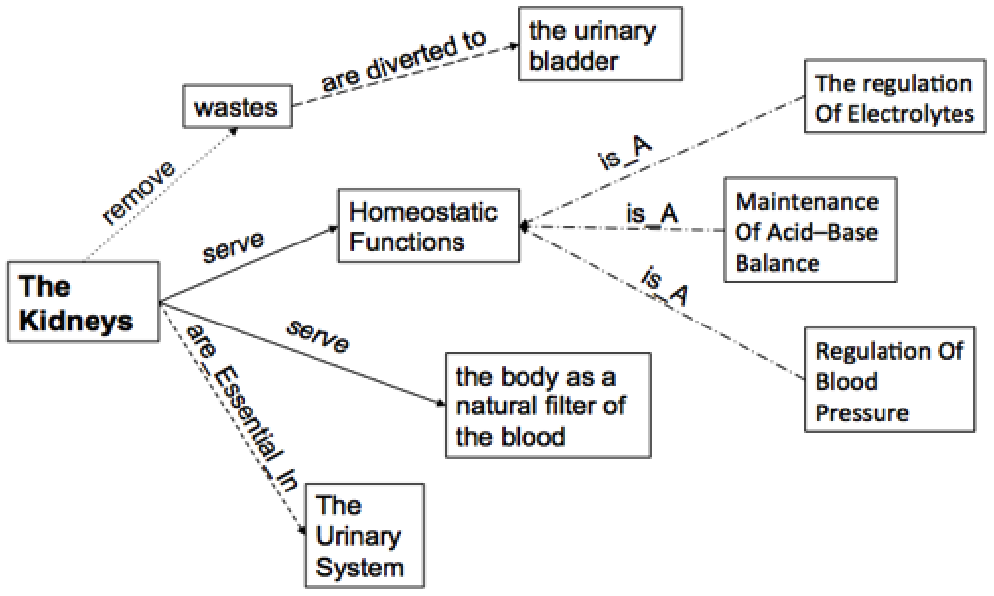


Fig. 2. Ontology extracted from the text: *"The kidneys are essential in the urinary system and also serve homeostatic functions such as the regulation of electrolytes, maintenance of acid–base balance, and regulation of blood pressure. They serve the body as a natural filter of the blood, and remove wastes, which are diverted to the urinary bladder."*. Rectangles represent concepts while arrows represent relation among them.

2.1.2 Some examples of biomedical ontologies

Biomedical ontologies are widely used by researcher to annotate their data with ontology terms for better data integration, and to increase the interoperability between data repositories [19].

One of the most notable biomedical ontologies is the **Gene Ontology** (GO) [4]: *"The goal of the Gene Ontology Consortium is to produce a dynamic, controlled vocabulary that can be applied to all eukaryotes even as knowledge of gene and protein roles in cells is accumulating and changing."* [4]. It was born in 1998 and it is still growing: GO is a collaborative project to address the need for consistent descriptions of gene products in different databases [4]. It describes gene products in terms of their associated biological processes, cellular components and molecular functions in a species-independent manner. Thus the main aim of GO is to facilitate data integration and information retrieval in the Gene domain. It is also used as the standard ontology to annotate gene subject matter texts. At present, GO mainly focus on three areas: cellular component, molecular function and biological process. These areas are considered independent each other. The GO contains over 30,000 terms and it is freely downloadable [20].

The **Foundational Model of Anatomy** (FMA) [21] is a upper level ontology about the anatomy. Upper level ontology means that it "contains information about anatomical entities, which are independent elements of biomedical reality on which physiological and disease processes depend, and which, in response to etiological agents, can transform themselves into pathological entities" [5]. Every biomedical ontology designed to describe non-anatomical domains must anyway refer to anatomical entities: for this reason FMA [5] is largely used. FMA concerns with the representation of concepts and relations necessary for the symbolic representation of the phenotypic structure of the human body in a form that is both understandable to humans and machine readable. It has been developed and is being maintained by the Structural Informatics Group at the University of Washington, Seattle WA. FMA purpose is to serve as a foundational ontology for other biomedical applications: it is not designed as an end-user application for anatomy students, teachers or any other particular user group. It is a symbolic representation of the phenotypic structure of the human body. Over 75,000 anatomical classes ranging from macroscopic to molecular level are organized in an Aristotelian-type concept hierarchy [5]; it also contains over 170,000 terms, and over 2.4 million statements from over 227 type of relations [21].
The Unified Medical Language System (UMLS) [7] has been developed by National Library of Medicine (NLM) since 1986. UMLS aims "*to enable interoperability between computer systems*" [7]: if they use different vocabularies for some reason, and both vocabularies belonging to UMLS, its aim is to give mapping between concepts. It is a repository of over 160 biomedical vocabularies and sources, and contains over 1,900,000 biomedical terms and 800,000 concepts [6]. Among contained vocabularies, there are the International Classification of Disease 9 and 10 (ICD9 and ICD10) [22], the Systematic Nomenclature of Medicine – Clinical Terms (SNOMED-CT), the Medical Subject Headings (MeSH), and Gene Ontology (GO) [20]. Concepts are linked by relations in more than 19 millions of statements. UMLS is composed by the UMLS Metathesaurus, the Semantic Network, and the SPECIALIST Lexicon. Metathesaurus is a repository of inter-related concepts: it contains every word or phrase of every UMLS vocabulary, grouped by meaning; every group of words and phrases with the same meaning forms a synonym class [6]. The Semantic Network contains the relations of every Metathesaurus concept [10].

SPECIALIST Lexicon contains stems and lexical rules to build medical words and multi-terms medical words [6]. Low layer contains Information Sources Map (ISM): it contains information about the physical implementation of the data-base: thus it permits UMLS to answer to user queries.

2.1.3 Description Logic, First Order Logic and Web Ontology Language

Every ontology is built on some kind of formal logic framework, like the First Order Logic (FOL) [23] or some sub-Logic derived from it (e.g. Description Logic DL) [23]. This is the reason why an ontology is not a sort of relational database but has far more knowledge management possibilities [14]. Relational databases are built by the Structured query language [24]. This language does not offer any kind of logical deduction. A relational database is composed by linked tables by means of some attribute values [24]. With an ontology first, is possible to find every disease name in an external disease ontology, such as UMLS [6]. Then it is possible to use every relations such as "is localized in" and "afflicts" to discover every concept which is a sub concept of "disease" and which has these relations with "heart" concept. The rule to express person with cardiac problem becomes "person which has a disease which is localized in the heart or it afflicts the heart".

The First Order Logic (FOL) is a kind of mathematical logic [23]: it is a formal system where formulas of a formal language may be used to represent propositions. The FOL allows the user deriving some theorems from inference rules and axioms: this operation is called implication. FOL is composed of three elements: 1- a formal language to model a part of the natural language, 2 - a semantics to establish if a statement of the language is true or false, and 3 - a reasoner to determine the validity of theorems. A formal language is defined by means of a dictionary of symbols (logical symbols, descriptive symbols, and structural symbols) and a grammar. A grammar is a set of rules to connect symbols in a "valid" way [25]. To assign semantics to a logical language means to define an approach for determining whether an argument is valid (if this argument is entailable from other axioms). A reasoner is used to establish if a theorem is impliable from the axioms. Thus, the question our logic would like to answer is "Does some selected axioms imply a specific theorem?": this kind of question is linked to the "Decidability problem"[25]. An axiom is a valid statement (i.e. it is considered true). The Decidability problem is modelled in Figure 3: the "$\frac{A}{T}$" symbol means "Does A imply T ?". The system could replies in three different ways: “Yes” , “No” , or just not terminate (it means the system does not know the answer).

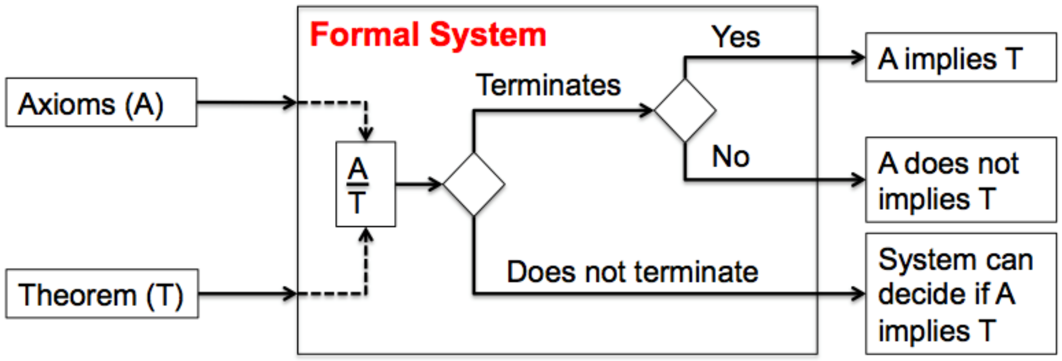


Fig. 3. The decidability problem for a Formal System.

The point we do not know is how many loops the system needs to give an output: until we do not see the output, we can not establish if the process will terminate or not. If the system always replies to every answer within a finite number of loop, it is called “decidable”. If the system always replies within a finite number of loop to every answer when the answer is “Yes”, it is called “semi-decidable”. Otherwise is called “undecidable”. A formal system could have two properties to determine its decidability: the “soundness” and the “completeness ” [25].

A system R is sound if, and only if: if R derives the conclusion φ from the statements K, then K entails always (implies) φ.

A system R is (strongly) complete if, and only if: if K entails φ, then R is guaranteed to derive φ from K in a finite number of reasoning steps.

Prove if a formal system is complete is an hard task: the (not) completeness of FOL has been proven by Kurt Gödel in 1929, as part of his PhD thesis [26]. In every mathematical logic there is a trade-off between expressivity and decidability: the more the logic is expressive, the less it is “decidable”. Thus, ontology uses a part of FOL, with a smaller expressivity, called Description Logic (DL): it is a “decidable” logic [27]. Description Logic (DL) can be identified as a decidable fragment of FOL. In DL, building blocks are classes, roles and individuals: classes are concepts. Every class could have one or more roles: roles are relation between one class and itself or between it and other classes. Individuals are specific instantiation of a class (e.g. if in ontology universe exists a person Rudy Studer, thus “Rudy Studer” is a instantiation of the class “Person”). Every class can be combined through symbols: conjunction ($\sqcap$), disjunction ($\sqcup$) and negation ($\neg$).

DL is different from other logics (like Propositional Logic PL [23]) as its use quantified variables. There are two quantifier: Existential Quantifier (∃) and Universal Quantifier (∀). The first quantifier means every class which has some kind of relation with others. The second quantifier means every class which is linked to another class with a specific relation or it does not have that relation at all; quantifier can be qualified (the class with which the relation exist is specified) or unqualified: universal quantifier is always "qualified". Thus:

- unqualified existential $\exists hasExaminer$: means every class which has the relation "hasExaminer" with at least one other class. For example $Exam \sqsubseteq \exists hasExaminer$ means the class Exam is a subclass of every class which has the relation "hasExaminer" with another class (every exam has at least one examiner);
- qualified existential $\exists hasExaminer.Professor$: means every class which has the relation "hasExaminer" with the Professor class. For example $Exam \sqsubseteq \exists hasExaminer$.Professor means every Exam always has as examiner a Professor and always has an examiner.
- Qualified universal $\forall hasExaminer.Professor$: means every class which or it does not have any relation "hasExaminer" or it has this relation with the "Professor" class. For example $Exam \sqsubseteq \forall hasExaminer$.Professor means only professor could be the examiner of an Exam (but also means every class which does not have the relation "hasExaminer", it is a superclass of "Exam" class).

To write DL statements Web Ontology Language (OWL) is commonly used [27].

OWL is a family of knowledge representation languages: they are markup computer languages for defining ontologies [27]. OWL is characterized by formal semantics and a lot of different possible serializations (syntaxes) for the Semantic Web: a serialization is a way to encode a triplets-like information in a machine-readable way (as ontologies can be represented using tree-like graphs). The simplest serialization is named "Turtle": it uses RDF to encode triplets in a machine readable file. RDF is a family of World Wide Web Consortium (W3C) specifications originally designed as a metadata data model [18]. It is a formal language: its goal is to enable applications to exchange data on the web while preserving their original meaning. The base idea of RDF is to store ontologies as triplets of resources: every resource is identified with a Uniform Resource Identifier (URI); an URI could be a online resource (identified though a Uniform Resource

Locator URL) or an item, like a person, a event, a concept, not accessible through the web. It is based on XML for describing structured information: XML is fundamental data format for data exchange and electronic publishing which is widely in use [28]; it is a Markup language. The Turtle representation of RDF can easily be processed by machines but it is no the most commonly used serialization [18]. One reason for this might be that many programming languages do not offer standard libraries to manage RDF. As of today, the main syntax for Ontology is the XML-based serialization named "XML/RDF" because several libraries to process XML language exist [18].

Some different versions of OWL exist as shown in Table 1: the more used is OWL2 DL. They are sorted for expressivity (OWL Full is the most expressive) [27]: the expressivity of a language is the range of ideas that can be represented in that language [27].

Table 1. Ontology Web Language version, their decidability and based-on logic.

Language Name	**Decidability**	**Based on**
OWL Lite	full decidable	Description Logic
OWL DL	full decidable	Description Logic
OWL2 DL	full decidable	Description Logic
OWL Full	semi-decidable	First Order Logic

2.1.4 Ontologies and Semantics

Ontologies add semantic meanings to stored strings: this allows computers to intelligently search [29], combine [30], and process [31] these contents based on the meaning that this contents have to humans.

Semantic search is defined as a information retrieval process that exploits somehow domain knowledge [29]. Domain knowledge can be formalized through an ontology; thus, if we store the knowledge through an ontology, lot of problems of traditional search scenario, such as synonymy (i.e. same concept can be expressed with several words) and word acceptance (i.e. same word can have several meanings), do not exist: in a single ontology, every concept must be unique and have only one "mean" (the concept itself is its meaning). If we use more than one ontology for our search, we should have mapped them before merging (we will explain more deeply mapping problems in par. 2.1.6), thus every concept is again unique. If we want to discover every possible information about a topic, the easier way is to find the best word or short sentence (also known as circumlocution) to describe this topic, and then search on a Google-like search engine: the search engine mostly relies on the occurrence of searched words (or similar one) in documents. This approach has two big problems: synonyms and term acceptances. If we want to extract every little piece of information about a topic, we have to know every synonym and every circumlocution to express a term: there is no string similarity between "heart", "ticker" and "cardiac pump". Second issue is more complicated: same word can have more meanings. If we consider a researcher who is studying the allelic variations in a gene, the search issue becomes clearer [1]: he/she would like to discover every piece of information about that gene, such as every clinical trials that have studied diseases related to that gene, or all the pathways that are affected by that gene. The knowledge needed to address such questions could be available in some public biomedical resources, such as articles indexed in PubMed/Medline [32]; the problem is finding that information. Now consider a user who uses a semantic search approach [29] and needs fundamental information on organs. he/she inputs the keywords 'introduction' and 'organs' to his/her semantic search engine. An ontology lookup tells the system that the term 'organ' can have more meanings: for example, organ could refer to organ pipe or to human organ. It prompts the user if he/she is looking for information on organs related to anatomy or music. According to the user's answer, the system finally retrieves introductory documents about human organs. It not only returns documents that contain the term 'introduction', but also documents that contain 'overview' or 'fundamentals'; it not only returns document that contain the term 'organ', but also documents that contain "vital part" or "functional human structure".

Adding semantic meaning to a data allows to combine more efficiently that data with an external source. Merging different data sources is called **Data-Exchange** (or Data Integration): it is the problem of taking data structured under a source schema and creating an instance of a target schema that reflects the source data as accurately as possible [30]. Data exchange is used in many tasks that require data to be transferred between independently created applications: for example, if we want to merge pieces of information of the same patient created in different hospital we have to merge them somehow. This is also important for progress in large-scale scientific projects, where data sets are being produced independently by multiple researchers, and in offering good search quality across the millions of structured data sources on the World-Wide Web [33]. In Data-Integration scenario ontologies create great advantages as they provide a shared and common understanding of a domain that can be communicated across different application systems [16]. An example of data integration through ontologies is the work proposed by Berges et al. [34] They proposed the use of formal ontology to achieve the interoperability among Electronic Health Records (EHRs) from different clinical centers in Spain; their proposal allows one system to interpret on the fly clinical data sent by another clinical center even when they use different representations [34]. They proposed a double layer approach: an upper level ontology which is a canonical representation of EHR statements and some lower level ontologies, one for each clinical center. Lower level ontologies describe the particular EHRs data storage structures used by every specific center: a translator module from SQL language to OWL language has been developed to facilitate the creation of the lower level ontologies [34].

One example of **knowledge processing** is the logical deduction (2.1.1). Rubin et al. demonstrated the capabilities of logical deduction [31]: they created a system to infer about the consequences of penetrating injuries. They created a knowledge model of the chest, and the heart anatomy and physiology. Then they used a domain-independent classifier to infer ischemic regions of the heart as well as anatomic spaces containing ectopic blood after a penetrating injury. Given a set of anatomic structures that are directly injured by a projectile, they wanted to create a reasoning application that deduces secondary injuries (regions of myocardium that will be ischemic if a coronary artery is injured, and propagation of injury as bleeding occurs into damaged anatomic compartments that surround the heart). They applied this reasoning service to try to infer the effects of some injuries and the results were confirmed by a physician: “Our results suggest that inferring the consequences of penetrating injury can be formalized as a classification task. There are benefits in using OWL as a representation language.” [31]. Ontologies offer to enhance extracting information service: if a user want to extract some tags from a text, he/she must read and find out best concepts to describe what he/she just reads [16]. This is avoidable thanks to an “annotator”: it is a tool which can automatically assigns concept from an external ontology to a text the open biomedical annotator [1]).

An **annotator** is a tool to associate metadata to sentences or phrases (i.e. an annotator could assign the concept “7088”, which represent the “heart” concept in the Foundational Model of Anatomy [21], to the sentence “The adult human heart has a mass of between 250 and 350 grams and is about the size of a fist.”): every author of a paper chooses some keywords to describe his/her paper content. This kind of metadata is pretty useless: in fact, freely appending terms from a domain to describe something results in a “personalized” list of descriptors. A step forward in annotation is to force every author to use keywords from one standardized ontology of terminology. Last step in annotation problem is to use automatic annotator: the idea is to create some rules to automatically map phrases in a text with concept from an “external” ontology. This way, we can select which external ontology use, or process a large amount of papers with a specific ontology to extract every paper with some particular concepts. Jonquet et al. proposed the Open Biomedical Annotator [1] to annotate biomedical texts.

2.1.5 Ontology development

Several different approaches to ontologies development have been carried out [35]. All these approaches follow three steps:

4. Searching for a exhaustive set of terms
5. Organizing the terms into a taxonomy of classes
6. Choose which of the taxonomy display in the ontology

Ontology creators usually use two possible approaches based upon the direction of ontology construction; in bottom-up approach they start with some descriptions of the domain and obtain a classification [35], while in top-down one, they start with an a priori abstract view of the domain itself [35]. The real problem is every ontology is an handmade product, created from a team of experts (domain experts and IT experts): *“Although ontologies have been proposed as an important and natural means of representing real world knowledge for the development of database designs, most ontology creation is not carried out systematically.”* [36]. In addition, nevertheless the large number of existing ontologies, there is no state-of-art methodology for building them. We can make a comparison with database development field, where the conceptual modeling of every database follows standardized steps, such as the creation of an Entity-Relationship (E–R) model. Once every step has been followed, the creation of a database is almost automatic. In the ontologies subject matter does not exist any similar tool to E-R model or a standardized workflow: thus any team of experts will use their own approach. However, some best-practice tutorials exist ([37] or [38]).

Protégé and Web Protégé [39] are commonly used ontology editor to build ontologies: they are user friendly software which can permit anyone to build his/her own ontology. If the user does not deeply know Description Logic and Ontologies subject matter, he/she can easily build inconsistent ontologies, or ontologies full of mistakes, without being aware of that. For example, if you want to say there is an heart inside every human body, you could use both quantifiers (see 2.1.2 for an explanation on quantifiers):

$$(1)\ human\_Body \sqsubseteq \exists contains.heart$$

or

$$(2)\ human\_Body \sqsubseteq \forall contains.heart$$

The expression (1) means "$human\_Body$" class is the subclass of every class which has a "contains" relationship with the "heart" class. The expression (2) means people are subclass of every class which has an "contains" relationship with the "heart" class or which not has the "contains" relation at all. Thus in (2) the heart itself is a human body, because no heart can contains another heart inside it: this kind of sneaky errors are very common if the user does not deeply know ontologies subject matter from a technical point of view. Another relevant issue is every ontology has a lot of classes, thus developing an ontology is a time consuming process: some approaches to speed-up this have been proposed [40]; these approaches exploit Natural Language Processing-NLP (we will explain deeply in par. 2.2) to reduce the difficulty of the ontology building process.

2.1.6 Ontology mapping task

With the term "mapping" or "alignment" is meant an algorithm which is able to (automatically) relate every concept belonging to an ontology with every concept belonging to another one [41]. The result is given the ontologies O_i and O_j, and the concepts C_i and C_j , the algorithm creates every triplet

"$C_i + R_{ij} + C_j$"

where using symbols,

$C_i \in O_i$,

$C_j \in O_j$,

$R_{ij} \in \{\sqsubset, \sqsupset, \equiv, \perp\}$;

thus:

"$C_i \sqsubset C_j$" means "C_i" sub-concept of "C_j" (e.g. femur sub-concept of bone),
"$C_i \sqsupset C_j$" means "C_i" super-concept of "C_j" (e.g. organ super-concept of kidney),
"$C_i \equiv C_j$" means "C_i" is the same concept of "C_j" (e.g. heart same concept as ticker),
and
"$C_i \perp C_j$" means "C_i" is disjoint with "C_j" (e.g. blood is disjoint with hair).

Nowadays ontologies mapping is essential: ontologies are often developed to model little domains of the biological science. This leads to greater expressivity in knowledge representation, but also creates a new problem in information sharing. Software applications using different ontologies have to face interoperability issues because relationships between concepts in different ontologies are not explicitly stated.

A state-of-art algorithm for ontology mapping does not exist: researchers are proposing a lot of mapping algorithms. There are four main issues in almost every algorithm proposed.

First, every algorithm proposed **shows different performances based on the domain topics**: for example an algorithm built to map efficiently couples of foundational ontologies could score bad results with couple of ontologies about other topics, such as mouse anatomy [12]. As we already explained, a foundational ontology is an ontology which describes very general and not-domain-specific concepts [12].

Second, it does not exist **any totally automatic algorithm**: every algorithm is semi-automatic in two different ways. Some algorithms such as PROMPT [42] need some interactions with the user during their executions. Other algorithms require some domain experts to check if the new relations are correct and if every possible relation has been established.

Third, **almost every algorithm has the same approach**: they just calculate similarity among string with some string similarity metrics [43]. A string similarity metric is a measure of similarity or dissimilarity (distance) between two text. For example, a good similarity string metric algorithm have to score high result with "The kidneys" and "The human kidneys" inputs (because they have the same meaning), and it has to score low result with "Hearth" and "Heart" inputs (because they have different meanings). A perfect similarity score metric does not exist, thus the choice of one algorithm rather than another could change the performance of the mapping algorithm and the similarity score metric could work very well in some topics and not so well with other topics.

After a bibliography research, we have reviewed some existing algorithms (Table 2).

Three significant evaluation parameters for mapping algorithms have been used by authors to test their developed tool: they are precision, Recall and F-measure. They are defined as follow.

Precision (p) is: $p = \frac{(\#rel_{corr} \sqcap \#rel_{tot})}{\#rel_{tot}}$,
where $\#rel_{corr}$ means number of correct relation found, and $\#rel_{tot}$ means total relation found.
Recall (r) is: $r = \frac{(\#rel_{corr} \sqcap \#rel_{tot})}{\#rel_{corr}}$.
F-measure (F_M) is: $F_M = 2 * \frac{p*r}{p+r}$.

In Table 2 this background research is shown.

Table 2. Existing mapping algorithms and their performances.

Algorithm / Tool name [Ref.]	**Algorithm description**	**Test corpus**	**Performance**

SURD [41]	A corpus P of text is annotated using the ontologies-to-map $O_i \; and \; O_j$, obtaining two annotated corpus $P_i \; and \; P_j$.
	The idea is to evaluate how many time a couple of concepts belonging to different ontologies is used to annotate same phrase.
	The evaluation is achieved with the score metric Co-Annotation Ratio: $CAR_{pq} = \frac{\#C_p^* \cap \#C_q^*}{\#C_p^*}$, where "$\#C_i^*$" means times where C_i^* is used to annotate a phrase, and "$\#C_i^* \cap \#C_j^*$" means times where both C_i^* and C_j^* are used to annotate same phrase.
	If $CAR_{ij} \geq 0.5$ and $CAR_{ji} < 0.5$, $C_i \sqsubset C_j$.
	If $CAR_{ij} < 0.5$ and $CAR_{ji} \geq 0.5$, $C_i \sqsupset C_j$.
	If $CAR_{ij} < 0.5$ and $CAR_{ji} < 0.5$, $C_i \perp C_j$.
	If $CAR_{ij} \geq 0.5$ and $CAR_{ji} \geq 0.5$, $C_i \equiv C_j$.
		SURD has been tested on two corpus, first from Gene Regulation Ontology (GRO http://www.ebi.ac.uk/Rebholz-srv/GRO/GRO.html) , second from GENIA (http://www.medlingmap.org/taxonomy/term/102) (both available at http://nlp.sce.ntu.edu.sg/SURD)	p = 0.866
			r = 0.577
			F_M = 0.693

Marquet et al. [44]	The algorithm follows four steps: 1. Synonyms acquisition: every concept name of both ontologies is related with synonyms concept from Unified Medical Language Systems (UMLS) concept. 2. If two concepts $C_1 \; and C_2$ from different ontologies has same synonyms, the algorithm creates a relation C_1 "is_a" C_2 3. The algorithm creates a list of couple of concepts from different ontologies where one concept name is included in the other one. 4. Starting from the list created at the third step, the system looks for every couple in some UMLS [7] tables to create new relations.	Four different ontologies from "Open Biological and Biomedical Ontologies" (OBO http://www.obofoundry.org). OBO is a ontologies web-repository.	It has been created a "result ontology", built mapping the four ontologies. 131 new relations has been discovered with a precision of 100%.

Algorithm / Tool name [Ref.]	**Algorithm description**	**Test corpus**	**Performance**

BOAT [45]	The algorithm consider two kind of matching:
	- Trivial matches: couple of concepts with same normalized name.
	- Non-trivial matches: couple of concepts with same meaning and different names.
	The algorithm follows three steps:
	1. Trivial matches discovery: every concept is normalized (e.g. every plural is converted in singular) and linked to its synonyms; then the algorithm uses some "exact string matching" algorithms to discover equivalent concepts.
	2. Candidate selection: every concept is represented through a Vector Space Model (VSM) which contains words used to annotate the concept itself and parent the concepts (in a graph tree visualization of the ontology). Then a similarity score is calculated from every couple of concept as the number of identical element of their VSMs.
	3. Non-trivial matches discovery: every couple of concepts which has been scored a high similarity score at step 2 is processed. Every item of the VSM is converted in a token: every couple of identical tokens (one for concept) are erased. Every couple of tokens where one token is equal to a synonym of other token are erased. If there is no more token, this couple is considered equivalent.
		BOAT has been tested on some OBO ontologies.	p = 0.98
			r = 0.8
			F_M = 0.88

OAANN [46]	The similarity score S among two different concepts is defined as:
	$S = w_1 * s_1 + w_2 * s_2 + w_3 * s_3$
	where w_1, w_2 and w_3 are weights and s_1, s_2 and s_3 are three different similarity metrics.
	s_1 considers difference among the labels,
	s_2 considers the number of equivalent properties,
	and s_3 is the similarity score S among the couple of closer concepts more similar.
	The three weights are calculated through a machine learning approach and $w_1 + w_2 + w_3 = 1$: the training phase is done on 30 couples of equivalent concepts.
		OAANN has been tested with a couple of ontologies: BiologicalProcess (13922 concepts) and Pathway (571 concepts).	p = 0.89
			r = 0.85
			Number of discovered relations: 120 (on 139)
			Number of correct discovered relations: 108.

Algorithm / Tool name [Ref.]	**Algorithm description**	**Test corpus**	**Performance**

OntoMAS [47]	OntoMAS uses one ontology (the one with lesser concepts) as base ontology and maps it with another ontology. It is a multi-agent system: every agent has assigned one task and does not know what other agents are doing.
	OntoMAS creates one Alignment Request Agent (AR) for every concept of base ontology and one Concept Resource Agents (CR) for every concept of the second ontology. OntoMAS also creates some String Matching Resource Agents(SMRs) and some Linguistic Matching Resource Agents (LMRs): they are used by CRs to interact with every AR. SMRs are used to discover concepts with similar normalized label while LMRs are used to discover couple of concepts which have similar synonyms.
	Every agent uses a common “knowledge module”, constituted by some external sources (e.g. Wordnet) and domain specific rules.
		One ontology created from the concepts of HARTI and one from the concepts of WikiGoviya.	p = 0.67
			r = 0.61

Belhadef et al. [48]	It is a four stages algorithms to maps concepts of two different ontologies: 1. String normalization (e.g. every upper case to lower case, every hyphen is deleted, ...) 2. Syntactical verification: Syntactic similarities among concepts are scored. 3. Structural verification: Structural similarities among concepts are scored. 4. Bidirectional verification: it is done to make a matching in both directions of the two ontologies in question to eliminate the ambiguity in the candidates found	No test corpus is specified.	Authors does not make any test on their algorithm.
Cotterell et al. [49]	The algorithm starts creating a new graph C from the two ontologies A and B: every node of C contains a couple of concepts (one belong to A and one to B). Then the algorithm creates some edges between some nodes of C (it creates an edge between two nodes if and only if both couples of concepts belonging to A has some kind of relation in A, and both couples of concepts belonging to B has some kind of relation in B). Then the algorithm sets a weight for every edge: the weighted sum of the edges of a node is the node similarity score (thus the similarity score of contained concepts).	Ontology Alignment Evaluation Initiative (OAEI)	In the paper, authors created several diagrams to show their good results.

Algorithm / Tool name [Ref.]	**Algorithm description**	**Test corpus**	**Performance**
Nasir et al. [50]	This algorithm is based on the "**cognitive support and visualization for semi-automatic ontology mapping**"(CogZ) to create base mapping relations. CogZ creates some possible mappings relations as inter-edges between the two ontologies: the user only has to select right one and to discard others. The authors adds an Edge Bundling approach to inter-edges. **Edge bundling** is the process of distorting the shapes of the edges in a graph to provide paths that are easier for the human eye to follow. By bundling edges that have sources and destinations in common regions of the graph, many individual edges are replace with a smaller set of bundles, "cleaning up" the clutter in the visual representation. Thus using Edge bundling is easier for the user to confirm some proposed mapping relations and to discard others: the final outcome is an expectation of more efficient and effective analytical reasoning, decision-making, and problem-solving. Edge Bundling is achieved **using a spring approach**: every edge is divided in some segments, and then for each couple of edge segments, a spring is attached between these segments. To each segment is also associated a virtual electrostatic force. Springs create attractive forces while couples of electrostatic forces creates repulsive ones: to achieve the best edge bundling, the system attempts to **minimize the forces** exerted upon the edge segments due to the springs and electromagnetic forces.	This approach **has been tested** mapping a couple of ontologies: 24 testers has been called. 12 has used only CogZ while 12 has used CogZ with Edge Bundling.	This approach diminished mapping task time by 75% for the validation of proposed alignment. The time spent for discovering missing relations is the same. The precision was the same, both achieves 100% (this underlines the testers had worked hard).

Algorithm / Tool name [Ref.]	**Algorithm description**	**Test corpus**	**Performance**
Anchor-PROMPT [51]	Anchor-PROMPT is a tool of PROMPT suite: it treats an ontology as a graph with concepts as nodes and relations as links. It takes as input a set of anchors: an anchor is a couple of related concepts, one per ontology to map; anchors can be defined by the user or by the Anchor-PROMPT itself using lexical matching. The algorithm analyses the path in the subgraph limited by the anchors to discover semantically similar concepts: couples of concepts with similar topology receive a high score. The user can set a maximum path length: the closer is a concept to the anchor, the more similar meaning they have.	The algorithm has been tested on two ontologies: an ontology for describing organization of the University of Maryland (UMD), and an ontology for describing organization of Carnegie Mellon University (CMU). They have been created by different developers independently but they covered similar topics.	Recall is defined as the number of discovered mapping relations divides the number of total discoverable mapping relations while precision is the number of correct discovered mapping relations divides the number of discovered mapping relations. The best trade-off between precision and recall has been get with a maximum path length of 4. With this setting, the precision is 65%.
YAM ++ [52]	YAM++ follows a machine learning approach to map two ontologies. YAM++ has three steps: 1. It uses some similarity metric scores to evaluate similarity among every couple of concepts. 2. It considers topological structure to create more mapping relations. 3. It uses Alcomo to verify relations created in the first two steps. Alcomo is an available tool (http://web.informatik.uni-mannheim.de/alcomo/) to implement the global constraint optimization method.	Some Ontology Alignment Evaluation Initiative (OAEI) 2011 benchmark has been used to test YAM++	YAM++ scores a precision of among 78% in proposed texts.

2.2 Element of Natural Language Processing

Natural Language Processing (NLP) is a computerized approach for analyzing untagged electronic texts [53]. Allen et al. defined NLP as *"a theoretically motivated range of computational techniques for analyzing and representing naturally occurring texts at one or more levels of linguistic analysis for the purpose of achieving human-like language processing for a range of tasks or applications"* [53]. We have to examine in depth this definition. First, the notion of "*range of computational techniques"* is imprecise on purpose because several different methods to accomplish a particular type of language analysis exist [53]. Second, with *"Naturally occurring texts"* authors refer to any kind of text [53]. Third, *"Human-like language processing"* underlines NLP tries to duplicate Human mental approach to understanding the meaning of a text. Fourth, the notion of *"levels of linguistic analysis"* refers to the fact that the same text can be analyzed at different linguistic level [53]. Seven different levels of linguistic analysis exist: 1 Phonology (it is the interpretation of speech sounds), 2 Morphology (it deals with the componential nature of words, which are composed of morphemes), 3 Lexical (it is the interpretation of the meaning of individual words), 4 Syntactic (it focuses on analysing the words in a sentence to uncover the grammatical structure of that sentence), 5 Semantic (it determines the possible meanings of a sentence by focusing on the interactions among word-level meanings in the sentence), 6 Discourse (it focuses on the properties of the text as a whole that convey meaning by making connections between component sentences), and 7 Pragmatic (it utilizes context over and above the contents of the text to increased the extracted information from a text) [53]. Humans normally use all of these levels since each level conveys different types of meaning: thus a *"Human-like language processing"* must try to use all of them [53]. So, the goal of NLP is *"to accomplish human-like language processing"* [53]. In our work we focus mainly on Lexical, Syntactic and Semantic levels [53].

To perform NLP, some approaches exist: Symbolic approaches and Statistical ones are the more common. Firsts are based on logic, set of rules, or semantic networks, while seconds are based on machine learning techniques [53].

3. Methods

3.1 An approach to Triplet Extraction

The aim of this paragraph is to describe the approach for the development of a software application to extract triplets from natural language texts, in English.

As already explained in the Background, a triplet is composed by a Subject, also known as Domain, a Object, also known as Range, and the Predicate which links them, also known as Relation (Figure 3): this structure is similar to the clause structure (Figure 3). We will use this notation to write a statement to express a triplet:

"Domain + Relation + Range".

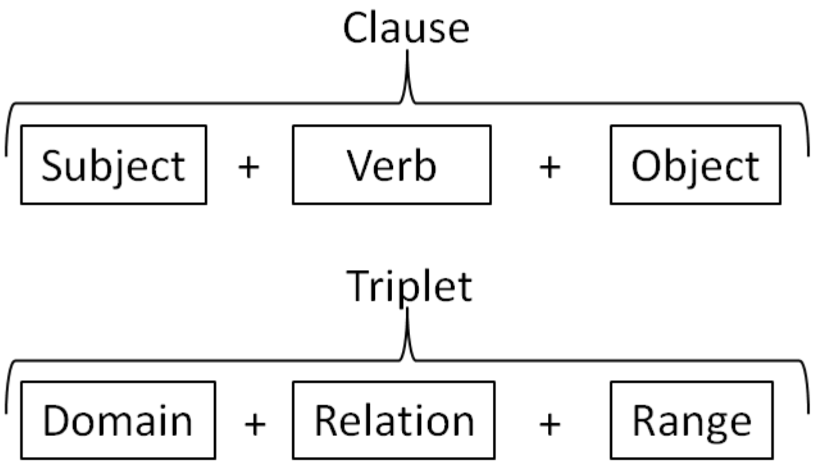


Fig. 4. Triplet and clause structure.

Because this structure is very similar to a clause one, the idea behind our approach is to exploit every piece of linguistic information embedded in every natural language text to extract triplets. We named our approach "Deterministic Triplet Extraction Approach" (DeTEA); it is named "Deterministic" because it relies on a rule-based approach; an example of what we would like to create is as follow:

1. Natural Language text: "*Cancer can be managed with removal of the kidney, or nephrectomy*."
2. Processing of the text
3. Output: extracted triplets (as shown in Figure 3):

   Cancer + can be managed with + removal of the kidney

   Cancer + can be managed with + nephrectomy

4. Output: semantic network (Figure 5).

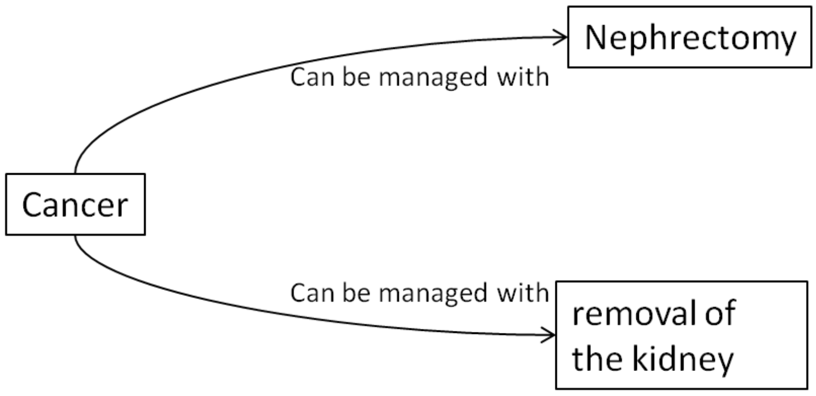


Fig. 5. Triplets extracted from the sentence: *"Cancer can be managed with removal of the kidney, or nephrectomy."*.

Thus the first idea behind our approach is we want to find one Verb Phrase (VP) and two Noun Phrases (NP) because triplets and clauses have similar structures (Figure 3) and in clauses objects and subjects are Noun Phrases and verbs are Verb Phrases; thus:

1. the VP will be the Relation;
2. first NP will be the Domain: it will be one of the NP on the left to the VP;
3. second NP will be the Range: it will be one of the NP on the right to the VP.

To extract phrases from text, we plan to use Statistical Natural Language Tools.

3.1.1 Elements of English Linguistic

To understand our efforts and our choices, it is important to understand how a sentence is created and how to gather every piece of grammatical information from it.

A list of the English linguistic structures [54] is shown in Figure 6: words are the smallest unit; words are grouped in phrases; phrases are grouped in clauses which are grouped in sentences; a group of sentences composes the discourse. Phrases and clauses are related with our work.

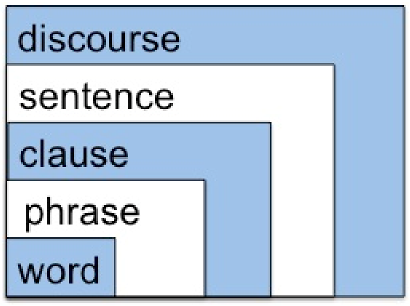


Fig. 6. Structures of English Linguistic: every element is composed by a combination of smaller one (e.g. a clause is composed by a combination of phrases and word).

The smallest building block of English language is the **word** (Figure 6) [55]. Words can be classified in nine categories, also known as *word main classes* or *Parts Of Speech* (POSs): 1) Noun (N), 2) Verb (VB), 3) Adjective (Adj), 4) Adverb (Adv), 5) Pronoun (Prn), 6) Determiner (Det), 7) Numeral (Num), 8) Auxiliary (Aux), 9) Preposition (P) and 10) Conjunction (C).

These categories group words together according to particular characteristics which they share and indicating which labels use from the previous list to use when referring to certain words.

A step forward we find the **phrase** [56]: a phrase is a sequence of at least one word. Every phrase has only one main word, called *head word*: it is a lexical item which is central to the phrase in the sense that some crucial information would be missing without it; a phrase without an head would seem structurally incomplete (this is the way to discover which word is the head one). The POS tag of head word implies the phrase type, but not every word class can originate a phrase. Five different kinds of phrase exist [56]:

1. Noun Phrase (NP)
2. Verb Phrase (VP)
3. Adjective Phrase (AdvP)
4. Adverbial Phrase (AdvP)
5. Prepositional Phrase (PP).

Every kind of phrase is composed by a fixed structure with some mandatory parts and some non-mandatory ones.

1. The **Noun Phrase (NP)** is the most complex phrase in English: a grammatically correct NP starts with a not mandatory determiner, followed by a not mandatory pre-modification part, followed by the head word which is mandatory and can be followed by a not mandatory post-modification part. Determiners are only found in Noun Phrases and they occur only at the beginning of it [54]. A pre-modification part is almost always composed by one or more Adjectives. Nouns as well as adjectives can be part of pre-modification. Post-modification is more complex because both clauses and phrases can be part of it. An head noun can be post-modified with Prepositional Phrases, relative clauses, some Adverbial Phrases, that-clauses or comparative clauses. For example, *"these large sugary doughnuts filled with jam and cream"* is a complete and correct NP where "*these*" is the

determiner, *"large sugary"* is the pre-modification, composed by two adjectives, *"doughnuts"* is the head word and *"filled with jam and cream"* is the post-modification part, composed by a verb at past participle, a preposition and two nouns [56].

2. A **Verb Phrase (VP)** can be simple and consists of just a lexical verb (which may be a multi-word verb) or it may include one or more auxiliaries up to a maximum of four. The modal auxiliaries are used to add shades of meaning, such as obligation (must) or possibilities (might). A verb phrase will not be longer than six elements, like "*might have been being told off*", where *"might"* is a modal auxiliaries, *"have been being"* is the primary auxiliaries and *"told off"* is the head word, a multi-word verb [56].
3. and 4. **Adjective Phrase** and **Adverbial Phrase** are not dissimilar in their range of possibilities for pre- and post-modification. Often, both consist only of the head word: they both could be enriched with a pre-modification part and a post-modification part. Head words are often pre-modified by a single adverb, normally an intensifier, such as *"incredibly unconfortable"* where *"incredibly"* is the pre-modification of the head word *"unconfortable"*. Very occasionally a AdjP or a AdvP is post-modified by an adverb, such as "*enough*" or "*indeed*". More typically, they are post-modified by a Prepositional Phrase [56].
5. A **Prepositional Phrase** is composed by the head word, a preposition which has to be accompanied by another element, or *prepositional complement*. Most typically, the *prepositional complement* is a NP or an AdvP [56].

From now on, we will use this notation

$$\{A\}_{xP}$$

to represent the type "xP" of the phrase "A".

The occurrence of a phrase within another one is referred to as ***embedding*** [56]. For example, consider the phrase $\{$a very serious injury$\}_{NP}$: the pre-modification part is composed by an AdjP. This is clear if you consider "very" is referred to "serious" and not to "injury" (as it should be if very is only the premodification of the NP): thus the correct POS tagging is $\{$a $\{$very serious $\}_{AdjP}$ injury$\}_{NP}$. The embedding greatly increase the complexity of automatic analysis of English. The phrases can also be linked together in two specific ways: by ***coordination*** (the joining together of two linguistic units on an equal footing) [56] and by ***apposition*** (two adjacent phrases of the same type which refers to the same object) [56]. As we will explain better later, this three kinds of expressions are very hard to manage for an automatic system and are the main source of its mistakes.

Clauses are unit of syntactic construction formed by phrases" [57]. A verb element is central to a clause: it contains always at least one Verb Phrase (VP) normally preceded by a subject element and followed by any elements needed to make the clause grammatically complete. Every clause must have at least a finite verb: a verb is called finite if it has a tense and a person. Clauses are made up of a combination of phrases with the role of clause element. The elements of a clause are: Subject (S), Object (O), Verb (V), Complement (C) and Adverbial (A). Six types of verb exist [57]:

1. Transitive
2. Intransitive
3. Ditransitive
4. Complex Transitive
5. Copular
6. some few other verb with very peculiar structure.

Every verb type enforces a minimal fixed clause structure (Figure 7): for example, a Ditransitive verb requires two different Objects with different semantic role.

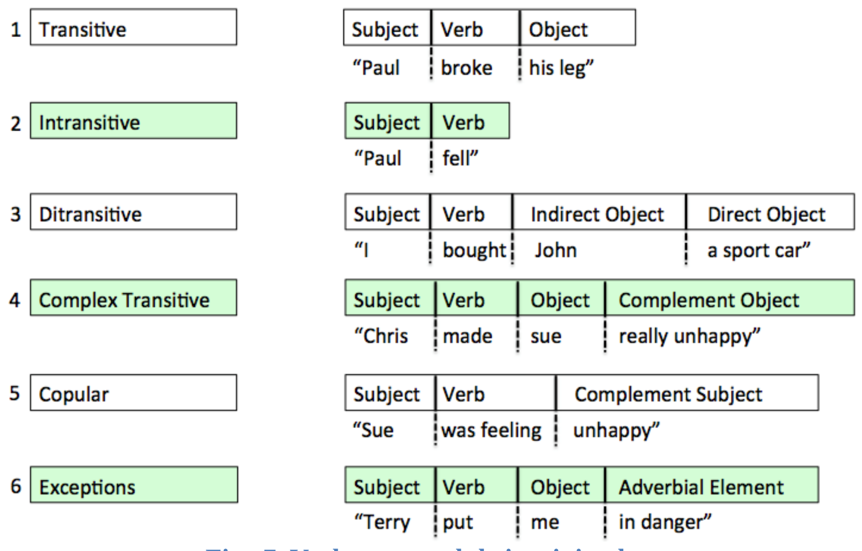

1 Transitive
Subject Verb Object
"Paul broke his leg"
2 Intransitive
Subject Verb
"Paul fell"
3 Ditransitive
Subject Verb Indirect Object Direct Object
"I bought John a sport car"
4 Complex Transitive
Subject Verb Object Complement Object
"Chris made sue really unhappy"
5 Copular
Subject Verb Complement Subject
"Sue was feeling unhappy"
6 Exceptions
Subject Verb Object Adverbial Element
"Terry put me in danger"

Fig. 7. Verb types and their minimal structure.

Every kind of phrase can have any kind of role in the clause. In addition a VP could be either a Subject (e.g. "*To heal* is the firs task of a clinician"), a Verb ("The clinician *healed* the patient ") or a Object ("The clinician tried to heal the patient").

Thus, understanding the kind of clause can help us predicting identifying the structure of the sentence and understanding better the role of a phrase within a sentence.

3.1.2 Exploiting Natural Language: how our approach processes a text to extract an Enriched Phrase

In this paragraph, we explain which Natural Language Processing (NLP) tools are needed to design and then implement our Deterministic Triplet Extraction Approach (DeTEA). One of the bases of Deterministic Triplet Extraction Approach is the idea of Grammatical Patterns: the combination of Phrase type, Part Of Speech-tags and words used can lead us to unique group of word with a specific behavior and role inside the sentence.

The more basic NLP tool is a **tokenizer** (Figure 8): it just splits a sentence in an array of words (also called tokens). Once a tokenizer has split the sentence in an array of tokens, we could use this array as the input for a **Part Of Speech (POS)-tagger** [44] (Figure 8): POS-tagging is a simple and not time consuming task, but it is not enough to extract triplets from sentences. For example, let consider the sentence of Figure 5 "*Cancer can be managed with removal of the kidney, or nephrectomy.*". If we append POS-tag to every word, we obtain:

[Cancer]$_{N}$ [can]$_{VB}$ [be]$_{VB}$ [managed]$_{VB}$ [with]$_{P}$ [removal]$_{N}$ [of]$_{P}$ [the]$_{DET}$ [kidney]$_{N}$, [or]$_{C}$ [nephrectomy]$_{N}$.

POS-Tags are not enough to get the output of Figure 5 from the input text. It would be very hard trying to extract triplets using only POS-tags: there is a lack of pieces of information. For example, "can" has the VB tag, but it is a modal: this kind of information is not obtainable with POS-tags.

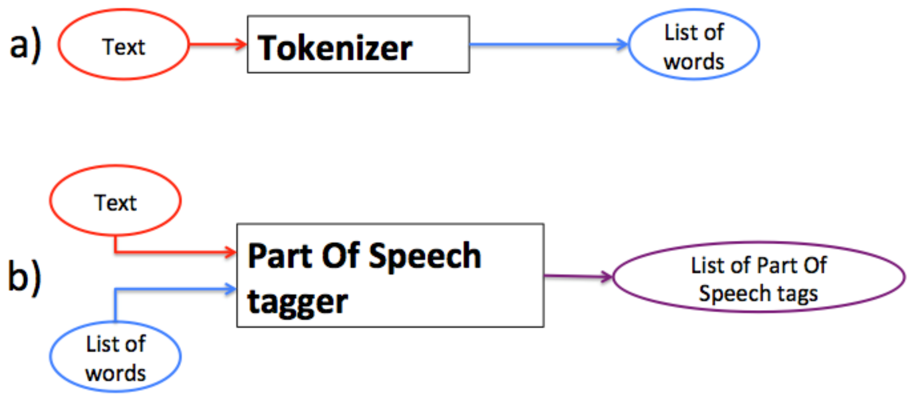


Fig. 8. a) Tokenizer and b)Part Of Speech tagger with their inputs and output.

A **Chunker** [58] (Figure 9) is a tool to merge group of words in Phrases: it uses both the output of the tokenizer (Figure 8a) and the output of the POS-tagger (Figure 8b) to create phrases. The chunker output of the input sentence "*Cancer can be managed with removal of the kidney, or nephrectomy.*" is:

[Cancer]$_{NP}$ [can be managed]$_{VP}$ [with]$_{PP}$ [removal]$_{NP}$ [of]$_{PP}$ [the kidney]$_{NP}$ [nephrectomy]$_{NP}$.

However, Chunker ignores commas and Conjunctions: this pieces of information are used to link clauses inside a sentence; for example the output for the input text "*removal of the kidney, or nephrectomy.*" is

[the kidney]$_{NP}$ [nephrectomy]$_{NP}$.

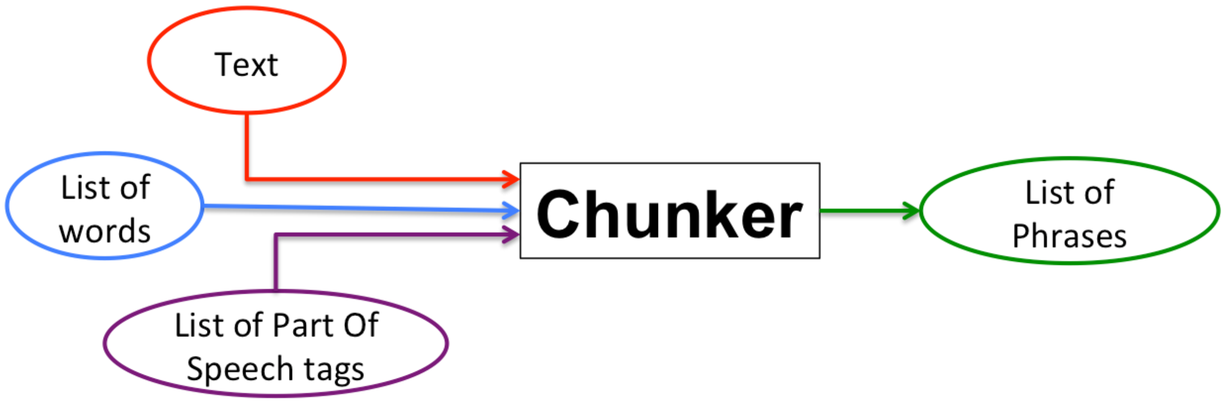


Fig. 9. Input and outputs of a Chunker, a tool to extract phrases from a text. It requires the text itself, the list of words (output of fig. 7 a) and the list of Part Of Speech tags (output of fig. 7 b).

A **Parser** [58] (Figure 10) creates linguistic trees: it considers more deeply the grammatical sentence structure. It uses the outputs of a tokenizer, a POS-tagger and a chunker.

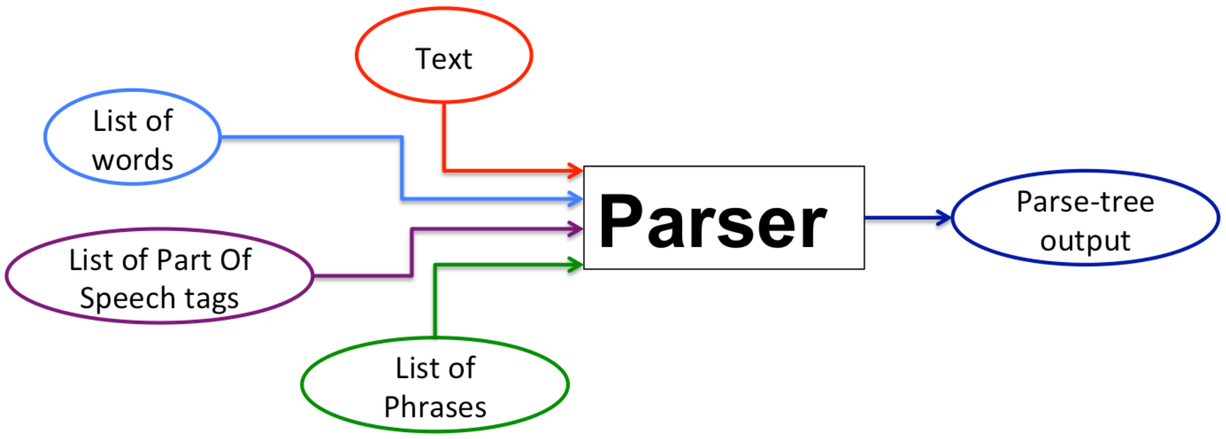


Fig. 10. Input and outputs of a Parser, a tool to extract Parser-tree from a text. It requires the text, the list of words (output of fig. 7 a), the list of Part Of Speech tags (output of fig. 7 b), and the list of phrases (output of fig. 8).

The output of a Parser is called Parser tree (Figure 11): Parsers give us every piece of information of a sentence.

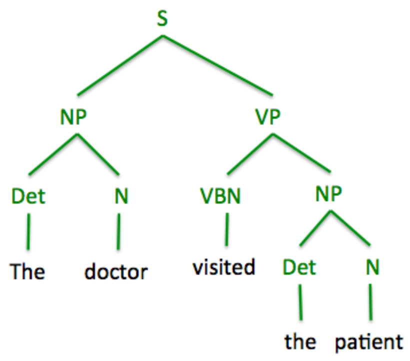


Fig. 11. Example of a Parser-tree structure for the sentence "*The doctor visited the patient*".

A Parser creates tree structures where the embedded Phrases are considered both as a Phrase and as a pre-modification (or post-modification) of another phrase; also conjunction, coordination, and subordination are displayed through the use of different level of the tree: coordinated clauses have same level while subordinated ones are closer to the leaves level.

We decided to design a new structure to store grammatical information (Figure 12): it is richer than a chunking output but is quicker than a parser from a computational time point of view (as shown in figure 10, a Parser uses the output of a Chunker to create the Parser-tree: this operation is more complex than just add some element to a Chunker output). It uses the output of a tokenizer, of a POS-tagger and of a Chunker. It creates enhanced Phrases where some items hid by Chunker are clearly exposed: commas become ","-type Phrase and conjunctions become "AndOr"-type Phrase. This way, the phrase punctuation is included into the grammatical structure. Furthermore some Phrases are merged together and other are erased, edited or moved. We called this structure “Enriched Phrases” and we called the tool to extract them “Enriched Chunker”.

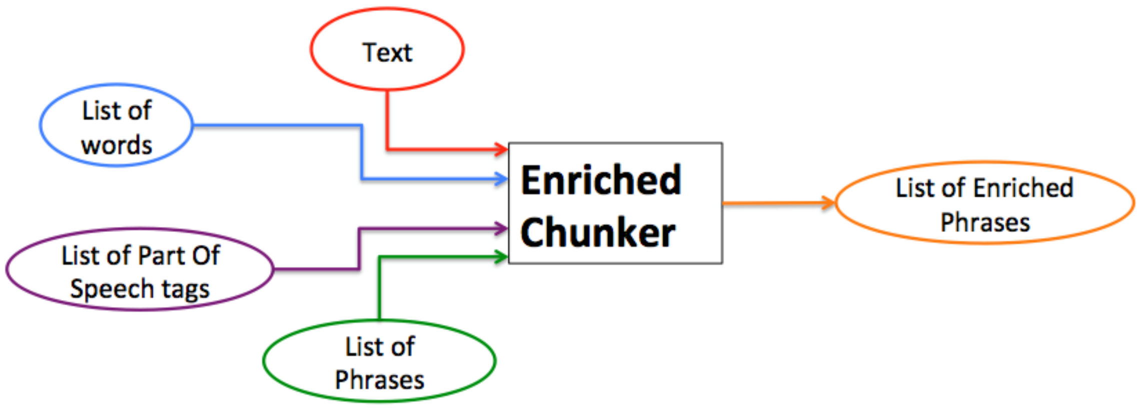


Fig. 12. Input and output of an Enriched Chunker. It requires the text, the list of words (output of fig. 7 a), the list of Part Of Speech tags (output of fig. 7 b), and the list of phrases (output of fig. 8).

3.1.3 Algorithm overview

From now on, we will call Deterministic Triplet Extraction Tool (DeTET) the desired implementation of Deterministic Triplet Extraction Approach. The input for DeTET is a medical text (e.g. a PubMed abstract, a definition from Unified Medical Language System UMLS [7],...): every text analyzed by DeTET is supposed to be grammatically correct and to contain only correct information. DeTET is composed by three different phases: a pre-processing phase, a triplet extraction phase and a post-processing phase.

There is a simple cleaning text pre-processing phase: at this point, DeTET removes some useless items inside the text, as “\n” (escape character to start new line), or unnecessary white space. Also anything inside between brackets is cleaned (this is done to avoid the extraction of incorrect triplets).

The next phase is needed to transform every sentence of inserted text in Enriched Phrases arrays (Figure 13); it starts with the splitting of input text (Figure 13 step 1) in an array of sentences (Figure 13 step 2): DeTET needs to process sentences one by one to minimize mistakes spread one sentence to another. Once sentences are splitted (Figure 13 step 3) in an array of sentences, every sentence is “tokenized” (i.e. DeTET puts every word in a words array, step 3 of Figure 13) and Part-Of-Speech (POS)-tagged (i.e. DeTET appoints a POS tag to every token, step 4 of Figure 13). In the end, DeTET uses a chunker to merge group of words in phrases (Figure 13 step 5): the chunker tool needs both the tokens array and the POS-tags array. Once DeTET has extracted every grammatical information needed, the creation of Enriched Phrases begins (Figure 13 step 6): using the rule exposed in the paragraph 3.1.2 every sentence is transformed in an array of Enriched Phrases.

The second phase is the Triplet Extraction. Starting from Enriched Phrases, DeTET try to extract as much triplets as possible. Firstly DeTET look for the first VBF, tagged 1VBF: the Domain will be the first NP on the left and the Range the first on the right. This way, every not VBF could refer to the Domain of 1VBF. Then DeTET looks for every VBF, VBN, VBG and Such_As. Every triplet also receives a Boolean tag: this tag has value "true" if that triplet has been extracted from a VBF or 1VBF, and receives a value "false" otherwise. This tag is used in the last phase of DeTET.

The last phase is the post-processing. We want to increase the meaning of every triplet to improve the performances. This phase is subdivided in some sub-phases. Firstly, we want to complete every triplets of "and VP" pattern with the correct Domain. Secondly, we want to switch every pronoun, *"which"* and *"that"* with the Noun it refers to: in the sentence *"The kidneys are bean shaped organs that serve several essential regulatory roles in vertebrate animals. They are essential in the urinary system.", "that"* and *"They"* both refers to *"The kidneys"*.

Every pronoun and "That" are substituted with the domain of the first previous triplet with the Boolean set to true. "Which" are substituted with the range of the previous triplet. Then, every triplets with a list is splitted in the correct amount of triplets. At this point, the last phase of post-processing starts. DeTET tries to collapse similar range and domain to a common label: this way, the triplets are more easy to read and to visualize. Single word similarity is an open issue of NLP. Group of words similarity are harder to manage than single word similarity: this is due to the exponential complexity of this problem.

In DeTET system, discovering semantic equivalent triplets is useful but not mandatory: thus we decided to implement an approach with the highest precision (precision is defined as the number of correct semantic equivalent triplets discovered divides the number of total semantic equivalent triplets discovered) and acceptable accuracy (accuracy is defined as the number of total semantic equivalent triplets discovered divides the number of total existing semantic equivalent triplets);

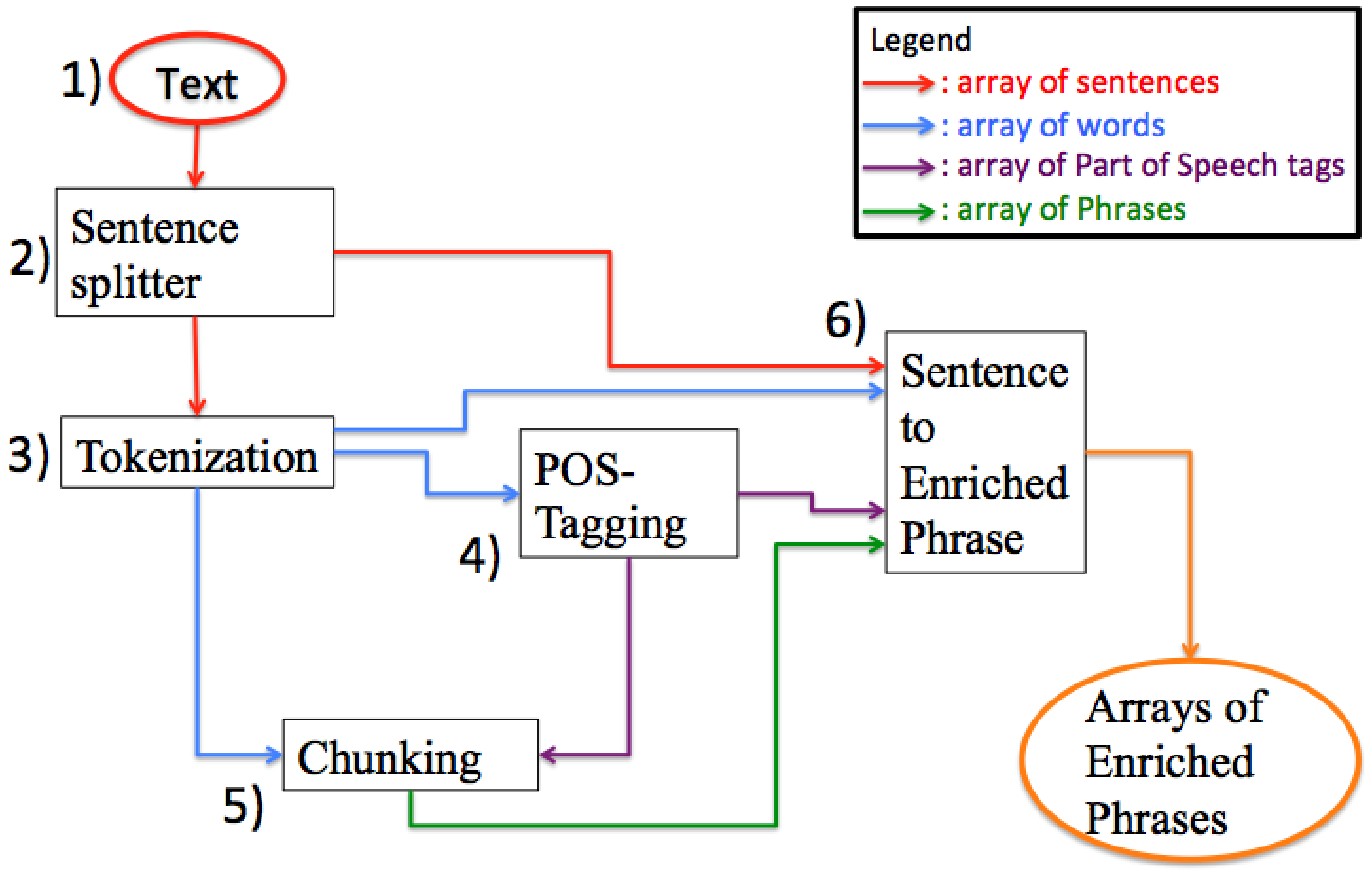


Fig. 13. Steps to create Enriched Phrases

we said “discovering semantic equivalent triplets is useful but not mandatory” because we want to collapse string if and only if we are pretty sure they are the same thing and we accept to not collapse every possible string. For example, we want “Kidneys” “The Kidneys” and “Other Kidneys” are collapsed while “you” and “your kidney” not. Our solution is based on the mean of Smith Waterman Gotoh Windowed similarity score [59] and Soundex [60] one: if the mean is higher than 0.85, the two string are semantically equivalent. This threshold has been found empirically. Smith Waterman Gotoh algorithm performs local sequence alignment: instead of looking at the total sequence, the Smith Waterman Gotoh algorithm compares segments of all possible lengths and optimizes the similarity measure [59]. Soundex algorithm is a phonetic approach to similarity: two words are similar if and only if they have similar pronunciation [60].

3.1.4 Grammatical patterns to create Enriched Phrases

Grammatical patterns manageable by the designed system are described as it follows. We want to point out the Relation is not forced to be just a verb: for example, “father_Of” is the first relation a student meets in any ontology university class [25]. Furthermore, what DeTEA would try to do is to extract the Domain and the Range of a relation, not to extract the Subject and the Object of the sentence: for example, an Adjective could be the Object (this happens with copular verb). The aim of this pre-processing phase is to make every Domain the closer NP to VP on the left and the Range the closer one on the right: this way the triplet extraction will be easier. Furthermore, a single label for every VP is not enough: the implementation of DeTEA could use several label to tag every kind of verb: when is used “VP” we mean a generic Verb Phrase, when is used VBF or VB_{ft} we means every Verb Phrase with a finite tense verb as head word; first VBF receives the special tag 1VBF. The other VP tags will be showed at the end of this list.

1) “Noun Phrase + Prepositional Phrase + Noun Phrase” pattern

“[...] the removal of the kidney is necessary”: this sentence is a clear example of a **NP+PP+NP pattern**; { *the removal* }$_{NP}$ { *of* }$_{PP}$ { *the kidney* }$_{NP}$ { *is* }$_{VP}$ { *necessary* }$_{ADJP}$. There are two different NP before the VP, both could be the subject. If we consider “*the removal”* as subject, we are missing some kind of information: the triplet “the removal + is + necessary” is grammatically correct but not so meaningful. The triplet: “the kidney + is + necessary” is grammatically not correct. What DeTET would like to extract is: “the

removal of the kidney + is + necessary" which is both grammatically correct and meaningful. The rule is: if DeTET finds "NP PP NP" pattern, it will merge those three phrases in a single NP one; from now on, we will use this layout style: "NP PP NP [...] + $\{NP, PP, NP\}_{NP}$ " means to merge the three phrases in a single one typed NP. The symbol "!" means not (e.g. "NP !PP" means NP followed by any Phrase that is not a Prepositional one). If the text of a phrase is specific (e.g. if we want to consider only "such as" and not every Prepositional Phrase), we will write the text in lower case (thus we will write for example "NP such as NP"). In the end, "[...]" means any kind of text. Thus the rule for NP PP NP pattern can be written as:

[...] NP PP NP [...] + $\{NP, PP, NP\}_{NP}$

2) "VP + PP" pattern

"Cysts can be managed with removal of the kidney" : this sentence contains a **VP+PP pattern**; { *Cysts* $\}_{NP}$ { *can be managed* $\}_{VP}$ { *with* $\}_{PP}$ { *removal of the kidney* $\}_{NP}$. First, NP PP NP pattern is applied to "*removal of the kidney"*. The triplet "cyst + can be managed + removal of the kidney" is correct but could be improved including "with" in the relation. Thus, the rule is

[...] VP PP [...] + [...] $\{VP, PP\}_{VP}$ [..].

3) "Verb Phrase + Adjectives Phrase" pattern

"Kidneys are essential in the urinary system" and "Diseases of the kidneys are diverse". This two sentences are useful to understand the **VP+AdjP pattern**. In the first sentence, the verb is not the relation between *"Kidneys"* and *"urinary system"*, which is the Range we would like to extract. In the second sentence, *"diverse"* is the Range of our relation. We would like to extract: "Kidneys + are essential in + the urinary system" and "Diseases of the kidneys + are + diverse". Thus, the rules are:

[...] VP AdjP PP NP + { VP, AdjP, PP $\}_{VP}$ NP

VP AdjP !PP + VP $\{AdjP\}_{NP}$!PP

4) "Verb Phrase Noun Phrase" as pattern

"Kidneys serve the body as natural filter" : this sentence contains a **VP NP as pattern** (it is peculiar with some specific verb like to serve or to act). The triplet "the kidneys + serve + the body as natural filter" (once again we use the NP PP NP pattern in "the body as natural filter") is grammatically correct but is worse than "the kidneys + serve the body as + natural filter", where the relation is more meaningful. Thus the rule is:

$$[...]\ \text{VP NP as NP} + \{\text{VP, NP, as}\}_{VP}\ \text{NP}$$

5) “and Verb Phrase” pattern

Sometimes the subject is not expressed. For example, in the sentence *“Kidneys are a natural filter and are essential in several regulatory roles”* there is not a subject next to *“are essential”*: the subject is the same of the other verb. Thus the rules to implement **and VP pattern** are:

$$NP_1\ 1VBF\ NP_2\ \text{and}\ VBF\ NP_3 + NP_1\ 1VBF\ NP_2\ \text{and}\ NP_1\ VBF\ NP_3$$

$$NP_1\ 1VBF\ NP_2,\ VBF\ NP_3 + NP_1\ 1VBF\ NP_2,\ NP_1\ VBF\ NP_3$$

These rules are implemented during the post-processing phase because during the Enriched Phrase conversion phase DeTET does not know which is the Subject of a verb.

6) “Adverbial Phrase” management

AdvP management requires some very specific rules. The AdvP is often ignorable without any loss in meaning. Thus AdvP is ignored unless:

- It is *“when”: “when”* and *“if”* need a more careful rules to avoid some errors, we will explain them later.
- Is the word before or after a verb: in this case, the AdvP is appended in the VP (e.g. *“Kidneys are not bones ”, “not”* is an Adverb and can not be ignored).

7) “such as” pattern

“The kidneys serve homeostatic functions such as the regulation of electrolytes”. *“Such as”* is a PP very interesting. What is the meaning of the sentence? The kidneys serve homeostatic function and one of the homeostatic function is the regulation of electrolytes. We can change the sentence this way without any loss of meaning “*The kidneys serve homeostatic functions ~~such as~~ ; the regulation of electrolytes is a homeostatic functions*”. This way we have created a new correct triplets which enriches our final performance. DeTET uses the “Such_As” tag to label the added “is a”. Thus the rule for **such as pattern** is:

$$[...]\ NP_1\ \text{such as}\ NP_2 + [...]\ NP_1,\ NP_2\ \{\text{is a}\}_{Such\_As}\ NP_1$$

As we will see later, also list of NP are processed to improve the performances. The such as pattern only considers the closer NP on the left as range of *“is a”* relation, and considers a single NP or a list of NP as domain: for example, in the sentence *“The kidneys serve regulatory roles and homeostatic functions such as the regulation of electrolytes, maintenance of acid–base balance, and regulation of blood pressure”* , the rule leads to *“homeostatic functions” as* NP1 and three different NP_2; the sentence become: *“The kidneys serve regulatory roles and homeostatic functions, the regulation of electrolytes is a homeostatic functions , maintenance of acid–base balance is a homeostatic functions, regulation of blood pressure is a homeostatic functions”* . Thus the extended rules are:

[...] NP_1 such as NP_2 + [...] NP_1, NP_2 is a NP_1

[...] NP_1,..., and NP_n such as NP0 + [...] NP_1,..., and NP_n, NP0 is a NP_n

[...] NP_0 such as NP_1,..., and NP_n + [...] NP_0, NP_1 is a NP_0, ..., NP_n is a NP_0

8) “including” and “involving” pattern

The gerund *“including”* is often used with the same meaning of *“such as”*. For example, it is true in the sentence: *”The kidneys also produce hormones including calcitriol, erythropoietin, and the enzyme renin.”.* Thus the **including pattern** is similar to such as one. The gerund could also be used in the middle of a sentence: thus we need more careful rules; if including is followed by a NP which is followed by a VP with the verb at finite tense, DeTET moves the including pattern at the end of the sentence.

NP_1 including NP_2 VP_{ft} [...] + NP_1 ~~including NP_2~~ VP_{ft} [...], NP_1 is a NP_2

NP_1 including NP_2 ! VP_{ft} [...] + NP_1 is a NP_2

“Common clinical conditions involving the kidney include the nephritic” . The gerund *“involving”* is often used to link a Domain or a Range to a subject matter. Involving is pre-processed with a similar pattern as including one, but the gerund is substituted with “linked to”. Thus the involving pattern rules are:

NP_1 involving NP_2 VP_{ft} [...] + NP_1 ~~including NP_2~~ VP_{ft} [...], NP_1 linked to NP_2

NP_1 involving NP_2 ! VP_{ft} [...] + NP_1 linked to NP_2

9) and 10) “if” and “when” patterns

“If” and “when” are pre-processed in a strict way. The part of the sentence with if or when is erased and the main part have its verb turned into a modal one. As we will see in the discussion part, conditional statement are impossible to write using the formal logic of ontologies. Thus the rule to implement that is:

If/when [...] NP1 VP NP2 + ~~If/when [...]~~ NP1 “could be ”+VP NP2
NP1 VP NP2, if/when [...] + NP1 “could be ”+VP NP2
[...] if/when [...], NP1 VP NP2 +[...] ~~if/when [...]~~,NP1 VP NP2

11) Infinitives
Infinitives are tagged with “VBinF”. A VBinF can have several roles within a sentence: it could be a subject or a object (e.g. *“To see is important for the clinician”*); it also could be used as a verb. To manage VBinF, DeTET uses this rules:

NP VBinF NP [...] + NP ~~VBinF NP~~ [...], NP {VBinF}$_{VBF}$ NP [VBinF as parenthesis]

!NP VBinF NP [...] + {VBinF+NP}$_{NP}$ [...][VBinF as NP complement]

!NP VBinF [...] + {VBinF}$_{NP}$ [...] [VBinF as Subject or Object]

12) Gerundives management
Gerundives are tagged with “VBG”. Gerundives can be used as Verb, as Noun or as Adjective. To manage VBG, DeTET uses the following rules:

NP VBG NP + NP {VBG}$_{VBF}$ NP [VBG as Relations]

VBG NP VP NP + {VBG NP}$_{NP}$ VP NP [VBG as Subject complement]

NP VP VBG NP + NP VP {VBG NP}$_{NP}$ [VBG as Object complement]

VBG VP NP + {VBG}$_{NP}$ VP NP [VBG as Subject]

13) Past Participles management
Past Participles are tagged with “VBN”. They are quite hard to manage because sometimes is impossible to understand if they are used as Verb or as Adjectives without a human understanding of the sentence. The rules for VBN are:

!NP VBN NP [...] + [Subject of the main part] {VBN}$_{VBF}$ NP
NP VBG NP + NP {VBG}$_{VBF}$ NP

14) List merging management
List merging is the last part of text pre-processing. DeTET considers a list as a sequence of NP (or enriched NP), divided by a comma. If the last two elements of a list are also divided by a conjunction like “and” or “or”, DeTET merge every NP of the list in a big one containing them all: every element of the list is separated from others with the text “DIVIDEDBY”.

During DeTET post-processing this big NP is splitted to create three different triplets as we will see later. The desired results from this process are shown step by step in Table 3.

Table 3. Envisaged output for Deterministic Triplet Extraction Tool implementation.

	First Text	**Second Text**
Input	*"Kidneys are also responsible for the reabsorption of water, glucose, and amino acids."*	*"Cancer can be managed with removal of the kidney, or nephrectomy."*
Enriched Phrase tag	{*Kidneys* }$_{NP}$ {*are also responsible for* }$_{1VBF}$ {*the reabsorption of water*}$_{NP}$ { }$_{,}$ { *glucose*}$_{NP}$ {}$_{,}$ { *and* }$_{AndOr}$ { *amino acids* }$_{NP}$	{ *Cancer* }$_{NP}$ { *can be managed with* }$_{1VBF}$ { *removal of the kidney* }$_{NP}$ { }$_{,}${ *or*}$_{AndOr}$ { *nephrectomy* }$_{NP}$
List merging phase	{*Kidneys* }$_{NP}$ {*are also responsible for* }$_{1VBF}$ {*the reabsorption of water DIVIDEDBY glucose* DIVIDEBY *amino acids* }$_{NP}$.	{ *Cancer* }$_{NP}$ { *can be managed with* }$_{1VBF}$ { *removal of the kidney DIVIDEBY nephrectomy* }$_{NP}$
Output	*1. Kidneys + are also responsible for + the reabsorption of water;* *2. Kidneys + are also responsible for + + the reabsorption of glucose;* *3. Kidneys + are also responsible for + + the reabsorption of amino acids;*	*1. Cancer + can be managed with + removal of the kidney;* *2. Cancer + can be managed with + nephrectomy;*

3.2 Deterministic Triplet Extraction Approach Design and Development

As we have explained in the Introduction or Background, Ontologies creation is still a bottleneck for a common use of semantic technologies: creating an ontology is an hard and time consuming task. Ontologies creation relies both on domain knowledge and Ontologies Web Language (OWL) knowledge: it is very uncommon a clinician has a deep knowledge about OWL and it is very uncommon a Information Technology specialist has deep knowledge about some medical domains.

3.2.1 Deterministic Triplet Extraction Approach Design and Development

Our system has been developed using the Software Engineering waterfall model with returns [61].

We envisage our system will be used by two different kind of users: biomedical researchers and consumers, as a graph representation (with not more than 15-20 nodes) is clearer and better understandable than a similar length text. Researchers could also use our system to create an ontology because creating an ontology by hand is pretty hard: this way they will have every semantic features expressed in par. 2.1.4 with a little effort. It could provide a easy way to create ontologies from a selected text: the user is “responsible” for the correctness of the text, because our system itself cannot be aware of incorrect knowledge inside an input text (as a first implementation). It will be available on every personal computer: it will not require high performance hardware, thus almost every personal computer could use it. Our system has to be as correct as possible and as fast as possible: the user would use our system in an everyday work scenario, thus it would not want to spend to much time waiting for the output. The interaction possibilities among users and system will be as low as possible: the user will be driven step by step by a clear Graphic User Interface but the whole process will be as automatic as possible. The user will only provide the text and choose how to display or save the created ontology. Because this system is not for Information Technology specialist, the system will be as user friendly as possible and as easy to use as possible.

Our approach is based on the fact that English Linguistic follows precise rules as we have reported in par. 3.1.1: a clause minimal structure is de facto fixed. Thus, the system will extract the clause structures and then it will convert them in ontology statements. We based our approach on some free available Natural Language Processing (NLP) tools. The problem is NLP tools are a machine learning tool which requires huge training sets to perform well and the creation of such structures requires thousands of hours of work. Thus our approach will rely on the performances of other NLP tools.

The system will be constituted by an implementation of the Triplet Extraction Approach we will refer to as Deterministic Triplet Extraction Tool (DeTET), and a clear Graphical User Interface (GUI). The GUI will allow the user to input the text and to see or to store the output, while DeTET will do all the work and the user will not even know DeTET exists. DeTET will be constituted by some NLP tools, such as a chunker and a Part Of Speech-tagger, by some methods to convert the text in an appropriate structure. It will also have some methods to extract triplets from the structures and some methods to post-process the extracted triplets (e.g. mapping task among extracted concepts).

Both GUI and DeTET will be implemented using Java programming language: the design of the tests will be deeply explained in par. 3.5.

3.3 The design of a Graphical User Interface for the Triplet Extraction Tool

We have designed a Graphical User Interface (GUI) to interact and use the Triplet Extraction Tool. The GUI has to allow the user to insert the input as text and to view the results. It will offer two different kind of visualization: a user could view the result as a text composed by triplets or in a graphical way (as a graph where every node is a concept and every edge is a relation between two nodes, see Figure 5).

Creating readable graph is a pretty hard task [62]; we decided to offer the user three different kind of graph visualizations: a user can view all the triplets, or every triplet related to a concept, or only the first “n” triplets (with “n" a user preselected number). The relation of triplets to a concept is made with a semantic similarity approach (we will explain deeply the algorithm of similarity measurement in 3.3.3): if the Domain or the Range score a high value with a chosen word or group of word, the triplet is shown.

We have planned to use the Fruchterman-Reingold [63], a force-directed algorithm for representing the layout. In a force-directed algorithm, every nodes is represented by steel ring with an electrical charge and every edge is represented by a spring: this way the system has a lot of forces with different directions and magnitudes. The idea is to minimize the energy of the system by moving the nodes. In the Fruchterman-Reingold algorithm, the sum of the force vectors determines which direction a node should move and the step width (how far a node is moved) is constant. The algorithm go ahead until no more node movement is needed: at that point, the energy of the system is minimized.

The System also would allow the user to save the triplets as an ontology. We will make available two different formats for the output file: “RDF/XML” format [25] and “Turtle” format [25]. “RDF/XML” [25] are a set of rules (or syntax) to express a RDF file in an XML valid style: it has been defined by World Wide Web Consortium (W3C, the main consortium of standards for World Wide Web). “Turtle” [25] is another format to express an ontology, easier to read than “RDF/XML” for a human. In addition, the user could load output files into the system for representing them.

3.4 The implementation of the system

3.4.1 The Java programming language

Deterministic Triplet Extraction Tool has been developed using Java programming language. It is object oriented. We have chosen to use Java as programming language because the majority of NLP tools are already developed as Java tools.

3.4.2 Application Program Interfaces used

An Application Program Interface (API) is similar to C language libraries: it is a list of classes and methods available to be used by a programmer. The programmer does not know how a class of an API generates the output: he/she just know how to interact with it through its methods, what kind of input every method needs, and what he/she has to expect as output.

There are several API embedded in Java Standard Edition (Java SE) and any programmer can publish their own API.

DeTET uses the Apache OpenNLP library [64] as NLP tool: the Apache OpenNLP library is a machine learning based toolkit for the processing of natural language text. It is free, open source and rich of documentation and tutorial. It uses a machine learning approach to work: the training set is available with the OpenNLP API itself as an external file. OpenNLP has been used for the POS-tagging, the chunking and the tokenization (split the word of a sentence in an array of String). OpenNLP sentencer (a tool to split sentences of a text in an array of String) score bad results: DeTET uses a simple String.split() to complete this step.

DeTET uses JUNG2 API [65]for graphical visualization of extracted triplets. Jung2 is a free API for drawing graph.

DeTET uses JENA [66] for translating triplets into a RDF/XML ontology.

The Graphical User Interface (GUI) uses JSwing API, the more common API for GUI design.

3.5 Designing Tests for a Triplet Extraction Tool

To text the performance of the system, we have selected some texts from two main sources: Wikipedia texts [67] and PubMed/Medline [32] abstracts.

First source is compatible with a non technical user who would like to increase the knowledge about a medical topic. Thus, his/her source will not be a technical one: probably, he/she will use Wikipedia [67]. We have selected the introduction of six common and less common topics: Trauma, Radiography, Pneumonia, The human body, The kidneys and Deep Brain Stimulation. We have added also the “Signs and Symptoms” paragraph of Pneumonia topics, also found on Wikipedia. Every text has been download at 02-19-2014.

Second source is composed by abstracts of some papers: this source try to simulate the general practitioner who would like to increase his/her knowledge about a specialist topic to improve his/her care quality. Ten abstracts has been downloaded from PubMed [32] and one from New England Journal of Medicine. PubMed’s ones are splitted this way: five of them are related to “Kidney” while other five are related to “Deep Brain Stimulation”. We have chosen the “Deep Brain Stimulation” topic because it is a specialist topic which could be looked for by non specialist clinician. The NEJM’s one is related to Thrombotic Event (see the appendix A for more accurate references).

A last text has been created to test Noun Phrase lists: it contains some artificially created sentences to evaluate the performances with lists. In Table 4 there is the resume of test texts: the test corpus contains 219 sentences.

Table 4. Texts for testing DeTET performances.

#	Core concept	Title	Reference	Number of sentences
1	Trauma	Introduction	http://en.wikipedia.org/wiki/Trauma_(medicine)	30
2	Radiography	Introduction	http://en.wikipedia.org/wiki/Radiography	11
3	Pneumonia	Signs and Symptoms	http://en.wikipedia.org/wiki/Pneumonia	12
4	The human body	Introduction	http://en.wikipedia.org/wiki/Human_anatomy	10
5	Thrombotic Event	Risk of a Thrombotic Event after the 6-Week Postpartum Period	The New England Journal Of Medicine [68]	8
6	Test NP Lists			4
7	The kidneys	Introduction	http://en.wikipedia.org/wiki/Kidney	17
8	Pneumonia	Introduction	http://en.wikipedia.org/wiki/Pneumonia	11
9	Deep Brain Stimulation	Introduction	http://en.wikipedia.org/wiki/Deep_Brain_Stimulation	23
10	Kidney abstract	Determination of relative Notch1 and gamma-secretase-related gene expression in puromycin-treated microdissected rat kidneys.	PubMed [69]	9
11	Kidney abstract	Chronic Kidney Disease and the Risks of Death, Cardiovascular Events, and Hospitalization	PubMed [70]	9
12	Kidney abstract	Association of chronic kidney graft failure with recipient blood pressure.	PubMed [71]	12
13	Kidney abstract	PKD1 gene and its protein	PubMed [72]	7
14	Kidney abstract	Acute Kidney Injury, Mortality, Length of Stay, and Costs in Hospitalized Patients	PubMed [73]	8
15	Deep Brain Stimulation	Deep Brain Stimulation	PubMed [74]	8
16	Deep Brain Stimulation	Deep brain stimulation	PubMed [75]	12
17	Deep Brain Stimulation	Deep brain stimulation for intractable chronic cluster headache: proposals for patient selection.	PubMed [76]	6
18	Deep Brain Stimulation	Deep brain stimulation and cluster headache	PubMed [77]	8
19	Deep Brain Stimulation	Asymmetric pallidal neuronal activity in patients with cervical dystonia.	PubMed [78]	14

We have used this metrics to evaluate the developed tool:

1) **Number of correct triplets**: a triplet is correct if and only if the Domain is the subject (or an enriched version) of the related predicate, the Relation is the predicate and the Range is the object of it. We do not link the correctness of a triplet to the meaning preservation: the meaning preservation is a subjective evaluation.
2) **Computation time**: we have used “System.nanoTime()” Java embedded method to calculate a 6 digit precision time as a float-type number. The calculation time includes also the time to build the graph visualization of every extracted triplet.
3) **Precision (p)**: it is the ratio between correct triplets and total number of them. Is a float value number included in (0,1) interval.
   a. $p = \frac{number\ of\ correct\ triplets}{number\ of\ extracted\ triplets}$
4) **Number of triplets per sentence**: we do not want to evaluate accuracy, because is not possible (we will come back at this point inside discussion paragraph). Thus we evaluate the number of triplets extracted for every sentence.

We also evaluated the performance of the global system and of the GUI.

Performance tests has been done on a 2011 21,5” iMAC (8 GB of RAM, Intel Core i3 3,06GHz). 500 Gb Hard Disk Serial ATA, Java 7.

4. Results

4.1 Deterministic Triplet Extraction Approach implementation

We have implemented our approach using Java; we have also developed a User Graphic Interface (GUI) to allow the user to use the implemented tool as a standalone software; the GUI has been implemented using Java, too.

4.1.1 Algorithm implementation: classes and methods

In this paragraph we show the results of software implementation.

The "Phrase" class implements the Enriched Phrase object (Figure 14) . It has two attributes: a String-type attribute to store the text of the Phrase (called text) and a String-type attribute to store the Enriched Phrase type (called type); for example, in{ *the removal* }$_{NP}$ the text is set as "the removal" and the type is set as "NP". It has just base methods: it has two get/set methods to read/write attributes and a "stmp()" method to print its content. It has just two constructor: the generic one (Phrase()) and the complete one (which embeds the two set methods). The "Phrase" linguistic item is implemented in the OpenNLP API within the class "Span": thus, when we refer to "Phrase" class, we will mean Enriched Phrases while when we want to refer to Phrases linguistic item we will use "Span" label.

Every extracted triplet is stored in a "Triplet" class (Figure 14): it is similar to Phrase one. It has only get/set/print methods. It has four attributes: "Domain", "Range" and "Relation" are String, while "is_principle" is a boolean tag: it is true if the triplet is extracted from a verb-finite Phrase, false otherwise.

The main class of DeTET is LogAnaliz: it does not have any attributes but has several methods (Figure 14) they are all shown. Both Phrase and Triplet classes use LogAnaliz.

The main methods is PHanalizzatore: it is the one the GUI calls to make all the work; it has just one input, the text it has to process. The output of PHanalizzatore is an ArrayList of extracted triplets. Firstly, PHanalizzatore cleans the text, erasing useless or not-processable items (e.g. brackets are ignored to avoid some extraction errors, "\n" are deleted because they just add bugs to DeTET); then it splits the text in an array of sentences: it processes every sentence one by one to minimize error spreading one sentence to another. The splitting operation is achieved using a Java Regular Expression (Java

Regex) with the embedded method String.split(Regex). A looping phase starts to process sentences one by one: we will refer to a single sentence as sentence$_i$ which means we are referring to i-esm sentence. The sentence$_i$ is splitted in an array of words using OpenNLP tokenizator tool (this array is named "tokens"); then PHanalizzatore creates an array of POS-tag (String) to link every token (word) to its POS-tag using OpenNLP POS-tagging tool (this array is named "tags"). Then DeTET extracts phrases from the sentence using "tags" and "tokens" array: it complete this operation using OpenNLP chunking tool; the phrases are stored using the Span class (thus we will have an array of Spans we will refer to as "span"). At this point, PHanalizzatore calls "span2phr(Span[], String[], String[])" methods: it uses very pieces of information obtained at previous step to extract an ArrayList of Enriched Phrases (thus, the output is ArrayList<Phrase>).

"span2phr" uses some LogAnaliz methods to look for special grammatical patterns (refer par. 3.1.4 for a list of implemented grammatical patterns) and returns an ArrayList of Enriched Phrases. It is implemented as an iterative process where every phrase (Span class) is processed to create Enriched Phrases (Phrase class). We have implemented two methods to manage every grammatical pattern (grammatical patterns are explained in par. 3.1.4). First method is called "is_grammatical_pattern_name" (e.g. is_NP_and_PP(int, Span[], String[], String[])): it uses every array previously created (tokens, tags and span) and returns a boolean; the "int" input considers the loop number to avoid redundant checks: it means, every element of span is considered just once. Second method is called "grammatical_pattern_name" (e.g. NP_and_PP(int, Span[], String[], String[])): it returns an "int" value. The value is used to set the number of span elements to merge in a single Enriched Phrase element. For example, the code of "NP_and_PP" method is:

```
static int NP_and_PP(int pos, Span[] span, String[] tokens, String[] tags)
        {
                if(is_NP_and_PP( pos, span,  tokens, tags))
                        return pos+2
                else
                return pos;
        }
```

This way, if there is a NP in position "pos", if there is also a PP in position "pos+1", and if there is also a NP in position "pos+2", the "is_NP_and_PP" method returns true and the "NP_and_PP" return "pos+2": then "span2phr" will merge text contained in span[pos], span[pos+1] and span [pos+2] in an NP-type Enriched Phrase. To merge the text of some span elements, span2phr uses a method called “stmp_span (int, int, Span[], String[], String[])”: first int is the initial span, second int is the final span, Spans array contains the phrases, first array of strings is “tags” and last is “tokens”. Thus the merged text and its relative type is added to result ArrayList using ArrayList.add (new Phrase()) method. Every time PHanalizzatore calls stmp_span(int, int, Span[], String[], String[]), it checks for commas and Conjunction: they are not part of phrases, thus they are just ignored tokens between phrases. Every time PHanalizzatore finds these items, it adds them to the ArrayList<Phrase> in the correct position and with the correct type.

Now PHanalizzatore calls “fondi_liste2 (ArrayList<Phrase>)” method: it processes the ArrayList<Phrase> to merge list of NP, as explained in 3.3.2. The output is again an ArrayList of Enriched Phrases (ArrayList<Phrase>).

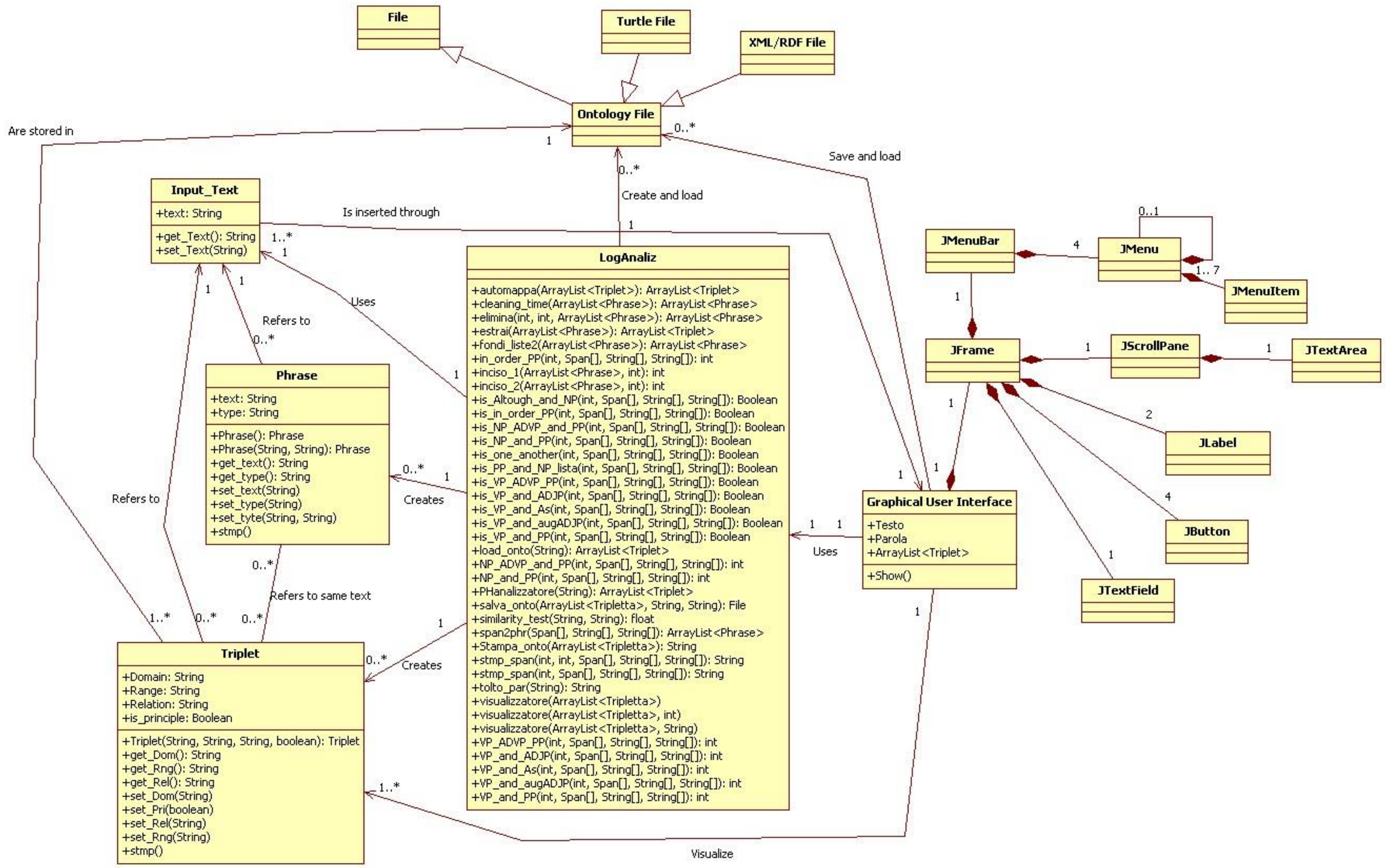
Next method called by PHanalizzatore is “such_and_ing(ArrayList<Phrase>)”: it has two tasks. First, it has to manage correctly “such_and_ing patterns”. Second, it has to manage conditional piece of sentences and piece of sentences containing “when”. The output is again an ArrayList of Enriched Phrases (ArrayList<Phrase>).

Last method called by PHanalizzatore to post-process Enriched Phrases is cleaning_time (ArrayList<Phrase>). It has to manage Gerundives, Past Participle and infinitives verb Phrases. It tries to set the right Phrase type to every non finite verb: for example, if an infinite tense verb is used as a Subject, then the method changes its type to “NP”. The output is again an ArrayList of Enriched Phrases (ArrayList<Phrase>).

Next method called by PHanalizzatore is “estrai (ArrayList<Phrase>)”: it is used to create and store triplets from the ArrayList of Enriched Phrases. Its output is an ArrayList of Triplets (thus, ArrayList<Triplet>). The extracted Triplets are raw: they have to be polished by next method, “automappa”.

Last method called by PHanalizzatore is “automappa (ArrayList<Tripletta>)”: it is a fine tuning of “estrai” method output. It swaps pronouns with corresponding nouns, it tries to discover semantic equivalent triplets with different labels: it is the last method for the triplet extraction phase and its output is the same ArrayList of triplets user will see.

“similarity_test(String, String)” is a method of LogAnaliz: it evaluates the similarity of the two input String as explained in 3.1.3.



File
Turtle File
XML/RDF File
Ontology File
Are stored in
Save and load
Create and load
Is inserted through
Input_Text
+text: String
+get_Text(): String
+set_Text(String)
LogAnaliz
+automappa(ArrayList<Triplet>): ArrayList<Triplet>
+cleaning_time(ArrayList<Phrase>): ArrayList<Phrase>
+elimina(int, int, ArrayList<Phrase>): ArrayList<Phrase>
+estrai(ArrayList<Phrase>): ArrayList<Triplet>
+fondi_liste2(ArrayList<Phrase>): ArrayList<Phrase>
+in_order_PP(int, Span[], String[], String[]): int
+inciso_1(ArrayList<Phrase>, int): int
+inciso_2(ArrayList<Phrase>, int): int
+is_Altough_and_NP(int, Span[], String[], String[]): Boolean
+is_in_order_PP(int, Span[], String[], String[]): Boolean
+is_NP_ADVP_and_PP(int, Span[], String[], String[]): Boolean
+is_NP_and_PP(int, Span[], String[], String[]): Boolean
+is_one_another(int, Span[], String[], String[]): Boolean
+is_PP_and_NP_lista(int, Span[], String[], String[]): Boolean
+is_VP_ADVP_PP(int, Span[], String[], String[]): Boolean
+is_VP_and_ADJP(int, Span[], String[], String[]): Boolean
+is_VP_and_As(int, Span[], String[], String[]): Boolean
+is_VP_and_augADJP(int, Span[], String[], String[]): Boolean
+is_VP_and_PP(int, Span[], String[], String[]): Boolean
+load_onto(String): ArrayList<Triplet>
+NP_ADVP_and_PP(int, Span[], String[], String[]): int
+NP_and_PP(int, Span[], String[], String[]): int
+PHanalizzatore(String): ArrayList<Triplet>
+salva_onto(ArrayList<Tripletta>, String, String): File
+similarity_test(String, String): float
+span2phr(Span[], String[], String[]): ArrayList<Phrase>
+Stampa_onto(ArrayList<Tripletta>): String
+stmp_span(int, int, Span[], String[], String[]): String
+stmp_span(int, Span[], String[], String[]): String
+tolto_par(String): String
+visualizzatore(ArrayList<Tripletta>)
+visualizzatore(ArrayList<Tripletta>, int)
+visualizzatore(ArrayList<Tripletta>, String)
+VP_ADVP_PP(int, Span[], String[], String[]): int
+VP_and_ADJP(int, Span[], String[], String[]): int
+VP_and_As(int, Span[], String[], String[]): int
+VP_and_augADJP(int, Span[], String[], String[]): int
+VP_and_PP(int, Span[], String[], String[]): int
Uses
Refers to
Phrase
+text: String
+type: String
+Phrase(): Phrase
+Phrase(String, String): Phrase
+get_text(): String
+get_type(): String
+set_text(String)
+set_type(String)
+set_tyte(String, String)
+stmp()
Creates
Refers to
Refers to same text
Triplet
+Domain: String
+Range: String
+Relation: String
+is_principle: Boolean
+Triplet(String, String, String, boolean): Triplet
+get_Dom(): String
+get_Rng(): String
+get_Rel(): String
+set_Dom(String)
+set_Pri(boolean)
+set_Rel(String)
+set_Rng(String)
+stmp()
Creates
Visualize
Uses
Graphical User Interface
+Testo
+Parola
+ArrayList<Triplet>
+Show()
JMenuBar
JMenu
JMenuItem
JFrame
JScrollPane
JTextArea
JLabel
JButton
JTextField
0..*
0..*
1
1..*
1
0..1
4
1.. 7
2

Fig. 14. Deterministic Triplet Extraction Tool Class Diagram.

“visualizzatore” is a method of LogAnaliz, called by GUI class: three overloaded kind of this method exists. All three methods are used to show extracted triplets as graph, using JUNG2 API. First version of visualizzatore is “visualizzatore(ArrayList<Tripletta>)”: it shows every triplet as a graph. Second is ” visualizzatore(ArrayList<Tripletta>, int)”: it shows just first “n” triplets as a graph, where “n” is the inserted value. Last version of visualizzatore is “visualizzatore(ArrayList<Tripletta>, String)”: it shows every triplet if and only if the Domain or the range scores a value greater than 0,85 using the “similarity_test(String, String)” method.

Last two interesting methods are the loading one and saving one. “load_onto(String)” is a LogAnaliz method used by the GUI class to load an already stored ontology; it accepts “RDF/XML” format or “Turtle” one. The output is an ArrayList of triplets. “salva_onto(ArrayList<Tripletta>, String, String)” is a LogAnaliz method used by the GUI class to save an ArrayList of triplets as an Ontology: the user can chooses the format (only “RDF/XML” or “Turtle”).

GUI is a class used as Graphical User Interface: it has been developed using Jswing, the Java GUI embedded builder. In 4.2 we will show its aspect and how to use it.

In Table 5 every method with a brief description, the input(s) and the output is shown, sorted by class.

Table 5. The list of methods for every developed class, with a short description of its utility. For every method are specifed both the types of the inputs and the type of the output. ArrayList<type> means an ArrayList of "type"-elements (e.g. ArrayList<String> means an ArrayList of Strings).

Method name	Input	Output	Description	Class
automappa	ArrayList<Triplet>	ArrayList<Triplet>	Triplet post-processing method	LogAnaliz
cleaning_time	ArrayList<Phrase>	ArrayList<Phrase>	Phrase post-processing method	LogAnaliz
elimina	int, int, ArrayList<Phrase>	ArrayList<Phrase>	Phrase post-processing method	LogAnaliz
estrai	ArrayList<Phrase>	ArrayList<Triplet>	Extraction of Triplets from Phrases	LogAnaliz
fondi_liste2	ArrayList<Phrase>	ArrayList<Phrase>	Phrase post-processing method	LogAnaliz
inciso_1	ArrayList<Phrase>, int	boolean	Looking for VBG inside the Sentence	LogAnaliz
inciso_2	ArrayList<Phrase>, int	boolean	Looking for VBN inside the Sentence	LogAnaliz
is_Altough_and_NP	int, Span[], String[], String[]	boolean	Grammatical pattern check	LogAnaliz
is_in_order_PP	int, Span[], String[], String[]	boolean	Grammatical pattern check	LogAnaliz
is_NP_ADVP_and_PP	int, Span[], String[], String[]	boolean	Grammatical pattern check	LogAnaliz
is_NP_and_PP	int, Span[], String[], String[]	boolean	Grammatical pattern check	LogAnaliz
is_one_another	int, Span[], String[], String[]	boolean	Grammatical pattern check	LogAnaliz
is_PP_and_NP_lista	int, Span[], String[], String[]	boolean	Grammatical pattern check	LogAnaliz
is_VP_ADVP_PP	int, Span[], String[], String[]	boolean	Grammatical pattern check	LogAnaliz
is_VP_and_ADJP	int, Span[], String[], String[]	boolean	Grammatical pattern check	LogAnaliz
is_VP_and_As	int, Span[], String[], String[]	boolean	Grammatical pattern check	LogAnaliz
is_VP_and_augADJP	int, Span[], String[], String[]	boolean	Grammatical pattern check	LogAnaliz
is_VP_and_PP	int, Span[], String[], String[]	boolean	Grammatical pattern check	LogAnaliz
load_onto	String	ArrayList<Triplet>	Loading a stored ontology	LogAnaliz
NP_ADVP_and_PP	int, Span[], String[], String[]	int	Grammatical pattern implementation	LogAnaliz
NP_and_PP	int, Span[], String[], String[]	int	Grammatical pattern implementation	LogAnaliz
PHanalizzatore	String	ArrayList<Triplet>	Main method which embed almost every other method	LogAnaliz
salva_onto	ArrayList<Triplet>, String, String	void	Saving an ontology to file	LogAnaliz

Method name	Input	Output	Description	Class
similarity_test	String, String	float	Evaluation of similarity between two multi word string	LogAnaliz
span2phr	Span[], String[], String[]	ArrayList<Phrase>	Extraction of Phrase from span, tags and tokens	LogAnaliz
Stampa_onto	ArrayList<Triplet>	String	Converting an ArrayList of Triplets in a String	LogAnaliz
stmp_span	int, int, Span[], String[], String[]	String	Extract the text inside some spans	LogAnaliz
stmp_span	int, Span[], String[], String[]	String	Extract the text inside one span	LogAnaliz
such_and_ing	ArrayList<Phrase>	ArrayList<Phrase>	Phrase post-processing method	LogAnaliz
tolto_par	String	String	Inserted text post-processing method	LogAnaliz
visualizzatore	ArrayList<Triplet>	void	Graph visualization of all extracted Triplets	LogAnaliz
visualizzatore	ArrayList<Triplet>, int	void	Graph visualization of some extracted Triplets	LogAnaliz
visualizzatore	ArrayList<Triplet>, String	void	Graph visualization of all extracted Triplets	LogAnaliz
VP_ADVP_PP	int, Span[], String[], String[]	int	Grammatical pattern implementation	LogAnaliz
VP_and_ADJP	int, Span[], String[], String[]	int	Grammatical pattern implementation	LogAnaliz
VP_and_As	int, Span[], String[], String[]	int	Grammatical pattern implementation	LogAnaliz
VP_and_augADJP	int, Span[], String[], String[]	int	Grammatical pattern implementation	LogAnaliz
VP_and_PP	int, Span[], String[], String[]	int	Grammatical pattern implementation	LogAnaliz
get_text	N/A	String	To get the text of the Phrase	Phrase
get_type	N/A	String	To get the type of the Phrase	Phrase
set_text	String	void	To set the text of the Phrase	Phrase
set_type	String	void	To set the type of the Phrase	Phrase
stmp	N/A	void	To visualize the Phrase	Phrase
get_Dom	N/A	String	To get the Domain of the Triplet	Triplet
get_Pri	N/A	boolean	To get the boolean of the Triplet	Triplet
get_Rel	N/A	String	To get the Relation of the Triplet	Triplet

Method name	Input	Output	Description	Class
get_Rng	N/A	String	To get the Range of the Triplet	Triplet
set_Dom	String	void	To set the Domain of the Triplet	Triplet
set_Pri	boolean	void	To set the boolean of the Triplet	Triplet
set_Rel	String	void	To set the Relation of the Triplet	Triplet
set_Rng	String	void	To set the Range of the Triplet	Triplet
stmp	N/A	N/A	To visualize the Triplet	Triplet

4.1.2 Workflow of Deterministic Triplet Extraction Tool

In Figure 15 is summarized the workflow of DeTET from a inner point of view, starting from a medical text inserted as a string. LogAnaliz receives the text from the Graphical User Interface and immediately pre-processes the text, erasing pieces of sentence between brackets. Then LogAnaliz calls "span2phr" method creating an array of Enriched Phrase. The Enriched Phrases class calls some methods to merge list of NP, to pre-process and to post-process the array of Enriched Phrases. Enriched Phrases class then sends the array of Enriched Phrases to LogAnaliz, which calls "estrai" method to create triplets from Enriched Phrases. The created array of triplets calls the "automappa" method itself and sends the post-processed arrays of triplets back to the Graphical User Interface.

In the appendix A is shown a step-by-step (method by method) output for the input text: *"The kidneys are essential in the urinary system and also serve homeostatic functions such as the regulation of electrolytes, maintenance of acid–base balance, and regulation of blood pressure (via maintaining salt and water balance). They serve the body as a natural filter of the blood, and remove wastes, which are diverted to the urinary bladder."*.

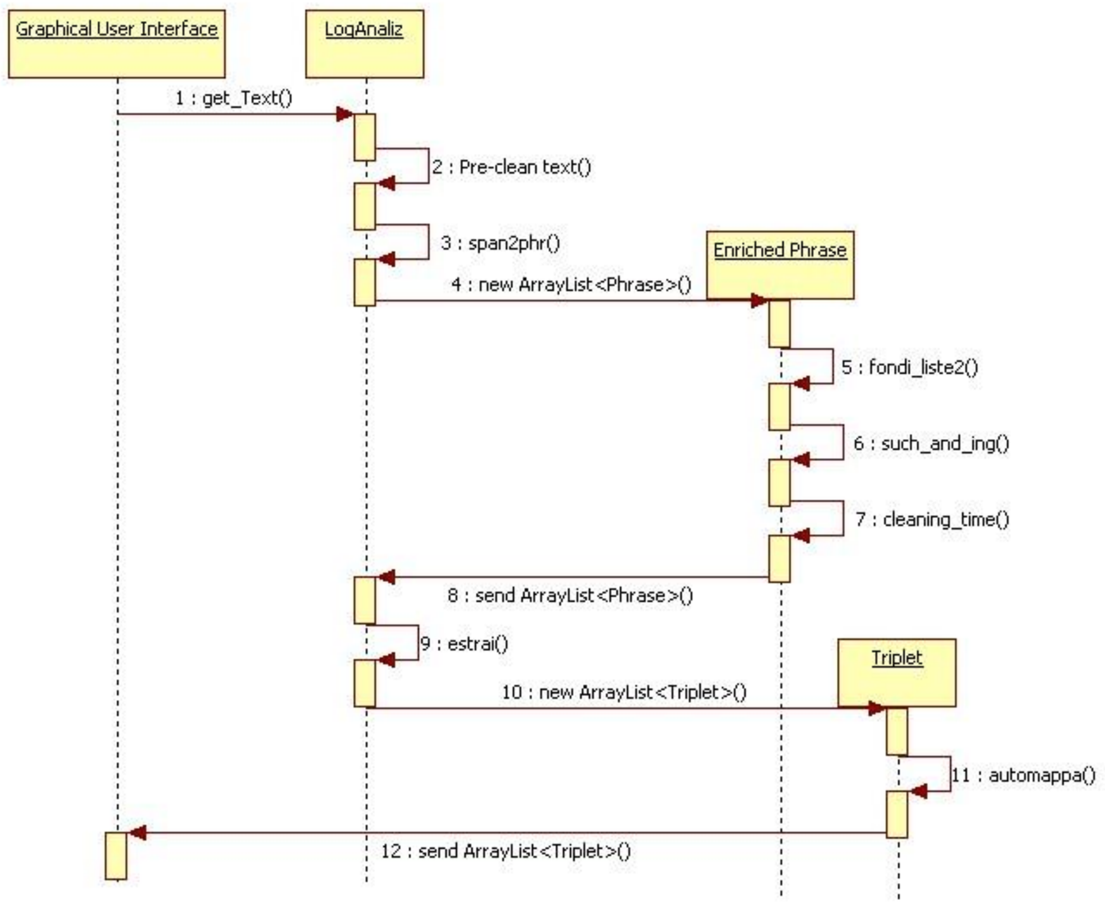


Fig. 15. Sequence diagram method calls inside LogAnaliz, Enriched Phrase and triplet classes.

In Figure 16 the activity diagram of the system is shown. In the activity diagram, we focus on what happens to the text during DeTET execution. Thus, a text is inserted by a user and the Graphical User Interface checks the correctness of the text. Then LogAnaliz starts the text processing phase, extracting an arrays of sentences. From every sentence of the array, it extracts the array of words, the array of Part Of Speech tags and the array of Phrases. Using all of these pieces of information, LogAnaliz creates an array of Enriched Phrases. The Enriched Phrases perform some pre-processing and post post-processing tasks, such as the list merging and the management of some peculiar situations (see paragraph 3.1.3). Then LogAnaliz using the polished array of Enriched Phrases, extract an array of triplets. Then the Triplet class processes the array and send the post-processed array to the Graphical User Interface.

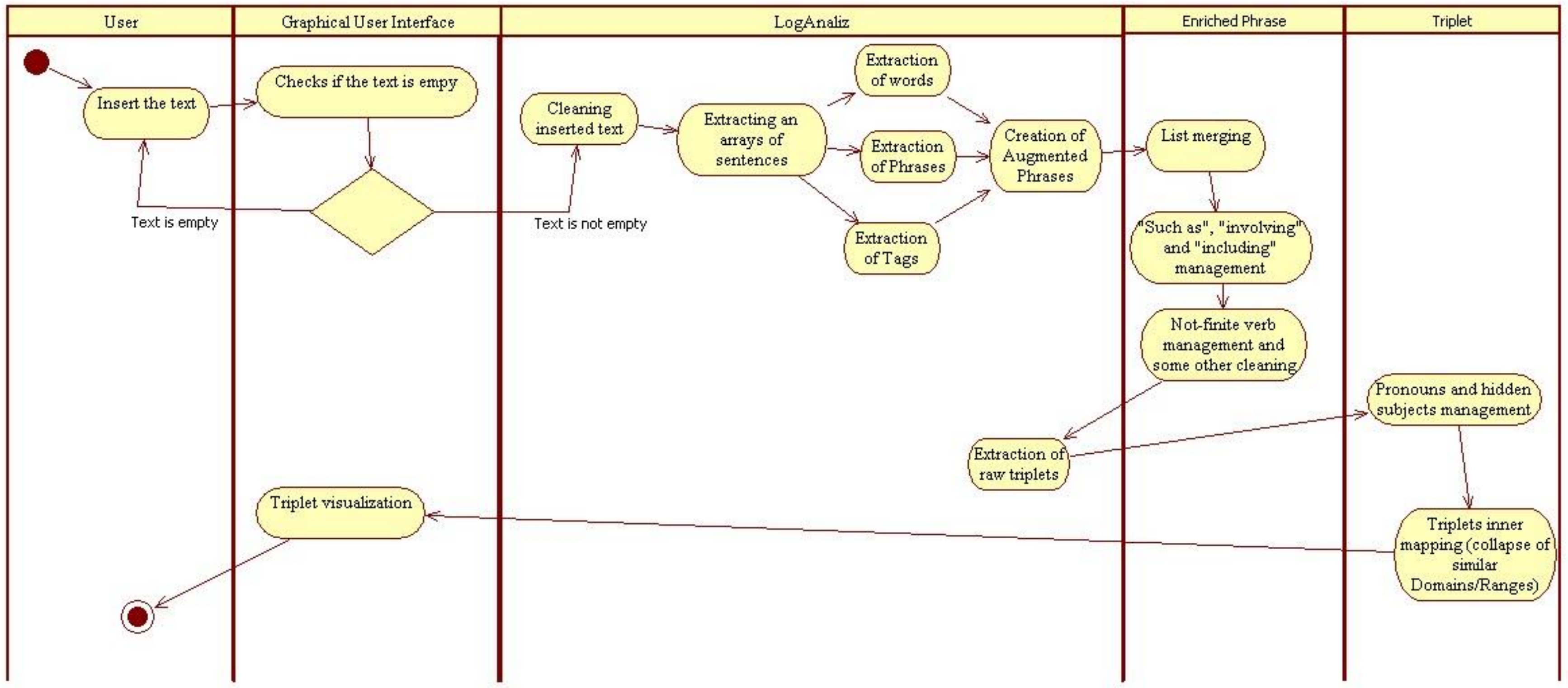

User
Graphical User Interface
LogAnaliz
Enriched Phrase
Triplet
Insert the text
Checks if the text is empy
Text is empty
Text is not empty
Cleaning inserted text
Extracting an arrays of sentences
Extraction of words
Extraction of Phrases
Extraction of Tags
Creation of Augmented Phrases
List merging
"Such as", "involving" and "including" management
Not-finite verb management and some other cleaning
Extraction of raw triplets
Pronouns and hidden subjects management
Triplets inner mapping (collapse of similar Domains/Ranges)
Triplet visualization

Fig. 16. Activity diagram of the system.

4.1.3 Performances

Table 6 summarizes the results of the DeTET with texts from Table 4 (par. 3.5).

Table 6. Performance of Deterministic Triplet Extraction Tool with texts from Table 4.

#	Core concept	Number of sentences	Correct triplets	Extracted triplets	Elapsed time (s)	Precision: Correct triplets / Extracted triplets	Number of triplets for sentence
1	Trauma	30	62	69	2,92	0,8985	2,3
2	Radiography	11	11	12	2,78	0,9166	1,0909
3	Pneumonia	12	35	38	3,14	0,9210	3,1666
4	The human body	10	18	20	2,87	0,9	2
5	Thrombotic Event	8	11	14	2,81	0,7857	1,75
6	Test List	4	18	18	2,88	1	4,5
7	The kidneys	17	49	51	3,36	0,9607	3
8	Pneumonia	11	24	27	2,93	0,8888	2,4545
9	Deep Brain Stimulation	23	37	41	3,41	0,9024	1,7826
10	Kidney abstract	9	20	23	2,99	0,8695	2,5555
11	Kidney abstract	9	22	24	3	0,9166	2,6666
12	Kidney abstract	12	14	14	2,93	1	1,1666
13	Kidney abstract	7	9	12	2,82	0,75	1,7142
14	Kidney abstract	8	13	20	2,78	0,65	2,5
15	Deep Brain Stimulation	8	16	16	2,99	1	2
16	Deep Brain Stimulation	12	19	22	3,15	0,8636	1,8333
17	Deep Brain Stimulation	6	10	11	2,8	0,9091	1,8333
18	Deep Brain Stimulation	8	13	17	2,94	0,7647	2,125
19	Deep Brain Stimulation	14	27	29	3,13	0,931	2,0714

There are three tests with 100% precision and three test with less than 80%.

In Table 7 a summary of DeTET global performance is shown.

Table 7. Summary of Deterministic Triplet Extraction Tool performances.

Metric	**Value**
Mean extracted triplets for sentence	2,237
Mean precision	0,885
Total extracted triplets	478
Total input sentences	219
Total correct triplets	428
Total mean (Total correct triplets / Total extracted triplets)	0,895
Total computation time	56,631 s

DeTET scores a mean precision of 0,885 on 478 extracted triplets from 219 sentences.

We have also evaluated DeTET speed, using texts composed by random sentences from precision test texts. Results are shown in Table 8.

Table 8. Results for tool speed tests.

# Sentences	**# Triplets extracted**	**Time elapsed (s)**
161	308	3,425
401	770	4,718
710	1.340	5,717
4686	8.844	23,957
9372	17.688	44,619

4.2 Graphical User Interface implementation

We have developed a GUI to assist the user during DeTET execution.

In Figure 17 we have described the sequence diagram from the user point of view: the user inserts the text in the GUI, which immediately checks if the inserted text is not a plain text and confirms the operation through a message box. Then the user selects the "extract" button from the "Triplets" menu. The GUI sends a message to LogAnaliz to extract triplets. LogAnaliz calls the method "PHanalizzatore" to extract the triplets and it sends them back to GUI.

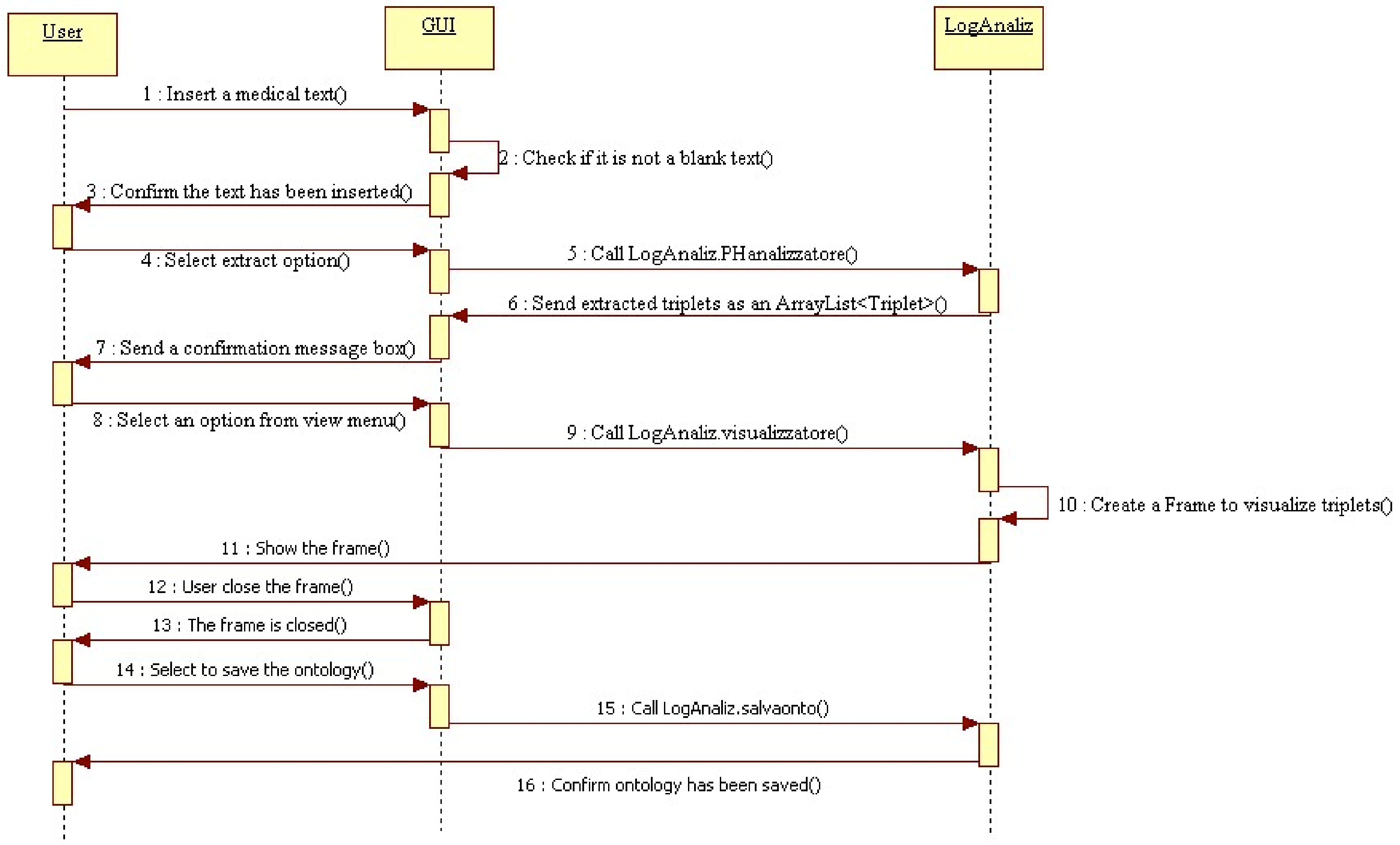

User
GUI
LogAnaliz
1 : Insert a medical text()
2 : Check if it is not a blank text()
3 : Confirm the text has been inserted()
4 : Select extract option()
5 : Call LogAnaliz.PHanalizzatore()
6 : Send extracted triplets as an ArrayList<Triplet>()
7 : Send a confirmation message box()
8 : Select an option from view menu()
9 : Call LogAnaliz.visualizzatore()
10 : Create a Frame to visualize triplets()
11 : Show the frame()
12 : User close the frame()
13 : The frame is closed()
14 : Select to save the ontology()
15 : Call LogAnaliz.salvaonto()
16 : Confirm ontology has been saved()

Fig. 17. Sequence diagram from the user point of view. "User" label means the user is using the application software, "GUI" is the Graphical User Interface class and LogAnaliz is the class which processes the text.

The GUI confirms the occurred extraction with a message box. Then the user can display the output: he/she selects one option from view menu. The GUI sends this message to LogAnaliz, which calls the correct version of the “visualizzatore” method which creates and shows the graph to the user. The user can also save the ontology. he/she sends this request to the GUI which sends the message to LogAnaliz. LogAnaliz calls “salvaonto” method and sends a confirm to the user.

In following Figures we show a run of a generic user to extract triplets from the text: *“The kidneys are essential in the urinary system and also serve homeostatic functions such as the regulation of electrolytes, maintenance of acid–base balance, and regulation of blood pressure (via maintaining salt and water balance). They serve the body as a natural filter of the blood, and remove wastes, which are diverted to the urinary bladder.”*. The whole process has been described in fig. 16: we will show screenshot of meaningful steps of fig. 16.

In fig. 18 the whole process is resumed:

5. The user selects “File” menu and “Insert text” option to insert the text inside the system.
6. The user selects “Triplets” menu and “Triplet extraction from text” option to extract triplets from the previously inserted text.
7. User selects one view option from the “View” menu to display extracted triplets
8. User selects “Save” option from “File” menu to save the ontology. The user has two choices, “Save as RDF/XML file” and “Save as Turtle file”: this way he/she can choose the format of the output ontology.

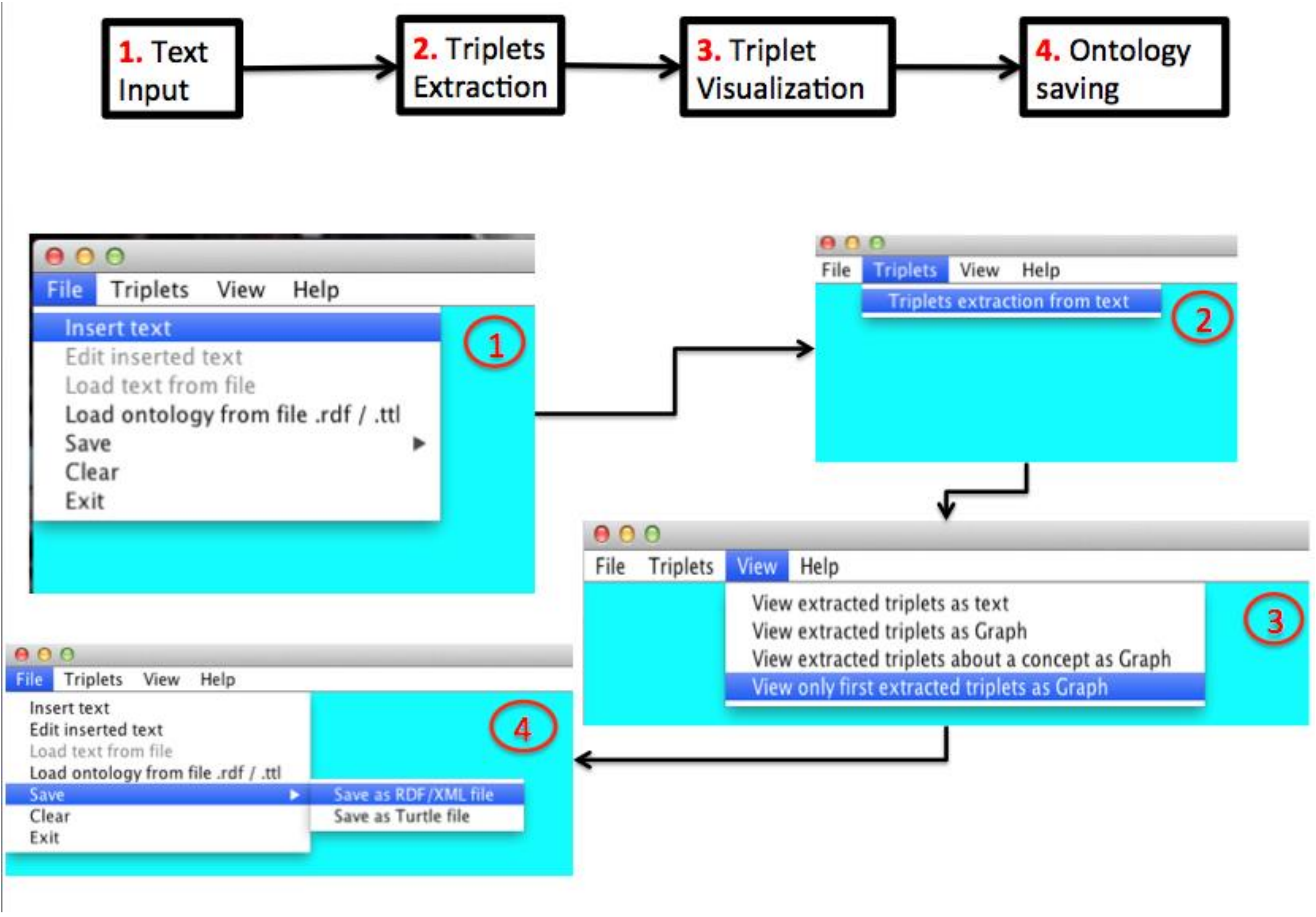
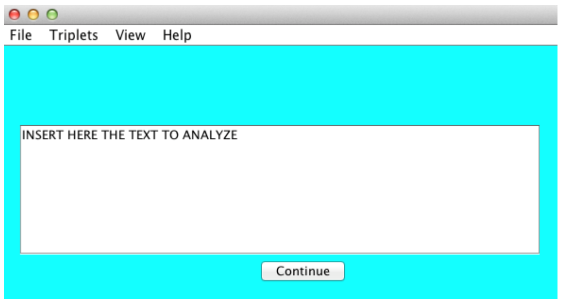

1. Text Input
2. Triplets Extraction
3. Triplet Visualization
4. Ontology saving
File
Triplets
View
Help
Insert text
Edit inserted text
Load text from file
Load ontology from file .rdf / .ttl
Save
Clear
Exit
1
Triplets extraction from text
2
View extracted triplets as text
View extracted triplets as Graph
View extracted triplets about a concept as Graph
View only first extracted triplets as Graph
3
Save as RDF/XML file
Save as Turtle file
4

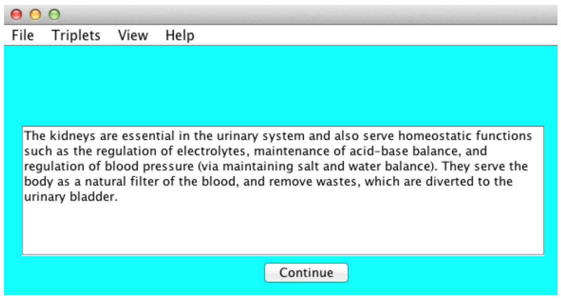
Fig. 18. The workflow to extract triplets from a text using the Graphical User Interface.

After the user has selected the “Insert text” option (Figure 18 step 1), the GUI allows the user to insert the text (Figure 19 and Figure 20). The GUI displays a confirmation massage box after a successful text input (Figure 20).



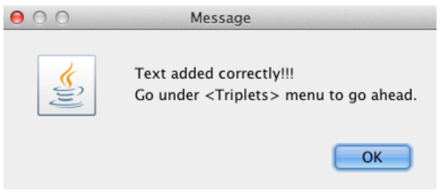
INSERT HERE THE TEXT TO ANALYZE

Fig. 19. The Graphical User interface allows the user to write the text after he/she has selected the “Insert text” option in “File” menu.



The kidneys are essential in the urinary system and also serve homeostatic functions such as the regulation of electrolytes, maintenance of acid-base balance, and regulation of blood pressure (via maintaining salt and water balance). They serve the body as a natural filter of the blood, and remove wastes, which are diverted to the urinary bladder.

Fig. 20. The user has insert the text "*The kidneys are essential in the urinary system and also serve homeostatic functions such as the regulation of electrolytes, maintenance of acid–base balance, and regulation of blood pressure (via maintaining salt and water balance). They serve the body as a natural filter of the blood, and remove wastes, which are diverted to the urinary bladder.*" in our simulation.



Text added correctly!!!
Go under <Triplets> menu to go ahead.

Fig. 21. The Graphical User Interface sends a message box to confirm the user he/she has successfully inserted the text.

After the text has been successfully inserted, the user can select “Triplet extraction from text” option from “Triplets” menu (Figure 18 step 2) as advised in Figure 20: the system tries to prevent users from making mistakes and tries to assist the user. Thus it does not allow the users to select an option before it can execute the relative task. For this reason, until the user has not inserted correctly a text, the "Triplet extraction from text" option is not selectable (Figure 22).

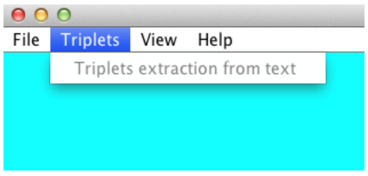
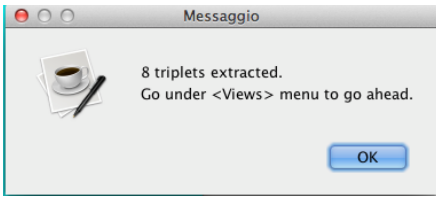


Fig. 22. The user can not select "Triplet extraction from text" option until he/she has inserted a text.

The system shows a message to confirm the occurred extraction and tells the user how many triplet it has extracted (Figure 23). Until the extraction has not occurred, no view option is available from "View" menu (Figure 24).


Messaggio

8 triplets extracted.
Go under <Views> menu to go ahead.

Fig. 23. The system shows a message box to confirm the occurred extraction.

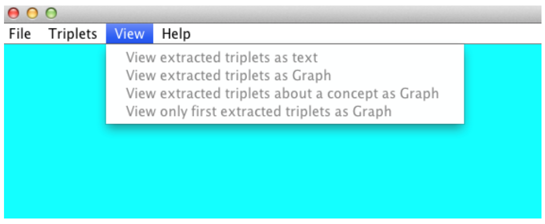
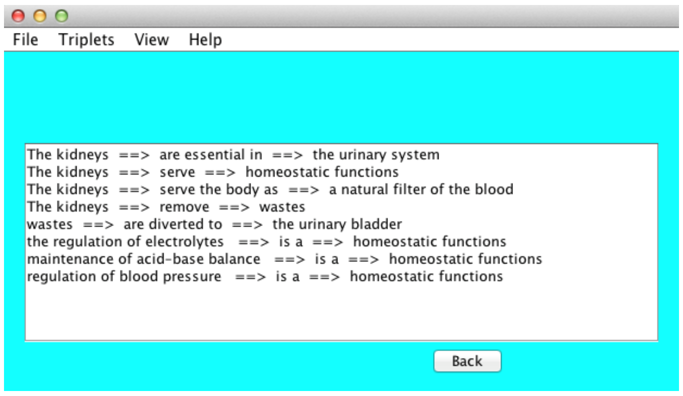

File
Triplets
View
Help
View extracted triplets as text
View extracted triplets as Graph
View extracted triplets about a concept as Graph
View only first extracted triplets as Graph

Fig. 24. The system does not allow the user to visualize an ontology until any triplet has been extracted.

After the occurred extraction (Figure 23), the user can select one option from "View" menu (Figure 18 step 3). Figure 25 shows the output when the "View extracted triplets as text" option is selected (Figure 24) while Figure 26 shows the output when the "View extracted triplets as Graph" option is selected (Figure 24).



```
The kidneys  ==>  are essential in  ==>  the urinary system
The kidneys  ==>  serve  ==>  homeostatic functions
The kidneys  ==>  serve the body as  ==>  a natural filter of the blood
The kidneys  ==>  remove  ==>  wastes
wastes  ==>  are diverted to  ==>  the urinary bladder
the regulation of electrolytes   ==>  is a  ==>  homeostatic functions
maintenance of acid-base balance   ==>  is a  ==>  homeostatic functions
regulation of blood pressure   ==>  is a  ==>  homeostatic functions
```

Fig. 25. Output when the "View extracted triplets as text" option is selected (fig. 23).

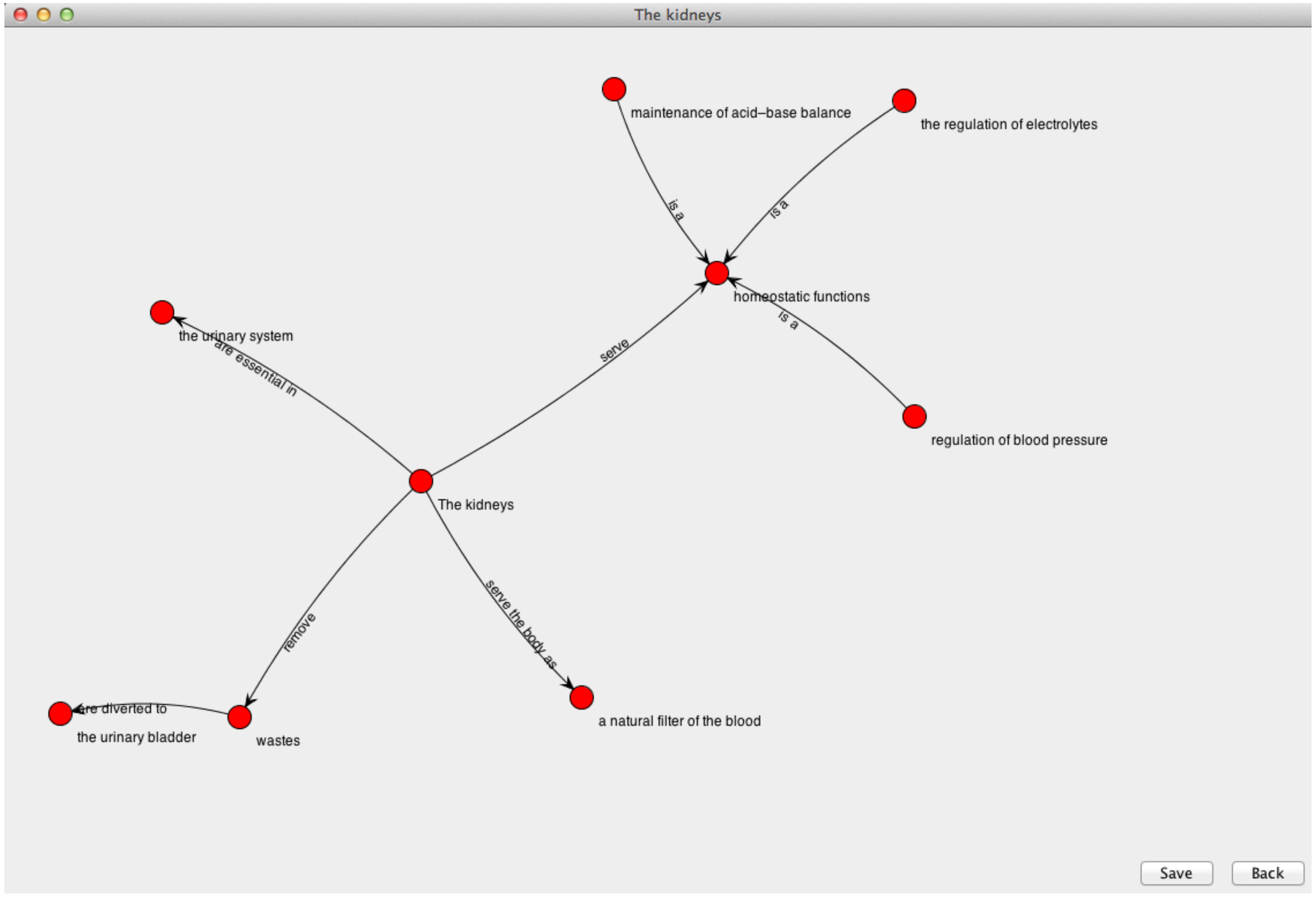
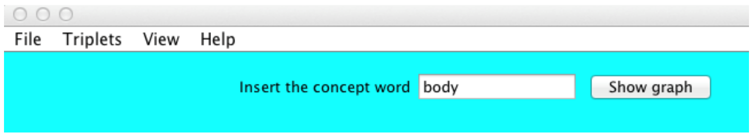


Fig. 26. Graph output from the text "*The kidneys are essential in the urinary system and also serve homeostatic functions such as the regulation of electrolytes, maintenance of acid–base balance, and regulation of blood pressure (via maintaining salt and water balance). They serve the body as a natural filter of the blood, and remove wastes, which are diverted to the urinary bladder.*".

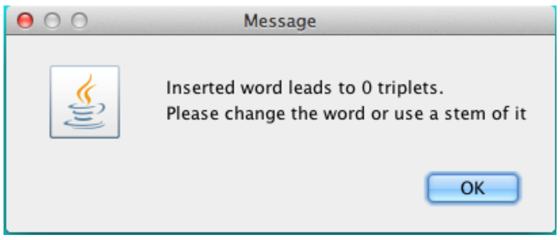
Figure 27 shows how the starting screen changes when the "View extracted triplets about a concept as a Graph" option is selected. The GUI allows the user to insert the concept name (Figure 27) and immediately checks if the inserted name leads to an empty graph (Figure 28).

Figure 29 shows how the starting screen changes when the "View only first extracted triplets as a Graph" option is selected. The GUI allows the user to insert the number of triplets it has to show, with a default value of 10 (Figure 29).



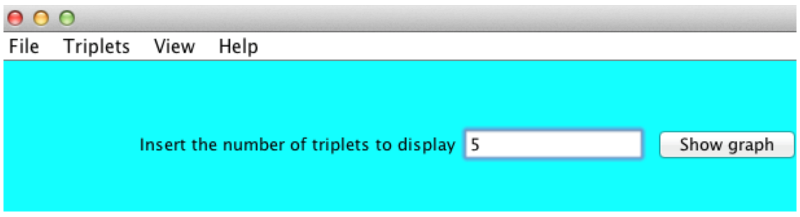
Insert the concept word: body

Fig. 27. How the GUI allows the user to select the concept for the "View extracted triplets about a concept as a Graph" option from "View" menu.



Inserted word leads to 0 triplets.
Please change the word or use a stem of it

Fig. 28. The system immediately informs the user if the inserted concept name (fig. 26) leads to an empty graph.



Insert the number of triplets to display: 5

Fig. 29. How the GUI allows the user to select the concept for the "View extracted triplets about a concept as a Graph" option from "View" menu.

In Figure 30 the graph generated using "waste" concept in Figure 27 is shown, while in Figure 31 the graph generated from the first five triplets (Figure 29) is shown.

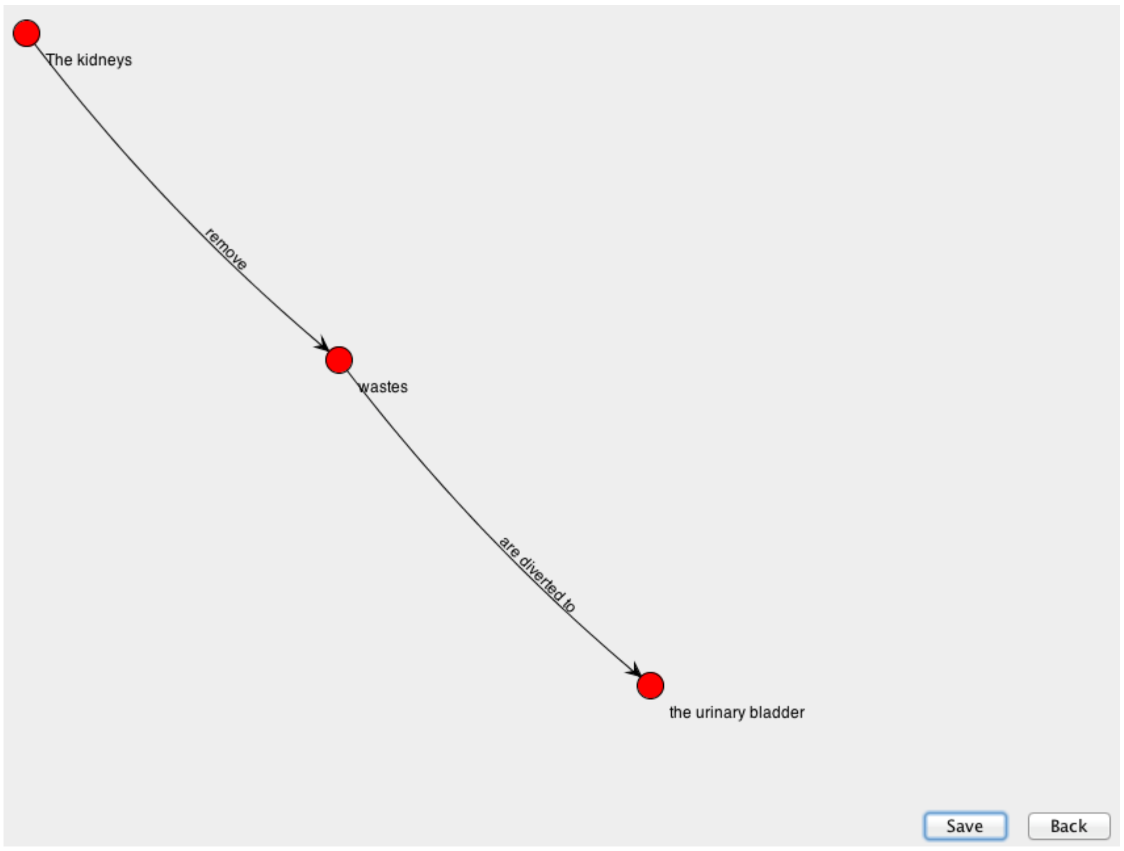


Fig. 30. Graph linked to the concept "waste" (Figure 27) generated from the text "*The kidneys are essential in the urinary system and also serve homeostatic functions such as the regulation of electrolytes, maintenance of acid–base balance, and regulation of blood pressure (via maintaining salt and water balance). They serve the body as a natural filter of the blood, and remove wastes, which are diverted to the urinary bladder.*

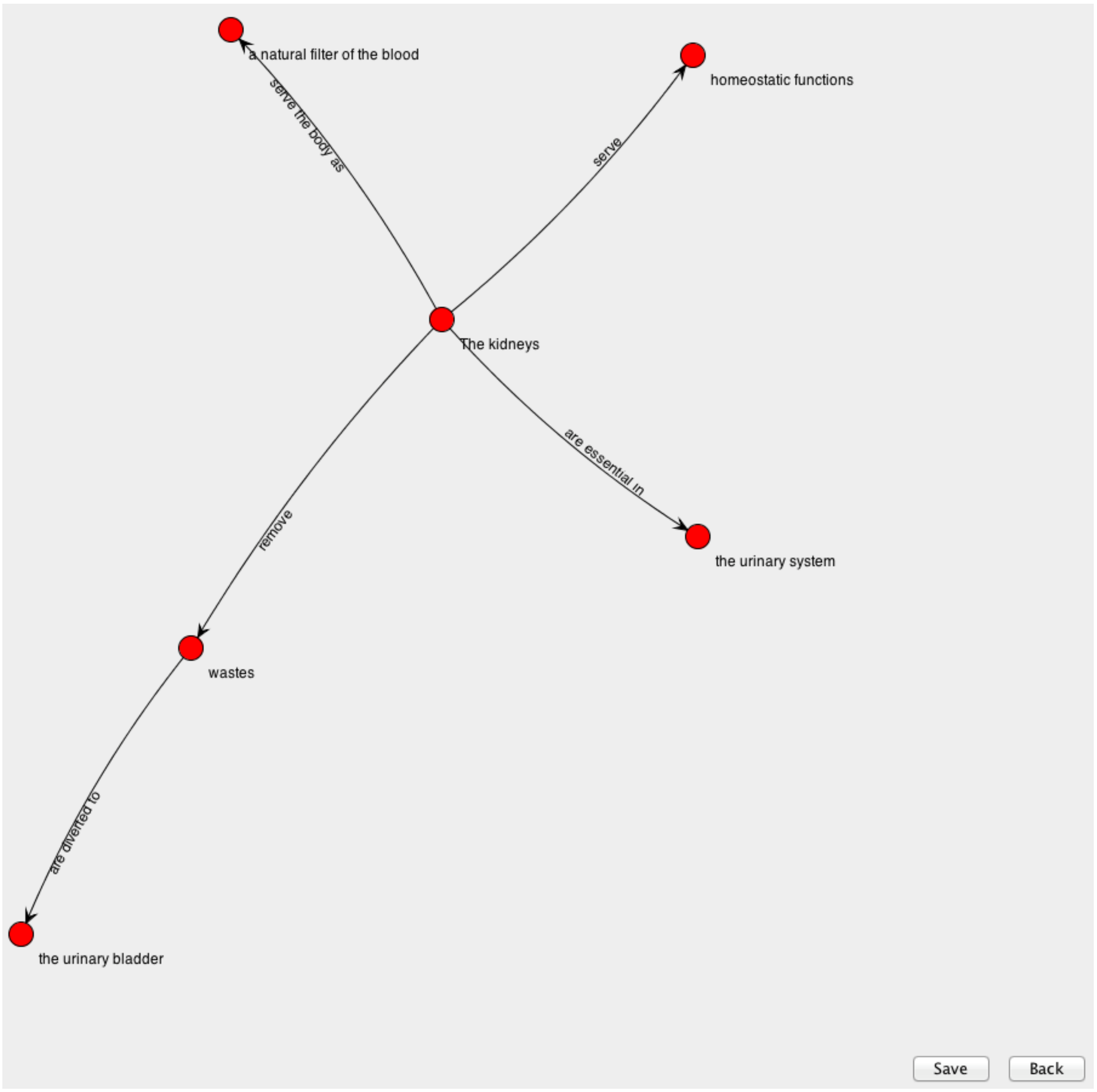
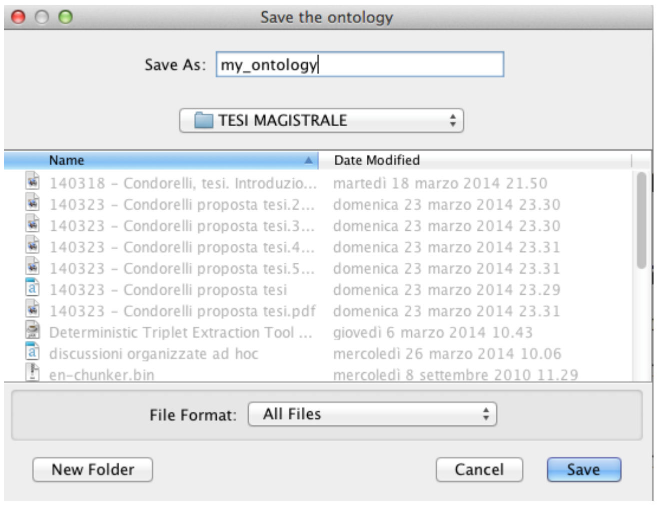


Fig. 31. First five triplets (Figure 29) extracted from the text "*The kidneys are essential in the urinary system and also serve homeostatic functions such as the regulation of electrolytes, maintenance of acid–base balance, and regulation of blood pressure (via maintaining salt and water balance). They serve the body as a natural filter of the blood, and remove wastes, which are diverted to the urinary bladder.*".

Once the user has viewed the output ontology (step 3 of Figure 18), he/she can save it (step 4 of Figure 18). The GUI supports the user in the saving phase through a standard “dialog box” (Figure 32). As we have already explained, the user can choose one from two different format (“RDF/XML” and “Turtle” Figure 18 step 4) to increase created ontologies reusability.


Save the ontology

Save As: my_ontology

TESI MAGISTRALE

Name	Date Modified
140318 – Condorelli, tesi. Introduzio…	martedì 18 marzo 2014 21.50
140323 – Condorelli proposta tesi.2…	domenica 23 marzo 2014 23.30
140323 – Condorelli proposta tesi.3…	domenica 23 marzo 2014 23.30
140323 – Condorelli proposta tesi.4…	domenica 23 marzo 2014 23.31
140323 – Condorelli proposta tesi.5…	domenica 23 marzo 2014 23.31
140323 – Condorelli proposta tesi	domenica 23 marzo 2014 23.29
140323 – Condorelli proposta tesi.pdf	domenica 23 marzo 2014 23.31
Deterministic Triplet Extraction Tool …	giovedì 6 marzo 2014 10.43
discussioni organizzate ad hoc	mercoledì 26 marzo 2014 10.06
en-chunker.bin	mercoledì 8 settembre 2010 11.29

File Format: All Files

Fig. 32. The "save" phase opens a dialog box

At the end, the user can exit the program (Figure 33) or just start again using “Clear” option from “File” menu (Figure 34). The “Clear” option reset the software application, erasing every store text and every created triplets.

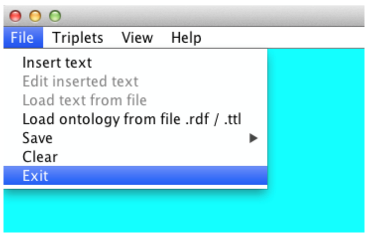


Fig. 33. Software screenshot, exiting the software.

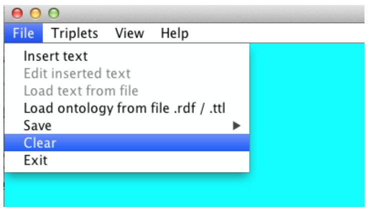
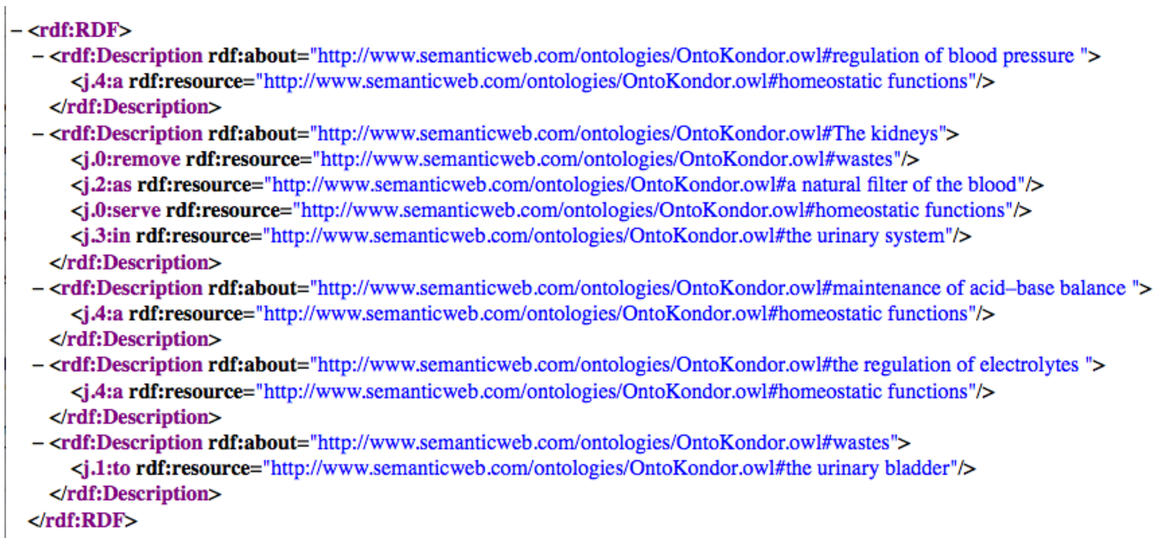


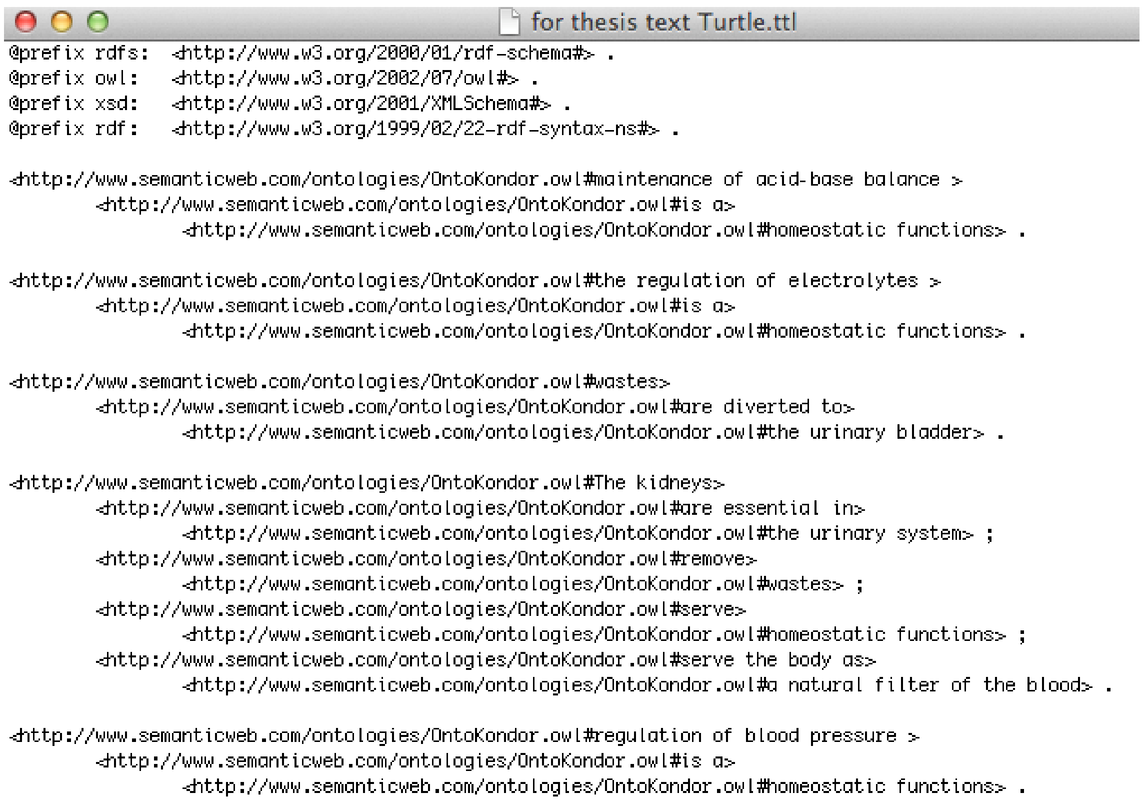
Fig. 34. The user can select "Clear" option from "File" menu to reset the software application.

In Figure 35 the “RDF/XML” ontology (saved as it has been shown in Figure 32 and reopened through a web browser) is shown, while in Figure 36 is shown the “Turtle” version of the same file, opened with the Mac Text Edit.



```
– <rdf:RDF>
  – <rdf:Description rdf:about="http://www.semanticweb.com/ontologies/OntoKondor.owl#regulation of blood pressure ">
      <j.4:a rdf:resource="http://www.semanticweb.com/ontologies/OntoKondor.owl#homeostatic functions"/>
    </rdf:Description>
  – <rdf:Description rdf:about="http://www.semanticweb.com/ontologies/OntoKondor.owl#The kidneys">
      <j.0:remove rdf:resource="http://www.semanticweb.com/ontologies/OntoKondor.owl#wastes"/>
      <j.2:as rdf:resource="http://www.semanticweb.com/ontologies/OntoKondor.owl#a natural filter of the blood"/>
      <j.0:serve rdf:resource="http://www.semanticweb.com/ontologies/OntoKondor.owl#homeostatic functions"/>
      <j.3:in rdf:resource="http://www.semanticweb.com/ontologies/OntoKondor.owl#the urinary system"/>
    </rdf:Description>
  – <rdf:Description rdf:about="http://www.semanticweb.com/ontologies/OntoKondor.owl#maintenance of acid–base balance ">
      <j.4:a rdf:resource="http://www.semanticweb.com/ontologies/OntoKondor.owl#homeostatic functions"/>
    </rdf:Description>
  – <rdf:Description rdf:about="http://www.semanticweb.com/ontologies/OntoKondor.owl#the regulation of electrolytes ">
      <j.4:a rdf:resource="http://www.semanticweb.com/ontologies/OntoKondor.owl#homeostatic functions"/>
    </rdf:Description>
  – <rdf:Description rdf:about="http://www.semanticweb.com/ontologies/OntoKondor.owl#wastes">
      <j.1:to rdf:resource="http://www.semanticweb.com/ontologies/OntoKondor.owl#the urinary bladder"/>
    </rdf:Description>
  </rdf:RDF>
```

Fig. 35. Saved ontology in “RDF/XML” format.



for thesis text Turtle.ttl

```
@prefix rdfs:   <http://www.w3.org/2000/01/rdf-schema#> .
@prefix owl:    <http://www.w3.org/2002/07/owl#> .
@prefix xsd:    <http://www.w3.org/2001/XMLSchema#> .
@prefix rdf:    <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .

<http://www.semanticweb.com/ontologies/OntoKondor.owl#maintenance of acid-base balance >
        <http://www.semanticweb.com/ontologies/OntoKondor.owl#is a>
                <http://www.semanticweb.com/ontologies/OntoKondor.owl#homeostatic functions> .

<http://www.semanticweb.com/ontologies/OntoKondor.owl#the regulation of electrolytes >
        <http://www.semanticweb.com/ontologies/OntoKondor.owl#is a>
                <http://www.semanticweb.com/ontologies/OntoKondor.owl#homeostatic functions> .

<http://www.semanticweb.com/ontologies/OntoKondor.owl#wastes>
        <http://www.semanticweb.com/ontologies/OntoKondor.owl#are diverted to>
                <http://www.semanticweb.com/ontologies/OntoKondor.owl#the urinary bladder> .

<http://www.semanticweb.com/ontologies/OntoKondor.owl#The kidneys>
        <http://www.semanticweb.com/ontologies/OntoKondor.owl#are essential in>
                <http://www.semanticweb.com/ontologies/OntoKondor.owl#the urinary system> ;
        <http://www.semanticweb.com/ontologies/OntoKondor.owl#remove>
                <http://www.semanticweb.com/ontologies/OntoKondor.owl#wastes> ;
        <http://www.semanticweb.com/ontologies/OntoKondor.owl#serve>
                <http://www.semanticweb.com/ontologies/OntoKondor.owl#homeostatic functions> ;
        <http://www.semanticweb.com/ontologies/OntoKondor.owl#serve the body as>
                <http://www.semanticweb.com/ontologies/OntoKondor.owl#a natural filter of the blood> .

<http://www.semanticweb.com/ontologies/OntoKondor.owl#regulation of blood pressure >
        <http://www.semanticweb.com/ontologies/OntoKondor.owl#is a>
                <http://www.semanticweb.com/ontologies/OntoKondor.owl#homeostatic functions> .
```

Fig. 36. Saved ontology in Turtle format (.ttl).

5. Discussion

In this master thesis, a Deterministic Triplet Extraction Tool (DeTET) to automatically extract triplets from a natural language text and to automatically save these triplets as an ontology has been designed, developed and implemented. A Graphic User Interface has been designed, too.

Our approach is totally deterministic: the idea behind DeTET is that Part Of Speech tags, Phrases and Tokens create unique structures. Pieces of information on one single NLP level is not enough: for example "Noun Phrase Noun Phrase" could refer to various situation such as "the kidney, it", "organs such as kidney", or "kidney when the situation"; listed examples clearly have to be processed differently: in the first example "kidney" is probably an object while "it" is the subject of another relation; in the second "kidney" is a kind of "organ"; in the third "kidney" is probably an object while the part starting with "when" has to be deleted as explained in paragraph 3.1.3. Thus, using only Phrase level is not enough: commas and conjunction are non displayed at Phrase level because they do not form any kind of Phrase and are not embeddable in any existing Phrase; although we lose some pieces of information about Verb type (finite or not-finite) and about the role of the Verb Phrase within the sentence (as we have already explained in the 3.1.1, for example a Verb Phrase could be the subject of a clause). Using every level of information is enough to create very specific structure fingerprints which allows DeTET to extract triplets. A similar work has been proposed by Cimiano et al. [79]: they tried to extract triplets from text using a statistical approach. The main difference with our system is they only consider a small list of extractable relations (e.g. "is a") to create the ontology while we do not have this restriction.

The results in extracting triplets from testing texts (Table 6) are encouraging: the system scores a 100% precision with three different texts, and over 85% precision with almost every other texts. Only with three texts DeTET scores a precision lower than 80%. A triplet has been considered correct if:

- the Relation is a finite tense verb
- the Domain is the its subject
- the Range is the its object

Every wrong element switching has been considered an error (e.g. a wrong pronoun to noun switch, or two different concepts considered as the same one through our similarity score approach). Every couple of similar concept which are not recognized as the same has not been considered as an error (this kind of issue does not generate wrong pieces of information).

DeTET seems to be a very fast tool: the processing time was calculated on the command line version of DeTET, not on the GUI-version. It is quicker than similar systems: for example Rusu et al. [80], performed an extraction of 168 triplets from 100 sentences in 29,5 seconds. Authors did not specify the sentences, thus a run with same inputs in our system is not possible. However, in almost the same time (23,95 seconds) DeTET extracts 8.844 triplets from 4686 sentences (Table 8). The test on random (and then, maybe repeated) sentences is meaningful because DeTET does not have memory: the time to process one sentence is exactly half time to process that sentence two times.

Sources for texts from Table 4 has been chosen accurately to simulate different scenarios. First possible scenario is a non-technical user who would like to understand better some topic linked to biomedical subject matter: he/she will probably use some common source of information, like PubMed [32] or if he/she is not a professional some source like Wikipedia [67]. Thus we have selected some texts from PubMed and from Wikipedia about different kinds of biomedical topics, such as “Kidneys” or “Pneumonia”. In this scenario, the user would use the developed software application just to the graph visualization capability. Another possible scenario is the general practitioner who would like to increase his/her knowledge about a specialist topic, such as “Deep Brain Stimulation”: we have selected five abstracts from PubMed papers about “Deep Brain Stimulation”. One last possible scenario is a technical user who would like to build an ontology for some purpose: ontologies are often build using PubMed papers as the main source of knowledge. Thus we have selected some papers about “Kidney” from PubMed.

We figured two kind of possible users: a consumer user and a medical user. We have designed the GUI to lead the user in every phase of the process to create an ontology. To avoid any kind of user mistakes, the GUI dynamically enables and disables menu options and buttons: step by step, only usable options are enabled. The GUI allows the user to create an ontology both to store it or just to visualize it: it has been done this way because some users could use DeTET only for representing knowledge, while others just need to build ontologies to support other software applications. To increase DeTET ontologies reusability, we provided the user with the “save” function. Two common ontology formats are available: “RDF/XML” and “Turtle”.

DeTET has some **weakness**. First, if the sentence contains any grammatical or linguistic error, DeTET will make mistakes: this kind of issue is unavoidable. If the text contains any kind of conceptual mistake, DeTET will not notice it and will make mistakes. We want to point out that the logical capability (e.g. the skill to implies conclusions from statements) of an ontology could lead to discover a posteriori this kind of mistakes as we have explained in par. 2.1.4 . For example the triplets:

- Bone + contains + Cells
- Femur + is_A + Bone
- Femur + not_Contains + Cells

will lead to an inconsistency. Thus a reasoner (the tool to carry out logical deduction from an ontology as explained in par. 2.1.3), could tell the user that the ontology contains some errors: the reasoner can not fix the error itself because it does not know which statement is true outside ontology universe and which is false (in the previous example, it discovers a contradiction but it does not know if not every bone contains cell, if a femur is not a bone or if a femur contains cells). A reasoner can not deal with other kind of mistakes (e.g. text “Ice sets on fire trees” will create the triplet “Ice + sets on fire + trees” which is grammatically correct and correctly extracted, but contains meaningless information).

DeTET has some problem in processing complex structure sentences and phrases such as the embedding phrases, which means some phrases that contain other phrases (e.g. the NP “the kidney damaged by alcohol” contains also a VP, “damaged”). This kind of issue is very hard to manage, because NLP tools often make mistakes in processing embedding phrase: for example, often an embedded verb phrase is considered as a standalone phrase. We tried to solve this problem during post processing phases of Enriched Phrases: this way DeTET tags Verb Phrase correctly (it is more precise than DeTET itself without this kind of error handling). Then our Triplet Extraction Tools relies on other statistical NLP tools issue [13]: when the openNLP confuses one POS with another, DeTET is not able to fix it. Sometimes an error spreads from one sentence to another (e.g. when there is a mistake on subject identification on a sentence and the following one starts with “they”): to minimize error spreading from a sentence to another DeTET analyses sentence one-by-one, but this solution does not completely avoid this kind of mistake. To increase the effectiveness of this solution, we implemented some checks during Enriched Phrases generation (Figure 12), such as commas and conjunctions ones (DeTET checks if there is a comma or some conjunction inside the Phrases). Some verbs imply clauses with very peculiar structure: it is very hard to map every exception also because some exceptions are linked to the verb meaning in that specific phrase; for example, the verb “to put” can both has an Subject Verb Object Adverbial structure when it carries the meaning of placing something somewhere (e.g. “The clinician put the needle in the body”: “in the body” is a compulsory adverbial phrase, without it the sentence is grammatically incomplete) or just Subject Verb Object structure when it means lead something by water (e.g. “Terry put the boat in danger”: ”in danger” is a complement of “the boat” phrase and it is not linked with “Terry”). DeTET preprocesses **if-clauses** excluding the if part and processing the main one: this is because description logic can not consider dynamic statements (a statement which is true if and only if another one is true). To preserve a piece of the original information, the verb of the main clause is arranged as a conditional tense. In description logic is possible to create relations chain such as $R_a \circ R_b \dot{\sqsubseteq} R_c$ (with R_a, R_b *and* R_c relations and “∘” relations operator which means “chained with”) which means if A R_a B and B R_b C so A R_c C (e.g. if "$father\_Of \circ brother\_Of \dot{\sqsubseteq} father\_Of$ “and “A father_Of B” and “B brother_Of C”, so we can imply “A father_Of C”: this triplet could be generated by the sentence ”If Tom is brother of Tim, they have same father”); this kind of solution can catch some kind of dynamic, but needs a higher abstraction than DeTET rules can make and can also lead to a lot of mistakes, decreasing the precision. DeTET pre-processes also **when-clauses** in the same manner: something which happens when something else has just happened means that something could happen; thus the when-part is not considered and the tense of the main-part gains a conditional tense.

We are aware that we do not consider the accuracy: it is defined as the number of extracted items divides the total number of extractable items; we have not considered it on purpose because it is hard (and arbitrary) to establish the total number of extractable triplets from a text. We could have set the number of extractable triplets from a text as the number of triplets a human could extract from the text, but we would have succumbed to an issue: a human-triplet-extraction is a very abstract process which is deeply related to the understanding of the text by the extractor. Thus, every different human extractor could find different triplets from the same text. Instead of accuracy, we have shown the number of extracted triplets for sentence (t/s) as a metrics to underline how the system has worked with a input text: DeTET scores an average of 2,3 t/s.

We have tested DeTET with short texts on purpose (at least 30 sentences, see Table 4): we have done this way for some reasons. The main reason is the system does not have any kind of memory, as we have already seen in Methods: thus, in Symbols

Suppose to have the texts "A" and "B"

Suppose to have the text "C" : C = (A,B) where "," means append text B to text A

if we call t_X the triplets extracted from the text "X" and t_X = DeTET(X)

then

t_A = DeTET(A), t_B = DeTET(B), t_C = DeTET(C)

Because of the system does not have memory,

DeTET(C) = t_A+ t_B

where

"," means append text B to text A

"+" means add triplets t_A to triplets t_B.

Because of "automappa" method, which look for similar concepts to collapse on the same labels, if the text "A" and "B" are about the same topic:

$$t_C \cong t_B + t_A$$

t_C will have same number of triplets as the sum of the number of the triplets of t_A and t_B. Thanks to our implementation of similarity metric, which is really conservative (it collapse concepts only if they score a very high value, does the number of wrong collapsed concepts is very close to zero), we can state precision on text C is the weighted sum of precision on texts A and B, using as weight the number of extracted triplet from A and from B. Thus using longer texts is almost the same as using a lot of shorter one: we have preferred to use shorter different texts instead of lesser longer texts because performances of NLP tools and ontologies algorithm often depends on the domain topic.

DeTET tuning followed three approaches: improving number of correct triplets, increasing the number of extracted triplets and increasing the triplet meaning (e.g. in the sentence "The removal of the kidney could be necessary" a correct Domain is "The removal", a more meaningful Domain is "The removal of the kidney"). First, to improve number of correct triplets, we implemented some frequent linguistic patterns to extract right elements of triplets. For example, the verb “to be” has very specific extraction rules because it is a copular verb with specific grammatical construction. Thus DeTET number of correct triplets is very high on simple sentence, very close to 100%. DeTET scores lower performance with peculiar structure sentences. To increase the triplet meaning, we managed some typical situation where the triplet is correct but the meaning of the original sentence is not carried out. For example, every pronoun is automatically substituted with corresponding noun. To extract the most enriched element, we implemented Enriched Phrase structure. For example, the sentence “Cancer can be managed with removal of the kidney” generates the triplet {cancer, can be managed with, removal of the kidney} and not {cancer, can be managed, removal}, which is correct but meaningless. Finally, to increase the number of extracted triplets we implemented some improvements of DeTET beyond subject/verb/object identification: for example, every time DeTET finds “<something> such as <something else>” it creates the triplet “<something else> is_A <something>”. The addition of this kind of new triplets greatly increase the knowledge gathered from text to ontology; furthermore, if an element of the triplet is composed by a list of noun or verb, DeTET automatically splits this triplet in some triplets.

We have deeply studied wrong extracted triplets to understand how to improve performances. Main sources of mistakes are uncommon use of grammatical construction but still correct. For example, “which” refers almost always to the closer Noun/Pronoun: it is still correct if it refers to another Noun/Pronoun in the same sentence. The human reader can understand what “which” refers to from the general meaning of the sentence, DeTET has not those human ability. Thus our approach to DeTET implementation has been to study side-effects in any possible new grammatical pattern or rule before deciding if to apply it or not. Thus every rule has been implemented if it allows a correct extraction every time it is applied, or if it manages correctly more common situations and makes mistakes in rare ones. If a rule is sometimes correct and sometimes not, we tried to avoid the rule itself: for example, implementation of a “when pattern” to extract both triplets from a sentence including “when” would have lead to some correct triplets and some incorrect ones (maybe half and half). We have avoided this situation discarding a priori every part of sentence containing “when” token (see par. 3.1.3).

There are some open issues we have not completely deal with. For example, String Similarity issue. We have proposed an easy approach to the problem because a deeper approach lead to another project. Also we have avoided statement with same Domain and Range (e.g. “Cells generate theirself”): this kind of situation could lead not only to more precise results, but also requires to much time to be implemented. Also an improved pronoun management approach could lead to enhanced results.

As part of the future work we plan to enhance graph visualization too, allowing user to interact with graph nodes.

6. Conclusions

We presented an approach to Next Generation of Biomedical Ontologies, and the implementation of a software tool for extracting triplets from biomedical texts, we called Deterministic Triplet Extraction Tool (DeTET). It exploits Natural Language Processing techniques to extract structured pieces of information from natural language digital texts. We have developed new structure to store grammatical information from a sentence we called Enriched Phrase. It has been developed as a standalone Java tool with a clear and user-friendly Graphic User Interface. We performed some investigation tests, obtaining around 90% correct extracted triplets and low computation time (less than a couple of seconds for a page-length text). Thanks to that, Deterministic Triplet Extraction Tool seems to be the essential step to solve ontology automatic generation issue. Open issues to improve its performance on peculiar sentences and to improve graph visualization are the subjects of our future works.

Appendix A

Starting from the input text *"The kidneys are essential in the urinary system and also serve homeostatic functions such as the regulation of electrolytes, maintenance of acid–base balance, and regulation of blood pressure (via maintaining salt and water balance). They serve the body as a natural filter of the blood, and remove wastes, which are diverted to the urinary bladder."* the outputs of the system are represented as it follows.

Inside ArrayList<Tripletta> PHanalizzatore(String Text).

First loop.

Sentence before cleaning:

The kidneys are essential in the urinary system and also serve homeostatic functions such as the regulation of electrolytes, maintenance of acid–base balance, and regulation of blood pressure (via maintaining salt and water balance)

Sentence after cleaning:

The kidneys are essential in the urinary system and also serve homeostatic functions such as the regulation of electrolytes, maintenance of acid–base balance, and regulation of blood pressure

Extracted tags:

(The)DT (kidneys)NNS (are)VBP (essential)JJ (in)IN (the)DT (urinary)JJ (system)NN (and)CC (also)RB (serve)VB (homeostatic)JJ (functions)NNS (such)JJ (as)IN (the)DT (regulation)NN (of)IN (electrolytes)NNS (,), (maintenance)NN (of)IN (acid–base)JJ (balance)NN (,), (and)CC (regulation)NN (of)IN (blood)NN (pressure)NN

Extracted phrases:

(The kidneys)NP

(are)VP

(essential)ADJP

(in)PP

(the urinary system)NP

(also)ADVP

(serve)VP

```
(homeostatic functions)NP
(such as)PP
(the regulation)NP
(of)PP
(electrolytes)NP
(maintenance)NP
(of)PP
(acid–base balance)NP
(regulation)NP
(of)PP
(blood pressure)NP
```

Inside ArrayList<Phrase> span2phr(Span[] span, String[] tokens, String[] tags).

Extracted Enriched Phrases:

```
(The kidneys)NP
(are essential in)1VBF
(the urinary system)NP
(and)AndOr
(serve)VBF
(homeostatic functions)NP
(homeostatic functions)NP
(such as)PP
(the regulation of electrolytes)NP
(),
(maintenance of acid–base balance)NP
(),
(and)AndOr
(regulation of blood pressure)NP
```

Closing ArrayList<Phrase> span2phr(Span[] span, String[] tokens, String[] tags).

Inside ArrayList<Phrase> fondi_liste2(ArrayList<Phrase> p)

(The kidneys)NP

(are essential in)1VBF

(the urinary system)NP

(and)AndOr

(serve)VBF

(homeostatic functions)NP

(homeostatic functions)NP

(such as)PP

(the regulation of electrolytes DIVIDEDBYmaintenance of acid–base balance DIVIDEDBYregulation of blood pressure)NP

Closing ArrayList<Phrase> fondi_liste2(ArrayList<Phrase> p)

Inside ArrayList<Phrase> such_and_ing(ArrayList<Phrase> p).

(The kidneys)NP

(are essential in)1VBF

(the urinary system)NP

(and)AndOr

(serve)VBF

(homeostatic functions)NP

(homeostatic functions)NP

(is a)Such_As

(the regulation of electrolytes DIVIDEDBYmaintenance of acid–base balance DIVIDEDBYregulation of blood pressure)NP

Closing ArrayList<Phrase> such_and_ing(ArrayList<Phrase> p).

Inside ArrayList<Phrase> cleaning_time(ArrayList<Phrase> phrases).

(The kidneys)NP

(are essential in)1VBF

(the urinary system)NP

(and)AndOr

(serve)VBF

(homeostatic functions)NP

(homeostatic functions)NP

(is a)Such_As

(the regulation of electrolytes DIVIDEDBYmaintenance of acid–base balance DIVIDEDBYregulation of blood pressure)NP

Closing ArrayList<Phrase> cleaning_time(ArrayList<Phrase> phrases).

Inside ArrayList<Tripletta> estrai(ArrayList<Phrase> p).

(P)The kidneys + are essential in + the urinary system

(P)The kidneys + serve + homeostatic functions

the regulation of electrolytes DIVIDEDBYmaintenance of acid–base balance DIVIDEDBYregulation of blood pressure + is a + homeostatic functions

Closing ArrayList<Tripletta> estrai(ArrayList<Phrase> p).

Second loop.

Sentence before cleaning:

They serve the body as a natural filter of the blood, and remove wastes, which are diverted to the urinary bladder

Sentence after cleaning:

They serve the body as a natural filter of the blood, and remove wastes, which are diverted to the urinary bladder

Extracted tags:

(They)PRP (serve)VBP (the)DT (body)NN (as)IN (a)DT (natural)JJ (filter)NN (of)IN (the)DT (blood)NN (,), (and)CC (remove)VB (wastes)NNS (,), (which)WDT (are)VBP (diverted)VBN (to)TO (the)DT (urinary)JJ (bladder)NN

Extracted phrases:

(They)NP

(serve)VP

(the body)NP

(as)PP

(a natural filter)NP

(of)PP

(the blood)NP

(remove)VP

(wastes)NP

(which)NP

```
(are diverted)VP
(to)PP
(the urinary bladder)NP
```

Inside ArrayList<Phrase> span2phr(Span[] span, String[] tokens, String[] tags).

Extracted Enriched Phrases:

```
(They)NP
(serve the body as)1VBF
(a natural filter of the blood)NP
(),
(and)AndOr
(remove)VBF
(wastes)NP
(),
(which)NP
(are diverted to)VBF
(the urinary bladder)NP
```

Closing ArrayList<Phrase> span2phr(Span[] span, String[] tokens, String[] tags).

Inside ArrayList<Phrase> fondi_liste2(ArrayList<Phrase> p)

```
(They)NP
(serve the body as)1VBF
(a natural filter of the blood)NP
(),
(and)AndOr
(remove)VBF
(wastes)NP
(),
(which)NP
(are diverted to)VBF
(the urinary bladder)NP
```

Closing ArrayList<Phrase> fondi_liste2(ArrayList<Phrase> p)

Inside ArrayList<Phrase> such_and_ing(ArrayList<Phrase> p).

```
(They)NP
(serve the body as)1VBF
(a natural filter of the blood)NP
(),
(and)AndOr
(remove)VBF
(wastes)NP
(),
(which)NP
(are diverted to)VBF
(the urinary bladder)NP
```

Closing ArrayList<Phrase> such_and_ing(ArrayList<Phrase> p).

Inside ArrayList<Phrase> cleaning_time(ArrayList<Phrase> phrases).

```
(They)NP
(serve the body as)1VBF
(a natural filter of the blood)NP
(),
(and)AndOr
(remove)VBF
(wastes)NP
(),
(which)NP
(are diverted to)VBF
(the urinary bladder)NP
```

Closing ArrayList<Phrase> cleaning_time(ArrayList<Phrase> phrases).

Inside ArrayList<Tripletta> estrai(ArrayList<Phrase> p).

```
(P)They + serve the body as + a natural filter of the blood
(P)They + remove + wastes
(P)which + are diverted to + the urinary bladder
```

Closing ArrayList<Tripletta> estrai(ArrayList<Phrase> p).

Loops are over: now it is the time for "ArrayList<Tripletta> automappa(ArrayList<Tripletta> t)".

Inside ArrayList<Tripletta> automappa(ArrayList<Tripletta> t).

```
The kidneys + are essential in + the urinary system
The kidneys + serve + homeostatic functions
The kidneys + serve the body as + a natural filter of the blood
The kidneys + remove + wastes
wastes + are diverted to + the urinary bladder
the regulation of electrolytes  + is a + homeostatic functions
maintenance of acid–base balance  + is a + homeostatic functions
regulation of blood pressure  + is a + homeostatic functions
```

Closing ArrayList<Tripletta> automappa(ArrayList<Tripletta> t).

Extracted triplets:

```
The kidneys + are essential in + the urinary system
The kidneys + serve + homeostatic functions
The kidneys + serve the body as + a natural filter of the blood
The kidneys + remove + wastes
wastes + are diverted to + the urinary bladder
the regulation of electrolytes  + is a + homeostatic functions
maintenance of acid–base balance  + is a + homeostatic functions
regulation of blood pressure  + is a + homeostatic functions
```

Extracted triplets: 8 triplets from 2 sentences

Closing ArrayList<Tripletta> PHanalizzatore(String Text).

Ends in 3.26436 seconds.

In Figure 37 the graph representation of the 8 extracted triplets is shown.

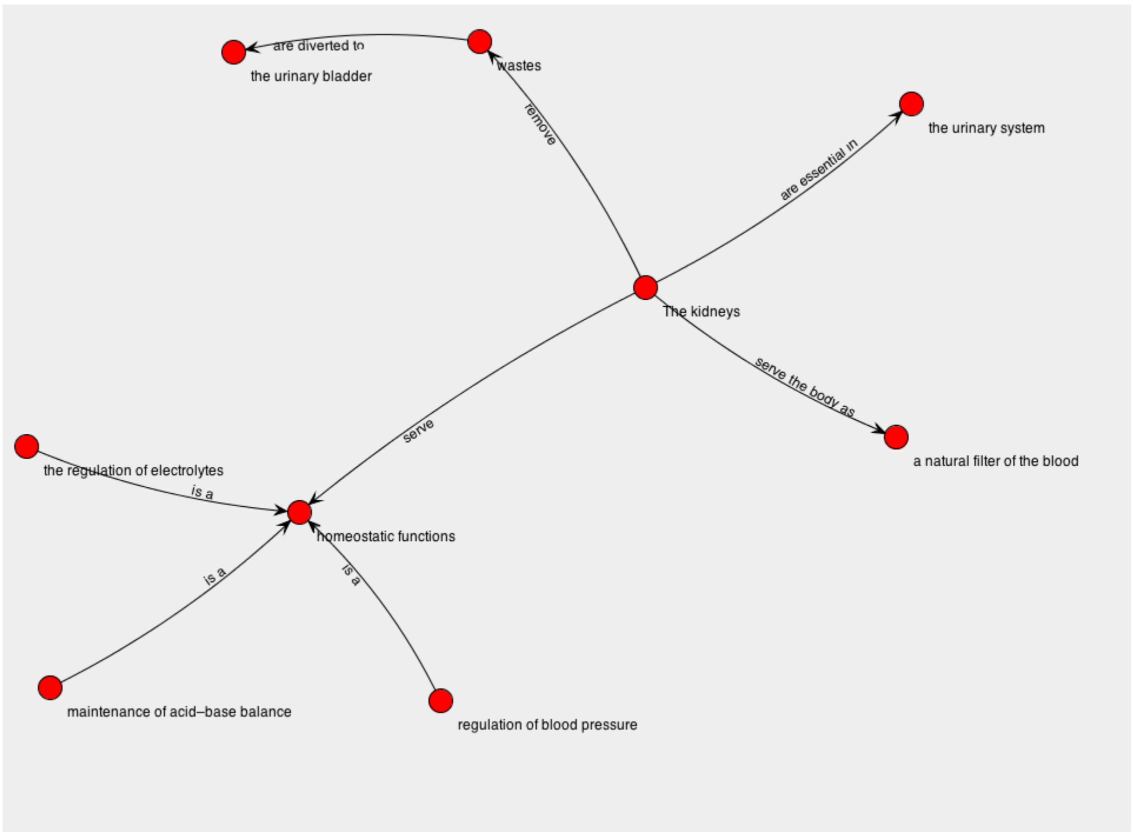


Fig. 37. Output for the input text: "*“The kidneys are essential in the urinary system and also serve homeostatic functions such as the regulation of electrolytes, maintenance of acid–base balance, and regulation of blood pressure (via maintaining salt and water balance). They serve the body as a natural filter of the blood, and remove wastes, which are diverted to the urinary bladder.”*"

Appendix B

In the appendix B some papers about ontologies subject matters are reviewed (table 9).

With "**mapping task**" in Table 9 we refer to an algorithm which is able to (automatically) relate every concept belonging to an ontology with every concept belonging to another one [41]. The **annotation** of a text is the task to associate one (or more) concept(s) from an external ontology to a phrase of a text [1]: the annotation is done through tool called annotator which uses tools called **concept recognizers** (Table 9). **Protégé** is an ontology editor with a clear Graphical User Interface developed by Stanford. The **Systematized Nomenclature of Medicine - Clinical Terms** is a medical terms dictionary.

Table 9. Review of papers on ontologies subject matters.

#	Title	Authors	Complete reference	Aim	Methods	Results	Utility	Query
1	Building a biomedical ontology recommender web service.	Clement Jonquet, Mark Musen, Nigam Shah	Jonquet C, Musen M, Shah N. **Building a biomedical ontology recommender web service.** J Biomed Inform. 2010;1(Suppl. 1):1–18.	The aim of the work is the presentation of “Biomedical Ontology Recommender web service”: it is a web service to find the best fit ontology for a selected text.	Authors have **created an ontology** from a corpus of more than 200 ontologies (122 are from BioPortal, 98 from Unified Medical Language System UMLS): every ontology has some mapping relations with other ones. The created ontology has 4,222,921 concepts and 7,943,757 terms. First step is to annotate the input text with concepts from the created ontology. Then, using mapping information, the system creates more annotations (this step is not mandatory, users can select to use it or not). Then a score is calculated for every ontology: the score is the number of annotations divides by the number of concepts inside the ontology itself. The ontology which scores the higher result is elected **as best fit ontology** for input text. Some tests have been executed on a corpus of three lists of biomedical keywords and three biomedical texts. A group of expert evaluate the selected ontology with a scale from 1 (worst fit ontology) to 5 (best fit ontology).	The system scored a mean of 4.41 if the mapping relations are not used and 4 otherwise.	This work underlines the fact that a lot of different ontologies exist and it is not easy to find the best fit one.	PubMed / Musen+ M[au] AND ontology [tiab]

#	Title	Authors	Complete reference	Aim	Methods	Results	Utility	Query
2	The ontology life cycle: Integrated tools for editing, publishing, peer review, and evolution of ontologies.	Natalya Noy, Tania Tudorache,Csongor Nylulas, Mark Musen	Noy N, Tudorache T, Nyulas C, Musen M. **The ontology life cycle: Integrated tools for editing, publishing, peer review, and evolution of ontologies.** Proceeding of the AMIA Annual Symposium. 2010. p. 552–6.	The aim of the work is to present a suite to support ontology during its whole life cycle.	The suite is composed by WebProtegè, BioPortal, BioPortal reference widget and Notes API. **Web Protégé** is an online version of Protégé: it allows more user to work on the same ontology at the same time. **BioPortal** is an ontology web repository: it allows users to publish and to download ontologies; it also allows users to send notification to ontologies creators to suggest a new concept or to edit an existing one. **BioPortal reference widget** is a tool to search ontologies on BioPortal. **Notes API** is a tool to connect directly WebProtegè with BioPortal: this way ontologies creators can immediately see the notifications from users.	BioPortal satisfies more than 11.000 users every month.	Authors present some useful tools to deal with ontologies subject matters	PubMed / Musen+ M[au] AND ontology [tiab]

#	Title	Authors	Complete reference	Aim	Methods	Results	Utility	Query
3	Comparison of concept recognizers for building the Open Biomedical Annotator.	Shah NH, Bhatia N, Jonquet C, Rubin D, Chiang AP, Musen M a	Shah NH, Bhatia N, Jonquet C, Rubin D, Chiang AP, Musen M a. **Comparison of concept recognizers for building the Open Biomedical Annotator.** BMC Bioinformatics. 2009 Jan;10 Suppl 9:S14.	The aim of this work is to compare two concepts recognizers: the National Library of Medicine's MetaMap (NLM's MetaMap) and the Mgrep.	The task to recognize a given ontology concept in a text is generally referred to as concept recognition. **NLM's MetaMap** and **Mgrep** are concepts recognizers. They both were tested with four sets of 200 sentences randomly extracted from: - ClinicalTrials.gov (http://www.clinicaltrials.gov) - Gold Miner http://http//goldminer.arrs.org Gene Expression - Omnibus http://www.ncbi.nlm.nih.gov/geo/ - PubMed http://www.ncbi.nlm.nih.gov/pubmed As external ontology has been used Systematized Nomenclature of Medicine - Clinical Terms (Snomed-CT).	The authors reported all the results as tables and diagrams. **Mgrep scored always an higher precision** than MetaMap, which always recognizes more concepts.	This work shows the importance of concept recognizer in knowledge management .	PubMed / Musen+ M[au] AND ontology [tiab]

#	Title	Authors	Complete reference	Aim	Methods	Results	Utility	Query
4	Low-cost ontology development	Grác M, Rambousek A, Center N	Grác M, Rambousek A, Center N. **Low-cost ontology development.** Proceeding 6th Int Glob Wordnet Conf. 2012;	The aim of this work is to show every step to develop a general purposes non-English ontology , authors called **Sholva ontology**.	Some tools and sources of knowledge are used: **WordNet ontology**: it is a general purposes ontology created by Princeton and translated in every European Language tanks to EuroWordNet project. **ILI (inter lingual index):** it is a tool to translate concepts and to create multilingual dictionary. **VerbaLex**: it is a Czech language verb meaning lexicon developed by the “Faculty of Informatics, Masaryk University”.	Authors have published **Sholva ontology**: it will be used for machine translation tasks, syntactic parsing tasks and word sense disambiguation tasks.	This work shows how hard could be to create a non-English ontology.	Google scholar / allintitle: "ontology development"
5	The Open Biomedical Annotator	Jonquet C, Shah N, Musen M	Jonquet C, Shah N, Musen M. **The Open Biomedical Annotator.** Summit on translational bioinformatics.	The aim of the work is to present the ***Open Biomedical Annotator*** (OBA): it is a	OBA uses a set of predefined biomedical external ontologies. This set is constituted of every Unified Medical Language System (UMLS) and BioPortal ontology. First OBA chooses the best fit ontology to annotate as the ontology which creates more annotations. Then it create some annotations	OBA has created 174,840,027 annotations: 18%	The results of this work show the importance of mapping task to extract every	PubMed / Musen+M[au] AND ontology [tiab]

#	Title	Authors	Complete reference	Aim	Methods	Results	Utility	Query
			2009. p. 56–60.	web service to automatically create annotations on biomedical texts without asking the user to provide an external ontology.	using **Mgrep**: it is a concepts recogniser developed by the National Center for Integrative Biomedical Informatics. This set of annotations is enriched using three more tools: 1.“**semantic expansion components**”: it creates more annotations using relations inside selected ontology. 2.“**semantic distance component**”: it creates more annotations using similar concepts inside selected ontology; similar concepts are closer concepts in the ontology three-structure. 3.”**ontology-mapping component**” : it creates more annotations using relations among selected ontology and other external ontologies; this tool is referred to as the “*mapping step*”. OBA has been tested on a corpus of 1,050,000 short texts from PubMed, such as papers titles and papers abstracts.	annotations has been obtained using without the mapping step, while others thanks to it. 99% of corpus texts has been annotated with a mean of 165 annotations for text.	piece of knowledge. Furthermore, OBA is a free and available web service for biomedical researchers.	

#	Title	Authors	Complete reference	Aim	Methods	Results	Utility	Query
6	Towards valid and reusable reference alignments - ten basic quality checks for ontology alignments and their application to three different reference data sets	Beisswan-ger E, Hahn U	Beisswanger E, Hahn U. **Towards valid and reusable reference alignments - ten basic quality checks for ontology alignments and their application to three different reference data sets.** J Biomed Semantics. 2012 Jan;3 Suppl 1(Suppl 1):S4.	The aim of the work is to propose an assessment questionnaire to evaluate mapping algorithms.	The authors design **a ten-step flowchart** to evaluate a mapping algorithm from every point of view, such as the implementation point of view (e.g. does exist an implemented version of the algorithm?), the implementation results point of view (e.g. how many non-trivial relations are discovered?). Every step is a multiple choice question: every different answer forces the user to follow one edge which leads him to another question. This questionnaire has been tested on three mapping algorithms: anatomy track reference alignment (OAEI), Linked Open Data (LOD) and BRIDGE.	Tests have shown that every question is useful to assess a mapping algorithm.	This work underlines one mapping algorithm issue: it is pretty hard to find ready-to-use mapping tools for a specific domain.	PubMed / ontology alignment[Title/Abstract] NOT "genes"[MeSH Terms]

#	Title	Authors	Complete reference	Aim	Methods	Results	Utility	Query
7	The Biomedical Resource Ontology (BRO) to enable resource discovery in clinical and translational research.	Tenenbaum JD, Whetzel PL, Anderson K, Borromeo CD, Dinov ID, Gabriel D, et al.	Tenenbaum JD, Whetzel PL, Anderson K, Borromeo CD, Dinov ID, Gabriel D, et al. **The Biomedical Resource Ontology (BRO) to enable resource discovery in clinical and translational research.** J Biomed Inform. 2011 Feb;44(1):137–45.	The aim of the work is to present the design and the implementation of Biomedical Resource Ontology version 3.0 (BRO).	BRO is composed by some tools: 1. **Biositemaps**: a tool to append biomedical tags to biomedical data; it is also a tag-search engine. Tags are written in Resource Description Framework (RDF) format. 2. **The Biositemaps Information Model** (BIM): it describes the standard structure of a valid RDF file. 3. **The Biomedical Resource Ontology** (BRO): it is the tag dictionary; users can only chooses tags to append from this source . 4. **Biositemap RDF file generation and back-end data storage**: graphical user interface to append tags to data. 5. **The resource discovery system (RDS) query tool**: web service to retrieve data from Biositesmap.		This work shows the strength of the ontology approach in information retrieval tasks.	PubMed / Musen+ M[au] AND ontology [tiab]

#	Title	Authors	Complete reference	Aim	Methods	Results	Utility	Query
8	Mapping between the OBO and OWL ontology languages,	Tirmizi SH, Aitken S, Moreira D a, Mungall C, Sequeda J, Shah NH, et al.	Tirmizi SH, Aitken S, Moreira D a, Mungall C, Sequeda J, Shah NH, et al. **Mapping between the OBO and OWL ontology languages.** J Biomed Semantics. BioMed Central Ltd; 2011 Jan;2 Suppl 1(Suppl 1):S3.	The aim of the work is to create the standard for the conversion from Open Biological and Biomedical Ontologies (OBO) format to Ontology Web Language - Description Logic (OWL-DL) format.	OWL-DL format **has more expressivity** than OBO format, thus it is easy to map OBO ontologies into OWL one. OBO ontologies have an header with some meta-data: they will be stored inside an “owl:ontology” element. Other elements are almost equal, such as OBO terms and OWL class, OBO name and “rdfs:label”, OBO comments and “rdfs:comment”.... The main difference between OBO and OWL is OBO uses local identifiers while OWL uses global identifiers (URI and URL). Authors used information in the OBO header to convert local identifiers to global ones. In the end, authors created the “oboInOwl:Subset” to represent OBO subsets.	Converted ontologies has been validated thorugh **WonderWeb OWL Ontology Validator**, a World Wide Web Consortium (W3C) tool to check if an ontology is valid.	Authors present and implement a tool to facilitate ontologies interoperability.	Pubmed / ontology [All Fields] AND mapping[All Fields] AND ("Biomedicine"[Journal] OR "biomedicine"[All Fields])

#	Title	Authors	Complete reference	Aim	Methods	Results	Utility	Query
9	Applications of ontology design patterns in biomedical ontologies.	Mortensen JM, Horridge M, Musen M a, Noy NF.	Mortensen JM, Horridge M, Musen M a, Noy NF. **Applications of ontology design patterns in biomedical ontologies.** Proceeding of AMIA Annual Symposium. 2012. p. 643–52.	The aim of the work is to determine the use and applicability of **Ontology design patterns** (ODPs) in a case study of biomedical ontologies. They also verified if some old ontologies has been developed thanks to ODP **without creators being aware of using it**.	**Ontology design patterns** (ODPs) are a proposed solution to facilitate ontology development, and to help users avoid some of the most frequent modeling mistakes. Authors used two different ODP: "**Ontology Design Patterns.org**" (ODP-Wiki) and "**The Manchester catalogue of ontology design patterns for bio-ontologies**" (MBOP); they are both a set of rules expressed in natural language. Author implemented a software to check if a specific rule has been used inside an ontology: they implemented 64 patterns using Ontology Pre-Processing (OPPL), and 5 patterns using Java programming language. The implemented rules has been **tested on some OWL ontologies** (Gene Ontology, Foundational Model of Anatomy, National Cancer Institute Thesaurus). Results has been validated by an expert.	**Only 5 patterns** has been identified inside ontologies.	This work underlines how **ontology creation is still an handmade process**: it does not follow any standardized rule.	PubMed / Musen+ M[au] AND ontology [tiab]

#	Title	Authors	Complete reference	Aim	Methods	Results	Utility	Query
10	Comparison of ontology-based semantic-similarity measures.	Lee W-N, Shah N, Sundlass K, Musen M	Lee W-N, Shah N, Sundlass K, Musen M. **Comparison of ontology-based semantic-similarity measures.** Proceeding of the the AMIA Symposium. 2008. p. 384–8.	The aim of the work is **to compare** three different similarity measure metrics.	Semantic-similarity measures quantify concept similarities in a given ontology. First algorithm has been presented by **Al-Mubaid et al.** : it uses the features of a concept's location within a cluster of nodes to modify the shortest path distance between two concepts. The similarity measure metrics for two concepts C_1 *and* C_2 is defined as $$Sem(C_1, C_2) = \log\left((Path - 1)^{\alpha} * (Cspec)^{\beta} + k\right)$$ where Path is the shortest path length between the two concepts, k is parameters which models local topography; Cspec, α and β are the weighted contributions of local granularity. Second algorithm is named **Descendant Distance** (DD) proposed by Melton: it measures the clinical distance between two patients (P_a, P_b) based on the diseases present in each, and it is analagous to distances between a pair of diseases. The generalization of this distance metric is: $$distance(C_1, C_2) = \sum P(V_y)$$	The only significant, albeit weak, correlation to experts evaluation occurred with **Al-Mubaid et al.** algorithm.	This work is really useful because it shows how weak are similarity metrics algorithm in a biomedical domain.	PubMed / Musen+ M[au] AND ontology [tiab]

#	Title	Authors	Complete reference	Aim	Methods	Results	Utility	Query
					where $P(V_y) = \frac{No.descendants\ of\ each\ path\ node}{Total\ No.of\ Terms}$, and V_y are nodes among C_1, C_2 in the graph. Last metrics is named **Term Frequency** (TF), proposed by Melton: it uses a concept's term frequency in a corpus of text to weight the path edges within an ontology. The term frequency came from a clinical data repository of more than 14,000 patients to derive a set of all SNOMED-CT concepts used. The generalization of this distance metric is: $$distance(C_1, C_2) = \sum P(V_y)$$ where $P(V_y) = \frac{Term\ Frequency\ of\ disease\ concept}{Max.Term\ Frequency\ in\ Corpus}$, and V_y are nodes among C_1, C_2 in the graph. **Test corpus** is composed by 190 couples from 20 disease names extracted from Systematized Nomenclature of Medicine Clinical Terms (SNOMED CT). The assessment of the result has been done by 25 clinicians with a 7-point Likert scale of least similar to most similar.			

#	Title	Authors	Complete reference	Aim	Methods	Results	Utility	Query
11	Use of description logic classification to reason about consequence s of penetrating injuries.	Rubin DL, Dameron O, Musen M	Rubin DL, Dameron O, Musen M a. **Use of description logic classification to reason about consequences of penetrating injuries.** Proceeding of the AMIA Annual Symposium Proceedings. 2005. p. 649–53.	The aim of the work is to demonstrate the capabilities of automated classification using the Web Ontology Language (OWL) to reason about the consequences of penetrating injuries.	Authors **designed an ontology** of chest and heart anatomy describing the heart structure and the surrounding anatomic compartments, as well as the perfusion of regions of the heart by branches of the coronary arteries. They used two knowledge sources: - **Foundational Model of Anatomy** (FMA), for anatomical pieces of information. - **Foundational Model of Physiology** (FMP), for physiological pieces of information. The ontology has been imlemented using Protegé. They extended the base OWL ontology in two ways to create two different reasoning services: 1.“**Cardiac Ischemia Reasoner**”: it infers regions of heart damage secondary to coronary artery injuries; 2. “**Injury Propagation Reasoner**”: it infers the propagation of initial injury caused by bleeding into breached anatomic compart- ments. The base OWL ontology was extended to create these applications by adding class restrictions and defined classes to represent additional anatomic and physiological knowledge needed by their	Both reasoner **has been validated by the clinicians**: they always infer injuries damage **correctly**.	This work shows how some of the semantic capabilities of ontologies we explained in par 2.5 can be used to create new knowledge.	PubMed / Musen+ M[au] AND ontology [tiab]

#	Title	Authors	Complete reference	Aim	Methods	Results	Utility	Query
					application but not available in the FMA and FMP. Thus, they reused the original knowledge representation of anatomy and physiology, and they developed two different reasoning services in a modular manner. Some **clinician** tested the system with some simulated injuries.			

#	Title	Authors	Complete reference	Aim	Methods	Results	Utility	Query
12	Biotea: RDFizing PubMed Central in support for the paper as an interface to the Web of Data.	Garcia Castro LJ, McLaughlin C, Garcia A. Biotea: RDFizing PubMed Central in support for the paper as an interface to the Web of Data.	Garcia Castro LJ, McLaughlin C, Garcia A. **Biotea: RDFizing PubMed Central in support for the paper as an interface to the Web of Data.** J Biomed Semantics. BioMed Central Ltd; 2013 Apr 15;4(Suppl 1).	The aim of the work is to present authors' approach to the generation of self-describing machine-readable scholarly documents.	Authors has created some tools **to automatically encode** some pieces of information about a PubMed paper using Resource Description Framework (**RDF**) format. These pieces of information are both papers meta-data and annotations created using some external ontologies. To **extract meta-data** authors have used Bibliographic Ontology (BIBO), Dublin Core Metadata Initiative (DCMI), Friend of a Friend (FOAF) and Provenance Ontology (PROV-O). **Annotations** has been appended using National Center for Biomedical Ontology(NCBO) Annotator.	Authors have semanti-cally processed the full-text, open-access subset of PubMed Central. The enriched dataset is available at http://biotea.idiginfo.org/	This work is an example of an effort to automatically adds semantic to dataset and shows how this approach increase the retrievability of contents on web.	Google Scholar / allintitle: ontology alignment -gene -genome -RNA – protein

#	Title	Authors	Complete reference	Aim	Methods	Results	Utility	Query
13	NCBO Tech-nology: Powering semantic-cally aware applica-tions.	Whetzel PL	Whetzel PL. **NCBO Technology: Powering semantically aware applications.** J Biomed Semantics. BioMed Central Ltd; 2013 Apr 15;4 Suppl 1(Suppl 1):S8.	The aim of the work is to present some tools developed by the National Center for Biomedical Ontology(NCBO).	These tools are: - **NCBO BioPortal**: web repository to publish ontologies. - **Ontology Web services**: web interface to search published ontologies on BioPortal. - **Widgets services**: API to use Ontology Web services on others websites - **Mapping web services**: it allows users to obtain every available mapping relations about mapped ontologies published on BioPortal. - **NCBO Annotator Web service**: web service to automatically append annotations to biomedical texts.	No result is presented.	The NCBO tools are useful tool to work in the ontology subject matter.	Google Scholar / allintitle: ontology alignmen t -gene -genome -RNA – protein

#	Title	Authors	Complete reference	Aim	Methods	Results	Utility	Query
14	Semantic interoperability of clinical data.	Berges I, Bermudez J.	1. Berges I, Bermudez J. **Semantic Interoperability of Clinical Data.** Proceedings of the First International Workshop on Model-Driven Interoperability ACM. 2010. p. 10–4.	The aim of the work is to present a proposal which allows one system to interpret on the fly clinical data sent by another one even when they use different data representation.	This approach relies on three components: 1. **An ontology** that provides –in its **upper level**– a canonical representation of EHR statements, more precisely of medical observations, which can be then specialized –in the **lower level**– by health institutions according to their proprietary models, 2. **A translator module** that facilitates the definition of the lower level of the ontology from the particular EHRs data storage structures following a semi-automatic approach: first a translation process of underlying data structures, using –whenever possible– information about properties (functional dependencies, etc.) into ontology elements described in OWL2, and next, an edition process where the health system administrators can define new axioms to adjust and enrich the result obtained in the semi-automatic process 3. **A mapping module** that helps in the task of defining the links among the terms of the upper and lower levels of the ontology.	No result is presented.	This work shows a scenario where some issues of data-base subject matter are solved using ontologies.	Google Scholar / allintitle: Semantic clinical data

#	Title	Authors	Complete reference	Aim	Methods	Results	Utility	Query
15	OWLlink.	Liebig T, Luther M, Noppens O, Wessel M.	Liebig T, Luther M, Noppens O, Wessel M. **OWLlink**. Semant Web. 2011;1:23–32.	The aim of the work is to present OWLlink protocol.	OWLlink is **an implementation-neutral protocol** for communication between OWL2 components .	No result is presented.	OWLlink API.	Google Scholar / OWLlink
16	The OWL API : A Java API for Working with OWL 2 Ontologies.	Horridge M, Bechhofer S.	Horridge M, Bechhofer S. **The OWL API : A Java API for Working with OWL 2 Ontologies.** OWLED. 2009;2009(Owled).	The aim of the work is to present the Application Programming Interface (API) **OWL API**.	The **OWL API** consists of a set of interfaces for inspecting, manipulating and reasoning with OWL ontologies. The OWL API supports loading and saving ontologies is a variety of syntaxes. An ontology is simply viewed as a set of axioms and annotations.	No result is presented.	OWL API.	Google Scholar / OWL API

#	Title	Authors	Complete reference	Aim	Methods	Results	Utility	Query
17	The OWLlink API: Teaching OWL Components a Common Protocol.	Noppens O, Luther M, Liebig T.	Noppens O, Luther M, Liebig T. **The OWLlink API: Teaching OWL Components a Common Protocol.** OWLED. 2010;3–6.	The aim of the work is to present the Application Programming Interface (API) **OWLlink API**.	The **OWLlink API** is a Java API which implements every rule of OWLlink protocol.	No result is presented.	OWLlink API.	Google / The OWLlink API

#	Title	Authors	Complete reference	Aim	Methods	Results	Utility	Query
18	Addressing Issues in Foundational Ontology Mediation.	Khan Z, Keet C.	1. Khan Z, Keet C. **Addressing Issues in Foundational Ontology Mediation.** Proceedings of the 5th International Conference on Knowledge Engineering and Ontology Development (KEOD'13). 2013.	The aim of the work is to compare the performances of seven mapping algorithms in mapping three foundational ontologies.	Authors have compared **these algorithms**: H-Match, PROMPT, LogMap, YAM++, HotMatch, Hertuda, Optima. Two different metrics has been used to evaluate performances: Accuracy and Found. In symbol: $\boldsymbol{Accuracy}_{j-esim\ algorithm} = \frac{r_{aj,correct}}{r_{aj,total}} * 100$ and $\boldsymbol{Found}_{j-esim\ algorithm} = \frac{r_{aj,correct}}{r_{baseline}} * 100,$ where $r_{baseline}$ are the relations created by a group of expert and $r_{aj,total}$is the number of created relation by the j-esm algorithm. The **ontologies corpus** is composed by BFO, GFO e DOLCE.	LogMap scores the best results: Accuracy = 94% and Found = 40%	This work shows how almost every algorithm scores bad results when used to map different ontologies than the ones algorithm creators had used to develop the algorithm itself.	Google Scholar / protegé mapping tool - gene -rna - genome

#	Title	Authors	Complete reference	Aim	Methods	Results	Utility	Query
19	The PROMPT suite: interactive tools for ontology merging and mapping.	Noy NF, Musen M.	Noy NF, Musen M a. **The PROMPT suite: interactive tools for ontology merging and mapping.** Int J Hum Comput Stud. 2003 Dec;59(6):983–1024.	The aim of the work is to present the PROMPT suite of tool.	PROMPT suite is a Protégé plugin. It is composed by 1.**IPROMPT**: it is a tool to semi-automatically map ontologies. It shows the user some possible equivalent couples of concepts and the user only has to confirm or reject every couples. 2.**ANCHORPROMPT**: it shows some other possible equivalent couples of concepts than IPROMPT ones. 3.**PROMPTDIFF**: it is a tool to notify difference about some versions of the same ontology. 4.**PROMPTFACTOR**: it is a tool to create a sub-ontology selecting some concepts from an ontology.	No result is presented.	PROMPT suite is almost a “plug and play” application software. The real use of PROMPT is not as easy as the paper states.	Google Scholar / PROMPT ontology tool

#	Title	Authors	Complete reference	Aim	Methods	Results	Utility	Query
20	JOINT: Java ontology integrated toolkit.	Holanda O, Isotani S, Bittencourt II, Elias E, Tenório T.	Holanda O, Isotani S, Bittencourt II, Elias E, Tenório T. **JOINT: Java ontology integrated toolkit.** Expert Syst Appl. Elsevier Ltd; 2013 Nov;40(16):6469–77.	The aim of the work is to propose a framework and a tool for supporting the efficient development of ontology-based applications through the integration of existing technologies. The tool is called JOINT.	JOINT **is a Java toolkit** to help users in the creation of ontology-based application software. The JOINT architecture is based on the layers pattern where each layer uses only the services of the layer below. JOINT provides three layers (modules) for users (mainly developers and knowledge engineers): 1. **an API**, for ontology-based application developers to implement functionalities; 2. **a Desktop interface**, for knowledge engineers unfamiliar with programming; 3. **plugins** on (and for) external tools, to optimize the work of both users. JOINT **has been tested** asking to four couple of students to implement a programming software given specific requirements: no student had expertise in ontologies development. JOINT has been assigned to two couples to fulfill the requirements while Jestor/Jena (another API to implement ontology based software) has been assigned to other two couples. Results metrics are the number of lines code written, the time needed to complete the application software.	Couples using JOINT has been faster (6 and 7 hours versus 15 and 18). All the couples wrote almost the same number of lines.	JOINT is an useful suite to develop ontology application software.	Google Scholar / Java Ontology tool

#	Title	Authors	Complete reference	Aim	Methods	Results	Utility	Query
21	Consumer health concepts that do not map to the UMLS: where do they fit?	Keselman A, Smith C, Divita G.	1. Keselman A, Smith C, Divita G. **Consumer health concepts that do not map to the UMLS: where do they fit?** J Am Med Informatics Assoc. 2008;15(4):496–505.	The aim of the work is to present a systematic approach to deal with every common health term (CHT) which does not have any direct map to any UMLS concept.	Any CHT can be in one of the following categories: - same label as a UMLS concept and same meaning (e.g. “pain” and “UMLS:pain”). - same meaning as a UMLS concept but different label (e.g. “noisebleed” and “UMLS:epistaxis”) - same label as UMLS concept but different meaning (e.g. “leg” as CHT means the anathomic part from ankle to hip while in UMLS Methatesaurus is defined as “the inferior part of the lower extremity between the KNEE and the ANKLE”). - does not any UMLS concept with same meaning (thus it is not mappable inside UMLS). Authors want to deal with the fourth category: they create some teams of domain expert to find the more similar UMLS concepts of every non-mappable concept and to find the relation between non-mappable concepts and more similar UMLS concepts (e.g. “diet pills” is similar to “UMLS: Weight-Loss Agents”, and the relation among them is “diet pills” “is more specific than” “UMLS: Weight-Loss Agents”).	Experts found 36 non-mappable CHT and thus they created by hand missing UMLS concepts and relations.	The mapping among biomedical ontologies and common health vocabularies could be very useful to spread knowledge also to non-domain expert users.	Paper has been downloaded from http://www.consumerhealthvocab.org/

#	Title	Authors	Complete reference	Aim	Methods	Results	Utility	Query
22	Identifying consumer-friendly display (CFD) names for health concepts.	Zeng QT, Tse T, Crowell J, Divita G, Roth L, Browne AC.	Zeng QT, Tse T, Crowell J, Divita G, Roth L, Browne AC. **Identifying consumer-friendly display (CFD) names for health concepts.** Proceeding of the AMIA Annual Symposium . 2005 Jan;859–63.	The aim of the work is to present a systematic approach to append to every Unified Medical Language System (UMLS) concept the best-fit **consumer-friendly display** (CFD) names.	Authors **created a corpus of common health terms** (CHTs) from 12.5 millions queries on NLM MedlinePlus®. Every time more than one CHTs refer to the same UMLS concept, authors have chosen by hand one of this list as CFD. They mapped CHTs to 96.029 UMLS concepts by hand. To evaluate the corpus of CFD, authors have created a questionnaire with **34 fill-in-the-blank questions**, each with four multiple-choice selections: an answer and three distractors. Each question, designed to test a person's ability to understand a health concept, has two versions: one using the CFD name of a concept; the other using either the UMLS preferred term or the most frequently used alternate name (other than a lexical variant of the CFD name). Every test had 17 questions with UMLS answers and 17 questions with CFD answers. **Participants** (n=10; non-clinician, at least 18 years old, English speaking) **were recruited** from the lobbies of the Brigham and Women's	Testers have scored a mean of 15.6/17 in the CFD answers part and only 6.0 / 17 in the UMLS one.	This work underlines how hard is to map common health terms into biomedical ontologies terms.	Paper has been downloaded from http://www.consumerhealthvocab.org/

#	Title	Authors	Complete reference	Aim	Methods	Results	Utility	Query
					Hospital. A paired t-test was used for the hypothesis that the mean score on CFD questions was greater than that on non-CFD questions.			

7. References

1. Jonquet C, Shah N, Musen M. The Open Biomedical Annotator. Proceeding of the Summit on translational bioinformatics. 2009. p. 56–60.

2. Gruber T. The role of common ontology in achieving sharable, reusable knowledge bases. KR. 1991;601–2.

3. Tirmizi SH, Aitken S, Moreira D a, Mungall C, Sequeda J, Shah NH, et al. Mapping between the OBO and OWL ontology languages. J Biomed Semantics. 2011 Jan;2 Suppl 1:S3.

4. Ashburner M, Ball C, Blake J, Botstein D. Gene Ontology: tool for the unification of biology. Nat Genet. 2000;25(1):25–9.

5. Rosse C, Jr JLVM. A reference ontology for biomedical infortmatics: the Foundational Model of Anatomy. J Biomed Inform. 2003;36(6):478 – 500.

6. Bodenreider O. The Unified Medical Language System (UMLS): integrating biomedical terminology. Nucleic Acids Res. 2004 Jan 1;32(1):267–70.

7. Medicine USNL of. Unified Medical Language System [Internet]. [cited 2014 Mar 3]. Available from: http://www.nlm.nih.gov/research/umls/

8. Pirró G, Talia D. UFOme: An ontology mapping system with strategy prediction capabilities. Data Knowl Eng. 2010 May;69(5):444–71.

9. Xueyong L, Quanrui W. The design and analysis of semantic web-based ontology mapping model. Proceedings of the International Conference on Educational and Network Technology (ICENT 2010). 2010. p. 75–8.

10. Kashyap V. The UMLS Semantic Network and the Semantic Web. Proceeding of AMIA Annual Symposium. 2003. p. 351.

11. Smith B, Brochhausen M. Putting biomedical ontologies to work. Methods Inf Med. 2010 Jan;49(2):135–40.

12. Khan Z, Keet C. Addressing Issues in Foundational Ontology Mediation. Proceedings of the 5th International Conference on Knowledge Engineering and Ontology Development (KEOD'13). 2013.

13. Lindberg, DA Humphreys, BL McCray A. Unified Medical Language System. Methods Inf Med. 1993;32(4):281 – 291.

14. Hitzler P, Krötzsch M, Rudolph S. The quest for Semantic. Foundations of Semantic Web Technologies. New York: Chapman & Hall/CRC; 2009. p. 1–16.

15. Gruber T. A Translation Approach to Portable Ontology Specifications. Knowl Acquis. 1993;5(2):199–220.

16. Fensel D, Horrocks I, Harmelen F Van, Mcguinness D. OIL Ontology Infrastructure to Enable the Semantic Web. Intell Syst IEEE. 2001;16(2):38–45.

17. Jazirehi AR, Nazarian R, Torres-Collado AX, Economou JS. Aberrant apoptotic machinery confers melanoma dual resistance to BRAF(V600E) inhibitor and immune effector cells: immunosensitization by a histone deacetylase inhibitor. Am J Clin Exp Immunol. 2014 Jan;3(1):43–56.

18. Hitzler P, Krötzsch M, Rudolph S. Simple Ontologies in RDF and RDF Schema. Foundations of Semantic Web Technologies. New York: Chapman & Hall/CRC; 2009. p. 19–72.

19. Jonquet C, Musen M, Shah N. Building a biomedical ontology recommender web service. J Biomed Inform. 2010;1(Suppl. 1):1–18.

20. Consortium TGO. the Gene Ontology [Internet]. [cited 2014 Mar 3]. Available from: http://www.geneontology.org

21. University of Washington School of Medicine. Foundational Model of Anatomy [Internet]. [cited 2014 Mar 5]. Available from: http://sig.biostr.washington.edu/projects/fm/

22. Word Health Organization. International Classification of Diseases [Internet]. Available from: http://www.who.int/classifications/icd/en/

23. Colombetti, M. Some Properties of Logical Systems. 2013. Lecture notes for the Knowledge Engineering course. Downloadable from the course's website, http://home.dei.polimi.it/colombet/KE/

24. Sumathi S, Esakkirajan. S. Structured Query Language. Fundamentals of Relational Database Management Systems. New York, New York, USA: Springer; 2007. p. 111–212.

25. Hitzler P, Krotzsch M, Rudolph S. Foundations of semantic web technologies. New York: Chapman & Hall/CRC; 2011.

26. Wang H. Some facts about Kurt Gödel. J Symb Log. 1981;1(1):653–9.

27. Hitzler P, Krötzsch M, Rudolph S. OWl Formal Semantics. Foundations of Semantic Web Technologies. New York: Chapman & Hall/CRC; 2009. p. 159–210.

28. Hitzler P, Krötzsch M, Rudolph S. Extensible Markup Language XML. Foundations of semantic web technologies. New York: Chapman & Hall/CRC; 2009. p. 353–62.

29. Mangold C. A survey and classification of semantic search approaches. Int J Metadata, Semant Ontol. 2007;2(1):23–34.

30. Fagin R, Kolaitis PG, Miller RJ, Popa L. Data exchange: semantics and query answering. Theor Comput Sci. 2005 May;336(1):89–124.

31. Rubin DL, Dameron O, Musen M a. Use of description logic classification to reason about consequences of penetrating injuries. Proceeding of the AMIA Annual Symposium. 2005. p. 649–53.

32. National Library of Medicine, National Institutes of Health. PubMed [Internet]. [cited 2014 Feb 2]. Available from: http://www.ncbi.nlm.nih.gov/pubmed

33. Halevy A, Rajaraman A, Ordille J. Data integration: the teenage years. Proceedings of the 32nd international conference on Very large data bases. 2006. p. 9–16.

34. Berges I, Bermudez J. Semantic Interoperability of Clinical Data. Proceedings of the First International Workshop on Model-Driven Interoperability ACM. 2010. p. 10–4.

35. Cristani M, Cuel R. A survey on ontology creation methodologies. Int J Semant Web Inf Syst. 2005;1(2):49–69.

36. Sugumaran V, Storey VC. Ontologies for conceptual modeling: their creation, use, and management. Data Knowl Eng. 2002 Sep;42(3):251–71.

37. Noy NF, McGuinnes DL. Ontology Development 101: A Guide to Creating Your First Ontology [Internet]. [cited 2014 Mar 12]. Available from: http://protege.stanford.edu/publications/ontology_development/ontology101-noy-mcguinness.html

38. AI3. A Reference Guide to Ontology Best Practices [Internet]. 2010 [cited 2014 Mar 12]. Available from: http://www.mkbergman.com/911/a-reference-guide-to-ontology-best-practices/

39. Noy N, Tudorache T, Nyulas C, Musen M. The ontology life cycle: Integrated tools for editing, publishing, peer review, and evolution of ontologies. Proceeding of the AMIA Annual Symposium. 2010. p. 552–6.

40. Grác M, Rambousek A, Center N. Low-cost ontology development. Proceeding of the 6th Int Glob Wordnet Conf. 2012. p. 299-305.

41. Chua W, Kim J. Discovering cross-ontology subsumption relationships by using ontological annotations on biomedical literature. Proceeding of the ICBO. 2012. p. 1–5.

42. Noy NF, Musen M a. The PROMPT suite: interactive tools for ontology merging and mapping. Int J Hum Comput Stud. 2003 Dec;59(6):983–1024.

43. Lee W, Shah N. Comparison of Ontology-based Semantic-Similarity Measures. AMIA annual symposium. 2008. p. 384–8.

44. Marquet G, Mosser J, Burgun A. A method exploiting syntactic patterns and the UMLS semantics for aligning biomedical ontologies: the case of OBO disease ontologies. Int J Med Inform. 2007 Dec;76 Suppl 3:53–61.

45. Chua WWK, Kim J-J. BOAT: automatic alignment of biomedical ontologies using term informativeness and candidate selection. J Biomed Inform. 2012 Apr;45(2):337–49.

46. Huang J, Dang J, Huhns MN, Zheng WJ. Use artificial neural network to align biological ontologies. BMC Genomics. 2008 Jan;9(Suppl 2):S16.

47. Rajakaruna G. Agent based Protégé plugin for ontology alignment. Proceeding of the conference on Industrial and Information Systems (ICIIS) 2012 7th IEEE International. 2012. p. 1–6.

48. Belhadef H. A New Bidirectional Method for Ontologies Matching. Procedia Eng. 2011 Jan;23:558–64.

49. Cotterell ME, Medina T. A Markov Model for Ontology Alignment. arXiv Prepr arXiv13045566. 2013 Apr 19;

50. Nasir M, Hoeber O, Evermann J. Supporting Ontology Alignment Tasks with Edge Bundling. Proceedings of the 13th International Conference on Knowledge Management and Knowledge Technologies. 2013. p. 11.

51. Noy N, Musen M. Anchor-PROMPT: Using non-local context for semantic matching. Proceedings of the workshop on Ontologies and Information Sharing at the International Joint Conference on Artificial Intelligence (IJCAIW). 2001.

52. Ngo D, Bellahsene Z. YAM ++: A multi-strategy based approach for Ontology matching task. Proceedings of the Knowledge engineering and management: 18th international conference, EKAW 2012 18th, European Knowledge Acquisition Workshop; 2012. p. 421–5.

53. Allen K, Lancour H, Daily J, Kent A. Natural Language Processing. In: Marcel Decker, editor. Encyclopedia of library and information science. 2nd ed. 1973.

54. Ballard K. Introduction. The frameworks of English. New York: Palgrave MacMillan; 2001. p. 3–14.

55. Ballard K. Word classes. The frameworks of English. New York: Palgrave MacMillan; 2001. p. 15–49.

56. Ballard K. Phrases. The frameworks of English. New York: Palgrave MacMillan; 2001. p. 93–118.

57. Ballard K. Clauses. The frameworks of English. New York: Palgrave MacMillan; 2001. p. 119–45.

58. Sang E. Transforming a Chunker to a Parser. Lang Comput. 2001;37(1):177–88.

59. Chávez A, Orquín A, Dávila H, Gutiérrez Y. UMCC-DLSI: multidimensional lexical-semantic textual similarity. Proceedings of the First Joint Conference on Lexical and Computational Semantics. 2012. p. 608–16.

60. Raghavan H, Allan J. Using soundex codes for indexing names in ASR documents. Proceedings of the Workshop on Interdisciplinary Approaches to Speech Indexing and Retrieval at HLT-NAACL 2004. 2004. p. 22–7.

61. Boehm B. A view of 20th and 21st century software engineering. Proceedings of the 28th international conference on Software engineering. 2006. p. 12–29.

62. Bennett C, Ryall J, Spalteholz L, Gooch A. The Aesthetics of Graph Visualization. Comput Aesthet. 2007;57–64.

63. Noack A. An energy model for visual graph clustering. Graph Draw. 2004;425–36.

64. The Apache Software Foundation. openNLP [Internet]. Available from: http://opennlp.apache.org

65. The JUNG Framework Development Team. Java Universal Network/Graph Framework.

66. Apache Software Foundation. Apache Jena [Internet]. Available from: https://jena.apache.org

67. Wikipedia [Internet]. Available from: http://en.wikipedia.org

68. Kamel H, Navi BB, Sriram N, Hovsepian D a, Devereux RB, Elkind MS V. Risk of a Thrombotic Event after the 6-Week Postpartum Period. N Engl J Med. 2014 Feb 13;1–9.

69. Simic D, Simutis F, Euler C, Thurby C, Peden WM, Bunch RT, et al. Determination of relative Notch1 and gamma-secretase-related gene expression in puromycin-treated microdissected rat kidneys. Gene Expr. 2013 Jan;16(1):39–47.

70. Go A, Chertow G. Chronic kidney disease and the risks of death, cardiovascular events, and hospitalization. N Engl J Med. 2004;351(13):1296–305.

71. Opelz G, Wujciak T, Ritz E. Association of chronic kidney graft failure with recipient blood pressure. Kidney Int. 1998;53(1):217–22.

72. Consortium International Polycystic Kidney Disease. Polycystic kidney disease: the complete structure of the PKD1 gene and its protein. The International Polycystic Kidney Disease Consortium. Cell. 1995 Apr 21;81(2):289–98.

73. Chertow G, Burdick E. Acute kidney injury, mortality, length of stay, and costs in hospitalized patients. J Am Soc Nephrol. 2005;16(11):3365–70.

74. Perlmutter JS, Mink JW. Deep brain stimulation. Annu Rev Neurosci. 2006 Jan;29:229–57.

75. Breit S, Schulz JB, Benabid A-L. Deep brain stimulation. Cell Tissue Res. 2004 Oct;318(1):275–88.

76. Leone M, May A, Franzini A, Broggi G, Dodick D, Rapoport A, et al. Deep brain stimulation for intractable chronic cluster headache: proposals for patient selection. Cephalalgia an Int J headache. 2004 Nov;24(11):934–7.

77. Leone M, Franzini A, Felisati G, Mea E, Curone M, Tullo V, et al. Deep brain stimulation and cluster headache. Neurol Sci Off J Ital Neurol Soc Ital Soc Clin Neurophysiol. 2005 May;26(Suppl 2):s138–9.

78. Moll CKE, Galindo-Leon E, Sharott A, Gulberti A, Buhmann C, Koeppen JA, et al. Asymmetric pallidal neuronal activity in patients with cervical dystonia. Front Syst Neurosci. 2014 Jan;8:15.

79. Cimiano P, Völker J. Text2Onto. Natural language processing and information systems. Springer B. Berlin; 2005. 227–38.

80. Rusu D, Dali L, Fortuna B. Triplet extraction from sentences. Proceedings of the 10th International Multiconference" Information Society-IS. 2007. p. 8 – 12.

81. Berges I, Bermudez J. Semantic interoperability of clinical data. Proceeding of MDI '10 Proceedings of the First International Workshop on Model-Driven Interoperability. 2010. p. 10–4.

82. Mortensen JM, Horridge M, Musen M a, Noy NF. Applications of ontology design patterns in biomedical ontologies. Proceeding of AMIA Annual Symposium. 2012. p. 643–52.

Acknowledgement

Vorrei ringraziare il mio relatore, Professor Stefano Bonacina per avermi accompagnato e sostenuto in questa lunga avventura, per i suoi preziosi consigli e per tutto il tempo che mi ha sempre dedicato con passione e entusiasmo. Vorrei ringraziarlo di cuore per aver reso questi mesi di lavoro estremamente interessanti e stimolanti, per le sue critiche precise e costruttive, per avermi imposto un ordine da seguire, lasciandomi però libero di esprimermi. Vorrei infine ringraziarlo per aver creduto in me.

Vorrei ringraziare il Professore Francesco Pinciroli per avermi ospitato nel laboratorio, per essersi sempre interessato al mio lavoro, per tutti quei consigli che hanno contribuito allo sviluppo finale del mio lavoro, per la passione con la quale fa il suo lavoro e per avermi fatto scoprire dei posti incredibili all'interno del Politecnico.

Vorrei ringraziare la Professoressa Sara Marceglia per avermi sopportato in laboratorio nella scrivania di fianco alla sua, per gli articoli che nei mesi mi ha suggerito di leggere, per l'interesse reale che ha mostrato in quanto stavo facendo.

Vorrei ringraziare Giulia Tanzi per la parte di linguistica inglese, per il supporto che ogni giorno mi ha saputo dare: senza di lei questo lavoro non sarebbe stato possibile.

Vorrei ringraziare la mia famiglia per essermi stata vicina in questi mesi difficili e per aver da sempre creduto in me anche quando le evidenze avrebbero suggerito il contrario.

Vorrei infine ringraziare il Professor Rodolfo Fiorini, persona che ha cambiato il mio modo di vedere le cose. Per quanto il nostro rapporto non sia sempre stato "facile", non smetterò mai di ringraziarlo per tutto quello che mi ha dato.