

An Ontology System for Rehabilitation Robotics

by
Zeynep Dođmuş

August, 2013

Submitted to the Graduate School of Sabancı University
in partial fulfillment of the requirements for the degree of
Master of Science

Sabancı University

August, 2013

An Ontology System for Rehabilitation Robotics

APPROVED BY:

Assoc. Prof. Dr. Esra Erdem
(Thesis Advisor)

Assoc. Prof. Dr. Volkan Patoglu
(Thesis Advisor)

Prof. Dr. Kemal İnan

Assoc. Prof. Dr. Albert Levi

Asst. Prof. Dr. Hüsnü Yenigün

DATE OF APPROVAL:

© Zeynep Dođmuş, 2013
All Rights Reserved

An Ontology System for Rehabilitation Robotics

Zeynep Dođmuş

CS, Master of Science, 2013

Thesis Supervisors: Assoc. Prof. Esra Erdem, Assoc. Prof. Volkan Patoglu

Keywords: rehabilitation robotics, ontology development, query answering

Abstract

Representing the available information about rehabilitation robots in a structured form, like ontologies, facilitates access to various kinds of information about the existing robots, and thus it is important both from the point of view of rehabilitation robotics and from the point of view of physical medicine. Rehabilitation robotics researchers can learn various properties of the existing robots and access to the related publications to further improve the state-of-the-art. Physical medicine experts can find information about rehabilitation robots and related publications (possibly including results of clinical studies) to better identify the right robot for a particular therapy or patient population. Therefore, considering also the advantages of ontologies and ontological reasoning, such as interoperability of various heterogeneous knowledge resources (e.g., patient databases or disease ontologies), such an ontology provides the underlying mechanisms for translational physical medicine, from bench-to-bed and back, and personalized rehabilitation robotics.

In this thesis, we introduce the first formal rehabilitation robotics ontology, called REHABROBO-ONTO, to represent information about rehabilitation robots and their properties. We have designed and developed REHABROBO-ONTO in OWL, collaborating with experts in robotics and in physical medicine. We have also built a software (called REHABROBO-QUERY) with an easy-to-use intelligent user-interface that allows robot designers to add/modify information about their rehabilitation robots to/from REHABROBO-ONTO. With REHABROBO-QUERY, the experts do not need to know about the logic-based ontology languages, or have experience with the existing Semantic Web technologies or logic-based ontological reasoners. REHABROBO-QUERY is made available on the cloud, utilizing Amazon Web services, so that rehabilitation robot designers around the world can

add/modify information about their robots in REHABROBO-ONTO, and rehabilitation robot designers and physical medicine experts around the world can access the knowledge in REHABROBO-ONTO by means of questions about robots, in natural language, with the guide of the intelligent user-interface of REHABROBO-QUERY.

The ontology system consisting of REHABROBO-ONTO and REHABROBO-QUERY is of great value to robot designers as well as physical therapists and medical doctors. On the one hand, robot designers can access various properties of the existing robots and to the related publications to further improve the state-of-the-art. On the other hand, physical therapists and medical doctors can utilize the ontology to compare rehabilitation robots and to identify the ones that serve best to cover their needs, or to evaluate the effects of various devices for targeted joint exercises on patients with specific disorders.

Rehabilitasyon Robotları için Bir Ontoloji Sistemi

Zeynep Dođmuş

CS, Yüksek Lisans Tezi, 2013

Tez Danışmanları: Doç. Dr. Esra Erdem, Doç. Dr. Volkan Patoglu

Anahtar Kelimeler: rehabilitasyon robotları, ontoloji geliştirme, sorgu
cevaplama

Özet

Rehabilitasyon robotları ile ilgili bilgilerin yapısal olarak; örneğin ontolojiler ile gösterimi, mevcut robotlar hakkında bilgilere erişime olanak vermekte, hem rehabilitasyon robotları hem de fizik tedavi açısından önem arz etmektedir. Rehabilitasyon robot bilimi araştırmacıları mevcut robotların çeşitli özelliklerini öğrenebilir ve ilgili yayınlara erişerek mevcut teknikleri geliştirebilir. Fizik tedavi uzmanları ise rehabilitasyon robotlarıyla ilgili bilgilere ve klinik çalışmaların sonuçlarını da içerebilen yayınlara erişebilir, bu sayede belirli bir terapi veya hasta popülasyonu için uygun robotu belirleyebilir. Bundan dolayı, ontolojilerin ve ontolojik akıl yürütmenin çeşitli heterojen bilgi kaynakları (hasta veri tabanları veya hastalık ontolojileri) için birlikte işlerlik gibi avantajları da ele alındığında, böyle bir ontoloji, dönüşümsel fizik tedavi ve kişiselleştirilmiş rehabilitasyon robot bilimi için temel yöntemler sağlamaktadır.

Bu tezde, rehabilitasyon robotları ve robotların özellikleri hakkında bilgileri gösterebilmek amacıyla yaratılan ilk biçimsel rehabilitasyon robotları ontolojisi (REHABROBO-ONTO) sunulmaktadır. REHABROBO-ONTO, robotik ve fizik tedavi uzmanları ile işbirliği içerisinde, OWL ontoloji dili ile tasarlanmış ve geliştirilmiştir. Aynı zamanda, REHABROBO-QUERY adında, kullanımı kolay, akıllı bir kullanıcı arayüzüne sahip bir yazılım geliştirilmiştir. REHABROBO-QUERY, robot tasarımcılarının REHABROBO-ONTO'ya rehabilitasyon robotları hakkında bilgi ekleyebilmelerine ve bu bilgileri güncelleyebilmelerine olanak vermektedir. REHABROBO-QUERY ile, uzmanların mantık tabanlı ontoloji dillerini bilmelerine, anlamsal ağ teknolojileri ya da mantık tabanlı ontolojik akıl yürütücüleri ile ilgili deneyim sahibi olmalarına

gerek kalmamaktadır. REHABROBO-QUERY, Amazon Web hizmetleri kullanılarak internet üzerinden erişilebilir hale getirilmiştir. Böylece, dünya çapındaki rehabilitasyon robotu tasarımcıları REHABROBO-ONTO'ya robotları hakkında bilgi ekleyebilir ve bu bilgileri güncelleyebilirler, ve dünya çapındaki rehabilitasyon robotu tasarımcıları ve fizik tedavi uzmanları REHABROBO-ONTO içerisindeki bilgiye robotlar hakkında doğal dilde sorular aracılığıyla, REHABROBO-QUERY'nin rehberliği ile erişebilirler.

REHABROBO-ONTO ve REHABROBO-QUERY'den oluşan ontoloji sistemi, robot tasarımcıları için olduğu kadar fizyoterapistler ve tıp doktorları için de büyük öneme sahiptir. Bir taraftan, robot tasarımcıları mevcut robotların çeşitli özelliklerine ve ilgili yayınlara erişebilir, mevcut teknikleri geliştirebilirler. Diğer taraftan, fizyoterapistler ve tıp doktorları rehabilitasyon robotlarını mukayese etmek ve ihtiyaçları için en uygun olan robotları saptamak, veya belirli hastalıklara sahip hastalar üzerindeki hedeflenmiş eklem egzersizlerini gerçekleştiren çeşitli cihazların etkilerini değerlendirmek için bu ontolojiyi kullanabilirler.

Acknowledgements

First and foremost I would like to thank my thesis advisors Assoc. Prof. Esra Erdem and Assoc. Prof. Volkan Patođlu for their invaluable guidance and support throughout this thesis. I would also like to express my regards to the jury members, Prof. Kemal İnan, Assoc. Prof. Albert Levi, and Asst. Prof. Hüsni Yenigün for their time and feedback.

I would like to thank my friends and colleagues, including Zeynep Gözen Sarıatur, Giray Havur, Ahmetcan Erdoğan, Gizem Gezici, Peter Schüller and Erdi Aker, for useful discussions. I am also grateful to Ayça Tekiner, İpek Özdemir and Bahriye Karakaş for their friendship and for making the laboratory enjoyable.

My special thanks are due to Ozan Tokatlı, for being the source of inspiration and motivation throughout my time at Sabancı University. I am grateful to him for enriching my life, and empowering me to complete this thesis with his encouragement and support. My graduate years would not be same without him.

Finally, I am pleased to thank my dearest parents Cangül and Cebrail Dođmuş, for their unconditional love and infinite support throughout my life.

This thesis was partially supported by the Scientific and Technological Research Council of Turkey (TÜBİTAK) under grants 111E116 and 111M186, and by Sabancı University under IRP Grant IACF09-00643.

Contents

1	Introduction	1
1.1	Outline	4
2	Preliminaries	5
2.1	Ontologies	5
2.2	Representing Ontologies in RDF(S)	8
2.3	Query Answering over RDF(S) Graphs	12
2.4	Representing Ontologies in Description Logics	19
2.5	RDF(S) vs. DLs	23
2.6	Web Ontology Language (OWL)	25
2.7	Pellet: A DL Reasoner	26
2.8	PROTÉGÉ: An Ontology Editor	29
2.9	Jena: An Ontology Management Framework	29
2.10	Discussion	30
3	REHABROBO-ONTO	33
3.1	Design of REHABROBO-ONTO	34
3.2	Development of REHABROBO-ONTO	39
3.3	Maintaining REHABROBO-ONTO	42
3.4	Overall System Architecture	52
3.5	REHABROBO-ONTO on the Cloud	57
4	REHABROBO-QUERY	60
4.1	REHABROBO-CNL	63
4.2	Transforming a Query in REHABROBO-CNL to a SPARQL Query	70

4.3	Answering Queries Using Pellet	81
4.4	Intelligent User Interface for REHABROBO-QUERY	84
5	Interoperability of REHABROBO-ONTO	89
5.1	Integration with FMA	90
5.2	Integration with DO	97
5.3	On Extending REHABROBO-QUERY	102
5.4	Integration of REHABROBO-ONTO with Patient Data	105
6	Related Work	113
6.1	Ontologies and Robots	113
6.1.1	Ontologies for Robots	113
6.1.2	Ontologies about Robots	114
6.1.3	Standardization of Rehabilitation Robots	115
6.2	Ontology Systems that Support Natural Language Queries . .	116
7	Conclusion	119
	Appendix A Transformations of Sample Queries	122
	Appendix B Example Queries over FMA and REHABROBO-ONTO	135
	Appendix C Example Queries over DO and REHABROBO-ONTO	139

List of Figures

2.1	Example Ontology.	6
2.2	Class hierarchy in an RDF(S) knowledge base.	11
2.3	Specifying domain and range of a relation in an RDF(S) knowledge base.	12
2.4	Example Knowledge Base in DL.	21
3.1	REHABROBO-ONTO with main classes.	35
3.2	Hierarchy of lower extremity rehabilitation robots.	37
3.3	Hierarchy of lower extremity joint movements targeted by rehabilitation robots.	37
3.4	Hierarchy of Assessments	38
3.5	Declaring disjoint classes in PROTÉGÉ.	40
3.6	Declaring ownedBy relation in PROTÉGÉ.	41
3.7	Restricting range of has_Year in PROTÉGÉ.	41
3.8	Adding a new robot to REHABROBO-ONTO.	44
3.9	Warning for an existing robot with available options.	45
3.10	Adding to REHABROBO-ONTO general information about the rehabilitation robot ASSISTON-WRIST.	45
3.11	Adding to REHABROBO-ONTO kinematic properties of the rehabilitation robot ASSISTON-WRIST.	46
3.12	Adding to REHABROBO-ONTO targeted joints of the rehabilitation robot ASSISTON-WRIST.	46
3.13	Adding to REHABROBO-ONTO a targeted joint (wrist) of the rehabilitation robot ASSISTON-WRIST, from a pop-up window.	47
3.14	Adding to REHABROBO-ONTO power transmissions of the rehabilitation robot ASSISTON-WRIST.	47

3.15	Adding to REHABROBO-ONTO power transmissions of a targeted joint (wrist), from a pop-up window.	48
3.16	Adding to REHABROBO-ONTO assessments of the rehabilitation robot ASSISTON-WRIST.	48
3.17	Adding to REHABROBO-ONTO related publications of the rehabilitation robot ASSISTON-WRIST.	49
3.18	Robots that are owned/maintained by a particular user.	51
3.19	An overview of REHABROBO-QUERY.	52
3.20	Work flow of addition.	57
3.21	Data flow of addition.	58
3.22	Main work flow, including login.	59
4.1	Tree representation of the sample query.	71
4.2	Consistency check for REHABROBO-ONTO.	84
4.3	Explanation generation with PELLET.	85
4.4	Constructing Q1 (1).	86
4.5	Constructing Q1 (2).	87
4.6	Answer to Q1.	88
5.1	Hierarchy and integration of concepts for FMA1.	93
5.2	Importing ontologies and adding SWRL rules to answer FMA1.	94
5.3	Answer to FMA1.	95
5.4	Hierarchy and integration of concepts for FMA2.	96
5.5	SWRL rules to answer FMA2.	96
5.6	Defining a new concept with SWRL.	97
5.7	Answer to FMA2.	98
5.8	Hierarchy and integration of concepts for FMA3.	99
5.9	SWRL rules to answer FMA3.	99

5.10	Answer to FMA3.	100
5.11	Hierarchy and integration of concepts for DO1.	101
5.12	SWRL rule to answer DO1.	101
5.13	Answer to DO1.	102
5.14	Hierarchy and integration of concepts for DO2.	103
5.15	SWRL rule to answer DO2.	103
5.16	Answer to DO2.	104
5.17	Hierarchy and integration of concepts for DO3.	105
5.18	SWRL rule to answer DO3.	105
5.19	Answer to DO3.	106
5.20	Patient ontology with main classes.	107
5.21	SWRL rules over patient ontology and REHABROBO-ONTO.	108
A.1	Transformation for Q1	123
A.2	Transformation for Q2	124
A.3	Transformation for Q3	125
A.4	Transformation for Q4	126
A.5	Transformation for Q5	127
A.6	Transformation for Q6	128
A.7	Transformation for Q7	129
A.8	Transformation for Q8	130
A.9	Transformation for Q9	131
A.10	Transformation for Q10	132
A.11	Transformation for Q11	133
A.12	Transformation for Q12	134

List of Tables

2.1	Reasoners	8
2.2	Ontology Management Frameworks	9
2.3	A fragment of the SPARQL grammar.	13
2.4	Extensions to \mathcal{ALC}	23
2.5	DL Reasoners	27
4.1	Sample Queries in REHABROBO-CNL	61
4.2	The Grammar of REHABROBO-CNL	64
4.3	The Ontology Functions	65
4.4	Verbs that can occur after the nouns	66
4.5	Types that can occur after the verbs	67
4.6	Instances that can occur after the types	67
4.7	Nouns that can occur after the types	68
4.8	Values that can occur after the nouns	69
4.9	DL to SPARQL Transformation Examples	80
4.10	Answers for the query types	88

Chapter 1

1 Introduction

As the number of rehabilitation robots increase, the information about them also increases, but most of the time in unstructured forms (e.g., as text in publications), which make it harder to access the requested knowledge (e.g., the flexion/extension range of motion (RoM) of ASSISTON-SE [76]) and thus automatically reason about it (e.g., finding the rehabilitation robots that target shoulder movements and also have at least 210° RoM for the flexion/extension movements of the shoulder).

Also, due to interdisciplinary nature of rehabilitation robotics, sometimes requested knowledge requires integration of further knowledge from related disciplines (e.g., physical medicine). Consider, for instance, finding rehabilitation robots that can be used to treat a patient with rotator cuff lesions. For that, we need to know that rotator cuffs are muscle units for moving the shoulder, and that, for patients with rotator cuff lesions, abduction and flexion movements of the shoulder should not have more than 90° RoM. Then we can look for relevant rehabilitation robots.

On the other hand, there are efforts, e.g., by European Network on Robotics for Neurorehabilitation¹, for standardizing terminology as well as assessment measures for rehabilitation robots. Given the growing number of

¹<http://www.rehabilitationrobotics.eu/>

different approaches introduced by various research groups and the variability of results available, the development of such a standardization would be a critical step forward in the field, helping robotic rehabilitation technology become widely understood and accepted as a useful tool.

Motivated by these challenges and efforts, we have designed and developed the first formal rehabilitation robotics ontology, called REHABROBO-ONTO. REHABROBO-ONTO represents knowledge about rehabilitation robotics in a structured form, and allows automated reasoning about this knowledge. It is open-source and available on the cloud via Amazon Web Services (in particular, Amazon Elastic Compute Cloud)² so that every rehabilitation robotics researcher can easily add information about his/her robot to it, and every rehabilitation robotics researcher and every physical medicine expert can access information about all available rehabilitation robots. REHABROBO-ONTO has been designed in a way that enables integration with other medical ontologies, such as ontologies that capture rehabilitation protocols, patient data and disorder details. Considering the standards of World Wide Web Consortium (W3C), REHABROBO-ONTO is represented in OWL (Web Ontology Language) [4, 34].

To facilitate such modifications and uses of REHABROBO-ONTO, we have also developed a Web-based software (called REHABROBO-QUERY)³ with an intelligent user-interface. In this way, experts do not need to know the underlying logic-based representation languages of ontologies, like OWL, or Semantic Web technologies, for information entry, retrieval and modification. REHABROBO-QUERY also utilizes Amazon Web Services for cloud computing.

²<http://aws.amazon.com/ec2/>

³http://hmi.sabanciuniv.edu/?page_id=781

Since REHABROBO-ONTO is publicly available, both rehabilitation robotics experts and physical medicine experts can ask queries over it. To query over ontologies, queries should be represented using formal query languages, such as SPARQL. However, the experts, who can benefit from this ontology by means of asking queries, may lack the knowledge of such formal query languages. Thus, we need to enable users to represent queries in a simpler language. For that, we have developed a controlled natural language for rehabilitation robots, called REHABROBO-CNL. In addition, we have developed an intelligent user interface (in REHABROBO-QUERY) that allows experts to enter natural language queries about the existing robots and get the answers in an understandable form, without having to know about the logical formalism of the ontology or the formalism to represent queries. Furthermore, the experts do not have to know about the use of the technologies for computing answers to their questions about rehabilitation robots. By means of such queries over REHABROBO-ONTO, right rehabilitation robots for a particular patient or a physical therapy can be found or designed; this further paves the way for translational physical medicine (from bench-to-bed and back) and personalized physical medicine.

The ontology system consisting of REHABROBO-ONTO and REHABROBO-QUERY is of great value to robot designers as well as physical therapists and medical doctors. On the one hand, robot designers can benefit from the system, for instance, to identify robotic devices targeting similar therapeutic exercises or to determine systems using a particular kind of actuation-transmission pair to achieve a range of motion that exceeds some threshold. Availability of such information may help inspire new designs or may lead to a better decision making process. The ontology can also be utilized to group

similar robots by quantifiable characteristics and to establish benchmarks for system comparisons. Overall, an ontology designed to specifically meet the expectations of the overall rehabilitation robotics effort has the potential to become an indispensable tool that helps in the development, testing, and certification of rehabilitation robots. On the other hand, physical therapists and medical doctors can utilize the ontology to compare rehabilitation robots and to identify the ones that serve best to cover their needs, or to evaluate the effects of various devices for targeted joint exercises on patients with specific disorders.

It is important to emphasize that the ontology REHABROBO-ONTO and the tool REHABROBO-QUERY introduced in this thesis have been developed to initiate efforts in utilizing ontological technologies for the field of rehabilitation robotics. Therefore, by making REHABROBO-ONTO available open-source via REHABROBO-QUERY, it is our intention to continually update and enhance capabilities of these tools according to the feedback provided by the community.

1.1 Outline

The rest of the thesis is organized as follows. In Chapter 2, we introduce some preliminaries for this thesis. We briefly introduce ontologies, RDF(S) and Description Logic (DL). Next, we present the first formal rehabilitation robotics ontology, REHABROBO-ONTO in Chapter 3. Then, in Chapter 4, we describe the software system REHABROBO-QUERY. After discussing the interoperability of REHABROBO-ONTO in Chapter 5, we provide in Chapter 6 the related work. We conclude the thesis in Chapter 7 with a summary of our contributions and a discussion of possible future work.

Chapter 2

2 Preliminaries

In this chapter, we introduce some preliminaries for this thesis. We first give an overview on ontologies. Then, we briefly introduce Resource Description Framework (RDF) and RDF Schema (RDFS), with examples. After that we give a short overview on SPARQL, a query language for RDFS knowledge bases. Next, we describe Description Logics by providing a basic knowledge of a simple Description Logic, and covering some reasoning tasks and reasoners. Finally, we briefly introduce the semantic web technologies that we utilize in this thesis.

2.1 Ontologies

Ontologies (like databases) are formal frameworks for representing knowledge in a structured form, to aid access to relevant parts of the knowledge and automate reasoning over it. An ontology can be viewed as a graph where nodes denote concepts (e.g., rehabilitation robots, joint movements) and the edges between the nodes denote relations between the corresponding concepts. For instance, as shown in Figure 2.1, an edge from a node that denotes “Upper Extremity Rehabilitation Robots” to a node that denotes “Rehabilitation Robots” may characterize the “is-a” hierarchy relation; whereas an edge from

a node that denotes “Rehabilitation Robots” to a node that denotes “Joint Movements” may characterize “targets” relation.

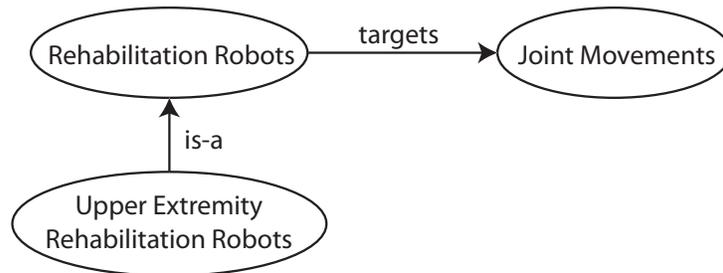


Figure 2.1: Example Ontology.

Due to their flexible graph-like structure, ontologies (unlike databases) allow representation of incomplete knowledge, can easily be extended by new information (e.g., with new sorts/features of rehabilitation robots).

Due to their formal representations, ontologies developed by different parties at different locations can be integrated, and reasoning (e.g., query answering) can be automated over concepts and their relations represented in these ontologies. Therefore, it is not surprising that more and more knowledge-intensive systems (including Semantic Web [10] that is planned to provide automated services to Web by giving meaning to concepts) rely on ontologies to enable content-based access, interoperability, and communication across the Web.

There are several formalisms to represent an ontology. One of them is Resource Description Framework (RDF). It relies on a data model of a directed labeled graph, called RDF graph. Each edge in an RDF graph corresponds to an RDF triple: $\langle \textit{subject}, \textit{predicate}, \textit{object} \rangle$. For instance, to represent “targets” relation in an RDF graph, we may add an edge with label “targets”, from “Rehabilitation Robots” to “Joint Movements”. This edge corresponds

to a triple whose subject is “Rehabilitation Robots”, predicate is “targets” and object is “Joint Movements”. Therefore, it is possible to describe an RDF graph by its edges, which results in a set of triples.

Another way to represent an ontology is using one of the languages in the family of Description Logics (DLs). A DL knowledge base consists of logical statements. Logical statements include concept descriptions, which are built using atomic concepts and relations, with the use of logical constructors (e.g., Boolean, existential restriction and value restriction constructors) as well as assertions. For instance, we may describe `ShoulderRobots` concept by adding the following statement to our DL knowledge base:

```
ShoulderRobots  $\equiv$   
RehabilitationRobots  $\sqcap$   $\exists$ targets.ShoulderMovements
```

We may also assert that a rehabilitation robot whose name is `ASSISTON-ARM` is a shoulder robot. Then, it can be inferred that `ASSISTON-ARM` is a rehabilitation robot which targets some shoulder movements.

There are several ontology editors available to represent and manipulate ontologies. One of them is `PROTÉGÉ` [28], which supports developing ontologies in different formats. It is easy to use since it provides a graphical user interface for designing ontologies. `PROTÉGÉ` supports DL reasoners such as `PELLET` [67], `HERMIT` [66] and `FACT++` [71] to check the ontology for consistency and to answer queries. With these reasoners, we can ensure that our ontologies are meaningful and correct.

To query and reason over ontologies, many query engines and reasoners have been developed. The reasoners are able to infer implicit knowledge from a set of asserted facts and axioms. With reasoners, we can also ensure that the ontologies that we developed are logically consistent, so that we

can query over our ontology. Some of the reasoners are shown in Table 2.1. There are also some frameworks that can store and manipulate ontologies while providing support for reasoners. They are listed in Table 2.2.

Table 2.1: Reasoners

Reasoner	Language support
BaseVISor [49]	RDF, OWL 2 RL
BOSSAM [36]	RDF, OWL DL
FACT++ [71]	OWL DL, OWL 2 DL
HERMIT [66]	OWL 2 DL
HOOLET ⁴	OWL DL
KAON2 [51]	OWL DL, SWRL, F-Logic
PELLET [67]	OWL DL, OWL 2 DL
RACERPRO [30]	OWL DL

In this thesis, we use the ontology language OWL 2 DL, the reasoner PELLET, and the framework Jena. In the following sections, we give further details about them.

2.2 Representing Ontologies in RDF(S)

Resource Description Framework (RDF) relies on a data model of a directed labeled graph, called RDF graph. Each edge in an RDF graph corresponds to an RDF triple:

subject predicate object.

For instance, to represent “targets” relation in an RDF graph, we may add an edge with label “targets”, from “Rehabilitation Robots” to “Joint Movements”. This edge corresponds to a triple whose subject is “Rehabilitation Robots”, predicate is “targets” and object is “Joint Movements”. Therefore,

⁴<http://owl.man.ac.uk/hoolet/>

it is possible to describe an RDF graph by its edges, which results in a set of triples.

Nodes in an RDF graph can be URIs, literals or blank nodes. A Uniform Resource Identifier (URI) identifies a concept or an instance on the Web. Concepts can be regarded as sets of instances. For instance, a “Rehabilitation Robot” concept denotes a set of rehabilitation robots, whereas the instance “AssistOn-Wrist” denotes an individual rehabilitation robot. There may exist multiple URIs to identify the same concept or instance. A URI does not have to physically correspond to a Web address. For instance, the rehabilitation robot ASSISTON-WRIST may be identified using the URI:

`http://en.wikipedia.org/wiki/AssistOn-Wrist`

It may also be possible to identify ASSISTON-WRIST using the URI:

`http://www.myresources.com/AssistOn-Wrist`

Table 2.2: Ontology Management Frameworks

Framework	Language Support	Reasoner Support*
AllegroGraph ⁵	RDF, RDFS, OWL (limited)	Default
Euler ⁶	OWL, RDF	Default
Jena [50]	RDF, RDF(S), OWL	Default, PELLET
OWL API [32]	OWL 2, RDF (limited)	Default, HERMIT, PELLET, FACT++
Redland [8]	RDF	Default
Sesame [2]	RDF, OWL (limited)	Default
Virtuoso [22]	RDF, OWL (limited)	Default

*If the framework provides its own generic reasoner, it is stated as “default”.

Literals represent values, such as 15 or “Sabancı University” which have Integer and String types, respectively. A blank node represents a concept or instance having an unknown URI.

Consider a knowledge base about rehabilitation robots. With RDF, we

⁵<http://www.franz.com/agraph/allegrograph/>

⁶<http://eulerssharp.sourceforge.net/>

want to specify rehabilitation robots, targeted joint movements, and their relations. Assume that the concepts and instances use the same namespace for their URIs and we abbreviate it using `rkb`, that stands for rehabilitation robot knowledge base at `http://rehabrobotkb.com/`. With the relation `targets`, we can say that `AssistOn-Wrist` targets `WristFlexion/Extension` joint movement:

```
rkb:AssistOn-Wrist rkb:targets rkb:WristFlexion/Extension.
```

Although we intuitively know that `ASSISTON-WRIST` is a rehabilitation robot and `wrist flexion/extension` is a joint movement, we cannot represent them with RDF. In other words, we can represent instances and their relations with RDF but we cannot state that one instance is in the set of a concept. For that, we need terminological/schema knowledge.

RDF Schema (RDFS or RDF(S)) is a W3C RDF recommendation which provides generic constructs to represent schema knowledge. An RDF(S) document is also a well-formed RDF document because RDF(S) statements are simply RDF triples. Therefore, the tools that support RDF can read and process RDF(S) documents. According to the formal ontology terminology in RDF(S), concepts are called classes. With RDF(S), it is possible to represent classes and specify instances of classes using the predicates `rdfs:Class` and `rdf:type`. We can characterize `RehabilitationRobots` as a class and `AssistOn-Wrist` as its instance with the following triples:

```
rkb:RehabilitationRobots rdf:type rdfs:Class.
```

```
rkb:AssistOn-Wrist rdf:type rkb:RehabilitationRobots.
```

Assume that we have one more class, `WristRobots`, in our knowledge base. Since `AssistOn-Wrist` is both a wrist robot and a rehabilitation robot, we have to add one more triple indicating that `AssistOn-Wrist` is a wrist

robot as well:

```
rkb:WristRobots rdf:type rdfs:Class.
```

```
rkb:AssistOn-Wrist rdf:type rkb:WristRobots.
```

RDF(S) prevents these kinds of repetitions by supporting class hierarchies.

With the predicate `rdfs:subClassOf` (Figure 2.2), we can say that every wrist robot is also a rehabilitation robot:

```
rkb:WristRobots rdfs:subClassOf rkb:RehabilitationRobots.
```

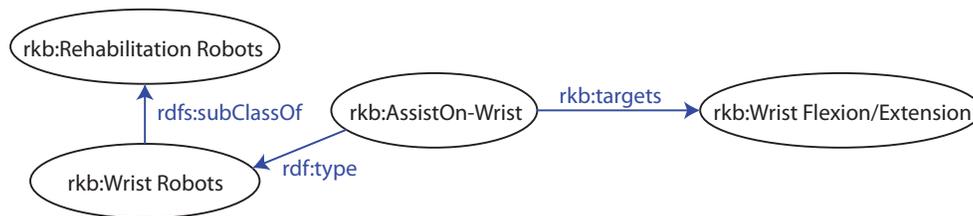


Figure 2.2: Class hierarchy in an RDF(S) knowledge base.

It is also possible to specify hierarchies for relations in the ontology.

There is no need to declare that `AssistOn-Wrist` is a rehabilitation robot and `WristFlexion/Extension` is a joint movement if the domain and the range of `targets` relation are known. RDF(S) provides predicates `rdfs:domain` and `rdfs:range` (Figure 2.3) to represent the domain and the range of a relation:

```
rkb:targets rdfs:domain rkb:RehabilitationRobots.
```

```
rkb:targets rdfs:range rkb:JointMovements.
```

Our final knowledge base consists of the following triples:

```
rkb:targets rdfs:domain rkb:RehabilitationRobots.
```

```
rkb:targets rdfs:range rkb:JointMovements.
```

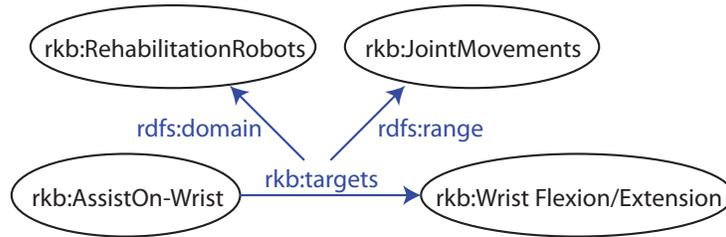


Figure 2.3: Specifying domain and range of a relation in an RDF(S) knowledge base.

```

rkb:AssistOn-Wrist rkb:targets rkb:WristFlexion/Extension.
rkb:AssistOn-Wrist rdf:type rkb:WristRobots.
rkb:WristRobots rdfs:subClassOf rkb:RehabilitationRobots.
  
```

2.3 Query Answering over RDF(S) Graphs

The SPARQL Protocol and RDF Query Language (SPARQL) [58] is a query language that can be used for querying ontologies in RDF(S). A SPARQL query starts with a `SELECT` clause to specify the variables to appear in the result, and continues with a `WHERE` clause that specifies what to query for in an RDF(S) graph.

We introduce the syntax of SPARQL briefly by presenting a fragment of the SPARQL grammar in Table 2.3. Full syntax of SPARQL can be found in [58]. The queries in SPARQL are formed as follows: A SPARQL query starts with an optional prefix declaration (`PrefixDecl`) which abbreviates the namespace of a knowledge base. Then, in `SelectQuery`, we specify the variables that we want to see in the results. It is possible to eliminate duplicate results with `DISTINCT` keyword. After that, the graph patterns (abbreviated as `GP`) that need to be matched in the RDF(S) graph are spec-

ified in `WhereClause`. They are represented with triples (`TriplesBlock`), or with expressions (`GPNotTriples`) constructed by combining triples using the operators `UNION` or `FILTER`.

Table 2.3: A fragment of the SPARQL grammar.

<code>Query::=</code>	<code>PrefixDecl* SelectQuery</code>
<code>PrefixDecl::=</code>	<code>PREFIX PrefixName : < Namespace ></code>
<code>SelectQuery::=</code>	<code>SELECT (DISTINCT)? (Var+ '*') WhereClause</code>
<code>WhereClause::=</code>	<code>WHERE? GroupGP</code>
<code>GroupGP::=</code>	<code>'{ TriplesBlock? ((UnionGP Filter) .? TriplesBlock?)* }'</code>
<code>Filter::=</code>	<code>FILTER? Constraint</code>
<code>Constraint::=</code>	<code>BrackettedExpression NotExistsFunc</code>
<code>NotExistsFunc::=</code>	<code>NOT EXISTS GroupGP</code>

We now describe the syntax and semantics of the simple graph patterns. Let I , L , and V be disjoint infinite sets, denoting URIs, literals, and variables, respectively. Then, the tuple $t \in (I \cup L \cup V) \times (I \cup V) \times (I \cup L \cup V)$ is called a triple pattern, and $var(t)$ denotes the set of variables that occur in t . A simple graph pattern is a finite set of triple patterns; then, $var(P) = \bigcup_{t \in P} var(t)$ is the set of variables that occur in a simple graph pattern P .

Let $\mu : V \rightarrow I \cup L$ be a partial function that maps variables in V to URIs and literals. The set of variables that are mapped to URIs and literals via μ is called the domain of μ , and denoted by $dom(\mu)$. Mapping of the variables in a triple pattern t with μ results in a new triple $\mu(t)$, where $var(t) \subseteq dom(\mu)$. Thus, $\mu(P)$ is the resulting set of triples obtained by applying μ on a simple graph pattern P , where $dom(\mu)$ consists of the variables in $var(P)$.

It is possible to obtain complex expressions from a number of simple graph patterns, and represent them with SPARQL. In this thesis, we consider the binary operators `And`, `Union` and `Filter` to construct such expressions. We

use a dot sign as a substitute for **And**. If the expressions $(G_1 \cdot G_2)$ and $(G_1 \text{ Union } G_2)$ are constructed with simple graph patterns G_1 and G_2 , then these expressions are graph patterns. Similarly, if the expression $(G \text{ Filter } C)$ is constructed with a simple graph pattern G and a value constraint C , then this expression is also a graph pattern. We consider the value constraints that are constructed using an element of the set V , an equality or inequality symbol, and a constant.

To give the semantics of the binary operators **And**, **Union** and **Filter**, we first define compatible mappings and then define required algebra operators. Two mappings μ_1, μ_2 are compatibles if $\mu_1(v) = \mu_2(v)$ for every variable $v \in \text{dom}(\mu_1) \cap \text{dom}(\mu_2)$. Therefore, compatible mappings must agree on their shared variables. For instance, mappings of disjoint domains are always compatible.

Let M_1 and M_2 be sets of mappings. We define algebra operator **Join**, that extends mappings in M_1 with the mappings in M_2 , as follows.

$$M_1 \bowtie M_2 = \{\mu_1 \cup \mu_2 \mid \mu_1 \in M_1 \text{ and } \mu_2 \in M_2 \text{ are compatible mappings}\}$$

We define algebra operator **Union**, that gets the union of the mappings in M_1 and the mappings in M_2 , as follows.

$$M_1 \cup M_2 = \{\mu \mid \mu \in M_1 \text{ or } \mu \in M_2\}$$

The binary operators **And**, **Union** and **Filter** are evaluated as follows. Let G be an RDF graph, C a value constraint, and P_1, P_2 graph patterns. Then we recursively define

$$\llbracket P_1 \cdot P_2 \rrbracket_G = \llbracket P_1 \rrbracket_G \bowtie \llbracket P_2 \rrbracket_G$$

$$\llbracket P_1 \text{ UNION } P_2 \rrbracket_G = \llbracket P_1 \rrbracket_G \cup \llbracket P_2 \rrbracket_G$$

$$\llbracket P \text{ FILTER } C \rrbracket_G = \{\mu \in \llbracket P \rrbracket_G \mid \mu \models C\}$$

where the base case is the simple graph pattern evaluation, described

above.

We also consider NOT EXISTS in SPARQL queries. It is evaluated in the satisfiability check of a filter expression, and defined as follows. Let G be an RDF graph, μ a mapping, and P a graph pattern. Then, $\mu \models \text{NOT EXISTS}(P)$ iff $\llbracket \mu(P) \rrbracket_G$ is an empty set.

Further information about the semantics of SPARQL, and the complexity of evaluating graph patterns that can contain several operators, can be found in [17] and in [57].

We now give some examples of queries over the knowledge base in Section 2.2 and the SPARQL representations of these queries. Consider the following query.

“What are the robots that target WristFlexion/Extension?”

The SPARQL query that corresponds to this query consists of PREFIX, SELECT and WHERE parts. In PREFIX part, we declare the namespace of our ontology. We assumed that the namespace of rehabilitation robotics ontology is `http://rehabrobotkb.com/`. In SELECT part, we specify the variable “robot” to be appeared in the result. Variables are characterized with the special symbol ‘?’ at the beginning of their names. In WHERE part, we specify the condition that must be met: The robot that we want must target WristFlexion/Extension movement. The SPARQL query is as follows.

```
PREFIX rkb: <http://rehabrobotkb.com/>
SELECT DISTINCT ?robot
WHERE {
    ?robot rkb:targets rkb:WristFlexion/Extension.
}
```

The first step to compute a SPARQL query involves translations into SPARQL algebra. Each expression that can be constructed with SPARQL has a direct transformation to SPARQL algebra. To express the (possibly complex) graph patterns in the queries, SPARQL algebra uses some operators such as `BGP`, `Join`, or `Union`, corresponding to basic graph pattern, conjunctions and disjunctions, respectively. Since our query consists of one triple pattern, its transformation into SPARQL algebra is as follows.

```
BGP(?robot rkb:targets rkb:WristFlexion/Extension.)
```

With the mapping μ introduced earlier in this chapter, it is possible to assign variables to URIs or literals in the RDF(S) graph. Then, a solution for a basic graph pattern expression is found by applying the partial function μ . This is the basic evaluation method to find an answer to a SPARQL query. With this method, it is possible to answer more complex queries. For instance, an expression with the operator `Union` is evaluated by applying μ to each expression in `Union`, and then getting the disjunction of the solutions.

The answer to this query is:

robot
http://rehabrobotkb.com/AssistOn-Wrist

We can represent the answers to SPARQL queries using a table. Here, each row of this table denotes a mapping found by applying μ over the translated SPARQL algebra expression.

Another query example is as follows.

“What are the movements that are targeted by some wrist robots?”

SPARQL representation of this query:

```
PREFIX rkb: <http://rehabrobotkb.com/>
```

```

SELECT DISTINCT ?movement
WHERE {
    ?robot rkb:targets ?movement.
    ?robot rdf:type rkb:WristRobots.
}

```

Since the query above contains a relative clause, we need a variable `movement` to represent the targeted movements and another variable `robot` to represent the wrist robots. The answer to this query:

movement
http://rehabrobotkb.com/WristFlexion/Extension

Our knowledge base currently contains one robot targeting one wrist movement. Therefore, only movement that is retrieved is `WristFlexion/Extension`.

We can also choose to retrieve both the robots and the movements. For that, we append the `robot` variable in `SELECT` part of the query, with a space between the preceding variables:

```

PREFIX rkb: <http://rehabrobotkb.com/>
SELECT DISTINCT ?movement ?robot
WHERE {
    ?robot rkb:targets ?movement.
    ?robot rdf:type rkb:WristRobots.
}

```

The answer to this query is as follows.

There are several SPARQL query engines that can compute answers to SPARQL queries. Since matching of the variables in the `WHERE` clause

movement	robot
http://rehabrobotkb.com/WristFlexion/Extension	http://rehabrobotkb.com/AssistOn-Wrist

to URIs and literals makes up a graph pattern, the main query evaluation mechanism of SPARQL involves matching the graph pattern in the WHERE clause to the queried RDF(S) graph. However, the queried RDF(S) graph may contain implicit knowledge. For instance, in the knowledge base in Section 2.2, we do not explicitly say that `AssistOn-Wrist` is a rehabilitation robot; however, we may query about it. In order to cover such cases, the query engines use different implementation and optimization techniques for processing and evaluating queries, as well as semantic inferencing.

One implementation technique is materialization, which involves extending the RDF(S) graph with all inferences that can be computed before processing the SPARQL queries. This implementation technique is used by Jena, Virtuoso, and AllegroGraph. Another approach proposes to rewrite the queries instead of extending the RDF(S) graph. This approach is used by Sesame, sparql2sql, and GiaBATA. Third principal approach is to modify existing approaches of mapping so that matching graph patterns can be done along with complex inferencing. This approach is implemented in ARQ, which also provides optimizations on translating algebra expressions, such as introducing new operators. The order of evaluations can also be specified with ARQ, which is one of the low-level optimizations.

In order to provide logic-based inferencing along with graph-based processing of SPARQL queries, some query engines provide inference engines. For instance, the inference engine in Sesame uses forward chaining whereas the inference engine in Virtuoso uses backward chaining. The inference en-

gine in Jena utilizes both forward and backward chaining to obtain inferences from data.

2.4 Representing Ontologies in Description Logics

Description Logics (DLs) [6] are a family of logic-based formalisms for knowledge representation, that are decidable fragments of first-order logic. Ontologies represented formally in Web Ontology Language (OWL) are based on variations of Description Logics (DL). DL provides the logical formalism not only for such formal ontologies but also the Semantic Web.

DL terminology consists of concepts, roles, and objects. Objects denote entities of our world with characteristics and attributes; concepts are interpreted as sets of objects; and roles are interpreted as binary relations on objects or concepts. A DL knowledge base consists of logical statements. Logical statements include concept descriptions, which are built using atomic concepts and roles, with the use of logical constructors as well as assertions.

The statements in a DL knowledge base are divided into two groups: TBox and ABox. TBox statements contain terminological knowledge, which corresponds to the database schema. ABox statements contain assertional knowledge, which corresponds to the data in a database. Concepts and roles are defined in TBox whereas objects are defined in ABox.

In this thesis, we consider a DL called Attribute Concept Language with Complements (\mathcal{ALC}) [63]. \mathcal{ALC} includes conjunction, disjunction, negation, universal restriction and existential restriction.

Let A be an atomic concept name and r be an atomic role name. The concept descriptions C, D are formed in \mathcal{ALC} as follows:

$C, D ::= A$		
\top		(universal concept)
\perp		(ground concept)
$\neg C$		(complement of concept)
$C \sqcup D$		(union)
$C \sqcap D$		(intersection)
$\exists r.C$		(existential restriction)
$\forall r.C$		(universal restriction)

For instance, we can describe a new concept `ShoulderRobots` in terms of another concept `RehabilitationRobots` and with an existential restriction consisting of a role `targets` and a concept `ShoulderMovements`:

`ShoulderRobots` \equiv
`RehabilitationRobots` \sqcap \exists `targets`.`ShoulderMovements`

Consider a DL knowledge base that consists of the following TBox:

`Robot` \sqcap `Movement` \sqsubseteq \perp
 \exists `targets`. \top \sqsubseteq `Robot`
 \top \sqsubseteq \forall `targets`.`Movement`
`ShoulderRobot` \sqsubseteq `Robot`
`Robot` \sqsubseteq ≤ 2 `targets`

The first statement expresses disjointness of `Robot` and `Movement`. The latter two statements specify the domain and the range of `targets`. The fourth statement declares that `ShoulderRobot` is a subset of `Robot`. Finally, we assume that every robot should target at least two movements and we express it with the last statement. This knowledge base can be illustrated by a figure as in Figure 2.4.

We describe the semantics of \mathcal{ALC} in terms of interpretations. An inter-

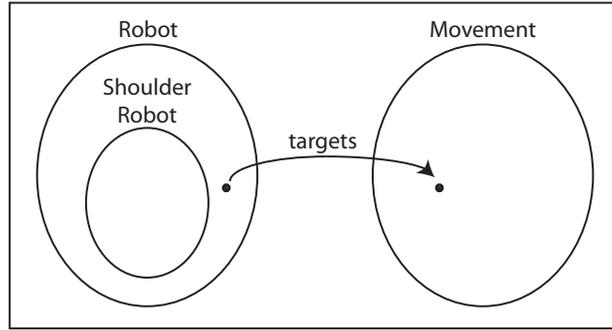


Figure 2.4: Example Knowledge Base in DL.

pretation I consists of a non-empty set Δ^I (the domain of I) and a function that maps each \mathcal{ALC} concept to a subset of Δ^I , and each atomic role to a subset of $\Delta^I \times \Delta^I$. Let C, D be \mathcal{ALC} concept descriptions and r be a role name. The extensions C^I (resp. r^I) of the concept C (resp. role r) in the interpretation function are extended by inductive definitions, as follows:

$$\begin{aligned}
\top^I &= \Delta^I \\
\perp^I &= \emptyset \\
(C \sqcap D)^I &= C^I \cap D^I \\
(C \sqcup D)^I &= C^I \cup D^I \\
\neg C^I &= \Delta^I \setminus C^I \\
(\exists r.C)^I &= \{x \in \Delta^I \mid \text{There is some } y \in \Delta^I \text{ with } \langle x, y \rangle \in r^I \text{ and } y \in C^I\} \\
(\forall r.C)^I &= \{x \in \Delta^I \mid \text{For all } y \in \Delta^I, \text{ if } \langle x, y \rangle \in r^I, \text{ then } y \in C^I\}
\end{aligned}$$

If $x \in C^I$, then x is an instance of concept C in interpretation I .

There are direct translations of \mathcal{ALC} concept descriptions into first-order formulas [6]. Consequently, a DL interpretation containing atomic concepts and atomic roles is equivalent to a first-order interpretation, containing unary and binary predicates. Since the two variable fragment of first-order logic

is decidable [29], reasoning problems such as satisfiability or entailment for \mathcal{ALC} are also decidable; there are decision procedures and algorithms for them.

Reasoning tasks allow inferring additional knowledge which we do not explicitly state in the knowledge base. Some reasoning tasks that can be performed over a DL KB include consistency checking, which checks whether the assertions and terminological knowledge in a KB have a contradiction. For instance, consider an ABox that contains assertions declaring that an object, `AssistOn-Wrist` is both a robot and a movement. If robot and movement concepts are disjoint in TBox, then this causes inconsistency, meaning that the assertions are not satisfiable with respect to the terminological axioms.

Another reasoning task is subsumption, which determines the relationships of concepts that are in a hierarchy. A subsumption algorithm checks whether a concept is subsumed by another concept, for all interpretations of these concepts. For instance, if we state in our KB that all shoulder robots target a shoulder movement, and that shoulder movement is a subconcept of joint movement; then it is inferred that all shoulder robots target a joint movement.

Instance checking is also an important reasoning task, which determines whether an object is an instance of a specific concept. For example, if in the terminology it is stated that rehabilitation robots are owned by either physical therapists or robotics researchers, and in the assertions it is stated that `AssistOn-Wrist` is a rehabilitation robot, it is owned by Jack, and Jack is not a physical therapist, then Jack is inferred to be the instance of a robotics researcher.

Other DLs

The names given to DLs reflect their expressiveness: each letter in the name of a DL describes a particular constructor. The letters and the features they express are listed in Table 2.4. Among these letters, letter \mathcal{S} is one exception because it is used to abbreviate \mathcal{ALC} , extended with role transitivity. This is done to prevent long names for the expressive DLs which provide many constructors.

Table 2.4: Extensions to \mathcal{ALC}

Letter	Stands for, example
\mathcal{S}	\mathcal{ALC} + transitive roles :If likes is transitive, and if John likes Sue and Sue likes Jane , then John likes Jane .
\mathcal{H}	role hierarchies: hasMother is the subrole of hasParent .
\mathcal{O}	nominals: Functionality of a rehabilitation robot is one of the set $\{clinic, home\}$.
\mathcal{I}	inverse roles: is targeted by is the inverse of targets .
\mathcal{N}	cardinality restrictions: A rehabilitation robot can have only one owner.
\mathcal{D}	datatypes: Degree of freedom of a rehabilitation robot must be integer.
\mathcal{Q}	qualified cardinality restrictions: A rehabilitation robot targets at least one joint movement.
\mathcal{F}	role functionality: Minimum range of motion of a rehabilitation robot.

2.5 RDF(S) vs. DLs

Both DLs and RDF(S) are formalisms that can be used to represent ontologies. With these formalisms, it is possible to represent terminological/schema knowledge. However, RDF(S) has some semantic limits compared to DL. To infer logical consequences, RDF(S) uses deduction rules. Using these

deduction rules, it is not possible to derive some information that we can infer with DL. For instance, if we say that the domain of `targets` relation is `ShoulderMovements`, and that `ShoulderMovements` is a subconcept of `Movements`, we cannot deduce that domain of `targets` is also `Movements` with RDF(S) semantics. We illustrate the semantic limits and limited modeling capabilities of RDF(S) over an example below.

Consider the knowledge base (KB) illustrated in Figure 2.4. It is not possible to express some parts of this knowledge base with RDF(S). For instance, it is not possible with RDF(S) to restrict the minimum number of movements that a robot targets. In addition, we cannot model some classes as disjoint. This is an important limitation of RDF(S) since reasoning with negation is not possible. To illustrate, the following assertions

```
Robot(AssistOn)
Movement(AssistOn)
```

would not cause an inconsistency in an RDF(S) KB. On the contrary, in a DL KB like the one above, these two assertions cause inconsistency due to the disjointness statement added to the KB. Furthermore, it is possible with DLs to declare some classes as the union/intersection/complement of other classes. For instance, we can add a statement describing upper extremity proximal robots as a new concept and as the union of elbow, lumbar spline and shoulder robots. However, this is not possible with RDF(S). These semantic limits of RDF(S) also limit the further logical consequences that can be inferred. In conclusion, DL provides powerful semantics and reasoning for further logical consequences, over RDF(S).

2.6 Web Ontology Language (OWL)

Web Ontology Language (OWL) is a W3C recommendation for ontology modeling. It is emerged as a new ontology language to balance the expressivity and reasoning services. To address the first challenge and to overcome the limited expressivity of RDF(S), OWL introduces new constructs to design an ontology. For the second challenge, OWL is based on variations of Description Logics (DL). According to the formal ontology terminology in OWL, concepts are called classes, attributes of classes are called data properties, roles are called object properties, and objects are called individuals.

OWL provides translations of its constructs into DL. Therefore, the modeling capabilities of DLs compared to RDF(S) also applies to OWL. However, to provide different levels of expressivity, OWL introduces some variants, such as OWL Lite, OWL DL or OWL 2 DL.

OWL Lite is designed to have little expressivity; thus it has no support for describing union or complement of classes, disjoint classes or value restrictions. It has little support for cardinality restrictions; only the numbers 0 and 1 can be used in these restrictions. OWL Lite corresponds to the description logic $\mathcal{SHIF}(\mathcal{D})$.

Another variant, OWL DL, provides support for the restricted features in OWL Lite, such as representing disjoint classes and combination of classes. In addition, it contains OWL Lite; that is, it supports all of the features that are fully supported in OWL Lite.

OWL DL is contained by OWL 2 DL, another variant of OWL, which is an extension of OWL DL that preserves decidability. For instance, it is possible in OWL 2 DL to restrict the range of the values allowed for some properties, and infer the inverse of a role without an assertion. For instance,

if `is targeted by` is declared to be the inverse of `targets` role, and there is an assertion stating that robot A `targets` movement B, then we know that movement B `is targeted by` robot A. OWL DL corresponds to the description logic $\mathcal{SHOIN}(\mathcal{D})$ whereas OWL 2 DL corresponds to the description logic $\mathcal{SROIQ}(\mathcal{D})$. $\mathcal{SROIQ}(\mathcal{D})$ supports qualified cardinality restrictions (denoted by ' \mathcal{Q} ') and role inclusion axioms (denoted by ' \mathcal{R} '), which encompass cardinality restrictions (denoted by ' \mathcal{N} ') and role hierarchies (denoted by ' \mathcal{H} ') supported by $\mathcal{SHOIN}(\mathcal{D})$. Therefore, OWL 2 DL contains OWL DL and extends it with new constructors.

In this thesis, we use OWL 2 DL to design our ontology about rehabilitation robots. In our ontology, we need to represent some concepts as disjoint and use cardinality restrictions, as well as inverse roles. OWL DL covers most of them; however, it does not cover some restrictions that we need. We need to restrict the range of values allowed for some properties. For instance, we want to restrict maximum/minimum range of motion of the movements, and we want to restrict the year of the related publications about the robots. OWL 2 DL covers these features, and as long as role inclusions are not recursive, it is decidable. Therefore, it is our choice of ontology language.

2.7 Pellet: A DL Reasoner

There are many DL reasoners such as FACT++, HERMIT, or PELLET that implement different kinds of algorithms for theorem proving, such as a tableau-based algorithm [6] for consistency checking. Some of them provide wrappers to the frameworks presented in Section 2.1. They might support different DLs with different expressivities as summarized in Table 2.5.

In this thesis, we use the DL reasoner PELLET for reasoning over on-

Table 2.5: DL Reasoners

Reasoner	Expressivity	Features
CEL [5] (v.1.1.2)	$\mathcal{EL}+$	Accepts inputs in KRSS (Knowledge Representation System Specification) [64] syntax. Provides an OWL API wrapper as a PROTÉGÉ plug-in. Supports axiom pinpointing to compute a justification for a consequence.
FACT++ [71] (v.1.6.2)	$SROIQ(\mathcal{D})$	Accepts inputs in FACT++ input language. Provides an OWL API wrapper. Supports explanation generation through OWL API.
FUZZYDL [12] (August, 2013)	Fuzzy $SHIF$	Accepts ontologies in lisp-like syntax. Extends $SHIF$ to the fuzzy case.
HERMIT [66] (v.1.3.8)	$SROIQ$	Provides an OWL API wrapper and uses it as a parser. Supports explanation generation through OWL API.
KAON2 [51] (August, 2013)	$SHIQ$	Additionally manages SWRL and F-Logic ontologies. Supports SPARQL querying. Provides a wrapper through PROTÉGÉ.
MSPASS [35] (August, 2013)	ALB	A prover for modal logics and relational calculus as well. Provides a proof in addition to a yes/no answer.
PELLET [67] (v.2.3.1)	$SROIQ$	Provides wrappers for OWL API and Jena. Supports SPARQL querying. Supports explanation generation.
RACERPRO [30] (v.2.0)	$SHIQ$	Provides a wrapper for OWL API. Supports explanation generation.

tologies in OWL 2 DL. It can be used for consistency checking. In fact, PELLET reduces the reasoning tasks of DL, which are exemplified above, to consistency checking. For instance, checking a subsumption in the form `WristRobots \sqsubseteq RehabilitationRobots` can be done by checking the inconsistency of `WristRobots \sqcap \neg RehabilitationRobots` in TBox. Since concepts can be viewed as sets of instances, recall that if set A contains set B , then the set difference of B from A must be an empty set. In addition, PELLET reduces instance checking to consistency checking. For example, to test whether `AssistOn-Wrist` is a `WristRobot`, a negated statement `\neg WristRobot(AssistOn-Wrist)` can be added to ABox and then the knowledge base can be checked for inconsistency. For consistency checking, PELLET implements a tableau-based theorem proving approach [6].

PELLET also supports conjunctive query answering by means of the query languages SPARQL and RDQL [65]. For that, it first parses the query using the parser that is provided by ARQ query engine. ARQ generates the SPARQL algebra that corresponds to the SPARQL query, and then PELLET evaluates basic graph patterns. For that, it maps the statements in OWL, to RDF triples. SPARQL query evaluation over RDFS knowledge bases involves simple entailment; however, query evaluation over ontologies in DL-based OWL variants should allow using logical entailment relations. Therefore, PELLET matches graph patterns using OWL-DL entailments, that extend simple entailment. According to the answers that PELLET generates, ARQ handles complex queries. For instance, if a SPARQL query contains the UNION construct, ARQ gets the answers to each disjoint basic graph pattern from PELLET, and then gets disjunction of these answers.

PELLET is implemented in Java, and available online as open source⁷. It also supports many Java based APIs and can be reached through a command line interface as well.

2.8 PROTÉGÉ: An Ontology Editor

Ontology editors are applications that provide graphical user interfaces to help the users design and manipulate ontologies easily. There are many ontology editors, such as PROTÉGÉ [28], NEON TOOLKIT [31] or VITRO [48] that are available online. They are able to visualize ontologies as well. For instance, PROTÉGÉ provides both tree-based and graph-based visualizations for ontologies.

We used PROTÉGÉ to design our ontology, in particular, our terminology. In other words, we represented general concepts and described their relations using the ontology editor PROTÉGÉ. PROTÉGÉ supports design of ontologies in OWL 2 DL, and it also supports design of ontologies in several representation formats. Therefore, we were able to represent REHABROBO-ONTO in the logic-based ontology language OWL 2 DL and in OWL/XML format. Since PROTÉGÉ provides plug-ins for DL reasoners, we utilized the plug-in of PELLET to ensure the consistency of our terminology.

2.9 Jena: An Ontology Management Framework

Among the frameworks that are presented in Section 2.1, we decided to use Jena. Jena provides an application programming interface (API) to read and process OWL ontologies. Using its API, we can add new assertions to

⁷<http://clarkparsia.com/pellet/>

REHABROBO-ONTO. Therefore, we are able to add and extract assertions using Jena framework. Jena contains various internal reasoners; however, we are interested in using PELLET. Since PELLET provides a wrapper for Jena, we are able to check the consistency in our ontology while adding a new assertion. We also query over REHABROBO-ONTO via Jena framework. We transform natural language queries into SPARQL, and thanks to availability of the DL reasoner PELLET in Jena, it computes answers to these queries automatically. In this way, Jena differs from other ontology management frameworks. For instance, OWL API, which is another framework to manipulate ontologies, does not provide support for SPARQL queries. Querying with SPARQL over ontologies is done in OWL API via Jena.

2.10 Discussion

In this section, we discuss our choices of semantic web technologies, as well as query languages and reasoners. We review our choices and decisions for this thesis, compare our approach with other possible approaches and discuss several directions for future work in terms of other technologies that can be used.

Why SPARQL?

In order to query over an ontology, we need to use a query language. Although the queries are entered in natural language, they should be transformed into an existing query language to execute a query over a query engine and then, to get an answer. There are several options to choose from. One option is to use a DL query language, and another option is to use SPARQL.

Ontology editor PROTÉGÉ and OWL API framework support DL queries in Manchester OWL syntax. With DL queries, it is possible to query about the logical structure of the ontology, and the queries can include subsumption.

However, by the time we started developing REHABROBO-QUERY, OWL API provided no support for DL query languages. In addition, asking DL queries to PROTÉGÉ externally (e.g., via command line) was not possible. Therefore, instead of DL queries, we decided to query over REHABROBO-ONTO with SPARQL, considering that it is a W3C recommendation and a technology that is widespread in semantic web community.

Why PELLET?

We developed REHABROBO-ONTO in OWL, in particular, in OWL 2 DL. We also designed REHABROBO-ONTO with the features that OWL 2 DL provides, such as inverse roles, disjoint concepts and value restrictions for properties of concepts. With these features, we do not have to specify all knowledge explicitly; some of the knowledge are implicit in the ontology. Since OWL 2 DL is based on Description Logics (DL), we can use DL reasoners to infer implicit knowledge. In addition, DL reasoners can check the ontologies for consistency or do instance checking. Therefore, we decided to use a DL reasoner.

There are various DL reasoners such as HERMIT, FACT++, or PELLET. We decided to use PELLET because of several reasons. First, we have chosen to query REHABROBO-ONTO with SPARQL; and PELLET is one of the DL reasoners that provides support for SPARQL querying by conjunctive query answering and instance checking. Second, it can also be used for consistency checking in frameworks such as OWL API or Jena, like other DL reasoners,

and it can explain inconsistencies. However, SPARQL query support of PELLET is only available in Jena framework, and OWL API supports SPARQL queries over PELLET with a plug-in that Jena framework provides for OWL API. We want to benefit from the various reasoning tasks that PELLET provides by using one framework; thus, our decision about the framework is affected by our reasoner choice.

Why Jena?

The preliminary version of REHABROBO-QUERY was designed as a desktop application. It utilized OWL API and DL reasoner HERMIT for reasoning. After deciding to add a query feature in REHABROBO-QUERY, we realized that there is no support in OWL API for DL queries and it is only possible to execute DL queries in PROTÉGÉ manually. Then, we redesigned our system considering SPARQL, a W3C recommendation, and PELLET, that provides support for SPARQL querying over OWL ontologies. Since OWL API provides support for SPARQL queries through Jena, we also changed the ontology management framework that we use.

The final version of REHABROBO-QUERY is a Web-based software that utilizes Jena framework for ontology manipulation. For querying over REHABROBO-ONTO, we use SPARQL with PELLET, via Jena framework. Our choices are made according to the state-of-the-art in semantic web technologies.

Chapter 3

3 REHABROBO-ONTO

We have designed an ontology about rehabilitation robots, called REHABROBO-ONTO, considering suggestions of the rehabilitation robotics researchers and physical medicine experts whom we collaborate with.

Our goal of developing an ontology for rehabilitation robotics is mainly to maintain a knowledge repository containing information about all rehabilitation robots and relevant references, to facilitate access to requested information in this repository for both robot designers as well as physical medicine experts. In this way, not only it will be easier for robot designers to improve the state-of-the-art in rehabilitation robotics but also it will aid translation from bench-to-bed and back, and personalized physical medicine by allowing the physical medicine experts to choose the right rehabilitation robots for specific patients/therapies.

As suggested in [72] about designing an ontology, we have first identified the purpose, and then identified and defined the basic concepts and their thematic classes, and their relationships for the chosen subject domain.

We have developed this ontology in OWL 2 DL using the ontology editor PROTÉGÉ. For the users to add/modify the ontology, we have also built an intelligent, interactive user interface for it. Let us tell these contributions in detail.

3.1 Design of REHABROBO-ONTO

We have designed our ontology (Figure 3.1) considering five main concepts (or thematic classes):

- **RehabRobots** (representing rehabilitation robots and their properties),
- **JointMovements** (representing targeted joint movements and their properties),
- **Owners** (representing robot designers who add/modify information in the ontology about their own robots),
- **References** (representing publications related to rehabilitation robots),
- **Assessments** (representing assessment measures for rehabilitation robots).

These concepts are related to each other by the following relations:

- a rehabilitation robot **targets** joint movements,
- a rehabilitation robot is **ownedBy** a robot designer,
- a rehabilitation robot **hasReferences** to some publications,
- a rehabilitation robot **hasAssessment** with respect to some evaluation measure.

As seen in Figure 3.1, each class has its own properties. **RehabRobots** have the following properties about rehabilitation robots:

- name
- active degree-of-freedom: integer

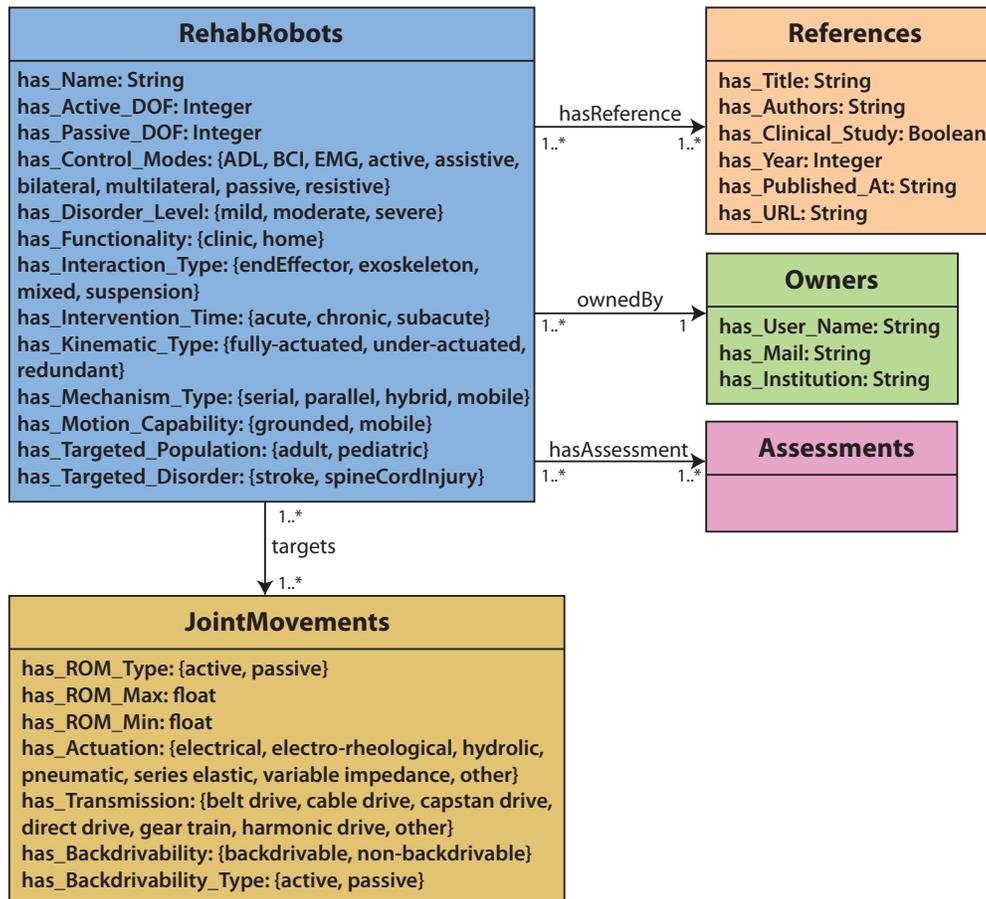


Figure 3.1: REHABROBO-ONTO with main classes.

- passive degree-of-freedom: integer
- control modes: bilateral, active, passive, resistive, assistive, ADL, multilateral, EMG, BCI
- disorder level: mild, moderate, severe
- functionality: clinic, home
- interaction type: end-effector, exoskeleton, suspension, mixed

- intervention time: acute, subacute, chronic
- kinematic type: fully-actuated, under-actuated, redundant
- mechanism type: parallel, hybrid, serial, mobile
- motion capability: grounded, mobile
- targeted population: pediatric, adult
- targeted disorder: stroke, spine cord injury

JointMovements have the following properties about the joint movements targeted by the rehabilitation robots:

- RoM type: active, passive
- minimum RoM: float
- maximum RoM: float
- actuation: electrical, hydraulic, pneumatic, series elastic, variable impedance, electro-rheological, other
- transmission: harmonic drive, belt drive, cable drive, direct drive, capstan drive, gear train, other
- backdrivability: backdrivable, nonbackdrivable
- backdrivability type: active, passive

Considering various sorts of rehabilitation robots and various sorts of joint movements, **RehabRobots** and **JointMovements** classes have subclasses; some of these subclasses are illustrated in Figures 3.2 and 3.3. Maintaining

such a hierarchy aids not only compact representation of knowledge about rehabilitation robots (by avoiding repetitions) but also efficient reasoning about it. Currently there are 147 classes represented in REHABROBO-ONTO.

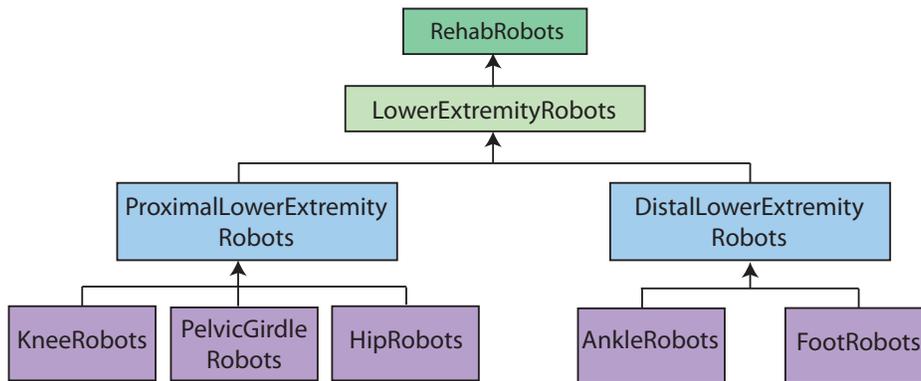


Figure 3.2: Hierarchy of lower extremity rehabilitation robots.

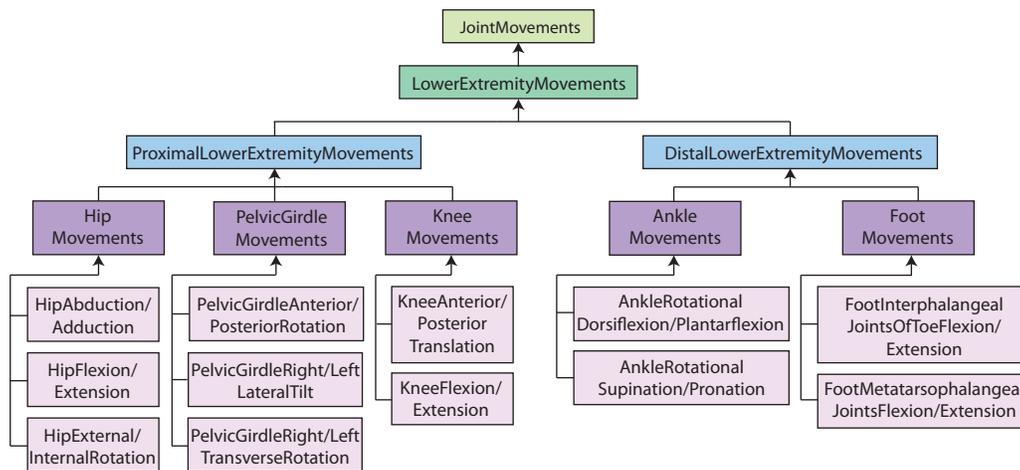


Figure 3.3: Hierarchy of lower extremity joint movements targeted by rehabilitation robots.

We have designed **Assessments** also as a hierarchy of evaluation metrics (Figure 3.4): movement quality assessments, effort assessments, psychomotoric assessments, muscle strength assessments, kinematic assessments. Each

assessment subclass has its own subclasses. For instance, `MovementQualityAssessments` class contains the following subclasses: bi-manual coordination, combined task coordination, compensation, dexterity, interlimb coordination, single joint coordination, visiomotor coordination.

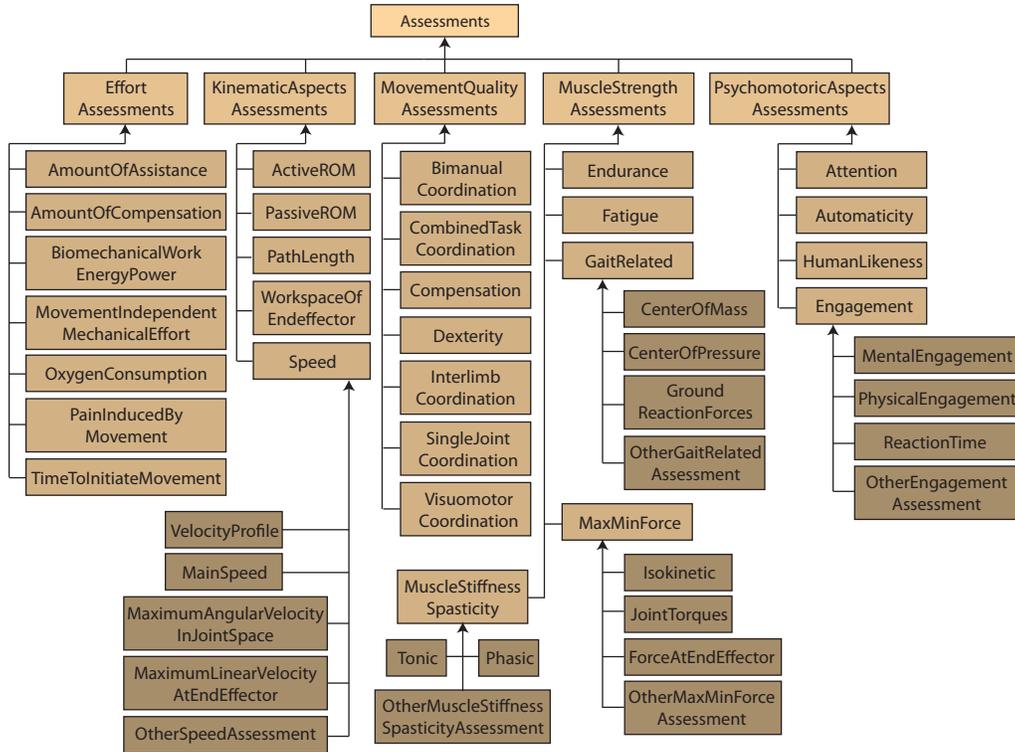


Figure 3.4: Hierarchy of Assessments.

The other concepts, `Owners` and `References`, do not have hierarchies. Though, for owners, we keep their contact information; and for references, in addition to their traditional descriptions, we also keep information about whether they contain results of some clinical studies.

REHABROBO-ONTO is essentially a DL knowledge base: the classes, class hierarchies, data properties and object properties constitute TBox; and the assertions about the robots constitute ABox.

3.2 Development of REHABROBO-ONTO

Based on the design of the classes, their properties, subclasses, relations between classes, we need to represent the ontology formally using a logic-based ontology language. Considering the standards of W3C⁸, we have decided to represent REHABROBO-ONTO in the ontology language OWL 2 DL, with OWL/XML syntax.

We have chosen OWL due to its expressive power over RDF, as explained in Section 2.6. OWL has DL based semantics and is decidable. On the one hand, DL provides well-defined semantics and supports powerful reasoning tools. We can express such disjoint classes, range restrictions and cardinality restrictions with OWL DL, one of the variants of OWL. Since we want to restrict the ranges of the properties and make some classes (e.g., subclasses of assessments) disjoint, OWL DL seems to be suitable; however, we need additional restrictions. For instance, we want to restrict the range of values allowed for some properties, such as maximum/minimum range of motion of the movements, or year of the references. This datatype restriction is allowed in OWL 2 DL, which is an extension of OWL DL that preserves decidability.

We now give references to the parts of REHABROBO-ONTO that contain these desired features. For instance, we can describe the following subclasses of assessments as disjoint with DL.

`EffortAssessment` \sqcap `MuscleStrengthAssessment` $\sqsubseteq \perp$

We describe such disjoint classes using PROTÉGÉ, ontology editor. In Figure 3.5, the same disjointness description is done in the graphical user interface of PROTÉGÉ.

Using PROTÉGÉ, we can also use a cardinality restriction for `ownedBy` re-

⁸<http://www.w3.org/standards/>

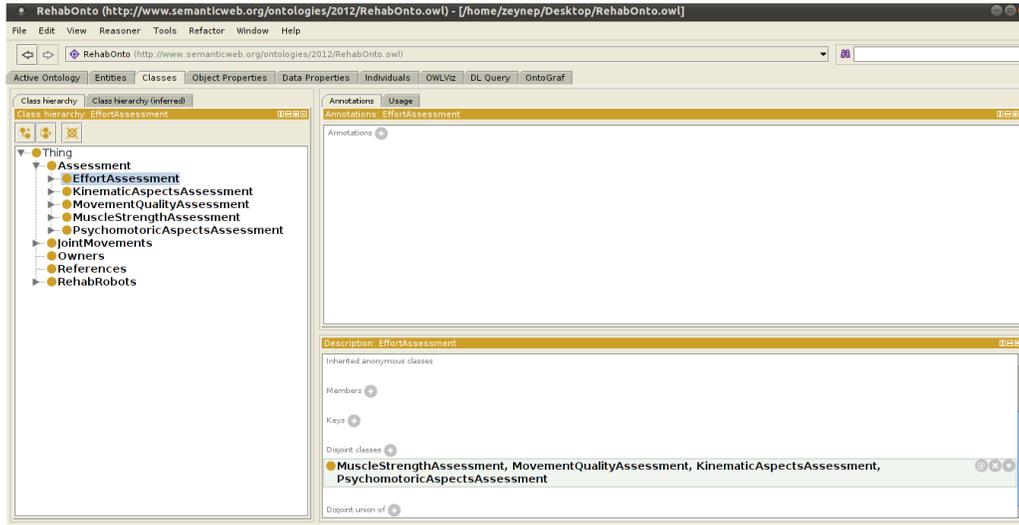


Figure 3.5: Declaring disjoint classes in PROTÉGÉ.

lation between `RehabRobots` and `Owners` to declare that every rehabilitation robot is owned by at most one owner. Declaring `ownedBy` as a functional object property, as shown in Figure 3.6, is sufficient to describe such knowledge.

We restrict the range of the `has_Year` property of `References` in the ontology, as shown in Figure 3.7.

After deciding on the ontology language, we have used the OWL ontology editor PROTÉGÉ [28] to construct REHABROBO-ONTO by describing general concepts and their properties/relations. REHABROBO-ONTO consists of TBox, which contains terminological knowledge; and ABox, which contains assertional knowledge. We define concepts and their properties/relations in TBox, and the instances in ABox. We have used PROTÉGÉ solely to describe the TBox; however, our knowledge base also contains some assertions, information about the robots, in the ABox. How we enable the robot designers to add assertions is explained in the next section.

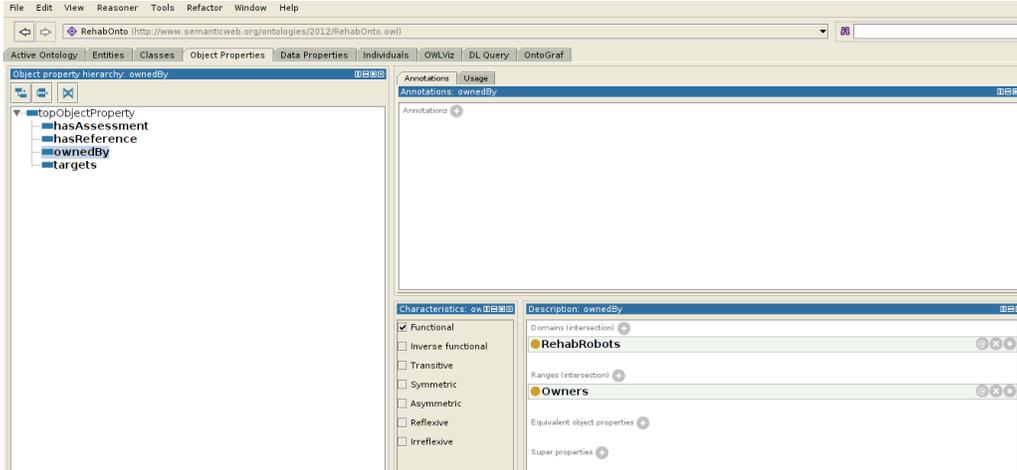


Figure 3.6: Declaring ownedBy relation in PROTÉGÉ.

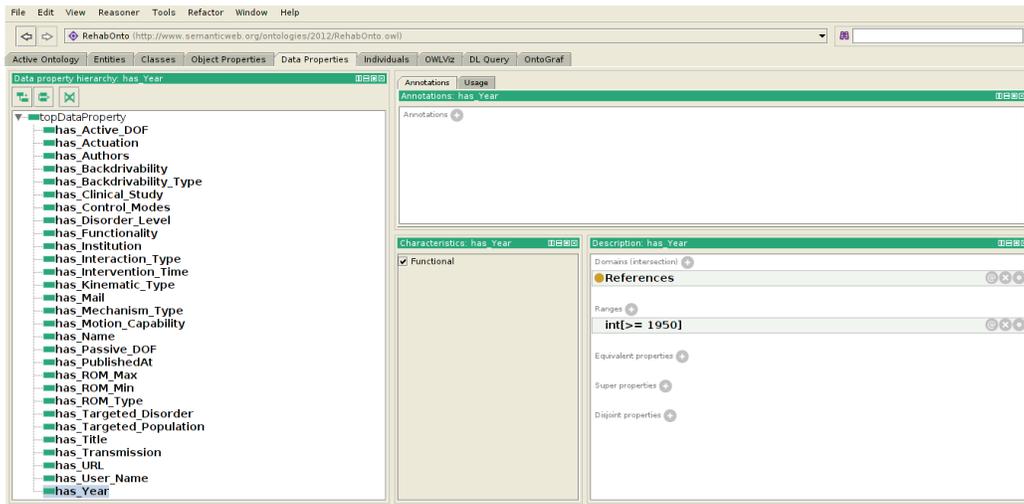


Figure 3.7: Restricting range of has_Year in PROTÉGÉ.

3.3 Maintaining REHABROBO-ONTO

Once we represent general concepts and their properties/relations about rehabilitation robots in OWL, we are not done yet. We would like the rehabilitation robotics ontology to be shared by researchers so that robot designers can add/modify information about their robots, and both rehabilitation robotics experts and physical medicine experts can ask queries over it. Therefore, we would like to allow researchers to add information about specific rehabilitation robots by “assertions” (like, “the rehabilitation robot whose name is ASSISTON-WRIST is a wrist robot and it has clinic use”) as well.

Such assertions about specific individuals can be added to REHABROBO-ONTO using PROTÉGÉ. However, since PROTÉGÉ downloads the whole ontology to be able to add new information, ensuring that the users add information to REHABROBO-ONTO without letting them modify other parts of the ontology may be problematic. Also, assuming that the existing robotics experts and physical medicine experts know about DL and logic-based ontology languages, that they have experience in using DL reasoners or Semantic Web technologies, and that they keep track of the most recent versions of these software, may not be reasonable along our goals for an effective use of the rehabilitation ontology.

To facilitate the effective use of the rehabilitation ontology by different users, we have designed a tool (called REHABROBO-QUERY) with an easy-to-use intelligent user interface. Such a user interface should be interactive such that it should guide the user through robot addition. Moreover, a user might forget to add an information or postpone entering a particular property to make sure that it is correct. In this case, the user interface should support modification of the assertions afterwards. There may be cases where a reha-

bilitation robot should be removed completely. For this purpose, deletion of an assertion should be supported as well. REHABROBO-QUERY provides all of these features by allowing robotics researchers to add/modify information to/in REHABROBO-ONTO about their robots by following consecutive tabs of the intelligent user interface, without having to know about the underlying logic-based formalism.

Registration

Only registered users can add/modify/delete information to/in/from REHABROBO-ONTO. One can register by entering his/her name, institution, e-mail address and a password. Next, REHABROBO-QUERY sends a confirmation mail including an activation link. Once the user clicks on the link, then his/her membership gets confirmed. Registered users and their passwords are administrated by a database. For security, access to this database utilizes encryption via the cryptographic hash function SHA-1 [1].

Adding Information to REHABROBO-ONTO

When the user wants to add information about his/her robot, he/she should enter the name of the robot first (Figure 3.8). Then, REHABROBO-QUERY checks whether a robot with the given name exists in the ontology. This is done using DL reasoner PELLET, with the following SPARQL query.

```
SELECT ?robot
WHERE {
    ?robot rdf:type rr:RehabRobots.
    ?robot rr:has_Name '<robotName>'.
}
```

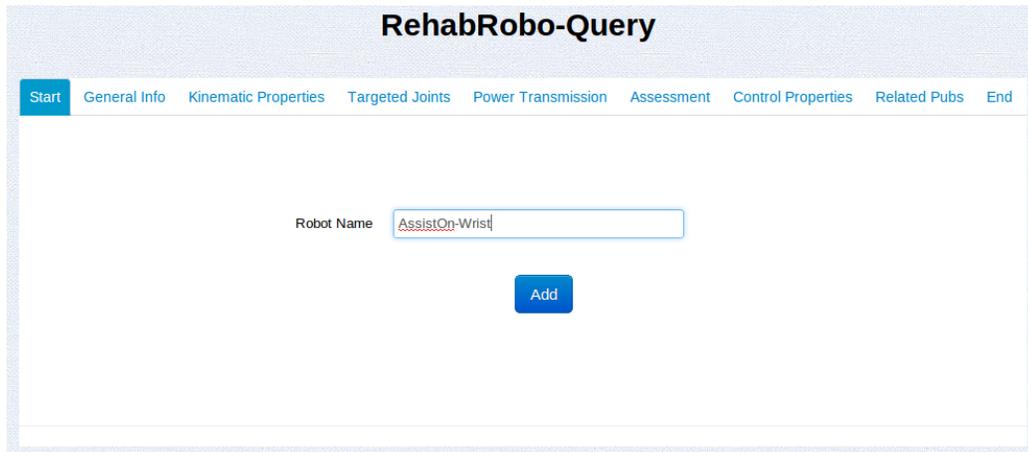


Figure 3.8: Adding a new robot to REHABROBO-ONTO.

}

We pass the name of the robot to this query before execution. If there is a robot in the ontology with such a name, then it lists some options, as shown in Figure 3.9. Otherwise, the user continues to add information by navigating the tabs.

Relevant properties are grouped into a tab in REHABROBO-QUERY. For instance, General Info tab (Figure 3.10) contains functionality, targeted population, targeted disorder, intervention time and disorder level. Kinematic Properties tab (Figure 3.11) contains motion capabilities, kinematic type, mechanism type and interaction type. We represent functional properties with radio buttons and relational properties with comboboxes. The targeted joint movements are entered via Targeted Joints and Power Transmission tabs, as shown in Figures 3.12, 3.13, 3.14 and 3.15. By clicking on a joint, its minimum/maximum ranges of motion and its range of motion type can be entered in Targeted Joints tab. Similarly, by clicking on a joint, its actuation, transmission and backdrivability properties can be specified in Power

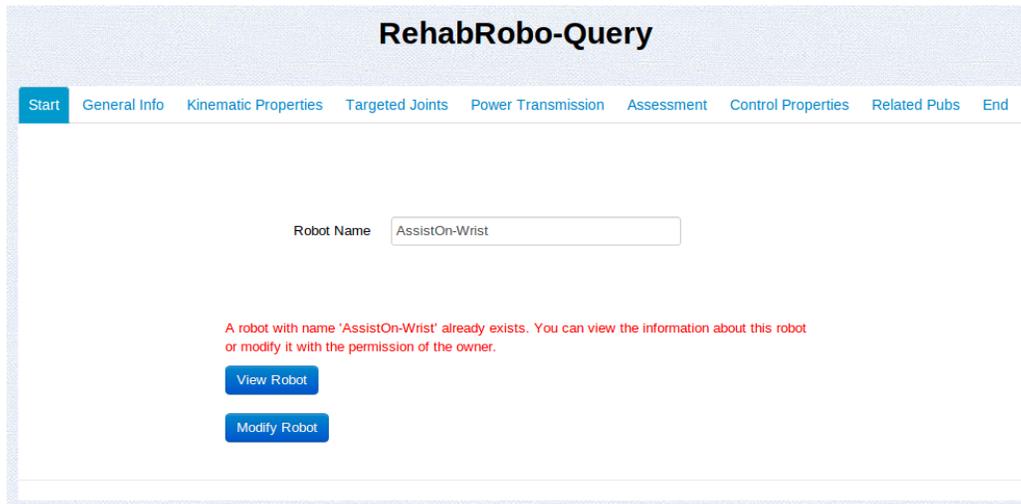


Figure 3.9: Warning for an existing robot with available options.

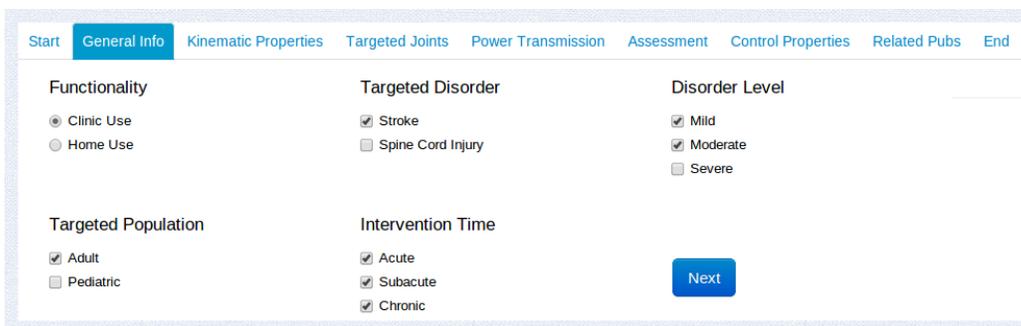


Figure 3.10: Adding to REHABROBO-ONTO general information about the rehabilitation robot ASSISTON-WRIST.

Transmission tab. The user can specify measured assessment metrics in Assessment tab (Figure 3.16). Since some of the subclasses of assessments have further subclasses, specifying particular metrics are handled by the pop-up windows, opened after clicking on a relevant assessment metric. Control Modes tab includes comboboxes for the control modes property only, and in Related Pubs tab (Figure 3.17) the user can add the publications related to his/her rehabilitation robot.

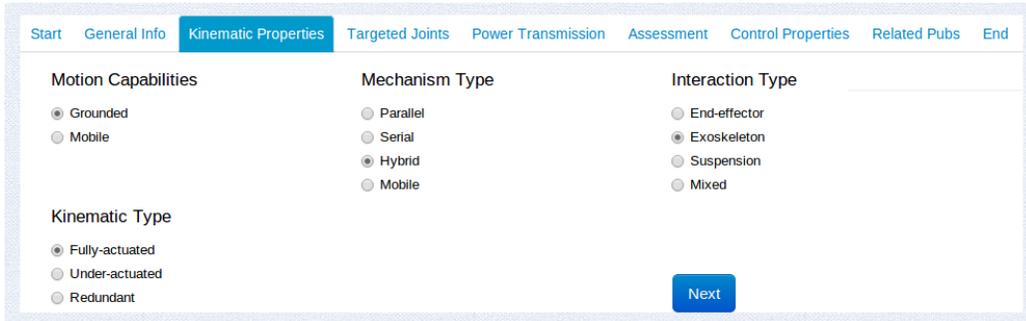


Figure 3.11: Adding to REHABROBO-ONTO kinematic properties of the rehabilitation robot ASSISTON-WRIST.

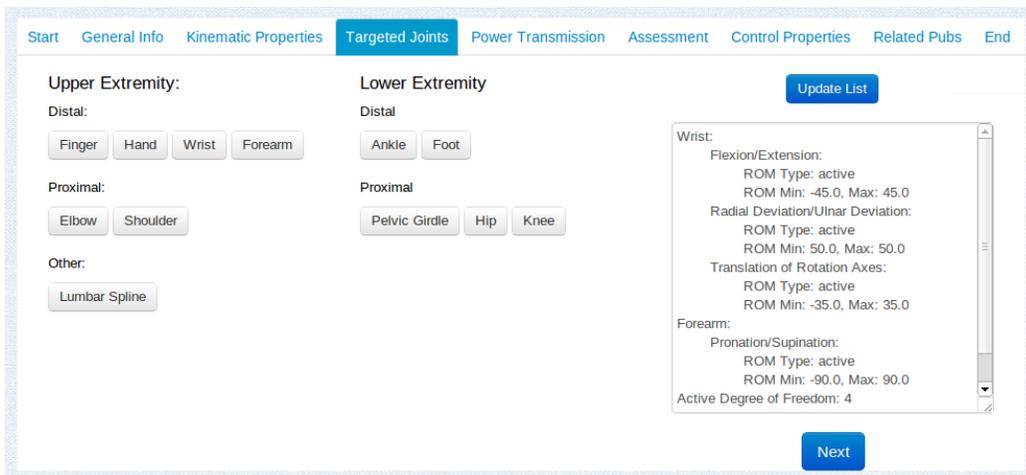


Figure 3.12: Adding to REHABROBO-ONTO targeted joints of the rehabilitation robot ASSISTON-WRIST.

Wrist ✕

Flexion/Extension: Range of Motion <input style="width: 40px;" type="text" value="-45.0"/> <input style="width: 40px;" type="text" value="45.0"/> min max <input checked="" type="radio"/> active <input type="radio"/> passive <input type="radio"/> none	Radial Deviation/Ulnar Deviation: Range of Motion <input style="width: 40px;" type="text" value="50.0"/> <input style="width: 40px;" type="text" value="50.0"/> min max <input checked="" type="radio"/> active <input type="radio"/> passive <input type="radio"/> none	Translation of Rotation Axes: Range of Motion <input style="width: 40px;" type="text" value="-35.0"/> <input style="width: 40px;" type="text" value="35.0"/> min max <input checked="" type="radio"/> active <input type="radio"/> passive <input type="radio"/> none
---	---	--

Figure 3.13: Adding to REHABROBO-ONTO a targeted joint (wrist) of the rehabilitation robot ASSISTON-WRIST, from a pop-up window.

Start
General Info
Kinematic Properties
Targeted Joints
Power Transmission
Assessment
Control Properties
Related Pubs
End

Upper Extremity

Distal

Proximal

Other

Please make sure that the joints in red have correct information.

Lower Extremity

Distal

Proximal

Update List

Wrist:

Flexion/Extension:
 ROM Type: active
 ROM Min: -45.0, Max: 45.0
 Actuation: electrical
 Transmission: [direct drive]
 Backdrivable: passive

Radial Deviation/Ulnar Deviation:
 ROM Type: active
 ROM Min: 50.0, Max: 50.0
 Actuation: electrical
 Transmission: [direct drive]
 Backdrivable: passive

Translation of Rotation Axes:
 ROM Type: active

Next

Figure 3.14: Adding to REHABROBO-ONTO power transmissions of the rehabilitation robot ASSISTON-WRIST.

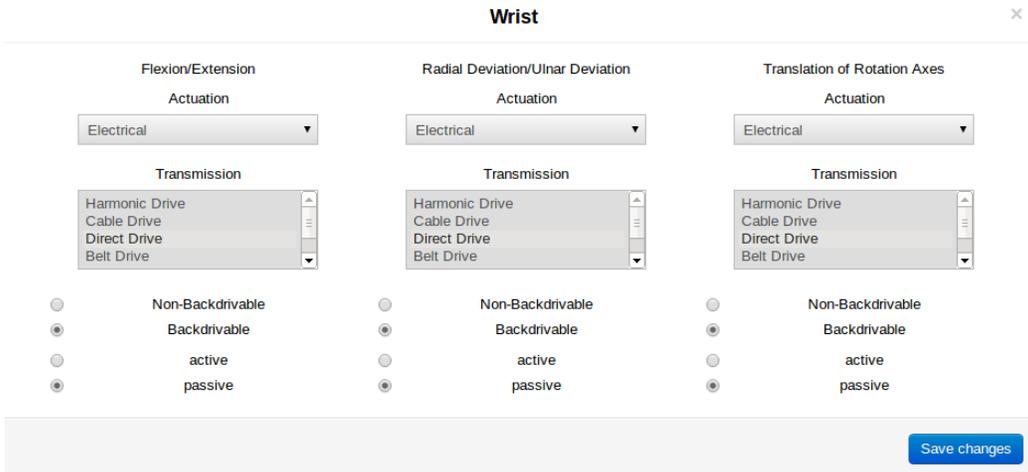


Figure 3.15: Adding to REHABROBO-ONTO power transmissions of a targeted joint (wrist), from a pop-up window.

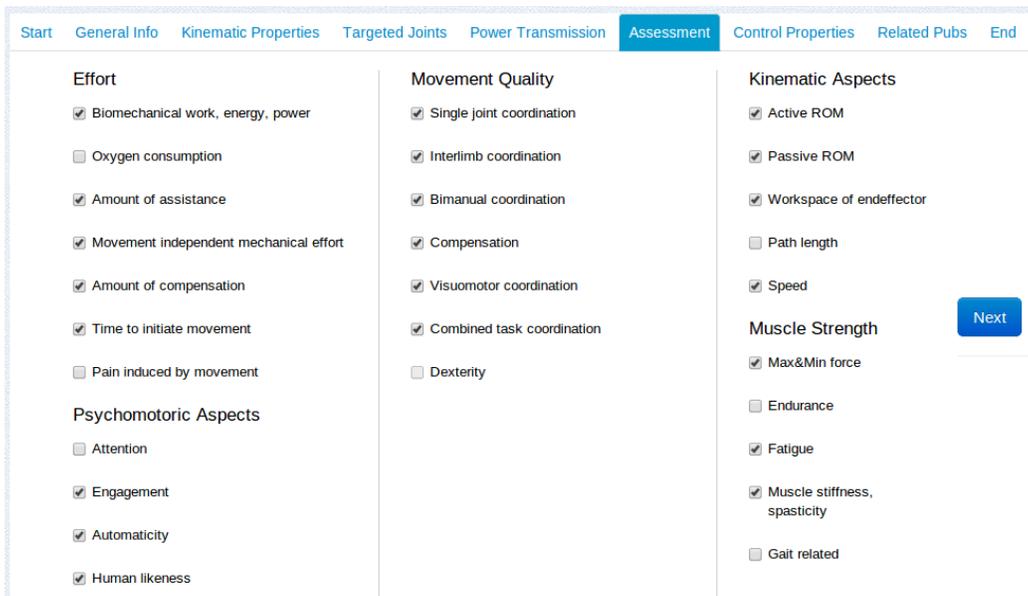


Figure 3.16: Adding to REHABROBO-ONTO assessments of the rehabilitation robot ASSISTON-WRIST.

Figure 3.17: Adding to REHABROBO-ONTO related publications of the rehabilitation robot ASSISTON-WRIST.

As the user describes the robot by filling the tabs, he/she has the chance to return to any tab to change the information. After entering all properties of the robot, in the End tab, all the information entered by the user is displayed as a summary for the last time. After the user checks the information and confirms its addition to REHABROBO-ONTO, the information about the rehabilitation robot is transformed into assertions in OWL and added to REHABROBO-ONTO.

Modifying REHABROBO-ONTO

A user can modify or delete his/her own robots only. First the relevant robots are found by querying REHABROBO-ONTO using the DL reasoner Pellet [67] via Jena [50], and listed in a pull-down menu (Figure 3.18). To find the robots that are owned by a user, we query REHABROBO-ONTO with

the following ready-to-use SPARQL query. To this query, we pass the e-mail address of the user, since it is the unique property of the user.

```
SELECT DISTINCT ?robotName
WHERE {
  ?owner rdf:type rr:Owners.
  ?owner rr:has_Mail '<mail>'.
  ?robot rr:ownedBy ?owner.
  ?robot rdf:type rr:RehabRobots.
  ?robot rr:has_Name ?robotName.
}
```

For modification, after the user chooses a robot from the list, the user interface that we have seen earlier for adding information appears but now with tabs filled with the robot's properties. The user can make changes via this interface and the updated information can be saved as a set of assertions in OWL, in a new file while keeping the previous version as "modified". For deletion, after the user chooses a robot from the list, the relevant file containing assertions about that robot is marked as "deleted". Note that in both cases, we keep the information about the robot before modification/deletion as well; these files may be needed if the user accidentally deletes his/her robot from REHABROBO-ONTO, or modifies it incorrectly.

Authorization

When a user wants to make changes on a robot that the user is not authorized to make changes about, the permission of its owner is required via REHABROBO-QUERY. After clicking Modify with Permission in Figure 3.9,

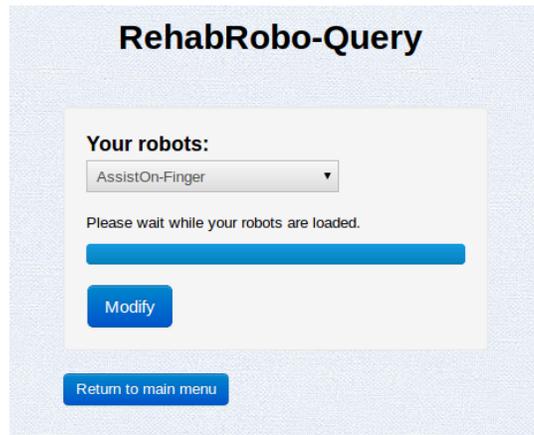


Figure 3.18: Robots that are owned/maintained by a particular user.

the same user interface for modification and addition appears with tabs filled with the robot's properties. The user makes modifications navigating the tabs and in the End tab, the user checks the information and confirms sending a request to the owner of the robot. The information entered by the user is saved as a set of assertions, marked as "requested". Then the information is sent to the e-mail address of the owner for confirmation. Once the owner confirms the new information, requested assertion is replaced as the latest version of this robot, while keeping the previous version as "modified". If the owner does not confirm the new information, the requested information is deleted from the server.

Feedback

After adding, modifying and deleting information, REHABROBO-QUERY allows users to provide feedback about the system. To prevent automated access to the system by computer programs, Google RECAPTCHA⁹ is used.

⁹<http://www.google.com/recaptcha>

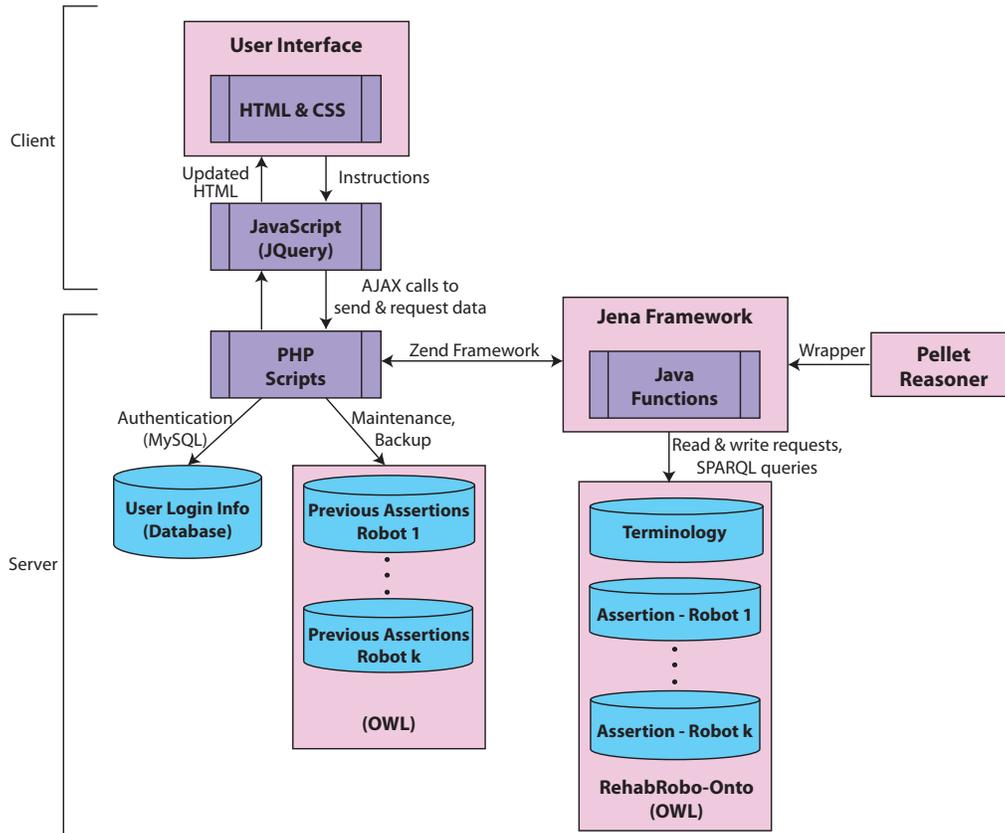


Figure 3.19: An overview of REHABROBO-QUERY.

3.4 Overall System Architecture

The overall system architecture for REHABROBO-QUERY on the cloud is illustrated in Figure 3.19.

Since REHABROBO-QUERY is a Web-based system available via Amazon Web Server, it consists of two parts: client and server. In general, client part provides interaction with the user whereas server part makes the operations that user does not actually see, in the background.

After entering the web page, the user sees a set of texts and buttons, that are designed to guide the user through pages. For adding or modify-

ing robots, REHABROBO-QUERY provides more graphical features, such as radiobuttons, checkboxes or pop-up windows to make this process easier for the user. These features are provided with a set of HTML and CSS files. We use the components that Twitter Bootstrap [44] provides for styling the web pages.

According to the instructions that the user specifies on the user interface, REHABROBO-QUERY modifies the components that the user sees, or changes the displayed web page. For that, we use JavaScript to specify how the user interface changes according to the instructions. Therefore, instructions are JavaScript calls that cause the web page or the components in the web page to change. For instance, when the user finishes entering information about a robot and clicks Add button, a JavaScript call is generated and according to the operations in the background, the web page displays an error message in the same window, or moves to the feedback page, indicating that addition is successful.

Up to this point, we have described the client part of REHABROBO-QUERY. In order to make the operations that are requested by the user, the client part should interact with the server part. Ajax is a technique to provide such interactions. Using Ajax scripts, we can make calls to the server side and then get a return value indicating success or error. We utilize Ajax, and we make Ajax calls from the scripts in JavaScript. In particular, we use the JavaScript library JQuery [13] because it enables Ajax interactions with the server side scripts. For example, assume that a user wants to add a new robot. Then, the user enters information and clicks Add button. When the user clicks Add button, the JavaScript calls are generated to the scripts that are associated with addition page. Then, in these scripts, the information

about the robot in the tabs are collected and serialized. After that, the script sends the serialized information about the robot to the server side with an Ajax call and waits for a response, or if the Ajax call is asynchronous, the script continues to execute rest of the script while waiting for the response from the Ajax call.

Reasoning, SPARQL querying, file operations, and authentication are done on the server side. Ajax calls from the client side start execution of associated PHP scripts. For instance, when the user tries to log in to REHABROBO-QUERY by entering the user name and password, then through the JavaScript file associated with login page, an Ajax call is done to a login script in PHP. Then, this script gets the entered information via Ajax, connects to the user database and runs an SQL query over MySQL, to check whether such a user exists. If a user exists, then PHP script sets the session for this user and returns success. Otherwise, it catches the possible exceptions and returns failure.

Both the user database and the rehabilitation robotics ontology are stored on the server. We store the assertions about each rehabilitation robot in a separate file, to make it easy to modify/delete REHABROBO-ONTO as well as for efficient query answering. In other words, when the user adds information about his/her robot, it is stored (in OWL/XML syntax) as an assertion in a unique file. The terminology of the ontology (that consists of classes, their properties and relations as described in Section 3.1) is kept in a separate file, also in OWL/XML syntax. The ontology REHABROBO-ONTO consists of the terminological part and the assertions of the robots. Users are not allowed to modify the terminological part of the ontology, but only assertions about the robots.

When the user modifies (resp., deletes) information about his/her robot, then we mark the related assertion file as “modified” (resp., “deleted”). We do such file manipulations from PHP scripts. Thus, when the user chooses a robot and clicks Delete from the user interface, then we call the associated deletion script in PHP, via an Ajax call. Then, we locate the related file in the server and mark it as deleted.

Adding new assertions and querying with SPARQL is done by making calls to Java functions that utilize Jena Framework and DL reasoner PELLETT. For that, we call our ready-to-use functions in Java, from PHP scripts. Calling Java functions from PHP scripts is enabled with the PHP-Java bridge of Zend Framework. We have functions to extract related information about robots, such as getting the references of a robot or getting the assessments of a robot. We also have a function to add a new assertion into REHABROBO-ONTO or to execute a query over REHABROBO-ONTO. For instance, after the user enters information about a robot and clicks Add button, the associated PHP script gets the information from the client side and delegates to the related Java function for robot addition. Then, the assertions about the properties of the robot are added one by one, and all of them are written on a unique file. If the user chooses to modify a robot, then the associated PHP scripts for modification call the Java functions to extract the information (targeted joints, general information etc.) about the robot. Such Java functions use PELLETT to load the terminology and the related assertions. Then, a ready-to-use SPARQL query is executed by completing the query with the related robot’s ID. The SPARQL query that we use to extract the references is as follows.

```
SELECT ?refInd
```

```
WHERE {  
rr:<robotID> rr:hasReference ?refInd.  
}
```

After extracting the indices of the references, we extract their relevant information. For instance, the following query returns the title of a reference.

```
SELECT ?title  
WHERE {  
rr:<refInd> rr:has_Title ?title.  
}
```

We get the answers from PELLET and serialize them to send back to the PHP scripts. On the client side, we unserialize them and fill the user interface with related information.

We give the details of the addition process by presenting the work flow in Figure 3.20 and the data flow in Figure 3.21. The main work flow, which covers the login of the user to reach to the main page, is shown in Figure 3.22. The work flow of modification starts by getting the user's robots via Jena framework. Then, the system displays robot names to the user and the user chooses a robot to modify. After modifying information and clicking Modify, the name of the file containing assertions for this robot, say "RehabOntoInstance_R5.owl" is changed to "modified_RehabOntoInstance_R5.owl". The rest of this process involves connecting to Jena framework again, which is the same process as in addition. The work flow of deletion begins by displaying user's robots and getting a choice, which is same as modification. Then, the system asks if the user is sure about the deletion. If the user chooses "yes",

then the name of the file containing assertions for this robot, say “RehabOntoInstance_R5.owl” is changed to “deleted_RehabOntoInstance_R5.owl”. If the user chooses “no”, then the system returns to main menu.

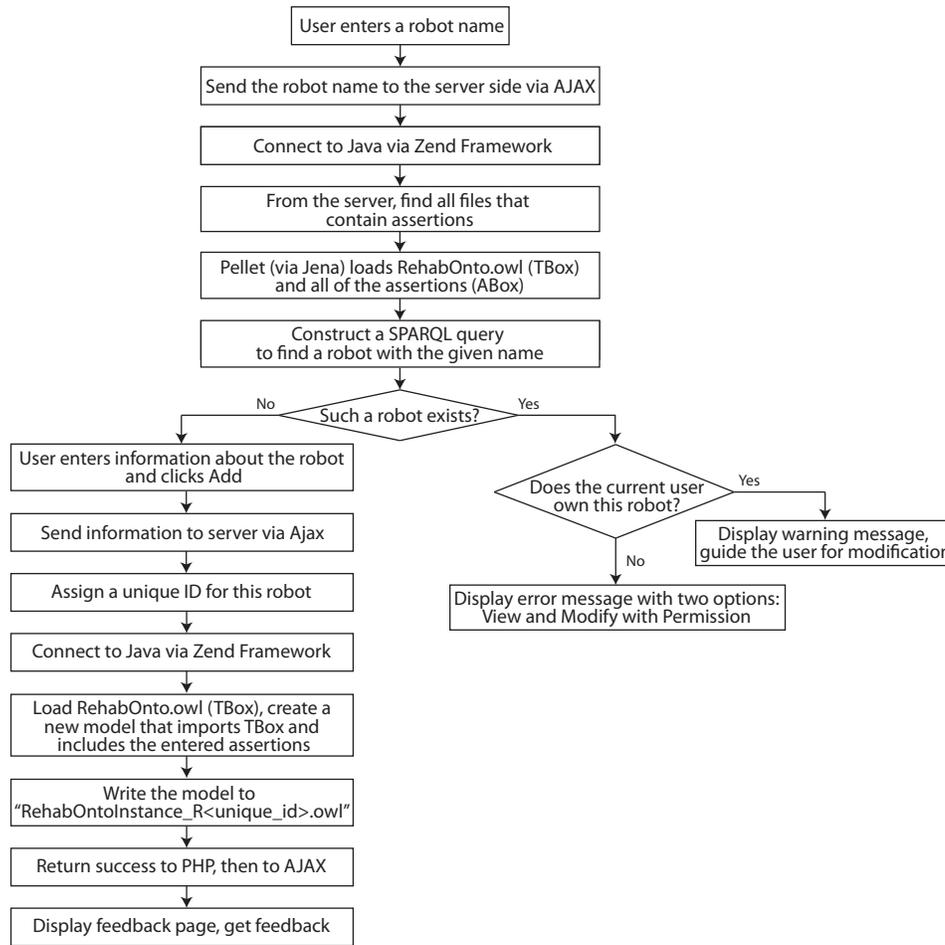


Figure 3.20: Work flow of addition.

3.5 REHABROBO-ONTO on the Cloud

We utilize Amazon Elastic Compute Cloud (Amazon EC2) for both developing and maintaining REHABROBO-ONTO, and for querying REHABROBO-

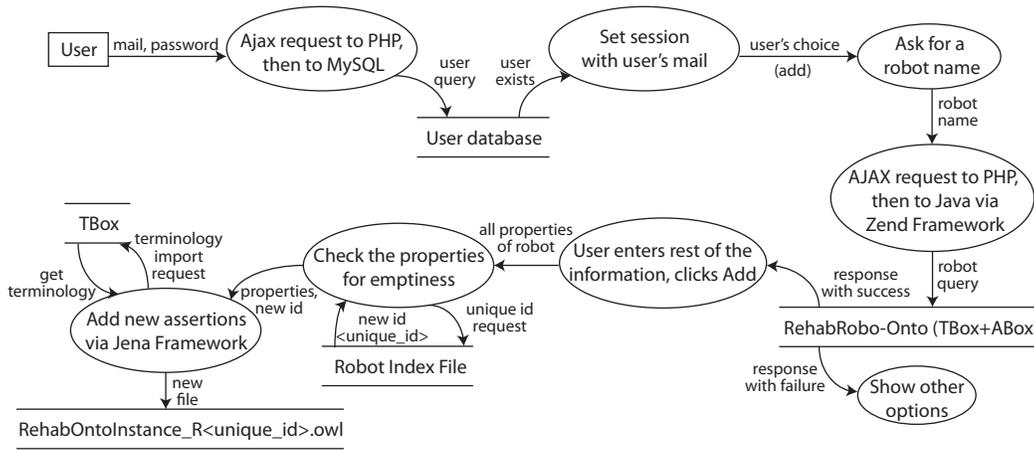


Figure 3.21: Data flow of addition.

ONTO via REHABROBO-QUERY. Amazon EC2 is a web service that provides resizable compute capacity in the cloud, and makes web-scale computing easier for developers. Considering the possibility of various researchers from around the world add/modify/query REHABROBO-ONTO via REHABROBO-QUERY, and the possibility of integrating various sorts of knowledge on the web related to rehabilitation robotics, Amazon EC2 provides a reliable environment for development and maintenance of REHABROBO-ONTO and REHABROBO-QUERY. REHABROBO-QUERY is available via a webpage¹⁰ hosted by a Web server running on Amazon EC2.

The server utilizes a standard LAMP stack, which refers to Linux, Apache HTTP Server, MySQL and PHP. In particular, the server runs on Ubuntu 12.04 (64 bit), utilizing Apache v2, MySQL v5.5 and PHP v5.3. While Apache provides a web server, server side development is done by PHP and the user database is stored in a MySQL database.

To access REHABROBO-QUERY, having a javascript enabled web browser

¹⁰http://hmi.sabanciuniv.edu/?page_id=781

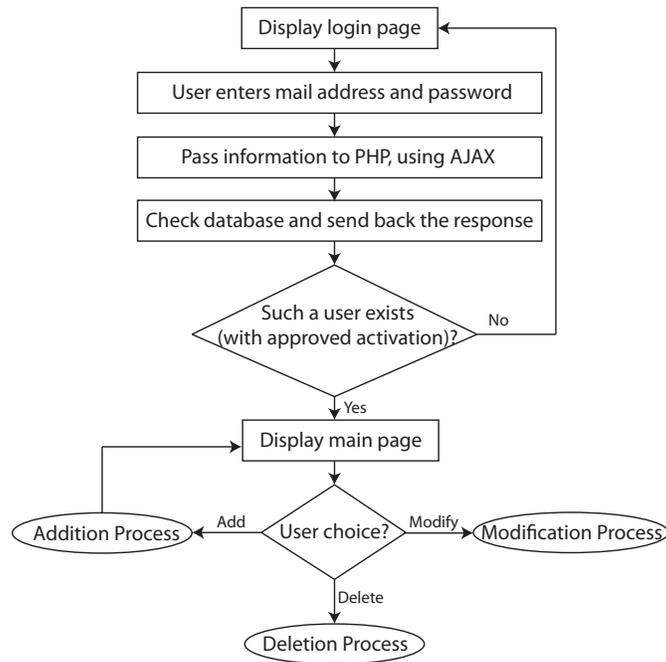


Figure 3.22: Main work flow, including login.

is sufficient. In particular, REHABROBO-QUERY is tested with Google Chrome 24.0.1312.70, Mozilla Firefox 16.0.2, Opera 12.01 and Internet Explorer 8+.

For reliable maintenance, we conduct regular backups of REHABROBO-ONTO. With regular backups, it is also possible to restore further data that may be lost, such as requested but declined modifications of a robot.

Chapter 4

4 REHABROBO-QUERY

Reasoning over REHABROBO-ONTO is done by means of answering questions posed by the user in natural language. To overcome the ambiguities in the vocabulary and grammar of natural languages, we introduce a Controlled Natural Language (CNL), a subset of a natural language with a restricted vocabulary and grammar. A CNL is essentially formal language, and thus it is not difficult to convert a CNL to a logic-based formalisms. In that sense, a CNL facilitates the use of automated reasoners to find answers to queries expressed in a CNL.

In order to express queries about rehabilitation robots, we designed and developed a new CNL, called REHABROBO-CNL. Although we designed REHABROBO-CNL considering REHABROBO-ONTO, it is possible to expand it to support queries about integrated knowledge resources (e.g., patients, diseases, genetic information).

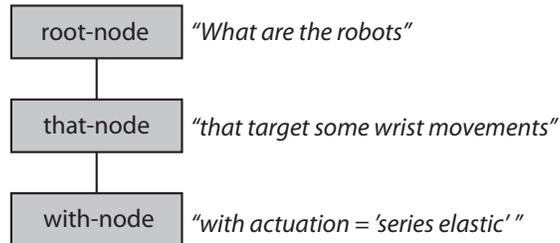
Some example queries are listed in Table 4.1. To answer these queries, REHABROBO-QUERY transforms them into the formal query description language SPARQL, also an official W3C Recommendation. To give an overall idea of the transformation, consider, for instance, the following query (Q1 in Table 4.1):

Table 4.1: Sample Queries in REHABROBO-CNL

Q1	What are the robots that target some wrist movements with actuation='series elastic' ?
Q2	What are the effort metrics that are evaluated by some robots with active degree of freedom ≥ 2 ?
Q3	What are the movement quality metrics that are evaluated by all robots with motion capability = 'grounded' ?
Q4	What are the users with institution='Sabanci University' and that own/maintain the robot 'AssistOn-Wrist' ?
Q5	What are the publications with clinical study and that do not reference any robots with active degree of freedom ≤ 1 ?
Q6	What are the movements that are targeted by some robots with (some intervention time or with all targeted disorders)?
Q7	What are the publications without clinical study or that reference some robots that do not evaluate any movement quality metrics?
Q8	What are the robots that target the shoulder horizontal abduction/adduction with range of motion type='active' or that target the elbow flexion/extension with transmission={belt drive, cable drive}?
Q9	What are the robots that target all foot movements and (with targeted population='pediatric' or with control modes={active, assistive})?
Q10	What are the publications with place of publication 'ICORR' and that reference some robots that are owned/maintained by some users with institution 'Sabanci University' ?
Q11	What are the robots with no targeted disorder or (with intervention time!='chronic' and with motion capability='grounded') or with no disorder level?
Q12	What are the robots with interaction type = 'exoskeleton' and that target some finger movements (with actuation = 'electrical' or with actuation = 'hydraulic' or with actuation = 'series elastic') ?

What are the robots that target some wrist movements with actuation='series elastic'?

First, we parse this query and represent it as a tree. The following tree describes a concept: A robot that targets some wrist movements with actuation 'series elastic'.



Second, we transform the tree into a DL concept by traversing the tree. The following DL concept corresponds to the tree above.

```
Robot  $\sqcap$   $\exists$ targets.(WristMovement  $\sqcap$   $\exists$ actuation.{series elastic})
```

Third, we transform the DL concept into a SPARQL concept and construct the SPARQL query:

```
SELECT DISTINCT ?name
WHERE {
  ?robot1 rr:has_Name ?name.
  ?robot1 rdf:type rr:RehabRobots.
  ?robot1 rr:targets ?movement1.
  ?movement1 rdf:type rr:WristMovements.
  ?movement1 rr:has_Actuation 'series elastic'.
```

}

Finally, we execute the SPARQL query and get the answer from PELLET (answer is “AssistOn-Mobile”).

The answers to a given query are presented to the user as a list. If the query is about robots, then we also enable the users to click on one of the robots. By clicking one of the listed robots, the users can view further properties of these robots.

4.1 REHABROBO-CNL

REHABROBO-QUERY allows users to express queries about rehabilitation robots. We introduced a controlled natural language (CNL), called REHABROBO-CNL, to express these queries.

The grammar of REHABROBO-CNL is shown in Table 4.2. With REHABROBO-CNL, it is possible to construct queries that contain nested relative clauses, disjunctions, conjunctions, negations, and quantifications; such as some, all, any, none.

To eliminate the ambiguities in nesting of conjunctions and disjunctions, REHABROBO-CNL provides two ways of constructing a query: A query in REHABROBO-CNL should either be in Conjunctive Normal Form (CNF), or in Disjunctive Normal Form (DNF). In other words, REHABROBO-CNL supports conjunctions of simple disjunctions, and disjunctions of simple conjunctions. No further nesting of conjunctions (resp., disjunctions) in a simple disjunction (resp., conjunction) is allowed. A query can contain any number of conjunctions and disjunctions on the condition that they match to the rule above. An example of a query in CNF is as follows.

Table 4.2: The Grammar of REHABROBO-CNL

QUERY →	WHATQUERY QUESTIONMARK
WHATQUERY →	What are the <i>Type()</i> GENERALRELATION
GENERALRELATION →	SIMPLERELATION NESTEDRELATION*
SIMPLERELATION →	(that RELATIVECLAUSE)+
SIMPLERELATION →	that INSTANCERELATION
SIMPLERELATION →	WITHRELATION
NESTEDRELATION →	(and LP SIMPLEDISJUNCTION RP)*
NESTEDRELATION →	(or LP SIMPLECONJUNCTION RP)*
SIMPLEDISJUNCTION →	(SIMPLERELATION or)* SIMPLERELATION
SIMPLECONJUNCTION →	(SIMPLERELATION and)* SIMPLERELATION
RELATIVECLAUSE →	<i>Verb()</i> (some all the) <i>Type()</i>
RELATIVECLAUSE →	NEG <i>Verb()</i> any <i>Type()</i>
INSTANCERELATION →	NEG? <i>Verb()</i> the <i>Type()</i> <i>Instance()</i>
WITHRELATION →	with <i>Noun()</i> EQCHECK <i>Value()</i> +
WITHRELATION →	with QUANTIFIER <i>Noun()</i>
WITHRELATION →	(with without) <i>Noun()</i>
EQCHECK →	= != ≤ ≥
QUANTIFIER →	some all none
NEG →	<i>Neg()</i>
LP →	(
RP →)
QUESTIONMARK →	?

What are the robots with mechanism type='hybrid' and (with motion capability = 'grounded' or with functionality='clinic') and that target some wrist movements?

The following query is in DNF:

“What are the robots with no targeted disorder or (with active degree of freedom > 1 and with control modes = '{active, assistive}' and with no disorder level)?”

The italic functions in the grammar extract relevant information from REHABROBO-ONTO. These ontology functions are described in Table 4.3.

Table 4.3: The Ontology Functions

<i>Type()</i>	Returns the types that correspond to concept names. They are: Robots, movements, users, publications and metrics.
<i>Instance()</i>	Returns robot names for robots and user names for users.
<i>Verb()</i>	Returns the verbs that correspond to object properties between concepts. Returns both active and passive forms of these verbs. Active forms of these verbs are: Target, evaluate, reference, own.
<i>Noun()</i>	Returns the nouns that correspond to data properties. ex. targeted disorder, active degree of freedom.
<i>Value()</i>	Returns the suitable values according to a given noun. Corresponds to the pre-defined ranges of data properties.
<i>Neg()</i>	Returns a suitable negation phrase. These phrases are: do not, are not.

The information extracted with the ontology functions are coupled by their relevance. For instance, only the verb “reference” can appear after the type Publications. By matching types with verbs, it is possible to prevent semantically wrong queries like “What are the publications that target some shoulder movements?”. All of the matches between types and verbs are shown in Table 4.4. Similarly, it is necessary to match verbs with types.

Table 4.5 lists the available types that can occur after a verb in the query (e.g., in a RELATIVECLAUSE).

Table 4.5 also demonstrates what kinds of types are extracted from the ontology according to the query. If a quantifier such as “some” is used in a relative clause, then the types which have a number of subclasses are extracted. If “the” keyword is used after the verb, then the leaf classes are extracted to select one specific type. The following query illustrates the first case.

What are the robots that evaluate some wrist movements?

Since wrist movements class have subclasses (e.g., wrist flexion/extension, wrist radial deviation/ulnar deviation) in REHABROBO-ONTO, this query will retrieve all robots that target at least one of these subclasses. An example for a query with “the” is as follows.

What are the robots that target the wrist radial deviation/ulnar deviation?

Wrist radial deviation/ulnar deviation is a leaf class. It is also a subclass of wrist movements. This query will retrieve the robots that target this specific wrist movement. If there is a robot that targets some other wrist movements but wrist radial deviation/ulnar deviation, then this robot will not be included in the answer to this query.

Table 4.4: Verbs that can occur after the nouns

<i>Type()</i>	that	<i>Verb()</i>
robots	→	target
robots	→	are owned/maintained
robots	→	evaluate
movements	→	are targeted by
users	→	own/maintain
publications	→	reference
effort metrics	→	are evaluated by
kinematic aspect metrics	→	are evaluated by
movement quality metrics	→	are evaluated by
muscle strength metrics	→	are evaluated by
psychomotoric aspect metrics	→	are evaluated by

In REHABROBO-CNL, the instances of the concepts are represented by one of their distinctive properties. For robots, this distinctive property is its name; for users, it is the user name. To illustrate, when the user wants to query about AssistOn-Shoulder, s/he specifies the instance using the name of the robot. For movements and metrics, there is no such distinctive property

Table 4.5: Types that can occur after the verbs

<i>Verb()</i>	(some all any* the)	<i>Type()</i>
target	(some all any)	all movements except leaf classes
target	the	leaf classes of movements
evaluate	(some all any)	all metrics except leaf classes
evaluate	the	leaf classes of metrics
are targeted by	(some all any)	robots
are evaluated by	(some all any)	robots
reference	(some all any)	robots
own	(some all any)	robots
are owned by	(some all any)	users

* *any* is used after negative verbs.

because the concept names are sufficient to specify a movement or metric. In fact, using `RELATIVECLAUSE` is sufficient to query about them. To query about robots or users, however, we use `INSTANCERELATION` to extract the instances. Table 4.6 demonstrates the relevant properties of the instances that appear when a type is selected.

Table 4.6: Instances that can occur after the types

<i>Type()</i>		<i>Instance()</i>
robot	→	name of the robot
user	→	username of the user

In addition to types and verbs, types are matched with the relevant nouns, as shown in Table 4.7. For instance, control modes are matched with robots whereas actuation is matched with movements. Further, the values for the nouns are extracted to allow suitable entries from the users. These values are listed in Table 4.8. The values can be considered as ranges of the nouns, that the user can choose from.

Table 4.7: Nouns that can occur after the types

<i>Type()</i>	with	<i>Noun()</i>
robots	→	active degree of freedom
robots	→	control modes
robots	→	disorder level
robots	→	functionality
robots	→	interaction type
robots	→	intervention time
robots	→	kinematic type
robots	→	motion capability
robots	→	name
robots	→	passive degree of freedom
robots	→	targeted disorder
robots	→	targeted population
movements	→	actuation
movements	→	backdrivability
movements	→	backdrivability type
movements	→	maximum range of motion
movements	→	minimum range of motion
movements	→	range of motion type
movements	→	transmission
publications	→	authors
publications	→	clinical study
publications	→	place of publication
publications	→	title
publications	→	url
publications	→	year
users	→	institution
users	→	mail
users	→	username

Table 4.8: Values that can occur after the nouns

<i>Noun()</i>	*	<i>Value()</i>
active DoF**	→	any integer value entered by the user
actuation	→	electrical, electro-rheological, hydrolic, pneumatic, series elastic, variable impedance, other
authors	→	one of the authors that are added to the ontology up to now
backdrivability	→	backdrivable, non-backdrivable
backdrivability type	→	active, passive
control modes	→	ADL, BCI, EMG, active, assistive, bilateral, multilateral, passive, resistive
disorder level	→	mild, moderate, severe
functionality	→	clinic,home
institution	→	one of the institutions that are added to the ontology up to now
interaction type	→	exoskeleton, mixed, suspension, end effector
intervention time	→	acute, chronic, subacute
kinematic type	→	hybrid, parallel, serial
motion capability	→	grounded, mobile
name	→	one of the robot names that are added to the ontology up to now
passive DoF	→	any integer value entered by the user
place of publication	→	one of the places of publication that are added to the ontology up to now
maximum RoM***	→	any float value entered by the user
minimum RoM	→	any float value entered by the user
RoM type	→	active, passive
targeted disorder	→	stroke, spine cord injury
targeted population	→	adult, pediatric
title	→	one of the publication titles that are added to the ontology up to now
transmission	→	belt drive, cable drive, capstan drive, direct drive, gear train, harmonic drive, other
url	→	one of the urls that are added to the ontology up to now
username	→	one of the usernames that are added to the ontology up to now
year	→	any year (integer value) entered by the user

* (EQCHECK|QUANTIFIER)

** degree of freedom

*** range of motion

4.2 Transforming a Query in REHABROBO-CNL to a SPARQL Query

To answer a query in REHABROBO-CNL, we transform the query into a SPARQL query with the following steps.

1. We parse the query and form a query description tree.
2. We traverse the tree and obtain a DL concept description.
3. We transform the DL concept into a SPARQL concept.
4. We form a SPARQL query.

Query Description Trees (QDT)

We use a rooted, directed tree, called query description tree (QDT), to parse the REHABROBO-CNL query entered by the user. In this tree, there are five types of nodes:

- root-node: Represents the sort of the query.
- that-node: Represents a relative clause beginning with “that”.
- with-node: Represents a relative clause beginning with “with”.
- and-node: Represents a conjunction.
- or-node: Represents a disjunction.

Every root/that/with-node characterizes a phrase and a type/instance. An and/or-node cannot be a leaf. For each path from the root node to a leaf

node, there can be at most one and-node and one or-node. With-nodes are leaves only. That-node has one child only.

Consider, for instance, the QDT in Figure 4.1 for the query: “What are the robots that target some shoulder movements with actuation=’electrical’ and (with transmission=’cable drive’ or with transmission=’direct drive’)?”

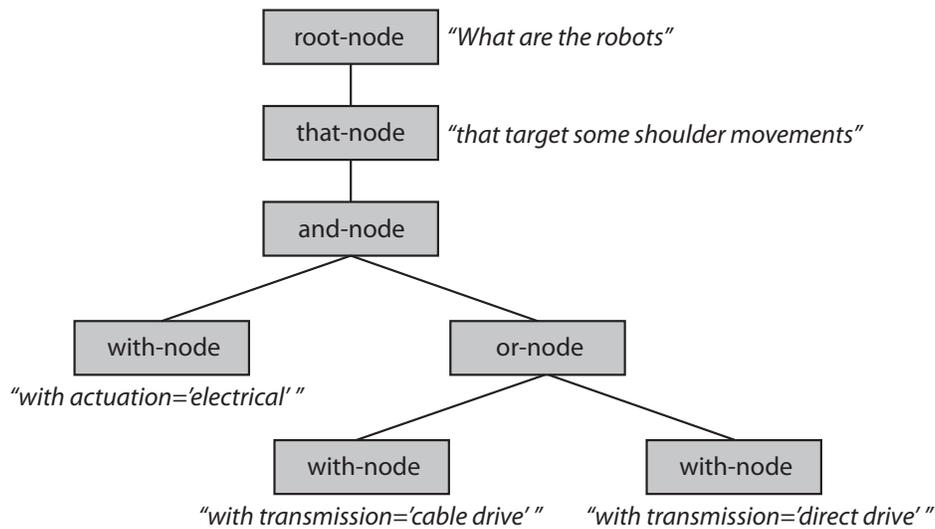


Figure 4.1: Tree representation of the sample query.

This tree is constructed online while the user expresses the query by the help of the user interface. The root denotes the beginning of the query “What are the robots...”. According to the root, the answer to the query will contain robot names only. Since the query is about robots, the type contained in the root is “robot”.

The relative clause about these robots is the child of the root, and since this relative clause starts with “that”, it is a that-node. The type contained in this node is “shoulder movement”.

The query continues with a conjunction of relative clauses, including a simple disjunction. Clauses joined with a conjunction (resp., disjunction) are

characterized by an and-node (resp., or-node) as their parent. Since these relative clauses start with “with”, they are with-nodes. They include values of properties instead of types.

From QDT to a DL concept

The tree representing the query, in fact, represents a concept. While creating a query, we define a new concept and search for its instances. Retrieved instances that fit our description are the answers to our query.

By a depth-first traversal of a QDT, we represent the corresponding concept in Description Logics (DL) as presented in Algorithm 1. Algorithm 2 demonstrates the transformation for a that-node and Algorithm 3 demonstrates the transformation for a with-node.

As the algorithm traverses a QDT, according to the types of nodes, it generates parts of the DL concept. For instance, let us explain Algorithm 1 by an example. Suppose that the input is the tree in Figure 4.1. It starts from the root node and enters the first if condition. Since the associated class of the node is “robot”, our concept description starts with `Robot`. Then, the algorithm calls *transform* recursively for each child of the root node. In the first recursive call, since the current node is a `that-node`, the algorithm calls *transformThatNode* (Algorithm 2) and passes `that-node` as an input. Since it has a child node, the first condition of having children is satisfied. Next, the algorithm calls *transform* for its child node, `and-node`, whose children are conjoined after each transformation. The first child of `and-node` is transformed into an existential restriction by calling *transformWithNode* (Algorithm 3). The second child of `and-node` contains a simple disjunction. The algorithm transforms the information in the children of `or-node` into

existential restrictions and disjoins them. Finally, the algorithm finishes recursions and returns to **that-node**, covers the transformation of its children with brackets, and transforms **that-node** into an existential restriction. The resulting DL concept that is returned from the algorithm is as follows.

$$\text{Robot} \sqcap \exists \text{targets.}(\text{ShoulderMovements} \sqcap \\ (\exists \text{actuation.}\{\text{electrical}\}) \sqcap \\ (\exists \text{transmission.}\{\text{cabledrive}\} \sqcup \exists \text{transmission.}\{\text{directdrive}\})))$$

Algorithm 1: transform

Input : A tree T representing the concept that the user described
Output: A DL concept description Q that represents the concept in T
// $n.class$ denote associated class of a node n
// $n.children$ denote children of a node n
 $Q \leftarrow \emptyset$;
 $n \leftarrow$ first (root) node in T ;
if n is a root-node **then**
| $Q \leftarrow Q \sqcap n.class$;
| **foreach** child node $c \in n.children$ **do**
| | $Q \leftarrow Q \sqcap \text{transform}(c)$;
else if n is a that-node **then**
| $Q \leftarrow Q \sqcap \text{transformThatNode}(n)$;
else if n is a with-node **then**
| $Q \leftarrow Q \sqcap \text{transformWithNode}(n)$;
else if n is an and-node OR n is an or-node **then**
| $tempQ \leftarrow \emptyset$;
| **foreach** child node $c \in n.children$ **do**
| | **if** n is an and-node **then**
| | | $tempQ \leftarrow tempQ \sqcap \text{transform}(c)$;
| | **else**
| | | $tempQ \leftarrow tempQ \sqcup \text{transform}(c)$;
| | $Q \leftarrow Q \sqcap (tempQ)$;
return Q

Algorithm 2: transformThatNode

Input : A that-node n

Output: A DL concept description Q that represents the concept in n

// $n.class$ denote associated class of a node n

// $n.verb$ denote associated verb of a node n

// $n.negative$ denote the negativity of a node n

// $n.quantifier$ denote the quantifier of a node n

// $n.instance$ denote the instance of a node n , if exists

// $n.child$ denote child of a node n

// $n.class.identifierNoun$ denote the noun that identifies $n.class$

$Q \leftarrow \emptyset$;

$childQ \leftarrow \emptyset$;

if $n.child$ is not empty **then**

└ $childQ \leftarrow transform(n.child)$;

else if n includes an instance **then**

└ $childQ \leftarrow \exists(n.class.identifierNoun).\{n.instance\}$;

if $n.quantifier = ALL$ **then**

└ **if** $n.verb$ is passive **then**

└ $Q \leftarrow Q \sqcap \forall(n.verb)^{\neg} . ((n.class) \sqcap childQ)$;

└ **else**

└ $Q \leftarrow Q \sqcap \forall(n.verb) . ((n.class) \sqcap childQ)$;

else

└ // If there is no quantifier or the quantifier is SOME

└ **if** $n.verb$ is passive **then**

└ $Q \leftarrow Q \sqcap \exists(n.verb)^{\neg} . ((n.class) \sqcap childQ)$;

└ **else**

└ $Q \leftarrow Q \sqcap \exists(n.verb) . ((n.class) \sqcap childQ)$;

if $n.negative = True$ **then**

└ $Q \leftarrow \neg Q$;

return Q

Algorithm 3: transformWithNode

Input : A with-node n
Output: A DL concept description Q that represents the concept in n
// $n.noun$ denote associated noun of a node n
// $n.values$ denote the list of values of a node n
// $n.quantifier$ denote the quantifier of a node n
// $n.aggregator$ denote the aggregator of a node n
// $n.datatype$ denote datatype of the noun in a node n
 $Q \leftarrow \emptyset$;
if $n.noun$ is functional **then**
 if $n.datatype = \text{boolean}$ **then**
 $Q \leftarrow Q \sqcap \exists(n.noun).\{n.values_0\}^{\wedge}xsd : \text{boolean}$ };
 else
 if $n.aggregator = \geq$ or $n.aggregator = \leq$ **then**
 $Q \leftarrow Q \sqcap \exists(n.noun).(n.aggregator)_{n.values_0}$;
 else if $n.aggregator = =$ **then**
 $Q \leftarrow Q \sqcap \exists(n.noun).\{n.values_0\}$;
 else if $n.aggregator = ! =$ **then**
 foreach value $v \in n.values$ **do**
 $Q \leftarrow Q \sqcap \neg \exists(n.noun).\{v\}$;
 else
 if $n.quantifier = NO$ **then**
 $Q \leftarrow Q \sqcap \neg \exists(n.noun).xsd : (n.datatype)$;
 else if $n.quantifier = ALL$ **then**
 $Q \leftarrow Q \sqcap \forall(n.noun).xsd : (n.datatype)$;
 else if $n.quantifier = SOME$ **then**
 $Q \leftarrow Q \sqcap \exists(n.noun).xsd : (n.datatype)$;
 else
 if $n.aggregator = =$ **then**
 foreach value $v \in n.values$ **do**
 $Q \leftarrow Q \sqcap \exists(n.noun).\{v\}$;
 else if $n.aggregator = ! =$ **then**
 foreach value $v \in n.values$ **do**
 $Q \leftarrow Q \sqcap \neg \exists(n.noun).\{v\}$;
return Q

From a DL concept to a SPARQL concept

To obtain a SPARQL concept from a DL concept, we utilize some of the existing translations in related publications, such as [53] and [24]. We also introduce some novel transformations that are not covered by other work. Some transformation examples are shown in Table 4.9. The transformations without a citation are novel transformations introduced in this thesis.

Novel transformations include the transformation of inverse roles. Consider the inverse role example in Table 4.9. DL representation of this concept corresponds to the following first-order formula.

$$\exists x.target(x, y) \wedge Robot(x)$$

In this formula, we state that “there exists a robot x that targets a movement y ”. Our transformation to SPARQL includes two triples, having a common variable x . The variable x should satisfy two conditions: it must be a robot and it must target a movement y . According to the semantics of AND operator (denoted with a dot) in Section 2.3, the result contains the mappings of x and y to the nodes in the ontology, that agree on the nodes that correspond to x . This corresponds to the existential restriction in the first-order formula, that should satisfy two conditions combined with a conjunction. Therefore, evaluation of the DL concept and the SPARQL concept will return the same result.

Consider the complement example in Table 4.9. DL representation of this concept contains a negated existential quantifier. SPARQL transformation contains a triple covered with `FILTER NOT EXISTS`. The triple searches for a mapping of variable x to a node, that is related to another node `AssistOn` with an edge that characterizes `has_Name` relation. This corresponds to an existential restriction. However, we do not want such mappings of x . Therefore,

according to the semantics of `NOT EXISTS` in a filter expression (explained in Section 2.3), `FILTER NOT EXISTS {C}` is satisfied if the mapping of `C` is an empty set. Therefore, there should not be any mapping of the variables in `C` to a node in the ontology. The result that is returned from our SPARQL concept will not contain any node that satisfies the condition in the triple, and that corresponds to a negated existential restriction: all `x` must not have name `AssistOn`. Therefore, evaluation of the DL concept and the SPARQL concept will return the same result.

Finally, consider the universal restriction example in Table 4.9. DL description of this concept represents the publications that reference all robots, and for that, it contains a universal quantifier. To represent this concept with SPARQL, we need to describe such publications by making sure that there is no robot in the ontology that is not referenced by that publication. To describe such publications in SPARQL, we use an expression constructed with two `FILTER NOT EXISTS`. Since a universal restriction such as $\forall x A(x)$ corresponds to a negated existential restriction $\neg \exists x \neg A(x)$, each `FILTER NOT EXISTS` operator in the SPARQL query corresponds to a negation.

We first use two triples to represent a publication that references a robot. Then, we refer to that publication with its variable, `x`. We also use these triples to make sure that the query does not return an answer if there is no robot in the ontology. In such a case, even though the remaining `FILTER` expression is satisfied, the set of mappings for `x` will be an empty set because none of the publications in the ontology could reference a robot.

The first `FILTER NOT EXISTS` expression contains another `FILTER NOT EXISTS` expression, which contains a triple that represents the robots `y2` referenced by the previously described publication `x`. There is a triple in the

first expression as well, that states y_2 is a robot. Since both expressions contain the same variable, the mappings should agree on the robot y_2 . In addition, the set of mappings for y_2 must be an empty set to satisfy the inner `FILTER NOT EXISTS` expression. The outer `FILTER NOT EXISTS` expression then contains the instances of the robots (mappings for y_2) that are not referenced by the publication x . The set of such instances must be empty to satisfy this expression. Otherwise, the query does not return an answer. Therefore, we represent a universal restriction as a negated existential restriction in SPARQL using its operators.

Let us explain the transformation of a DL concept to a SPARQL concept by explaining it over our DL concept. The following DL concept is the concept description that we obtained from our example query.

```
Robot  $\sqcap$   $\exists$ targets.(ShoulderMovements $\sqcap$ 
  ( $\exists$ actuation.{electrical}) $\sqcap$ 
  ( $\exists$ transmission.{cabledrive}  $\sqcup$   $\exists$ transmission.{directdrive})))
```

First, we transform the concept `Robot`. For that, we assign a variable for robot, and specify its type, `RehabRobots`:

```
?robot1 rdf:type rr:RehabRobots.
```

Second, we transform the existential restriction `\exists targets.(ShoulderMovements)`.

The transformation of an existential restriction results in two triples. The first triple is about the relation, and the second triple is about the type of the second variable in the relation. With the following triple, we say that our robot targets a movement of type `ShoulderMovements`.

```
?robot1 rr:targets ?movement1.
?movement1 rdf:type rr:ShoulderMovements.
```

Then, the concept description contains a `hasValue` restriction, which is transformed into one triple. The triple specifying that the targeted movement has an electrical actuation is as follows.

```
?movement1 rr:has_Actuation 'electrical'.
```

Finally, we transform the simple disjunction that contains two `hasValue` restrictions as follows. We combine the triples with `UNION`, a keyword that SPARQL provides for disjunctions.

```
{?movement1 rr:has_Transmission 'cable drive'.}  
UNION  
{?movement1 rr:has_Transmission 'direct drive'.}
```

The resulting SPARQL concept:

```
?robot1 rdf:type rr:RehabRobots.  
?robot1 rr:targets ?movement1.  
?movement1 rdf:type rr:ShoulderMovements.  
?movement1 rr:has_Actuation 'electrical'.  
{?movement1 rr:has_Transmission 'cable drive'.}  
UNION  
{?movement1 rr:has_Transmission 'direct drive'.}
```

Then, we can construct a SPARQL query as follows. We start with a `PREFIX` part and we declare the namespace (the location of an ontology on the Web) of `REHABROBO-ONTO`. Next, we continue with a `SELECT` clause. The instances of type `Robot`, by themselves, are not meaningful to the users. Thus, we want to display the names of the instances to the users. We specify

Table 4.9: DL to SPARQL Transformation Examples

Constructor	DL	SPARQL
Concept [24]	Robot	?x rdf:type ns:RehabRobots.
Role [53]	targets	?x ns:targets ?y.
Complement	$\neg \exists \text{name}.\{\text{AssistOn}\}$	FILTER NOT EXISTS { ?x ns:has_Name 'AssistOn'. }
Inverse Role	$\exists \text{targets}^-.\text{Robot}$?x ns:targets ?y. ?x rdf:type ns:RehabRobots.
Existential Restriction [53]	$\exists \text{targets}.\text{ShoulderMovements}$?x ns:targets ?y. ?y rdf:type ns:ShoulderMovements.
hasValue Restriction [53]	$\exists \text{name}.\{\text{AssistOn}\}$?x ns:has_Name 'AssistOn'.
Universal Restriction	$\forall \text{reference}.\text{Robot}$?x rr:reference ?y. ?y rdf:type rr:RehabRobots. FILTER NOT EXISTS { FILTER NOT EXISTS { ?x rr:reference ?y2. ?y2 rdf:type rr:RehabRobots.}
Intersection [24]	$\text{Robot} \sqcap \exists \text{functionality}.\{\text{clinic}\}$?x rdf:type ns:RehabRobots. ?x ns:has_Functionality 'clinic'.
Union [24]	$\exists \text{functionality}.\{\text{clinic}\} \sqcup \exists \text{motionCapability}.\{\text{grounded}\}$	{?x ns:has_Functionality 'clinic'.} UNION {?x ns:has_Motion_Capability 'grounded'.}

it with an additional triple in the beginning of the WHERE clause, and continue the clause with the transformed SPARQL concept:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
```

```
PREFIX rr: <http://www.semanticweb.org/ontologies/2012/RehabOnto.owl#>
```

```
SELECT DISTINCT ?name
```

```
WHERE {
```

```
  ?robot1 rr:has_Name ?name.
```

```
  ?robot1 rdf:type rr:RehabRobots.
```

```
  ?robot1 rr:targets ?movement1.
```

```

?movement1 rdf:type rr:ShoulderMovements.
?movement1 rr:has_Actuation 'electrical'.
{?movement1 rr:has_Transmission 'cable drive'.}
UNION
{?movement1 rr:has_Transmission 'direct drive'.}
}

```

In Appendix A, the transformation of each query in Table 4.1 is demonstrated step by step, including its CNL, tree, DL and SPARQL representations; and the answers to the queries computed by PELLET.

4.3 Answering Queries Using Pellet

We use the DL reasoner PELLET to find answers to queries, through the Jena framework. With PELLET, it is possible

- to check the consistency of REHABROBO-ONTO, and
- to generate explanations in the case of inconsistencies, and
- to query REHABROBO-ONTO with SPARQL.

Consider, for instance, the query Q1 from Table 4.1, whose SPARQL representation is as follows.

```

SELECT DISTINCT ?name
WHERE {
  ?robot1 rr:has_Name ?name.
  ?robot1 rdf:type rr:RehabRobots.
  ?robot1 rr:targets ?movement1.
}

```

```

?movement1 rdf:type rr:WristMovements.
?movement1 rr:has_Actuation 'series elastic'.
}

```

After loading REHABROBO-ONTO into PELLET, we present this SPARQL query to PELLET, and get the following answer:

```
( ?name = "AssistOn-Mobile" )
```

Consider, for instance, another query, Q10 from Table 4.1:

What are the publications with place of publication 'ICORR' and that reference some robots that are owned/maintained by some users with institution 'Sabanci University' ?

SPARQL representation of this query is as follows.

```

SELECT DISTINCT ?name
WHERE {
  ?publication1 rr:has_Title ?name.
  ?publication1 rdf:type rr:References.
  ?publication1 rr:has_PublishedAt 'ICORR'.
  ?robot1 rr:hasReference ?publication1.
  ?robot1 rdf:type rr:RehabRobots.
  ?robot1 rr:ownedBy ?user1.
  ?user1 rdf:type rr:Owners.
  ?user1 rr:has_Institution 'Sabanci University'.
}

```

We get the following answers from PELLET to this query:

```
( ?name = "Brain Computer Interface based Robotic Rehabilitation  
with Online Modification of Task Speed" )  
( ?name = "Passive Velocity Field Control of a Forearm-Wrist  
Rehabilitation Robot" )  
( ?name = "Design of a reconfigurable ankle rehabilitation robot  
and its use for the estimation of the ankle impedance" )
```

In order to prevent inconsistencies in REHABROBO-ONTO, we do consistency checks with PELLET. A consistency check can be done before adding an assertion to REHABROBO-ONTO, or before querying REHABROBO-ONTO.

An example consistency check in Java is demonstrated in Figure 4.2. Here, we load the terminology and all of the assertions into the reasoner. Then, PELLET creates an inference graph and traces the ontology to pinpoint a contradictory fact, if exists in the ontology. Since REHABROBO-ONTO is consistent, PELLET does not enter the inconsistency condition and displays a message stating that the ontology is consistent.

Assume that one of the assertions about a robot causes inconsistency in REHABROBO-ONTO, and it is about the motion capability of that robot. We defined motion capability as a functional property in REHABROBO-ONTO; that is, a robot must have at most one motion capability. Suppose that a robot has both “grounded” and “mobile” motion capabilities, even though entering such information is prevented in REHABROBO-QUERY, by providing radiobuttons for functional properties. PELLET can detect the statements in the ontology that cause an inconsistency, as shown in Figure 4.3. We can instruct PELLET to display the inconsistent statements in OWL/XML format (as specified in Figure 4.2), or we can iterate over the statements one by one in triples as shown in Figure 4.3. PELLET can also detect inconsistencies in

```

416
417 OntModel rehabOnto = ModelFactory.createOntologyModel( PelletReasonerFactory.THE_SPEC );
418 rehabOnto.read("http://"+hostName+"/rehabrobo/RehabOnto.owl"); //Load TBox
419 rehabOnto.addSubModel(owners); //Load owners
420 for(int i=0;i<robots.length;i++) //Load ABox
421     rehabOnto.addSubModel(robots[i]);
422
423 //Create a reasoner and load the ontology to create an inference graph for pellet
424 PelletOptions.USE_TRACING = true;
425 Reasoner pelletReasoner = PelletReasonerFactory.theInstance().create();
426 InfModel infModel = ModelFactory.createInfModel(pelletReasoner, rehabOnto);
427 PelletInfGraph pellet = (PelletInfGraph) infModel.getGraph();
428
429 //If ontology is not consistent, explain; otherwise, display success message
430 if(!pellet.isConsistent()){
431     Model explanation = pellet.explainInconsistency();
432     explanation.write(System.out);
433 }
434 else
435     System.out.println("Ontology is consistent.");
436

```

The screenshot shows an IDE window with a Java code editor and a console. The code defines an ontology model, loads TBox and ABox, creates a Pellet reasoner, and checks for consistency. The console output shows "Ontology is consistent."

Figure 4.2: Consistency check for REHABROBO-ONTO.

TBox and pinpoint the unsatisfiable concepts.

4.4 Intelligent User Interface for REHABROBO-QUERY

REHABROBO-QUERY allows users to express queries in REHABROBO-CNL, about rehabilitation robots, by the help of its intelligent and interactive user interface.

The main user interface for querying includes a drop-down list, showing the possible ways to begin a query. Then, according to the user’s choices, it provides different types of features.

REHABROBO-QUERY provides auto-completion to help users enter values for nouns that correspond to data properties of type string.

If the user should choose a concept among a hierarchy, then REHABROBO-QUERY displays an accordion view and enables the user to click on the option

```

423
424 //Create a reasoner and load the ontology to create an inference graph for pellet
425 PelletOptions.USE_TRACING = true;
426 Reasoner pelletReasoner = PelletReasonerFactory.theInstance().create();
427 InfModel infModel = ModelFactory.createInfModel(pelletReasoner, rehabOnto);
428 PelletInfGraph pellet = (PelletInfGraph) infModel.getGraph();
429
430 //If ontology is not consistent, explain; otherwise, display success message
431 if(!pellet.isConsistent()){
432     Model explanation = pellet.explainInconsistency();
433     // iterate over the axioms in the explanation
434     for( Statement statement : explanation.listStatements().toList() ) {
435         System.out.println(statement.asTriple().toString( infModel ) );
436     }
437 }
438 else
439     System.out.println("Ontology is consistent.");
440
441

```

The screenshot shows an IDE window with a Java code editor and a console. The code defines a Pellet reasoner, loads an ontology, and checks for consistency. If inconsistent, it prints the explanation; otherwise, it prints a success message. The console output shows the ontology is consistent and lists some properties and instances.

```

RehabOnto:has_Motion_Capability @rdf:type :FunctionalProperty
RehabOnto:R3 @RehabOnto:has_Motion_Capability "grounded"
RehabOnto:R3 @RehabOnto:has_Motion_Capability "mobile"

```

Figure 4.3: Explanation generation with PELLET.

s/he wants.

In addition, REHABROBO-QUERY allows multiple selection of values for relational properties. For functional properties, user is able to select multiple items for inequality. User can choose a number of options among “less than or equal”, “more than or equal”, “equal” and “not equal” while entering values for a data property of type integer or float.

In Figures 4.4 and 4.5, constructing the query Q1 with REHABROBO-QUERY is shown.

How the results of a query is displayed to user depends on the query. For instance, if the query is about robots, then the user sees the names of the retrieved robots. If the query is about movements or metrics, then the user sees the concept names instead of the instance URIs which would make no sense to the user. In Table 4.10 the answers corresponding to the starting

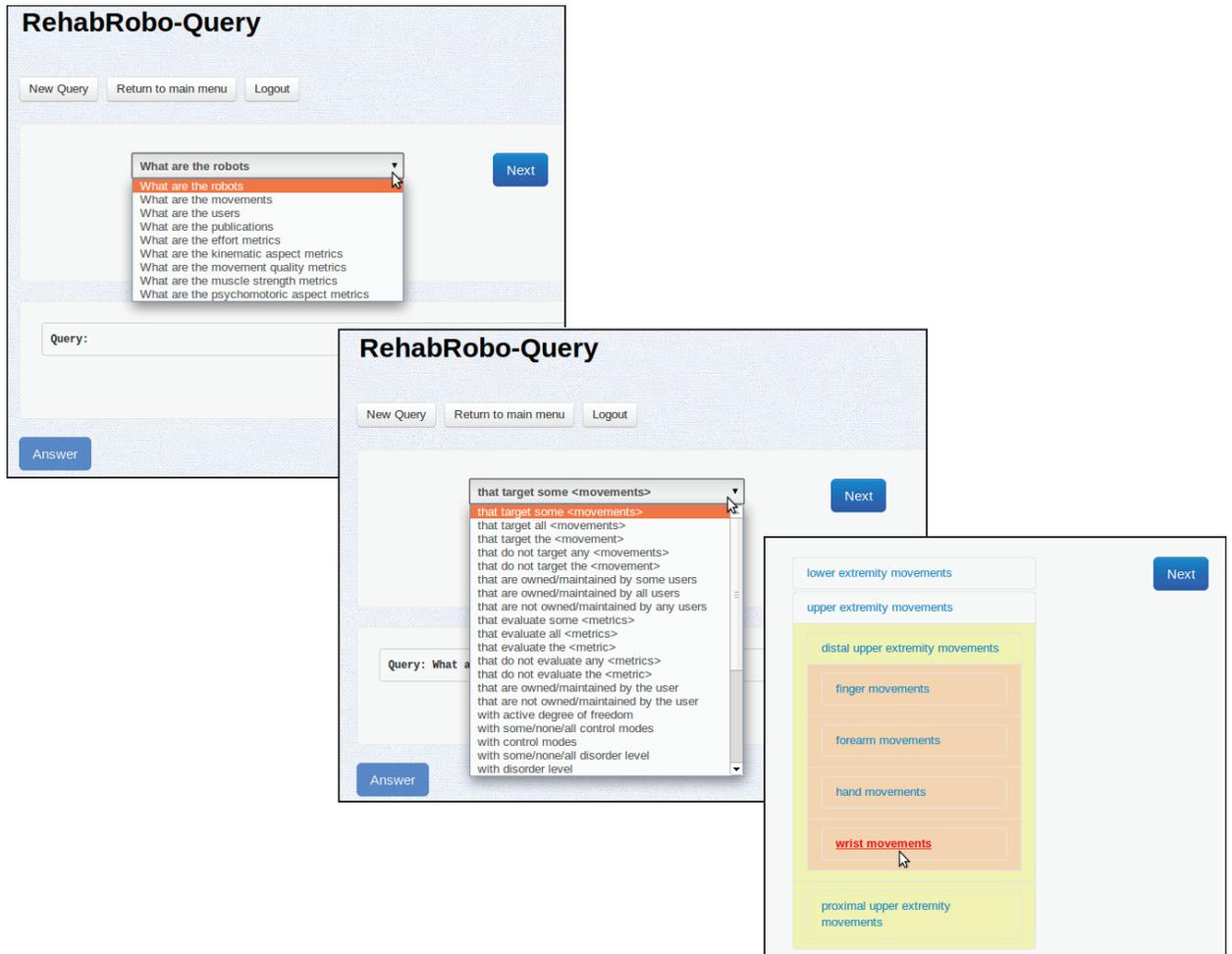


Figure 4.4: Constructing Q1 (1).

type of the queries are shown.

The answer to this query is shown in Figure 4.6. As seen from this figure, it is possible to click on the robot name. The user can click on the robot name to see further properties of this robot.

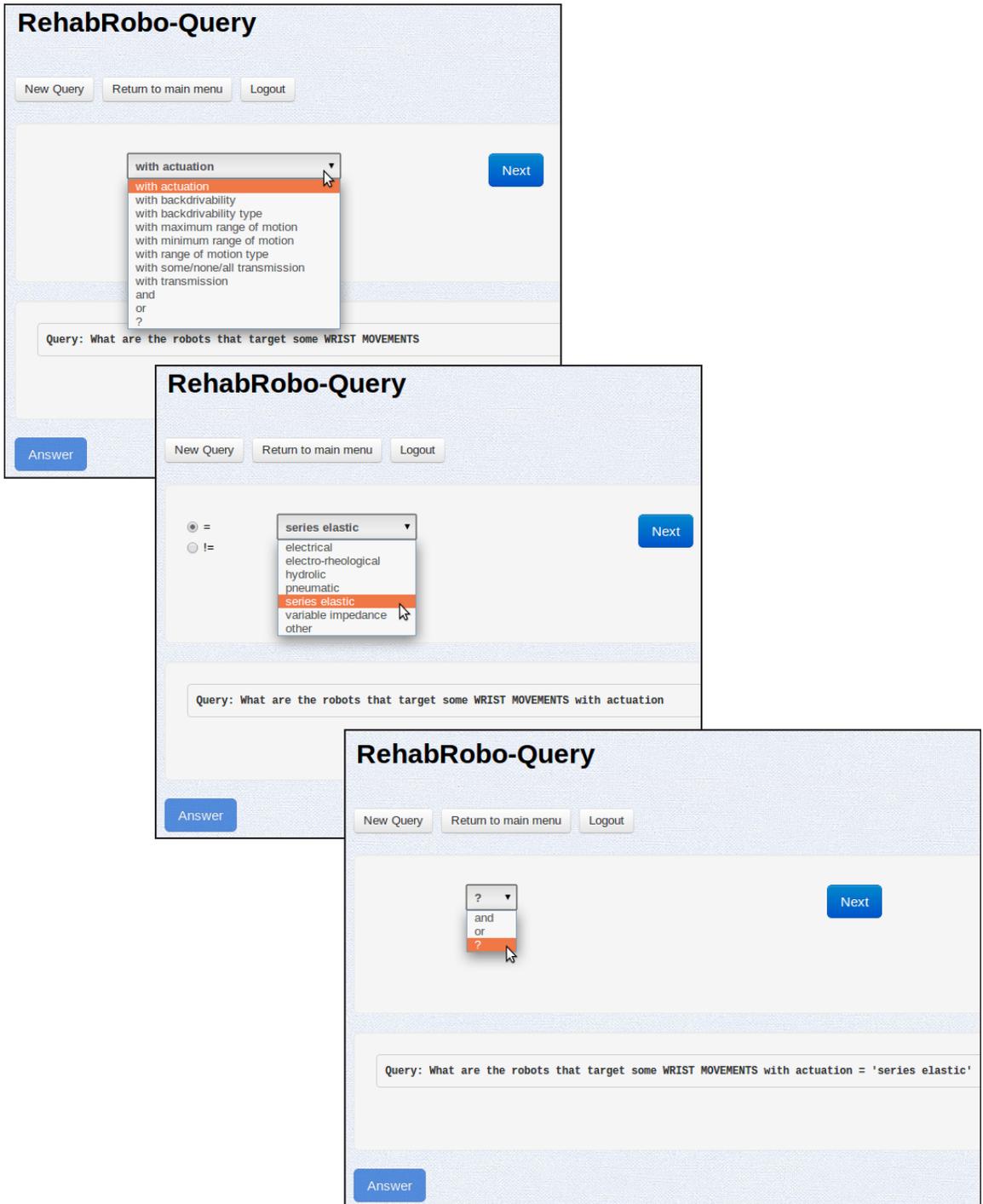


Figure 4.5: Constructing Q1 (2).

Table 4.10: Answers for the query types

Q*	Type()	Answer
	robots	→ names of the robots
	movements	→ leaf classes of the movements
	users	→ usernames of the users
	publications	→ the titles of the publications
	effort metrics	→ leaf classes of the effort metrics
	kinematic aspect metrics	→ leaf classes of the kinematic aspect metrics
	movement quality metrics	→ leaf classes of the movement quality metrics
	muscle strength metrics	→ leaf classes of the muscle strength metrics
	psychomotoric aspect metrics	→ leaf classes of the psychomotoric aspect metrics

* Q represents the beginning of the query: “What are the”



Figure 4.6: Answer to Q1.

Chapter 5

5 Interoperability of REHABROBO-ONTO

Having a structured formal representation of knowledge about rehabilitation robots allows reasoning (e.g., answering complex queries) that requires integration with other knowledge resources. We consider two existing ontologies. The first one is Foundational Model of Anatomy (FMA) [60], which is an ontology about human anatomy. The second related domain is Human Disease Ontology (DO) [54], which provides a hierarchy for human diseases.

FMA is first created as a MySQL relational database, then there had been ongoing efforts to convert FMA into a DL-based representation in OWL, to stay close to the original representation and to represent the knowledge correctly. The ontology is open source and its OWL representation is available online.

DO is also initially developed as a relational database, then made available in obo format ¹¹. Currently, its OWL representation is available online, which is created using a conversion script obo2owl, owned by The Open Biological and Biomedical Ontologies (OBO) Foundry.

¹¹http://www.geneontology.org/GO.format.obo-1_2.shtml

5.1 Integration with FMA

Below are example queries (FMA1-FMA3) that can be asked over FMA and REHABROBO-ONTO. Under each query, additional information about the concepts in FMA, the rules in Semantic Web Rules Language (SWRL) [33] to integrate relevant concepts, SPARQL representations of the queries, and the answers to the queries are presented. Queries FMA4 -FMA8 are presented in Appendix B.

SWRL combines OWL DL with the Unary/Binary Datalog RuleML sub-language. A SWRL rule contains two parts. They are called body and head, that constitute the antecedent part and the consequent part, respectively. According to the human readable syntax in [33], where the syntax and semantics of SWRL are described, a SWRL rule has the form as follows.

antecedent \rightarrow **consequent**

As seen from the form above, the rules in SWRL are in the form of an implication. Therefore, if the conditions in the body are true, then the conditions in the head are also true. The conditions are constituted using zero or more atoms; and multiple atoms are conjoined. The atoms that we consider in this thesis have the form $C(?x)$ where C is a concept and x is a variable that corresponds to instances. Atoms can be of other forms, such as properties; however, we do not consider them in this thesis and refer to [33] for further information.

The variables in a rule are universally quantified, and their scope are limited to the corresponding rule. According to the safety condition, the variables in the head of a rule must occur in the body.

We define the semantics of a SWRL rule as follows [33]. An interpretation I is a tuple $I = \langle R, EC, S \rangle$, where R is a set of resources, EC is a mapping

from concepts to subsets of R , and S is a mapping from names of the instances to elements of $EC(owl : Thing)$. A binding $B(I)$ extends an interpretation I such that S maps variables to elements of $EC(owl : Thing)$. Let C be an OWL concept and x a variable, then an atom $C(x)$ is satisfied by an interpretation I under the condition of $S(x) \in EC(C)$.

A binding $B(I)$ satisfies a body A (resp., head C) iff $B(I)$ satisfies every atom in A (resp., C). An interpretation I satisfies a rule iff every binding $B(I)$ that satisfies the body also satisfies the head.

It is possible to add rules in SWRL into an ontology in OWL, using PROTÉGÉ ontology editor. In the examples below, we demonstrate the rules window of PROTÉGÉ from where we add rules. In order to add rules that contain concepts of different ontologies, we import the relevant ontologies. Then, PROTÉGÉ allows storing the rules in a separate file from the ontologies. To ask queries over the ontologies and the rules, we utilize DL reasoner PELLET. We load the ontologies and the relevant files that keep the rules. Then we query over instances as shown in the examples below.

FMA1. What are the body parts that can be affected by some forearm robots?

In order to answer FMA1, we can add a rule in SWRL to integrate some parts that can be affected by forearm pronation/supination movement. For instance, “Ulna” or “Radius” are modeled as constitutional parts of “Forearm”, and regional parts of “Skeleton of forearm” in FMA (Figure 5.1). In the figures, we label the edges that relate the relevant concepts as “related”, for illustration purposes.

Both “constitutional_part_of” and “regional_part_of” are properties that are subproperties of “part_of”. The rules that integrate concepts **Forearm**

and `Skeleton of forearm` in FMA with the concept `ForearmMovements` according to Figure 5.1 are shown in Figure 5.2. These concepts are integrated with an SWRL rule, to state that the instances of `ForearmMovements` are also instances of `Forearm` and `Skeleton of forearm`. We consider the concepts of REHABROBO-ONTO in the body of the rules, and the concepts of the other ontologies in the head. However, it is possible to add rules that contain concepts of the other ontology in the body whereas the concepts of REHABROBO-ONTO are in the head. Figure 5.2 demonstrates both types of rules.

The following SPARQL query corresponds to FMA1. With the implication rule stating that the instances of forearm pronation/supination movement are related to concepts about forearm, we can obtain related concepts in FMA using the variable of the related movement and `rdf:type` predicate.

```
SELECT DISTINCT ?bodyPartLabel
WHERE {
    ?robot rdf:type rr:ForearmRobots.
    ?robot rr:targets ?movement.
    ?movement rdf:type rr:JointMovements.
    ?movement rdf:type ?fmaConcept.
    ?bodyPart rdfs:subClassOf ?restriction.
    ?restriction owl:onProperty fma:part_of.
    ?restriction owl:someValuesFrom ?fmaConcept.
    ?bodyPart rdfs:label ?bodyPartLabel.
}
```

The answer to FMA1, with the code snippet that presents the query to PELLET for execution, is shown in Figure 5.3. In this code snippet, the parts

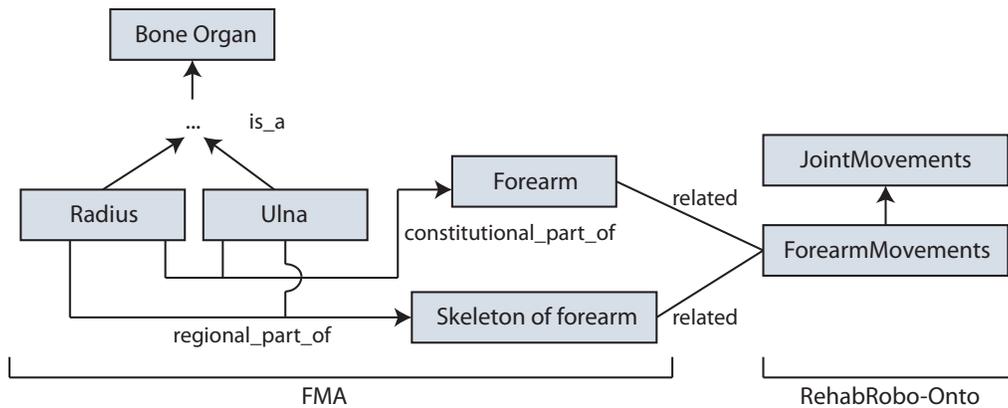


Figure 5.1: Hierarchy and integration of concepts for FMA1.

of the SPARQL query that relate the integrated concepts are highlighted with a circle.

FMA2. What are the rehabilitation robots that do not affect a joint under synovial joint of free limb segment?

```

SELECT DISTINCT ?robotName
WHERE {
  ?robot rdf:type rr:RehabRobots.
  ?robot rr:has_Name ?robotName.
  FILTER NOT EXISTS{
    ?robot rr:targets ?movement.
    ?movement rdf:type rr:JointMovements.
    ?movement rdf:type ?fmaConcept.
    ?fmaConcept rdfs:subClassOf ?superClass.
    ?superClass rdfs:label 'Synovial joint of free limb segment'.
  }
}

```

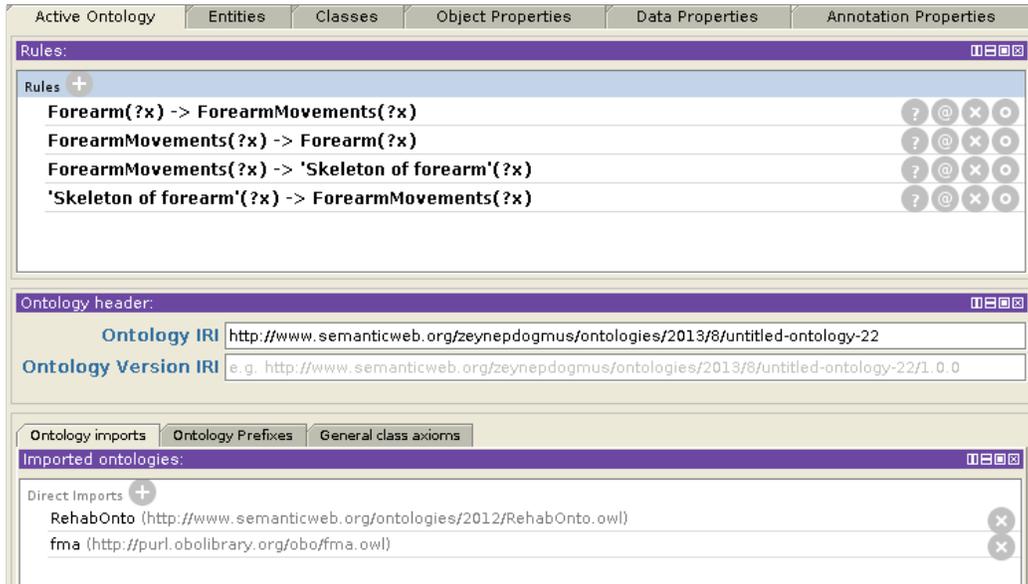


Figure 5.2: Importing ontologies and adding SWRL rules to answer FMA1.

“Synovial joint of free limb segment” concept in FMA has subclasses such as “Elbow joint”, “Knee joint”, “Wrist joint”, “Ankle joint”. They can be integrated with the concepts “ElbowMovements”, “KneeMovements”, “WristMovements” and “AnkleMovements” in REHABROBO-ONTO (Figure 5.4). The rules that integrate relevant concepts for FMA2 are shown in Figure 5.5.

With SWRL, it is also possible to add rules that define new concepts based on the existing concepts of the different ontologies. For instance, defining a new concept called “AffectedElbowParts” is shown in Figure 5.6. We define this new concept with the concepts of FMA and REHABROBO-ONTO, using conjunction.

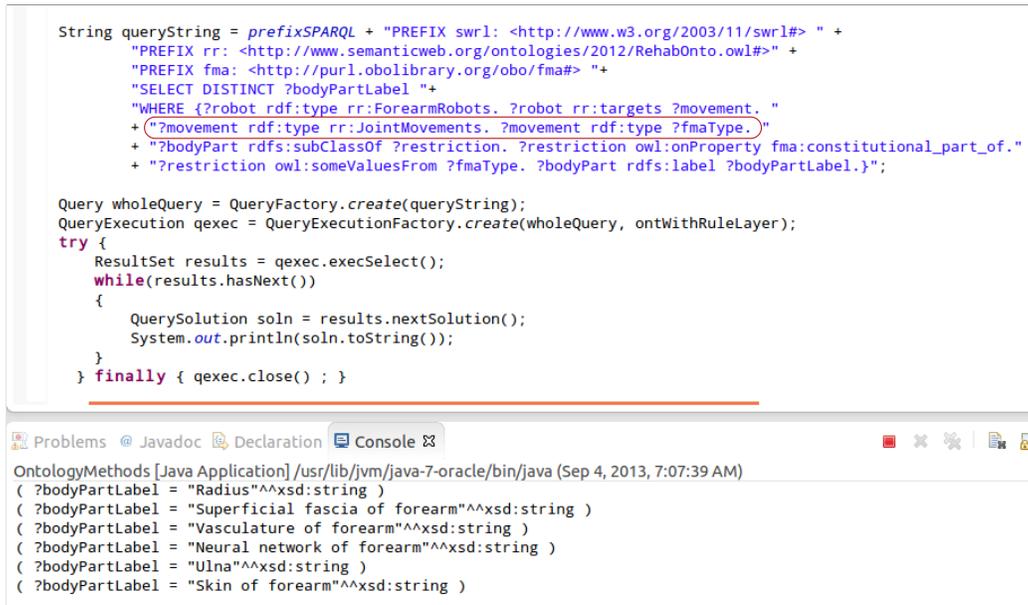
The answer to FMA2, with the code snippet that presents the query to PELLET for execution, is shown in Figure 5.7. The answer includes robots that do not target any movements that can affect elbow, knee, wrist or ankle joints.

```

String queryString = prefixSPARQL + "PREFIX swrl: <http://www.w3.org/2003/11/swrl#> " +
    "PREFIX rr: <http://www.semanticweb.org/ontologies/2012/RehabOnto.owl#> " +
    "PREFIX fma: <http://purl.obolibrary.org/obo/fma#> "+
    "SELECT DISTINCT ?bodyPartLabel "+
    "WHERE {?robot rdf:type rr:ForearmRobots. ?robot rr:targets ?movement. "
    + ("?movement rdf:type rr:JointMovements. ?movement rdf:type ?fmaType.")
    + "?bodyPart rdfs:subClassOf ?restriction. ?restriction owl:onProperty fma:constitutional_part_of."
    + "?restriction owl:someValuesFrom ?fmaType. ?bodyPart rdfs:label ?bodyPartLabel.}";

Query wholeQuery = QueryFactory.create(queryString);
QueryExecution qexec = QueryExecutionFactory.create(wholeQuery, ontWithRuleLayer);
try {
    ResultSet results = qexec.execSelect();
    while(results.hasNext())
    {
        QuerySolution soln = results.nextSolution();
        System.out.println(soln.toString());
    }
} finally { qexec.close() ; }

```



```

OntologyMethods [Java Application] /usr/lib/jvm/java-7-oracle/bin/java (Sep 4, 2013, 7:07:39 AM)
( ?bodyPartLabel = "Radius"^^xsd:string )
( ?bodyPartLabel = "Superficial fascia of forearm"^^xsd:string )
( ?bodyPartLabel = "Vasculature of forearm"^^xsd:string )
( ?bodyPartLabel = "Neural network of forearm"^^xsd:string )
( ?bodyPartLabel = "Ulna"^^xsd:string )
( ?bodyPartLabel = "Skin of forearm"^^xsd:string )

```

Figure 5.3: Answer to FMA1.

FMA3. What are the rehabilitation robots that do not affect any muscle of upper limb?

```

SELECT DISTINCT ?robotName
WHERE {
    ?robot rdf:type rr:RehabRobots.
    ?robot rr:has_Name ?robotName.
    FILTER NOT EXISTS{
        ?robot rr:targets ?movement.
        ?movement rdf:type rr:JointMovements.
        ?movement rdf:type fmaConcept.
        ?fmaConcept rdfs:subClassOf ?superClass.
        ?superClass rdfs:label 'Muscle of upper limb'.
    }
}

```

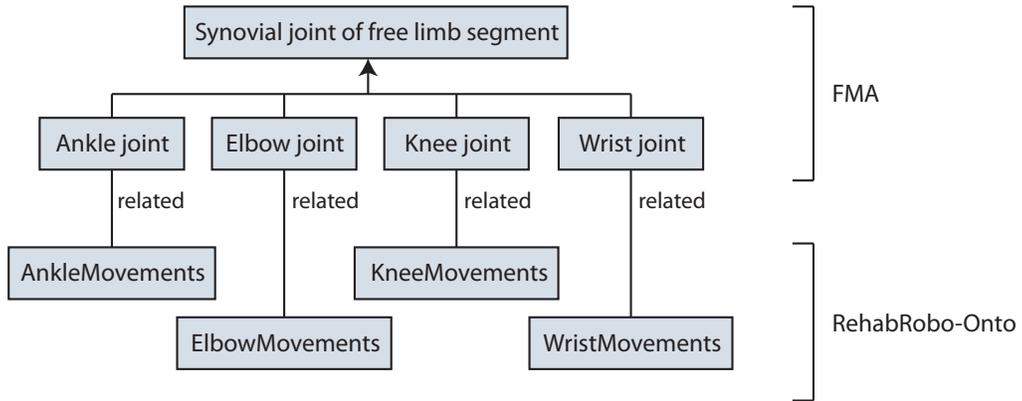


Figure 5.4: Hierarchy and integration of concepts for FMA2.

```

Rules:
Rules +
ElbowMovements(?x) -> 'Elbow joint'(?x)
AnkleMovements(?x) -> 'Ankle joint'(?x)
WristMovements(?x) -> 'Wrist joint'(?x)
KneeMovements(?x) -> 'Knee joint'(?x)

```

Figure 5.5: SWRL rules to answer FMA2.

}

“Muscle of upper limb” class in FMA has many subclasses that can be mapped to joint movements in REHABROBO-ONTO, such as “Muscle of forearm” or “Muscle of shoulder” (Figure 5.8), with the rules shown in Figure 5.9.

The answer to FMA3, with the code snippet that presents the query to PELLET for execution, is shown in Figure 5.10. The robots that do not target forearm, shoulder, or hand movements are included in the answer.

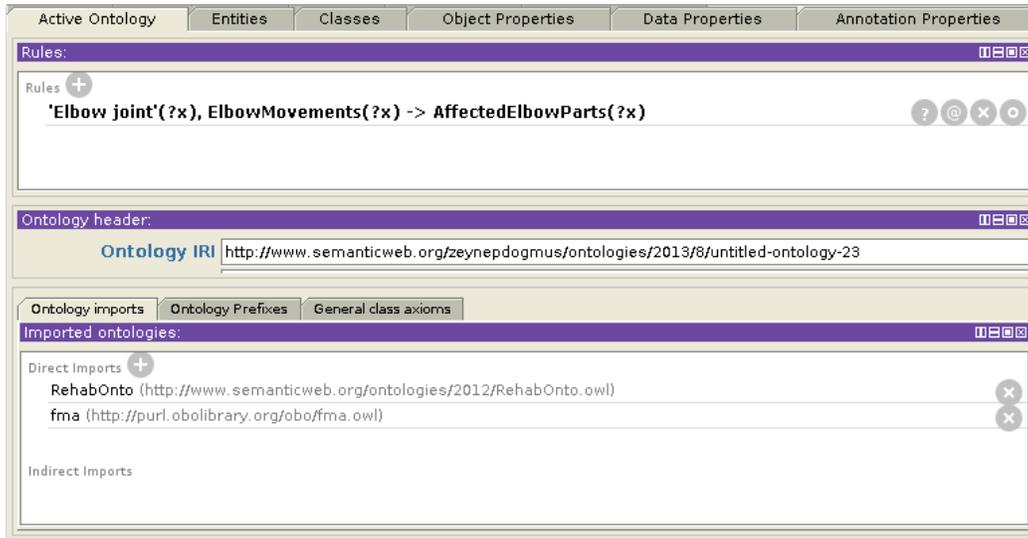


Figure 5.6: Defining a new concept with SWRL.

5.2 Integration with DO

The possible queries (DO1-DO3) that can be asked over DO and REHABROBONTO are as follows. Under each query, additional information about the concepts in DO, the rules in SWRL to integrate relevant concepts, SPARQL representations of the queries, and the answers to the queries are presented. Queries DO4-DO12 are presented in Appendix C.

DO1. What are the rehabilitation robots that can be used to treat ‘shoulder impingement syndrome’ and that target the shoulder scapular elevation/depression?

The following SPARQL query corresponds to DO1. With the implication rule stating that the instances of shoulder movements are related to shoulder impingement syndrome, we can obtain related concepts in DO using the variable of the related movement and `rdf:type` predicate.

```
SELECT DISTINCT ?robotName
```

```

String queryString = prefixSPARQL + "PREFIX swrl: <http://www.w3.org/2003/11/swrl#> " +
"PREFIX rr: <http://www.semanticweb.org/ontologies/2012/RehabOnto.owl#> " +
"PREFIX fma: <http://purl.obolibrary.org/obo/fma#> "+
"SELECT DISTINCT ?robotName "+
"WHERE {?robot rr:has_Name ?robotName. ?robot rdf:type rr:RehabRobots. "
+ "FILTER NOT EXISTS{ ?robot rr:targets ?movement. } ?movement rdf:type rr:JointMovements."
+ "?movement rdf:type ?fmaConcept. ?fmaConcept rdfs:subClassOf ?superClass."
+ "?superClass rdfs:label 'Synovial joint of free limb segment'. }";

Query wholeQuery = QueryFactory.create(queryString);
QueryExecution qexec = QueryExecutionFactory.create(wholeQuery, ontWithRuleLayer);
try {
    ResultSet results = qexec.execSelect();
    while(results.hasNext())
    {
        QuerySolution soln = results.nextSolution();
        System.out.println(soln.toString());
    }
} finally { qexec.close(); }

```

Problems @ Javadoc Declaration Console

```

<terminated> OntologyMethods [Java Application] /usr/lib/jvm/java-7-oracle/bin/java (Sep 4, 2013, 7:37:32 AM)
Classifying: 100% complete in 00:06
Classifying finished in 00:06
Realizing 79126 elements

Realizing: 0% complete in 00:00
Realizing: 100% complete in 00:00
Realizing finished in 00:00
( ?robotName = "AssistOn-Finger" )

```

Figure 5.7: Answer to FMA2.

```

WHERE {
    ?robot rdf:type rr:RehabRobots.
    ?robot rr:has_Name ?robotName.
    ?robot rr:targets ?movement.
    ?movement rdf:type rr:JointMovements.
    ?movement rdf:type ?doConcept.
    ?doConcept rdfs:label 'shoulder impingement syndrome'.
    ?robot rr:targets ?movement2.
    ?movement2 rdf:type rr:ShoulderScapularElevation/Depression.
}

```

“shoulder impingement syndrome” is a bone disease and can be mapped to (possibly particular) shoulder movements (Figure 5.11) with the rule shown in Figure 5.12.

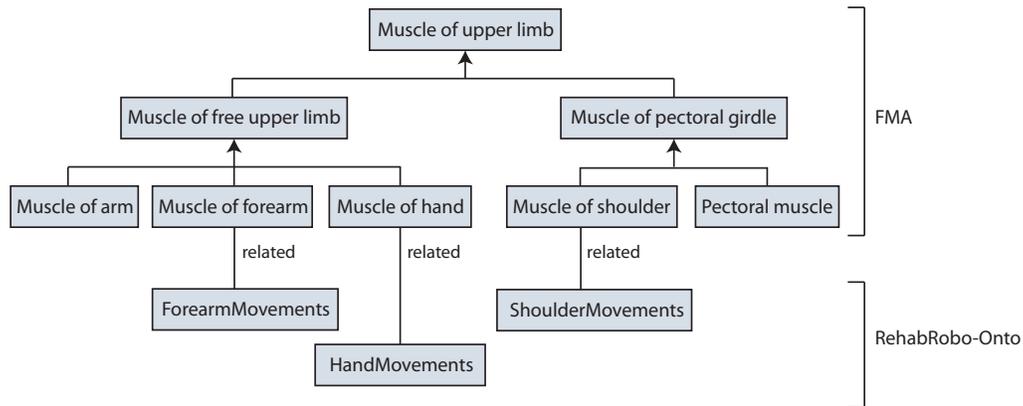


Figure 5.8: Hierarchy and integration of concepts for FMA3.

```

Rules:
Rules +
HandMovements(?x) -> 'Muscle of hand'(?x)
ForearmMovements(?x) -> 'Muscle of forearm'(?x)
ShoulderMovements(?x) -> 'Muscle of shoulder'(?x)

```

Figure 5.9: SWRL rules to answer FMA3.

The answer to DO1, with the code snippet that presents the query to PELLET for execution, is shown in Figure 5.13.

DO2. What are rehabilitation robots that can be used to treat some 'agnosia'?

```

SELECT DISTINCT ?robotName
WHERE {
  ?robot rr:has_Name ?robotName.
  ?robot rdf:type rr:RehabRobots.
  ?robot rr:targets ?movement.
  ?movement rdf:type rr:JointMovements.
  ?movement rdf:type ?doConcept.
  ?doConcept rdfs:subClassOf ?superClass.

```

```

String queryString = prefixSPARQL + "PREFIX swrl: <http://www.w3.org/2003/11/swrl#> " +
    "PREFIX rr: <http://www.semanticweb.org/ontologies/2012/RehabOnto.owl#>" +
    "PREFIX fma: <http://purl.obolibrary.org/obo/fma#> "+
    "SELECT DISTINCT ?robotName "+
    "WHERE { ?robot rr:has_Name ?robotName. ?robot rdf:type rr:RehabRobots. "
    + "FILTER NOT EXISTS { "
    + "?robot rr:targets ?movement. ?movement rdf:type rr:JointMovements. "
    + "?movement rdf:type ?fmaConcept. ?fmaConcept rdfs:subClassOf ?superClass."
    + "?superClass rdfs:label 'Muscle of upper limb'. } }";

Query wholeQuery = QueryFactory.create(queryString);
QueryExecution qexec = QueryExecutionFactory.create(wholeQuery, ontWithRuleLayer);
try {
    ResultSet results = qexec.execSelect();
    while(results.hasNext())
    {
        QuerySolution soln = results.nextSolution();
        System.out.println(soln.toString());
    }
}

```

Problems @ Javadoc Declaration Console

```

<terminated> OntologyMethods [Java Application] /usr/lib/jvm/java-7-oracle/bin/java (Sep 4, 2013, 1:10:50 PM)
Realizing: 0% complete in 00:00
Realizing: 100% complete in 00:00
Realizing finished in 00:00
( ?robotName = "AssistOn-Knee" )
( ?robotName = "Assist-On Ankle" )
( ?robotName = "deneme-kamadan" )
( ?robotName = "AssistOn-Finger" )

```

Figure 5.10: Answer to FMA3.

```

?superClass rdfs:label 'agnosia'.
}

```

In FMA, “finger agnosia” is a subclass of “agnosia”, which can be mapped to (possibly particular) finger movements in REHABROBO-ONTO(Figure 5.14). The rule that relates these concepts is shown in Figure 5.15.

The answer to DO2, with the code snippet that presents the query to PELLET for execution, is shown in Figure 5.16. The answer includes the robots that target some finger movements.

DO3. What are the rehabilitation robots that can be used to treat some ‘spinal canal and spinal cord meningioma’ and that are target some lumbar spline movements?

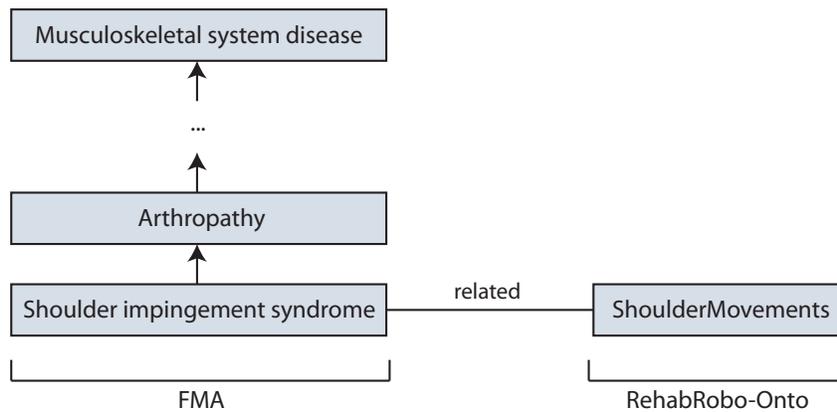


Figure 5.11: Hierarchy and integration of concepts for DO1.

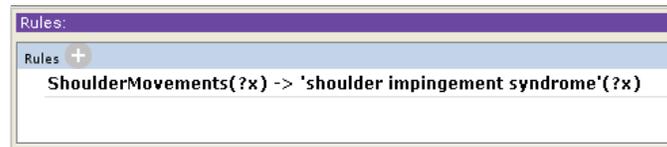


Figure 5.12: SWRL rule to answer DO1.

```

SELECT DISTINCT ?robotName
WHERE {
  ?robot rr:has_Name ?robotName.
  ?robot rdf:type rr:RehabRobots.
  ?robot rr:targets ?movement.
  ?movement rdf:type rr:JointMovements.
  ?movement rdf:type ?doConcept.
  ?doConcept rdfs:subClassOf ?superClass.
  ?superClass rdfs:label 'spinal canal and spinal cord meningioma'.
  ?robot rr:targets ?movement2.
  ?movement2 rdf:type rr:LumbarSplineMovements.
}

```

```

String queryString = prefixSPARQL + "PREFIX swrl: <http://www.w3.org/2003/11/swrl#> " +
"PREFIX rr: <http://www.semanticweb.org/ontologies/2012/RehabOnto.owl#> " +
"PREFIX do: <http://purl.obolibrary.org/obo/doid#> " +
"SELECT DISTINCT ?robotName " +
"WHERE { ?robot rr:has_Name ?robotName. ?robot rdf:type rr:RehabRobots. "
+ "?robot rr:targets ?movement. ?movement rdf:type rr:JointMovements."
+ "?movement rdf:type ?doConcept. ?doConcept rdfs:label 'shoulder impingement syndrome'."
+ "?robot rr:targets ?movement2. ?movement2 rdf:type rr:ShoulderScapularElevation\\Depression. }";

Query wholeQuery = QueryFactory.create(queryString);
QueryExecution qexec = QueryExecutionFactory.create(wholeQuery, ontWithRuleLayer);
try {
    ResultSet results = qexec.execSelect();
    while(results.hasNext())
    {
        QuerySolution soln = results.nextSolution();
        System.out.println(soln.toString());
    }
}

```

Figure 5.13: Answer to DO1.

Subclasses of “spinal canal and spinal cord meningioma” include “lumbar spinal canal and spinal cord meningioma”. It can be mapped to lumbar spline movements of REHABROBO-ONTO (Figure 5.17) with the rule shown in Figure 5.18.

The answer to DO3, with the code snippet that presents the query to PELLET for execution, is shown in Figure 5.19. Currently there is no such robot in REHABROBO-ONTO that can be used to treat spinal canal and spinal cord meningioma, and that targets some lumbar spline movements. Therefore, we do not get an answer.

5.3 On Extending REHABROBO-QUERY

By enabling interoperability of FMA and DO with REHABROBO-ONTO by means of a rule layer, it is possible to answer complex queries over multiple ontologies, in the same way complex queries are answered in [20]. We rep-

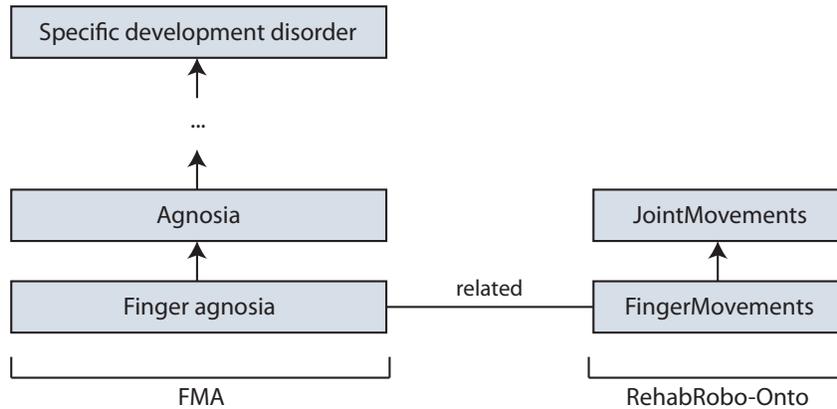


Figure 5.14: Hierarchy and integration of concepts for DO2.

```

Rules:
Rules +
FingerMovements(?x) -> 'finger agnosia'(?x)

```

Figure 5.15: SWRL rule to answer DO2.

resent the rules in the rule layer, in the rule language Semantic Web Rules Language (SWRL) [33]. Using SWRL, we can integrate relevant concepts in the ontologies and perform reasoning tasks (e.g., query answering). DL reasoners such as PELLET [67] and KAON2 [51] support reasoning over ontologies in OWL that include a rule layer.

Some of the queries are grammatically correct with respect to REHABROBO-CNL, such as

“What are the body parts that can be affected by some forearm robots?”

or

“What are the diseases that require some rehabilitation robots that target the elbow flexion/extension?”.

To enable such queries, extensions in the vocabulary are sufficient. We need

```

String queryString = prefixSPARQL + "PREFIX swrl: <http://www.w3.org/2003/11/swrl#> " +
"PREFIX rr: <http://www.semanticweb.org/ontologies/2012/RehabOnto.owl#> " +
"PREFIX do: <http://purl.obolibrary.org/obo/doi#> "+
"SELECT DISTINCT ?robotName "+
"WHERE { ?robot rr:has_Name ?robotName. ?robot rdf:type rr:RehabRobots. "
+ "?robot rr:targets ?movement. ?movement rdf:type rr:JointMovements."
+ "?movement rdf:type ?doConcept. ?doConcept rdfs:subClassOf ?superClass."
+ "?superClass rdfs:label 'agnosia'. }";

Query wholeQuery = QueryFactory.create(queryString);
QueryExecution qexec = QueryExecutionFactory.create(wholeQuery, ontWithRuleLayer);
try {
    ResultSet results = qexec.execSelect();
    while(results.hasNext())
    {
        QuerySolution soln = results.nextSolution();
        System.out.println(soln.toString());
    }
} finally { qexec.close(); }

```

Figure 5.16: Answer to DO2.

to provide “body part” and “disease” as types, “require” and “can affect” as verbs. Additionally, enabling passive forms of the verbs solves our problem.

However, some queries are not covered by REHABROBO-CNL, such as

“What are the rehabilitation robots that do not affect any muscle of upper limb?”.

Here, “muscle” and “upper limb” are different concepts. The keyword “of” specifies the muscles that are subclasses of upper limb. REHABROBO-CNL does not support such clauses.

In addition, the beginning of the query

“What types of spinal muscular atrophy can be treated using rehabilitation robot X? ”

violates the grammar of REHABROBO-CNL. We need to add one more query

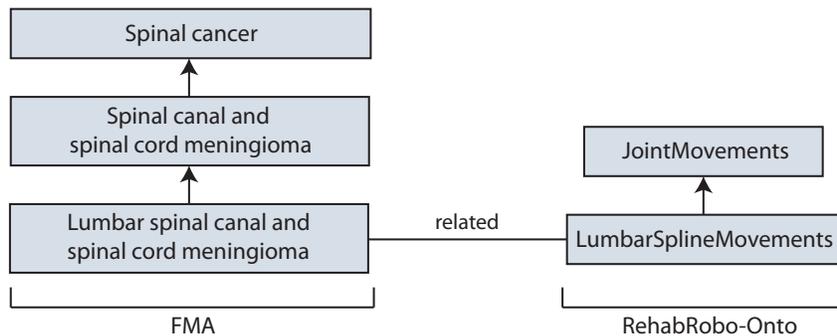


Figure 5.17: Hierarchy and integration of concepts for DO3.

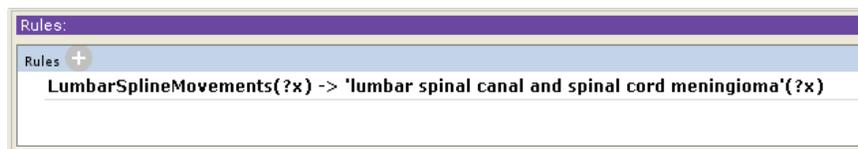


Figure 5.18: SWRL rule to answer DO3.

type for WHATQUERY to answer this query.

While extending the grammar and the vocabulary, we can also extend the user interface by providing templates for most common queries. Since an extension in the grammar would increase the number of options in the drop-down list, instead of forcing the users to construct a query from the very beginning, providing templates may assist the users for constructing new queries as well.

5.4 Integration of REHABROBO-ONTO with Patient Data

Enabling integration with patient data allows answering queries about therapies and related information, such as lesions. This paves the way for further integration with the patient data and the existing ontologies (e.g., disease ontology) presented in Sections 5.1 and 5.2.

```

String queryString = prefixSPARQL + "PREFIX swrl: <http://www.w3.org/2003/11/swrl#> " +
    "PREFIX rr: <http://www.semanticweb.org/ontologies/2012/RehabOnto.owl#> " +
    "PREFIX do: <http://purl.obolibrary.org/obo/doid#> "+
    "SELECT DISTINCT ?robotName "+
    "WHERE { ?robot rr:has_Name ?robotName. ?robot rdf:type rr:RehabRobots. "
    + "?robot rr:targets ?movement. ?movement rdf:type rr:JointMovements."
    + "?movement rdf:type ?doConcept. ?doConcept rdfs:subClassOf ?superClass."
    + "?superClass rdfs:label 'spinal canal and spinal cord meningioma'. "
    + "?robot rr:targets ?movement2. ?movement2 rdf:type rr:LumbarSplineMovements. }";

Query wholeQuery = QueryFactory.create(queryString);
QueryExecution qexec = QueryExecutionFactory.create(wholeQuery, ontWithRuleLayer);
try {
    ResultSet results = qexec.execSelect();
    while(results.hasNext())
    {
        QuerySolution soln = results.nextSolution();
        System.out.println(soln.toString());
    }
}

```

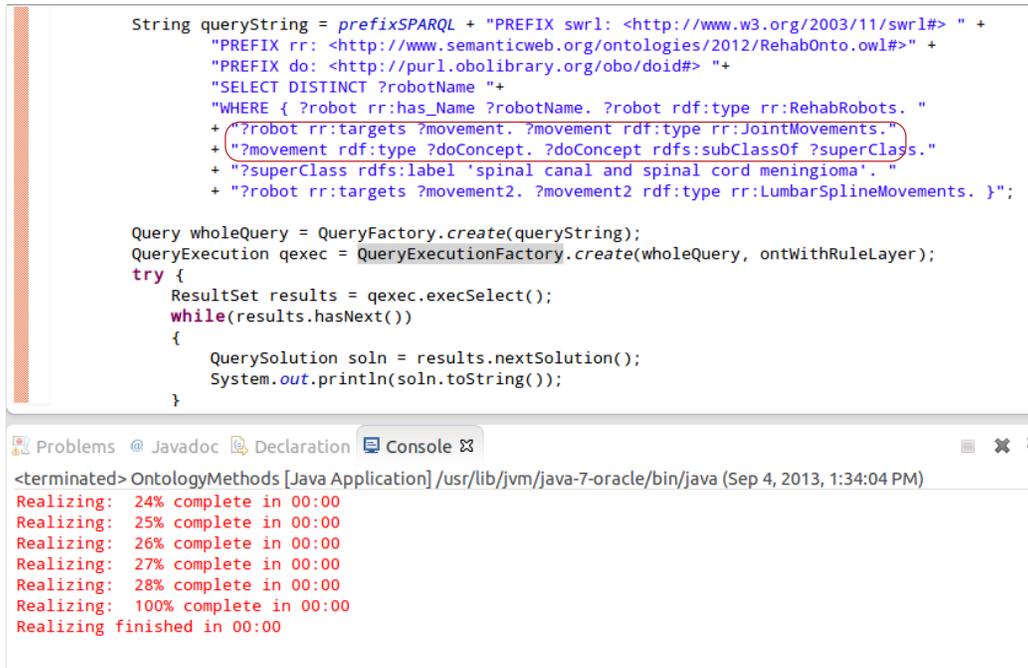


Figure 5.19: Answer to DO3.

We present a patient ontology (Figure 5.20), built by Sinan Yurtsever (2010), which provides information about the patients, their lesions, and how the therapy is done on a particular patient. It also provides a hierarchy for movements similar to REHABROBO-ONTO. The possible queries that can be asked over this patient ontology and REHABROBO-ONTO are as follows.

We can integrate the **Movement** concept in the patient ontology with the **JointMovements** concept in REHABROBO-ONTO. Movements in the patient ontology also have subconcepts. These subconcepts are integrated with the concepts in the **JointMovements** hierarchy in REHABROBO-ONTO. With the SWRL rules shown in Figure 5.21, it is possible to answer the following queries.

P1. What are the rehabilitation robots that have been used for some

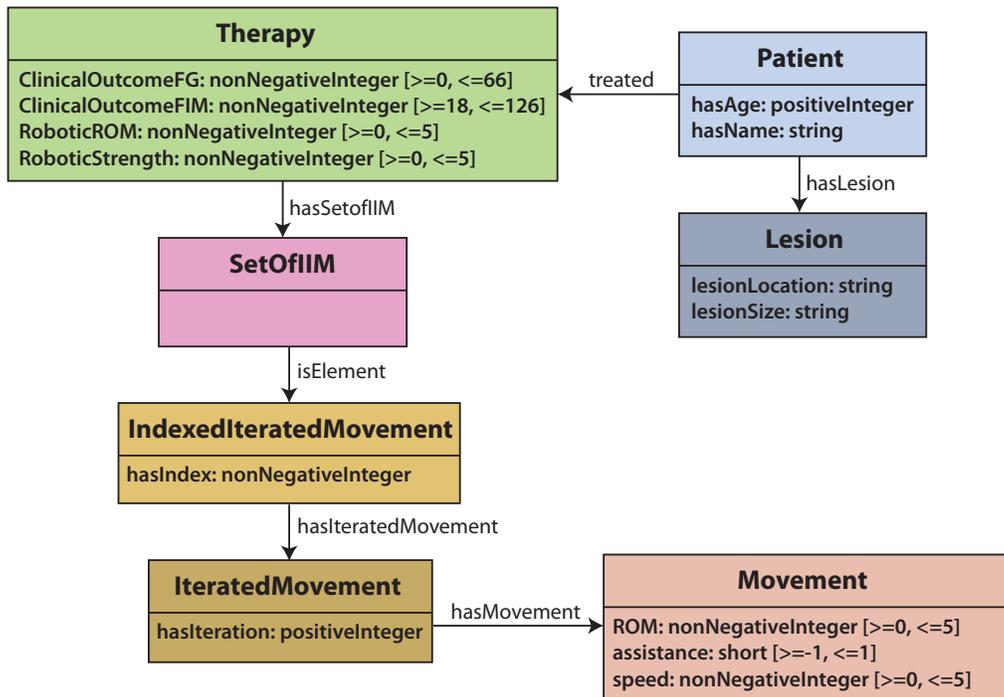


Figure 5.20: Patient ontology with main classes.

therapies with clinical outcome FIM ≥ 100 ?

```

SELECT DISTINCT ?robotName
WHERE {
  ?robot rdf:type rr:RehabRobots.
  ?robot rr:has_Name ?robotName.
  ?robot rr:targets ?movement.
  ?movement rdf:type rr:JointMovements.
  ?movement rdf:type ?movement2.
  ?movement2 rdf:type po:Movement.
  ?iteratedMov po:hasMovement ?movement2.
  ?indexedIteratedMov po:hasIteratedMovement ?iteratedMov.
}
  
```

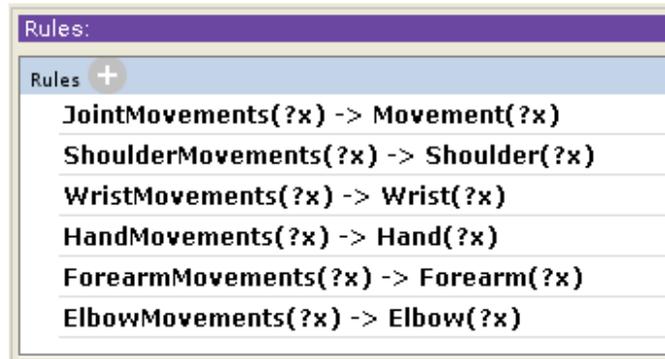


Figure 5.21: SWRL rules over patient ontology and REHABROBO-ONTO.

```

?setOfIIM po:isElement ?indexedIteratedMov.
?therapy po:hasSetofIIM ?setOfIIM.
?therapy po:ClinicalOutcomeFIM ?value. FILTER(?value >= 100)
}

```

P2. What are the rehabilitation robots that have been used for some therapies that treated some patients with age ≤ 15 ?

```

SELECT DISTINCT ?robotName
WHERE {
  ?robot rdf:type rr:RehabRobots.
  ?robot rr:has_Name ?robotName.
  ?robot rr:targets ?movement.
  ?movement rdf:type rr:JointMovements.
  ?movement rdf:type ?movement2.
  ?movement2 rdf:type po:Movement.
  ?iteratedMov po:hasMovement ?movement2.
  ?indexedIteratedMov po:hasIteratedMovement ?iteratedMov.
  ?setOfIIM po:isElement ?indexedIteratedMov.
}

```

```

?therapy po:hasSetofIIM ?setOfIIM.
?therapy po:treated ?patient.
?patient po:hasAge ?value. FILTER(?value <= 15)
}

```

P3. What are the lesion locations of the patients that have been treated with the rehabilitation robot 'AssistOn-SE'?

```

SELECT DISTINCT ?lesionLocation
WHERE {
  ?robot rdf:type rr:RehabRobots.
  ?robot rr:has_Name 'AssistOn-SE'.
  ?robot rr:targets ?movement.
  ?movement rdf:type rr:JointMovements.
  ?movement rdf:type ?movement2.
  ?movement2 rdf:type po:Movement.
  ?iteratedMov po:hasMovement ?movement2.
  ?indexedIteratedMov po:hasIteratedMovement ?iteratedMov.
  ?setOfIIM po:isElement ?indexedIteratedMov.
  ?therapy po:hasSetofIIM ?setOfIIM.
  ?therapy po:treated ?patient.
  ?patient po:hasLesion ?lesion.
  ?lesion po:lesionLocation ?lesionLocation.
}

```

P4. What are the therapies that have used some rehabilitation robots with motion capability='mobile'?

```

SELECT DISTINCT ?therapy
WHERE {
    ?robot rdf:type rr:RehabRobots.
    ?robot rr:has_Motion_Capability 'mobile'.
    ?robot rr:targets ?movement.
    ?movement rdf:type rr:JointMovements.
    ?movement rdf:type ?movement2.
    ?movement2 rdf:type po:Movement.
    ?iteratedMov po:hasMovement ?movement2.
    ?indexedIteratedMov po:hasIteratedMovement ?iteratedMov.
    ?setOfIIM po:isElement ?indexedIteratedMov.
    ?therapy po:hasSetofIIM ?setOfIIM.
}

```

P5. What are the publications that reference some rehabilitation robots that have been used for some therapies that treat some patients that have lesion size='small'?

```

SELECT DISTINCT ?publicationTitle
WHERE {
    ?publicationTitle rr:has_Title ?publication.
    ?publication rdf:type rr:References.
    ?robot rr:has_Reference ?publication.
    ?robot rdf:type rr:RehabRobots.
    ?robot rr:targets ?movement.
    ?movement rdf:type rr:JointMovements.
    ?movement rdf:type ?movement2.
}

```

```

?movement2 rdf:type po:Movement.
?iteratedMov po:hasMovement ?movement2.
?indexedIteratedMov po:hasIteratedMovement ?iteratedMov.
?setOfIIM po:isElement ?indexedIteratedMov.
?therapy po:hasSetofIIM ?setOfIIM.
?therapy po:treated ?patient.
?patient po:hasLesion ?lesion.
?lesion po:lesionSize 'small'.
}

```

P1 is grammatically correct with respect to REHABROBO-CNL. By providing “therapy” in types, “have been used for” in verbs and “clinical outcome” in nouns that relate to therapies, it is possible to answer P1. Now consider P3. In the patient ontology, lesions and patients are related to each other with “hasLesion” relation, and lesions have a lesion location property. Not only we query for the data properties of the lesion instances, but we also relate lesions with patients using “of” keyword. Both of them violate the grammar of REHABROBO-CNL. By providing a new query type (e.g., a new WHATQUERY) in REHABROBO-CNL, or by providing templates for queries as such, we can overcome these queries which are not covered grammatically or vocabulary-wise. Note that we use the verbs in present perfect tense since we are asking about the therapies that have been and still being used for real patients.

For CNL and user interface, the extensions that we propose here are very similar to the extensions that we proposed in Section 5.3. The approaches that we propose for integrating ontologies are also similar to them. By determining the alignments between REHABROBO-ONTO and the patient on-

tology, we can provide background knowledge with some approaches, such as rule-based integration. Moreover, by using the automated reasoners that support these approaches, we can answer queries about the relevant parts of these two ontologies.

The integration of REHABROBO-ONTO and patient ontology is slightly different from the integration with FMA and DO. It is actually easier due to the semantic equivalence between the `JointMovements` concept in REHABROBO-ONTO and `Movement` concept in patient ontology. Moreover, their hierarchies are almost equivalent. Therefore, it is possible to provide rules to integrate these concepts.

Note that the sample queries directly ask about therapies that use robots. However, in the patient ontology, movements and therapies are not directly related; there are some other concepts between them such as iterated movements or indexed iterated movements. Since therapies, patients and lesions in the patient ontology are more important than these concepts, it is possible to skip these concepts to provide simple natural language queries. We can add an exceptional condition for patient ontology in our algorithm to make the necessary connections in the SPARQL query. It is also possible to provide further integrations, such as mapping of the lesions in the patient ontology with the diseases in the disease ontology.

Chapter 6

6 Related Work

In this section, we discuss related work in two parts. First, we present the ontologies that are designed and developed for the use of robots. Then, we provide ontologies about robots. After discussing the most recent efforts to standardize the terminology about rehabilitation robots, we conclude the related publications for the first part. The second part involves ontology systems and tools that support natural language queries over ontologies. We present most recent research related to our work, and discuss the methods used in these systems, compared to our system.

6.1 Ontologies and Robots

6.1.1 Ontologies for Robots

There are some ontologies maintaining information about objects or environments [14, 37, 55, 68, 75, 77], developed for the use of robots.

Recent works having similar purposes include [39], which constructs an ontology about the environment of a robot in a semi-automatic way. As their system makes text analysis, they add the classified terms as ontology entities.

Another work in [9] propose a space ontology to model space with spatial relations and to define interest zones for a robot.

[42] introduce a system built on humanoid robot Nao, which utilizes a Japanese Wikipedia Ontology as a dictionary for semantic grammar to make it able to dialogue with the users on many topics. It also utilizes a Robot Action Ontology to perform related actions to dialogue topics. To integrate actions and dialogues, they align two ontologies. For that, they register related keywords manually using `rdfs:label` property of RDF(S), editing the keywords from PROTEGÉ.

Another recent work is [59], which proposes to use an ontology for the contexts of the objects recognized by a robot. With this ontology, which is made beforehand, the robots re-evaluate their regions of interest to send minimum information to each other.

In [70], an ontology (in OWL) is introduced that serves as a knowledge base about actions, objects and environments of the robots.

6.1.2 Ontologies about Robots

The related ontologies that are designed and developed for robots cover the environment and objects around robots, and actions of robots. However, there are only several works in the literature that have proposed ontologies specifically about robots.

In particular, Amigoni and Neri [3] introduce two ontologies (in OWL): one to store general concepts and properties/relations about the movement capabilities of mobile robots (e.g., wheels and their properties) and the other to describe the high level tasks that these robots can perform (e.g., move, rotate). The idea is then to allocate tasks and/or assign roles to mobile robots by means of querying these two ontologies using a description logics reasoner.

Schlenoff and Messina [62] introduce an ontology (in OWL) for urban search and rescue robots. The ontology captures structural characteristics (such as size), functional capabilities (such as locomotion capabilities) and operational considerations (such as display type) of the robots with a goal of assisting in the development and testing of search and rescue robot systems.

Juarez et al. [38] introduce a database (called ROBODB) for storing physical characteristics of robots; but also note that they plan to transform the knowledge stored in ROBODB into an OWL ontology to benefit from this “common” language of ontologies and related reasoners.

Nilsson et al. [52] propose to use an ontology (in OWL-S) for modeling and design of robots. They introduce some possible properties (e.g., cost, height, length, material, environment conditions) and classes (e.g., sensor, connector, workpiece) and demonstrate them using PROTÉGÉ.

6.1.3 Standardization of Rehabilitation Robots

There are some efforts to standardize the methodology for knowledge representation in robotics and automation. For instance, a newly formed IEEE RAS Ontologies and Automation (ORA) working group has recently published a paper [61] including ideas for the ontology development process, introducing their subgroups (upper ontology/methodology, autonomous robots, service robots and industrial robots) and describing domains where the developed ontologies could be utilized.

Moreover, in [56], ORA working group specifies some elements which will be represented in the ontologies that they plan to develop, such as path planning, control, robotic platform, or sensor. As an ontology language, they plan to use OWL and its variations.

None of the existing and proposed robot ontologies have been designed to target rehabilitation robots and, without further customization, they fail to capture many important aspects of rehabilitation robots, including the interoperability with the existing ontologies in physical medicine.

6.2 Ontology Systems that Support Natural Language Queries

We now give an overview of the tools that support natural language queries over ontologies. Development of natural language interfaces that provide query answering over ontologies has been subject of research for many years. For this reason, many systems [7, 11, 15, 26, 40, 41, 43, 47, 69, 74, 78] have been developed that propose various approaches over some common challenges, such as processing of the natural language input (balancing ambiguity and expressiveness) and support for broad or narrow domains (portability).

The most recently developed systems include BIOQUERY-ASP [21], which is a software system that answers natural language queries over biomedical databases and ontologies. It utilizes Answer Set Programming (ASP) [45] to query such knowledge resources. It allows the users to enter queries in a controlled natural language from its user interface, and then answers the queries by transforming the query in a controlled natural language into an ASP program. To enable interoperability over multiple biomedical ontologies and databases, it integrates ontologies via a rule layer in ASP. To answer queries, it utilizes ASP solvers such as CLASP [27] and CLASP-NK, and it also provides explanations to the queries.

Ferrández et al. [25] introduce QACID, which covers a movie ontology. The idea is to train the system using many queries and keep the resulting

set of clusters (mostly asked questions) in a database. Then, manually, each query type is associated with a SPARQL query. Finally, the queries are answered by a query engine that is implemented in QACID and proposed as a new entailment-based engine.

FREYA [18] is developed by the creators of and as a development upon QUESTIO [69]. In order to support natural language queries, it uses Stanford Parser [19] to generate a parse tree. Then, using GATE libraries [16], it tries to find some ontology concepts that can be mapped to the query terms. Then, it generates a SPARQL query and executes the query using the inference engine in BIGOWLIM, that supports SPARQL, on the top of Sesame. It relies on clarification dialogues with users in the cases of ambiguity or in the cases where the system cannot find an answer to a query. Over time, the system learns to ask the correct questions to the users by placing correct suggestions on top of similar queries. The system is tested on one dataset, and it is stated that FREYA failed to answer some questions (e.g., queries including negation) correctly. These questions could not be mapped to a SPARQL query in spite of clarification dialogues and learning mechanism.

Lopez et al. [46] introduce PowerAqua, which is evolved from Aqua-Log [47]. It provides natural language querying over multiple ontologies; thus, supports high scalability and portability. It uses GATE libraries and WordNet [23] to process natural language queries. It transforms the queries to triples and answers them with its own query engine. To limit the search space, it uses filtering and ranking heuristics. Since it does not contain any linguistic knowledge in the background, it has a limited linguistic coverage. It is good at answering simple questions yet it fails on questions that contain comparisons and quantifiers.

Valencia-García et al. [73] introduce OWLPath, which gets user queries in a controlled natural language, transforms it into a SPARQL query and executes the query over an ontology via Jena framework and using the DL reasoner PELLETT. The statements in its CNL start with “View any...” and follow English grammar. However, they are not full and valid English sentences. Although it is stated that OWLPath provides a Web interface through AJAX, it is not available online. For each condition in the query, OWLPath adds a FILTER statement in the SPARQL query. Therefore, the transformation of the query into SPARQL is not, in fact, a transformation to triples but a set of FILTER statements. Evaluations are done on ontologies in OWL DL.

Chapter 7

7 Conclusion

In this thesis, we presented the first formal rehabilitation robotics ontology, called REHABROBO-ONTO, to represent information about rehabilitation robots; and a software system REHABROBO-QUERY to facilitate access to this ontology. We have made REHABROBO-QUERY available on the cloud, utilizing Amazon EC2 Web services, so that rehabilitation robot designers around the world can add/modify information about their robots in REHABROBO-ONTO, and rehabilitation robot designers and physical medicine experts around the world can access the knowledge in REHABROBO-ONTO by means of questions about robots, in natural language, with the guide of the intelligent user-interface of REHABROBO-QUERY. The users do not have to know about the underlying logical formalism of the ontology or the formalism to represent queries; they do not have to know about the use of the technologies for computing answers to their questions.

By means of such queries over REHABROBO-ONTO, right rehabilitation robots for a particular patient or a physical therapy can be found or designed; this further paves the way for translational physical medicine (from bench-to-bed and back) and personalized physical medicine. REHABROBO-QUERY aids exchange of information across rehabilitation robots researchers over the world, and thus to improve the state-of-the-art; it allows to identify “gaps”

in functionality of rehabilitation robots, that can further improve research efforts. Furthermore, having a structured formal representation of knowledge about rehabilitation robots, allows answering complex queries that requires integration with other knowledge resources (e.g., patient databases, disease ontologies).

The importance of designing and developing ontologies for robotics is emphasized by IEEE-RAS Ontologies for Robotics and Automation Working Group¹². The group has initiated the design and development of ontologies for several sorts of robots (e.g., mobile robots, urban search and rescue robots). However, none of the existing robot ontologies have been designed to target rehabilitation robots and, without further customization, they fail to capture many important aspects of rehabilitation robots, including the interoperability with the existing ontologies in physical medicine. Furthermore, none of them is open-source where the researchers are allowed to contribute and access. In that sense, our work contributes to efforts towards designing and developing robotics ontologies.

For many years, there has been an ongoing effort to develop an expressive and portable natural language interface to query over ontologies. Therefore, many tools have been developed that cover some aspects of this challenge. However, the existing tools fail to answer complex questions, such as the questions that include quantifiers or negation. Moreover, most of them are not available online to evaluate. In this regard, the grammar of REHABROBO-CNL supports queries with negation and quantifiers, as well as conjunctions and disjunctions. Our natural language query answering system is available online for evaluation.

¹²<http://www.ieee-ras.org/industrial/standards.html>

We presented possible extensions for interoperability of REHABROBO-ONTO. As future work, integration of REHABROBO-ONTO with existing anatomy, disease and patient ontologies can be achieved by providing a rule layer between these ontologies and REHABROBO-ONTO, for integration of the related concepts. In addition, some extensions in the grammar, the algorithms and the user interface are needed to be able to answer complex queries about therapies, diseases and anatomy. Providing templates for most common queries may be interesting as well.

Appendix

A Transformations of Sample Queries

Every SPARQL query in REHABROBO-QUERY starts with the prefixes:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
```

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
```

```
PREFIX owl: <http://www.w3.org/2002/07/owl#>
```

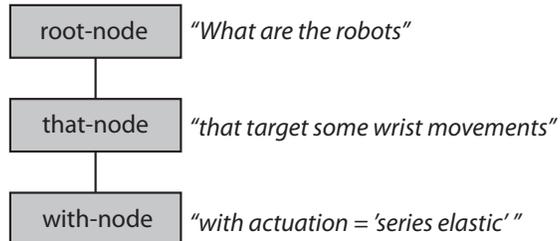
```
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
```

```
PREFIX rr: <http://www.semanticweb.org/ontologies/2012/RehabOnto.owl#>
```

Prefix part is removed from the examples in this chapter for simplicity.

Q1. What are the robots that target some wrist movements with actuation='series elastic'?

Q1 in Query Description Tree:



Q1 in Description Logics (DL):

$\text{Robot} \sqcap \exists \text{target} . (\text{WristMovement} \sqcap \exists \text{actuation} . \{\text{series elastic}\})$

Q1 in SPARQL:

```
SELECT DISTINCT ?name
WHERE {
  ?robot1 rdf:type rr:RehabRobots.
  ?robot1 rr:has_Name ?name.
  ?robot1 rr:targets ?movement1.
  ?movement1 rdf:type rr:WristMovements.
  ?movement1 rr:has_Actuation 'series elastic'.
}
```

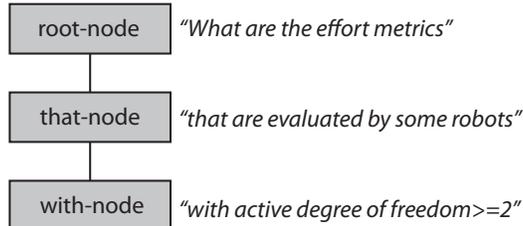
Answer to Q1:

- AssistOn-Mobile

Figure A.1: Transformation for Q1

Q2. What are the effort metrics that are evaluated by some robots with active degree of freedom ≥ 2 ?

Q2 in Query Description Tree:



Q2 in Description Logics (DL):

$\text{EffortMetric} \sqcap \exists \text{evaluate}^-. (\text{Robot} \sqcap \exists \text{activeDOF} \geq 2)$

Q2 in SPARQL:

```
SELECT DISTINCT ?leafClass
WHERE {
  ?effortmetric1 rdf:type ?leafClass.
  ?leafClass rdfs:subClassOf rr:EffortAssessment.
  FILTER NOT EXISTS {?c1 rdfs:subClassOf ?leafClass.
    FILTER(?c1 != owl:Nothing)}
  ?robot1 rr:hasAssessment ?effortmetric2.
  ?effortmetric2 rdf:type ?leafClass.
  ?robot1 rdf:type rr:RehabRobots.
  ?robot1 rr:has_Active_DOF ?val1. FILTER(?val1 >= (2))
}
```

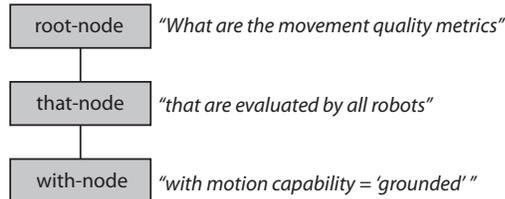
Answer to Q2:

- Time To Initiate Movement
- Amount Of Compensation
- Biomechanical Work Energy Power
- Movement Independent Mechanical Effort
- Pain Induced By Movement
- Amount Of Assistance

Figure A.2: Transformation for Q2

Q3. What are the movement quality metrics that are evaluated by all robots with motion capability = 'grounded' ?

Q3 in Query Description Tree:



Q3 in Description Logics (DL):

$\text{MovementQualityMetric} \sqcap$
 $\forall \text{evaluate}^-. (\text{Robot} \sqcap \exists \text{motionCapability} . \{\text{grounded}\})$

Q3 in SPARQL:

```

SELECT DISTINCT ?leafClass
WHERE {
  ?movementqualitymetric1 rdf:type ?leafClass.
  ?leafClass rdfs:subClassOf rr:MovementQualityAssessment.
  FILTER NOT EXISTS {?c1 rdfs:subClassOf ?leafClass.
    FILTER(?c1 != owl:Nothing)}
  FILTER NOT EXISTS {
    FILTER NOT EXISTS {
      ?robot1 rr:hasAssessment ?movementqualitymetric2.
      ?movementqualitymetric2 rdf:type ?leafClass.
    }
    ?robot1 rdf:type rr:RehabRobots.
    ?robot1 rr:has_Motion_Capability 'grounded'.
  }
}
  
```

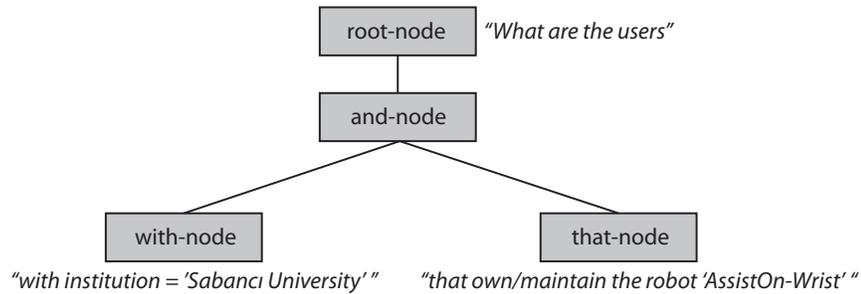
Answer to Q3:

- Visuomotor Coordination
- Combined Task Coordination
- Single Joint Coordination
- Bimanual Coordination
- Compensation
- Interlimb Coordination

Figure A.3: Transformation for Q3

Q4. What are the users with institution='Sabanci University' and that own/maintain the robot 'AssistOn-Wrist'?

Q4 in Query Description Tree:



Q4 in Description Logics (DL):

$$\text{User} \sqcap (\exists \text{institution}.\{\text{Sabanci University}\} \sqcap \exists \text{own}.\{\text{Robot} \sqcap \exists \text{name}.\{\text{AssistOn - Wrist}\}\})$$

Q4 in SPARQL:

```

SELECT DISTINCT ?name
WHERE {
  ?user1 rdf:type rr:Owners.
  ?user1 rr:has_User_Name ?name.
  ?user1 rr:has_Institution 'Sabanci University'.
  ?robot1 rr:ownedBy ?user1.
  ?robot1 rdf:type rr:RehabRobots.
  ?robot1 rr:has_Name 'AssistOn-Wrist'.
}
  
```

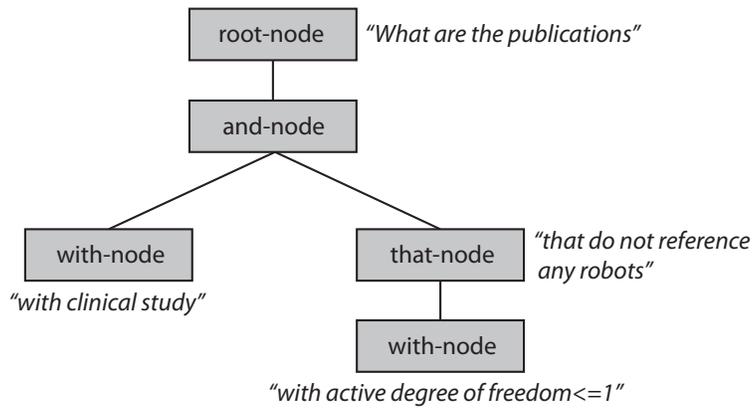
Answer to Q4:

- RehabRobo Onto

Figure A.4: Transformation for Q4

Q5. What are the publications with clinical study and that do not reference any robots with active degree of freedom ≤ 1 ?

Q5 in Query Description Tree:



Q5 in Description Logics (DL):

$\text{Publication} \sqcap \exists \text{clinicalStudy}.\{\text{'true'}^{\wedge\wedge \text{xsd: boolean}}\} \sqcap$
 $\neg \exists \text{reference}.\text{(Robot} \sqcap \exists \text{activeDOF} \leq_1)$

Q5 in SPARQL:

```

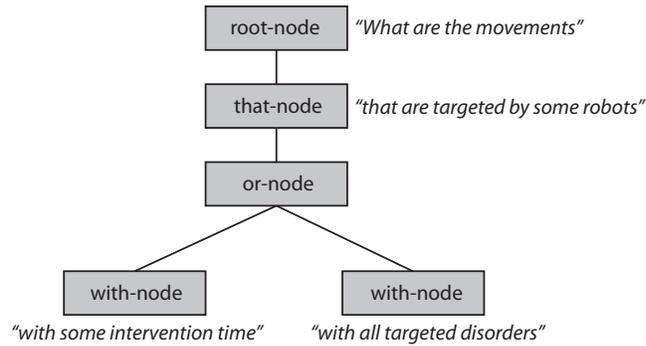
SELECT DISTINCT ?name
WHERE {
  ?publication1 rdf:type rr:References.
  ?publication1 rr:has_Title ?name.
  ?publication1 rr:has_Clinical_Study 'true'^^^xsd:boolean.
  FILTER NOT EXISTS {
    ?robot1 rr:hasReference ?publication1.
    ?robot1 rdf:type rr:RehabRobots.
    ?robot1 rr:has_Active_DOF ?val1. FILTER(?val1<=(1))
  }
}
  
```

Answer to Q5: No answer

Figure A.5: Transformation for Q5

Q6. What are the movements that are targeted by some robots with (some intervention time or with all targeted disorders)?

Q6 in Query Description Tree:



Q6 in Description Logics (DL):

Movement $\sqcap \exists \text{target}^{\neg} . (\text{Robot} \sqcap (\exists \text{interventionTime} . (\text{xsd} : \text{string}) \sqcup \forall \text{targetedDisorder} . (\text{xsd} : \text{string})))$

Q6 in SPARQL:

```

SELECT DISTINCT ?leafClass
WHERE {
  ?movement1 rdf:type ?leafClass.
  ?leafClass rdfs:subClassOf rr:JointMovements.
  FILTER NOT EXISTS {?c1 rdfs:subClassOf ?leafClass.
    FILTER(?c1 != owl:Nothing)}
  FILTER NOT EXISTS{?movement1 rr:has_Actuation 'hydrolic'.}
  FILTER NOT EXISTS{?movement1 rr:has_Actuation
'pneumatic'.}
  FILTER NOT EXISTS{?movement1 rr:has_Actuation 'other'.}
  ?robot1 rr:targets ?movement1.
  ?robot1 rdf:type rr:RehabRobots.
  {?robot1 has_Intervention_Time ?val1.} UNION
  {?robot1 has_Targeted_Disorder 'stroke'.
  ?robot1 has_Targeted_Disorder 'spineCordInjury'.}
}
  
```

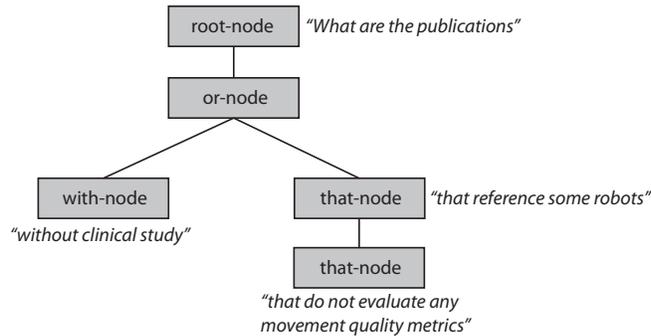
Answer to Q6:

- Index Finger DIP Flexion/Extension
- Shoulder Scapular Elevation/Depression
- Wrist Flexion/Extension
- ...

Figure A.6: Transformation for Q6

Q7. What are the publications without clinical study or that reference some robots that do not evaluate any movement quality metrics?

Q7 in Query Description Tree:



Q7 in Description Logics (DL):

$$\text{Publication} \sqcap (\exists \text{clinicalStudy}.\{\text{'false'}^{\wedge\wedge} \text{xsd: boolean}\} \sqcup \exists \text{reference}.\text{Robot} \sqcap \neg \exists \text{evaluate}.\text{MovementQualityMetric})$$

Q7 in SPARQL:

```

SELECT DISTINCT ?name
WHERE {
  ?publication1 rdf:type rr:References.
  ?publication1 rr:has_Title ?name.
  {FILTER NOT EXISTS {
    ?publication1 rr:has_Clinical_Study 'true'^xsd:boolean.
  }} UNION
  {?robot1 rr:hasReference ?publication1.
  ?robot1 rdf:type rr:RehabRobots.
  FILTER NOT EXISTS {
    ?robot1 rr:hasAssessment ?metric1.
    ?metric1 rdf:type rr:MovementQualityAssessment.
  }}
}
  
```

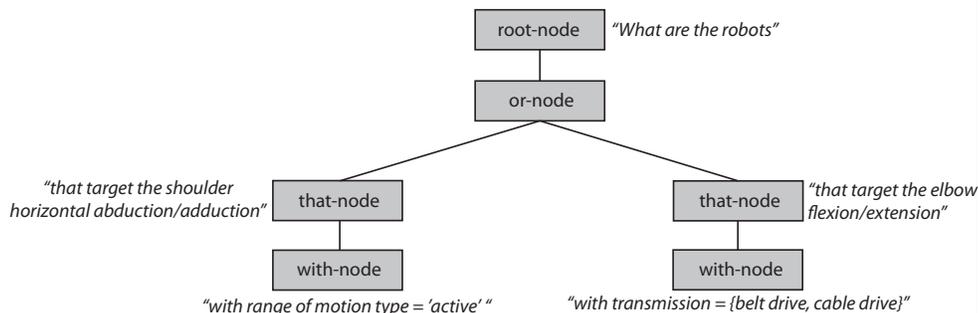
Answer to Q7:

- Kinematics and design of AssistOn-SE: A self-adjusting shoulder-elbow exoskeleton
- A Self-Adjusting Knee Exoskeleton for Robot-Assisted Treatment of Knee Injuries
- Brain Computer Interface based Robotic Rehabilitation with On-line Modification of Task Speed
- ...

Figure A.7: Transformation for Q7

Q8. What are the robots that target the shoulder horizontal abduction/adduction with range of motion type='active' or that target the elbow flexion/extension with transmission={belt drive, cable drive}?

Q8 in Query Description Tree:



Q8 in Description Logics (DL):

```

Robot  $\sqcap$  ( $\exists$ target.(ShoulderHorizontalAbduction/Adduction  $\sqcap$ 
 $\exists$ ROMType.{active}) $\sqcup$ 
 $\exists$ target.(ElbowFlexion/Extension  $\sqcap$ 
 $\exists$ transmission.{belt drive}  $\sqcap$ 
 $\exists$ transmission.{cable drive}))
  
```

Q8 in SPARQL:

```

SELECT DISTINCT ?name
WHERE {
  ?robot1 rdf:type rr:RehabRobots.
  ?robot1 rr:has_Name ?name.
  {?robot1 rr:targets ?movement1.
  ?movement1 rdf:type rr:ShoulderHorizontalAbduction/
  Adduction.
  ?movement1 rr:has_ROM_Type 'active'. } UNION
  ?movement2 rdf:type rr:ElbowFlexion/Extension.
  ?movement2 rr:has_Transmission 'belt drive'.
  ?movement2 rr:has_Transmission 'cable drive'.}
}
  
```

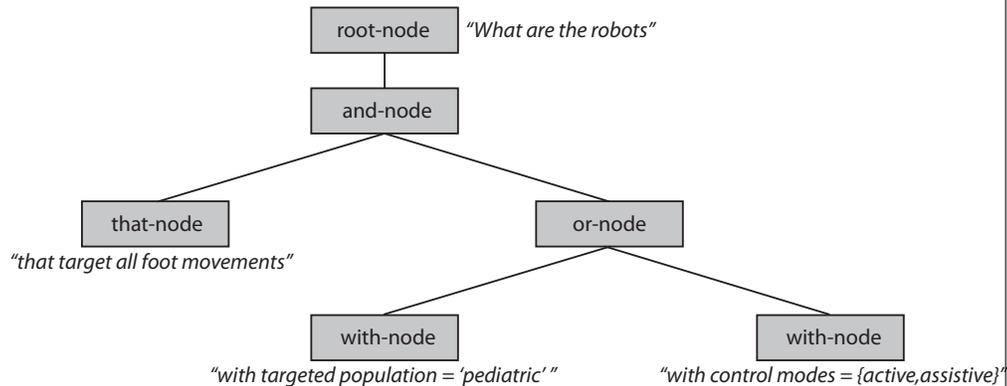
Answer to Q8:

- AssistOn-Mobile
- AssistOn-Arm

Figure A.8: Transformation for Q8

Q9. What are the robots that target all foot movements and (with targeted population='pediatric' or with control modes={active, assistive})?

Q9 in Query Description Tree:



Q9 in Description Logics (DL):

Robot $\sqcap \forall \text{target.FootMovement} \sqcap$
 $(\exists \text{targetedPopulation}.\{\text{pediatric}\} \sqcup (\exists \text{controlModes}.\{\text{active}\} \sqcap$
 $\exists \text{controlModes}.\{\text{assistive}\}))$

Q9 in SPARQL:

```

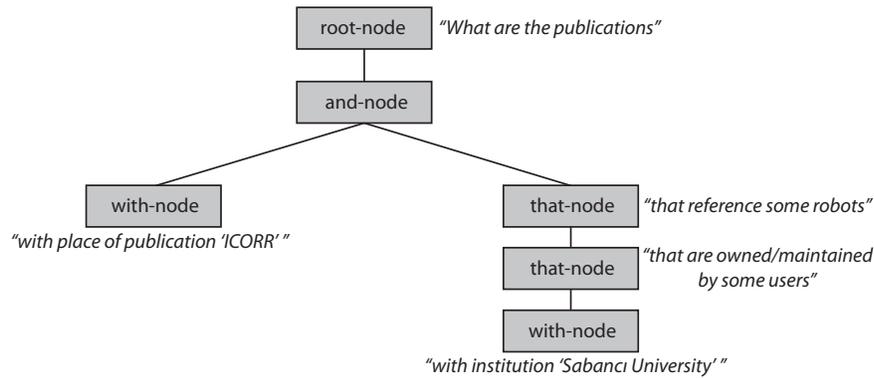
SELECT DISTINCT ?name
WHERE {
  ?robot1 rdf:type rr:RehabRobots.
  ?robot1 rr:has_Name ?name.
  {?robot1 rr:targets ?movement1.
  ?movement1 rdf:type rr:FootInterphalangealJointsOfToe
  Flexion/Extension.
  ?robot1 rr:targets ?movement2.
  ?movement2 rdf:type rr:FootMetatarsophalangealJoints
  Flexion/Extension.
  {?robot1 rr:has_Targeted_Population 'pediatric'.} UNION
  {?robot1 rr:has_Control_Modes 'active'.
  ?robot1 rr:has_Control_Modes 'assistive'.}
}
  
```

Answer to Q9: No answer

Figure A.9: Transformation for Q9

Q10. What are the publications with place of publication 'ICORR' and that reference some robots that are owned/maintained by some users with institution 'Sabanci University' ?

Q10 in Query Description Tree:



Q10 in Description Logics (DL):

```

Publication ⊓ ∃placeOfPublication.{ICORR} ⊓
∃reference.(Robot ⊓
  ∃own⁻.(User ⊓ ∃institution.{Sabanci University}))
  
```

Q10 in SPARQL:

```

SELECT DISTINCT ?name
WHERE {
  ?publication1 rdf:type rr:References.
  ?publication1 rr:has_Title ?name.
  ?publication1 rr:has_PublishedAt 'ICORR'.
  ?robot1 rr:hasReference ?publication1.
  ?robot1 rdf:type rr:RehabRobots.
  ?robot1 rr:ownedBy ?user1.
  ?user1 rdf:type rr:Owners.
  ?user1 rr:has_Institution 'Sabanci University'.
}
  
```

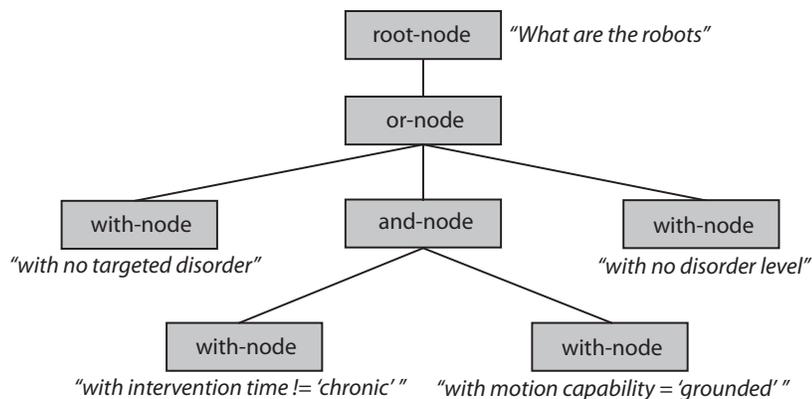
Answer to Q10:

- Brain Computer Interface based Robotic Rehabilitation with On-line Modification of Task Speed
- Design of a reconfigurable ankle rehabilitation robot and its use for the estimation of the ankle impedance
- Passive Velocity Field Control of a Forearm-Wrist Rehabilitation Robot

Figure A.10: Transformation for Q10

Q11. What are the robots with no targeted disorder or (with intervention time!=‘chronic’ and with motion capability=‘grounded’) or with no disorder level?

Q11 in Query Description Tree:



Q11 in Description Logics (DL):

Robot $\sqcap (\neg \exists \text{targetedDisorder}.\langle \text{xsd} : \text{string} \rangle) \sqcup$
 $(\neg \exists \text{interventionTime}.\{\text{chronic}\} \sqcap \exists \text{motionCapability}.\{\text{grounded}\}) \sqcup$
 $\neg \exists \text{disorderLevel}.\langle \text{xsd} : \text{string} \rangle)$

Q11 in SPARQL:

```

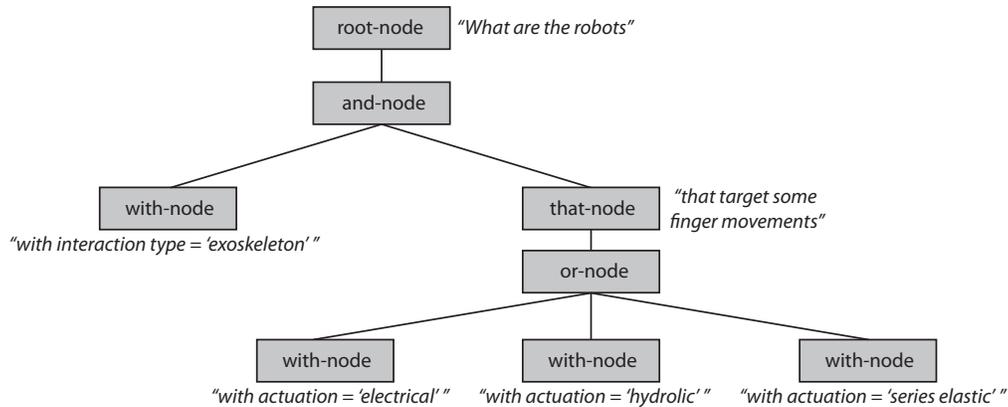
SELECT DISTINCT ?name
WHERE {
  ?robot1 rdf:type rr:RehabRobots.
  ?robot1 rr:has_Name ?name.
  {FILTER NOT EXISTS {
    ?robot1 rr:has_Targeted_Disorder ?val1.
  }} UNION
  {FILTER NOT EXISTS {
    ?robot1 rr:has_Intervention_Time 'chronic'.
  }}
  ?robot1 rr:has_Motion_Capability 'grounded'.
} UNION
{FILTER NOT EXISTS {
  ?robot1 rr:has_Disorder_Level ?val2.
}}
}
  
```

Answer to Q11: No answer

Figure A.11: Transformation for Q11

Q12. What are the robots with interaction type = 'exoskeleton' and that target some finger movements (with actuation = 'electrical' or with actuation = 'hydrolic' or with actuation = 'series elastic') ?

Q12 in Query Description Tree:



Q12 in Description Logics (DL):

$$\text{Robot} \sqcap \exists \text{interactionType}.\{\text{exoskeleton}\} \sqcap \exists \text{target}.\left(\text{FingerMovement} \sqcap \left(\exists \text{actuation}.\{\text{electrical}\} \sqcup \exists \text{actuation}.\{\text{hydrolic}\} \sqcup \exists \text{actuation}.\{\text{series elastic}\}\right)\right)$$

Q12 in SPARQL:

```

SELECT DISTINCT ?name
WHERE {
  ?robot1 rdf:type rr:RehabRobots.
  ?robot1 rr:has_Name ?name.
  ?robot1 rr:has_Interaction_Type 'exoskeleton'.
  ?robot1 rr:targets ?movement1.
  ?movement1 rdf:type rr:FingerMovements.
  { ?movement1 rr:has_Actuation 'electrical'. } UNION
  { ?movement1 rr:has_Actuation 'hydrolic'. } UNION
  { ?movement1 rr:has_Actuation 'series elastic'. }
}
  
```

Answer to Q12: No answer

Figure A.12: Transformation for Q12

B Example Queries over FMA and REHABROBO- ONTO

FMA4. What are the publications that reference some rehabilitation robots that can affect palmaris brevis?

```
SELECT DISTINCT ?publicationTitle
WHERE {
    ?publicationTitle rr:has_Title ?publication.
    ?publication rdf:type rr:References.
    ?robot rr:has_Reference ?publication.
    ?robot rdf:type rr:RehabRobots.
    ?robot rr:targets ?movement.
    ?movement rdf:type rr:JointMovements.
    ?movement rdf:type ?fmaConcept.
    ?fmaConcept rdfs:label 'palmaris brevis'.
}
```

In FMA, palmaris brevis is a muscle of hand. The answer to this query includes the publications that reference the robots that target some hand movements. We can also specify the hand movements that may affect this muscle. However, since there is one movement of hand in REHABROBO-ONTO, we match hand movements to some muscles of hand directly.

FMA5. What are the muscles that can be affected by some shoulder robots?

```
SELECT DISTINCT ?bodyPartLabel
WHERE {
```

```

?robot rdf:type rr:ShoulderRobots.
?robot rr:targets ?movement.
?movement rdf:type rr:JointMovements.
?movement rdf:type ?fmaConcept.
?movement rdfs:label ?bodyPartLabel.
?movement rdfs:subClassOf ?superClass.
?superClass rdfs:label 'Muscle of shoulder'.
}

```

FMA includes the sets of all extrinsic and intrinsic muscles of the shoulder. Particular shoulder movements can be mapped to particular muscles.

FMA6. What are the users that own/maintain some rehabilitation robots that can affect interphalangeal joint?

```

SELECT DISTINCT ?userName
WHERE {
  ?userName rr:has_User_Name ?user.
  ?user rdf:type Owners.
  ?robot rr:ownedBy ?user.
  ?robot rr:targets ?movement.
  ?movement rdf:type rr:JointMovements.
  ?movement rdf:type ?fmaConcept.
  ?fmaConcept rdfs:subClassOf ?superConcept.
  ?superConcept rdfs:label 'Interphalangeal Joint'.
}

```

“Interphalangeal joint” has subclasses such as “Interphalangeal joint of finger” and “Interphalangeal joint of toe” in FMA. Further subclasses of

“Interphalangeal joint of finger” include distal/proximal joints of left/right index/little/middle/ring fingers and thumb. They can be mapped to the particular finger movements in REHABROBO-ONTO.

FMA7. What are the rehabilitation robots that can affect some muscles of lower limb that attach to 'Greater trochanter'?

```
SELECT DISTINCT ?robotName
WHERE {
  ?robot rdf:type rr:RehabRobots.
  ?robot rr:has_Name ?robotName.
  ?robot rr:targets ?movement.
  ?movement rdf:type rr:JointMovements.
  ?movement rdf:type ?fmaConcept.
  ?fmaConcept rdfs:subClassOf ?superClass.
  ?superClass rdfs:label 'Muscle of lower limb'.
  ?fmaConcept fma:attaches_to ?attachedBodyPart.
  ?attachedBodyPart rdfs:label 'Greater trochanter'.
}
```

In FMA, subclasses of “muscle of pelvic girdle” such as “Piriformis” or “Gluteus minimus” attach to “Greater trochanter”. We can get this information by matching robots that target pelvic girdle movements to the (possibly particular) muscles of pelvic girdle.

FMA8. What are the rehabilitation robots that can affect some body parts that contains some sacro-iliac joint?

```
SELECT DISTINCT ?robotName
WHERE {
```

```
?robot rdf:type rr:RehabRobots.  
?robot rr:has_Name ?robotName.  
?robot rr:targets ?movement.  
?movement rdf:type rr:JointMovements.  
?movement rdf:type ?fmaConcept.  
?bodyPart fma:part_of ?fmaConcept.  
?bodyPart rdfs:label 'Sacro-iliac joint'.  
}
```

In FMA, “left sacro-iliac joint” and “right sacro-iliac joint” (subclasses of “sacro-iliac joint”) are parts of “Left side of bony pelvis” and “Right side of bony pelvis”, respectively. We use “contains” for the inverse of “part of”.

C Example Queries over DO and REHABROBO- ONTO

DO4. What are the rehabilitation robots that can be used to treat 'cerebrovascular accident'?

```
SELECT DISTINCT ?robotName
WHERE {
    ?robot1 rdf:type rr:RehabRobots.
    ?robot rr:has_Name ?robotName.
    ?robot rr:targets ?movement.
    ?movement rdf:type rr:JointMovements.
    ?movement rdf:type ?doConcept.
    ?doConcept rdfs:label 'cerebrovascular accident'.
}
```

In FMA, “cerebrovascular accident” is used as a synonym of “stroke”. Some particular movements in REHABROBO-ONTO can be matched to this class.

DO5. What are the diseases that can be treated using some rehabilitation robots that target the shoulder scapular elevation/depression?

```
SELECT DISTINCT ?diseaseLabel
WHERE {
    ?robot rdf:type rr:RehabRobots.
    ?robot rr:targets ?movement.
    ?movement rdf:type rr:ShoulderScapularElevation/Depression.
    ?robot rr:targets ?movement2.
```

```

?movement2 rdf:type rr:JointMovements.
?movement2 rdf:type ?doConcept.
?doConcept rdfs:label ?diseaseLabel.
}

```

DO6. What are the rehabilitation robots that can be used to treat 'frozen shoulder'?

```

SELECT DISTINCT ?robotName
WHERE {
  ?robot1 rdf:type rr:RehabRobots.
  ?robot rr:has_Name ?robotName.
  ?robot rr:targets ?movement.
  ?movement rdf:type rr:JointMovements.
  ?movement rdf:type ?doConcept.
  ?doConcept rdfs:label 'Frozen Shoulder'.
}

```

“Frozen shoulder” concept in DO can be mapped to particular shoulder movements in REHABROBO-ONTO.

DO7. What are the muscular dystrophies that may be treated using rehabilitation robot X?

```

SELECT DISTINCT ?diseaseLabel
WHERE {
  ?robot rdf:type rr:RehabRobots.
  ?robot rr:has_Name 'X'.
  ?robot rr:targets ?movement.
}

```

```

?movement rdf:type rr:JointMovements.
?movement rdf:type ?doConcept.
?doConcept rdfs:label ?diseaseLabel.
?doConcept rdfs:subClassOf ?superClass.
?superClass rdfs:label 'Muscular Dystrophy'.
}

```

In DO, “Muscular Dystrophy” has some subclasses that can be mapped to REHABROBO-ONTO movements, such as “Distal Muscular Dystrophy” or “Limb-Girdle Muscular Dystrophy”. They can be mapped to distal joint movements or pelvic girdle movements in REHABROBO-ONTO, respectively.

DO8. What are publications that reference some rehabilitation robots that can be used to treat 'hip enthesopathy'?

```

SELECT DISTINCT ?publicationTitle
WHERE {
    ?publicationTitle rr:has_Title ?publication.
    ?publication rdf:type rr:References.
    ?robot rr:has_Reference ?publication.
    ?robot rdf:type rr:RehabRobots.
    ?robot rr:targets ?movement.
    ?movement rdf:type rr:JointMovements.
    ?movement rdf:type ?doConcept.
    ?doConcept rdfs:label 'hip enthesopathy'.
}

```

DO9. What are the diseases that can be treated using some rehabilitation robots that target the wrist flexion/extension?

```

SELECT DISTINCT ?diseaseLabel
WHERE {
    ?robot rdf:type rr:RehabRobots.
    ?robot rr:targets ?movement.
    ?movement rdf:type rr:WristFlexion/Extension.
    ?movement rdf:type ?doConcept.
    ?doConcept rdfs:label ?diseaseLabel.
}

```

DO10. What are the rehabilitation robots that have pediatric targeted population and that can be used to treat spastic hemiplegia?

```

SELECT DISTINCT ?robotName
WHERE {
    ?robot rdf:type rr:RehabRobots.
    ?robot rr:has_Name ?robotName.
    ?robot rr:has_Targeted_Population 'pediatric'.
    ?robot rr:targets ?movement.
    ?movement rdf:type rr:JointMovements.
    ?movement rdf:type ?doConcept.
    ?doConcept rdfs:label 'spastic hemiplegia'.
}

```

In DO, “spastic hemiplegia” is a subclass of “spastic cerebral palsy”, which is the subclass of “cerebral palsy”, under brain diseases. They can be mapped to particular movements in REHABROBO-ONTO.

DO11. What types of spinal muscular atrophy can be treated using rehabilitation robot X?

```

SELECT DISTINCT ?diseaseLabel
WHERE {
    ?robot rdf:type rr:RehabRobots.
    ?robot rr:has_Name 'X'.
    ?robot rr:targets ?movement.
    ?movement rdf:type rr:JointMovements.
    ?movement rdf:type ?doConcept.
    ?doConcept rdfs:label ?diseaseLabel.
    ?doConcept rdfs:subClassOf ?superClass.
    ?superClass rdfs:label 'Spinal Muscular Atrophy'.
}

```

DO12. What are the publications that reference some rehabilitation robots that can be used to treat nervous system diseases?

```

SELECT DISTINCT ?publicationTitle
WHERE {
    ?publicationTitle rr:has_Title ?publication.
    ?publication rdf:type rr:References.
    ?robot rr:has_Reference ?publication.
    ?robot rdf:type rr:RehabRobots.
    ?robot rr:targets ?movement.
    ?movement rdf:type rr:JointMovements.
    ?movement rdf:type ?doConcept.
    ?doConcept rdfs:label 'nervous system disease'.
}

```

In DO, “nervous system disease” has subclasses such as “spinal cord disease”, “cerebral palsy”. They have further subclasses and they can be mapped to particular movements in REHABROBO-ONTO.

References

- [1] *Secure Hash Standard*. National Institute of Standards and Technology, Washington, 1995. Federal Information Processing Standard 180-1.
- [2] Sesame: A generic architecture for storing and querying rdf and rdf schema. In Ian Horrocks and James Hendler, editors, *The Semantic Web — ISWC 2002*, volume 2342 of *Lecture Notes in Computer Science*, 2002.
- [3] F. Amigoni and M.A. Neri. An application of ontology technologies to robotic agents. In *Proc. of IEEE/WIC/ACM International Conference on Intelligent Agent Technology*, pages 751–754, 2005.
- [4] Grigoris Antoniou and Frank Van Harmelen. Web ontology language: Owl. In *Handbook on Ontologies in Information Systems*, pages 67–92. Springer, 2003.
- [5] F. Baader, C. Lutz, and B. Suntisrivaraporn. CEL—a polynomial-time reasoner for life science ontologies. In U. Furbach and N. Shankar, editors, *Proceedings of the 3rd International Joint Conference on Automated Reasoning (IJCAR’06)*, volume 4130 of *Lecture Notes in Artificial Intelligence*, pages 287–291. Springer-Verlag, 2006.
- [6] Franz Baader, Ian Horrocks, and Ulrike Sattler. Description Logics. In Frank van Harmelen, Vladimir Lifschitz, and Bruce Porter, editors, *Handbook of Knowledge Representation*, chapter 3, pages 135–180. Elsevier, 2008.

- [7] Alexander De Leon Battista, Natalia Villanueva-Rosales, Myroslav Palenychka, and Michel Dumontier. SMART: A web-based, ontology-driven, semantic web query answering application. In *Semantic Web Challenge*, volume 295, 2007.
- [8] David Beckett. The design and implementation of the redland rdf application framework. In *Proceedings of the 10th international conference on World Wide Web*, WWW '01, pages 449–456, 2001.
- [9] L. Belouaer, M. Bouzid, and A. Mouaddib. Ontology based spatial planning for human-robot interaction. In *Temporal Representation and Reasoning (TIME), 2010 17th International Symposium on*, pages 103–110, 2010.
- [10] Hendler J. Berners-Lee, T. and O. Lassila. The semantic web. *Scientific American*, 2001.
- [11] Abraham Bernstein and Esther Kaufmann. GINO - a guided input natural language ontology editor. In *Proceedings of the 5th international conference on The Semantic Web*, pages 144–157, 2006.
- [12] F. Bobillo and U. Straccia. fuzzydl: An expressive fuzzy description logic reasoner. In *Fuzzy Systems, 2008. FUZZ-IEEE 2008. (IEEE World Congress on Computational Intelligence). IEEE International Conference on*, pages 923–930, 2008.
- [13] Jonathan Chaffer and Karl Swedberg. *Learning jquery: better interaction design and web development with simple javascript techniques*. Packt Publishing, 2007.

- [14] Antonio Chella, Massimo Cossentino, Roberto Pirrone, and Andrea Ruisi. Modeling ontologies for robotic environments. In *Proc. of SEKE*, pages 77–80, 2002.
- [15] Philipp Cimiano, Peter Haase, Jörg Heizmann, Matthias Mantel, and Rudi Studer. Towards portable natural language interfaces to knowledge bases - the case of the ORAKEL system. *Data Knowl. Eng.*, 65(2):325–354, 2008.
- [16] Hamish Cunningham, Diana Maynard, Kalina Bontcheva, Valentin Tablan, Niraj Aswani, Ian Roberts, Genevieve Gorrell, Adam Funk, Angus Roberts, Danica Damljanovic, Thomas Heitz, Mark A. Greenwood, Horacio Saggion, Johann Petrak, Yaoyong Li, and Wim Peters. *Text Processing with GATE (Version 6)*. 2011.
- [17] Carlos Viegas Damásio, Anastasia Analyti, and Grigoris Antoniou. Provenance for sparql queries. In *The Semantic Web–ISWC 2012*, pages 625–640. Springer, 2012.
- [18] Danica Damljanovic, Milan Agatonovic, and Hamish Cunningham. FREyA: an interactive way of querying linked data using natural language. In *Proceedings of the 8th international conference on The Semantic Web, ESWC’11*, pages 125–138, 2012.
- [19] Marie-Catherine de Marneffe, Bill MacCartney, and Christopher D. Manning. Generating typed dependency parses from phrase structure trees. In *LREC*, 2006.

- [20] Esra Erdem, Yelda Erdem, Halit Erdogan, and Umut Öztok. Finding answers and generating explanations for complex biomedical queries. In *AAAI*, 2011.
- [21] Esra Erdem, Halit Erdogan, and Umut Oztok. BIOQUERY-ASP: Querying biomedical ontologies using answer set programming. In *Proc. of RuleML2011@BRF Challenge*, 2011.
- [22] Orri Erling and Ivan Mikhailov. Rdf support in the virtuoso dbms. In Tassilo Pellegrini, Sören Auer, Klaus Tochtermann, and Sebastian Schaffert, editors, *Networked Knowledge - Networked Media*, volume 221 of *Studies in Computational Intelligence*, pages 7–24, 2009.
- [23] C. Fellbaum. *WordNet: An Electronic Lexical Database*. Language, Speech and Communication. Mit Press, 1998.
- [24] D. Y. S. Fernandes. *Using Semantics to Enhance Query Reformulation in Dynamic Distributed Environments*. PhD thesis, Federal University of Pernambuco, 2009.
- [25] Oscar Ferrández, Rubén Izquierdo, Sergio Ferrández, and José Luis Vicedo. Addressing ontology-based question answering with collections of user queries. *Information Processing and Management*, 45(2):175 – 188, 2009.
- [26] Anette Frank, Hans-Ulrich Krieger, Feiyu Xu, Hans Uszkoreit, Berthold Crysmann, Brigitte Jörg, and Ulrich Schäfer. Question answering from structured knowledge sources. *Journal of Applied Logic*, 5(1):20 – 48, 2007.

- [27] Martin Gebser, Benjamin Kaufmann, André Neumann, and Torsten Schaub. clasp: A conflict-driven answer set solver. In *Logic Programming and Nonmonotonic Reasoning*, pages 260–265. Springer, 2007.
- [28] John H. Gennari, Mark A. Musen, Ray W. Ferguson, William E. Grosso, Monica Crubézy, Henrik Eriksson, Natalya Fridman Noy, and Samson W. Tu. The evolution of Protégé: an environment for knowledge-based systems development. *Int. J. Hum.-Comput. Stud.*, 58(1):89–123, 2003.
- [29] Erich Grädel, Phokion G. Kolaitis, and Moshe Y. Vardi. On the decision problem for two-variable first-order logic. *Bulletin of Symbolic Logic*, 3(1):53–69, 1997.
- [30] Volker Haarslev, Kay Hidde, Ralf Möller, and Michael Wessel. The RacerPro knowledge representation and reasoning system. *Semantic Web*, 3(3):267–277, 2012.
- [31] Peter Haase, Holger Lewen, Rudi Studer, Duc Thanh Tran, Michael Erdmann, Mathieu d’Aquin, and Enrico Motta. The neon ontology engineering toolkit. *WWW*, 2008.
- [32] Matthew Horridge and Sean Bechhofer. The owl api: A java api for owl ontologies. *Semantic Web*, 2(1):11–21, 2011.
- [33] Ian Horrocks, Peter F Patel-Schneider, Harold Boley, Said Tabet, Benjamin Grosz, Mike Dean, et al. SWRL: A semantic web rule language combining OWL and RuleML. *W3C Member submission*, 21:79, 2004.

- [34] Ian Horrocks, Peter F. Patel-Schneider, and Frank Van Harmelen. From shiq and rdf to owl: The making of a web ontology language. *Journal of Web Semantics*, 1:2003, 2003.
- [35] Ullrich Hustadt and RenateA. Schmidt. Mspass: Modal reasoning by translation and first-order resolution. In Roy Dyckhoff, editor, *Automated Reasoning with Analytic Tableaux and Related Methods*, volume 1847 of *Lecture Notes in Computer Science*, pages 67–71, 2000.
- [36] Minsu Jang and Joo-Chan Sohn. Bossam: An extended rule engine for owl inferencing. In Grigoris Antoniou and Harold Boley, editors, *Rules and Rule Markup Languages for the Semantic Web*, volume 3323 of *Lecture Notes in Computer Science*, pages 128–138. Springer Berlin Heidelberg, 2004.
- [37] Benjamin Johnston, Fangkai Yang, Rogan Mendoza, Xiaoping Chen, and Mary-Anne Williams. Ontology based object categorization for robots. In *Proc. of PAKM*, pages 219–231, 2008.
- [38] Alex Juarez, Christoph Bartneck, and Loe M. G. Feijs. Using semantic technologies to describe robotic embodiments. In *Proc. of HRI*, pages 425–432, 2011.
- [39] Dongyeop Kang, Eugene Seo, Sookyung Kim, and Ho-Jin Choi. Automatically learning robot domain ontology from collective knowledge for home service robots. In *Advanced Communication Technology, 2009. ICACT 2009. 11th International Conference on*, volume 03, pages 1766–1771, 2009.

- [40] Esther Kaufmann, Abraham Bernstein, and Lorenz Fischer. NLP-Reduce: A naive but domain-independent natural language interface for querying ontologies. In *4th European Semantic Web Conference*, 2007.
- [41] Esther Kaufmann, Abraham Bernstein, and Renato Zumstein. Querix: A natural language interface to query ontologies based on clarification dialogs. In *5th ISWC*, pages 980–981. Springer, 2006.
- [42] S. Kobayashi, S. Tamagawa, T. Morita, and T. Yamaguchi. Intelligent humanoid robot with japanese wikipedia ontology and robot action ontology. In *Human-Robot Interaction (HRI), 2011 6th ACM/IEEE International Conference on*, pages 417–424, 2011.
- [43] Yuanguai Lei, Victoria Uren, and Enrico Motta. SemSearch: A search engine for the semantic web. In *Proc. 5th International Conference on Knowledge Engineering and Knowledge Management Managing Knowledge in a World of Networks, Lect. Notes in Comp. Sci.*, pages 238–245. Springer-Verlag, 2006.
- [44] Reuven M. Lerner. At the forge: twitter bootstrap. *Linux J.*, 2012(218), 2012.
- [45] Vladimir Lifschitz. What is answer set programming?. In *AAAI*, volume 8, pages 1594–1597, 2008.
- [46] Vanessa Lopez, Miriam Fernández, Enrico Motta, and Nico Stieler. PowerAqua: Supporting users in querying and exploring the semantic web. *Semantic Web*, 3(3):249–265, 2012.
- [47] Vanessa Lopez, Victoria Uren, Enrico Motta, and Michele Pasin. Aqua-Log: An ontology-driven question answering system for organizational

- semantic intranets. *Web Semantics: Science, Services and Agents on the World Wide Web*, 5(2):72–105, 2007.
- [48] Brian Lowe, Brian Caruso, Nick Cappadona, Miles Worthington, Stella Mitchell, and Jon Corson-Rikert. The vitro integrated ontology editor and semantic web application. In *ICBO*, 2011.
- [49] C.J. Matheus, K. Baclawski, and M.M. Kokar. BaseVISor: A triples-based inference engine outfitted to process ruleml and r-entailment rules. In *Rules and Rule Markup Languages for the Semantic Web, Second International Conference on*, pages 67–74, 2006.
- [50] Brian McBride. Jena: Implementing the rdf model and syntax specification. In *SemWeb*, 2001.
- [51] Boris Motik. KAON2 - Scalable reasoning over ontologies with large data sets. *ERCIM News*, (72), 2008.
- [52] A. Nilsson, R. Muradore, K. Nilsson, and P. Fiorini. Ontology for robotics: A roadmap. In *Advanced Robotics, 2009. ICAR 2009. International Conference on*, pages 1–6, 2009.
- [53] Giorgio Orsi. *Context Based Querying of Dynamic and Heterogeneous Information Sources*. PhD thesis, Politecnico di Milano, 2011.
- [54] John D. Osborne, Jared Flatow, Michelle Holko, Simon M. Lin, Warren A. Kibbe, Lihua J. Zhu, Maria I. Danila, Gang Feng, and Rex L. Chisholm. Annotating the human genome with disease ontology. *BMC genomics*, 2009.

- [55] Massimo Paolucci and Katia Sycara. Ontologies in agent architectures. In Rudi Studer Steffen Staab, editor, *Handbook on Ontologies in Information Systems*. Springer, 2004.
- [56] L. Paull, G. Severac, G.V. Raffo, J.M. Angel, H. Boley, P.J. Durst, W. Gray, M. Habib, B. Nguyen, S.V. Ragavan, G. Sajad Saeedi, R. Sanz, M. Seto, A. Stefanovski, M. Trentini, and H. Li. Towards an ontology for autonomous robots. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 1359–1364, 2012.
- [57] Jorge Pérez, Marcelo Arenas, and Claudio Gutierrez. Semantics and complexity of sparql. In *The Semantic Web-ISWC 2006*, pages 30–43. Springer, 2006.
- [58] Eric Prud’Hommeaux, Andy Seaborne, et al. Sparql query language for rdf. *W3C recommendation*, 15, 2008.
- [59] M. Rokunuzzaman, K. Sekiyama, and T. Fukuda. Common region of interest generation between mobile robots by cognitive ontology. In *Electrical Computer Engineering (ICECE), 2012 7th International Conference on*, pages 117–120, 2012.
- [60] Cornelius Rosse and José L. V. Mejino. *The foundational model of anatomy ontology*, pages 59–117. Springer, 2007.
- [61] C. Schlenoff, E. Prestes, R. Madhavan, P. Goncalves, H. Li, S. Balakirsky, T. Kramer, and E. Miguelanez. An IEEE standard ontology for robotics and automation. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 1337–1342, 2012.

- [62] Craig Schlenoff and Elena Messina. A robot ontology for urban search and rescue. In *Proc. of CIKM-KRAS*, pages 27–34, 2005.
- [63] Manfred Schmidt-Schauß and Gert Smolka. Attributive concept descriptions with complements. *Artificial Intelligence*, 48(1):1–26, 1991.
- [64] Peter P. Schneider and Bill Swartout. Description-logic knowledge representation system specification from the KRSS group of the ARPA knowledge sharing effort. Technical report, DARPA Knowledge Representation System Specification (KRSS) Group of the Knowledge Sharing Initiative, 1993.
- [65] Andy Seaborne. RDQL - A Query Language for RDF. Technical report, W3C (proposal), 2004.
- [66] Rob Shearer, Boris Motik, and Ian Horrocks. HermiT: A Highly-Efficient OWL Reasoner. In Alan Ruttenberg, Ulrike Sattler, and Cathy Dolbear, editors, *Proc. of the 5th Int. Workshop on OWL: Experiences and Directions (OWLED 2008 EU)*, Karlsruhe, Germany, October 26–27 2008.
- [67] Evren Sirin, Bijan Parsia, Bernardo Cuenca Grau, Aditya Kalyanpur, and Yarden Katz. Pellet: A practical owl-dl reasoner. *Web Semantics: Science, Services and Agents on the World Wide Web*, 5(2):51 – 53, 2007.
- [68] Il Hong Suh, Gi Hyun Lim, Wonil Hwang, Hyowon Suh, Jung-Hwa Choi, and Young-Tack Park. Ontology-based multi-layered robot knowledge framework (omrkf) for robot intelligence. In *Proc. of IROS*, pages 429–436, 2007.

- [69] Valentin Tablan, Danica Damljanovic, and Kalina Bontcheva. A natural language query interface to structured information. In *Proceedings of the 5th European semantic web conference on The semantic web: research and applications*, ESWC'08, pages 361–375, 2008.
- [70] M. Tenorth, A.C. Perzylo, R. Lafrenz, and M. Beetz. The roboearth language: Representing and exchanging knowledge about actions, objects, and environments. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 1284–1289, 2012.
- [71] Dmitry Tsarkov and Ian Horrocks. FaCT++ Description Logic Reasoner: System Description. In Ulrich Furbach and Natarajan Shankar, editors, *Automated Reasoning*, volume 4130 of *Lecture Notes in Computer Science*, pages 292–297. Springer Berlin Heidelberg, 2006.
- [72] M. Uschold and M. King. Towards a methodology for building ontologies. In *Proc. of Workshop on Basic Ontological Issues in Knowledge Sharing, held in conjunction with IJCAI-95*, 1995.
- [73] R. Valencia-García, F. García-Sánchez, D. Castellanos-Nieves, and J.T. Fernández-Breis. OWLPath: An OWL ontology-guided query editor. *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, 41(1):121–136, 2011.
- [74] Chong Wang, Miao Xiong, Qi Zhou, and Yong Yu. PANTO: A portable natural language interface to ontologies. In *Proceedings of the 4th European conference on The Semantic Web: Research and Applications*, ESWC '07, pages 473–487, 2007.

- [75] Eric Wang, Yong Se Kim, Hak Soo Kim, Jin Hyun Son, Sanghoon Lee, and Il Hong Suh. Ontology modeling and storage system for robot context understanding. In *Proc. of KES (3)*, pages 922–929, 2005.
- [76] M. Yalcin and V. Patoglu. Kinematics and design of AssistOn-SE: A self-adjusting shoulder-elbow exoskeleton. In *Proc. of IEEE BioRob*, pages 1579–1585, 2012.
- [77] Holly A. Yanco and Jill L. Drury. Classifying human-robot interaction: an updated taxonomy. In *Proc. of SMC (3)*, pages 2841–2846, 2004.
- [78] Qi Zhou, Chong Wang, Miao Xiong, Haofen Wang, and Yong Yu. SPARK: adapting keyword query to semantic search. In *Proceedings of the 6th international The semantic web and 2nd Asian conference on Asian semantic web conference, ISWC'07/ASWC'07*, pages 694–707, 2007.