REHABROBO-QUERY: Answering Natural Language Queries about Rehabilitation Robotics Ontology on the Cloud

Zeynep Dogmus and Esra Erdem and Volkan Patoglu
Faculty of Engineering and Natural Sciences, Sabancı University, Istanbul, Turkey
E-mail: {zeynepdogmus,esraerdem,vpatoglu}@sabanciuniv.edu

Abstract We introduce a novel method to answer natural language queries about rehabilitation robotics, over the formal ontology REHABROBO-ONTO. For that, (i) we design and develop a novel controlled natural language for rehabilitation robotics, called REHABROBO-CNL; (ii) we introduce translations of queries in REHABROBO-CNL into SPARQL queries, utilizing a novel concept of query description trees and description logics concepts; (iii) we use an automated reasoner to find answers to SPARQL queries. To facilitate the use of our method by experts, we develop an intelligent, interactive query answering system, called REHABROBO-QUERY, using Semantic Web technologies, and make it available on the cloud via Amazon web services. REHABROBO-QUERY guides the users to express their queries in natural language and displays the answers to queries in a readable format, possibly with links to detailed information. Easy access to information on REHABROBO-ONTO through complex queries in natural language may help engineers inspire new rehabilitation robot designs, while also guiding practitioners to make more informed decisions on technology based rehabilitation.

Keywords: Ontology systems, query answering, rehabilitation robotics, intelligent user-interfaces, controlled natural languages

1. Introduction

We have earlier designed and developed the first formal rehabilitation robotics ontology, called REHABROBO-ONTO, in OWL (Web Ontology Language) [1,19], and made it available on the cloud via Amazon Web Services [8,9,11] (in particular, Amazon Elastic Compute Cloud)\(^1\) so that every rehabilitation robotics researcher can easily add information about his/her robot to it, and modify this information. To facilitate such modifications of the rehabilitation robotics ontology REHABROBO-ONTO, we have also developed a Web-based software (called REHABROBO-QUERY)\(^2\) 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 and modification. REHABROBO-QUERY also utilizes Amazon Web Services for cloud computing. For further information about the design, development and maintenance of REHABROBO-ONTO, and how REHABROBO-QUERY facilitates modification of REHABROBO-ONTO, we refer the reader to our earlier article [11].

This article is concerned about reasoning about rehabilitation robotics over REHABROBO-ONTO. Note that a structured representation of information about rehabilitation robotics as the ontology REHABROBO-ONTO allows rehabilitation robotics researchers to learn various properties of the existing robots and access to the related publications to further improve the state-of-the-art. Physical medicine experts also can find information about rehabilitation robots and related publications to better identify the right robot for a particular therapy or patient population. Such requested information can be obtained from REHABROBO-ONTO by expressing the requested information as a

---

\(^1\)http://aws.amazon.com/ec2/

\(^2\)http://hmi.sabanciuniv.edu/?page_id=781

1570-0844/0-1900/$35.00 © 0 – IOS Press and the authors. All rights reserved
formal query in a query language, such as SPARQL [28], and then by computing answers to these queries by using a state-of-the-art automated reasoner, such as Pellet [29]. However, expressing the requested information as a formal query by means of formulas is challenging for many users, including robot designers and physical medicine experts. For that reason, we particularly focus on the process of query answering over REHABROBO-ONTO, and making it easier for the users to express their queries in a natural language and to obtain answers to their queries automatically.

Towards these goals, the main contributions of this article can be summarized as follows. For expressing queries about rehabilitation robotics, we design and develop a novel controlled natural language for rehabilitation robots, called REHABROBO-CNL. For automatically computing answers to natural language queries, we introduce an algorithm that transforms natural language queries in REHABROBO-CNL into formal queries in SPARQL. This transformation utilizes two intermediate representations: a novel tree structure, called a Query Description Tree (QDT), and Description Logics (DL) concepts. Once the natural language query is transformed into a formal query, Pellet is used to find relevant answers to the query. The software system REHABROBO-QUERY is extended with an interactive, intelligent user interface to guide the users to enter natural language queries in REHABROBO-CNL about the existing robots, and to present the answers in an understandable form, so that the users do not have to know about the logical formalism of the ontology or the formalism to represent queries, or the use of the technologies for computing answers to their questions about rehabilitation robots.

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. This further paves the way for translational physical medicine (from bench-to-bed and back) and personalized physical medicine.

The rest of the article is organized as follows. We present the controlled natural language REHABROBO-CNL for expressing queries about rehabilitation robotics (Section 3), and discuss the extension of REHABROBO-QUERY to support querying in REHABROBO-CNL via an interactive, intelligent user interface (Section 4). We present our transformation of a REHABROBO-CNL query into a SPARQL query (Section 5) whose answer can be computed using Pellet (Section 6). After we summarize the related work about software systems that support natural language queries over ontologies (Section 7), we conclude with a summary of our contributions (Section 8).

This article extends our earlier paper [10], presented at the International Conference on Knowledge Engineering and Ontology Development (KEOD 2014), by describing more details about the query answering system integrated in REHABROBO-QUERY, the query language REHABROBO-CNL, and the transformation algorithm.

## 2. A Brief Review of REHABROBO-ONTO

REHABROBO-ONTO is the first formal rehabilitation robotics ontology that represents knowledge about rehabilitation robotics in a structured form, and allows automated reasoning about this knowledge. It is developed in line with the efforts of European Network on Robotics for Neurorehabilitation\(^3\), for standardizing terminology as well as assessment measures for rehabilitation robots.

REHABROBO-ONTO has been designed by considering suggestions of various rehabilitation robotics researchers and physical medicine experts, by first identifying the purpose, and then the basic concepts and their thematic classes, and their relationships. The main classes and their relationships are illustrated in Figure 1.

\(^3\)http://www.rehabilitationrobotics.eu/
REHABROBO-ONTO has 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).

The main classes RehabRobots, JointMovements and Assessments have subclasses. Currently there are 147 classes represented in REHABROBO-ONTO.

The main 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.

REHABROBO-ONTO is developed using the ontology editor PROTEGE [18], in the ontology language OWL 2 DL, with OWL/XML syntax. It is open-source and available on the cloud via Amazon Web Services. Its maintenance (i.e., adding, deleting, modifying information about rehabilitation robots) can be done as part of the ontology system REHABROBO-QUERY.

For further information about the design and development of REHABROBO-ONTO, and how REHABROBO-QUERY facilitates maintenance of REHABROBO-ONTO on the cloud, we refer the reader to our earlier article [11].
3. REHABROBO-CNL: A Controlled Natural Language for Queries about Rehabilitation Robotics

Reasoning over REHABROBO-ONTO can be done by means of answering questions posed by the user in a natural language. However, natural languages may have ambiguities in their vocabularies and grammars. To overcome ambiguities of a natural language, a subset of it with a restricted vocabulary and grammar can be considered for specific domains; such languages are called Controlled Natural Languages (CNLs) [22]. Essentially, with its restricted vocabulary and grammar, a CNL is a formal language. Therefore, it is easier to convert a CNL (compared to a more general natural language) to a logic-based formalism. In that sense, a CNL facilitates the use of automated reasoners to find answers to queries expressed in a CNL. Due to these reasons, first we have identified the sorts of queries about rehabilitation robots, then we have designed and developed a new CNL, called REHABROBO-CNL, to express these queries in natural language.

3.1. Sorts of queries supported by REHABROBO-CNL

The variety of queries is important since the users may include not only rehabilitation robot designers and developers, but also physical therapists and medical doctors. Also, complex queries that not only require intelligent extraction of different sorts of knowledge, but their integration via conjunctions, disjunctions, negations, nested relative clauses, aggregates, and quantifications are inevitable in rehabilitation robotics, since robots, targeted movements, evaluation metrics, etc. have many features that are of interest to users. We have designed REHABROBO-CNL considering these aspects.

We have identified different sorts of complex queries useful from the perspectives of rehabilitation robotics and physical medicine. Some examples of these queries are presented in the following, with respect to their topics.

Queries about mechanical properties of rehabilitation robots These are queries to extract information about the rehabilitation robots whose information is available in REHABROBO-ONTO. Here are some examples:
- What are the robots that target shoulder movements and that have at least 2 active degrees of freedom?
- What are the shoulder robots that target flexion/extension movements and that do not target elevation/depression movements?
- What are the robots with active degree of freedom $\geq 3$ and that target all pelvic girdle movements with backdrivability type = ‘passive’?
- What are the ankle robots that target movements with electrical actuation and with cable drive transmission?

Queries about joint movements targeted by rehabilitation robots These are queries to extract information about joint movements targeted by rehabilitation robots in REHABROBO-ONTO. Here are some examples:
- What are the movements that are targeted by some robots with (some intervention time or with all targeted disorders)?
- What are the movements that are targeted by the robot ‘AssistOn-Finger’ and with minimum range of motion $\geq 20$°?
- What are the movements that are not targeted by any robots with kinematic type = ‘redundant’ and with mechanism type $\neq$ ‘parallel’?

According to REHABROBO-ONTO, for instance, some part of the answer to the first question is as follows:

<table>
<thead>
<tr>
<th>Movement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index Finger DIP Flexion/Extension</td>
</tr>
<tr>
<td>Shoulder Scapular Elevation/Depression</td>
</tr>
<tr>
<td>Wrist Flexion/Extension</td>
</tr>
</tbody>
</table>

Queries about the evaluation metrics for rehabilitation robots These are queries to extract information about metrics used to evaluate rehabilitation robots. Here are some examples:
- What are the effort metrics that are evaluated by some robots with active degree of freedom $\geq 2$?
- What are the movement quality metrics that are evaluated by all robots with motion capability = ‘grounded’?
- What are the kinematic aspect metrics that are evaluated by some robots that target all elbow movements?
- What are the muscle strength metrics that are evaluated by robots that target all wrist movements with transmission = ‘direct drive’?
What are the psychomotoric aspect metrics that are evaluated by all robots with kinematic type = ‘redundant’ and that target all ankle movements with minimum range of motion $\geq 30$? 

According to REHABROBO-ONTO, for instance, some part of the answer to the first question is as follows:

- Time To Initiate Movement
- Amount Of Compensation
- Biomechanical Work
- Energy
- Power

Some part of the answer to the second question is as follows:

- Visuomotor Coordination
- Combined Task Coordination
- Single Joint Coordination

Other queries These are queries to extract information about publications about rehabilitation robots, owners/institutes/laboratories of the robots. Here are some examples:

- What are the publications with clinical study and that do not reference any robots with active degree of freedom $\geq 1$?
- What are the publications without clinical study or that reference some robots that do not evaluate any movement quality metrics?
- 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’?
- What are the users that own/maintain some robots that target all ankle movements?

3.2. Grammar of REHABROBO-CNLI

The grammar of REHABROBO-CNLI is shown in Table 1. To eliminate the ambiguities in nesting of conjunctions and disjunctions, REHABROBO-CNLI provides two ways of constructing a query: A query in REHABROBO-CNLI should either be in Conjunctive Normal Form (CNF), or in Disjunctive Normal Form (DNF). In other words, REHABROBO-CNLI 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.

- 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 functions (in italic font) in the grammar extract relevant information from REHABROBO-ONTO. These ontology functions are described in Table 2. 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 such a matching of 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 in expressions of the form “Type() that Verb()” are shown in Table 3. Note that these matchings essentially come from the structure of REHABROBO-ONTO, e.g., concept names and relation names.

Similarly, it is necessary to match verbs with types in expressions of the form “Verb() (some | all | any∗ | the) Type()”. Table 4 lists the available types that can occur after a verb in the query (e.g., in a RELATIVECLAUSE), and 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 some subclasses are extracted. Here is an example query:

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
Table 1

The Grammar of REHABROBO-CNLP

<table>
<thead>
<tr>
<th>QUERY →</th>
<th>WHATQUERY QUESTIONMARK</th>
</tr>
</thead>
<tbody>
<tr>
<td>WHATQUERY →</td>
<td>What are the Type() GENERALRELATION</td>
</tr>
<tr>
<td>GENERALRELATION →</td>
<td>SIMPLERELATION NESTEDRELATION*</td>
</tr>
<tr>
<td>SIMPLERELATION →</td>
<td>(that RELATIVECLAUSE)+</td>
</tr>
<tr>
<td>SIMPLERELATION →</td>
<td>that INSTANCERELATION</td>
</tr>
<tr>
<td>SIMPLERELATION →</td>
<td>WITHRELATION</td>
</tr>
<tr>
<td>NESTEDRELATION →</td>
<td>(and LP SIMPLEDISJUNCTION RP)*</td>
</tr>
<tr>
<td>NESTEDRELATION →</td>
<td>(or LP SIMPLECONJUNCTION RP)*</td>
</tr>
<tr>
<td>SIMPLECONJUNCTION →</td>
<td>(SIMPLERELATION and)* SIMPLERELATION</td>
</tr>
<tr>
<td>RELATIVECLAUSE →</td>
<td>Verb() (some</td>
</tr>
<tr>
<td>RELATIVECLAUSE →</td>
<td>NEG Verb() any Type()</td>
</tr>
<tr>
<td>INSTANCERELATION →</td>
<td>NEG? Verb() the Type() Instance()</td>
</tr>
<tr>
<td>WITHRELATION →</td>
<td>with Noun() EQCHECK Value()+</td>
</tr>
<tr>
<td>WITHRELATION →</td>
<td>with QUANTIFIER Noun()</td>
</tr>
<tr>
<td>WITHRELATION →</td>
<td>(with</td>
</tr>
<tr>
<td>EQCHECK →</td>
<td>=</td>
</tr>
<tr>
<td>QUANTIFIER →</td>
<td>some</td>
</tr>
<tr>
<td>NEG →</td>
<td>Neg()</td>
</tr>
<tr>
<td>LP →</td>
<td>(</td>
</tr>
<tr>
<td>RP →</td>
<td>)</td>
</tr>
<tr>
<td>QUESTIONMARK →</td>
<td>?</td>
</tr>
</tbody>
</table>

Table 2

The Ontology Functions

| Type() | Returns the types that correspond to concept names. They are: Robots, movements, users, publications and metrics. |
| Instance() | Returns robot names for robots and user names for users. |
| Verb() | 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. |
| Noun() | Returns the nouns that correspond to data properties. ex. targeted disorder, active degree of freedom. |
| Value() | Returns the suitable values according to a given noun. Corresponds to the pre-defined ranges of data properties. |
| Neg() | Returns a suitable negation phrase. These phases are: do not, are not. |

movements but wrist radial deviation/ulnar deviation, then this robot will not be included in the answer to this query.

In REHABROBO-CNLP, 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 3

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

Table 4

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

* any is used after negative verbs.

Table 5

<table>
<thead>
<tr>
<th>Type()</th>
<th>Instance()</th>
</tr>
</thead>
<tbody>
<tr>
<td>robot</td>
<td>name of the robot</td>
</tr>
<tr>
<td>user</td>
<td>username of the user</td>
</tr>
</tbody>
</table>

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 5 demonstrates the relevant properties of the instances that appear when a type is selected.

In addition to types and verbs, types are matched with the relevant nouns, as shown in Table 6. For instance, control modes are matched with robots whereas actuation is matched with movements. Note that these matchings are due to properties of concepts in REHABROBO-ONTO.

Further, the values for the nouns are extracted from REHABROBO-ONTO to allow suitable entries from the users. These values are listed in Table 7. The values can be considered as ranges of the nouns, that the user can choose from.

4. User-Interface of REHABROBO-QUERY for Query Answering: Intelligent and Interactive

The user interface of REHABROBO-QUERY can guide the users to add/modify information in REHABROBO-ONTO. We have extended it further so that it can guide the users to ask questions in REHABROBO-CNL, and
**Table 7**

Values that can occur after the nouns

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

* (EQCHECK|QUANTIFIER)  ** degree of freedom  *** range of motion

present answers to these queries in an understandable form.

While the user constructs questions, **REHABROBO-QUERY**’s user interface can prevent nonsensical queries. For instance, shoulder elevation/depression is not a wrist movement. The user interface takes into account such information, and guides the user intelligently to construct his/her queries so that queries like

What are the wrist robots that target shoulder elevation/depression?

are not possible. In that sense, the user interface is intelligent.

**REHABROBO-QUERY**’s user interface is also interactive: it shows the possible choices (for instance, the sort of movements targeted by a shoulder robot) and allows auto-completion (for instance, to obtain the full name of a robotics device). To be able to identify the choices (e.g., parameters and their values) relevant to the query, **REHABROBO-QUERY**’s user interface utilizes automated reasoning methods online. Indeed, as the user enters his/her query, in the background the user interface considers the part of the query constructed so far and asks **REHABROBO-QUERY**’s ontological reasoning module to compute all possible values of parameters. Once the reasoning module returns these choices, the user interface presents them, e.g.,
within a pull-down menu. Figure 2 shows the construction of the following query with the guidance of the user interface:

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

Furthermore, 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 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.

Once the query is constructed, REHABROBO-QUERY asks the reasoning module to compute answers, and presents answers to queries concisely, while also including links to detailed information in case the user wants further information. The answer to the query constructed in Figure 2 is shown to the user as in Figure 3.

Note that how the results of a query is displayed to user depends on the sort of 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.

### 5. Transforming a Query in REHABROBO-CNL to a SPARQL Query

The process of answering a query in REHABROBO-CNL, illustrated in Figure 4, starts with a transformation of the query into a SPARQL query. We propose a transformation 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.

#### 5.1. Query Description Trees (QDT)

We introduce 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.

#### Table 6

<table>
<thead>
<tr>
<th>Type()</th>
<th>with</th>
<th>Noun()</th>
</tr>
</thead>
<tbody>
<tr>
<td>robots</td>
<td>→</td>
<td>active degree of freedom</td>
</tr>
<tr>
<td>robots</td>
<td>→</td>
<td>control modes</td>
</tr>
<tr>
<td>robots</td>
<td>→</td>
<td>disorder level</td>
</tr>
<tr>
<td>robots</td>
<td>→</td>
<td>functionality</td>
</tr>
<tr>
<td>robots</td>
<td>→</td>
<td>interaction type</td>
</tr>
<tr>
<td>robots</td>
<td>→</td>
<td>intervention time</td>
</tr>
<tr>
<td>robots</td>
<td>→</td>
<td>kinematic type</td>
</tr>
<tr>
<td>robots</td>
<td>→</td>
<td>motion capability</td>
</tr>
<tr>
<td>robots</td>
<td>→</td>
<td>name</td>
</tr>
<tr>
<td>robots</td>
<td>→</td>
<td>passive degree of freedom</td>
</tr>
<tr>
<td>robots</td>
<td>→</td>
<td>targeted disorder</td>
</tr>
<tr>
<td>robots</td>
<td>→</td>
<td>targeted population</td>
</tr>
<tr>
<td>movements</td>
<td>→</td>
<td>actuation</td>
</tr>
<tr>
<td>movements</td>
<td>→</td>
<td>backdrivability</td>
</tr>
<tr>
<td>movements</td>
<td>→</td>
<td>backdrivability type</td>
</tr>
<tr>
<td>movements</td>
<td>→</td>
<td>maximum range of motion</td>
</tr>
<tr>
<td>movements</td>
<td>→</td>
<td>minimum range of motion</td>
</tr>
<tr>
<td>movements</td>
<td>→</td>
<td>range of motion type</td>
</tr>
<tr>
<td>movements</td>
<td>→</td>
<td>transmission</td>
</tr>
<tr>
<td>publications</td>
<td>→</td>
<td>authors</td>
</tr>
<tr>
<td>publications</td>
<td>→</td>
<td>clinical study</td>
</tr>
<tr>
<td>publications</td>
<td>→</td>
<td>place of publication</td>
</tr>
<tr>
<td>publications</td>
<td>→</td>
<td>title</td>
</tr>
<tr>
<td>publications</td>
<td>→</td>
<td>url</td>
</tr>
<tr>
<td>publications</td>
<td>→</td>
<td>year</td>
</tr>
<tr>
<td>users</td>
<td>→</td>
<td>institution</td>
</tr>
<tr>
<td>users</td>
<td>→</td>
<td>mail</td>
</tr>
<tr>
<td>users</td>
<td>→</td>
<td>username</td>
</tr>
</tbody>
</table>
Figure 5. Tree representation of the sample query.

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 5 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')?

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.

5.2. 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.

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
\[ Q \leftarrow Q \sqcap transform(c); \]
else if $n$ is a that-node then
\[ Q \leftarrow Q \sqcap transformThatNode(n); \]
else if $n$ is a with-node then
\[ Q \leftarrow Q \sqcap transformWithNode(n); \]
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
\quad if $n$ is an and-node then
\quad \quad $tempQ \leftarrow tempQ \sqcap transform(c);$  
\quad else
\quad \quad $tempQ \leftarrow tempQ \sqcup transform(c);$  
\quad $Q \leftarrow Q \sqcap (tempQ);$  
return $Q$

By a depth-first traversal of a QDT, we represent the corresponding concept in Description Logics (DL) as
Algorithm 2: transformThatNode

**Input**: A that-node \( n \)

**Output**: A DL concept description \( Q \) that represents the concept in \( n \)

- \( n.\text{class} \) denote associated class of a node \( n \)
- \( n.\text{verb} \) denote associated verb of a node \( n \)
- \( n.\text{negative} \) denote the negativity of a node \( n \)
- \( n.\text{quantifier} \) denote the quantifier of a node \( n \)
- \( n.\text{instance} \) denote the instance of a node \( n \), if exists
- \( n.\text{child} \) denote child of a node \( n \)
- \( n.\text{identifierNoun} \) denote the noun that identifies \( n.\text{class} \)

\[
\begin{align*}
Q &\leftarrow \emptyset; \\
\text{child}Q &\leftarrow \emptyset; \\
\text{if } n.\text{child} \text{ is not empty } \text{ then} &\quad \text{child}Q \leftarrow \text{transform}(n.\text{child}); \\
\text{else if } n \text{ includes an instance } \text{ then} &\quad \exists(n.\text{class}.\text{identifierNoun}).\{n.\text{instance}\}; \\
\text{if } n.\text{quantifier} = \text{ALL} \text{ then} &\quad \text{if } n.\text{verb} \text{ is passive } \text{ then} &\quad Q \leftarrow Q \cap \forall(n.\text{verb}).((n.\text{class}) \cap \text{child}Q); \\
&\quad \text{else } &\quad Q \leftarrow Q \cap \forall(n.\text{verb}).((n.\text{class}) \cap \text{child}Q); \\
\text{else } &\quad \text{If there is no quantifier or the quantifier is } \text{ SOME} &\quad \text{if } n.\text{verb} \text{ is passive } \text{ then} &\quad Q \leftarrow Q \cap \exists(n.\text{verb}).((n.\text{class}) \cap \text{child}Q); \\
&\quad \text{else } &\quad Q \leftarrow Q \cap \exists(n.\text{verb}).((n.\text{class}) \cap \text{child}Q); \\
\text{if } n.\text{negative} = \text{True} \text{ then} &\quad Q \leftarrow \neg Q; \\
\text{return } Q
\end{align*}
\]

Algorithm 3: transformWithNode

**Input**: A with-node \( n \)

**Output**: A DL concept description \( Q \) that represents the concept in \( n \)

- \( n.\text{noun} \) denote associated noun of a node \( n \)
- \( n.\text{values} \) denote the list of values of a node \( n \)
- \( n.\text{quantifier} \) denote the quantifier of a node \( n \)
- \( n.\text{aggregator} \) denote the aggregator of a node \( n \)
- \( n.\text{datatype} \) denote datatype of the noun in a node \( n \)

\[
\begin{align*}
Q &\leftarrow \emptyset; \\
\text{if } n.\text{noun} \text{ is functional } \text{ then} &\quad \text{if } n.\text{datatype} = \text{boolean} \text{ then} &\quad Q \leftarrow Q \cap \exists(n.\text{noun}).\{n.\text{values}\}^\text{xsd}:\text{boolean}; \\
&\quad \text{else } &\quad \text{if } n.\text{aggregator} = =^{\geq} \text{ or } n.\text{aggregator} = =^{\leq} \text{ then} &\quad Q \leftarrow Q \cap \exists(n.\text{noun}).\{n.\text{values}\}; \\
&\quad \text{else if } n.\text{aggregator} = =^{=} \text{ then} &\quad Q \leftarrow Q \cap \exists(n.\text{noun}).\{n.\text{values}\}; \\
&\quad \text{else if } n.\text{aggregator} = =^{!} =^{=} \text{ then} &\quad \text{foreach } v \in n.\text{values} \text{ do} &\quad Q \leftarrow Q \cap \neg \exists(n.\text{noun}).\{v\}; \\
&\quad \text{else } &\quad \text{if } n.\text{quantifier} = \text{NO} \text{ then} &\quad Q \leftarrow Q \cap \exists(n.\text{noun}).\text{xsd}:\text{datatype}; \\
&\quad \text{else if } n.\text{quantifier} = \text{ALL} \text{ then} &\quad Q \leftarrow Q \cap \forall(n.\text{noun}).\text{xsd}:\text{datatype}; \\
&\quad \text{else if } n.\text{quantifier} = \text{SOME} \text{ then} &\quad Q \leftarrow Q \cap \exists(n.\text{noun}).\text{xsd}:\text{datatype}; \\
&\quad \text{else } &\quad \text{if } n.\text{aggregator} = =^{=} \text{ then} &\quad \text{foreach } v \in n.\text{values} \text{ do} &\quad Q \leftarrow Q \cap \exists(n.\text{noun}).\{v\}; \\
&\quad \text{else if } n.\text{aggregator} = =^{!} =^{=} \text{ then} &\quad \text{foreach } v \in n.\text{values} \text{ do} &\quad Q \leftarrow Q \cap \neg \exists(n.\text{noun}).\{v\}; \\
\text{return } Q
\end{align*}
\]

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 5. 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 \textit{Robot}. Then, the algorithm calls \textit{transform} recursively for each child of the root node. In the first recursive call, since the current node is a \textit{that}-node, the algorithm calls \textit{transformThatNode} (Algorithm 2) and
passes \textit{that-node} as an input. Since it has a child node, the first condition of having children is satisfied. Next, the algorithm calls \textit{transform} for its child node, \textit{and-node}, whose children are conjoined after each transformation. The first child of \textit{and-node} is transformed into an existential restriction by calling \textit{transformWithNode} (Algorithm 3). The second child of \textit{and-node} contains a simple disjunction. The algorithm transforms the information in the children of \textit{or-node} into existential restrictions and disjoins them. Finally, the algorithm finishes recursions and returns to \textit{that-node}, covers the transformation of its children with brackets, and transforms \textit{that-node} into an existential restriction. The resulting DL concept that is returned from the algorithm is as follows.

\[
\text{Robot} \land \exists \text{targets.}(\text{ShoulderMovements}\{\text{electrical}\}) \land \\
(\exists \text{transmission.}\text{cabledrive}) \lor \\
(\exists \text{transmission.}\text{directdrive}))
\]

5.3. 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 [27] and [14]. We also introduce some novel transformations that are not covered by other work. Some transformation examples are shown in Table 8. The transformations without a citation are novel.

Novel transformations include the transformation of inverse roles. Consider the inverse role example in Table 8. DL representation of this concept contains a negated existential quantifier. SPARQL transformation contains a triple covered with \texttt{FILTER NOT EXISTS}. The triple searches for a mapping of variable \(x\) to a node, that is related to another node \texttt{AssistOn} with an edge that characterizes \texttt{has\_Name} relation. This corresponds to an existential restriction. However, we do not want such mappings of \(x\). Therefore, according to the semantics of \texttt{NOT EXISTS} in a filter expression, \texttt{FILTER} \texttt{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 \texttt{AssistOn}. Therefore, evaluation of the DL concept and the SPARQL concept will return the same result.

Finally, consider the universal restriction example in Table 8. 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 \texttt{FILTER NOT EXISTS}. Since a universal restriction such as \(\forall x A(x)\) corresponds to a negated existential restriction \(\neg \exists x A(x)\), each \texttt{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 \texttt{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 \texttt{FILTER NOT EXISTS} expression contains another \texttt{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 \(y2\) is a robot. Since both expressions contain the same variable, the mappings should agree on the robot \(y2\). In addition, the set of mappings for \(y2\) must be an empty set to satisfy the inner \texttt{FILTER NOT EXISTS} expression. The outer \texttt{FILTER NOT EXISTS} expression then contains the in-
Table 8
DL to SPARQL Transformation Examples

<table>
<thead>
<tr>
<th>Constructor</th>
<th>DL</th>
<th>SPARQL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Role [27]</td>
<td>targets</td>
<td>?x ns:targets ?y.</td>
</tr>
<tr>
<td>Complement</td>
<td>¬∃name.{AssistOn}</td>
<td>FILTER NOT EXISTS {</td>
</tr>
<tr>
<td></td>
<td></td>
<td>?x ns:has_Name ‘AssistOn’ .</td>
</tr>
<tr>
<td></td>
<td></td>
<td>}</td>
</tr>
<tr>
<td>Inverse Role</td>
<td>∃targets~.Robot</td>
<td>?x ns:targets ?y.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>?x rdf:type ns:RehabRobots.</td>
</tr>
<tr>
<td>hasValue Restriction [27]</td>
<td>∃name.{AssistOn}</td>
<td>?x ns:has_Name ‘AssistOn’ .</td>
</tr>
<tr>
<td></td>
<td></td>
<td>?y rdf:type rr:RehabRobots.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FILTER NOT EXISTS {</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FILTER NOT EXISTS {</td>
</tr>
<tr>
<td></td>
<td></td>
<td>?x rr:reference ?y2.)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>?y2 rdf:type rr:RehabRobots.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>?x ns:has_Functionality ‘clinic’.</td>
</tr>
<tr>
<td>Union [14]</td>
<td>∃functionality.{clinic}/</td>
<td>(?x ns:has_Functionality ‘clinic’.)</td>
</tr>
<tr>
<td></td>
<td>∃motionCapability.{grounded}</td>
<td>UNION</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(?x ns:has_Motion_Capability ‘grounded’.)</td>
</tr>
</tbody>
</table>

stances 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.

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

∃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.


?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.

\[ \text{?movement1 rr:has_Actuation 'electrical'.} \]

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.

\[
\begin{align*}
\text{?movement1 rr:has_Transmission 'cable drive'.} & \text{ UNION } \\
\text{?movement1 rr:has_Transmission 'direct drive'.}
\end{align*}
\]

The resulting SPARQL concept:

\[
\begin{align*}
\text{?robot1 rdf:type rr:RehabRobots.} & \\
\text{?robot1 rr:targets ?movement1.} & \\
\text{?movement1 rdf:type rr:ShoulderMovements.} & \\
\text{?movement1 rr:has_Actuation 'electrical'.} & \text{ UNION } \\
\text{?movement1 rr:has_Transmission 'cable drive'.} & \\
\text{?movement1 rr:has_Transmission 'direct drive'.}
\end{align*}
\]

5.4. Some more examples

For a better understanding of the sequence of transformations above, that takes a REHABROBO-CNL and turns it into a SPARQL query, two more examples are provided in Figures 6 and 7.

6. Answering Queries Using Pellet

We use the DL reasoner PELLET to find answers to queries, through the Jena framework. Consider, for instance, the query

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

whose SPARQL representation is as follows.

\[
\begin{align*}
\text{SELECT DISTINCT ?name} & \text{ WHERE } \\
\text{?robot1 rr:has_Name ?name.} & \\
\text{?robot1 rdf:type rr:RehabRobots.} & \\
\text{?robot1 rr:targets ?movement1.} & \\
\text{?movement1 rdf:type rr:WristMovements.} & \\
\text{?movement1 rr:has_Actuation 'series elastic'.}
\end{align*}
\]

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

\[
\text{( ?name = "AssistOn-Mobile" )}
\]

Consider, for instance, another query:

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.

\[
\begin{align*}
\text{SELECT DISTINCT ?name} & \text{ WHERE } \\
\text{?publication1 rr:has_Title ?name.} & \\
\text{?publication1 rdf:type rr:References.} & \\
\text{?publication1 rr:has_PublishedAt 'ICORR'.} & \\
\text{?robot1 rr:hasReference ?publication1.} & \\
\text{?robot1 rdf:type rr:RehabRobots.} & \\
\text{?robot1 rr:ownedBy ?user1.} & \\
\text{?user1 rdf:type rr:Owners.} & \\
\text{?user1 rr:has_Institution 'Sabanci University'.}
\end{align*}
\]
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:

```
root-node
  |  or-node
  |     with-node
  |       "with no targeted disorder"
  |     and-node
  |     with-node
  |       "with no disorder level"
  |     with-node
  |       "with intervention time != 'chronic'"
  |     with-node
  |       "with motion capability = 'grounded'"
```

Q11 in Description Logics (DL):

```
Robot ⊓ (∃¬targetedDisorder.(xsd:string))⊔ (∃¬interventionTime.{'chronic'} ∩ ∃motionCapability.{'grounded'})⊔ ¬∃disorderLevel.(xsd:string))
```

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.
})
```

Figure 6. Transformation of a query, Q11, from REHAROBO-CN1 to a SPARQL.
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:

```
root-node "What are the robots"
and-node
  with-node "with interaction type = 'exoskeleton'"
  that-node "that target some finger movements"
or-node
  with-node "with actuation = 'electrical'"
  with-node "with actuation = 'hydrolic'"
  with-node "with actuation = 'series elastic'"
```

Q12 in Description Logics (DL):

\[
\text{Robot} \sqcap \exists \text{interactionType.} \{\text{exoskeleton}\} \sqcap \\
\exists \text{target.} \{\text{FingerMovement} \sqcap \exists \text{actuation.} \{\text{electrical}\} \sqcup \\
\exists \text{actuation.} \{\text{hydrolic}\} \sqcup \\
\exists \text{actuation.} \{\text{series elastic}\}\})
\]

Q12 in SPARQL:

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

Figure 7. Transformation of a query, Q11, from REHABROBO-CNL to a SPARQL.
We get the following answers from Pellet to this query:

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

7. Related Work

The most related work involves ontology systems and 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 [2,3,4,16,20,21,23,25,30,32,33] 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 [12], which is a software system that answers natural language queries over biomedical databases and ontologies. It utilizes Answer Set Programming (ASP) [24] 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 [17] and CLASP-NK, and it also provides explanations to the queries.

Ferrández et al. [15] 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 [6] is developed by the creators of and as a development upon QUESTIO [30]. In order to support natural language queries, it uses Stanford Parser [7] to generate a parse tree. Then, using GATE libraries [5], 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. [26] introduce PowerAqua, which is evolved from AquaLog [25]. It provides natural language querying over multiple ontologies; thus, supports high scalability and portability. It uses GATE libraries and WordNet [13] 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. [31] 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 Pellet. 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.

8. Conclusion

In this article, we have introduced methods for representing queries about rehabilitation robotics in a con-
controlled natural language, transforming them into formal queries, and computing answers to them using automated reasoners over the first formal rehabilitation robotics ontology REHABROBO-ONTO. We have also introduced an interactive intelligent user-interface as part of the software system REHABROBO-QUERY, that guides the users during this whole process in such a way that the users do not have to know about the underlying logical formalism of the ontology or the formalism to represent queries:, and they do not have to know about the use of the technologies for computing answers to their questions.

By means of answering sophisticated 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. Also, 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.

Having a structured formal representation of knowledge about rehabilitation robotics as an ontology, allows answering complex queries that requires integration with other knowledge resources (e.g., patient databases, disease ontologies). Along this research direction, 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 of REHABROBO-CNLI, the algorithms and the user interface of REHABROBO-QUERY are needed to be able to answer complex queries about therapies, diseases and anatomy. These studies concerning interoperability of REHABROBO-ONTO is part of our ongoing work.

Acknowledgment

We would like to thank Muhammed Kilinc and Sibel A. Yildirim (Hacettepe University, School of Physical Therapy and Rehabilitation) for their feedbacks on the sorts of queries from the perspectives of physical medicine, and Agis Papantoniou (Cognizone) for his help with the design of REHABROBO-QUERY. We would like to thank Ahmetcan Erdogan (Rehabilitation Institute of Chicago), Mustafa Yalcin, Murat Isik and Ata Otaran (Human-Machine Interaction Lab, Sabanci University), and Ezgi Demirel (Knowledge Representation and Reasoning Group, Sabanci University) for their helps with testing REHABROBO-QUERY. We also would like to thank members of European Network on Robotics for NeuroRehabilitation for useful discussions.

References


