UNIVERSITÀ DEGLI STUDI DI ROMA TOR VERGATA

FACOLTÀ DI INGEGNERIA

Artificial Intelligence

A.A. 2015/2016

Exam Report

Todo

PROFESSORE

Prof. Roberto Basili

STUDENTE

Giovanni Balestrieri

CORRELATORE

Dott. Danilo Croce

Contents

Abstract

This paper is an academic final report for the Artificial Intelligence course. The goal of the project is to implement a Ros module for building high-level representations of the environment that embody both metric and symbolic knowledge about it. A key issue in the interaction with robots is to establish a proper relationship between the symbols used in the representation and the corresponding elements of the operational environment.

1 Introduction

Robotics is in an exciting stage and human machine interaction is being intensively studied. Robots are expected to get closely involved into human life as they are being marketed for commercial applications such as telepresence, service or entertainment. However, although they are expected to become consumer products, there is still a gap in terms of user expectations and robot functionalities. A key limiting factor is the lack of awareness of the robot on the operational environment. This project investigates several strategies to integrate a knowledge base in the ROS framework. The implemented system provides knowledge processing capabilities that combine knowledge representation and reasoning methods to manipulate and interact with physical objects of the operational milieu through an *API system* and a graphical interface.

ADD ROADMAP

1.1 Objectives

1. Integrate a knowledge base in ROS,
2. Create a node to Parse owl files in ROS environment,
3. Consistency check of ontology,
4. Reasoner invocation,

5. Make the ROS environment aware of object's instances,
6. Graphical representation of the objects in the scene,
7. Insert new instance of object in the scene and check for consistency,
8. Delete instance and check for consistency,
9. List instances,
10. Graphical Scene configuration from Gazebo and automatic Abox update
11. Export current Abox in owl or different formats.

1.2 Project Schedule

A public git repository is available at the following github page

Week	General Task	Documentation	Implementation
1	• Perform a literature review on the previous HuRIC publications	• Knowledge representation and Reasoning [11] • Huric papers [1],[2],[3],[4], [5]	
2	• Literature review of ROS compatible triple store • ROS compatible simulation environments	• ROS Documentation [6], [7] • ROS compatible simulation environments [8], [9]	• Set up ROS environment. • Brainstorm Software Architecture
3	• Literature review of RDFlib based papers • Test of Gazebo Simulator	• Full Training session on Gazebo Simulator [10] • Python library RDFLib test	• Test Json Parser node • Populate a scene from json files.
4	• Kinect pointcloud literature review • Object recognition papers review	• PointCloudLibrary documentation [16] • Python SciPy library classifier documentation	• Parse test KnowledgeBase owl file with RDFLib • Clear scene script in Gazebo.
5	• Integrate classification algorithms • Optimize python code and ROS environment.	• ROS + PCL integration • RDF library deep inspection	• Analyze pointcloud from kinect • Scene analysis, plane segmentation • Object recognition, vote classifier
6	• Owl Reasoner integration • Test RDFlib and Apache Jena	• RDF lib documentation • Apache Jena Fuseki Documentation [15]	• Create init node • Spawn models script • Remove models script • Apache Jena basic setup

Week	General Task	Documentation	Implementation
7	• Consistency check • RosJava integration • Jena Ontology Api	• RosJava guidelines [12] • Apache Jena Ontology Api [13]	• Integrate RosJava in Ros. • Follow Jena Ontology Api tutorial. Basic rdf manipulation
8	• Consistency check	• RosJava guidelines [12] • Jena Reasoner Documentation [14]	• Implement consistency check with Jena. • Integrate Jena libraries in ROS • Reasoner invocation in ROS.
9	• Owl A-box extraction • Specialize Reasoner on Tbox	• RDFlib export documentation • Jena Reasoner documentation	• Implement A-box generator Jena • Retrieve information about instances
10	• Get info about model in Gazebo • Specialize Reasoner on Tbox	• Programming Robots with Ros [9] • Gazebo Documentation [10]	• Subscribe to gazebo model-States in ROSJava • Call spawn model service from ROSJava
11	• SPARQL queries Add, Delete, Update, GET Instance • Coordinator Node Definition • Interface between Semantic Map and Gazebo Node	• Jena Ontology Api Documentation [13] • Gazebo Documentation [10]	• Jena RDF graph manipulation • Jena SPARQL Delete and Add queries • Jena SPARQL GetInstance queries

Week	General Task	Documentation	Implementation
12	• Test Case validation • Use Case validation • Application Demo		• Testing of real world scenario • Bug fixing • Extern Node definition

2 The Ontology

It is expected that mobile robots undertake various tasks not only in the industrial fields such as manufacturing plants and construction sites, but also in the environment we live in.

2.1 Tbox

In this project a generic home domain model has been taken into account.

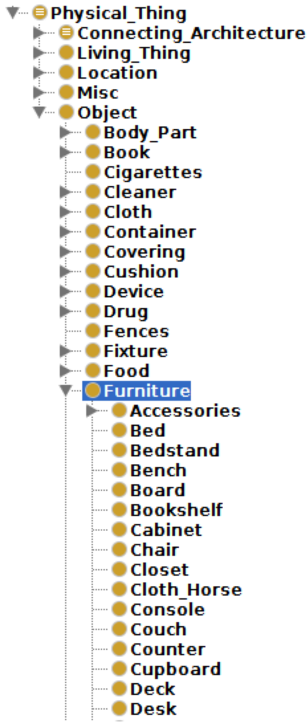

Figure 1: Physical Things

The furniture class describes several objects of a generic home environment a robot can interact with.

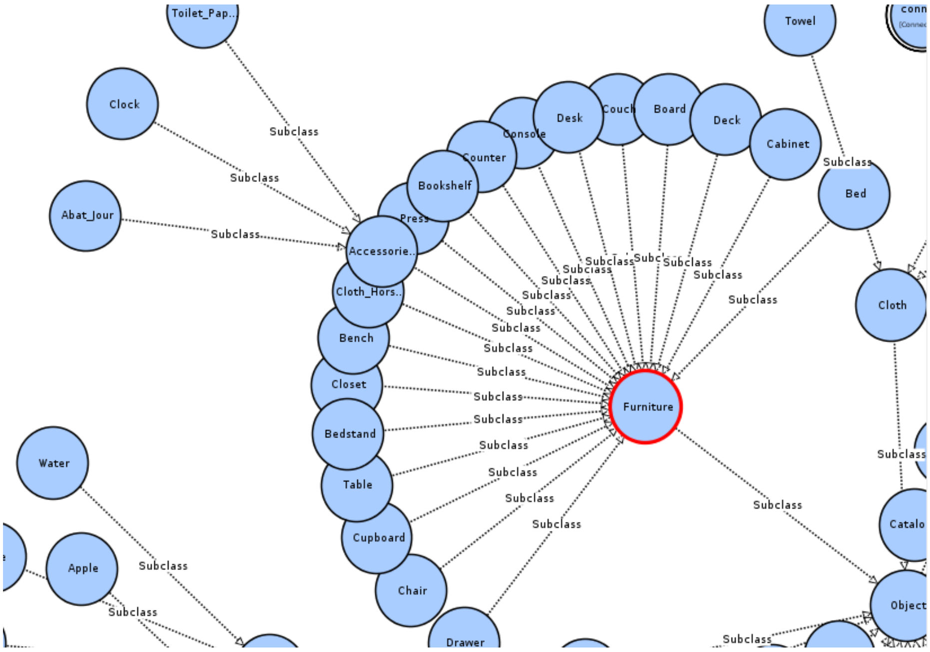

Figure 2: furniture subclasses

Properties

OWL distinguishes between two main categories of properties that an ontology builder may want to define:

- Object properties link individuals to individuals.
- Datatype properties link individuals to data values.

An object property is defined as an instance of the built-in OWL class owl:ObjectProperty. A datatype property is defined as an instance of the built-in OWL class owl:DatatypeProperty. Both owl:ObjectProperty and owl:DatatypeProperty are subclasses of the RDF class rdf:Property.

Properties

The datatypes involved are shown in the Figure 2.2 and 2.1

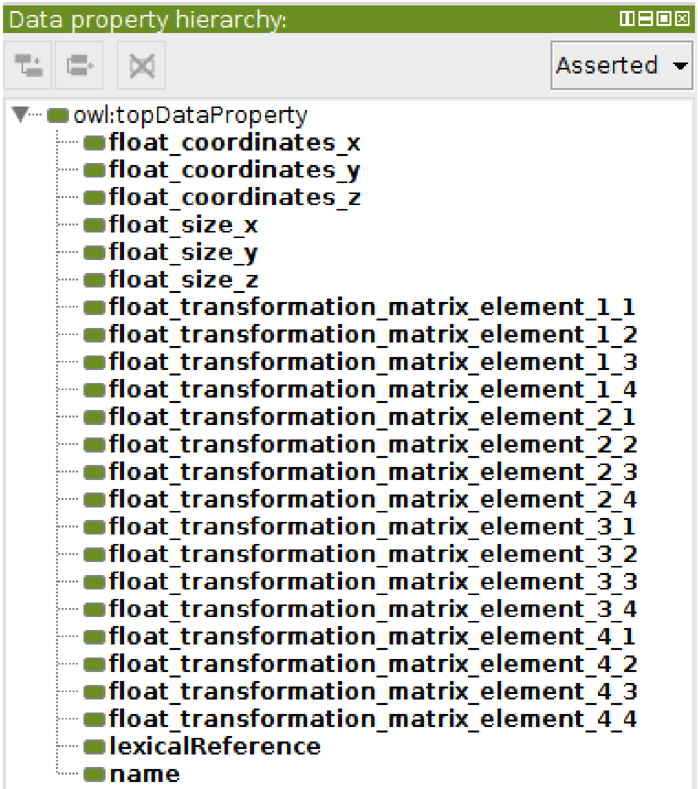

Figure 3: Datatypes Visual Protégé

Figure 4: Datatypes

Object Properties

The following Figure 2.1 shows the class tree diagram of the ObjectProperties involved in the project.

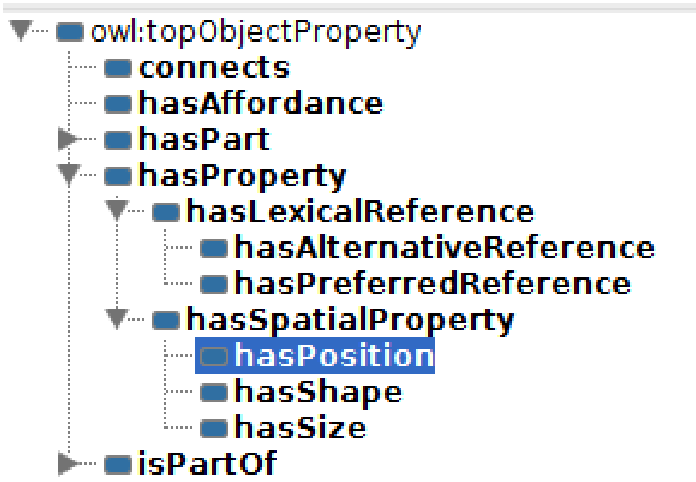

Figure 5: Object Properties Class Tree

As an example, consider the following set of owl statements about the ObjectProperty hasPosition. This property is of the type IrreflexivePropery and is a subProperty of SpacialProperty.

```
<owl:ObjectProperty rdf:about="sm#hasPosition">
    <rdfs:subPropertyOf rdf:resource="sm#hasSpatialProperty"/>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#IrreflexiveProperty"/>
</owl:ObjectProperty>
```

Let us consider another Property involved in this project.

```
<owl:Class rdf:about="sm#Coordinates">
    <rdfs:subClassOf rdf:resource="sm#Position"/>
    <rdfs:subClassOf>
       <owl:Restriction>
           <owl:onProperty rdf:resource="sm#float_coordinates_z"/>
           <owl:someValuesFrom ...
               rdf:resource="http://www.w3.org/2001/XMLSchema#float"/>
           </owl:Restriction>
     </rdfs:subClassOf>
     <rdfs:subClassOf>
        <owl:Restriction>
           <owl:onProperty rdf:resource="sm#float_coordinates_x"/>
           <owl:qualifiedCardinality ...
               rdf:datatype="http://www.w3.org/2001/XMLSchema#nonNegativeInteger">1
           </owl:qualifiedCardinality>
           <owl:onDataRange ...
               rdf:resource="http://www.w3.org/2001/XMLSchema#float"/>
           </owl:Restriction>
     </rdfs:subClassOf>
     <rdfs:subClassOf>
        <owl:Restriction>
           <owl:onProperty rdf:resource="sm#float_coordinates_y"/>
```

```
            <owl:qualifiedCardinality ...
                rdf:datatype="http://www.w3.org/2001/XMLSchema#nonNegativeInteger">1
             </owl:qualifiedCardinality>
            <owl:onDataRange ...
                rdf:resource="http://www.w3.org/2001/XMLSchema#float"/>
        </owl:Restriction>
    </rdfs:subClassOf>
</owl:Class>
```

The class Coordinates is a subclass of the class Position and of three anonymous classes. A property restriction describes an anonymous class, namely a class of all individuals that satisfy the restriction.
The first restriction is a Value contraint linked (using owl:onProperty) the Property float_coordinates_z to a class of all individuals for which at least one value of the property concerned is an instance of a data value in the data range.
The second and third restrictions are Cardinality contraints linked to the Property float_coordinates_x and float_coordinates_y.

2.2 Abox

The collection of individual are stored in a separate file called semantic_mapping, the Abox. The demo supports operations on four classes of instances since the 3D environment requires tridimentional models of the object to be represented. This constrain could be relaxed by adding an exhaustive collection of 3D models and by associating them to the corresponding classes.

An instance of the class Chair

Individuals are defined with individual axioms called “facts”. These facts are statements indicating class membership of individuals and property values of individuals. As an example, consider the following set of statements about an instance of the class Chair:

```
<NamedIndividual rdf:about="sm#chair1">
        <rdf:type rdf:resource="&semantic_mapping_domain_model;Chair"/>
        <semantic_mapping_domain_model:hasPosition ...
            rdf:resource="sm#chair1_coordinates"/>
        <semantic_mapping_domain_model:hasSize rdf:resource="sm#chair1_size"/>
        <semantic_mapping_domain_model:hasAlternativeReference ...
            rdf:resource="sm#chair_alternative_reference_1"/>
        <semantic_mapping_domain_model:hasAlternativeReference   ...
            rdf:resource="sm#chair_alternative_reference_2"/>
        <semantic_mapping_domain_model:hasAlternativeReference  ...
            rdf:resource="sm#chair_alternative_reference_3"/>
        <semantic_mapping_domain_model:hasPreferredReference ...
            rdf:resource="sm#chair_preferred_reference"/>
    </NamedIndividual>
```

This example includes a number of facts about the individual chair1, an instance of the class Chair. The chair has three alternative references and one preferred lexical reference. These properties link a chair to a typed literal with the XML Schema datatype date. The XML schema document on datatypes contains the relevant information about syntax and semantics of this datatype. The property hasPosition and hasSize link the chair to instances of the type Coordinates and Dimensions.

The following figure shows the same information on Protégé:

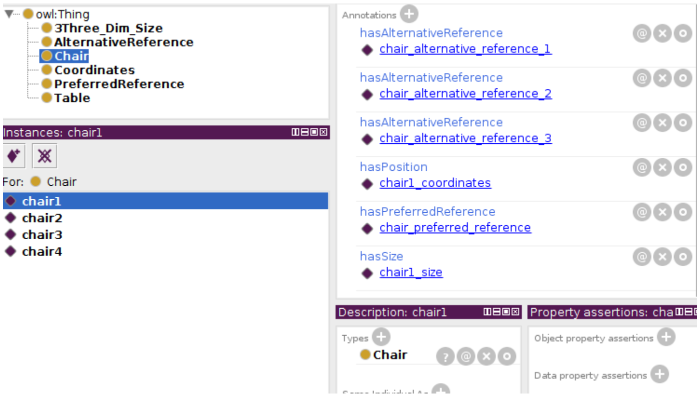

Figure 6: Chair1

Properties

The following example shows AlternativeReference instance property.

```
<NamedIndividual rdf:about="sm#chair_alternative_reference_1">
        <rdf:type ...
            rdf:resource="&semantic_mapping_domain_model;AlternativeReference"/>
        <semantic_mapping_domain_model:lexicalReference ...
            rdf:datatype="&xsd;string">chair
        </semantic_mapping_domain_model:lexicalReference>
    </NamedIndividual>
```

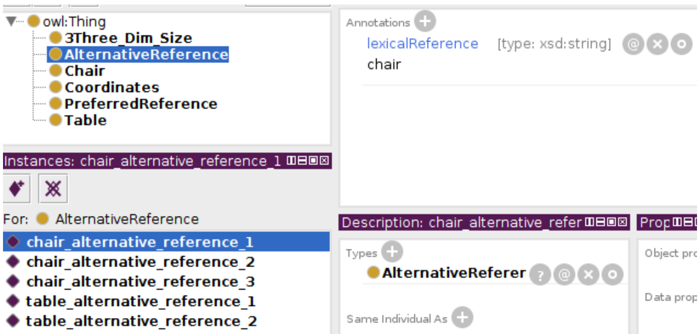

Figure 7: Alternative Reference 1

The following example shows the instance of the 3Three_Dim_Size property associated to the individual chair1

```
<NamedIndividual rdf:about="sm#chair1_coordinates">
        <rdf:type rdf:resource="&semantic_mapping_domain_model;Coordinates"/>
```

```
        <semantic_mapping_domain_model:float_coordinates_z ...
            rdf:datatype="&xsd;float">0.0
        </semantic_mapping_domain_model:float_coordinates_z>
        <semantic_mapping_domain_model:float_coordinates_y ...
            rdf:datatype="&xsd;float">0.0
        </semantic_mapping_domain_model:float_coordinates_y>
        <semantic_mapping_domain_model:float_coordinates_x ...
            rdf:datatype="&xsd;float">1.0
        </semantic_mapping_domain_model:float_coordinates_x>
    </NamedIndividual>
```

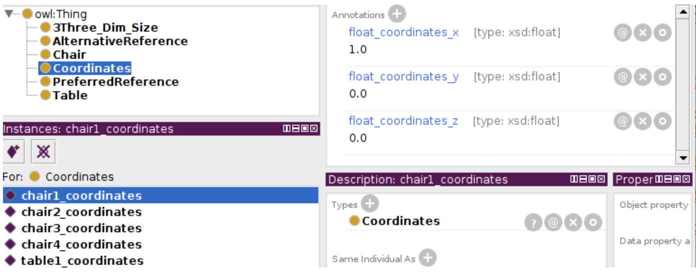

Figure 8: Coordinates chair 1

3 Involved Technologies

3.1 JENA Ontology API

The Jena ontology API is a Java programming toolkit. Jena's ontology support is limited to ontology formalisms built on top of RDF.

RDFS is the weakest ontology language supported by Jena. With RDFS it is possible to build a simple hierarchy of concepts, and a hierarchy of properties. There are various different ontology languages available for representing ontology information on the semantic web. They range from the most expressive, OWL Full, through to the weakest, RDFS.

The ontology language used in this project is the OWL FULL. OWL language allows properties to be denoted as transitive, symmetric or functional, and allows one property to be declared to be the inverse of another.

One of the key benefits of building an ontology-based application is using a reasoner to derive additional truths about the concepts you are modelling. Jena includes support for a variety of reasoners through the inference API.

A common feature of Jena reasoners is that they create a new RDF model which appears to contain the triples that are derived from reasoning as well as the triples that were asserted in the base model. The ontology API can query an extended inference model and extract information not explicitly given.

3.2 ROS Famework

The Robot Operating System (ROS) is a framework for writing robot software. It is a collection of tools, libraries and conventions that aims to simplify the task of creating complex and robust robot behavior across a wide variety of robotics platforms. The software is structured as a large number of modules that pass data to one another using a inter-process communication.

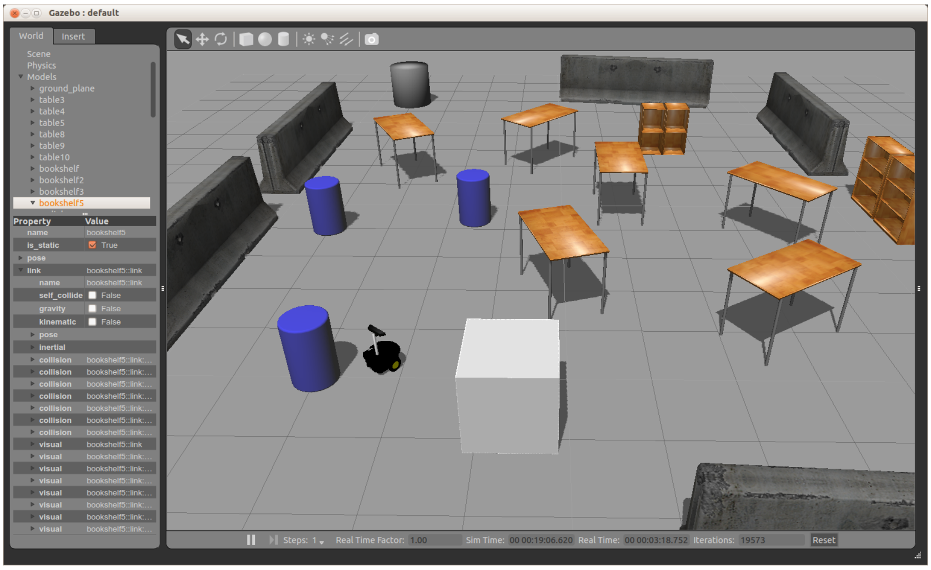

Figure 9: ROS simulation

ROS was built from the ground up to encourage collaborative robotics software development. A ROS system consists of numerous small computer programs that connect to one another and continuously exchange messages. There is no central routing service. It is a tools-based program. Tasks such as visualizing the system interconnections generating documentation, logging data, filtering sensor's data, etc. are all performed by separate programs. The individual tools themselves are relatively small and generic.
ROS chose a multilingual approach that allows programmers to accomplish tasks using scripting languages such as Python and MATLAB or using faster ones like C++. Client libraries exist for LISP, Java, JavaScript, Ruby,R and others. ROS libraries communicate with one another by following a convention that describes how messages are serialized before being transmitted over the network. The ROS conventions encourages contributors to create standalone libraries in order to allow the reuse of software and speed up development.

The core of ROS is released under the BSD license which allows commercial and noncommercial use.

3.3 Simulation Environment with Gazebo

Robotics implies robots. Most part of these platforms are used for research purposes and are custom built to investigate a particular aspect. However, there are a growing number of standard products that can be purchased and used out of the box for development and operations in many domains of robotics.

Although several robotics platforms are considered to be low cost they are still significant investments. Even the best robots can break periodically due to various combinations of operator error, environmental conditions, manufacturing and design defects.

All of these drawbacks can be avoided by using simulated robotic structures and a simulation environment. Gazebo is a $3D$ dynamic simulator with the ability to accurately and efficiently simulate populations of robots in complex indoor and outdoor environments. It offers physics engine with high degree of fidelity and a variety of sensors.

Many robots are provided including $PR2$, $Pioneer2$ DX, iRobot Create, and TurtleBot. Thanks to URDF format, robotics platforms can be created from scratch and deployed into the simulator. With this environment it is possible to run simulation on remote servers, and interface to Gazebo through socket-based message.

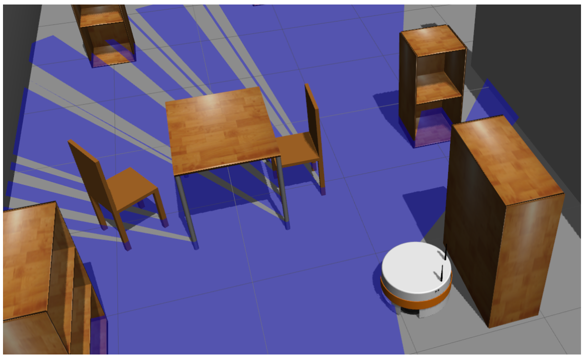

Figure 10: Gazebo Environment

Ros integrates closely with Gazebo through the *gazebo_ros* package. The latter provides a Gazebo plugin module that allows bidirectional communication between ROS and the simulator. Sensors, physics data, video input can stream from Gazebo to ROS and actuators commands can be forwarded to the simulation environment. By choosing consistent names and data types for these data streams it is possible to run the low level device-driver software on both the real robot and in the simulator.

4 The Architecture

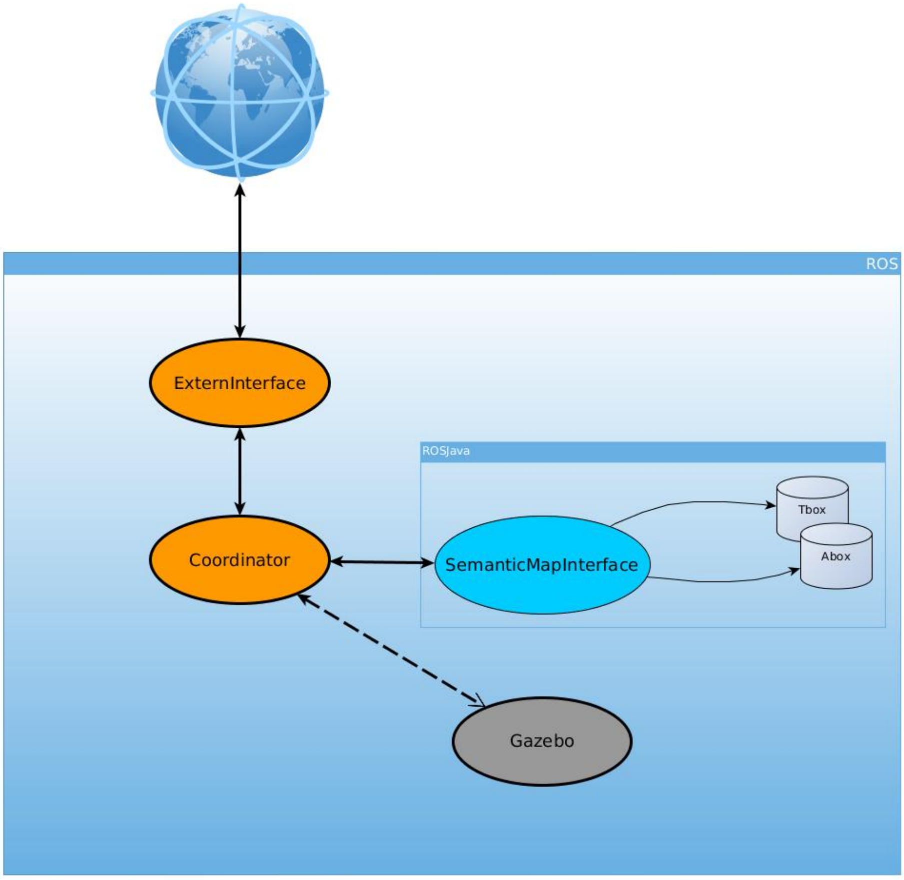

Figure 11: System Architecture

5 The Application

5.1 Module Flow

The Jena ontology API is a Java programming toolkit. Jena's ontology support is limited to ontology formalisms built on top of RDF.

RDFS is the weakest ontology language supported by Jena. With RDFS it is possible to build a simple hierarchy of concepts, and a hierarchy of properties. There are various different ontology languages available for representing ontology information on the semantic web. They range from the most expressive, OWL Full, through to the weakest, RDFS.

The ontology language used in this project is the OWL FULL. OWL language allows properties to be denoted as transitive, symmetric or functional, and allows one property to be declared to be the inverse of another.

One of the key benefits of building an ontology-based application is using a reasoner to derive additional truths about the concepts you are modelling. Jena includes support for a variety of reasoners through the inference API.

A common feature of Jena reasoners is that they create a new RDF model which appears to contain the triples that are derived from reasoning as well as the triples that were asserted in the base model. The ontology API can query an extended inference model and extract information not explicitly given.

5.2 Topics and Services

5.3 Exporting Ontology

5.4 Future works

The following points assume that the robot is equipped with a spoken command recognition

Lu4r

PointCloud

List of Figures

References

[1] E. Bastianelli, D. D. Bloisi, R. Capobianco, F. Cossu, G. Gemignani, L. Iocchi, and D. Nardi, *"On-line semantic mapping"* in 16th International Conference on Advanced Robotics, Nov 2013.

[2] Emanuele Bastianelli, Danilo Croce, Roberto Basili, Daniele Nardi, *"Using Semantic Maps for Robust Natural Language Interaction with Robots"*, DICII, 2 DII - University of Rome Tor Vergata, DIAG - Sapienza University of Rome - Rome, Italy.

[3] Emanuele Bastianelli, Giuseppe Castellucci, Danilo Croce, Luca Iocchi, Roberto Basili, Daniele Nardi, *"HuRIC: a Human Robot Interaction Corpus"*

[4] E. Bastianelli, Giuseppe Castellucci, Danilo Croce, Roberto Basili, Daniele Nardi, *"Effective and Robust Natural Language Understanding for Human -R obot Interaction"*, ECAI, 2014.

[5] E. Bastianelli, D. Bloisi, R. Capobianco, G. Gemignani, L. Iocchi, D. Nardi, *"Knowledge Representation for Robots through Human-Robot Interaction"*, Rome.

[6] Aaron Martinez, Enrique Fernández, *"Learning ROS for Robotics Programming"* A practical, instructive, and comprehensive guide to introduce yourself to ROS, the top-notch, leading robotics framework, 2013.

[7] Lentin Joseph, *"Learning Robotics Using Python"* Design, simulate, program, and prototype an interactive autonomous mobile robot from scratch with the help of Python, ROS, and Open-CV!, 2015.

[8] Lentin Joseph, *"Mastering ROS for Robotics Programming"*, 2015 Packt Publishing.

[9] Morgan Quigley, Brian Gerkey and William D. Smart,
"Programming Robots with ROS", 2016 O Reilly Media.

[10] Open Source Robotics Foundation,
"GazeboSim documentation", http://gazebosim.org/tutorials, 2016.

[11] R. Brachman, H. Levesque,
“Knowledge Representation and Reasoning”.

[12] Open Source Robotics Foundation, Ros Documentation,
“$http://wiki.ros.org/rosjava_{b}uild_{t}ools/Tutorials/hydro$”.

[13] Jena Ontology Api,
“https://jena.apache.org/documentation/ontology/”.

[14] Jena Reasoner and Inference documentation,
“http://jena.apache.org/documentation/inference/index.html”.

[15] Apache Jena Fuseki standalone server documentation,
“$https://jena.apache.org/documentation/serving_{d}ata/$”.

[16] Point Cloud Library Documentation,
“http://pointclouds.org/documentation/tutorials/index.php”.