**UNIVERSITÀ DEGLI STUDI DI ROMA
TOR VERGATA**

FACOLTÀ DI INGEGNERIA

Artificial Intelligence

A.A. 2015/2016

Exam Report

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.

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. Ontology manipulation from a 3D environment interface 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[1] and a step by step guide has been published at www.userk.co.uk[2].

[1]https://github.com/GiovanniBalestrieri/JinchiMiru

[2]More further information visit http://userk.co.uk/handling-ontologies-with-ros/

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 modelStates 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

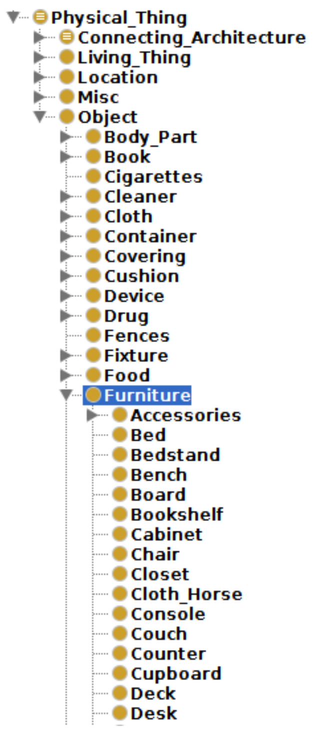
Week	General Task	Documentation	Implementation
12	• Test Case validation • Use Case validation • Application Demo		• Testing of real world scenario • Bug fixing • Extern Node definition

2 Ontological model of domestic environment

It is expected that mobile robots undertake various tasks not only in 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.



- Physical_Thing
  - Connecting_Architecture
  - Living_Thing
  - Location
  - Misc
  - Object
    - Body_Part
    - Book
    - Cigarettes
    - Cleaner
    - Cloth
    - Container
    - Covering
    - Cushion
    - Device
    - Drug
    - Fences
    - Fixture
    - Food
    - Furniture
      - Accessories
      - Bed
      - Bedstand
      - Bench
      - Board
      - Bookshelf
      - Cabinet
      - Chair
      - Closet
      - Cloth_Horse
      - Console
      - Couch
      - Counter
      - Cupboard
      - Deck
      - Desk

Figure 1: Taxonomy of classes in Protégé

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

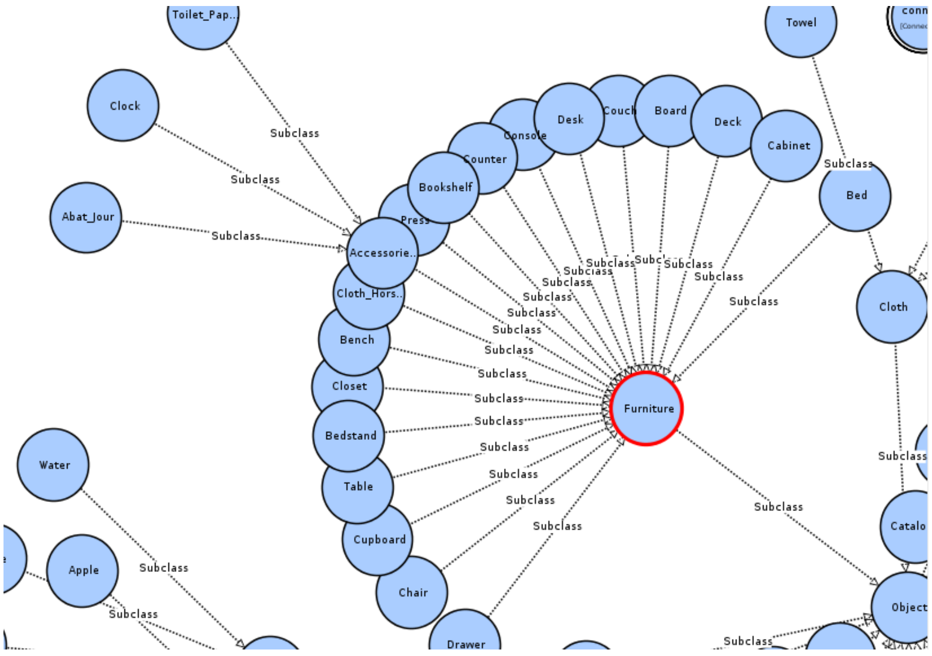


Figure 2: Furniture subclasses graph

As an example, consider the Coordinates class below.

```
<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">
           </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">
            </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 that links (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.

Properties

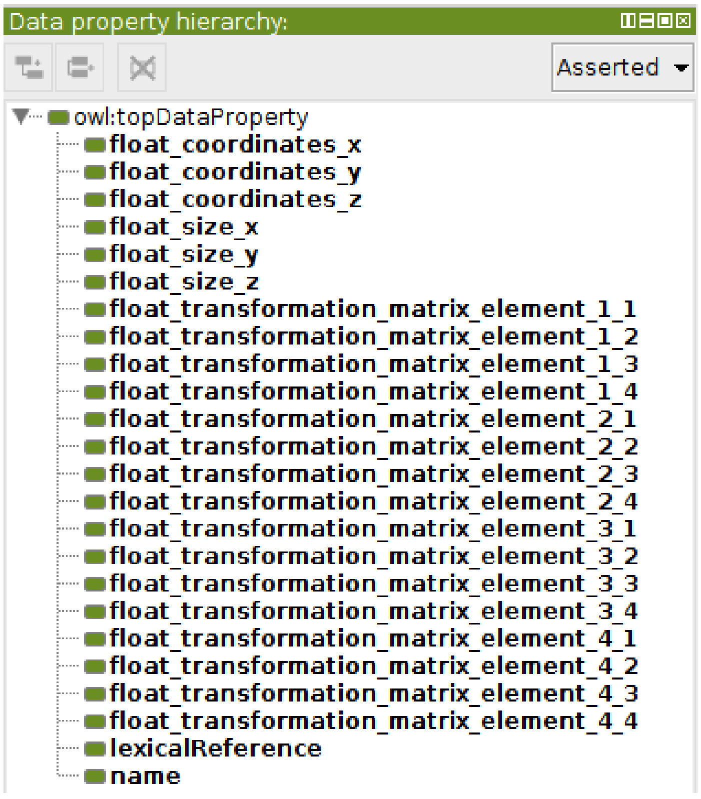
OWL distinguishes between two main categories of properties that an ontology builder may want to define:

- 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.

Datatype Properties

The datatypes involved are shown in the Figure 2.2 and 2.1



Data property hierarchy:

Asserted

- owl:topDataProperty
  - float_coordinates_x
  - float_coordinates_y
  - float_coordinates_z
  - float_size_x
  - float_size_y
  - float_size_z
  - float_transformation_matrix_element_1_1
  - float_transformation_matrix_element_1_2
  - float_transformation_matrix_element_1_3
  - float_transformation_matrix_element_1_4
  - float_transformation_matrix_element_2_1
  - float_transformation_matrix_element_2_2
  - float_transformation_matrix_element_2_3
  - float_transformation_matrix_element_2_4
  - float_transformation_matrix_element_3_1
  - float_transformation_matrix_element_3_2
  - float_transformation_matrix_element_3_3
  - float_transformation_matrix_element_3_4
  - float_transformation_matrix_element_4_1
  - float_transformation_matrix_element_4_2
  - float_transformation_matrix_element_4_3
  - float_transformation_matrix_element_4_4
  - lexicalReference
  - name

Figure 3: Datatypes Visualization in Protégé

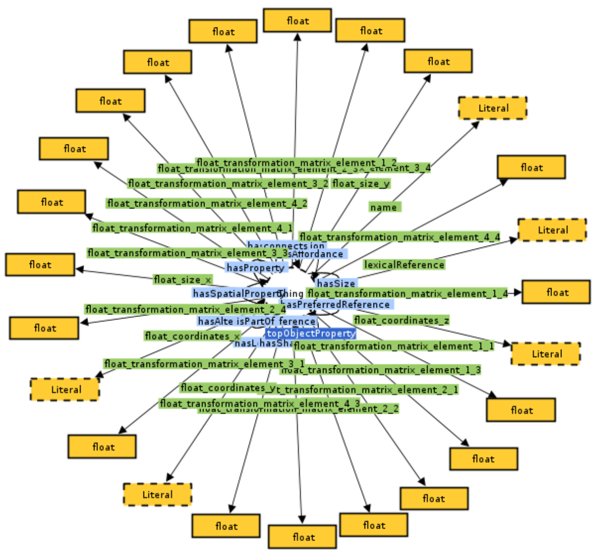

float
Literal
float_transformation_matrix_element_1_2
float_transformation_matrix_element_3_2
float_size_y
float_transformation_matrix_element_4_2
name
float_transformation_matrix_element_4_1
float_transformation_matrix_element_4_4
float_transformation_matrix_element_3_3
hasAffordance
hasProperty
lexicalReference
float_size_x
hasSize
hasSpatialProperty
Thing
float_transformation_matrix_element_1_4
float_transformation_matrix_element_2_4
hasPreferredReference
isPartOf
float_coordinates_z
float_coordinates_x
topObjectProperty
float_transformation_matrix_element_1_1
float_transformation_matrix_element_3_1
float_transformation_matrix_element_1_3
float_coordinates_y
float_transformation_matrix_element_2_1
float_transformation_matrix_element_4_3
float_transformation_matrix_element_2_2

Figure 4: Datatypes

Object Properties

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

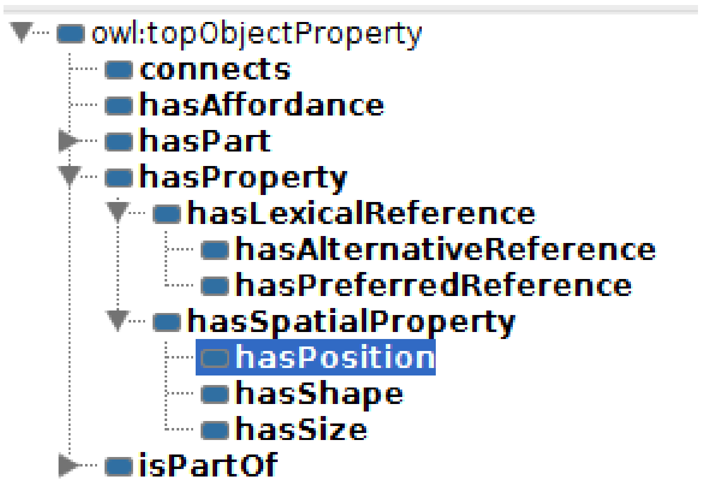


Figure 5: Object Properties Class Tree

As an example, consider the following set of owl statements about the ObjectProperty hasPosition. This property is of the type IrreflexiveProperty 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>
```

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 3D 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.

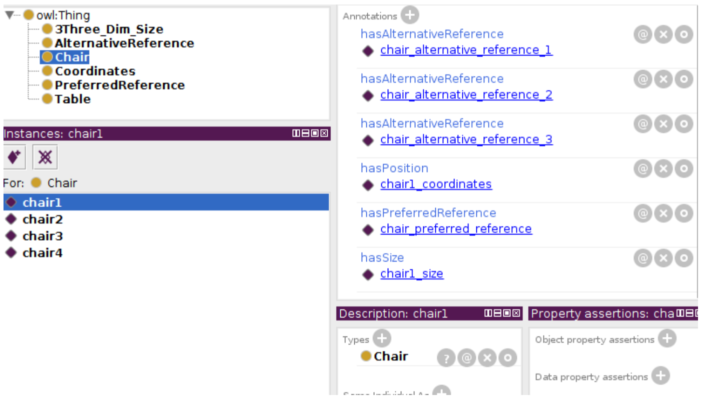
An instance of the class Chair

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:



owl:Thing
- 3Three_Dim_Size
- AlternativeReference
- Chair
- Coordinates
- PreferredReference
- Table

Instances: chair1

For: Chair
- chair1
- chair2
- chair3
- chair4

Annotations

hasAlternativeReference
chair_alternative_reference_1

hasAlternativeReference
chair_alternative_reference_2

hasAlternativeReference
chair_alternative_reference_3

hasPosition
chair1_coordinates

hasPreferredReference
chair_preferred_reference

hasSize
chair1_size

Description: chair1

Types
Chair

Property assertions: cha

Object property assertions

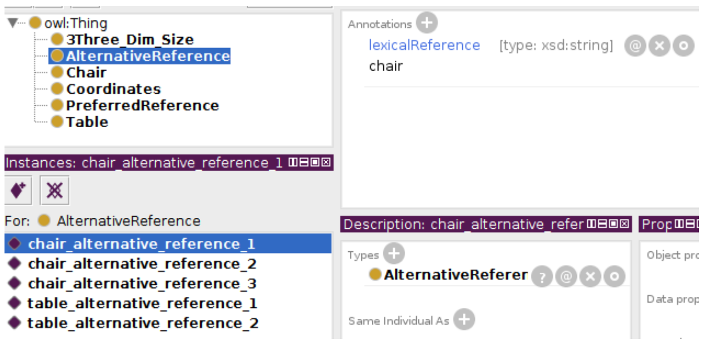
Data property assertions

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>
```


owl:Thing
3Three_Dim_Size
AlternativeReference
Chair
Coordinates
PreferredReference
Table

Annotations
lexicalReference [type: xsd:string]
chair

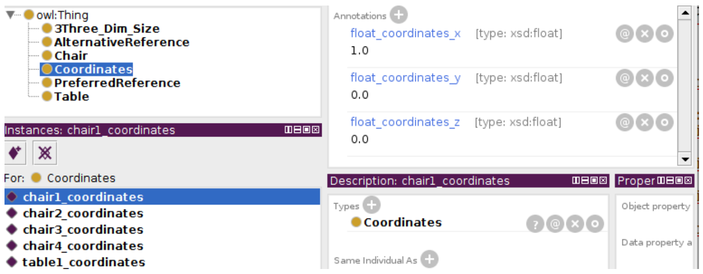
Instances: chair_alternative_reference_1
For: AlternativeReference
chair_alternative_reference_1
chair_alternative_reference_2
chair_alternative_reference_3
table_alternative_reference_1
table_alternative_reference_2

Description: chair_alternative_refer
Prop
Types
AlternativeReferer
Same Individual As
Object pro
Data prop

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>
```


owl:Thing
3Three_Dim_Size
AlternativeReference
Chair
Coordinates
PreferredReference
Table

Annotations
float_coordinates_x [type: xsd:float]
1.0
float_coordinates_y [type: xsd:float]
0.0
float_coordinates_z [type: xsd:float]
0.0

Instances: chair1_coordinates
For: Coordinates
chair1_coordinates
chair2_coordinates
chair3_coordinates
chair4_coordinates
table1_coordinates

Description: chair1_coordinates
Proper
Types
Coordinates
Same Individual As
Object property
Data property a

Figure 8: Coordinates chair 1

3 Involved Technologies

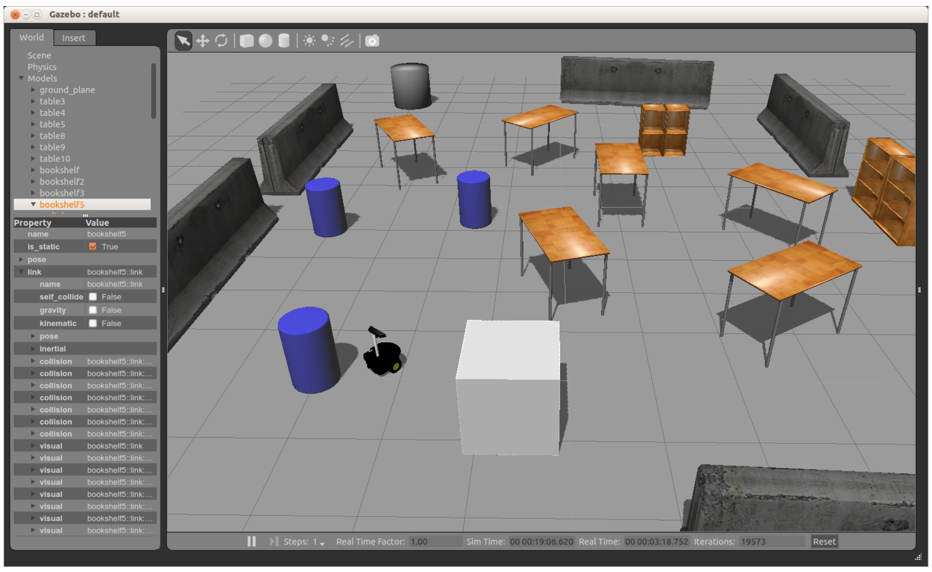
3.1 JENA Ontology API

There are several languages available for representing ontology information on the semantic web. They range from the most expressive, OWL Full, through to the weakest, RDFS. With RDFS it is possible to build a simple hierarchy of concepts, and a hierarchy of properties. 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 as the inverse of another.

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

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. ROS was built from the ground up to encourage collaborative robotics software development. A ROS system consists of a large number of modules that pass data to one another using a inter-process communication. Each node sends and receives information to and from other nodes of the system using *Topics*.



Property	Value
name	bookshelf5
is_static	✓ True
pose	
link	bookshelf5::link
name	bookshelf5::link
self_collide	☐ False
gravity	☐ False
kinematic	☐ False
pose	
inertial	
collision	bookshelf5::link:...
collision	bookshelf5::link:...
collision	bookshelf5::link:...
collision	bookshelf5::link:...
collision	bookshelf5::link:...
collision	bookshelf5::link:...
collision	bookshelf5::link:...
visual	bookshelf5::link
visual	bookshelf5::link:...
visual	bookshelf5::link:...
visual	bookshelf5::link:...
visual	bookshelf5::link:...
visual	bookshelf5::link:...
visual	bookshelf5::link:...
visual	bookshelf5::link:...

Steps: 1 Real Time Factor: 1.00 Sim Time: 00 00:19:06.620 Real Time: 00 00:03:18.752 Iterations: 19573

Figure 9: ROS simulation

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. There is no central routing service.
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 Gazebo Simulation Environment

Robotics implies robots. Most part of these platforms are used for research purposes and are custom built to investigate a particular aspect of interest. 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, or at least minimized, 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, TurtleBot and even champions of DARPA robotics challenge are available. 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 messages.

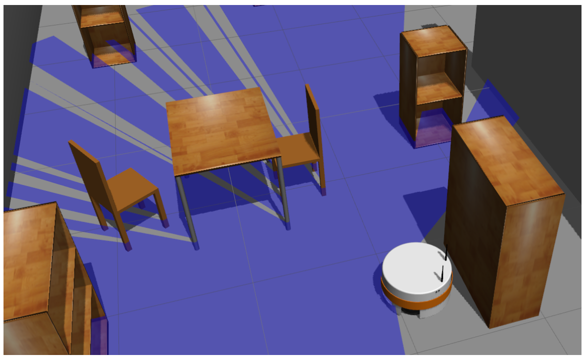


Figure 10: Gazebo Environment

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. In this project Gazebo 8.0 and ROS Indigo have been used. This combination ensures the compatibility required to flawlessly run the application.

4 The Architecture

This project has been implemented using ROS framework. It is a modular application that enables other nodes to query and interact with the knowledge base. An external interface node allows several operations including the export of the Assertion Box. Four nodes are involved in this application and the communication between them relies on an inter-process protocol handled by ROS.

4.1 System description

In the following subsection, the implemented nodes and their interaction with the system will be discussed.

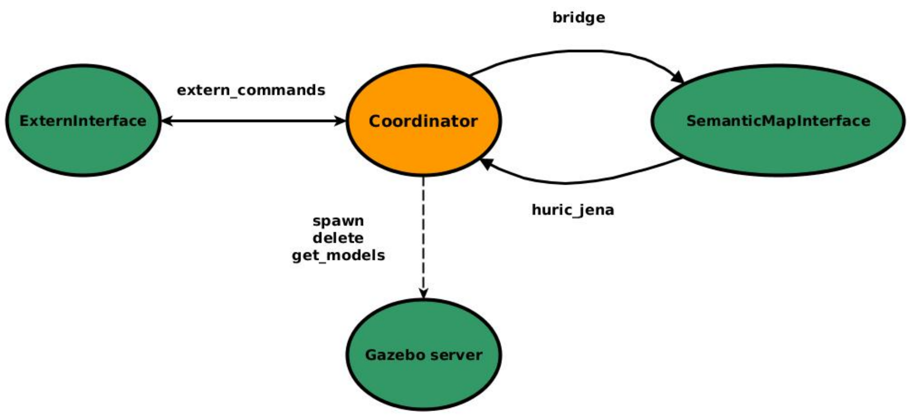


Figure 11: System Architecture

4.1.1 SemanticMapInterface Node

The knowledge base is loaded from the SemanticMapInterface which is a Java node based on the Jena Ontology API. This process allows requesting nodes to access and manipulate the ontology using a predefined set of operations.

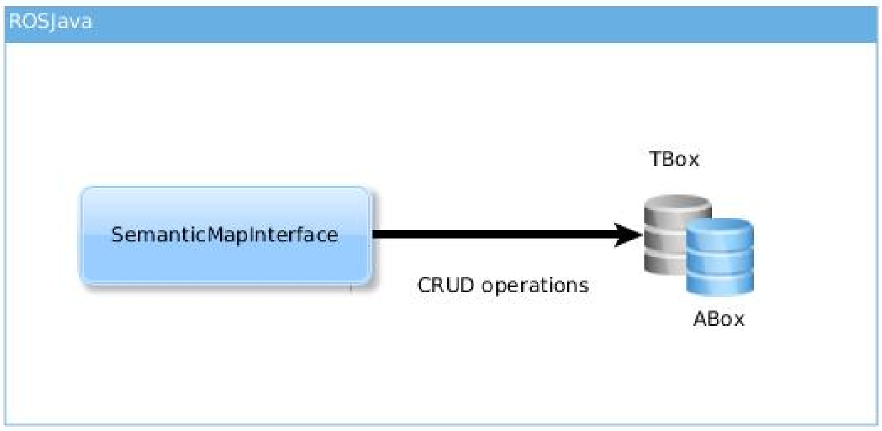


Figure 12: SemanticMapInterface Node

The exposed operations include:

- Loading the Terminology (TBox) and Assertion Box (ABox) into memory,
- Listing instances, their spacial properties and their preferred lexical reference,
- Adding a new entity of a particular class, with the spacial and lexical properties specified as arguments,
- Updating properties of active entities,
- Removing active entities,
- Invoking OWL-FULL reasoner and performing inference operation given a domain model,
- Exporting in OWL format the list of instances present or the augmented ABox with derived properties.

4.1.2 Gazebo Nodes

Gazebo is a free (freedom) and open source robot simulation environment. It provides high performance physics engines to model real world dynamics, render 3D objects and environments. The Gazebo server *gzserver* executes the simulation process including physics updates. The Gazebo client runs the UI and provides a 3D environment and handy controls to modify simulation properties. These nodes provide a set of ROS API's that allows users to modify and get information about various aspects of the simulated world.

Topics can be used to set the pose and twist of a model by publishing desired model state message to /gazebo/set_model_state_topic. It is possible to retrieve model and link states using Topics. Gazebo publishes /gazebo/link_states and /gazebo/model_states_topics, containing pose and twist information of objects in simulation with respect to the gazebo world frame. Services can be used to create/spawn and destroy models dynamically in simulation.

4.1.3 Extern Interface Node

This node has been implemented to test the required specifications. It simulates external requests by publishing predefined encoded messages on a topic to which the Coordinator is subscribed. The supported operation are listed in section 1.1

4.1.4 Coordinator Node

The Coordinator node is a python script that performs several actions and communicates with other nodes in the system. One responsibility of the coordinator is to keep track of instances and trigger update requests when a 3D model is manually grabbed and moved in the simulation. This node provides a dynamic rooting of requests from the External Interface node to the ontology handler or Gazebo.

Basic operations include:

- Handles high level requests from other nodes such as visualizing or exporting the list of instances present in the ontology.
- Tracks 3D absolute positions of objects in the simulation,
- Notifies the ontology manager that an instance's property has changed,
- Forwards 3D spawn and delete task to Gazebo,
- Ensures consistency between the Assertion Box and the simulated world.

4.2 Test cases

The following test cases have been performed:

- Initialization from non empty ontology (TBox and ABox),
- Retrieve collection of instances and render 3D models in Gazebo,
- Request a detailed list of properties about active entities,
- Manually drag objects in scene and sync new properties with the ABox,
- Request Instances enumeration from ExternalInterface,
- Perform a consistency check.
- Requests:
  - Deletion of entities,
  - Insertion of default instance,
  - Insertion of instance with specified type, spacial and lexical property,
  - Export to specified file.

5 The Application

The application consists of a number of indipendent nodes that comprise a graph. The communication between nodes strongly relies on a publish/subscribe mechanism useful to exchange data in a distributed system.

5.1 Use cases

5.1.1 Initialization

The application is launched by executing the init.launch file. The Roslaunch tool allows to launch multiple ROS nodes as well as set several parameters for the simulation environment. the initialization process brings up the master node, roscore, the coordinator, the semanticMapInterface and Gazebo.

Once initialized, the SemanticMapInterface loads from a specified absolute path the Terminology Box and the Assertions Box as Ontology Models.

```
/**
 * Importing Tbox
 */
 tbox = ModelFactory.createOntologyModel( OntModelSpec.OWL_MEM );
 OntDocumentManager dm_tbox = tbox.getDocumentManager();
 dm_tbox.addAltEntry(SOURCE+TBOX_FILE,"file:"+TBOX_FILE);
 tbox.read(SOURCE+TBOX_FILE,"RDF/XML");

/**
 * Importing Abox
 */
 abox = ModelFactory.createOntologyModel( OntModelSpec.OWL_MEM);
 OntDocumentManager dma = abox.getDocumentManager();
 dma.addAltEntry( SOURCE + ABOX_FILE , "file:" + ABOX_FILE);
 abox.read(SOURCE + ABOX_FILE,"RDF/XML");
```

The reasoner API supports the notion of specializing a reasoner by binding it to a set of schema or ontology data using the bindSchema call. The specialized reasoner can then be attached to different sets of instance data using bind calls. It is worth noting that in this project the schema (TBox) and instance (ABox) data were saved in two separate files.

```
Reasoner reasoner = ReasonerRegistry.getOWLReasoner();
reasoner = reasoner.bindSchema(tbox);
OntModelSpec ontModelSpec=OntModelSpec.OWL_MEM_MICRO_RULE_INF;
ontModelSpec.setReasoner(reasoner);
InfModel infmodel = ModelFactory.createInfModel(reasoner,abox);
```

This is equivalent to an Ontology Model with Reasoner capabilities specialized on the ABox.

```
infModel = ModelFactory.createOntologyModel( ...
    OntModelSpec.OWL_MEM_MICRO_RULE_INF, abox);
```

Typically the ontology languages used with the semantic web allow constraints to be expressed, the validation interface is used to detect when such constraints are violated by some data set. The InfModel.validate() interface performs a global check across the schema and instance data looking for inconsistencies.

```
/**
 * Consistency Check. Returns true if passed
 */
 private static boolean performConsistencyCheckWith(InfModel inf) {
 boolean res = false;

 ValidityReport validity = inf.validate();
 if (validity.isValid()) {
    System.out.println(""+ NODE_NAME + "\tConsistency Check:\n Passed\n");
    res = true;
 } else {
    System.out.println(""+ NODE_NAME + "\tConsistency Check:\n Conflicts\n");
    for (Iterator i = validity.getReports(); i.hasNext(); ) {
        System.out.println(" - " + i.next());
    }
 }
 return res;
}
```

When the Coordinator is initialized it sends on a topic called Bridge an init request. The request is captured by the SemanticMapInterface and the $getAllInstances(OntModel)$ method is called.

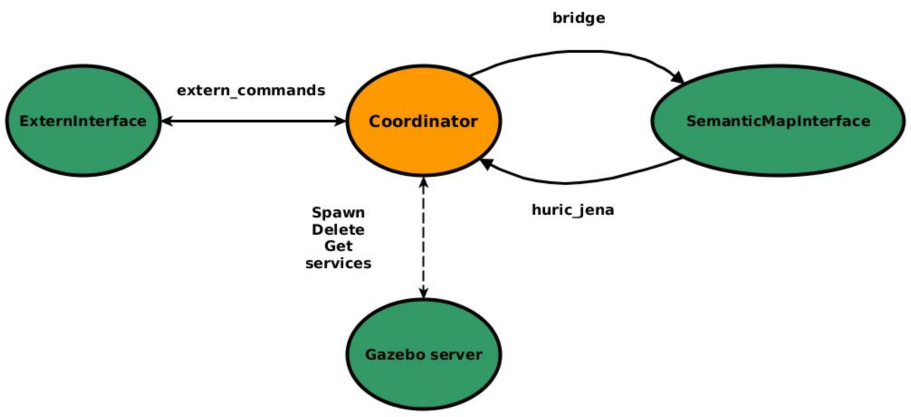


Figure 13: Data exchange between Coordinator and SemanticMapInterface

A collection of instances is retrieved and saved in a HashMap. This method performs the following SPARQL query which returns all Furniture and Drink[3] entities.

```
String queryString = "PREFIX rdf: ...
    <http://www.w3.org/1999/02/22-rdf-syntax-ns#>" +
                    "prefix rdfs: <"+RDFS.getURI()+">\n" +
                    "PREFIX xsd: <http://www.w3.org/2001/XMLSchema#> " +
                    "PREFIX hasPosition: <" + NS + POSITION +"> " +
                    "PREFIX hasRef: <" + NS + PREF_REF +"> " +
```

[3]Furniture and Drink are classes defined in the Terminology Box

```
                    "prefix semantic_mapping_domain_model: <" + DOMAIN_MODEL_NS + ...
                        "#> \n"+
                    "prefix semantic_mapping_1: <" + SEMANTIC_MAP_NS + "#> \n"+

                    "PREFIX coordx: <" + NS + COORD_X +"> " +
                    "PREFIX coordy: <" + NS + COORD_Y +"> " +
                    "PREFIX coordz: <" + NS + COORD_Z +"> " +
                    "PREFIX prefRef: <" + NS + LEXICAL +"> " +


                    "SELECT DISTINCT ?uri ?class ?x ?y ?z ?lex "+
                    "WHERE {" +
                        "{"+
                        "?uri a ?class ." +
                        "?class rdfs:subClassOf ...
                            semantic_mapping_domain_model:Furniture ."+
                        "?uri hasPosition: ?pos ." +
                        "?uri hasRef: ?ref ." +
                        "?ref prefRef: ?lex ." +
                        "?pos coordx: ?x . " +
                        "?pos coordy: ?y . " +
                        "?pos coordz: ?z " + "} UNION {"+
                        "?uri a ?class ." +
                        "?class rdfs:subClassOf ...
                            semantic_mapping_domain_model:Drink ." +
                        "?uri hasPosition: ?pos ." +
                        "?uri hasRef: ?ref ." +
                        "?ref prefRef: ?lex ." +
                        "?pos coordx: ?x . " +
                        "?pos coordy: ?y . " +
                        "?pos coordz: ?z " + "}"+
"}" ;
```

The resulting information are embedded in a message and sent on a topic called *Huric_jena*. The Coordinator Node receives and parses the message.

The communication between Gazebo and the Coordinator uses services which are defined as pair of messages: one for the request and one for the reply. A providing ROS node offers a service under a string name, and a client calls the service by sending the request message and awaiting the reply.

Once the reponse from the SemanticMapInterface has been received, the Coordinator calls a special service that allows to create and render 3D models dynamically in the simulation environment of Gazebo.

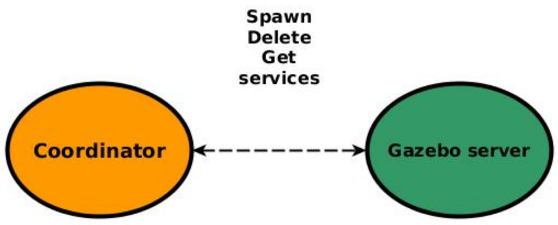


Figure 14: Data exchange between the Coordinator and Gazebo

5.1.2 Insertion of entities

Considering a real world scenario in which a perception node recognizes an object from a stream of a point cloud data, instances of a given class should be dynamically inserted in the Assertion Box by publishing a predefined command on a topic.
One node can create an instance by specifying its super class, its spacial positions and preferred lexical reference. Another faster way include the possibility to add a Chair in fixed position. These insertion methods are issued by the ExternInterface node, captured by the Coordinator and forwarded to the SemanticMapInterface.

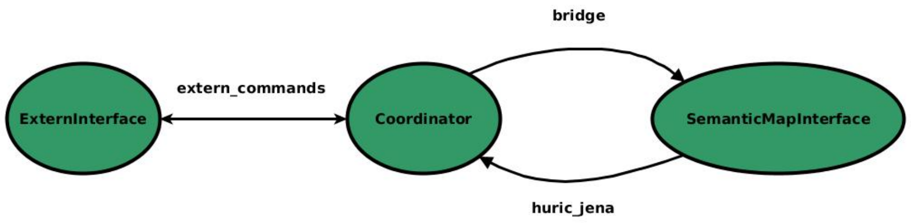


Figure 15: Data exchange between ExternInterface, Coordinator and SemanticMap-Interface

The Java Node provides two methods to insert an instance of given type into the ontology model loaded in memory:

- Triple manipulation based
- SPARQL based

Jena Ontology API provides a full set of methods to manipulate statements. The handleAddEntityRequest(ontModel,ontClass, uri, pose,lexicalReference) method is given below:

```
/**
 * Handles AddEntityRequests from orchestrator using Jena API
 *  Creates a new instance, adds properties to it and perform consistency check
 * If test is passed, leaves the newly created instance, otherwise deletes it
 */
    private static boolean handleAddEntityRequest(OntModel abox, OntClass ...
        ontClass, String uriInstance, String posX,
            String posY, String posZ, String lexicalReference){
        boolean res = false;

        OntClass prefrefClass = abox.getOntClass(NS+PREF_REF_CLASS);
        OntClass coordinatesClass = abox.getOntClass(NS+COORDINATES_CLASS);

        String uriBase = ...
            "http://www.semanticweb.org/ontologies/2016/1/semantic_mapping_1#";
        // Create instance
        //OntClass class1 = abox.getOntClass(NS+ontClass);

        if (ontClass == null)
            return false;
        else {
            // Create Individuals
            Individual i1 = abox.createIndividual(uriInstance,ontClass);
            Individual prefRefInd = ...
                abox.createIndividual(uriBase+lexicalReference
            +"_pref_ref",prefrefClass);
            Individual coordinatesInd = ...
                abox.createIndividual(uriBase+lexicalReference
            +"_pref_ref",coordinatesClass);


            // Create Datatype Property
            DatatypeProperty lexicalRef = abox.getDatatypeProperty(NS + LEXICAL);
            Literal ref = abox.createTypedLiteral(lexicalReference);
            Statement refStatement = abox.createStatement(prefRefInd, ...
                lexicalRef, ref);
            abox.add(refStatement);

            //  Bind pref reference individual to object individual
            ObjectProperty hasPrefRef = abox.getObjectProperty(NS+PREF_REF);
            Statement bindPrefRef = abox.createStatement(i1, hasPrefRef, ...
                prefRefInd);
            abox.add(bindPrefRef);


            // Create Datatype Property for coordinate X
            DatatypeProperty posx = abox.getDatatypeProperty(NS + COORD_X);
            Literal posXFloat = abox.createTypedLiteral(posX);
            Statement posXStatement = abox.createStatement(coordinatesInd, ...
                posx, posXFloat);
            abox.add(posXStatement);
            // Create Datatype Property for coordinate X
            DatatypeProperty posy = abox.getDatatypeProperty(NS + COORD_Y);
            Literal posYFloat = abox.createTypedLiteral(posY);
            Statement posYStatement = abox.createStatement(coordinatesInd, ...
                posy, posYFloat);
            abox.add(posYStatement);
            // Create Datatype Property for coordinate X
            DatatypeProperty posz = abox.getDatatypeProperty(NS + COORD_Z);
            Literal posZFloat = abox.createTypedLiteral(posZ);
            Statement posZStatement = abox.createStatement(coordinatesInd, ...
                posz, posZFloat);
            abox.add(posZStatement);


            //  Bind position individual to object individual
            ObjectProperty hasPosition = abox.getObjectProperty(NS+POSITION);
            Statement bindPose = abox.createStatement(i1, hasPosition, ...
                coordinatesInd);
            abox.add(bindPose);

            // Now perform consistency check

            InfModel infModel = ModelFactory.createOntologyModel( ...
                OntModelSpec.OWL_MEM_MICRO_RULE_INF, abox);
            res = performConsistencyCheckWith(infModel);
        }

        return res;
}
```

The second method is called AddEntityRequest(abox,ontclass,uri,pose,lexicalReference) and it is based on a SPARQL query:

```
/**
 * Handles AddEntityRequests using SPARQL
 *  Creates a new instance, adds properties to it and perform consistency check
 * If test is passed, leaves the newly created instance, otherwise deletes it
 */
 private static boolean AddEntityRequest(OntModel abox, OntClass ontClass, ...
     String uriInstance, String posX,
            String posY, String posZ, String lexicalReference){
 boolean res = true;

 String uri_Instance[] = uriInstance.split("#");

 if (ontClass == null) {
    res = false;
    System.out.print("\n"+ NODE_NAME + "\t[ Add Entity ] Class problem\n");
 } else {
            String queryString = "" +
            "prefix rdfs: <"+ RDFS.getURI() +">\n" +
            "prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> \n"+
            "PREFIX xsd: <http://www.w3.org/2001/XMLSchema#> "

            + "prefix semantic_mapping_domain_model: <" + DOMAIN_MODEL_NS + ...
                "#> \n"
            + "prefix semantic_mapping: <" + SEMANTIC_MAP_NS + "#> \n"
            + "PREFIX class: <"+ ontClass.getURI() +">\n"


            + "insert data { semantic_mapping:"+uri_Instance[1] +" rdf:type ...
                class: . "


            // Add hasPosition Irreflexive ObjectProperty
            + "semantic_mapping:" + uri_Instance[1] + " ...
                semantic_mapping_domain_model:hasPosition semantic_mapping:" ...
                + uri_Instance[1]
            + "_coordinates . "

            // Add Instance Coordinates
            + "semantic_mapping:" + uri_Instance[1]+ "_coordinates rdf:type ...
                semantic_mapping_domain_model:Coordinates . "

            // Add datatype Property
            + "semantic_mapping:" + uri_Instance[1] + "_coordinates ...
                semantic_mapping_domain_model:float_coordinates_x \""
            + posX + "\"^^xsd:float . "
            + "semantic_mapping:" + uri_Instance[1] + "_coordinates ...
                semantic_mapping_domain_model:float_coordinates_y \"" + posY
            + "\"^^xsd:float . "
            + "semantic_mapping:" + uri_Instance[1] + "_coordinates ...
                semantic_mapping_domain_model:float_coordinates_z \"" + posZ
            + "\"^^xsd:float . "

            // Add hasPreferredReference ObjectProperty
            + "semantic_mapping:" + uri_Instance[1] + " ...
                semantic_mapping_domain_model:hasPreferredReference ...
                semantic_mapping:" + uri_Instance[1]
            + "_preferredReference . "

            // Create Instance Lexical Reference
            + "semantic_mapping:" + uri_Instance[1] + "_preferredReference ...
                rdf:type semantic_mapping_domain_model:PreferredReference . "

            + "semantic_mapping:" + uri_Instance[1] + "_preferredReference ...
                semantic_mapping_domain_model:lexicalReference \"" + ...
                lexicalReference
            + "\"^^xsd:string . " + "} \n ";

```

```
            UpdateAction.parseExecute(queryString, abox);

            // Check inconsistencies
            InfModel infModel = ModelFactory.createOntologyModel( ...
                OntModelSpec.OWL_MEM_MICRO_RULE_INF, abox);
            res = performConsistencyCheckWith(infModel);
        }

        return res;
}
```

Once the instance has been inserted into the ontology model, the collection of new entities is once again sent on the *huric_jena* topic and captured by the coordinator which invokes the Gazebo service to render the 3D model of the newly created instances at the provided position in space.

5.1.3 Deletion of an entity

In a real world scenario, objects can move or be manipulated by humans. A delete function is needed in order to handle cases in which the object moves out of the field of perception of the robot. To delete a model a special routine is invoked.
The request is published on the external_commands topic and captured by the Coordinator. This node forwards the request to SemanticMapInterface node and waits for the response. If the abox has been correctly updated, it invokes the delete_model service exposed by the Gazebo node. The resulting 3D environment is consistent with the instance data loaded in memory.

The SemanticMapInterface provides a method the handle this request. In Section 2.2 we have seen that for each furniture object there is an linear position and lexical reference instance associated with it. In order to completely delete a furniture object, these instances have to be canceled as well. The query that handles this operation is in the Listing below.

```
/**
  * Handles removeEntityRequests from orchestrator:
  * Looks for an instance with uri 'uriInstance', if the instance does not ...
     exist returns false.
  * If there is match, it deletes it and returns true.
  *
  * Attention : What happens if an instance of a master concept relating two ...
     entities is deleted?
  *             No one should be able to alter its knowledge directly.
  *
  *             Maybe, add a negative weight to the instance or a tag
  */
private static boolean handleRemoveEntityRelatedRequest(OntModel abox, String ...
    uriInstance) {
    boolean res = false;

    String uri_Instance[] = uriInstance.split("#");
    //  Removing IS A statement
    String queryStringDelete = "" +
            "prefix rdfs: <"+RDFS.getURI()+">\n" +
            "prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> \n"+
            "PREFIX xsd: <http://www.w3.org/2001/XMLSchema#> "
            + "prefix semantic_mapping_domain_model: <" + DOMAIN_MODEL_NS + ...
                "#> \n"
            + "prefix semantic_mapping: <" + SEMANTIC_MAP_NS + "#> \n"
            + "delete { semantic_mapping:"+uri_Instance[1] +" ?pred ?obj }  "
        + "where { semantic_mapping:"+uri_Instance[1] + " ?pred ?obj }";

            UpdateAction.parseExecute(queryStringDelete, abox);


    //  Removing coordinates statement
    String queryStringPosDelete = "" +
            "prefix rdfs: <"+RDFS.getURI()+">\n" +
            "prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> \n"+
            "PREFIX xsd: <http://www.w3.org/2001/XMLSchema#> "
            + "prefix semantic_mapping_domain_model: <" + DOMAIN_MODEL_NS + ...
                "#> \n"
            + "prefix semantic_mapping: <" + SEMANTIC_MAP_NS + "#> \n"
            + "delete { semantic_mapping:"+uri_Instance[1] + "_coordinates ...
                ?pred ?obj }  "
        + "where { semantic_mapping:"+uri_Instance[1] + "_coordinates ?pred ...
            ?obj }";

            UpdateAction.parseExecute(queryStringPosDelete, abox);

    String queryStringRefDelete = "" +
            "prefix rdfs: <"+RDFS.getURI()+">\n" +
            "prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> \n"+
            "PREFIX xsd: <http://www.w3.org/2001/XMLSchema#> "
            + "prefix semantic_mapping_domain_model: <" + DOMAIN_MODEL_NS + ...
                "#> \n"
            + "prefix semantic_mapping: <" + SEMANTIC_MAP_NS + "#> \n"
            + "delete { semantic_mapping:"+uri_Instance[1] + ...
                "_preferred_reference ?pred ?obj }  "
        + "where { semantic_mapping:"+uri_Instance[1] + "_preferred_reference ...
            ?pred ?obj }";

            UpdateAction.parseExecute(queryStringRefDelete, abox);

    return res;
}
```

5.1.4 Update entity's properties

The system provides an update function to modify properties of an existing entity in the ABox. By publishing a formatted command on the extern_commands topic, the Coordinator node forwards the request to the SemanticMapInterface. The method responsible for the update is called handleUpdateEntityRequest(abox,ontClass, uriInstance, posX, posY, posZ, lexicalReference).

```
/**
  * Handles updateEntityRequests using SPARQL
  * Looks for an instance with uri 'uriInstance', if the instance does not ...
    exist returns flase.
  * If there is match, it looks for the properties Pose and LexicalReference ...
    and updates it.
  * A consistency if invoked. If the test is passed, true is returned. ...
    Otherwise False.
  */
private static boolean handleUpdateEntityRequest(OntModel abox,OntClass ontClass,
 String uriInstance, String posX,
         String posY, String posZ, String lexicalReference) {
    boolean res = false;

    System.out.println(""+ NODE_NAME + "\tDeleting Instance " + uriInstance);
    handleRemoveEntityRelatedRequest(abox,uriInstance);

    System.out.print("\n"+ NODE_NAME + "\tNow inserting: "+ posX + ", " + ...
        posY + " , " + posZ);
    if (AddEntityRequest(abox,ontClass,uriInstance,posX,posY,
    posZ,lexicalReference)) {
    System.out.println("\n"+ NODE_NAME + "\tInstance inserted successfully");
    res = true;
    } else {
            System.out.print(""+ NODE_NAME + "\tError could not insert ...
                instance consistency problem");
        }

    return res;
}
```

The update routine starts by looking for an instance with the provided uri. If the there is no match, false is returned. Then it deletes the instance itself and its related properties with the method presented in Section 5.1.3. Finally the new instance is added by invoking the method discussed in Section 5.1.2 and a consistency check is invoked.

It is also possible to update an instance by dragging it in the 3D environment of Gazebo. The coordinator node keeps track of the position of all instances and compare them with the corresponding coordinate properties in the Assertion Box. As soon as the position of an instance is modified and a predefined threshold is crossed, an update routine is triggered and the ABox is updated. Collisions have not been taken into account.

5.2 Exporting Ontology

Human machine interfaces require a context-aware interaction. In this process, since both parts make references to real world objects, an ontological model of the environment and a description of the involved entities may help in improving the accuracy of the interpretation of an user utterance.
The first step towards this direction is to provide an export feature that allows the robotic platform to be aware of the surrounding environment and nearby objects.

```
/**
  * Exports Ontology if consistency check passed
  */
private static boolean exportTo(InfModel inf, OntModel abox, String ...
    absolutePath, String fileName){
    boolean res = false;
    if (performConsistencyCheckWith(inf)) {
        exportOntologyOnt(abox,absolutePath,fileName);
        res = true;
    }
    else {
        res = false;
    }
    return res;
}

```

```java
/**
  * Exports Ontology to a file with provided name and path.
  */
private static boolean exportOntologyOnt(OntModel inf, String ...
    absoluteFileName, String fileName){
    boolean res = false;
    FileWriter out = null;

        try{
            out = new FileWriter(absoluteFileName+fileName);
            inf.write(out,"RDF/XML-ABBREV");
            res = true;
            System.out.println(""+ NODE_NAME + "\tConsistency check passed. ...
                Writing to file:\t"+fileName);
        }
        catch (IOException a){
            System.out.println(""+ NODE_NAME + "\tExport Problem");
            a.printStackTrace();
        }
        finally{
            if (out!=null){
                try {out.close();} catch(IOException ex) {}
            }
        }

        return res;
}
```

The SemanticMapInterface export the ontology to OWL file in the file system and sends the list of entities to the Coordinator Node.

Listing Entities

In order to retrieve the list of entities perceived by the robot, another feature has been taken into account. This action involves the Coordinator, the SemanticMapHandler and the ExternInterface nodes. The Coordinator prepares the response in two formats, a json encoded list of entities and corresponding properties

```json
{
   "http://www.semanticweb.org/ontologies/2016/1/semantic_mapping_1#table1":{
      "type_full":"http://www.semanticweb.org/ontologies/2016/1/
      semantic_mapping_domain_model#Table",
      "coordinates":"2.0,2.0,0.0",
      "preferredLexicalReference":"table",
      "atom":"table1",
      "coordinate":{
         "y":"2.0",
         "x":"2.0",
         "z":"0.0"
      },
      "type":"Table"
   },{
   ...
   },
   "http://www.semanticweb.org/ontologies/2016/1/semantic_mapping_1#beer_can_1":{
      "type_full":"http://www.semanticweb.org/ontologies/2016/1
      /semantic_mapping_domain_model#Beer",
      "coordinates":"1.8,2.2,1.0",
      "preferredLexicalReference":"Beer",
      "atom":"beer_can_1",
      "coordinate":{
         "y":"2.2",
         "x":"1.8",
         "z":"1.0"
      },
      "type":"Beer"
   }
}
```

and human readable output for debug purposes.

5.3 Future works

The application supports a limited number of entity types. Each instance has to be associated with its corresponding 3D model and inertia matrix. In this project four types of objects are supported: tables, chairs, beer and coke cans.

From a robotic perspective, the applications of this ROS package are related to Human Centered Robotics.

LU4R - Adaptive spoken Language Understanding For Robots

The adaptive spoken Language Understanding chain For Robots tool is the result of the collaboration between the SAG group at Tor Vergata University of Rome and the Laboratory of Cognitive Cooperating Robots (Lab.Ro.Co.Co.) at Sapienza, University of Rome. LU4R[4] receives as input one or more transcriptions of a spoken command and produces an interpretation that is consistent with a linguistically-justified semantic representation, coherent with the perceived environment. LU4R is based on a spoken language understanding (SLU) process that integrates perceptual and linguistic information in order to produce command interpretations that coherently express constraints about the world and the hosting robotic platform.

It is a ROS compatible Java application in which the client runs on the robotic platform and the server on a remote machine.
Actions expressed in user utterances can be modeled as semantic frames, sentences expressing commands are automatically analyzed by the chain by applying data driven methods trained over the Human Robot Interaction Corpus (HuRIC)[3].
In its default mode of operation, the chain requires a list of sentences paired with their transcription confidence score and the ranking position among the hypotheses list. In order to enable the environment based interpretation process, additional information are required. The json described in Section 5.2 can be forwarded to the LU4R node.

[4]For further information visit the official web page at http://sag.art.uniroma2.it/lu4r.html

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* $: //wiki.ros.org/rosjava_build_tools/Tutorials/hydro$*"*.

[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* $: //jena.apache.org/documentation/serving_data/$*"*.

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