Roboy Memory Module Documentation

Release 0.0.

Aug 31, 2017

Usage and Installation

The goal of the project is to provide Roboy with modern graph-based Knowledge Representation.

Roboy should feature ability to remember information about himself:

- his name
- his age
- his origin
- his location
- his friends

etc.

The same is applicable to Roboy speaking about people who are friends with him. Roboy should tell information about a person or an object and be able to provide basic automatic inference (supported by the graph nature of KR). This way, Roboy Memory Module serves as a long-term memory repository of actionable information acquired by other Roboy modules. Persistency layer is presented by a Neo4j graph database.

Upon incoming request, a Java client will pre-process the request and initiate transaction with the database. Two ways of communication between Roboy Java client and Neo4J database are supported: communication using Neo4J driver operating Cypher query language and Neo4J native Java API. Cypher query language offers more flexible querying while communications via Neo4J Java API are implemented as usage-specific routines. Interfaces are implemented on top of ros through the Java client. The input is any type of information Roboy can retrieve from environment abiding by Knowledge Representation reference in format of Roboy Communication Standard protocol, the output are pieces of data related to the requested scope in the same form.

The current main tasks of this project are:

- Fill the memory in with all possible information about Roboy team
- Ensure KR retention (through a population script)
- Finish and evaluate the rosjava service
- Improve KR (more powerful inference)

CHAPTER 1

Relevant Background Information and Pre-Requisits

A User should be familiar with:

- Knowledge Representation theory
- graph-based KRs
- Roboy Communication Protocol
- Roboy Knowledge Representation Architecture

A Developer should be familiar with:

- graph-based DBs (preferably Neo4j)
- Knowledge Representation theory
- Roboy Communication Protocol
- Roboy Knowledge Representation Architecture
- Java programming language
- Maven automation tool
- rosjava

Reading list for a User:

- Graph Structures for Knowledge Representation and Reasoning proceedings
- rosjava Documentation
- *Roboy Communication Standard*

Reading list for a Developer:

- OReilys Graph Databases
- Neo4j Getting Started
- Cypher RefCard
- Java Documentation

- Maven Documentation
- *Roboy Communication Standard*
- rosjava Documentation

CHAPTER 2

Requirements Overview

The **software requirements** define the system from a blackbox/interfaces perspective. They are split into the following sections:

- **User Interfaces** - *User Interfaces*
- **Technical Interfaces** - *Public Interfaces (ROS)*
- **Runtime Interfaces and Constraints** - *Technical Constraints / Runtime Interface Requirements*

CHAPTER 3

Contents:

Installation

Maven

The project requires Maven. You may get it here: Download Maven

Consider checking this entries: Install, Configure and Run

Local Neo4j Instance

There are several options (for a Unix-based OS)

Docker Container Distribution

- get the container with:

```
docker pull neo4j
```

Using the Debian Repository

- to use the repository, add it to the list of sources:

```
wget -O - https://debian.neo4j.org/neotechnology.gpg.key | sudo apt-key add -
echo 'deb https://debian.neo4j.org/repo stable/' | sudo tee /etc/apt/sources.list.
↪d/neo4j.list
sudo apt-get update
```

- install the latest Neo4j version:

```
sudo apt-get install neo4j
```

- **cd** into **/usr/bin** and run:

```
neo4j start
```

RPM repository

Follow these steps as **root**:

- add the repository:

```
rpm --import http://debian.neo4j.org/neotechnology.gpg.key
cat <<EOF>  /etc/yum.repos.d/neo4j.repo
[neo4j]
name=Neo4j RPM Repository
baseurl=http://yum.neo4j.org/stable
enabled=1
gpgcheck=1
EOF
```

- install by executing:

```
yum install neo4j-3.2.0-rc3 (or the newer version)
```

- **cd** into **/usr/bin** and run:

```
neo4j start
```

Tarball installation

- download the latest release from:

```
http://neo4j.com/download/
```

- select the appropriate **tar.gz** distribution for your platform
- extract the contents of the archive, using:

```
tar -xf <filename>
```

- refer to the top-level extracted directory as **NEO4J_HOME**
- change directory to **$NEO4J_HOME**
- run:

```
./bin/neo4j console
```

Build it yourself

- clone a git project with:

```
git clone git@github.com:neo4j/neo4j.git
```

- in the project directory do:

```
mvn clean install
```

- after building artifacts with Maven do:

```
export PATH="bin:$PATH" && make clean all
```

- **cd** into **packaging/standalone/target** and run:

```
bin/neo4j start
```

Congratulations! You have started the Neo4j instance!

Local Redis Instance

In order to compile Redis follow this simple steps:

- get the source code:

```
wget http://download.redis.io/redis-stable.tar.gz
```

- unzip the tarball:

```
tar xvzf redis-stable.tar.gz
```

- navigate to:

```
cd redis-stable
```

- compile:

```
make
```

Remote Neo4j Instance

If the local instance is not necessary, use a remote Neo4j instance by establishing a connection to the Roboy server. Please, refer to *Getting started*

Remote Redis Instance

If the local instance is not necessary, use a remote Redis instance by establishing a connection to the Roboy server. Please, refer to *Getting started*

Installing ROS

The project is using rosjava which requires ROS kinetic.

Simple installation (assuming Ubuntu 16.04 LTS):

- setup your sources.list:

```
sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc) main" 
↪>
/etc/apt/sources.list.d/ros-latest.list'
```

- set up your keys:

```
sudo apt-key adv --keyserver hkp://ha.pool.sks-keyservers.net:80
--recv-key 421C365BD9FF1F717815A3895523BAEEB01FA116
```

- update Debian package index:

```
sudo apt-get update
```

- commence desktop full installation of kinetic:

```
sudo apt-get install ros-kinetic-desktop-full
```

If the simple installation was not successful, please, refer to this guide.

Roboy Memory Package Installation

The project is implemented upon a build automation tool - Maven, so the dependencies are tracked automatically, if there is a dependency missing or dependency related exception, please leave a feedback at the GitHub repository.

- clone a git project with:

```
git clone git@github.com:Roboy/roboy_memory.git
```

Getting started

Local Neo4j Instance

Before proceeding further, please commence a user configuration step:

- please navigate inside the package folder **$ROBOY_MEMORY** to:

```
cd scripts
```

- run:

```
./user_conf.sh -u your_username -p your_password
```

- wait the script to execute.

You may proceed with your current DB now (you need to put the data there) or fetch the remote DB contents.

To copy remote Neo4j DB into your local instance:

- open the script intext editor:

```
vi backup.sh OR nano backup.sh
```

- enter the password to connect to bot.roboy.org into respective line
- run the script specifying the path where to copy the DB files:

```
./backup.sh
```

- wait the script to execute. You will find the DB in ~/Neo4J/Backups/"date"
- copy the contents of "date" directory to your local DB directory.

Warning: Be cautious! This procedure will overwrite your credentials with the remote ones, see below.

Local Redis Instance

In order to have Redis properly configured, go through the next steps:

- create a directory where to store your Redis config files and your data:

```
sudo mkdir /etc/redis
sudo mkdir /var/redis
```

- copy the template configuration file you'll find in the root directory of the Redis distribution:

```
sudo cp redis.conf /etc/redis/6379.conf
```

- create a directory that will work as data and working directory:

```
sudo mkdir /var/redis/6379
```

- in the configuration file: set the **pidfile** to /var/run/redis_6379.pid, set the **logfile** to /var/log/redis_6379.log, set the **dir** to /var/redis/6379

Before proceeding further, please commence a password configuration step:

- please navigate to Redis configuration:

```
cd /etc/redis/
```

- open configuration file with a text editor:

```
vi 6379.conf OR nano 6379.conf
```

- find the line conatining 'requirepass', uncomment it and enter your password:

```
requirepass some_passphrase
```

- save and start Redis with the updated configuration:

```
./redis-server /etc/redis/6379.conf
```

Remote Neo4j Instance

To use a remote intance of Neo4j containing the most recent Knowledge Representation, ensure your connectivity to the Roboy server. If the server is up, use the roboy_memory package in the remote mode (default):

- bolt://bot.roboy.org:7687 - for the package configuration (enter this in config file)
- http://bot.roboy.org:7474 - for the GUI access in web-browser

For this, please use a remote Neo4j password related to your specific user:

- user, a generic Roboy member
- dialog, a dialog team member
- vision, a vision team member
- memory, a memory team member (developer)

Remote Redis Instance

To use a remote instance of Redis containing the most recent faces features, ensure your connectivity to the Roboy server. If the server is up, use the roboy_memory package in the remote mode (default):

- redis://bot.roboy.org:6379/0 - for the package configuration (enter this in config file)

For this, please use the remote Redis password.

Configuring the Package

For using roboy_memory package properly, please update the configuration file with the username and password specified for you:

```
public final static String ROS_MASTER_URI = "***";
public final static String ROS_HOSTNAME = "***";
public final static String NEO4J_ADDRESS = "***";
public final static String NEO4J_USERNAME = "***";
public final static String NEO4J_PASSWORD = "***";
public final static String REDIS_URI = "***";
public final static String REDIS_PASSWORD = "***";
```

You may use either remote or local addresses and credentials. If using local configuration, then:

```
public final static String ROS_MASTER_URI = "http://127.0.0.1:11311/";
public final static String ROS_HOSTNAME = "127.0.0.1";
public final static String NEO4J_ADDRESS = "bolt://127.0.0.1:7687";
public final static String REDIS_URI = "redis://127.0.0.1:6379/0";
```

ROS Configuration (remote)

Before you can use ROS, you will need to initialize rosdep:

```
sudo rosdep init
rosdep update
```

To install dependencies for building ROS packages, run:

```
sudo apt-get install python-rosinstall python-rosinstall-generator python-wstool␣
↪build-essential
```

Afterwords, procceed with installing catkin:

```
sudo apt-get install ros-kinetic-catkin
```

Source the environment like this:

```
echo "source /opt/ros/kinetic/setup.bash" >> ~/.bashrc
source ~/.bashrc
```

Build a catkin workspace:

```
mkdir -p ~/catkin_ws/src
cd ~/catkin_ws/
catkin_make
```

Source your new setup.*sh file:

```
source devel/setup.bash
```

Then in separate Terminal, run:

```
roscore
```

If you are using Memory Module on the PC other then one with roscore, ROS interfaces require network setup.

For this two variables in Config class (util folder of the Memory Module) should be changed:

- ROS_MASTER_URI - defines an URI of roscore module in the network, e.g. "http://bot.roboy.org:11311/"
- ROS_HOSTNAME - defines the IP address of the machine with rosjava mudule in the network, e.g. "192.168.1.1"

If you running ros in a virtual machine, please configure bridged networking and use the respective IP addresses:

- VMware Fusion
- VMware Workstation
- Parallels
- VirtualBox
- Hyper-V. We don't recommend using this one, but as you like.

Running the Package

After you have entered the proper configuration:

- in the project directory do:

  ```
  mvn clean install
  ```

- navigate to:

  ```
  cd target
  ```

- run the package:

  ```
  java -jar roboy_memory-0.9.0-jar-with-dependencies.jar
  ```

Using Remote

Warning: Be careful while using remote and/or interacting with bot.roboy.org server! You are responsible to keep it functioning properly!

Please, do not crush everything. You would make little kittens very sad.

Development

For further development we recommend using Intellij IDEA IDE. The community edition is available here: Download IDEA.

If you are eligible, we suggest applying for this package containing the full versions of JetBrains software for free.

Using the ROS services

There you can find basic examples on how to access the memory with JSON-formed queries using ROS. For more information, please, refer to *Public Interfaces (ROS)*, *Neo4j Memory Architecture* and *Roboy Communication Standard*.

Available ROS services

The Roboy Memory Module offers the next services in order to work with the memory contents:

- create - creates a node in the Neo4j DB with provided properties and face features (Redis)
- update - adds new relationships between specified nodes or properties to the specified node
- get - retrieves information about the specified node or returns IDs of all nodes which fall into the provided conditions
- remove - removes properties or relationships from the specified node

In order to check available services, in your catkin environment, run:

```
rosservice list
```

You should get the next output:

```
/roboy/cognition/memory/create
/roboy/cognition/memory/cypher
/roboy/cognition/memory/get
/roboy/cognition/memory/remove
/roboy/cognition/memory/update
/rosout/get_loggers
/rosout/set_logger_level
```

Calling the ROS

General syntax for a ROS message:

```
rosservice call /roboy/cognition/memory/--service_name-- "\"---header---\"" "\"---
↪payload---\""
```

Sample Header:

The header (JSON object) consists of a timestamp and the module which is sending the query ('user'): You may try using the next header for your initial experience.

```
{
    'user':'test',
    'datetime':'0'
}
```

Payload Elements:

The payload (JSON object) may comprise several elements such as:

- 'label' specifies the class of node in the knowledge graph
- 'id' of a node is a unique number specified for each node that may be accessed be searched or modified in the knowledge graph
- 'relations' comprise a map of relationship types with an array of node IDs for each of them, providing multiple relationships tracing
- 'properties' = A map of property keys with values

Consider *Roboy Communication Standard* for the correct use use of properties, relationships and labels. Sample payloads as well as the whole structure of the calls are mentioned below.

Create queries

Create a node of the type 'Person' with properties:

```
rosservice call /roboy/cognition/memory/create "\"{
    'user':'vision',
    'datetime':'1234567'
}\"" "\"{
    'type':'node',
    'label':'Person',
    'properties':{
        'name':'Lucas',
        'sex':'male'
    }
}\""
```

On success you will get:

Answer: {'id': x } - //ID of the created node

On error you will get:

Error: {status:"FAIL", message:"error message"}

You can find detailed information in *Public Interfaces (ROS)*

Update queries

Add properties to the node with id 15:

```
rosservice call /roboy/cognition/memory/update "\"{
    'user':'vision',
    'datetime':'1234567'
}\"" "\"{
    'type':'node',
    'id':15,
    'properties':{
        'surname':'Ki',
        'xyz':'abc'
    }
}\""
```

Add relations to the node with id 15:

```
rosservice call /roboy/cognition/memory/update "\"{
    'user':'vision',
    'datetime':'1234567'
}\"" "\"{
    'type':'node',
    'id':15,
    'relations':{
        'LIVE_IN':[28,23],
        'STUDY_AT':[16]
    }
}\""
```

Add properties + relations to the node with id 15:

```
rosservice call /roboy/cognition/memory/update "\"{
    'user':'vision',
    'datetime':'1234567'
}\"" "\"{
    'type':'node',
    'id':15,
    'properties':{
        'surname':'Ki', 'xyz':123
    },
    'relations':{
        'LIVE_IN':[28,23],
        'STUDY_AT':[16]
    }
}\""
```

On success you will get:

Answer: {status:"OK"}

On error you will get:

Error: {status:"FAIL", message:"error message"}

You can find detailed information in *Public Interfaces (ROS)*

Get queries

Get properties and relationships of a node by id:

```
rosservice call /roboy/cognition/memory/get "\"{
    'user':'vision',
    'datetime':'1234567'
}\"" "\"{
    'id':15
}\""
```

Answer::

```
{
    'id': 15,
    'labels': ["person"],
    'properties': {
        "birthdate":"01.01.1970",
        "surname":"ki",
        "sex":"male",
        "name":"lucas"
    },
    'relations': {
        "from":[28],
        "friend_of":[124, 4, 26, 104, 106, 71, 96, 63],
        "member_of":[20], "study_at":[16], "is":[17],
        "has_hobby":[18],
        "live_in":[23, 28]
    }
}
```

Get ids of nodes which have all specified labels, relations and/or properties:

```
rosservice call /roboy/cognition/memory/get "\"{
    'user':'vision',
    'datetime':'1234567'
}\"" "\"{
    'label':'Person',
    'relations':{
        'FRIEND_OF':[15]
    },
    'properties':{
        'name':'Laura'
    }
}\""
```

On success you will get:

Answer: {'id':[x]} - an array with all fitting IDs

On error you will get:

Error: {status:"FAIL", message:"error message"}

You can find detailed information in *Public Interfaces (ROS)*

Remove queries

Warning: Please, do not try running **remove** queries without considering significant risks. Be responsible!

Remove properties of node 15:

```
rosservice call /roboy/cognition/memory/remove "\"{
    'user':'vision',
    'datetime':'1234567'
}\"" "\"{
    'type':'node',
    'id':15,
    'properties':['birthdate','surname']
}\""
```

Remove relations of node 15:

```
rosservice call /roboy/cognition/memory/remove "\"{
    'user':'vision','datetime':'1234567'
}\"" "\"{
    'type':'node',
    'id':15,
    'relations':{
        'LIVE_IN':[28,23],
        'STUDY_AT':[16]
    }
}\""
```

Remove properties and relations of node 15:

```
rosservice call /roboy/cognition/memory/remove "\"{
    'user':'vision',
    'datetime':'1234567'
}\"" "\"{
    'type':'node',
    'id':15,
    'properties':['birthdate','surname'],
    'relations':{
        'LIVE_IN':[23]
    }
}\""
```

On success you will get:

Answer: {status:"OK"}

On error you will get:

Error: {status:"FAIL", message:"error message"}

You can find detailed information in *Public Interfaces (ROS)*

Troubleshooting

Possible Common Exceptions

No ROS master connection:

```
org.ros.internal.node.client.Registrar callMaster
SEVERE: Exception caught while communicating with master.
java.lang.RuntimeException: java.net.ConnectException: Host is down
```

Check if the roscore master PC is connected to the network or master URI in configuration is stated properly.

No roscore running on ROS master:

```
org.ros.internal.node.client.Registrar callMaster
SEVERE: Exception caught while communicating with master.
java.lang.RuntimeException: java.net.ConnectException: Connection refused
```

Check if roscore is up on the master PC or master URI in configuration is stated properly.

Host PC is not reachable from ROS master:

```
ERROR: Unable to communicate with service [/roboy/cognition/memory/get],
address [rosrpc://127.0.0.1:51734/]
```

Check if hostname for ROS publisher (current PC) in configuration is stated properly.

No service is running on host from ROS master:

```
ERROR: transport error completing service call:
unable to receive data from sender, check sender's logs for details.
```

Check if the package is running and services were successfully published (on current PC).

No Neo4j connection:

```
Exception in thread "pool-1-thread-16" org.neo4j.driver.v1.exceptions.
↪ServiceUnavailableException:
Unable to connect to 127.0.0.1:7687, ensure the database is running and that there is␣
↪a working network connection to it.
```

Check if Neo4j is up and the Neo4j address in configuration is stated properly.

Neo4j credentials are incorrect:

```
Exception in thread "pool-1-thread-16" org.neo4j.driver.v1.exceptions.
↪AuthenticationException:
The client is unauthorized due to authentication failure.
```

Check if Neo4j credentials in configuration are stated properly.

No Redis connection:

```
Exception in thread "pool-1-thread-33" redis.clients.jedis.exceptions.
↪JedisConnectionException:
java.net.UnknownHostException: 127.0.0.1
```

Check if Redis is up and the Redis address in configuration is stated properly.

Redis credentials are incorrect:

```
Exception in thread "pool-1-thread-16" redis.clients.jedis.exceptions.
↪JedisDataException:
ERR invalid password
```

Check if Redis credentials in configuration are stated properly.

Missing parenthesis:

```
Exception in thread "pool-1-thread-13" com.google.gson.JsonSyntaxException:
java.io.EOFException: End of input at line 1 column 38 path $.datetime
```

Check JSON "{ }" parenthesis in query.

JSON index is present, but value is not:

```
Exception in thread "pool-1-thread-24" com.google.gson.JsonSyntaxException:
com.google.gson.stream.MalformedJsonException: Expected value at line 1 column 33␣
↪path $.properties
```

Check if any value in JSON query is missing.

JSON query is formed incorrectly:

```
Exception in thread "pool-1-thread-18" com.google.gson.JsonSyntaxException:
com.google.gson.stream.MalformedJsonException: Unterminated string at line 1 column 9␣
↪path $.
```

Check if JSON is formed properly: quotes, parenthesis. Refer to *Roboy Communication Standard*

Primitives are initialized with complex types in JSON query:

```
Exception in thread "pool-1-thread-14" com.google.gson.JsonSyntaxException:
java.lang.IllegalStateException: Expected an int but was BEGIN_ARRAY at line 1 column␣
↪8 path $.id

Exception in thread "pool-1-thread-22" com.google.gson.JsonSyntaxException:
java.lang.IllegalStateException: Expected a string but was BEGIN_ARRAY at line 1␣
↪column 11 path $.label

Exception in thread "pool-1-thread-22" com.google.gson.JsonSyntaxException:
java.lang.IllegalStateException: Expected a string but was BEGIN_OBJECT at line 1␣
↪column 11 path $.label
```

Check if the JSON query is type valid: JSON array instead of object is recieved. Change the respective values. Refer to *Roboy Communication Standard*.

Complex types are initialized with primitive types in JSON query:

```
Exception in thread "pool-1-thread-21" com.google.gson.JsonSyntaxException:
java.lang.IllegalStateException: Expected BEGIN_ARRAY but was STRING at line 1 column␣
↪35 path $.properties[0]
```

Check if the JSON query is type valid: primitive objects instead of JSON arrays are recieved. Change the respective values. Refer to *Roboy Communication Standard*.

Wrong complex type is applied on initialization in JSON query:

```
Exception in thread "pool-1-thread-22" com.google.gson.JsonSyntaxException:
java.lang.IllegalStateException: Expected BEGIN_ARRAY but was BEGIN_OBJECT at line 1␣
↪column 11 path $.label

Exception in thread "pool-1-thread-22" com.google.gson.JsonSyntaxException:
java.lang.IllegalStateException: Expected BEGIN_OBJECT but was BEGIN_ARRAY at line 1␣
↪column 11 path $.label
```

Check if the JSON query is type valid: JSON object instead of JSON array and vice versa are received. Change the respective values. Refer to *Roboy Communication Standard*.

Context

The Memory Module receives input from other Cognition modules in form of ROS messages containing RCS payload which is then parsed internally. RCS payload contains valid request, otherwise exeption would be raised and Memory Module would answer with "FAIL" and error message.

The main output of the Memory Module is either a single piece of data (JSON object) or set of **ID**s.

The context of Roboy Memory Module illustrated in the following diagram:

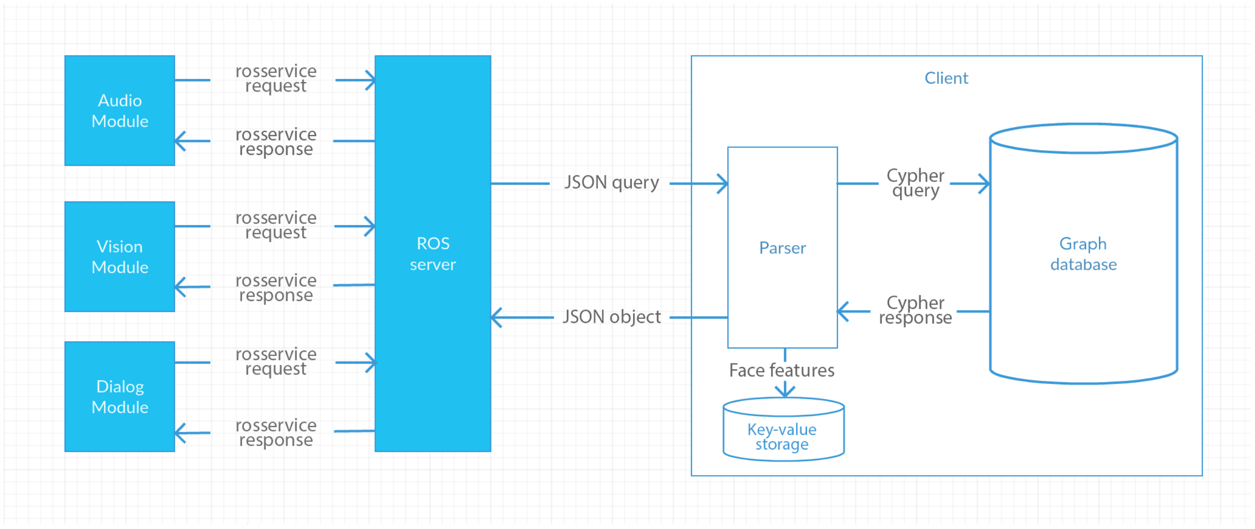


Fig. 3.1: UML System Context
UML-type context diagram - shows the birds eye view of the system (black box) described by this architecture within the ecosystem it is to be placed in. Shows orbit level interfaces on the user interaction and component scope.

Conventions

We follow the coding guidelines:

Table 3.1: Coding Guidelines

Language	Guideline	Tools
Java	https://google.github.io/styleguide/javaguide.html	
Cypher	https://neo4j.com/developer/cypher-query-language/	
Redis	https://redis.io/commands	

Architecture Constraints

Technical Constraints / Runtime Interface Requirements

Table 3.2: Operating System Constraints

Constraint Name	Description
Ubuntu => 16.04	Neo4j and ros are much more stable and easier to support on Linux and Ubuntu is the OS of Roboy as well

Table 3.3: Programming Constraints

Constraint Name	Description
IntelliJ IDEA	There were difficulties with importing the project to NetBeans and Eclipse
rosjava	Due to using both Java and ros
Java => 1.8.0	Reasonably recent and stable Java release
Neo4j => 3.2.1	Stable and tested in production

Public Interfaces (ROS)

Interfaces to other modules are realized through ROS (rosjava). Currently 5 interfaces (ROS services) have been designed for communication with Memory Module.

ROS Services

All calls are complaint to this general form:

```
rosservice call /roboy/cognition/memory/---service_name--- "\"---header---\"" "\"---
↪payload---\""
```

- **create service**: Service called to perform a query writing data into Neo4j database.:

```
# argument: String header String payload
# returns: String answer

rosservice call /roboy/cognition/memory/create
```

- **get service:** Service called to perform a query reading data from Neo4j database.:

```
## argument: String header String payload
# returns: String answer

rosservice call /roboy/cognition/memory/get
```

- **update service:** Service called to perform a query altering data in Neo4j database.:

```
## argument: String header String payload
# returns: String answer

rosservice call /roboy/cognition/memory/update
```

- **remove service:** Service called to perform a query deleting data from Neo4j database.:

```
## argument: String header String payload
# returns: String answer

rosservice call /roboy/cognition/memory/remove
```

- **cypher service:** Service called to perform any Cypher query in Neo4j database.:

```
## argument: String header String payload
# returns: String answer

rosservice call /roboy/cognition/memory/cypher
```

For the first 4 services the payload has to be defined according to *Roboy Communication Standard*.

Payload Elements:

- 'label' specifies the class of node in the knowledge graph
- 'id' of a node is a unique number specified for each node that may be accessed be searched or modified in the knowledge graph
- 'relations' comprise a map of relationship types with an array of node ids for each of them, providing multiple relationships tracing

- ‘properties’ = A map of property keys with values

Each of this element is peculiar to respective service payload.

The Cypher service uses a well-formed query in Cypher as the payload, see *Cypher Examples*.

Responses

Create query provides the following responses.

Success::

```
{
    'id': x
}
```

Failure:

- some properties are not specified properly:

```
{
    status:"FAIL",
    message:"no properties"
}
```

- when creating a node, the name property is obligatory, name is missing:

```
{
    status:"FAIL",
    message:"no name specified in properties : name required"
}
```

- trying to create a node with a non-existing label, see *Neo4j Memory Architecture*:

```
{
    status:"FAIL",
    message:"Label 'Xyz' doesn't exist in the DB"
}
```

Update query provides the following responses.

Success::

```
{
    status:"OK"
}
```

Failure:

- trying to create a relationship with a non-existing type, see *Neo4j Memory Architecture*:

```
{
    status:"FAIL",
    message:"The relationship type 'XYZ' doesn't exist in the DB"
}
```

Get query provides the following responses.

Success:

- getting by ID:

```
{
    'id': 15,
    'labels': ["person"],
    'properties': {
        "birthdate":"01.01.1970",
        "surname":"ki",
        "sex":"male",
        "name":"lucas"
    },
    'relations': {
        "from":[28],
        "friend_of":[124, 4, 26, 104, 106, 71, 96, 63],
        "member_of":[20], "study_at":[16], "is":[17],
        "has_hobby":[18],
        "live_in":[23, 28]
    }
}
```

- getting IDs:

```
{
    'id':[x, y]
}
```

Remove query provides the following responses.

Success::

```
{
    status:"OK"
}
```

User Interfaces

There is a GUI for development purposes provided by Neo4j. In order to invoke the GUI, a user has to run a Neo4j instance, open a browser and go to:

```
http://localhost:7474
```

or if using a remote Neo4j instance:

```
http://85.10.197.57:7474
```

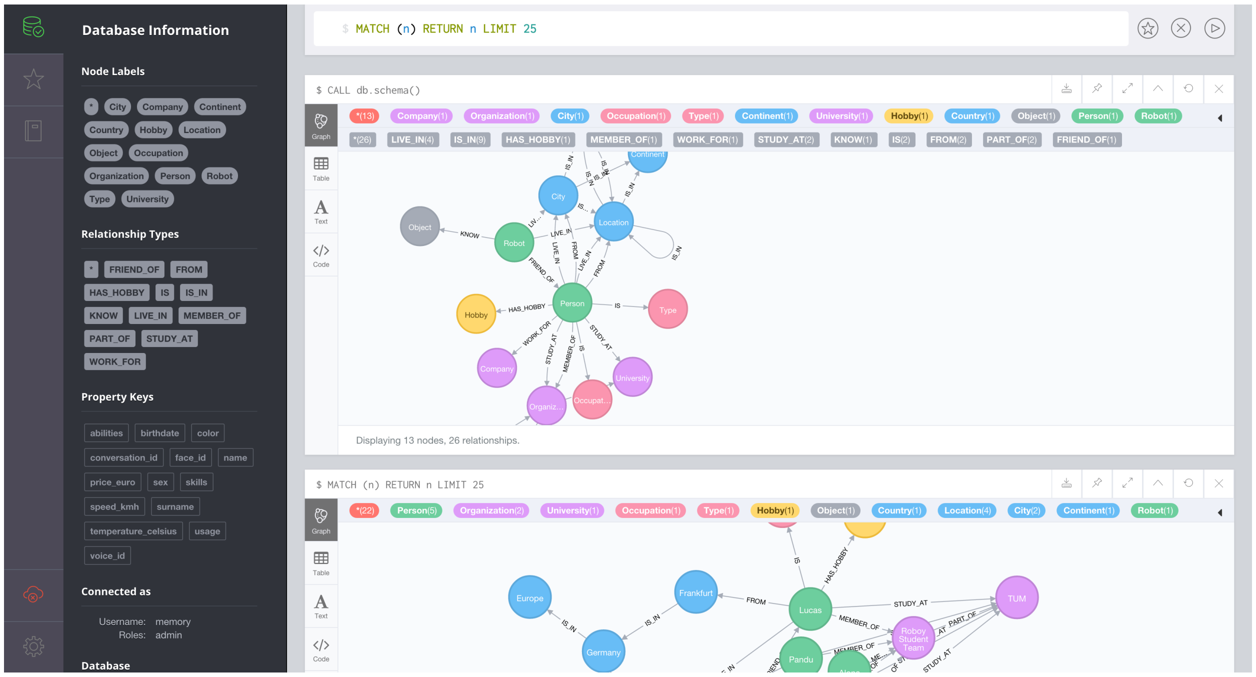
which is the Roboy server.

All other parts of the module are provided without GUI and offer interaction on a command line level.

Neo4j Memory Architecture

Architecture of the Neo4j database in remote. Current version: 1.1.1.

Visualization of a DB scheme.



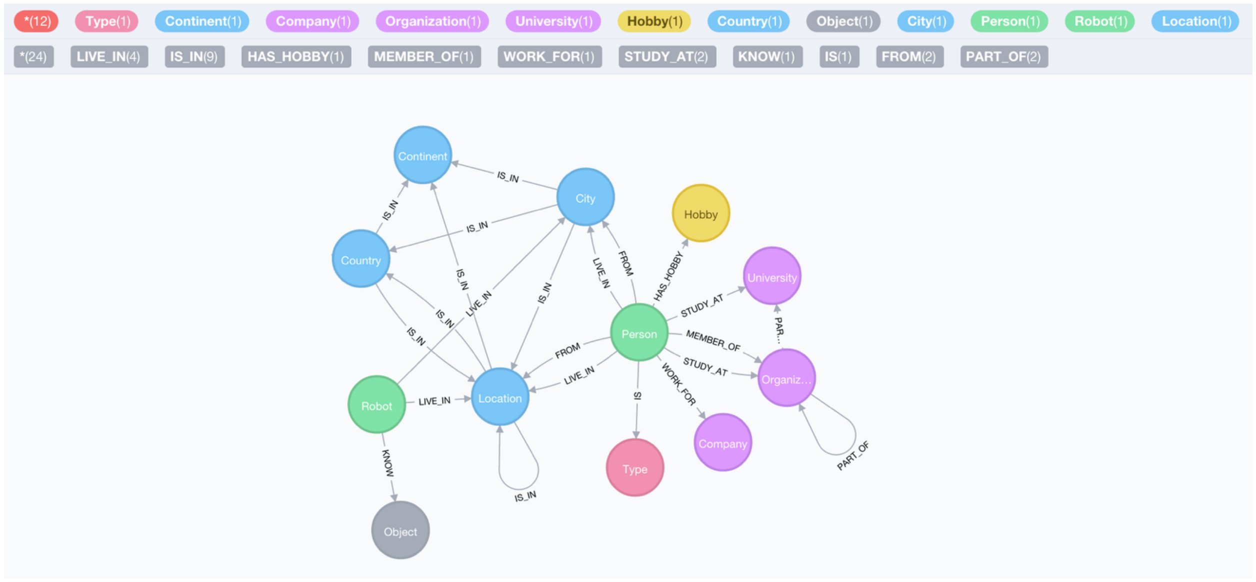
```
$ MATCH (n) RETURN n LIMIT 25
```

```
$ CALL db.schema()
```

Displaying 13 nodes, 26 relationships.

```
$ MATCH (n) RETURN n LIMIT 25
```

Database Information
Node Labels
* City Company Continent Country Hobby Location Object Occupation Organization Person Robot Type University
Relationship Types
* FRIEND_OF FROM HAS_HOBBY IS IS_IN KNOW LIVE_IN MEMBER_OF PART_OF STUDY_AT WORK_FOR
Property Keys
abilities birthdate color conversation_id face_id name price_euro sex skills speed_kmh surname temperature_celsius usage voice_id
Connected as
Username: memory
Roles: admin
Database
Graph
Table
Text
Code
*(13) Company(1) Organization(1) City(1) Occupation(1) Type(1) Continent(1) University(1) Hobby(1) Country(1) Object(1) Person(1) Robot(1)
*(26) LIVE_IN(4) IS_IN(9) HAS_HOBBY(1) MEMBER_OF(1) WORK_FOR(1) STUDY_AT(2) KNOW(1) IS(2) FROM(2) PART_OF(2) FRIEND_OF(1)
*(22) Person(5) Organization(2) University(1) Occupation(1) Type(1) Hobby(1) Object(1) Country(1) Location(4) City(2) Continent(1) Robot(1)
Europe
Germany
Frankfurt
Lucas
Pandu
Roboy Student Team
TUM


*(12) Type(1) Continent(1) Company(1) Organization(1) University(1) Hobby(1) Country(1) Object(1) City(1) Person(1) Robot(1) Location(1)
*(24) LIVE_IN(4) IS_IN(9) HAS_HOBBY(1) MEMBER_OF(1) WORK_FOR(1) STUDY_AT(2) KNOW(1) IS(1) FROM(2) PART_OF(2)
Continent
City
Hobby
Country
University
Person
Organiz...
Robot
Location
Company
Type
Object
IS_IN
LIVE_IN
FROM
HAS_HOBBY
STUDY_AT
MEMBER_OF
PAR...
WORK_FOR
IS
KNOW
PART_OF

Versioning of KR is performed by implementing architecture proposals and evaluating them, upon evaluation the version is fixed and then new proposals are collected. Adding nodes means major ver. X, adding relations is minor ver. Y, adding properties is patch ver. Z: ver. X.Y.Z.

Node Classes (Labels)

- Person
- Robot
- Organization
    1. Company
    2. University
- Location
    1. City
    2. Country
    3. Continent
- Hobby
- Type
    1. Occupation
- Object (which Roboy can detect/interact with)

Edge Classes

(Person, Robot : Person, Robot)

- FRIEND_OF

(Person, Robot : Location)

- LIVE_IN
- FROM

(Person : Organization)

- WORK_FOR
- STUDY_AT
- MEMBER_OF

(Person, Robot : Hobby)

- HAS_HOBBY

(Person, Robot : Object)

- KNOW

(Object, Robot, Person, Organization : Type)

- IS

(Organization, Robot : Organization)

- PART_OF

(Organization, Location : Location)

- IS_IN

Property Keys

General

Describes non-specific prameters for any node

- name [string]
- id [int]

Person

Describes prameters specific to a person

- surname [string]
- birthdate [String]
- sex [string]
- face_id (facial features) [int]: reference to a face representation.
- voice_id (voice signature) [int]: reference to a voice signature.
- conversation_id (Topic (scope) of the last conversation) [int]: reference to a topic marker for the last conversation. It would refer to a word or summary def by Dialog to recall the previous conversation with a person.

Roboy

Describes prameters specific to Roboy

- birthdate [string]
- abilities [list of strings]
- skills [list of strings]

Object

Describes prameters specific to objects

- color [string]
- speed [int]
- price [int]
- temperature [int]
- usage [list of strings]

Roboy Communication Standard

Roboy Communication Standard is a proposal on decorating standard ros messages with JSON-like payload.

Create Queries Payload Message

Creating a node:

```
{
    'label': 'some_label',
    'faceVector': [float, ..., float] // Under comsideration, OPTIONAL
    'properties': {
        'prop_a': 'value_a',
        'prop_b': 'value_b'
    }
}
```

This query requests creating node with **label** - some_label, **prop_a** having **value_a** and **prop_b** - **value_b**. The **faceVector** contains face features for the node with specified label **Person** (applicable only to nodes of this label).

There the label may be:

- Person
- Robot
- Organization
  1. Company
  2. University
- Location
  1. City
  2. Country
  3. Continent
- Hobby
- Type
- Occupation
- Object (which Roboy can detect/interact with)

Properties other than 'name' are not required on the creation and may be omitted. Later the node's properties may be updated by an update query. The query returns the **ID** of the created node on success. The **faceVector** is fed into Redis if present. The named properties and allowed values may be found in *Neo4j Memory Architecture*.

Update Queries Payload Message

Updating a node

```
{
    'id': 1, //REQUIRED, contains node id

    'relations':  {
        'rel_a': [2, 3],
        'rel_b': [3]
    }

    'properties': {
        'prop_a': 'value_a',
        'prop_b': 'value_b'
    }
}
```

This query requests updating node with **ID** - 1. This query requests creating relations between two nodes, where the relations are e.g. **rel_a**, the number denotes the **ID** of the node to where the relations is following from the current node. This query requests creating (changing) properties of the node, where the properties may be e.g. **prop_a** with value **value_a**.

Warning: You should be aware of the node label.

The query returns the **OK** message on success. The named properties and allowed values may be found in *Neo4j Memory Architecture*.

Get Queries Payload Message

Get nodes IDs

```
{
    'label': 'some_label',

    'relations': {
        'rel_a': [2],
        'rel_b': [3]
    },

    'properties': {
        'prop_a': 'value'
    }
}
```

This query requests getting all nodes which have node label - **some_label**, have relationship **rel_a** with the node having **ID** 2 and **rel_b** with the node of **ID** 3, as well as having **prop_a** equal to **value**. The query returns an array of node IDs on success (may be an empty array if no such nodes exist). The allowed relation types for each pair of nodes and named properties of nodes may be found in *Neo4j Memory Architecture*.

Get node by ID

```
{
    'id': 1
}
```

This query requests getting all information about a node with respective **ID**. The query returns a JSON containing all information about the node on success (may be an empty string if no such node exist).

Warning: You should be aware of the node label.

The respective information about what could be returned may be found in *Neo4j Memory Architecture*.

Remove Queries Payload Message

Remove properties and relations of the nodes

```
{
    'id': 1,

    'relations': {
        'rel_a': [2],
        'rel_b': [3]
    },

    'properties': {
        'prop_a'
    }
}
```

This query requests removing all respective properties and relations with regard to the node with **ID** = 1: relationships **rel_a** with the node having **ID** = 2 and **rel_b** with the node having **ID** = 3, as well as property **prop_a**.

Warning: You should be aware of the node label.

The query returns the **OK** message on success. The named properties and allowed values may be found in *Neo4j Memory Architecture*.

Cypher Examples

Cypher is a declarative graph query language that allows for expressive and efficient querying and updating of the graph store. Cypher is a relatively simple but still very powerful language. Very complicated database queries can easily be expressed through Cypher. This allows you to focus on your domain instead of getting lost in database access.

Useful Cypher queries related to actual Knowledge Representation (developer)

Create a „location"-node:

```
CREATE (n:Location {name: "Munich"})
```

Add a 2nd Lable (Organization) to a Node:

```
match (n:Company)
set n:Organization
return n
```

Create a relationship

if relationship type is not existing yet:

```
MATCH (a:Person),(b:City) WHERE a.name = 'Lucas' AND b.name = 'Frankfurt' CREATE (a)-
↪[r:FROM]->(b) RETURN r
```

if relationship type is existing::

```
MATCH (a:Country),(b:Continent) WHERE a.name = 'Germany' AND b.name = 'Europe' Merge␣
↪(a)-[r:IS_IN]->(b) RETURN r
```

Delete

all „location"-Nodes:

```
MATCH (n:Location) DETACH Delete n
```

a specific Node by ID:

```
MATCH (n:Person) where ID(n)=13 DELETE n
```

all relationships from Roboy:

```
MATCH (n:Robot { name: 'Roboy' })-[r:FRIEND_OF]->() DELETE r
```

Add Properties:

```
Match (n:Object {name: 'Ball'})
Set n.color = 'red'
Set n.price_euro = 15
Set n.usage = ["playing", "trowing", "rolling"]
Return n
```

Show

all nodes with relationships:

```
MATCH (n) RETURN n;
```

the database scheme:

```
CALL db.schema()
```

API

class **`Answer`**

Answer wrapper.

Outputs OK or error messages to ROS.

Public Static Functions

`static String org.roboy.memory.util.Answer.ok()`

Answer for ROS if no errors were detected.

Return JSON object {status:"OK"} to ROS

`static String org.roboy.memory.util.Answer.error(String message)`

Answer for ROS if an error occurred.

Return JSON object containing status and message

Parameters

- `message`: contains the error message according to the obstacle approached

Private Static Attributes

`Logger org.roboy.memory.util.Answer.logger` = Logger.getLogger(Answer.class.toString())

class **`Config`**

Configuration for ROS, Neo4J and Redis Server connectivity.

Public Static Attributes

`final String org.roboy.memory.util.Config.ROS_MASTER_URI` = "http://127.0.0.1:11311/"

ROS Configuration.

IP adress of the PC with roscore

`final String org.roboy.memory.util.Config.ROS_HOSTNAME` = "127.0.0.1"

IP address of the current PC in the network.

`final String org.roboy.memory.util.Config.NEO4J_ADDRESS` = "bolt://127.0.0.1:7687"

Neo4J Configuration.

Neo4j DB location

`final String org.roboy.memory.util.Config.NEO4J_USERNAME` = "***"

Neo4j instance username.

`final String org.roboy.memory.util.Config.NEO4J_PASSWORD` = "***"

Neo4j instance password.

`final String org.roboy.memory.util.Config.REDIS_URI` = "redis://127.0.0.1:6379/0"

Redis Configuration.

Redis storage location

`final String org.roboy.memory.util.Config.REDIS_PASSWORD` = "***"

Redis storage instance password.

`final String [] org.roboy.memory.util.Config.LABEL_VALUES` = new String[] { "Person","Robot","Compa

KR Entries Configuration.

Available label types

`final String [] org.roboy.memory.util.Config.RELATION_VALUES` = new String[] { "FRIEND_OF","LIV

Available reltionship types.

class **`Create`**

Data model for JSON parser.

Creates objects, that contain the elements of the *Create* queries.

Public Functions

`String org.roboy.memory.models.Create.getLabel()`

`String org.roboy.memory.models.Create.getType()`

`Map<String, String> org.roboy.memory.models.Create.getProperties()`

`String [] org.roboy.memory.models.Create.getFace()`

Private Members

`String org.roboy.memory.models.Create.type`
Currently only used to specify the type "node".

`String [] org.roboy.memory.models.Create.label`

`String org.roboy.memory.models.Create.label`
Specifies the type of node that shall be created, like "Person".

`String [] org.roboy.memory.models.Create.faceVector`
JSON array containing facial features from vision module.

`Map<String, String> org.roboy.memory.models.Create.properties`
Contains the node properties.

class **`Get`**
Data model for JSON parser.

Creates objects, that contain the elements of the *Get* queries.

Public Functions

`int org.roboy.memory.models.Get.getId()`

`String org.roboy.memory.models.Get.getLabel()`

`Map<String, String[]> org.roboy.memory.models.Get.getRelations()`

`Map<String, String> org.roboy.memory.models.Get.getProperties()`

Private Members

`String org.roboy.memory.models.Get.label`
Specifies the type of node that shall be searched, like "Person".

`int org.roboy.memory.models.Get.id`
The id of a node that shall be searched.

`Map<String, String[]> org.roboy.memory.models.Get.relations`
Contains the relationship type as key and an array of node IDs as value.

`Map<String, String> org.roboy.memory.models.Get.properties`
Contains the node properties.

class **`Header`**
Data model for JSON parser.

Creates objects, that contain the elements of the *Header*.

Public Functions

`LocalDateTime org.roboy.memory.models.Header.getDateTime()`

`String org.roboy.memory.models.Header.getUser()`

Private Members

`String org.roboy.memory.models.Header.user`
Contains the module which is sending the query, for example "vision".

`String org.roboy.memory.models.Header.datetime`
Contains a timestamp in seconds since 1.1.1970.

class `Main`

Public Static Functions

`static void org.roboy.memory.Main.main(String[] args)`

class `Neo4j`
Contains the methods for running GET, CREATE, UPDATE, REMOVE and Cypher queries.

Talks to the *Neo4j* and Redis databases. Handles the result retrieved from *Neo4j*.

Inherits from AutoCloseable

Public Functions

`void org.roboy.memory.util.Neo4j.close()`

Public Static Functions

`static Driver org.roboy.memory.util.Neo4j.getInstance()`
Singleton for the *Neo4j* class.

Return Neo4J Driver instance if the object of *Neo4j* class is initialized

`static Value org.roboy.memory.util.Neo4j.parameters(Object... keysAndValues)`
Wrapper for the *Neo4j* query parameters.

Return Set of keys and values for parameters

`static String org.roboy.memory.util.Neo4j.run(String query)`
Method to channel a plain Cypher query to *Neo4j*.

Return plain response from *Neo4j*

Parameters

- `query`: formed in Cypher

`static String org.roboy.memory.util.Neo4j.createNode(String label, String[] faceVector`
Method accepting JSON Create queries.

Return result obtained by createNode method

Parameters

- `label`: is denoting a type of the node to be created
- `faceVector`: contains face features for a node of label "Person". OPTIONAL
- `properties`: is a dictionary containing properties of the node

`static String org.roboy.memory.util.Neo4j.updateNode(int id, Map< String, String[]> re`

Method accepting JSON Update queries.

Return result obtained by update method

Parameters

- `id`: is a unique pointer to the node in *Neo4j* DB
- `relations`: is a dictionary containing relationships of the node with other nodes
- `properties`: is a dictionary containing properties of the node

`static String org.roboy.memory.util.Neo4j.getNodeById(int id)`

Method accepting JSON Get by ID queries.

Return result obtained by matchNodeById method

Parameters

- `id`: is a unique pointer to the node in *Neo4j* DB

`static String org.roboy.memory.util.Neo4j.getNode(String label, Map< String, String[]>`

Method accepting JSON Get IDs of nodes queries.

Return result obtained by matchNode method

Parameters

- `label`: is denoting a type of the nodes to be included
- `relations`: is a dictionary containing relationships of the nodes with other nodes
- `properties`: is a dictionary containing properties of the nodes

`static String org.roboy.memory.util.Neo4j.remove(int id, Map< String, String[]> relati`

Method accepting JSON Remove queries.

Return result obtained by removeRelsProps method

Parameters

- `id`: of the node which relations and properties need to be removed
- `relations`: is a dictionary containing relationships of the node with other nodes
- `properties`: is a dictionary containing properties of the node

Private Functions

`org.roboy.memory.util.Neo4j.Neo4j()`

`Driver org.roboy.memory.util.Neo4j.getDriver()`

Getter for the *Neo4j* driver instance.

Return Neo4J Driver instance

Private Static Functions

static String org.roboy.memory.util.Neo4j.createNode(Session session, Map< String, Str

Method processing JSON Create queries.

Return ID of the node that was created in *Neo4j* DB

Parameters

- `session`: is a session handler for transaction handling to query *Neo4j* DB
- `label`: is denoting a type of the node to be created
- `faceVector`: contains face features for a node of label "Person". OPTIONAL
- `properties`: is a dictionary containing properties of the node

static String org.roboy.memory.util.Neo4j.update(Transaction tx, int id, Map< String,

Method processing JSON Update queries.

Return response from *Neo4j* upon updating the node

Parameters

- `tx`: is a transaction handler to query *Neo4j* DB
- `id`: is a unique pointer to the node in *Neo4j* DB
- `relations`: is a dictionary containing relationships of the node with other nodes
- `properties`: is a dictionary containing properties of the node

static String org.roboy.memory.util.Neo4j.matchNodeById(Transaction tx, int id)

Method processing JSON Get by ID queries.

Return a JSON object containing node labels, properties and relationships

Parameters

- `tx`: is a transaction handler to query *Neo4j* DB
- `id`: is a unique pointer to the node in *Neo4j* DB

static String org.roboy.memory.util.Neo4j.matchNode(Transaction tx, String label, Map<

Method processing JSON Get IDs of nodes queries.

Return JSON array of nodes' IDs

Parameters

- `tx`: is a transaction handler to query *Neo4j* DB
- `label`: is denoting a type of the nodes to be included
- `relations`: is a dictionary containing relationships of the nodes with other nodes
- `properties`: is a dictionary containing properties of the nodes

static String org.roboy.memory.util.Neo4j.removeRelsProps(Transaction tx, int id, Map<

Method processing JSON Remove queries.

Return response from *Neo4j* upon removing the specified relations and properties

Parameters

- `tx`: is a transaction handler to query *Neo4j* DB

- `id`: of the node which relations and properties need to be removed
- `relations`: is a dictionary containing relationships of the node with other nodes
- `properties`: is a dictionary containing properties of the node

Private Static Attributes

`Neo4j org.roboy.memory.util.Neo4j._instance`
An instance of the class.

`Driver org.roboy.memory.util.Neo4j._driver`
An instance of *Neo4j* driver.

`Jedis org.roboy.memory.util.Neo4j.jedis`
An instance of Jedis for Redis handling.

`Gson org.roboy.memory.util.Neo4j.parser` = new Gson()
An instance of Gson parser for creating JSON response.

`Logger org.roboy.memory.util.Neo4j.logger` = Logger.getLogger(Neo4j.class.toString())
An instance of the logger.

class `Remove`
Data model for JSON parser.

Creates objects, that contain the elements of the *Remove* queries.

Public Functions

`int org.roboy.memory.models.Remove.getId()`

`String [] org.roboy.memory.models.Remove.getProperties()`

`Map<String, String[]> org.roboy.memory.models.Remove.getRelations()`

Private Members

`int org.roboy.memory.models.Remove.id`
The id of a node that shall be modified.

`String org.roboy.memory.models.Remove.type`
Currently only used to specify the type "node".

`String org.roboy.memory.models.Remove.label`
Specifies the type of node that shall be removes, like "Person".

`Map<String, String[]> org.roboy.memory.models.Remove.relations`
Contains the relationship type as key and an array of node IDs as value.

`String [] org.roboy.memory.models.Remove.properties`
Contains the node properties.

class `RosNode`
ROS Service for saving data object to DB.

Data is received as JSON object. JSON object is parsed using Parser and saved to neo4j.

Inherits from AbstractNodeMain

Public Functions

GraphName org.roboy.memory.ros.RosNode.getDefaultNodeName()

void org.roboy.memory.ros.RosNode.onStart(ConnectedNode connectedNode)
Initialising the ROS services and setting ROS services URIs.

Parameters

- `connectedNode`: is the ROS node carrying the services.

Package Static Functions

static void org.roboy.memory.ros.RosNode.register(NodeConfiguration nodeConfiguration,
Registers the ROS node.

Parameters

- `nodeConfiguration`: is the ROS node configurator
- `nodeMainExecutor`: is the ROS node executor

Private Static Attributes

String org.roboy.memory.ros.RosNode.name = "/roboy/cognition/memory"
URI for the ROS node.

class RosRun
This server is responsible for starting ros services.

Public Functions

org.roboy.memory.ros.RosRun.RosRun()
Constructor.

Initializes the ROS node.

void org.roboy.memory.ros.RosRun.start()
Registers the ROS node with services in the network.

void org.roboy.memory.ros.RosRun.stop()
Shutdowns the ROS node and terminates the services.

Private Members

NodeMainExecutor org.roboy.memory.ros.RosRun.nodeMainExecutor
ROS executor.

NodeConfiguration org.roboy.memory.ros.RosRun.nodeConfiguration
ROS node configurator.

class ServiceLogic
Contains service handlers to talk with ROS.

They parse the header and payload and check for invalid elements in the query. Then the functions to construct the cypher queries are excecuted and the answer returned.

Package Static Attributes

if (create.getFace() != null) { System.out.println("FaceVector: " + create.getFace().toString()); } if (create.getProperties() == null) { response.setAnswer(error("no properties")); return; } else if (!create.getProperties().containsKey("name")){ response.setAnswer(error("no name specified in properties : name required")); return; } else if (create.getLabel() != null && !labels.contains(create.getLabel().substring(0,1).toUpperCase() + create.getLabel().substring(1).toLowerCase())) { response.setAnswer(error("Label '" + create.getLabel() + "' doesn't exist in the DB")); return; } else { response.setAnswer(*Neo4j.createNode*(create.getLabel(), create.getFace(), create.getProperties())); } }] Create Service Handler. Parses the header and payload into a create object with Gson and checks for invalid elements in the query. Calls createNode() method to query Neo4j and the answer is returned.

if(update.getRelations() != null) { for (String rel : update.getRelations().keySet()) { if (!relations.contains(rel.toUpperCase())) { response.setAnswer(error("The relationship type '" + rel + "' doesn't exist in the DB")); return; } } } *Neo4j.updateNode*(update.getId(), update.getRelations(), update.getProperties()); response.setAnswer(ok()); }] Update Service Handler. Parses the header and payload into an update object with Gson and checks for invalid relationship types in the query. Calls updateNode() method to query Neo4j and the answer is returned.

if (get.getId() != 0) { response.setAnswer(*Neo4j.getNodeById*(get.getId())); } else { response.setAnswer(*Neo4j.getNode*(get.getLabel(), get.getRelations(), get.getProperties())); } }] Get Service Handler. Parses the header and payload into a get object with Gson and checks whether node IDs or information about a node is queried. Calls getNodeById() or getNode() methods to query Neo4j and the answer is returned.

`ServiceResponseBuilder<DataQueryRequest, DataQueryResponse> org.roboy.memory.ros.Servi`
Cypher Service Handler.

Directly runs a plain Cypher query which is contained in the payload and returns the response.

Neo4j.remove(remove.getId(), remove.getRelations(), remove.getProperties()); response.setAnswer(ok()); }] Remove Service Handler. Parses the header and payload into a remove object. Calls remove() method to query Neo4j and the answer is returned.

Private Static Attributes

`Logger org.roboy.memory.ros.ServiceLogic.logger` = Logger.getLogger(ServiceLogic.class.toString())
Logger.

`Gson org.roboy.memory.ros.ServiceLogic.parser` = new Gson()
Parses the JSON elements of the header and payload.

`HashSet<String> org.roboy.memory.ros.ServiceLogic.labels` = new HashSet<String>(Arrays.asList(LABI
Contains available label types.

`HashSet<String> org.roboy.memory.ros.ServiceLogic.relations` = new HashSet<String>(Arrays.asList(R
Contains available relationship types.

class `Update`
Data model for JSON parser.

Creates objects, that contain the elements of the *Update* queries.

Public Functions

int org.roboy.memory.models.Update.getId()

Map<String, String> org.roboy.memory.models.Update.getProperties()

Map<String, String[]> org.roboy.memory.models.Update.getRelations()

Private Members

int org.roboy.memory.models.Update.id
The id of a node that shall be modified.

String org.roboy.memory.models.Update.type
Currently only used to specify the type “node”.

String org.roboy.memory.models.Update.label
Specifies the type of node that shall be updated, like “Person”.

Map<String, String[]> org.roboy.memory.models.Update.relations
Contains the relationship type as key and an array of node IDs as value.

Map<String, String> org.roboy.memory.models.Update.properties
Contains the node properties.

namespace util

namespace org

namespace v1

namespace roboy

namespace memory

namespace models

namespace ros

namespace util

namespace Answer

namespace Config

namespace roboy_communication_cognition

file **Main.java**

file **Create.java**

file **Get.java**

file **Header.java**

file **Remove.java**

file **Update.java**

file **RosNode.java**

file **RosRun.java**

file **ServiceLogic.java**

file **Answer.java**

file **`Config.java`**

file **`Neo4j.java`**

dir **`/home/docs/checkouts/readthedocs.org/user_builds/roboy-memory/checkouts/docs/src/org/rob`**

dir **`/home/docs/checkouts/readthedocs.org/user_builds/roboy-memory/checkouts/docs/src/org/rob`**

dir **`/home/docs/checkouts/readthedocs.org/user_builds/roboy-memory/checkouts/docs/src/org`**

dir **`/home/docs/checkouts/readthedocs.org/user_builds/roboy-memory/checkouts/docs/src/org/rob`**

dir **`/home/docs/checkouts/readthedocs.org/user_builds/roboy-memory/checkouts/docs/src/org/rob`**

dir **`/home/docs/checkouts/readthedocs.org/user_builds/roboy-memory/checkouts/docs/src`**

dir **`/home/docs/checkouts/readthedocs.org/user_builds/roboy-memory/checkouts/docs/src/org/rob`**

Solution Strategy

Basic decisions for Memory Module:

- Separation of concern through decoupling request processing and a persistence layer.
- Iterative and incremental development is adopted.
- Highest priority is Knowledge Representation implementation to satisfy the requirements and abilities for Dialog. Roboy Communication Standard is of the second priority as it follows the KR structure. The following priority is providing other modules with actual client and interfaces for the usage.
- For Knowledge Representation, a graph-based approach was chosen. Thus the persistency layer is presented by Neo4j graph database.
- Client for request processing is implemented on top of rosjava.

Current implementation:

- Graph-based Knowledge Representation ver. 1.1 on remote server.
- Redis for face features storage on remote server.
- Roboy Communication Standard commands pool.
- Java client software.

Motivation

The motivation to use a graph-based approach was easier (and probably more obvious) maintenance of relations and basic inference contained in graph-models by definition.

Java was the choice for development because it is Neo4j native language, thus has better support.

Redis was chosen as a simple yet powerful and fast (which is important for online face recognition) key-value storage.

Choice of rosjava was forced by both the usage of Java and ros as means of communication between Roboy parts.

Roboy Communication Standard was introduced to make querying more human-readable and graceful.

Java Client Flowchart

Overview

The flowchart shows the process of parsing and processing the queries within the java client. Query elements that pass the validation are documented on the following page: *Neo4j Memory Architecture*.

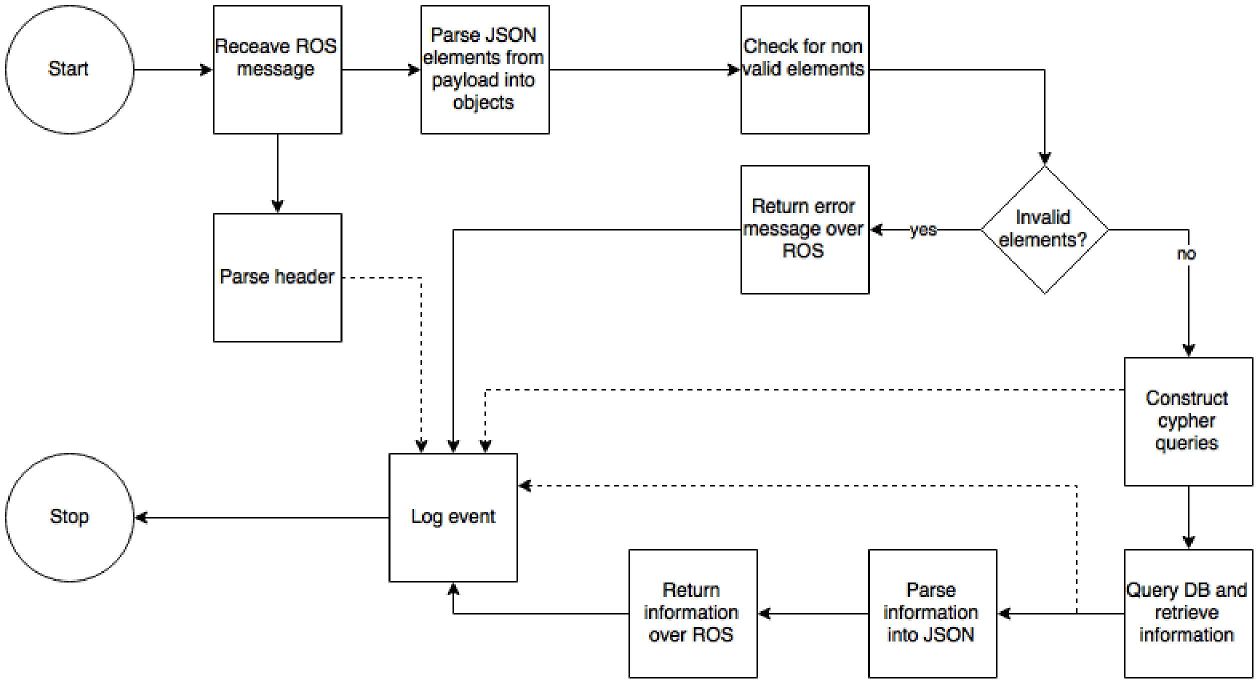

Start
Receave ROS message
Parse JSON elements from payload into objects
Check for non valid elements
Parse header
Return error message over ROS
yes
Invalid elements?
no
Construct cypher queries
Stop
Log event
Return information over ROS
Parse information into JSON
Query DB and retrieve information

Fig. 3.2: Java client overview

Building Block View

Overview

The white box view of the first level of the code. This is a white box view of the system as shown within the in Context in figure: *UML System Context*. External libraries and software are clearly marked.

Runtime View

UML-type sequence diagram - Shows how components interact with each other during runtime.

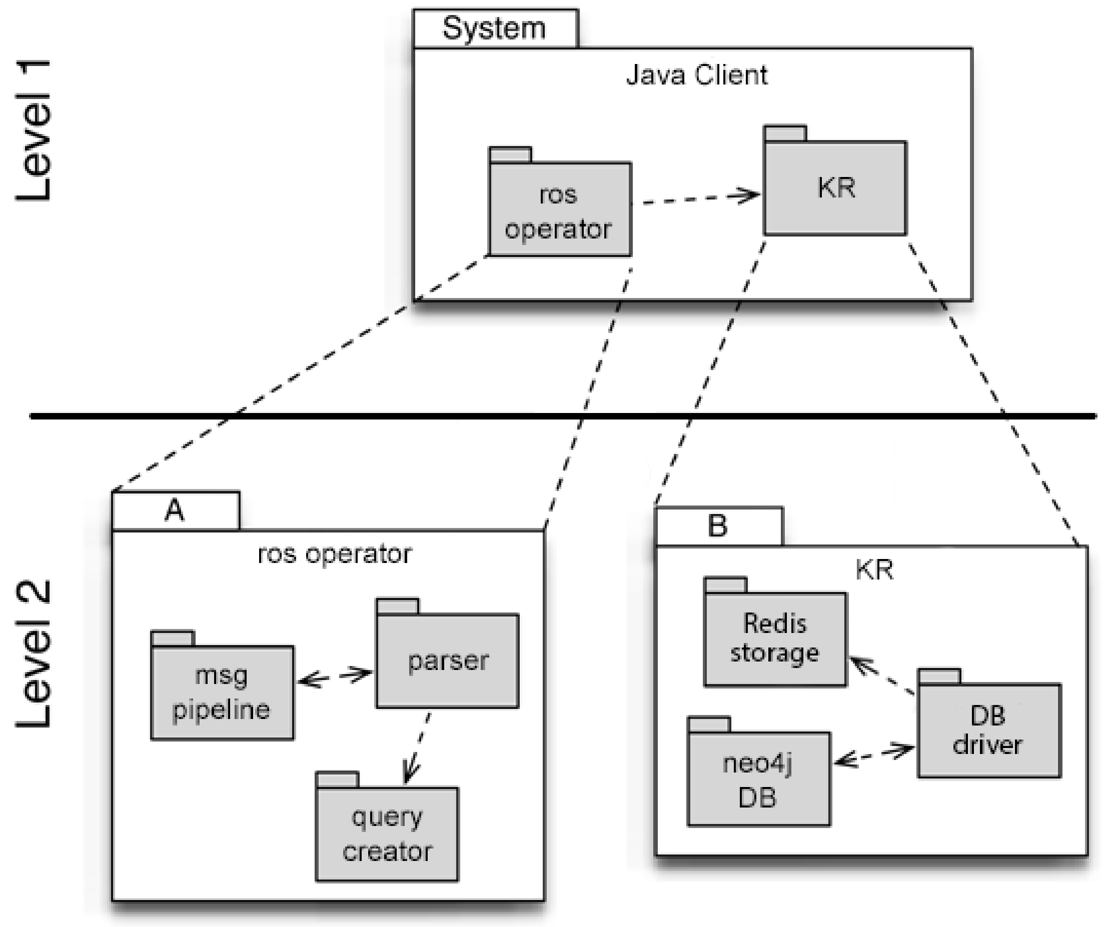
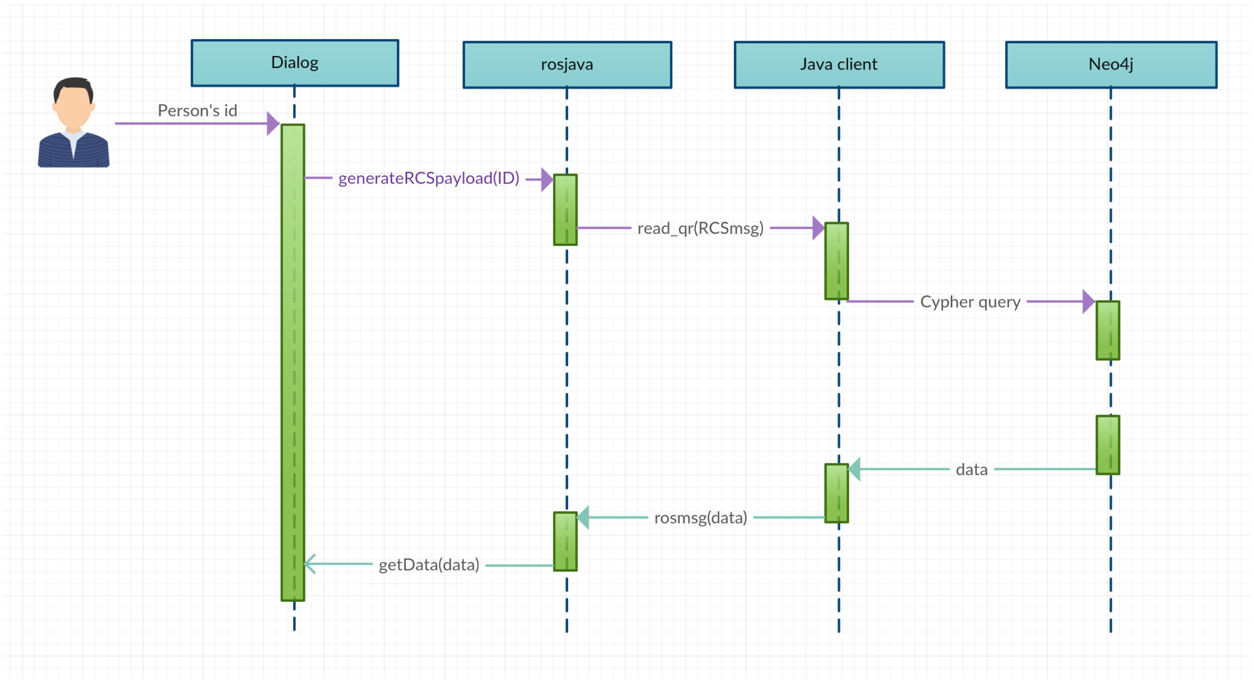
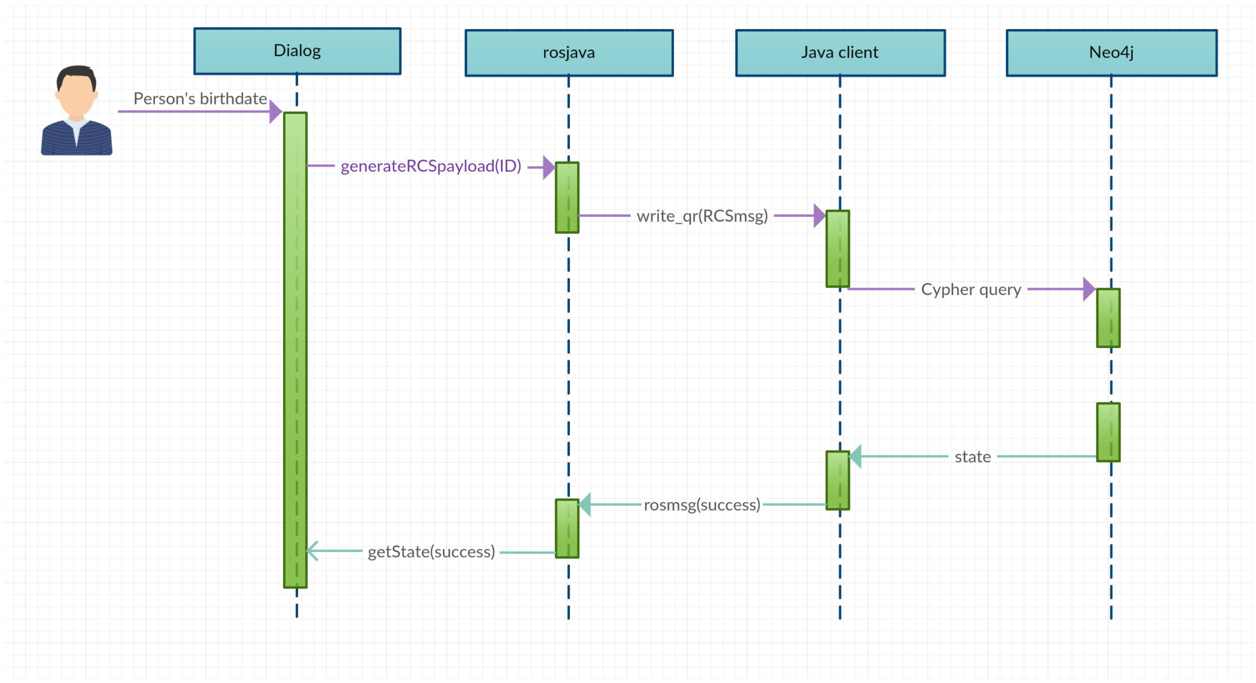


Fig. 3.3: Building blocks overview





Runtime Scenario 1 - Read person’s name by id

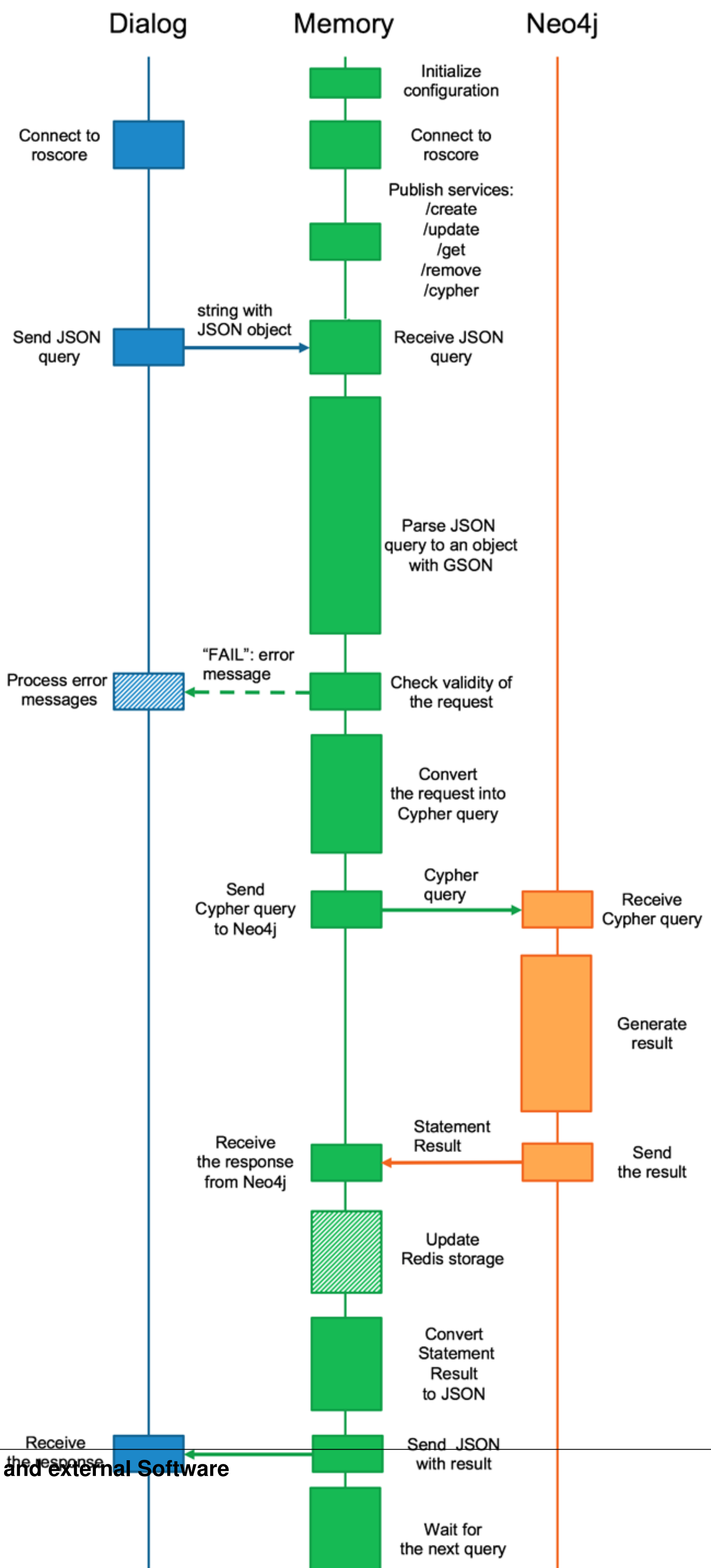
Runtime Scenario 2 - Write person’s birthday by id

General Sequence Workflow

Deployment View

Libraries and external Software

Contains a list of the libraries and external software used by this system.


Dialog
Memory
Neo4j
Initialize configuration
Connect to roscore
Connect to roscore
Publish services: /create /update /get /remove /cypher
string with JSON object
Send JSON query
Receive JSON query
Parse JSON query to an object with GSON
"FAIL": error message
Process error messages
Check validity of the request
Convert the request into Cypher query
Send Cypher query to Neo4j
Cypher query
Receive Cypher query
Generate result
Statement Result
Receive the response from Neo4j
Send the result
Update Redis storage
Convert Statement Result to JSON
Receive the response
Send JSON with result
Wait for the next query

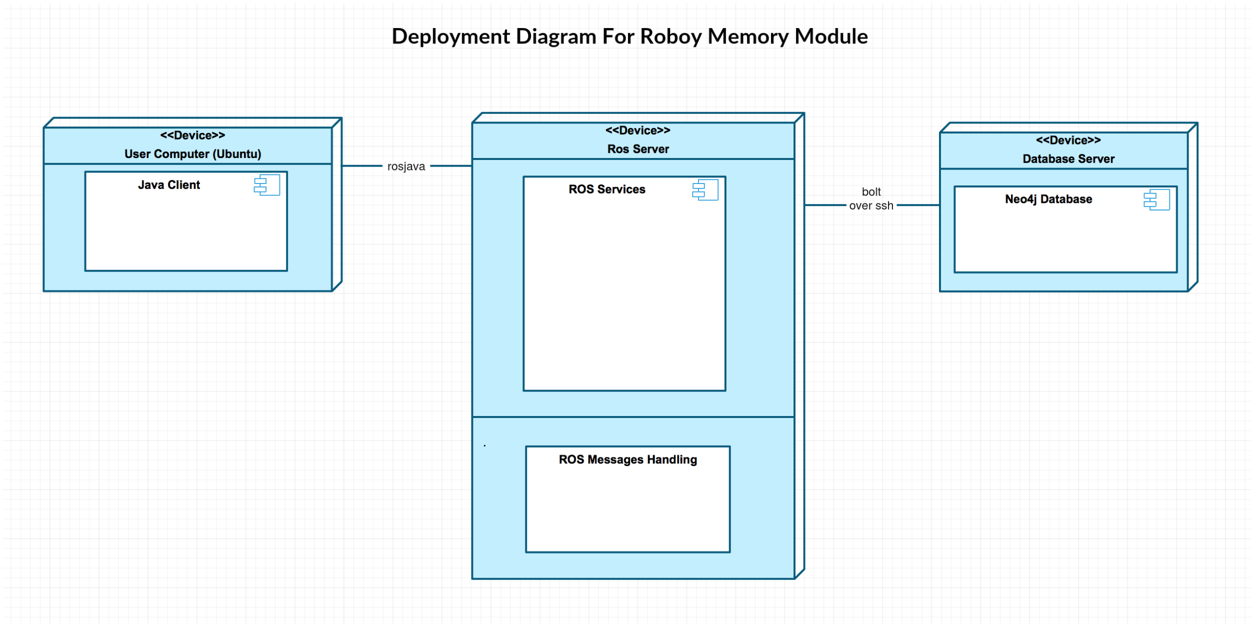


Table 3.4: Libraries and external Software

Name	URL/Author	License	Description
junit	http://junit.org/junit4/	Eclipse Public License - v 1.0	a simple framework to write repeatable tests
Neo4j driver	https://neo4j.com/download/other-releases/#drivers	Apache License, v. 2.0	access to the Neo4j graph database through Java
ros-java	https://github.com/rosjava/rosjava_core	Apache License, v. 2.0	a client library for ros communications in java as well as growing list of core tools (e.g. tf, geometry) and drivers (e.g. hokuyo)
Jedis	https://github.com/xetorthio/jedis	MIT License	a small and sane Redis java client
Gson	https://github.com/google/gson	Apache License, v. 2.0	a Java serialization/deserialization library to convert Java Objects into JSON and back

About arc42

html

> **Note**
>
> This version of the template contains some help and explanations. It is used for familiarization with arc42 and the understanding of the concepts. For documentation of your own system you use better the *plain* version.

Literature and references

Starke-2014 Gernot Starke: Effektive Softwarearchitekturen - Ein praktischer Leitfaden. Carl Hanser Verlag, 6, Auflage 2014.

Starke-Hruschka-2011 Gernot Starke und Peter Hruschka: Softwarearchitektur kompakt. Springer Akademischer Verlag, 2. Auflage 2011.

Zörner-2013 Softwarearchitekturen dokumentieren und kommunizieren, Carl Hanser Verlag, 2012

Examples

- HTML Sanity Checker
- DocChess (german)
- Gradle (german)
- MaMa CRM (german)
- Financial Data Migration (german)

Acknowledgements and collaborations

arc42 originally envisioned by Dr. Peter Hruschka and Dr. Gernot Starke.

Sources We maintain arc42 in *asciidoc* format at the moment, hosted in GitHub under the aim42-Organisation.

Issues We maintain a list of open topics and bugs.

We are looking forward to your corrections and clarifications! Please fork the repository mentioned over this lines and send us a *pull request*!

Collaborators

We are very thankful and acknowledge the support and help provided by all active and former collaborators, uncountable (anonymous) advisors, bug finders and users of this method.

Currently active

- Gernot Starke
- Stefan Zörner
- Markus Schärtel
- Ralf D. Müller
- Peter Hruschka

- Jürgen Krey

Former collaborators

(in alphabetical order)

- Anne Aloysius
- Matthias Bohlen
- Karl Eilebrecht
- Manfred Ferken
- Phillip Ghadir
- Carsten Klein
- Prof. Arne Koschel
- Axel Scheithauer

Index