Funded by the 7th Framework Programme
of the European Union

Project Acronym:	RAPP
Project Full Title:	Robotic Applications for Delivering Smart User Empowering Applications
Call Identifier:	FP7-ICT-2013-10
Grant Agreement:	610947
Funding Scheme:	Collaborative Project
Project Duration:	36 months
Starting Date:	01/12/2013

D3.2 RAPP Modules

Deliverable status:	First Version
File Name:	RAPP_D3.2_V1.0_20150526.pdf
Due Date:	31 May, 2015
Submission Date:	31 May, 2015
Dissemination Level:	Public
Task Leader:	5 - Ortelio
Author:	Alexandros Gkiokas

The RAPP project consortium is composed of:

CERTH	Centre for Research and Technology Hellas	Greece
INRIA	Institut National de Recherche en Informatique et en Automatique	France
WUT	Politechnika Warszawska	Poland
SO	Sigma Orionis SA	France
Ortelio	Ortelio LTD	United Kingdom
ORMYLIA	Idryma Ormylia	Greece
INGEMA	Fundacion Instituto Gerontologico Matia - Ingema	Spain
AUTH	Aristotle University of Thessaloniki	Greece

Revision Control

VERSION	AUTHOR	DATE	STATUS
0.9	Alexandros Giokas (ORTELIO)	May 15, 2015	Initial draft
1.0	Emmanouil Tsardoulias (CERTH/ITI)	March 10, 2015	Final corrections

Project Abstract

The RAPP project will provide an open-source software platform to support the creation and delivery of Robotic Applications (RApps), which, in turn, are expected to increase the versatility and utility of robots. These applications will enable robots to provide physical assistance to people at risk of exclusion, especially the elderly, to function as a companion or to adopt the role of a friendly tutor for people who want to partake in the electronic feast but don't know where to start.
The RAPP partnership counts on seven partners in five European countries (Greece, France, United Kingdom, Spain and Poland), including research institutes, universities, industries and SMEs, all pioneers in the fields of Assistive Robotics, Machine Learning and Data Analysis, Motion Planning and Image Recognition, Software Development and Integration, and Excluded People. RAPP partners are committed to identify the best ways to train and adapt robots to serve and assist people with special needs.
To achieve these goals, over three years, the RAPP project will implement the following actions:

- Provide an infrastructure for developers of robotic applications, so they can easily build and include machine learning and personalization techniques to their applications.
- Create a repository, from which robots can download Robotic Applications (RApps) and upload useful monitoring information.
- Develop a methodology for knowledge representation and reasoning in robotics and automation, which will allow unambiguous knowledge transfer and reuse among groups of humans, robots, and other artificial systems.
- Create RApps based on adaptation to individuals and taking into account the special needs of elderly people, while respecting their autonomy and privacy.
- Validate this approach by deploying appropriate pilot cases to demonstrate the use of robots for health and motion monitoring, and for assisting technologically illiterate people or people with mild memory loss.

The RAPP project will help to enable and promote the adoption of small home robots and service robots as companions to our lives. RAPP partners are committed to identify the best ways to train and adapt robots to serve and assist people with special needs. Eventually, our aspired success will be to open and widen a new 'inclusion market' segment in Europe.

Table of Contents

Executive summary

The present document is a deliverable of the RAPP project, funded by the European Commission’s Directorate-General for Communications Networks, Content & Technology (DG CONNECT), under its 7th EU Framework Programme for Research and Technological Development (FP7).

As our societies are affected by a dramatic demographic change, in the near future elderly and people requiring support in their daily life will increase and caregivers will not be enough to assist and support them. Socially interactive robots can help to confront this situation not only by physically assisting people but also functioning as a companion. The increasing sales figures of robots are pointing that we are in front of a trend break for robotics. To lower the cost for developers and to increase their interest on developing robotic applications, the RAPP introduces the idea of robots as platforms. RAPP project will provide a software platform in order to support the creation and delivery of robotic applications (RApps) targeted to people at risk of exclusion, especially older people. In the context of the RAPP project,

Deliverable named “D3.2 RAPP Modules” is a prototype. The code of the RAPP Modules is available in the project’s git account: https://github.com/rapp-project/. This document is a brief description of the implemented RAPP Modules.

Introduction

The RAPP project is a highly modularised platform, made up of different components, which complement each other. The overall architectural design is made up of three modules:

- The RAPP cloud
- The RAPP API/SDK
- The RAPP client

Each serves a purpose, is developed by one or more partners, and contains many other sub-modules, libraries and components of its own. Below, you will find a detailed description for each of those three main parts.

1 The RAPP Cloud

The cloud is made up of many different modules, which do not all necessarily interact. Those modules are:

- RAPP Store
- RAPP Services cloud
- RAPP Database

1.1 The RAPP Store

The RAPP store is essentially a frontend-backend web-server, which acts as the RAPP submission portal. Any developer interested in creating a RAPP, may submit it via the RAPP store, where after packaging and building, it can be distributed via the RAPP cloud.

The store relies on Apache2/PHP and MySQL as the back-end system, whereas the front-end is written in CSS/HTML and JavaScript. The back-end runs in a sand-boxed Virtual machine, as users are allowed to upload code, and compile it via the store's web-front, which gives access to low-level mechanisms. The Back-end also uses ROS/catkin, GNU GCC, and CMake.

The packaged RAPPs from the store, are available (upon approval) to all RAPP-cloud subscribers (robot clients). Currently, the store is in alpha phase, as it is being tested with the RAPP submission pages, which accept C++, ROS/C++ and JavaScript source code.

1.2 The RAPP Services

The services running in the cloud consist of mostly C++ or Python ROS-based nodes. Those are accessible via the RAPP API, which communicates HTTP traffic via TCP/IP to a HOP Server. The HOP server delegates service calls to the RAPP Services.

The main components are the HOP Server, the ROS master node and other nodes required to run the services, as well as other miscellaneous files needed for the execution of the services. The currently implemented ROS services that are directly connected with the HOP services are:

- /rapp/knowrob/subclasses_of
- /rapp/speech_detection_sphinx4_configure (configures speech detection based on a limited vocabulary)
- /rapp/speech_detection_sphinx4 (performs speech detection)
- /rapp/speech_detection_sphinx4_batch (configures and performs speech detection)
- /rapp/audio_processing/set_noise_profile (stores a user’s noise profile)

- /rapp/face_detection_service
- /rapp/qr_detection_service
- /rapp/audio_processing/denoise (denoises an audio file)

The corresponding HOP services that are available to the RAPP clients follow:

- QR-Detection Service:
    - URI/ServiceName: **qrDetection**
    - Parameters:
        - IN:
          {fileUrl: ''}
            - **fileUrl**: Returned by hop, path to the uploaded audio-file. Defined at the field of multipart/form-data post request.

OUT:

- Dynamic_vector that carries the qrCenters found in JSON representation
  ponts[{y:(int), x:(int), z(int)}]
  e.g
  [{y:14, x:164, z:134}, {x:13, y:13, z:67}]

- Face-detection Service:
    - URI/ServiceName: **faceFetection**
    - Parameters:
        - IN:
          {fileUrl: ''}
            - **fileUrl:** Returned by hop, path to the uploaded audio-file. Defined at the field of multipart/form-data post request.

OUT:

- A JSON representation of an object holding faces_up_left and faces_down_right dynamic_vectors points
  e.g
  { faces_up_left[{y:(int), x:(int), z:(in)}], faces_down_right[{y:(int), x:(int), z:(int)}] }

Query to ontology -- Subclasses-Of:
URI/ServiceName: **ontology_subclassesOf**
Parameters:
IN:

- **{ queryStr:"" }** The ontology_subclassesOf query string

OUT:

- A dynamic_vector holding the subclassesOf query return values in JSON representation.
  queryAns[] → String vector
  e.g:
  ['http://knowrob.org/kb/knowrob.owl#Oven',
  'http://knowrob.org/kb/knowrob.owl#MicrowaveOven',
  'http://knowrob.org/kb/knowrob.owl#RegularOven',
  'http://knowrob.org/kb/knowrob.owl#ToasterOven'

- Speech Detection Sphinx4:
    - URI/ServiceName: **speech_detection_sphinx4_batch**
    - Parameters:
        - IN:
          {fileUrl:'', language:'', audio_source'', words: [], sentences: [], grammar: [], user: ''}
            - **fileUrl**: Returned by hop, path to the uploaded audio-file. Defined at the field of multipart/form-data post request.
            - **language**: Language to be used by the speech_detection_sphinx4 module. Currently, valid language values are 'gr' for Greek and 'en' for English.

- **audio_source**: A value that presents the <robot>_<encode>_<channels> information for the audio source data. Used for correctly denoising the audio.
- **words[]**: A vector that carries the words to search for.
- **sentences[]**: The language model in the form of sentences.
- **grammar[]**: The grammar model. When a grammar model is specified the speech recognizer tries to only identify words that belong in the set.

OUT:

- JSON object that carries a vector of the words-found

It should be stated that many other ROS services exist (mainly performing tasks in the ontology and MySQL database), for which the respective HOP services are currently under development.

1.3 The RAPP Database

A MySQL Database is setup on the RAPP cloud. That database is shared across the RAPP Store and the RAPP cloud, and contains all data and meta-data related to the RAPPs available, as well as information stored by RAPP, or the RAPP clients (from the robot). Its purpose is to provide cloud-based storage ability to RAPPs, users, developers, as well as maintain the RAPP cloud information.

Locally (e.g., from the cloud itself) it is accessible directly via sockets, whereas remotely it is accessible only via the RAPP API.

2 The RAPP API/SDK

The API/SDK is the development library used for C++ and JavaScript development of RAPPS. Currently we have implemented the C++ API only, with JavaScript development being under-way. The API encapsulated and wraps around service calls, service object that relate to service calls (such as pictures, faces, qr-codes, etc.) as well as object-database management, such as objects that can be stored permanently on the cloud for reuse. Furthermore, the API provides delegation of computation to the cloud via dynamic execution, as well as execution of dynamic agents on the robot. The API is the bridge between the RAPP architecture on the robot and the cloud, as well as a wrapper around commonly-used ROS functionality. The RAPP API for C++11 is reliant on GNU GCC, and the RAPP API for JavaScript is reliant on HOP and Node.Js.

3 The RAPP Client

On the robot, the RAPP client is the main executing controller. It self, it is able to call other RAPPs, delegate execution to the dynamic agent. The RAPP client is installable as a package, and its dependencies are the RAPP SDK library, HOP, GNU GCC, and ROS.