Rapyuta: The RoboEarth Cloud Engine

Lei Nie
University of Minnesota - Twin cities
04/16/2015
Roadmap

- Introduction
- Architecture and design choices
- Use cases and performance
- Conclusion
Definition

Rapyuta is an open source Platform-as-a-Service (PaaS) framework designed specifically for robotics applications.
Advantages

- Interconnected computing environments
  - share services and information
  - platform for multi-robot deployment

- Assess to knowledge repository

- Open source ROS packages

- bidirectional, full duplex communications
Roadmap

- Introduction
- Architecture and design choices
- Use cases and performance
- Conclusion
Architecture Components

Rapyuta consists mainly of the computing environments for robots to offload their tasks, a set of communication protocols, a set of core tasks to administer the system, and a command data structure to organize the system administration.
A. Computing Environment

- Rapyuta’s computing environments are implemented using Linux Containers, which provide a lightweight and customizable solution.
- Applications run at native speed
- Allow easy configuration
B. Communication Protocols

- Split into three parts: internal, external and between part
- One of the basic building blocks of Rapyuta’s communication protocol is the Endpoint which represents a process that consists of Ports and Interfaces.
Internal Communication Protocol

- Twisted framework

- Type of messages can be split into two categories
  - Administrative messages
  - Data messages
External Communication Protocol

- WebSockets protocol

- Pure ASCII JSON messages
  top level structure: { "type":"...", "data": ... }
  which is an unordered collection of key/value pairs.
Handling Big Binary Messages

- For some types of data, it is better to transport them as a binary blob instead of using their corresponding ROS message type encoded as a JSON string.

```json
"iTAg" : "converter_modifyImage",
"type" : "sensor_msgs/Image",
"msgID" : "msgID_0",
"msg*" : "f9612e9b3c7945ef8643f9f590f0033a"
```
Basic Communication Example

1) Initialization

http://[domain]:[port]?userID=roombaOwner&robotID=roomba&key=secret&version=[version]

{
  "url" : "ws://[domain]:[port]/",
  "key" : "8f42eeedefb0463a834c582782a9e2bc"
}

ws://[domain]:[port]/?userID=roombaOwner&robotID=roomba&key=8f42eeedefb0463a834c582782a9e2bc

2) Container Creation

"containerTag" : "roombaClone"
3) Configure Nodes

```
"addNodes" : [{
    "containerTag" : "roombaClone",
    "nodeTag" : "positionRecorder",
    "pkg" : "testPkg",
    "exe" : "posRecorder.py"
}]

"addInterfaces" : [{
    "endpointTag" : "roomba",
    "interfaceTag" : "pos",
    "interfaceType" : "SubscriberConverter",
    "className" : "geometry_msgs/Pose2D"
}, {
    "endpointTag" : "roombaClone",
    "interfaceTag" : "pos",
    "interfaceType" : "PublisherInterface",
    "className" : "geometry_msgs/Pose2D",
    "addr" : "/posPub"
}]
```

4) Binding Interfaces

```
"connect" : [{
    "tagA" : "roomba/pos",
    "tagB" : "roombaClone/pos"
}]
```

5) Data

```
"iTag" : "pos",
"type" : "geometry_msgs/Pose2D",
"msgID" : "id",
"msg" : {
    "x" : 3.57,
    "y" : -44.5,
    "theta" : 0.581
}
```
C. Core Tasks

- **Master**: The master is the main controller task that monitors and maintains the command data structure.

- **Robot**: The Robot task set is defined by the capabilities necessary to communicate with a robot.

- **Environment**: The Environment task set is defined by the capabilities necessary to communicate with a computing environment.

- **Container**: The Container task set is defined by the capabilities necessary to start/stop computing environments. A process containing the container task set runs inside every machine.
The Network is the most complex part of the data structure.
The User represents generally a human who has one or more robots that need to be connected to the cloud.
The LoadBalancer is used to manage the Machines which are intended to run the computing environments.
the Distributor is used to distribute the incoming connections from the robots over the available robot Endpoints.
Roadmap

- Introduction
- Architecture and design choices
- Use cases and performance
- Conclusion
Use Case
The results in Figure show:

1) External communication (R2C) is the biggest constraint of Rapyuta’s throughput.

2) The difference between containers running in the same machine and different machines, due to the iptables and port forwarding overheads, is relatively small (< 1 [ms]).

3) Rapyuta introduces an overhead of 1 [ms] for data sizes up to 1 MB, which can be seen from the differences between C2C-1 and N2N.
Roadmap

- Introduction
- Architecture and design choices
- Use cases and performance
- Conclusion
Conclusions

- Described the design and implementation of Rapyuta
- Showed the flexibility of Rapyuta’s modular design by giving two specific cloud robotic configurations
- To sum up, it’s good!
Thank you!