ROS @ Aware Home

Rahul Rajan
Vikram Krishnamurthy
12/2/2009

Contents

1. Introduction

1.1 Problem Summary

Aware Home currently consists of various independent applications which, over the years, have become inaccessible, yet functioning. This makes it extremely difficult for someone who wishes to analyze the processed information by a device. This is mainly because of the fact that the information is lost as soon as the device processes its information and also because of the fact that most of these devices have its own access mechanism. The need was to provide a proof of concept of a centralized framework which could transfer data from a device to an application, or to a controller. The capability to locate an existing device and to poll it, automatically get data from it as soon as it transmits was also intended.

1.2 Motivation to use ROS

ROS stands for Robotic Operating System, which is a light weight OS used in Robots. This provides typical OS services like hardware abstraction, device control and Inter Process Communication. ROS is a distributed framework of processes (aka Nodes) that enables executables to be individually designed and loosely coupled at runtime. This was the primary motivation which allowed us to zero in on ROS as a candiadate for the middleware implementation at Aware Home. The idea was to use the OS primitives provided by ROS to dynamically communicate with various devices, poll for information and make devices context aware.

2. Scope

The following were the goals kept in mind while starting with the implementation of POC.

- Deploying a couple of sensors by making them into a ROS Node.
- Interaction among the devices by making use of asynchronous streaming of data over Topics.

3. ROS Concepts

A ROS Environment consists of peer-peer network of ROS Processes (explained below) which processes data. The environment could be described by these individual concepts.

Nodes: Nodes are proceeses which perform computation. It is written in C++ or Python.

Master: ROS Master provides name registration and lookup services. Without the Master, Nodes would not be able to find each other, exchange data.

Messages: Nodes communicate with each other by passing messages. Message is a simply a data structure, comprising typed fields.

Topics: Messages are routed with publish / subscribe semantics. A node sends out a message by publishing it to a given Topic. The Topic identifies the content of the message. A Node that is interested in a certain kind of data will subscribe to the appropriate Topic. There may be multiple concurrent publishers and subscribers for a single Topic, and a single Node may publish and/or subscribe to multiple Topics. In general, publishers and subscribers are not aware of each others' existence. This provides for a Producer Consumer scenario where both could operate without interference. Experiments and Accomplishments

4. Demonstration of feasibility

4.1 Wrapper for a Sensor

The first step was to abstract a sensor into a ROS Node so that it could operate upon the primitives of ROS. This essentially meant that we had to wrap a sensor with the ROS Client API's making it a full-fledged node. This was done by making use of roscpp and rospy, the ROS client libraries in C++ and Python respectively.

The devices had to be interfaced with the Kernel i.e., ROS Master. This required to get the device driver for the sensor. We were using X10 CM19A usb transceiver and we installed the driver for these class of sensors into the machine where ROS was deployed.

The next step was to associate a Topic with each class of sensors. This essentially meant that, every sensor could be identified with a Topic, making it easy for anyone who wants to read the data from it. We encapsulated the Sensor with a ROS Node and we began publishing the data from the device onto the Topic by reading and parsing the data from the device file. This greatly simplifies the things. Another node which needs to read the data from this has to just listen to the Topic and it would get the data. Also, this could be used to archive the information processed by a sensor, which could be later used for analytical purposes.

4.2 Distributed ROS Master

All the above were done with a single core running. This has implications in case of a server failure wherein the ros name look up table would be lost. This is why we explored the option having distributed servers. An added advantage is the ability to group devices by location (1st or 2nd floor) or functionality (wifi, zigbee), and also to distribute the load. To enable distributed functionality, we implemented ROS multimaster API in our code. We demonstrate its potential in an environment consisting of two roscores, A and B, both running their own set of ROS nodes. A device on roscore A is publishing its data onto a topic T. To listen to this topic, a listener node has to register itself on roscore A. As long as topic T is registered in roscore A, the listener can listen to the data from the publisher. The problem arises when roscore A has crashed and topic T is no longer registered. A new listener can register itself on roscore A once its been restarted, but will not be able to listen to any data on topic T because although the publisher continues to publish on topic T, it isn't registered with roscore A. It is important to note that all the nodes continue to run independent of the roscore after their initialization because roscore simply provides name

registration and lookup. So when roscore crashes, the nodes continue to run, but the name table is lost. To circumvent this, we continually poll the roscores to see if they are alive. If they aren't the nodes implicitly reregister themselves with the roscore B.

5. Research Scope

• We explored the capabilities by taking into account a class of Sensors. However, this could be carried over to all class of devices. The primary issue would be get the device to interface with the Kernel. Once that is done, making the device into a ROS Node would be a relatively simple task.
• Once a roscore dies all the associated mappings with the core dies. We were able to save the state of the system before it dies. Re-mapping the saved state onto a new core would present its own set of challenges, because once the registration is done, the Nodes no longer communicate through roscore. Instead, they communicate over TCP/IP. Hence, remapping would involve verification of the system state.

6. Conclusion

ROS has the potential to meet the current needs of Aware Home. The challenges , as we mentioned would be interface all the devices into the kernel. However, there are capabilities like Services , which is synchronous communication (Topics are asynchronous communication), which could be harnessed as well.

7. References

http://www.ros.org/wiki/
http://www.ros.org/wiki/APIs