An RPC-based Service Framework for Robot and Sensor Network Integration

Peter Corke∗, Wen Hu†, and Matthew Dunbabin‡
∗Queensland University of Technology, Australia
Email: peter.corke@qut.edu.au
†‡CSIRO ICT Centre, Australia
Email: {wen.hu, matthew.dunbabin}@csiro.au

Abstract—We describe and evaluate a new programming and communications framework that eases the creation of complex heterogeneous systems comprising robots and sensor networks. We use a light-weight RPC-based service framework that allows robots and static sensor nodes to be considered as services, accessible to an internet connected end-user, or to each other. We present experimental results for a very large environmental monitoring application comprising a floating sensor network and a robotic boat.

I. INTRODUCTION

We wish to exploit the complementary characteristics of robots and sensor networks, interacting and cooperating, to achieve tasks that neither can accomplish alone — perceiving the environment and responding to it. Robots are typically mobile, complex, with high cost and power budget, whereas sensor nodes are static and simple, with low cost and power consumption. We typically deploy few robots but hundreds of sensor nodes. A sensor network can sample simultaneously at multiple locations, whereas robots can sample the environment locally and sequentially.

Sensor networks can provide robots with fast and cheap access to data beyond their perceptual horizon, localization information, as well as a large-area ad-hoc communications fabric for robot-robot and robot-operator communications. Conversely robots can assist a sensor network by deploying its nodes, by localizing sensor nodes post deployment [1], by making repairs or extensions as required, and acting as “data mules” to relay information between disconnected sensor clusters [9].

Many types of robots are suitable for extending the capability of WSNs for environmental monitoring. Unmanned Aerial Vehicles (UAVs) are a very active area of research and have the advantage of moving in relatively empty space with good access to GPS for navigation and the ability to cover very large areas. Autonomous Underwater Vehicles (AUVs) have been used for water physical-chemical sampling, photographic survey, and data muling but communications underwater are difficult and access to GPS is only available upon surfacing. Ground robots are also well developed, but have the disadvantage of operating in complex and often cluttered environments. In this work, we use an Autonomous Surface Vehicle (ASV) or robotic boat. To exploit the capabilities of robots, Teh et al. test high-speed robotic aircraft as the mobile gateway for static ground nodes [8] but the nodes cannot re-task robots dynamically. Real-time adaptive sampling has been investigated in [4], [6], [7].

Integrating these two technologies requires that we create and program networks of heterogeneous elements that include robots, static sensor nodes as well as remote human operators. The ability to dynamically re-task sensor nodes and robots by sending commands or uploading new code will be essential to the utility of the system, allowing different elements or regions of the network to be tailored to specific purposes and capabilities added or changed.

In this paper we present a framework that allows heterogeneous devices that provide different classes of services to support cooperative activities. It bridges the two technologies by integrating autonomous control of robotic vehicles with multi-hop message routing in ad-hoc networks. In this paper we briefly present the design and implementation of light-weight RPC-based services framework that provides the underlying communications mechanism and supports servers running on mote-class and Linux-class computers. We demonstrate the utility of the framework using a complex real-world deployments of a robot providing a sensor calibration service to a large-scale outdoor sensor network deployed on a lake. The framework allowed the applications to be developed by robotics researchers who did not need to know details of underlying sensor node operating systems or network communications protocols.

The remainder of this Section introduces the sensor network and robot integration challenges and requirements. Sections II describes the light-weight RPC system which forms the heart of the service framework. A key component in achieving integration is the robot gateway which is described in Section III. Section IV describes the application and presents the results and performance measures obtained. Conclusions are presented in Section V.

A. Integration challenges and requirements

A robot typically has significant resources compared to a sensor node: more computational power and memory (typically one or more PC-class computers), higher communications bandwidth (WiFi) and more energy. Robots and sensor networks also use very different software environments. Our robots use Linux, a component-oriented distributed blackboard...
middleware system and socket-based communications. Sensor nodes use our own Fleck Operating System (FOS) [2] and Collection Tree Protocol (CTP).

The network architecture is shown in Figure 1 and we require the following functionalities:

1) Stream messages from any sensor node or robot to any internet-connected device; for example static nodes to report environmental measurements, or a robot to report its location.

2) Any network element or an internet-connected device can request some particular state from any other network element.

3) Any network element or an internet-connected device can set some particular state on any other network element — state change on a robot leads to actuation.

The first functionality is achieved using well-known techniques such as CTP. In our system data is sent using our own Tagged Data Format (TDF) which is a compact self-describing schema. The second and third functionalities can be achieved most generally by considering that the network elements expose a set of services. These services can include the setting and requesting of particular device states. State, in the most general sense, includes measurements, program parameters, and even the program itself. Using this single abstraction it is possible to enable real time re-tasking of sensor nodes and robots.

II. A LIGHTWEIGHT RPC FRAMEWORK

There is a long history in the use of Remote Procedure Calls (RPCs) for creating distributed systems, and there have been several reports on the use of RPCs to simplify software development in the wireless sensor network context [3], [5], [10]. To the best of our knowledge, we are the first to investigate how to use RPC to simplify the communications between sensor networks and robots.

Our tiny RPC system uses a stateless protocol that allows a client to seamlessly access one or more servers over a communications link (radio network or serial). Clients can be written in C or Python, and servers are written in C. The use of ANSI C allows easy integration in sensor nodes running FOS or on Linux-class computers on the host or a robot. The tiny RPC allows sensor node and robot programmers to write distributed applications without worrying about the details of the underlying communication networks.

Our RPC shares some characteristics with Embedded RPC (ERPC) [10] such as transmitting data in sensor node native order, and the RPC interfaces are parsed directly from the source code. However, we generate the client stub in Python rather than storing the interface in an XML representation, and we allow for an arbitrary number of arbitrarily typed return values, as well as a status value.

The server function to execute is indicated by a single-byte in the RPC call message. The mapping from function to integer value is kept in a simple text file, a master list of all data types used by our sensor networks. A Python tool consistently assigns the numeric tags to each data type and autogenerates .h files, Python and Java classes for incorporation into backend applications.

The RPC generator fosrpcgen, written in Python, operates on a directory full of files, one per server function, that are written in a C-like language. The generator emits a number of output files: rpc_server.c and rpc_server.h, the server-side C-code skeleton; rpc_client.c and rpc_client.h, the client-side C-code proxies; and FOSRPC.py the client-side Python proxy class. The C files are completely self-contained, containing all the marshalling and unmarshalling code, and the transformed service code. Marshalling is achieved efficiently by mapping message buffers to autogenerated structs.

To illustrate the use of our RPC system from Python running on some internet connected host we create an RPC instance object:

```python
import FOSRPC
rpc = FOSRPC.FOSRPC(s)
```

where the class FOSRPC.py is autogenerated and contains one proxy method for each RPC service. The sensor network is accessed via a connection to the serial forwarder represented by the object s. To toggle the red LED is simply

```python
rpc.leds(1, 1, 0, address=12);
```

and to read 4 bytes of RAM starting at 0x100 is

```python
ret = rpc.read_ram(0x100, 4, address=12);
for (i,d) in enumerate(ret[0].data):
    print "[%d] = 0x%02x" % (i, d)
```
The return value of the RPC call is a list of RPCReturn objects whose attributes include the output arguments listed in the RPC definition. For this example the output argument data was an 8-element array, so the corresponding Python attribute is an 8-element list.

RPC completion is defined as the receipt of a message with the corresponding method index and sequence number, or in the case of a broadcast RPC on a timeout. For the broadcast case the function is executed in parallel on many nodes which all return their results to the caller. For example:

```
ret = rpc.power(address=FOSRPC.BROADCAST);
for r in ret:
    print "node %d: batt=%.2fV, solar=%.2fV, charge=%.2fmA" % (r._id, r.vbattery/1000.0, r.vsolar/1000.0, r.isolar/1000.0)
```

where the loop (lines 2–6) displays the result received from responding nodes, and the attribute _id is the address of the originating node, resulting in, for example:

```
2: batt=3.36V, solar=4.32V, charge=12.01mA
3: batt=3.24V, solar=2.14V, charge=0.00mA
```

If the requested service is not provided, the server returns the status FOSRPC_NOSUCHCMD.

### III. ROBOT SENSOR NETWORK GATEWAY

The robot gateway is a standard node that serves as the gateway between the sensor network and the robot's computer. For some applications the node is connected to a GPS expansion board (via serial port) and/or an inertial measurement unit (IMU) expansion board (via SPI bus).

#### A. Gateway application

The commfleck application running on the node, see Figure 2, performs the following functions:

1) A data message forwarding service accepts data in the TDF schema over the serial link and injects it into the WSN which returns it to the sink using Collection Tree Protocol.

2) RPC server #1 handles service requests received from the WSN. The services include the FOS standard services (Table I) such as temperature, power, memory access, and so on. Results are returned over the WSN with the TDF_RPC tag.

3) Service requests from the WSN that cannot be handled by RPC server #1 (function 2 above) are forwarded over the serial port to the robot host and RPC server #2. The RPC return, tagged as TDF_RPC, is sent back via the serial port and function 1 will then return it to the host via the WSN.

4) RPC server #3 handles service requests received from the serial port. The services, shown in Table II, allow control of GPS and IMU sampling as well as a subset of standard FOS services such as temperature, power, radio channel, and transmission power. The return message is sent back over the serial port.

5) Forwarding of overhead WSN message headers to the serial port for approximate localization.

The application contains three RPC servers, each generated with the tools described in Section II. While servers #1 and #3 could be in principle be combined the structure shown is more modular and each server runs as its own thread, the small additional cost in terms of code is not significant compared to total code usage, Table III, and this node is not reprogrammed over the air.

#### B. Robot Host Software

A simple protocol is used over the serial link (57.6kbaud) from the gateway node to the robot control computer, which has a start of message header, length, message type, payload, and CRC16. The message types are used by the robot host to demultiplex the message stream.

1) **Serial services:** RPC server #3 in the commfleck application responds to calls from the robot host, as listed in Table II, over the serial link. These services provide control of the rate at which sensor data is sent to the robot host. The server can be controlled from the robot using the C or Python RPC client interface.

2) **Robot RPC handler:** A key aspect of our design is service forwarding: the commfleck RPC server #1 will forward service requests it cannot handle to the robot host’s RPC server #2, which runs its own RPC server to provide robot specific services including:

   - `getpose`, return the robot’s $(x, y)$ position and heading angle
   - `setvel`, set the robot’s speed, heading, and duration
   - `settime`, set the robot’s onboard clock

The server #2 C-code executing on the robot host is also autogenerated. This allows the set of functions executed on the robot by the host to be extended in a way that is transparent to the sensor network and the commfleck.
TABLE III

<table>
<thead>
<tr>
<th>Application</th>
<th>count2radio</th>
<th>Lake node*</th>
<th>commFleck*</th>
</tr>
</thead>
<tbody>
<tr>
<td>No. threads</td>
<td>1</td>
<td>9</td>
<td>11</td>
</tr>
<tr>
<td>code size</td>
<td>23800</td>
<td>46002</td>
<td>60108</td>
</tr>
<tr>
<td>data size</td>
<td>104</td>
<td>412</td>
<td>668</td>
</tr>
<tr>
<td>bss size</td>
<td>466</td>
<td>897</td>
<td>1166</td>
</tr>
<tr>
<td>stacks</td>
<td>100</td>
<td>1746</td>
<td>2178</td>
</tr>
<tr>
<td>total RAM</td>
<td>670</td>
<td>3055</td>
<td>4012</td>
</tr>
</tbody>
</table>

IV. APPLICATION AND EVALUATION

We illustrate the effectiveness of our proposed RPC-based service framework for an application that comprises multiple static sensor nodes and a robot. We use custom sensor nodes, Fleck-3B, that employ the Nordic NRF905 radio transceiver operating in the 915MHz ISM band. The processor is an Atmega 1281 with 8 kbytes of RAM.

The system comprises two components shown in Figure 3. The first are the floating sensor nodes, of which 50 are currently deployed. Each is a square floating frame for stability, anchored to a concrete block. It is fitted with a Fleck node, a solar cell, and a temperature sensing string. This string comprises six 1-wire sensors (Maxim DS28EA00 digital thermometer chip encapsulated in silicone rubber) spaced at 1m intervals downward from the float. A high-gain (6dB) antenna is mounted atop a mast 1.5 m above the water surface in order to improve range. Bright, but low-power, strobes are activated at night to prevent collision with other watercraft. Vertical temperature profiling of the water provides information about water mixing within the lake and can be used to predict the development of algal blooms.

The floating node application, see Table III, comprises three concurrent threads: sample and return engineering data, sample and return temperature string data, and a watchdog. The engineering data returned includes: battery voltage and current, solar cell voltage and charging current, onboard temperature, downstream and upstream link quality, the number of hops to sink, and parent node. All data is returned in TDF format reports and the sample interval is 5 minutes. The node runs six other system threads associated with the networking function and RPC server.

The second component is the mobile node, a 14 foot twin-hull boat. It is propelled by a pair of electric trolling motors, with another electric motor that moves the twin rudders for steering. Two large solar panels provide power, a scanning laser rangefinder mounted high provides obstacle detection, and a depth sounder is fitted. A controllable boom positions the temperature sensor at a specified depth. The onboard computer is a laptop running Linux that communicates with the onboard control and sensing subsystems via a serial port. Another serial port connects to the commFleck for communications with the floating sensor network. The boat transmits sensor readings every 10 s via CTP over the WSN.

We use the mobile node for two purposes. Firstly, the mobile node is used to collect fine-grained water temperatures since it can be tasked to go to any location on the lake or follow arbitrary transects. This tasking can be a wide area survey or directed to an area where a temperature inversion is suspected based on static node temperature observations.

Secondly, the mobile nodes can be used to periodically calibrate the static sensors. Calibration or recalibration of sensors in the field is an important operational issue that is not much discussed in the literature and becomes more important as network scale increases. The design tradeoff in our system is that the large number of temperature sensors on the static nodes (6 sensors per node by 50 nodes is 300 sensors total) have to be cheap to reduce overall system cost as well as minimizing the risk due to damage or theft. The mobile sensing platform is a higher value asset that returns to base periodically so we can afford to use higher quality sensors which can be more conveniently recalibrated. The platform could also carry additional sensors that we cannot afford to deploy at each static node such as dissolved oxygen, or those which require reagent replenishment such as pH sensors.

A. Results

The mobile robot was tasked to collect fine-grained water temperatures between the nodes.

Figure 4 (left) shows a section of the path taken by the boat, as well as the location of the static floating nodes. This path has two sections: 1–17, from node 102 to 101, and 18–49 from node 102 to 104 to 102 then 101. The locations from which a temperature measurement were received at the base, via the network, are shown as small circles. We can see some gaps where measurements were not reported.

Figure 4 (right) shows the temperature measured by the boat along the path just described. Also shown, as horizontal lines, are the average temperatures measured at the locations of the static nodes. Several things are evident from the graph. Firstly, the temperature measurements are highly quantized, and the total span is just 0.6 deg. Secondly, there is evidence of variation in water temperature as the robot moves between nodes. The robot starts (point index 1) in the vicinity of node 102 and moves through node 103 (index 6) to node 101. We can see the temperature measurements show an increasing trend, moving from the temperature near node 102 to that near 103 and approaching that at node 101. The temperature sensors have a significant time constant so the temperature measurements are lagging. The second path segment, starts in the vicinity of node 102 (index 18) heads toward node 104 and shows a decreasing temperature trend as expected, then moves back toward node 102 and then on to node 101. This demonstrates that the robot node can, indeed, provide fine-grained spatial physical phenomenon measurements in real-time. Furthermore, when the mobile node is close to a static node, we can also use the sensor readings of the mobile node to recalibrate the sensor readings of the static node.

Each node reports network performance at 5 minute intervals and this includes a bidirectional link quality estimate, current parent, and path length. Some reports will not be received at the sink due to communication unreliability.
Table IV shows the packet delivery ratio and the length of the communication path. It shows that, except for the disconnected node (101), the static nodes achieved an average of 90% end-to-end delivery ratio. Node 102 was located closest to the sink and had a mean communication path length of 2.4 hops, and the best end-to-end delivery ratio (100%) during the experiment. Although the mobile node (120) had the shortest path length of 1.5 hops, it was disconnected from the network temporarily when it traveled close to node 101 and had an end-to-end delivery ratio of 67.8%. Node 101 was
TABLE IV
Network performance statistics (Packet delivery ratio and the length of communication path).

<table>
<thead>
<tr>
<th>Node ID</th>
<th>Delivery Ratio</th>
<th>Hop Count</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Mean</td>
</tr>
<tr>
<td>101</td>
<td>10%</td>
<td>2</td>
</tr>
<tr>
<td>102</td>
<td>100%</td>
<td>2.4</td>
</tr>
<tr>
<td>103</td>
<td>80%</td>
<td>2.38</td>
</tr>
<tr>
<td>104</td>
<td>90%</td>
<td>3.67</td>
</tr>
<tr>
<td>120</td>
<td>67.8%</td>
<td>1.5</td>
</tr>
</tbody>
</table>

Figure 5 (left) shows the estimation of bi-directional link quality between static node 102 and the mobile node (120) and their communication parents. It show the link quality is very dynamic over water due to strong fading from multipath and polarization loss as antenna orientation changes due to wave motion. The network’s response to link quality dynamics is reflected in rapidly changing path length, see Figure 5 (right).

V. Conclusions

The contribution of this paper has been the design, implementation and evaluation of an RPC-based service framework. We have shown that this framework provides a powerful abstraction for all aspects of node and robot control, and thus a powerful paradigm for integration. We introduced the concept of service forwarding in which services that cannot be provided by one device, for example a sensor network gateway node, are forwarded to another, for example a robot. This allows the set of functions executed on the robot to be extended in a way that is transparent to the sensor network and the gateway node.

A key indicator of the success of this framework is that robotics engineers were able to add new services easily and invoke them from the Python interpreter without needing to know anything about serial forwarders, MAC or routing protocols. Similarly sensor network engineers were able to query and change the state of the robot in exactly the same way they currently do for a remote sensor node.

Robot control software running on a PC-class computer with a component-based middle architecture is easily abstracted as a service within our framework. Robotics engineers were able to add new services easily and invoke them from the Python interpreter without needing to know anything about serial forwarders, MAC or routing protocols. Similarly, the sensor network engineers were able to query and change the state of the robot in exactly the same way they currently do this for a remote sensor node.

We have shown the utility of the RPC-based framework by demonstrating how it can be used in real environments. We have shown how the sensor network can be used to communicate with the robot—to task it, and to request robot state. We have also shown how the robot can assist the sensor network by performing independent measurements which can be used for sensor recalibration, and to act as a mobile relay to improve connectivity in the network.

REFERENCES