Robot Raconteur® version 0.8: An Updated Communication System for Robotics, Automation, Building Control, and the Internet of Things

John D. Wason

Wason Technology, LLC
Tuxedo, NY 10987
wason@wasontech.com

Abstract— **Robot Raconteur® (RR) is a powerful communication system for robotics, automation, building control, and the "Internet of Things". It has been used extensively in research at several universities and has reached Version 0.8, the first "beta" version ready for commercial use. A paper describing an early experimental version was presented at CASE in 2011 [1]. This paper presents the new programmatic, transport, and security features available in the new library. These new features include a new C++ based core library, an "Augmented Object Oriented" data model, transport security using TLS, WebSocket support, HTML5/JavaScript Browser support, and ASP.NET web server support. The new C++ library provides bindings for Python, MATLAB, C#, and Java. The library can run on all major operating systems and device architectures. Several example systems are presented.**

I. Introduction

Robot Raconteur® (RR) is a powerful communication system for robotics, automation, building control, and the "Internet of Things". It has been used extensively in research at several universities and has reached Version 0.8, the first "beta" version ready for commercial use. A paper describing an early experimental version was presented at CASE in 2011 [1]. This paper presents the new programmatic, transport, and security features available in the new library.

RR was developed to aid in the rapid integration of complex systems consisting of components that differ in manufacturer, platform, interface, and application programming interface (API) language. With existing communication technologies, extensive time was required to develop and configure the communication interface so that commands and data could be understood by both the client and service. Existing technologies that were easier to configure generally were not platform/language independent, or had high latency. An example is Java RMI [2] which is fairly easy to use, but is only Java compatible and does not have automatic type configuration abilities. CORBA [3] is another example that is platform independent but requires significant configuration and development on both the client and service before use. RR is designed to provide "plug-and-play" connections, meaning that a client receives all the type and command information required to communicate with a service at runtime. This provides an "instant-on" user experience, where fully featured object-oriented interfaces to the service are created dynamically upon connection with no additional effort by the user. In compiled languages where dynamic type generation is not possible, code generation is used to provide a similar experience but with fixed supported types. Services also use either dynamic types or code generation to greatly simplify the development process.

Special thanks to the Smart Lighting Center and the Center for Automation Technologies and Systems, both at Rensselaer Polytechnic Institute, Troy, NY, for the use of example images in this paper.

RR has a number of advantages over existing technology:

- Automatic data type negotiation at runtime for dynamic languages and code generation for compiled languages. This allows for plug-and-play operation with minimal development effort.
- "Augmented Object Oriented" model allowing for transactional, streaming, and (soft) real-time communication both client-to-service and service-to-client.
- Support for complex value data types with dynamic type negotiation.
- Support for numerous hardware platforms and software languages with more being added over time.
- Asynchronous operations.
- Security using authentication and TLS.
- Automatic node discovery.
- WebSocket support.
- WebBrowser client support.
- ASP.NET server support.
- Services can accept multiple client connections.
- RR does not require a master node.

While some of these features may exist in other communication packages, the combination of features and ease of use with automatic negotiation is unique to RR.

RR is similar to other Remote Procedure Call technologies (RPC) [4] but provides a unique "Augmented Object Oriented" [5] interface in combination with communication transports that provides secure transport over different technologies. The "Augmented Object Oriented" design means that objects have eight types of "members" each providing unique functionality. The member types are "property", "function", "event", "objref", "pipe", "callback", "wire", and "memory". Objects also have inheritance to provide forward and backward compatibility. The definitions of the object and data types are sent at runtime as a "Service Definition" allowing languages like Python, MATLAB, and JavaScript to dynamically provide client interfaces to services without any information about the service before the client connects.

RR uses "Transports" to communicate between "Nodes". The currently available transports are "TCP" for use with standard wired and wireless networking, "Local" for communication between programs running on the same computer, and "Hardware" for communicating with devices connected through USB, PCIe, and industrial fieldbuses. The TCP transport provides security using TLS and X509 certificates issued by Wason Technology, LLC providing industry standard levels of security. The TCP transport is also capable of connecting and accepting the HTTP WebSocket [6] protocol providing compatibility with existing web browsers and web servers. Auto-discovery is available to find nodes and determine the connection URLs. Nodes can be discovered by identifier or type of service.

The core RR library is written in C++ using the Boost [7] libraries and is capable of running on a number of platforms. It has minimal dependencies making it highly portable. Currently RR has been compiled and tested on Windows (x86 and x64), Linux (x86, x86_64, ARM hard-float, ARM soft-float, PowerPC, MIPS), Mac OSX, Android (ARM, x86), and iOS. Language bindings are used to allow access to the library from other languages. Additional supported languages include C#, Java, Python, and MATLAB.

Two additional implementations are available for use in a web browser written in JavaScript and for use in an ASP.NET server written in pure C#. These implementations take advantage of the support for WebSockets [6] over TCP to allow for compatibility with existing web infrastructure. The HTML5/Javascript implementation has been tested with Chrome, Firefox, Internet Explorer, Edge, and Safari. The pure C# implementation (Called RobotRaconteur.NET/CLI) has been tested with IIS and allows RR to run inside a web server.

RR is very different than other currently used technologies in robotics communication, Robot Operating System (ROS) [8] and Data Distribution Service (DDS). ROS provides very simple communication using "services" and "topics". "Services" in ROS provide a single function, and "topics" provide a way to publish data to subscribed nodes. These methods are far less capable than the "Augmented Object Oriented" model provided by RR. ROS also requires a central "Master Node" and does not provide security for its connections. DDS is not an RPC technology and is instead analogous to a distributed real-time database. DDS has the advantage of being capable of hard real-time communication. This capability is being developed for RR. While they are both used in automation systems, DDS and RR are not similar technologies. The paper presented in 2011 [1] compares RR to other more traditional RPC technologies and for brevity the comparison is not repeated here.

RR is released under a commercial license that is royalty-free when using the TCP or Local transport with the exception of the TLS certificates for secure communication. The certificates for use with TLS are sold for a nominal cost by Wason Technology, LLC. RR is not currently open source, and the reasoning for this decision is discussed in Section VII.

The rest of this paper discusses the "Augmented Object Oriented" model, the available transports, the compatibility of the software, and the license. It concludes with some example systems and a discussion of future work. This paper is a brief introduction to the RR system. Please see [9] for more detailed information.

II. "Augmented Object Oriented" model

Example 1 Service definition file "experimental.create.robdef"

```
#Service to provide sample interface to the iRobot Create
service experimental.create

option version 0.5

struct SensorPacket
        field uint8 ID
        field uint8[] Data
end struct

object Create
        option constant int16 DRIVE_STRAIGHT 32767
        option constant int16 SPIN_CLOCKWISE -1
        option constant int16 SPIN_COUNTERCLOCKWISE 1


        function void Drive(int16 velocity, int16 radius)

        function void StartStreaming()
        function void StopStreaming()

        property int32 DistanceTraveled
        property int32 AngleTraveled
        property uint8 Bumpers

        event Bump()

        wire SensorPacket packets

        callback uint8[] play_callback(int32
             DistanceTraveled, int32 AngleTraveled)

end object
```

RR is designed to be used primarily with robotic systems. These systems require different data types and data delivery methods when compared to business applications that would use a technology such as SOAP [10]. Unlike other object-oriented RPC technologies such as Java RMI [2] and .NET Remoting [11], RR has a strong distinction between *data* and *objects*. All objects are owned by the service, and RR allows the clients to access them through "object references" or "proxies". Data can then be transfered between the client and the service using the object's "members". The description of what objects and data types are available in the service are described by "service definitions" that are transmitted to the client at the time of connection. This allows for dynamic "object references" to be generated using meta-programming in Python, MATLAB, JavaScript, and potentially many other languages. This capability in combination with the rich compatibility and object/data model is unique to RR. Examples

Example 2 Service definition file "experimental.createwebcam.robdef"

```
#Service to provide sample interface to webcams
service experimental.createwebcam

option version 0.5

struct WebcamImage
    field int32 width
    field int32 height
    field int32 step
    field uint8[] data
end struct

struct WebcamImage_size
    field int32 width
    field int32 height
    field int32 step
end struct

object Webcam
    property string Name
    function WebcamImage CaptureFrame()

    function void StartStreaming()
    function void StopStreaming()
    pipe WebcamImage FrameStream

    function WebcamImage_size CaptureFrameToBuffer()
    memory uint8[] buffer
    memory uint8[*] multidimbuffer

end object

object WebcamHost
    property string{int32} WebcamNames
    objref Webcam{int32} Webcams
end object
```

1 and 2 are example service definitions. See [9] for more information about these concepts.

RR supports a wide variety of data types, also referred to as "value types". The basic data types are called "primitives" and consist of floating point numbers of various precision, integers of various precision, and strings. Numeric primitives can be combined into arrays of single dimension or multiple dimensions. All data types can be combined into lists or maps. Maps can use integer or string keys. Structures are defined in service definitions and combine any data type (including other structures) into familiar organizational units. Finally, the "varvalue" keyword can be used to allow any allowed data type to be transmitted or received with its type determined at runtime. These data types are discussed in detail in [9].

An *object* is roughly defined as a collection of *members* that provide functionality acting on the *object*. Many programming languages have members of types: properties, functions, and events. RR uses an "Object Oriented Model" that has eight member types. These member types are necessary because of the latency in communication that exist between client and service. In a standard programming language where a function call has effectively zero latency, the members can all be emulated easily using just functions. With the latency of the communication, it becomes very difficult to emulate the members and requires significant "boilerplate code". Some members like the "wire" may have poor performance. The organization of the members is also useful in creating friendly programming interfaces that require minimal if any "boilerplate code".

Properties (Keyword: `property`)
: Properties are similar to class variables (field). They can be written to (set) or read from (get). A property can take on any value type.

Functions (Keyword: `function`)
: Functions take zero or more value type parameters, and return a single value type. The parameters of the functions must all have unique names. The return value of the function may be `void` if there is no return.

Events (Keyword: `event`)
: Events provide a way for the service to notify clients that an event has occurred. When an event is fired, every client reference receives the event. The parameters are passed to the client. There is no return.

Object References (Keyword: `objref`)
: A service consists of any number of objects. The *root object* is the object first referenced when connecting to a service. The other object references are obtained through the `objref` members. These members return a reference to the specified object. Because RR has a strong distinction between data and objects, only an `objref` can return a reference to another object. `objref` can be defined as a specific type or have type "varobject" which is determined at runtime.

Pipes (Keyword: `pipe`)
: Pipes provide full-duplex first-in, first-out (FIFO) connections between the client and service. Pipes are unique to each client, and are indexed so that the same member can handle multiple connections. The pipe member allows for the creation of "PipeEndpoint" pairs. One endpoint is on the client side, and the other is on the server side. For each connected pipe endpoint pair, packets that are sent by the client appear at the service end, and packets that are sent by the service end up on the client side. Pipes are useful for streaming data where each packet is important. It can also be used to transfer large amounts of data in sequential packet form. If only the most current value is needed, a "wire" member can be used instead.

Callbacks (Keyword: `callback`)
: Callbacks are essentially "reverse functions", meaning that they allow a service to call a function on a client. Because a service can have multiple clients connected, the service must specify which client to call.

Wires (Keyword: `wire`)
: Wires are very similar to pipes, however rather than providing a stream of packets the wire is used when only the "most recent" value is of interest.

It is similar in concept to a "port" in Simulink. Wires may be transmitted over lossy transports or transports with latency where packets may not arrive or may arrive out of order. In these situations the lost or out of order packet will be ignored and only the newest value will be used. Each packet has a timestamp of when it is sent (from the sender's clock). Wires are full duplex like pipes meaning they have two-way communication. An example use of a wire is to transmit the current angles of the joints of a robot.

Memories (Keyword: `memory`)
: Memories represent a random-access segment of numeric primitive arrays or numeric primitive multi-dim arrays. The memory member is available for two reasons: it will break down large reads and writes into smaller calls to prevent buffer overruns (most transports limit message sizes to 10 MB) and the memory also provides the basis for future shared-memory segments.

RR has support for inheritance and importing existing service definitions and using the imported types. This allows for forward and backward compatibility. Newer devices can add members while maintaining compatibility with an existing object type. If a client connects that does not understand the new object type, it can fall back to the older data type and still communicate with the device. This functionality also allows for the addition of vendor-specific features while maintaining compatibility with a widely understood standard object type.

Most of the functions in the RR core library that block during network activity can also run in asynchronous mode. When using the asynchronous mode, the command is started and provided with a handler function. The current thread will then return immediately. Once the activity has been completed, the handler function is called using a thread from the thread pool. Most object members can be used asynchronously where blocking would result due to network activity. Because of this asynchronous design, RR can interact with numerous nodes simultaneously with limited usage of computational resources. The exact number of maximum concurrent devices is unknown, however it is estimated that RR can interact with thousands of devices simultaneously.

III. Transports

RR uses pluggable transports to connect nodes. The currently supported transports are `TcpTransport`, `LocalTransport`, and `HardwareTransport`. The `TcpTransport` provides connection over standard network technology using IPv4 or IPv6. Auto-discovery is implemented using UDP. The TCP transport can be secured using TLS to prevent eavesdropping and to verify node identity. The TCP stream is *upgraded* after connection to TLS using StartTLS to trigger the upgrade. TLS uses X509 certificates issued by Wason Technology, LLC to validate the identity of the nodes. These certificates are tied to a 128-bit UUID [12]. By checking the certificate, a client can determine that the service is the expected device. The certificates can also be used to verify client identity to provide strong certificate based authentication. The `TcpTransport` can also create and accept WebSocket connections. WebSockets are a new HTTP feature that allow for persistent connections between clients and HTTP servers. They are supported by standard web browsers and web servers. By using the WebSocket support, standard web browsers that support HTML5/JavaScript can connect to RR nodes listening for TCP connections. Clients can then be developed using pure JavaScript. JavaScript programs have the advantage of being easier to deploy and highly compatible with different platforms, including mobile devices. Using WebSockets, it is also possible to embed an RR service inside an ASP.NET web server. ASP.NET is capable of accepting WebSockets. These WebSockets can then be passed to the RR node to be accepted as an incoming client connection. The use of WebSockets allows RR to be compatible with existing web technology.

The `LocalTransport` is used to communicate between nodes on the same machine. It uses operating system level inter-process communication (IPC) methods and does not use the loopback networking interface.

The `HardwareTransport` allows communication with USB, PCIe, and industrial devices. Producing devices compatible with RR requires licensing from Wason Technology, LLC. Currently only USB and PCIe are supported.

IV. Compatibility

RR was developed to assist in the development of a number of complex systems involving multiple devices developed by different vendors connected to computers in a network. The computers ran a variety of operating systems and the drivers for these devices were available in a variety of languages. These systems are discussed in [1]. These systems rapidly evolved and the communication interfaces between subsystems constantly changed. The available communication technology did not have the correct combination of compatibility, data type support, communication function support (functions, wires, callbacks, etc.), platform support, language compatibility, runtime type detection, and low latency. *RR was developed with the goal of maximizing compatibility between devices whenever possible.*

As discussed in the Introduction, RR is capable of running on Windows, Linux, Mac OSX, iOS, Android, and HTML5/JavaScript on a variety of processor architectures. RR has proven to be very effective running on the Raspberry Pi [13] which has been used extensively as a wireless sensor interface. It currently supports C++, C#, Java, Python, and JavaScript programming languages. RR has also been demonstrated running on the Arduino [14] microcontroller family and Particle Photon wireless microcontroller [15] using a custom bare-metal implementation, providing the opportunity for low cost, low power devices. The compatibility of RR is being constantly expanded by adding support for more operating systems and programming languages.

V. Real-time capability

The current RR library is capable of limited soft real-time communication utilizing the "wire" element. The "wire" element will transmit the most recent value to the connected node. The communication can be both client-to-service and service-to-client. Because of the design of traditional operating systems, the design of traditional networking technology, and the asynchronous design of the current software library it is not possible to guarantee hard real-time communication. The thread scheduling mechanisms in use cannot guarantee exact execution time [16] [17]. (While Windows has a "Real-Time" thread priority, this scheduling level is not accurate enough for control applications.) Implementations of the RR communication system based on a different design have been demonstrated on both xPC Target [18] and Linux PREEMPT_RT [19], with the communication software running in a hard real time context. However, both of these examples still used Ethernet as a transport mechanism and could not guarantee real-time communication. Future work will utilize hard real-time networking to provide true hard real-time communication between nodes.

VI. Experimental Results

To test the latency of the RR communication, a simple service was designed with a single function. This function is essentially empty, and does not impose any delay. The function was called 100,000 times with different configurations on both Windows 10 and Ubuntu 14.04 using high end Intel i7 processors. `LocalTransport` and `TcpTransport` were tested. The `TcpTransport` was tested using the loopback interface and over Ethernet, and also using TLS. The results are compared to ROS running in TCP loopback mode using Ubuntu 14.04. The test service is based on the examples on the ROS website [20]. The results are listed in Table I. Selected plots are shown in Figures 1, 2, and 3.

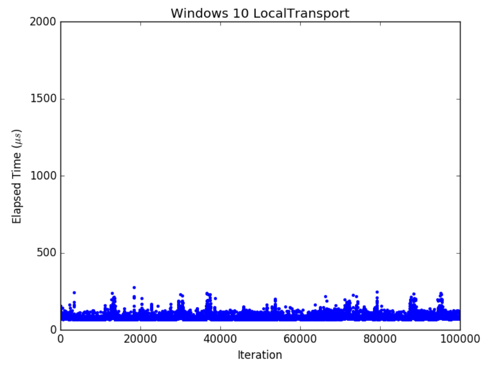

Fig. 1. Windows 10 LocalTransport Function Call Elapsed Time

The experimental results show that the mean time for the function call over `LocalTransport` is 77.1 μs on Windows and 136.6 μs on Ubuntu 14.04. The delay is mostly due to the thread switching that occurs during communication. The discrepancy between Windows and Ubuntu is due to the difference in the time required for the thread switches.

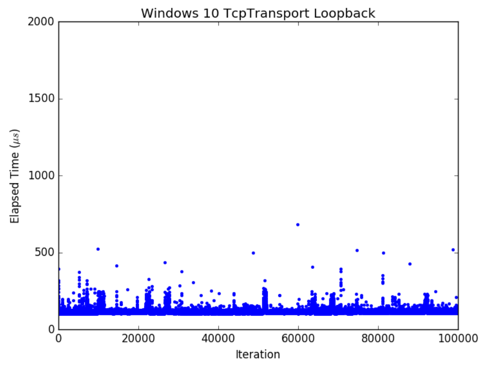

Fig. 2. Windows 10 TcpTransport Loopback Function Call Elapsed Time

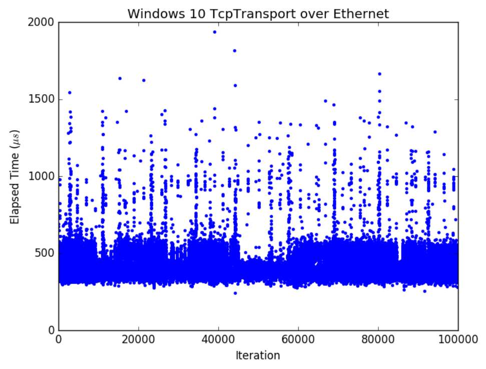

Fig. 3. Windows 10 TcpTransport Ethernet Function Call Elapsed Time (Some outliers off scale, see Table I)

Using the TCP loopback, Ethernet, and TLS all require more time than the `LocalTransport`. These results will vary depending on the exact equipment used to run the experiments. Results will vary depending on the equipment utilized. Because of clock skew between computers and processes it is not possible to directly measure the latency of the "wire" member. The "wire" is expected to have approximately half the latency of the function call due to the lack of the return transmission. As can be seen in Table I RR has roughly ten times lower latency than ROS for service calls. DDS was not available for comparison.

VII. License

RR is distributed under a commercial license that is royalty free for most uses. The exceptions are that certificates must be purchased from Wason Technology, LLC for each node to enable TLS communication, and hardware devices must purchase licensing from Wason Technology, LLC to connect to the driver. The software is not currently open source. The option of open sourcing the software was considered carefully, however it has not been released because of the risk of *ecosystem fragmentation*. RR is a technology that will be widely distributed over many devices that all must remain inter-operable under any circumstances. Open source projects have a tendency to be "forked" into different versions. For many projects this is not a problem as the changes will

TABLE I
COMPARISON OF FUNCTION CALL ELAPSED TIME

Configuration	Mean (μs)	Min (μs)	Max (μs)	σ
Windows 10 `LocalTransport`	77.1	66.4	275.9	8.9
Windows 10 `TcpTransport` Loopback	113.1	102.2	683.7	10.5
Windows 10 `TcpTransport` with TLS Loopback	141.2	128.6	931.8	12.6
Windows 10 `TcpTransport` over Ethernet	417.4	243.5	3754.9	84.7
Windows 10 `TcpTransport` with TLS over Ethernet	492.7	306.0	5859.8	79.9
Ubuntu 14.04 `LocalTransport`	136.6	126.8	956.3	8.2
Ubuntu 14.04 `TcpTransport` Loopback	166.7	151.3	1674.1	8.8
Ubuntu 14.04 `TcpTransport` with TLS Loopback	278.0	264.1	1481.4	7.9
Ubuntu 14.04 `TcpTransport` over Ethernet	518.0	303.3	7449.7	32.6
Ubuntu 14.04 `TcpTransport` with TLS over Ethernet	687.3	477.4	1605.7	47.5
Ubuntu 14.04 ROS TCP Loopback	1848.8	1362.7	14325.544	883.2

not diminish the functionality of the project. Examples of fragmentation can be found in the Bitcoin [21] network and the Android [22] ecosystem. For RR, even slight changes can severely compromise the ability of devices to communicate. A small change can result in intermittent failures that can be very difficult to troubleshoot and could damage the usefulness of the overall ecosystem. Another risk is the ”Embrace, Extend, Extinguish” strategy used in the past by Microsoft and other companies [23]. Open sourcing the software would unfortunately open up the technology to these threats. If a defense against these risks can be developed the software may be open sourced.

VIII. EXAMPLES

A. *Quick Start Sample*

Example 3 is a minimal Python example service that can drive the iRobot Create using the serial control cable. This is a subset of the more capable example service in Example 1. It has a single function “Drive” that takes parameters “velocity” and “radius”. This example only uses one of the eight member types that RR offers. The example begins by importing the RR module. It then contains the service definition and an implementation of the “create_impl” class. Next, the “TcpTransport” is initialized and begins listening on port 52222. Finally, the service type and object implementing the service object is registered as a service that can be accessed by a client. These basic steps are similar for all services.

Example 4 is a minimal Python example client that will drive the robot for a short distance. It is very simple compared to the service example. First, the module “RobotRaconteur.Client” is imported. This registers the default transports and returns the variable “RRN”, which contains the default node. The default node is then used to connect the service, and the “Drive” function can be accessed to drive the robot.

Example 5 is the example client using MATLAB instead of Python. It is very similar to the Python example.

B. *iRobot Create improved with Raspberry Pi and webcams*

Figure 4 is a picture of the example robot used in the documentation for RR. It is an iRobot Create improved with a Raspberry Pi, power converter, webcams, and a mast to

Example 3 Minimal Python service example

```
import RobotRaconteur as RR
RRN=RR.RobotRaconteurNode.s
import threading
import serial
import struct

minimal_create_interface="""
service experimental.minimal_create

object create_obj
    function void Drive(int16 velocity, int16 radius)
end object
"""

class create_impl(object):
    def __init__(self, port):
        self._lock=threading.Lock()
        self._serial=serial.Serial(port=port,baudrate
            =57600)
        dat=struct.pack(">4B",128,132,150, 0)
        self._serial.write(dat)

    def Drive(self, velocity, radius):
        with self._lock:
            dat=struct.pack(">B2h",137,velocity,radius)
            self._serial.write(dat)

#Create and register a transport
t=RR.TcpTransport()
t.StartServer(52222)
RRN.RegisterTransport(t)

#Register the service type
RRN.RegisterServiceType(minimal_create_interface)

create_inst=create_impl("/dev/ttyUSB0")

#Register the service
RRN.RegisterService("Create","experimental.minimal_create.
     create_obj",create_inst)

#Wait for program exit to quit
raw_input("Press enter to quit")
```

Example 4 Minimal Python client example

```
from RobotRaconteur.Client import *
import time

#RRN is imported from RobotRaconteur.Client
#Connect to the service.
obj=RRN.ConnectService('rr+tcp://localhost:52222/?service=
     Create')

#The "Create" object reference is now available for use
#Drive for a bit
obj.Drive(100,5000)
time.sleep(1)
obj.Drive(0,5000)
```

Example 5 Minimal MATLAB client example

```
o=RobotRaconteur.Connect('rr+tcp://localhost:52222/?
        service=Create');
o.Drive(int16(100),int16(5000));
pause(1);
o.Drive(int16(0),int16(0));
```

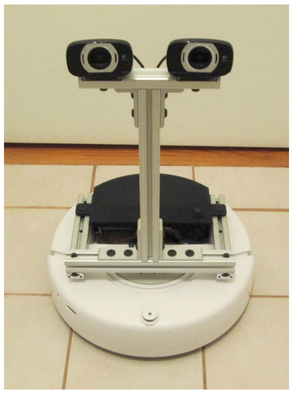

Fig. 4. iRobot Create with Raspberry Pi and webcams

hold the cameras and electronics. Examples 1 and 2 are the service definitions of the services used to access the on-board devices.

C. Microassembly System

The Multi-probe Microassembly Testbed [24] was an experimental system designed to manipulate parts that were 25 μm thick and had lengths and widths between 200 μm and 1500 μm. The system had 24 manipulation actuators, four microscope cameras, three camera actuators, and three illumination systems. These devices were controlled through four PCs and several microcontrollers. The development of this complex, constantly changing system was the primary motivation for the development of RR as the existing communication technologies were not flexible enough to keep up with the demands of the communication requirements while still being easy to rapidly modify as experiments progressed. Figure 5 shows the microassembly testbed work area, while Figure 6 shows the communication layout of the system. RR was critical to the success of the microassembly research as it significantly reduced the development time for system modifications yet allowed for high performance communication between system components.

D. Smart Conference Room

The Smart Conference Room [25] is a functional testbed for the development of advanced lighting and building control algorithms. It is located at the Smart Lighting Engineering Research Center (ERC), Rensselaer Polytechnic Institute (RPI), Troy, NY. Figure 7 shows the room with color-tunable LED lights. Figure 8 is a picture of one of the color sensors. The color sensor is a Raspberry Pi with an I^2C color sensor. The Raspberry Pi runs a Robot Raconteur service that provides access to the color sensor data.

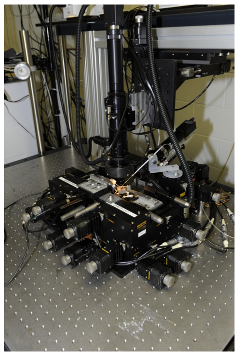

Fig. 5. Multi-probe Microassembly Testbed

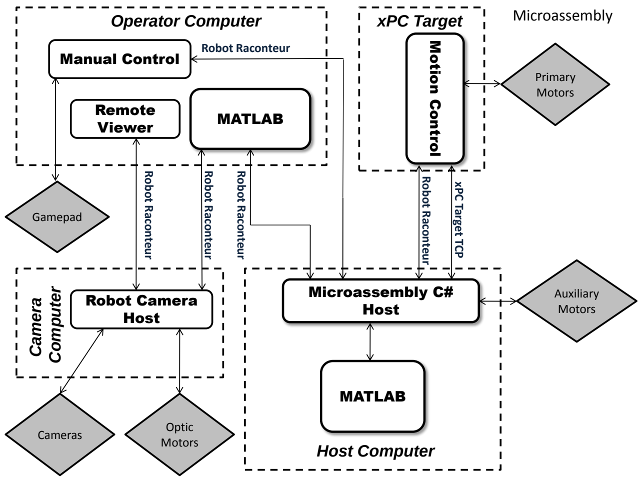

Fig. 6. Multi-probe Microassembly Testbed Communication Layout

E. Baxter on Wheels

The "Baxter on Wheels" [26] (BOW) (formerly the Jamster) is a ReThink Robotics Baxter combined with an electric wheelchair to provide mobility (Figure 9). It has been used in combination with a Jamboxx controller to allow disabled persons to operate the robot. It is developed by the Center for Automation Technologies and Systems (CATS), Rensselaer Polytechnic Institute, Troy, NY.

IX. Conclusion

RR version 0.8 is the first "beta" version ready for commercial use. It uses an "Augmented Object Oriented" data model that provides excellent flexibility and functionality. Different transports can be used to provide connection between nodes using different technologies. TLS provides network security when using the `TcpTransport`. RR is constantly being improved. New features being developed include real-time support over PCI express, a transport based on the "cloud" that will use the RR servers to help create connections, and additional language bindings. A plugin for the Gazebo robot simulator [27] is being developed to allow control of the simulator using a friendly object oriented interface. This plugin will be available on the RR website when it has been completed.

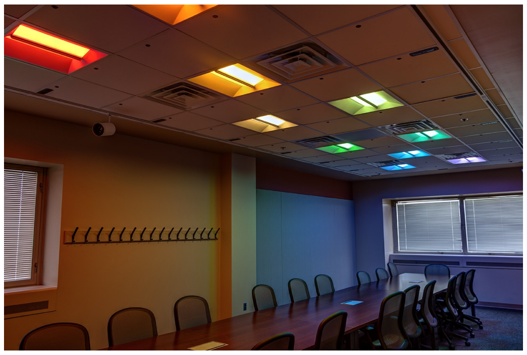

Fig. 7. Smart Conference Room, Smart Lighting ERC [25]

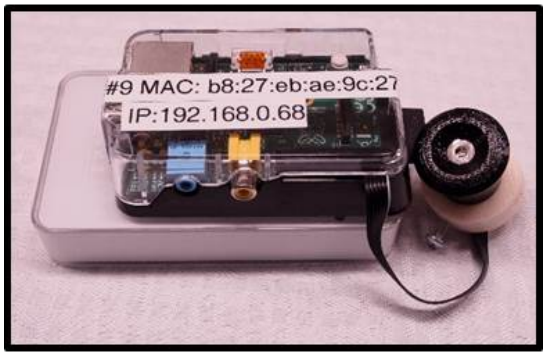

Fig. 8. Smart Conference Room color sensor, Smart Lighting ERC [25]

Creating a standards committee and industrial consortium to further the development of RR is being investigated. Please contact the author for more information on these organizations.

References

[1] J. D. Wason and J. T. Wen, "Robot raconteur: A communication architecture and library for robotic and automation systems," in *IEEE Conference on Automation Science and Engineering (CASE)*, 2011, pp. 761–766.

[2] A. Wollrath, R. Riggs, and J. Waldo, "A distributed object model for the Java TM system," *Computing*, vol. 9, no. 4, pp. 265–290, 1996.

[3] S. Vinoski, "Distributed object computing with CORBA," *C++ Report*, vol. 5, no. 6, pp. 32–38, 1993.

[4] R. H. Arpaci-Dusseau, *Introduction to Distributed Systems*. Arpaci-Dusseau Books, 2014.

[5] J. D. Wason, "System and method for implementing augmented object members for remote procedure call," U.S. patent application US20 150 081 774A1, 2013.

[6] *The Websocket Protocol*, IETF Std. RFC 6455, 2011.

[7] Boost C++ Libraries. [Online]. Available: http://www.boost.org/

[8] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and A. Ng, "ROS: an open-source Robot Operating System," in *International Conference on Robotics and Automation*, 2009.

[9] J. D. Wason, *Introduction to Robot Raconteur using Python*, Wason Technology, LLC, October 2014. [Online]. Available: https://robotraconteur.com/documentation

[10] F. Curbera, M. Duftler, R. Khalaf, W. Nagy, N. Mukhi, and S. Weerawarana, "Unraveling the web services web: an introduction to soap, wsdl, and uddi," *IEEE Internet computing*, vol. 6, no. 2, p. 86, 2002.

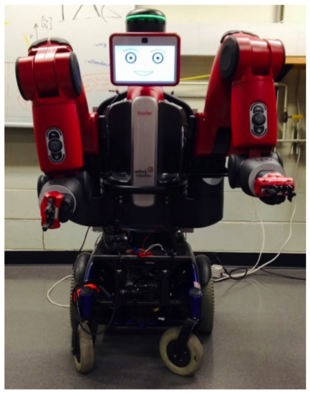

Fig. 9. Baxter on Wheels, Center for Automation Technologies and Systems [26]

[11] J. N. Scott McLean and K. Williams, *Microsoft .NET Remoting*. Microsoft Press, 2002.

[12] *Information technology - Procedures for the operation of object identifier registration authorities: Generation of universally unique identifiers and their use in object identifiers*, ITU Std. X.667, October 2012.

[13] Raspberry pi foundation. [Online]. Available: https://www.raspberrypi.org/

[14] Arduino AVR Prototyping Platform. [Online]. Available: http://www.arduino.cc/en/

[15] Particle - build your interenet of things. [Online]. Available: http://www.particle.io

[16] D. Solomon and M. Russinovich, *Windows Internals*, 5th ed. Microsoft Press, 2009, ch. 5, pp. 391–444.

[17] L. Trung. (2012, April) Comparing real–time scheduling on the Linux kernel and an RTOS. embedded.com. [Online]. Available: http://www.embedded.com/design/operating-systems/4371651/Comparing-the-real-time-scheduling-policies-of-the-Linux-kernel-and-an-RTOS-

[18] Simulink Real-Time. [Online]. Available: http://www.mathworks.com/products/simulink-real-time/

[19] CONFIG PREEMPT RT Patch. [Online]. Available: https://rt.wiki.kernel.org/index.php/CONFIG_PREEMPT_RT_Patch

[20] ROS.org Writing a Simple Service and Client (C++). [Online]. Available: http://wiki.ros.org/ROS/Tutorials/WritingServiceClient(c++)

[21] D. Floyd, "The hard fork: Will Bitcoin XT take?" *Nasdaq*, August 2015. [Online]. Available: http://www.nasdaq.com/article/the-hard-fork-will-bitcoin-xt-take-cm512449

[22] N. Swanner, "This is what android fragmentation looks like in 2015," *The Next Web*, August 2015. [Online]. Available: http://thenextweb.com/insider/2015/08/05/this-is-what-android-fragmentation-looks-like-in-2015/

[23] "Deadly embrace," *The Economist*, May 2000. [Online]. Available: http://www.economist.com/node/298112

[24] J. D. Wason, "Visually-guided multi-probe microassembly of spatial microelectromechanical systems," Ph.D. dissertation, Rensselaer Polytechnic Institue, Troy, NY, 2011.

[25] S. Afshari, S. Mishra, A. Julius, F. Lizarralde, J. D. Wason, and J. T. Wen, "Modeling and control of color tunable lighting systems," *Elsevier Energy and Buildings*, vol. 68, pp. 242–253, 2014.

[26] A. Cunningham, W. Keddy-Hector, U. Sinha, D. Whalen, D. Kruse, J. Braasch, and J. T. Wen, "Jamster: A mobile dual-arm assistive robot with jamboxx control," in *IEEE International Conference on Automation Science and Engineering (CASE)*, 2014, pp. 509–514.

[27] N. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator," in *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, vol. 3, 2004, pp. 2149–2154.