2011 IEEE International Conference on
Automation Science and Engineering
Trieste, Italy - August 24-27, 2011

Robot Raconteur: A Communication Architecture and Library for Robotic and Automation Systems

John D. Wason and John T. Wen

Center for Automation Technologies & Systems
Rensselaer Polytechnic Institute, Troy, NY 12180
{wasonj,wenj}@rpi.edu

Abstract— **Robot Raconteur is a new distributed communication architecture and library designed for robotic and automation systems with distributed resources, including data and program modules. The motivation for this architecture is based on the need to rapidly connect sensors and actuators distributed across a network together in a development environment, such as MATLAB, without time consuming development of data communication infrastructure. The architecture consists of interconnected nodes, communicating through message passing. Each node is typically a process running on a computer or embedded device, which may be a critical real-time, non-critical real-time, or event driven process. Robot Raconteur is organized as three hierarchical levels: channels that provide communication between nodes, message passing which routes messages between endpoints within the nodes, and an object-based client-service model that is built on top of message passing. The implementation of Robot Raconteur nodes so far consists of Microsoft C#, Microsoft C++, MATLAB, MATLAB/Simulink xPC Target, and the Arduino embedded processor. Implementation on four distributed control systems consisted of multiple sensors, actuators, and computation nodes are presented, including smart room (an instrumented room with distributed lighting control and sensor feedback), dual-arm robotic system, multi-probe microassembly station, and adaptive optical scanning microscope.**

I. Introduction

This paper presents a new communication architecture named Robot Raconteur (RR) for systems containing distributed resources such as sensors and actuators. Modern automation systems are typically highly distributed between multiple computers, processes within a computer, and embedded modules, supporting sensing, actuation, and computation needs. Communication between elements within this distributed system can become challenging due to the multitude of communication technologies and protocols used in the various devices. While there are some standard protocols and communication libraries, they are typically not uniformly supported across a large class of devices, software packages, and languages. It is often necessary to implement custom interface software which is time consuming and requires specialized skills. RR is an open source [1] implementation of a communication specification and reference libraries intended to provide consistent, device and language neutral communication for a large class of devices. It is currently implemented for some standard scenarios but the design is open source, and is intended to be extensible.

Currently available communication mechanisms generally fall into three categories: transport level, message passing, and remote procedure calls. Transport level is simply a pipe to send data streams or packets without any formatting specification, such as serial ports or TCP/IP. The next level is message passing that adds structure to the packets to define the content but still requires the user software to build and send the messages. The Neutral Messaging Language (NML) developed at National Institute of Standards and Technology (NIST) [2] and Robot Operating System (ROS) [3] are examples. (ROS provides additional software level control, while RR is simply a communication paradigm.) The highest level is remote procedure call (RPC) that attempts to expose functions or in many cases full objects across a process or network boundary without the user software being aware of the boundary. This type of RPC has been implemented by a number of previous technologies including Microsoft DCOM [4], Microsoft .NET Remoting [5], CORBA [6], and JAVA RMI [7]. RR consists of a message passing system and a limited RPC layer built on top of it. It provides a thin RPC layer by specifying the precise types of data and objects that can be exposed. Most of the alternative technologies listed above attempt to create a highly transparent boundary between objects through complex serialization and cross-process garbage collection, and tend to be language dependent because of this generality (CORBA is the one exception but is notoriously complex). The advantage of RR is that it exposes a powerful interface that allows the user software to ignore most of the communication specifics while remaining simple and compact enough to be easily extended to new computer or embedded systems.

The central component of Robot Raconteur is message passing between independent nodes. A node is typically a process running on a computer or an embedded device. Within

This work is supported in part by the Center for Automation Technologies and Systems (CATS) under a block grant from the New York State Foundation of Science, Technology and Innovation (NYSTAR) and in part by the National Science Foundation (NSF) Smart Lighting Engineering Research Center (EEC-0812056).

978-1-4577-1732-1/11/$26.00 ©2011 IEEE

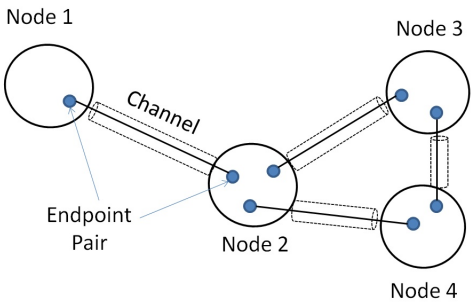


Fig. 1. Layout of the Robot Raconteur message passing system

Fig. 2. Camera Outputs of the Smart Room

each node there are endpoints that uniquely connect to an endpoint in another node (endpoints can only be paired). Nodes send messages from a start endpoint in one node to the end-endpoint in another node. Messages contain routing information and data serialized in a specific format. Channels connect nodes together, and the routing information in the message is used by the node to send the message through the correct channel. This basic architecture is illustrated in Figure 1.

Built on top of the message passing layer is a client-service model that is used to expose an object-based interface of a service to a remote client. Primitives, structures, and collections are all passed by serialization between the client and server. Object references are always references to the services object, and are connections to the remote node. When the client calls a property or function, the parameters are packed, sent to the service, executed, the return is packed, and sent to the client. When an event is fired by the service, it is packed and sent to all clients. The structures and objects exposed by a service are defined in a "service definition" file, similar to Interface Description Language (IDL) [4], [6]. When a client connects to a service, a plain text version of the service definition is sent to the client to assist in communication with the service. This paradigm provides an easily implemented yet powerful interface without the burden of a seamless object-oriented environment between client and server.

This paper will describe the main components of the RR architecture, and discuss the implementation on four distributed control systems, including the smart room for lighting control, dual arm testbed for multi-robot coordination with motion and force feedback, microassembly testbed with multi-probe coordination with motion and vision feedback, and adaptive scanning optical microscope with coordinate scanning mirror and adaptive mirror motion and image feedback. We will use smart room as an illustrative example throughout the paper.

II. Robot Raconteur Architecture

The RR architecture is based on the client-server model supported by message passing between nodes connected by channels. This section describes the key components of the architecture. We will use a distributed lighting control testbed (called "smart room") as an illustrative example. This system consists of multiple RGB tunable light and multiple sensors (cameras and RGB sensors). The outputs from four ceiling cameras are shown in Figure 2. We will consider the scenario where a remote MATLAB session accesses the sensor data from the smart room, processes the data, and send the lighting control command to the lights in the room. A host computer in the room interfaces with the sensors and lights, and is accessible through TCP/IP.

Client-Service Operation

RR implements an object based client-service model. A service consists of a base object reference which contains properties, functions, events, and references to other objects. The object and structure members are defined in a "Service Definition" file.

For the smart room, the MATLAB session is the client node, and the server node is the host computer that interfaces with the sensors and lights. The server node listens to requests on a specified port. When the MATLAB client requests the host service (sensor/actuator interface) from the server node (through TCP/IP channel communication), the server node returns the service definition. A (stripped-down) service definition example for the smart room is shown in Example 1.

When the MATLAB client connects to the service, an endpoint is created within the client node that is used for that client object reference. Any object references that are returned from the first object reference also use this endpoint. This endpoint is called the "Client Context" and contains the ability to find object references, process transaction requests for functions and properties, and dispatch events received from the service. On the service side, there is a "Service Context" which manages references to the "CameraHost" object and all its object references. The service creates an endpoint for each client connection since there can be multiple clients connection to one service context. See Figure 3 for a depiction of this client-server architecture. All the endpoints, client contexts,

Example 1 An example Service Definition file

```
service RobotCameraHost_interface
struct CameraBitmap
  field int32 width
  field int32 height
  field uint8[] data
end struct
object RobotDevice
  property string Name
  function double[] GetProperty(string name)
  function int32 SetProperty(string name, double[] val)
end object
object RobotCamera
  property string Name
  function double[] GetProperty(string name)
  function int32 SetProperty(string name, double[] val)
  function CameraBitmap GetCurrentStreamFrameCameraBitmap()
end object
object CameraHost
  objref RobotDevice[] robdevice
  objref RobotCamera[] robcamera
  property int32 RobotCameraCount
  property int32 RobotDeviceCount
end object
```

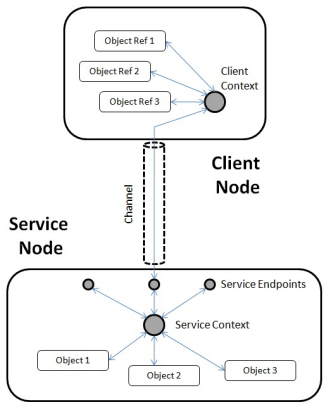


Fig. 3. Configuration of Client-Service communication

and service contexts exist within the node. Endpoints define connections between nodes through channels.

To facilitate the variation in member names and parameters between different types of objects, interface code is automatically generated. This code is often called "serialization" or "thunk" code [5], [7]. This generated code takes the member operations and parameters, and packs or unpacks them into or out of messages. For languages like C# that are strongly typed this code is generated at compile time. For languages like MATLAB that are weakly typed this code is generated as needed. The automatic generation saves the developer

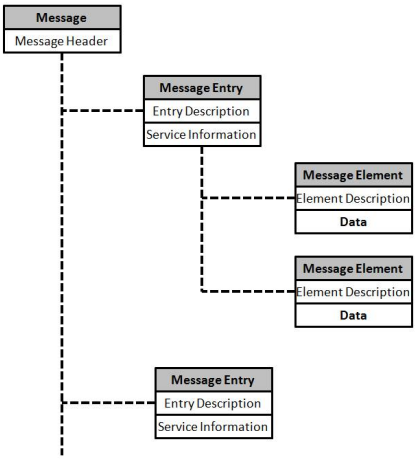


Fig. 4. Hierarchal contents of a message

significant time and makes using RR inexpensive in terms of development effort. The interfaces can also be written manually if necessary as the code is open source with well defined serialization format.

Message Contents

An RR message contains routing information followed by the data. The header contains protocol information, routing information, the number of message elements in the message, and the total size of the message. A message contains one or more specific message elements that communicate a piece of data or a command. A specific type of message element is a "service call" message element. These elements contain extra data to call a property or function of a specific object within a service, send an event from service to client, or manage connections to a service. The lowest level within a message is a "message element". A message element contains the actual data, and can be either a primitive, structure, or collection type. See Figure 4.

Channels

Channels carry message between nodes. They are seperate from endpoints, meaning that a single channel can be used to connect multiple endpoint pairs, or there may be multiple paths for a message to take to travel between two endpoints. The routing assumes that nodes are directly connected by a channel to communicate. Currently, there are three channel types implemented: TCP/IP channel for communicating over a network or the Internet, a Windows named pipe channel for communicating between processes on the same computer, and a serial port channel for communicating with low computational-power embedded devices such as the Arduino module [8]. While there is a published protocol for transmitting messages it is not necessary for channels to use the specified protocol. Different technologies may require specific protocol formats, and this is left to the

channel to determine. For example, the serial channel uses a highly simplified version of the protocol to communicate with embedded devices that are not capable of processing the standard protocol.

Smart Room Example

From MATLAB, the command below issues a client requests to connect to the service definition for the service path `CameraHost`:

```
smartroom=RobotRaconteur.Connect
('tcp://smartroom:6257
/RobotCameraHost/CameraHost');
```

Once the MATLAB client establishes connection to the service, RobotCameraHost, a MATLAB class `smartroom` is generated and returned, which sends all client requests to the objects at the `CameraHost` service path.

To access lights, we use the command below to request an object reference that corresponds to the lighting control on the server node:

```
lights=smartroom.get_robdevice(int32(0));
```

This client request gets translated to a server side request for the object type at the service path `CameraHost.robdevice[0]`. It then registers to receive events from this object. The class, `light`, is returned in MATLAB that references this service path. Similarly, the sensor access is established through:

```
camera1=smartroom.get_robcamera(int32(0));
```

To set the lighting level, the client sends a function call to the object reference that was requested in the previous commands. The parameters are packed into a message as message elements. The message entry tells the service that it is a function call, the service path of the reference object, and the name of the member (function) that is being called. The return is packed into a message and sent to the client. An example lighting control command for setting RGB intensity for lights 1 and 2 to (0.7,0.3,0), we use the command:

```
lights.SetProperty('SetRGBRange',
[1,2,0.7,0.3,0]');
```

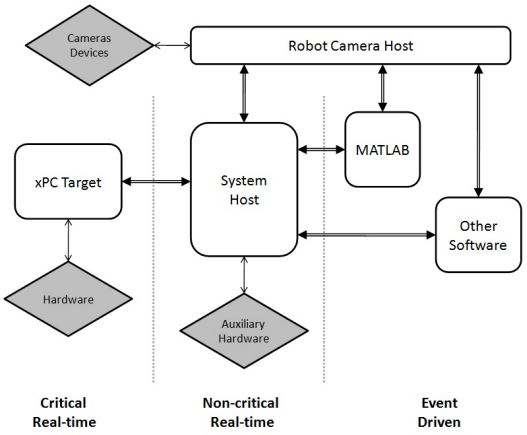
Similarly, to acquire a bitmap image, we use

```
I=CameraBitmapToImage
(cam.GetCurrentStreamFrameCameraBitmap);
```

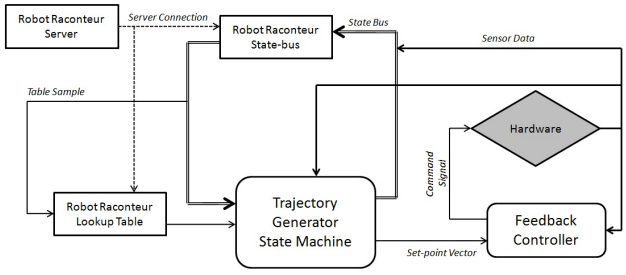
III. Example Implementation

Robot Raconteur has been used to implemented on a number of different distributed robotic and automation systems at Rensselaer Polytechnic Institute: dual arm system, adaptive scanning optical microscope (ASOM), microassembly testbed, and smart room. The first three contain real time components executed on the Mathworks xPC Target [9] platform. For the smart room, the operation does not involve real-time components, and RR simply allows the client control software (e.g., MATLAB) to connect to the sensors and lights in the room, read sensors, and change the lighting intensity in each color channel.

The three real-time systems use a distributed real-item model that includes three hierarchal levels: critical real-time, non-critical real time, and event driven (Figure 5(a)). The critical real-time portion is implemented on computers running the xPC Target and is typically the dynamic and motion control portion of the system. Robot Raconteur on the xPC Target systems allows for two types of objects to be exposed: state buses and lookup tables. The state bus is a Simulink bus object with bundled signals that can be read and set over RR. This state bus is usually augmented to contain signals for a state-machine trajectory controller, sensor measurements, and commands that are issued by adjusting the value of the signals. The lookup table is used to program movements by specifying positions at sampled times. The lookup table translates the current sample number to the corresponding value in the table. The RR interface allows the client to download new lookup table data during runtime to adjust the motion. Figure 5(b) is a signal flow diagram of the an xPC Target RR Simulink program.



(a) System diagram



(b) Simulink xPC Target program

Fig. 5. Signal flow diagram of overall system and real-time component

The non-critical real-time loop is typically on the Windows machine and includes operations that are more computationally intensive and do not need fast update rates, such as vision processing and trajectory generation. An event driven controller such as MATLAB or other logic/user interface

Fig. 6. Dual Arm Testbed

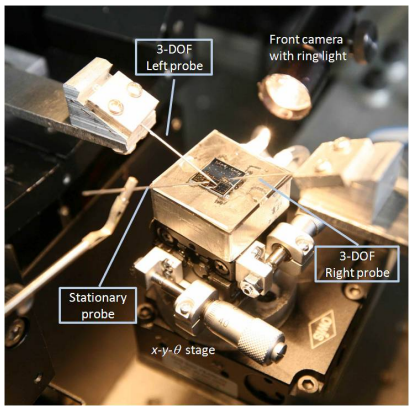


Fig. 7. Microassembly Testbed

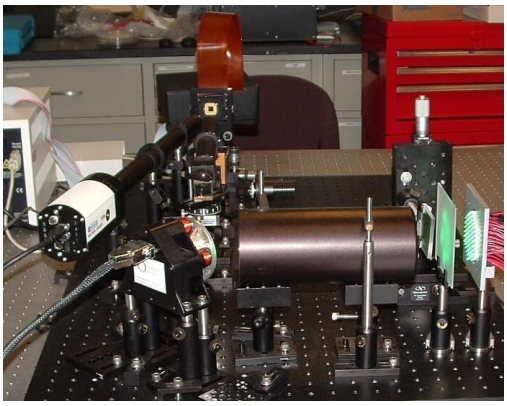


Fig. 8. ASOM Testbed

connects to this service.

Dual Arm System: The dual arm system consists of total 18 motion degrees of freedom working in coordination. The system has two 6-dof PUMA 560 robots and two 3-dof base transporters supporting the robots (Figure 6). The two PUMA robots and the two transporters (together) are controlled by three xPC Target computers (each supporting 6-dof) connected by a local area network. A fourth xPC Target machine is used for real-time coordination between the three robot computers. The xPC Target computers expose an RR service over TCP that is accessed by a separate Windows computer for event driven and non-critical real time information transfer between host and xPC Target. To communicate between xPC Target computers, UDP packets are transmitted at high rates at about 1 kHz. No UDP error correction is necessary, as each packet is independent of the previous packet and dropped packets will appear as delay in data. Due to the high update rate any delay due to dropped packet will only affect performance marginally. The packets contain time stamps so lost data stream can be quickly detected. On the Windows computer, there is a "Dual Arm System Host" program that connects to each of the xPC Target machines and adds some basic functionality, including the ability to move the robots using a wireless gamepad. MATLAB provides the high lever user interaction and programming environment.

Microassembly Testbed: The Microassembly Testbed [10] consists of two 6-dof probes (3-dof coarse stepper stage and 3-dof fine piezo positioner), a 3-dof platform, two microscope cameras, and a number of auxiliary sensors and actuators (shown in Figure 7). This system follows the same basic structure as the dual arm system, however there is only one xPC Target computer to control the motion of all axes and the host program connects to the auxiliary components as well.

ASOM: The ASOM [11] system has one camera, a two-axis fast steerable mirror, and a deformable mirror that adaptively corrects the optics of the microscope (shown in Figure 8). The scanning mirror is controlled by an xPC Target computer. On the Windows machine all three elements are controlled through plugins in "Robot Camera Host". MATLAB provides the high lever user interaction and programming environment.

Smart Room: The smart room system is controlled through a service provided by "Robot Camera Host," which in addition to supporting four cameras also has plugins for the lighting and sensor devices. Robot Raconteur allows users to connect over the Internet locally or even across the country in the MATLAB environment.

IV. Assessment of Robot Raconteur

Comparison with Other Distributed Architectures

Robot Raconteur belongs to the family of technologies facilitating communication in a distributed software environment which includes ROS, DCOM, CORBA, and others. RR is specifically designed for automation systems with distributed sensors and actuators containing real-time and event-driven processes, in contrast to other more general purpose technologies. CORBA and ROS have been commonly used in automation systems, however RR has several advantages. It creates a "soft" connection in terms of the message format and uses pluggable channels that allow for adaptation to technologies that is invisible to the end user programs. The "soft" connection format means that non-complied or weakly-typed languages utilize the requirement

that each service provide a plain-text service description that is easy to parse. The MATLAB binding for each service type is generated at runtime and does not need anything more than the URL to attach to a service and provide a powerful, fully featured interface. The pluggable channel architecture has allowed the Arduino channel to be developed in a way that is invisible to the client consuming the service, but allows for embedded software to be rapidly developed, and for new embedded services to be operated by any language that RR is implemented (assuming the channel has also been implemented for the platform). These two important features do not exist in CORBA or ROS.

Real-Time Consideration

Performance is critical to any real-time system. RR is a serialization and network protocol, and any component of this type will inherently have some overhead. Exact statistical analysis has not been conducted but anecdotal observation can provide some general understanding of performance. A transaction between the client and service over a local gigabit ethernet with TCP between two computers or over a named pipe on the same computer generally takes between 500 µs to 2 ms. The exact delay depends on the speed of the computer and network hardware. When large arrays on the order of megabytes are repeatedly sent over the network with the current C# reference library, network utilization can approach 80% of a gigabit ethernet connection. While on local ethernet networks the TCP channel usually provides excellent performance. However, an occasional lost packet can theoretically cause delays. A custom UDP channel has been considered but this would not take advantage of error correction capabilities in modern network interfaces, and also increases the CPU load. A better option is to adjust timeouts in the network interface (Windows computers do adjust network error checking parameters automatically), but this has not been implemented. If the underlying hardware and software is capable of real-time operation, real-time critical implementations can be developed within RR.

V. Conclusions

This paper presents a new distributed architecture, Robot Raconteur, to facilitate operation in a distributed automation system. It provides a simple yet sophisticated object-based client-service paradigm that enables two-way communication. At the lowest level it is a method to transmit messages between nodes through channels using either the predefined protocol or a custom channel design. The client-service model builds on top of the message passing. Interface code for specific services are automatically or dynamically generated, saving the developer from time consuming implementation of protocol software. The serialization method and service definition available at runtime without any specific *a priori* knowledge of the device means that RR will function well in a dynamic network environment. Reference implementations for Robot Raconteur have been implemented for Microsoft C#, Microsoft C++, MATLAB, MATLAB/Simulink xPC Target, and the Arduino embedded processor. Four systems are presented using RR architecture. The protocol and code are open source and extension to new software languages and devices is encouraged.

Current effort involves improving error handling, developing an ad-hoc communication method for automatic discovery and connection of nodes and available services, and supporting exclusive locks on service objects and session objects that are specific to a connection. Currently there is no language-independent method of error transmission during client-service operation. Errors thrown during operation are correctly relayed between client and service, however, the error names and error data are still language dependent. In longer term, we envision the automatic configuration of a distributed system where users may not have technical knowledge to configure a network with complex topology. Examples of similar technologies are Java JINI [12] and ad-hoc mobile networks [13].

References

[1] Robot raconteur development website. [Online]. Available: http://robotraconteur.sourceforge.net

[2] M. Moore, V. Gazi, K. Passino, W. Shackleford, and F. Proctor, "Complex control system design and implementation using the NIST-RCS software library," *IEEE Control Systems Magazine,*, vol. 19, no. 6, pp. 12–28, 2002.

[3] 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.

[4] R. Grimes and R. Grimes, *Professional DCOM programming*. Wrox Press Ltd. Birmingham, UK, UK, 1997.

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

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

[7] 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.

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

[9] MathWorks. (2010) xPC Target 4.3: Perform real-time rapid cotnrol prototyping and hardware-in-the-loop simulation. [Online]. Available: mathworks.com/mason/tag/proxy.html?dataid=13108&fileid=63506

[10] J. D. Wason, J. T. Wen, Y.-M. Choi, J. J. Gorman, and N. G. Dagalakis, "Vision guided multi-probe microassembly of 3d microstructures," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, Taipei, Taiwan, October 2010.

[11] B. Potsaid, Y. Bellouard, and J. Wen, "Adaptive scanning optical microscope (ASOM): a multidisciplinary optical microscope design for large field of view and high resolution imaging," *Opt. Express*, vol. 13, no. 17, pp. 6504–6518, 2005.

[12] W. Edwards, *core JINI*. Prentice Hall PTR, 1999.

[13] A. Boukerche, *Algorithms and protocols for wireless and mobile ad hoc networks*. Wiley-IEEE Press, 2009.