Robot Raconteur®: Updates on an Open Source Interoperable Middleware for Robotics

John D. Wason and John T. Wen

***Abstract*— Robot Raconteur® (RR) is a powerful communication framework for robotics, automation, building control, and the "Internet of Things". RR provides unique plug-and-play, standardized, augmented object-oriented capabilities that greatly accelerate the development of systems and provide stable production performance. The project became open-source in 2018, and is available on GitHub under the Apache 2.0 license. This paper provides updates on the development of RR, the standardization of RR, the development of an ecosystem of standardized drivers, and example use cases for the framework. The RR Core library is approaching a 1.0 release with ROS Quality Level 2.**

I. Introduction

Robot Raconteur® (RR) is a powerful communication framework for robotics, automation, building control, and the "Internet of Things". RR provides a combination of features that make it uniquely capable of meeting the needs of advanced automation systems as the scale and complexity of these systems increases. RR began as a simple project in 2010, and has matured into a flexible, performant, interoperable, and user-friendly suite of components. The core libraries were open-sourced in 2018 under the Apache 2.0 license to make the technology more accessible and provide confidence to users. RR has been covered in two previous CASE papers, in 2011 [1] and 2016 [2]. A patent has also been issued [3]. This paper presents updates on the development of RR, the standardization of RR, the development of an ecosystem of standardized drivers, and example use cases for the framework.

The development of RR was motivated by the difficulty of developing advanced automation and robotics systems. These systems use components that differ in manufacturer, platform, interface, and application programming interface (API) language. The components in these systems also typically have far more complex software interfaces than traditional automation components, making traditional solutions such as EtherCAT [4], Ethernet/IP [5], ModBus [6], OPC/UA [7], or digital/analog IO inadequate. The initial design of RR was inspired by object-oriented remote procedure call (RPC) frameworks such as Java RMI [8] and .NET Remoting [9], but designed to be straightforward and plug-and-play with common engineering software like MATLAB [10] and later LabView [11]. Over time and with the experience of dozens of projects, the capabilities of RR have grown into a mature design with features and standards that are unique. While RR is attempting to solve similar problems to the more common ROS [12] and ROS 2 [13], the solutions provided are quite different. (RR is a client-service RPC framework, while ROS and ROS 2 are Publish-Subscribe frameworks.) User experience has been that RR is easier to use and allows for more capable interfaces to be developed. More details on the comparison between ROS and RR can be found on the RR GitHub website [14].

A directory of links to available Robot Raconteur software and projects has been created [15]. Refer to the directory for URL links to the repositories listed in this paper.

At the time of this writing, the current RR Core release version is 0.16.0.

II. Robot Raconteur Overview

Robot Raconteur is an augmented object-oriented middleware inspired by remote procedure call (RPC) concepts but expanded to realize the needs of advanced robotics applications. The basic goal of RPC is to provide access to remote software functions, objects, members, and data as if they were local. RPC technologies are common, but most focus on business applications and have not been designed for robotics. The performance, semantics, reliability, and data types are not designed for handling the needs of advanced robotics applications, which are typically real-time, low latency, and communicate numeric data.

RR is a client-service communication framework. The service contains objects that implement the functionality of the service. For instance, a robot would be represented by one or more objects, with members that command the motion and provide state feedback. The client connects to the service, and creates "object references" (sometimes called "proxies") that have the same members as the service objects. The middleware transparently connects the client object reference and service. See [1], [2], and the project documentation for more details on the RPC implementation.

Wason Technology, LLC, wason@wasontech.com

Electrical, Computer, and Systems Engineering, Rensselaer Polytechnic Institute, wenj@rpi.edu

Funding for this research was provided in part by the ARM (Advanced Robotics for Manufacturing) Institute. The ARM Institute is sponsored by the Office of the Secretary of Defense under Agreement Number W911NF-17-3-0004. Views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Office of the Secretary of Defense or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation herein.

This work was supported in part by the New York State Empire State Development Division of Science, Technology and Innovation (NYSTAR) under contract C160142.

A. Service Definitions and Plug-and-Play

RR uses "service definitions" to define the objects, members, data types, exceptions, and constants provided by services. Service definitions are a form of interface definition language (IDL) [16]. Service definitions are transmitted to the client while connecting to a service. This allows the client to automatically generate object references in scripting languages such as Python, MATLAB, and LabView. Services can also be standardized to allow clients to communicate with any service that supports a specified service definition (See Section IV-B). This method would typically be used for production scenarios that need to interoperate with classes of devices. These two methods, runtime generated client proxies and standardized service definitions, provide plug-and-play capability [17].

B. Objects

RR objects support member types "property", "function","event", "objref", "pipe", "callback", "wire", and "memory". This is referred to as "augmented object-oriented", The "wire", "pipe", and "memory" members are designed to handle streaming, real-time, and shared-memory, respectively. These members are unique to RR. See [2] and the documentation for more details on object members.

C. Data Types

RR supports five primary categories of data types: primitives, structures, pods, containers, and namedarrays. Primitives are numeric scalars, numeric arrays, numeric multidimarrays, and strings. Supported data types for numeric types are one, two, four byte signed and unsigned integers, four, eight byte floating point reals, booleans, and eight, sixteen byte floating point complex numbers. See Section III-B for more information on the new boolean and complex number features. Structures are composite types containing one or more named fields. Containers represent lists or maps of other data types. Pods and namedarrays are composite types like structures, but store the information differently in memory. See Section III-C for more information on the new composite data type features. See the primary documentation for a full overview of data types.

D. Exceptions and Error Handling

RR has built-in exception transmission for property, function, objref, callback, and memory member types. If an exception is thrown during the request, it is passed back to the caller and rethrown. This removes the need for the developer to implement error transmission logic. Custom exceptions can be defined in service definitions using the `exception` keyword. Exception transmission is not available for the pipe and wire member since these are handling streaming data.

E. Transports

RR uses "transports" to pass messages between client and service nodes. These transports typically serialize the message to binary form. Currently supported transports use TCP/IP (`rr+tcp`), UNIX sockets (`rr+local`), intraprocess communication (`rr+intra`), USB (`rr+usb`), or Bluetooth (`rr+bluetooth`). Several transports are under development including QUIC [18] (`rr+quic`), PCIe (`rr+pci`), and Cloud using WebRTC [19] (`rr+cloud`). The TCP/IP transport can operate in several modes, including support for WebSockets [20] and encryption using TLS [21]. The use of WebSockets allows RR to communicate with existing Web and Cloud infrastructure, such as Web Browsers and ASP.NET [22] web servers.

F. Discovery and Subscriptions

RR provides discovery of services on the local system or over the network. On the local system special files are used, while on the network multicast UDP packets are used. Subscriptions utilize this discovery capability to automatically connect to services and manage the client connection lifecycle. See Section III-A.

G. Implementations

RR is a framework, with multiple software and hardware components providing implementations. The three primary software implementations are "Robot Raconteur Core", "Robot Raconteur Lite", and "Robot Raconteur Web".

RR Core is the primary software library, and implements all of the major framework features. It is written in C++ and uses the Boost ASIO [23] library. RR Core is designed for application level programming, and makes some performance trade-offs in favor of usability and dynamic type handling, although it is still performant enough for most soft real-time use cases (See Section VII). RR Core provides language wrappers for C++, Python, C#, Java, and MATLAB. A commercial RR LabView Add-on is available for licensing from Wason Technology. RR Core supports Windows, Linux, MacOS, iOS, Android, and FreeBSD. There is also a new implementation of RR Core for Pyodide [24], a Python implementation that runs in WebAssembly within a web browser. Future support for Rust, Go, Lua, QNX, and VxWorks are planned. Robot Raconteur Core is available in a variety of package managers. It is also included in the ROS Noetic [12] and ROS Humble [13] repositories.

RR Lite is a pure ANSI C implementation that is under development intended for real-time, embedded, and, in the future, safety-critical applications. While RR Core favors usability and dynamicism, RR Lite is designed purely for performance and portability. This implementation is currently under development.

RR Web is a C# implementation of the RR framework, and is used for applications, web servers, and within browsers. C# applications can use the library normally, as any other library. Web servers such as the ASP.NET [22] server use the library by accepting WebSockets, and then passing the connected WebSocket to the RR node running inside RR Web for processing. RR Web can also be transpiled to JavaScript to be run inside a Web Browser.

H. Version 1.0 Release

The RR Core library is approaching a 1.0 release with ROS Quality Level 2 [25]. The ROS quality levels are designed to provide a consistent way to judge the development maturity and quality of an open-source package. Quality Level 2 is the second highest quality level, defined as "These are packages which need to be solidly developed and might be used in production environments, but are not strictly required, or are commonly replaced by custom solutions. This can also include packages which are not yet up to 'Level 1' but intend to be in the future [25]". RR will be progressed to ROS Quality Level 1 as soon as possible after achieving ROS Quality Level 2. The 1.0 release is expected to be complete in mid to late 2023.

III. New Framework Capabilities

A. Subscriptions

Subscriptions were added in RR version 0.9.0, providing a capability that combines discovery with client connection lifecycle management. Normal discovery is a discrete operation, where the user requests a list of nodes or services that match the criteria specified. Subscriptions instead continuously monitor available nodes and services. If the critea match, the subscription will automatically create client connections to services. "Wire" and "pipe" subscriptions can also be created. These subscriptions will automatically connect to wires and pipes of connected services, and provide an aggregated representation of the received data. (Wire and pipe subscriptions are similar to pub-sub systems.) The connected client objects can be accessed, and used as normal clients. Using subscriptions removes the need to know the exact URL of a service, and simplifies the connection lifecycle management since it is no longer the responsibility of the library user.

Subscriptions can be created in three operational modes:

ServiceInfo2 Subscription
: Subscribe for `ServiceInfo2` discovery information. This mode returns discovery information and does not create connections.

Service By Type Subscription
: Subscribe to services by the root object type, and a set of filter criteria. This mode looks for a specific object type, for instance `com.robotraconteur.robotics.robot.Robot`, and connects to all services that have this service type. The criteria for connection can be modified using a filter to prevent connecting to unintended services.

Service By URL Subscription
: Subscribe to a service using a URL. This operates using the same URL that would be used for a normal client connection, but provides automatic lifecycle management.

B. Complex Numbers and Booleans

New primitive types for booleans and complex numbers were added in RR version 0.9.0. `csingle` represents an eight byte complex number, and `cdouble` represents a sixteen byte complex number. `bool` is a single byte representation of a boolean.

C. Pods and Namedarrays

Pods and namedarrays were introduced in RR version 0.9.0. These new types are composite value types that provide a different representation of data compared to the standard structure type. With the structure data type, every field is a reference (or pointer) to another piece of data stored separately from the structure. This allows for the structure to hold any valid data type, with arbitrary size. For performance and semantic reasons, this is often not the ideal representation of data. "Plain old data" [26] is a term for passive structures that contain only data, and in C++ and C are special because all of the data contents are stored within the structure itself[1]. Because all of the data is local to the structure, an array of pods will have its memory in the same contiguous space, instead of having references to other data stored separate in memory. Pods have significant advantages for real-time and constrained systems that are not allowed to allocate or manage memory. They are also more efficient for storing large amounts of simple data, since only one allocation is required for the contiguous space instead of many smaller allocations.

Namedarrays take the pod concept once step further, and consider data that can be represented either as an array or a composite type. Consider a vector, that can be represented as a 3x1 array, or as a composite type with fields x, y, and z. Both representations are valid, and are used interchangeably. A namedarray is a union type that formalizes this concept. The memory is stored as a numeric array, but can also be accessed using the composite type fields. This is a very efficient representation, since all of the data is stored in a simple array, and can also be transmitted as a simple array without further processing.

D. Function Generators

Function generators were introduced in RR version 0.9.0. Function generators are modeled after Python generators [27], and allow a function to return a simple "coroutine". The coroutine has three methods: `Next()`, `Close()`, and `Abort()`. The client calls `Next()` repeatedly until the generator is closed, either by the client or the service. `Next()` can optionally send and/or return a parameter.

Generators are primarily used for two scenarios: transferring large amounts of data, or executing a long running operation (action). RR by default has a limit on the maximum message size, so a generator can be used to break a large amount of data into smaller transfers. RR by default has a timeout for a function call, typically around 10 - 15 seconds. When a generator is used as an action, the client can repeatedly call `Next()` to receive periodic updates, and confirm to the service that the client is still functioning.

[1]C pods may contain passive pointers, but RR usage assumes only data.

E. Wire Peek and Poke

Wire "peek" and "poke" functionality was added in RR version 0.9.0. Often times, a client needs to read or write the value stored in a wire, but does not require real-time streaming updates. The "peek" and "poke" operations allow the client to use a request to read or write the wire value without creating a streaming connection.

F. Intra Transport

The Intra Transport was added in RR Core version 0.9.3. This provides an efficient transport for nodes running in the same process (intra-process communication).

G. Diagnostic Logging and Taps

Beginning in version 0.9.3, RR Core has had extensive diagnostic capabilities implemented. Detailed logging provides information on the internal operation of RR for debugging and auditing purposes. Taps allow for every message passing to and from a node to be intercepted, cloned, filtered, and logged by a companion diagnostic service.

IV. Standardization

A. Framework Standards

Precise, well defined standards are critical for interoperability in communication software. A set of standards are being developed to formalize the following aspects of the RR framework:

- Framework Architecture
- Service Definitions
- Object Protocol and Value Types
- Message Structure and Serialization
- Transports
- Discovery

The draft versions of these standards are available on GitHub.

B. Standard Service Types

As discussed in Section II-A, "service definitions" define objects, members, data types, exceptions, and constants provided by services. For older projects, a new service definition was created for each service. These services were often device drivers, that allowed for a RR client to interact with the device. Because these service definitions were all bespoke, it was not possible for a client to use a different device without modifying the client, even if the device provided similar functionality. As part of the ARM project "Robot Raconteur (RR): An Interoperable Middleware for Robotics" [28], standard service definitions were created that represented classes of devices and allow for interoperability between different devices. The developed service definitions are available on GitHub. Currently the published "Group 1" has 45 service definitions, 41 object types, 162 structures, 71 namedarrays, and 5 pods defined.

Example 1 shows the abridged service definition entry for the "standard robot" type, `com.robotraconteur.robotics.robot.Robot`. This type specifies multiple command modes and metadata. The metadata members (not shown in the example) contains generic information about the device, and robot specific information such as kinematics. The interface allows for robots to be used interoperably, with the same client able to control multiple robots using the same software without modification (See Section VIII-A).

RR has a commitment to long term compatibility. Industrial equipment can operate unchanged for decades, compared to software that can change several times a year. RR is designed to provide long term stability while also designing in enough flexibility to unobtrusively evolve over time. The framework standards and standard service types are important pieces of the strategy to maintain long-term compatibility.

V. Standardized Drivers

Standardized service definitions have been developed to allow for interoperability between devices, as discussed in Section IV-B. Device drivers developed prior to the standardized service types all used incompatible service types, and were not interoperable. As part of the ARM project "Robot Raconteur (RR): An Interoperable Middleware for Robotics" [28] and following projects, standardized drivers have been developed that implement interoperability. These standard drivers were used for the demonstration presented in Section VIII-A.

The following subsections contain lists of the drivers that are available. See the Robot Raconteur directory [15] for direct links to each driver[2].

A. Standard Robot Drivers

Standard robot drivers have been developed for the following robots:

- Rethink Robotics Sawyer Robot
- Rethink Robotics Baxter Robot
- Universal Robots CB2, CB3, and e-Series
- ABB IRC5 6-axis Robots
- Gazebo Simulated Robots

B. Standard Camera Drivers

A standard camera driver has been developed using OpenCV camera capturing. OpenCV supports a wide range of camera types, including standard USB webcams and many embedded cameras such as the Raspberry Pi camera.

Drivers for the Kinect Azure and Intel RealSense have also been developed. These devices implement multiple services to represent the different capabilities provided by the sensors.

A driver for Cognex object recognition cameras has been developed. Unlike the other camera drivers the Cognex is a logical camera. It returns the detected coordinates of the detected objects instead of an image.

C. Standard Input Drivers

Drivers for standard USB joysticks/gamepads have been developed. A Linux only driver for the 3Dconnexion SpaceMouse has been developed.

[2]Citations for specific products have been omitted for space. Please see the individual projects for product citations.

Example 1 Abridged standard Robot service definition entry

```
object Robot
    implements Device
    # ... (abridged)
    property DeviceInfo device_info [readonly,nolock]
    property RobotInfo robot_info [readonly,nolock]
    property RobotCommandMode command_mode [nolockread]
    property RobotOperationalMode operational_mode [readonly, nolock]
    property RobotControllerState controller_state [readonly, nolock]
    # ... (abridged)
    function void jog_joint(double[] joint_velocity, double timeout, bool wait)
    function TrajectoryStatus{generator} execute_trajectory(JointTrajectory trajectory)
    wire RobotState robot_state [readonly,nolock]
    wire AdvancedRobotState advanced_robot_state [readonly,nolock]
    # ... (abridged)
    wire RobotJointCommand position_command [writeonly]
    # ... (abridged)
end
```

VI. Robot Motion Programs

The standard robot drivers discussed in Section V use external control interfaces to the robot. These interfaces allow for the driver to stream setpoint commands to the robot in real-time. Examples of this type of interface include EGM on ABB robots and RTDE on UR robots. While this is a convenient and relatively simple way for the driver to command the robot, the interface tends to have very poor performance in terms of response times and accuracy tracking a signal. When operating using the internal controller, robots use ”motion primitive” commands to move the robot [29]. These commands typically consist of some combination of “MoveJ”, “MoveL”, and ”MoveC” [29]. These commands interpolate in joint space, linear Cartesian space, or circular Cartesian space respectively. The setpoints for the motions are specified in joint or Cartesian coordinates in various combinations depending on the vendor. Other parameters such as motion velocity and blending radius between segments can be specified, with the exact parameters depending on the vendor. “Motion Program” drivers are designed to command robots using their built in motion controller using a sequence of motion primitive commands.

Two ARM funded projects, “Optimized Robot Motion Program for Tracking Complex Geometric Paths” [30] and “Convergent Manufacturing using Multiple Industrial Robots” [31] focus on using motion programs to command the robots. These applications require high precision motion of the robots that is difficult to achieve using the streaming command interfaces and do not require real-time correction using sensors.

Figure 1 shows the turbine blade testbed used to demonstrate the successful development of optimization algorithms for multi-robot trajectory. As part of this project, the `abb_robotraconteur_driver_hmp` was developed. The name is short for “ABB Robot Raconteur Driver Hybrid Motion Program”. This driver implements all the functionality of the standard robot drivers, but adds the additional capability of being able to be commanded using motion programs. The driver is able to switch the robot between

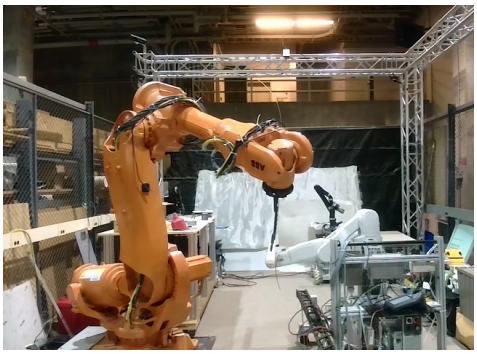

Fig. 1: Motion program turbine blade multi-robot testbed [30]

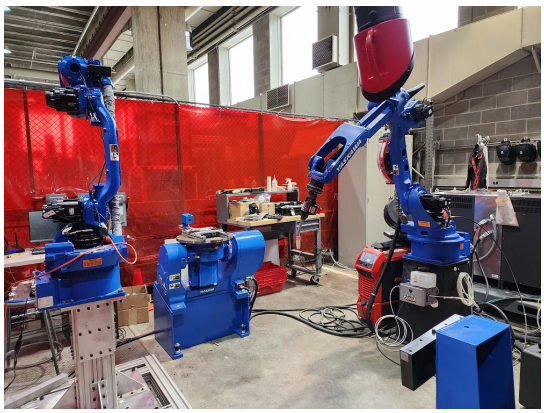

Fig. 2: Robotic WAAM testbed for Convergent Manufacturing [31]

ABB EGM streaming commands and motion programs that execute using ABB RAPID commands. The motion program types are being developed into standardized service definitions that will be frozen in the future.

The ongoing “Convergent Manufacturing using Multiple Industrial Robots” ARM project focuses on developing a framework based on Robot Raconteur for convergent manufacturing. The objective is to use the standardized driver service types and the standards track motion program ser-

vice types to develop an interopable, rapidly reconfigurable architecture. The testbed for this project implements robotic wire arc additive manufacturing (WAAM). Figure 2 shows a photo of the testbed. It consists of several major components. The robot system includes two Motoman robots (MA2010 and MH12), a Motoman positioner (P500), and a DX200 controller. A Fronius TPS 500i is used for the welder. The primary sensor is an Artec Spider laser scanner. An OptiTrack motion tracker system is used for logging and calibration.

During the convergent manufacturing project, drivers will be developed for all the major components. A hybrid motion program driver is being developed for the Motoman system using MotoPlus, an SDK allowing for C based tasks to be executed on the DX200 controller. A driver for the Artec Scanner is also under development and is being open-sourced on GitHub.

VII. Performance

Basic performance measurements were captured for the Round Trip Time (RTT) for sending packets between nodes for Robot Raconteur, ROS 2 Iron, and ROS 1 Noetic. Robot Raconteur has several features that the ROS versions do not that provide usability improvements but can affect performance. These features include an asynchronous message serialization, and and highly parallel multi-threading. The experimental data was captured with these features enabled and disabled to demonstrate the effect on performance. The full results of these tests can be found in the GitHub repository [32].

Tables I, II, and III show results for tests with different configurations. The tests were done on identical Intel based computers. Each computer has an Intel i5-6500 CPU, 16 GB of RAM, and a Realtek RTL8111HSD-CG Gigabit Ethernet LAN network adapter. A Netgear GS108 switch was used to connect the devices. Ubuntu 22.04 was used for Linux, with Windows 11 Professional used for Windows.

Overall the results of the testing do not a give a decisive answer into which technology is "faster", since the results were inconsistent. Because the RTT is so small (often less than 100 µs), the effect of the hardware and the operating system often more important than the effect of the software being measured. RR Core is roughly in the same class as ROS and ROS 2, and for almost all use cases the latency differences will not have any tangible effect. For situations where hard real-time latency is important, the RR Lite library that is currently under development should be used instead of RR Core.

VIII. Examples

A. Interoperable Box Packing using Robot Raconteur

The ARM project "Robot Raconteur (RR): An Interoperable Middleware for Robotics" [28] focused on the development of interoperable drivers for RR. The resulting standard service definitions and drivers are discussed in Section IV-B and Section V. The demonstration for the project involved multiple robots in the same workspace picking and placing

TABLE I: Round Trip Time (RTT) for 64 Byte Payload (microseconds)

Linux Loopback				
Middleware	Mean	Std Dev	Min	Max
RR Default	128.46	269.15	87.97	14247.54
RR No Asyncio	85.92	38.14	74.13	3662.17
RR No Thread	71.15	22.68	63.28	1452.80
ROS 2	66.09	38.10	49.46	733.59
ROS Noetic	74.62	45.39	58.59	4082.73
Linux Ethernet				
Middleware	Mean	Std Dev	Min	Max
RR Default	841.13	87.52	439.67	3247.55
RR No Asyncio	541.62	307.41	153.50	3804.10
RR No Thread	804.67	109.46	416.86	4403.11
ROS 2	294.13	6786.54	103.73	678719.55
ROS Noetic	1042.72	12406.51	81.20	208680.04
Windows to Linux Ethernet				
Middleware	Mean	Std Dev	Min	Max
RR Default	1525.53	223.12	517.50	2083.00
RR No Asyncio	1534.86	182.25	809.30	3325.00
RR No Thread	1429.01	288.50	839.60	6802.60
ROS 2	995.30	113.42	522.50	1527.90

TABLE II: Round Trip Time (RTT) for 1 KB Payload (microseconds)

Linux Loopback				
Middleware	Mean	Std Dev	Min	Max
RR Default	137.53	187.34	95.17	4531.75
RR No Asyncio	195.11	146.71	75.72	1538.84
RR No Thread	70.34	13.04	67.74	321.31
ROS 2	114.60	120.00	55.80	829.78
ROS Noetic	43261.09	5642.37	81.64	48032.40
Linux Ethernet				
Middleware	Mean	Std Dev	Min	Max
RR Default	841.13	87.52	439.67	3247.55
RR No Asyncio	541.62	307.41	153.50	3804.10
RR No Thread	804.67	109.46	416.86	4403.11
ROS 2	294.13	6786.54	103.73	678719.55
ROS Noetic	1042.72	12406.51	81.20	208680.04
Windows to Linux Ethernet				
Middleware	Mean	Std Dev	Min	Max
RR Default	1439.32	194.66	922.10	1908.20
RR No Asyncio	1361.66	218.57	861.70	2595.60
RR No Thread	1427.00	382.87	873.00	6760.30
ROS 2	1053.82	120.68	640.60	1466.20

TABLE III: Round Trip Time (RTT) for 1 MB Payload (microseconds)

Linux Loopback				
Middleware	Mean	Std Dev	Min	Max
RR Default	1134.54	452.19	620.33	3160.29
RR No Asyncio	2151.53	2242.68	905.25	11585.46
RR No Thread	2213.07	1404.46	877.72	9725.43
ROS 2	1461.70	2352.81	919.96	22756.96
ROS Noetic	4340.28	1301.32	3978.75	13953.20
Linux Ethernet				
Middleware	Mean	Std Dev	Min	Max
RR Default	18389.83	360.15	18213.07	20523.54
RR No Asyncio	18999.78	590.55	18507.99	21618.37
RR No Thread	19257.81	4321.46	18471.42	50079.31
ROS 2	21925.65	3862.22	19318.12	30736.35
ROS Noetic	21799.66	919.67	21447.52	30439.87
Windows to Linux Ethernet				
Middleware	Mean	Std Dev	Min	Max
RR Default	19156.84	2844.64	18460.30	47363.10
RR No Asyncio	20988.33	6546.25	19052.10	55396.80
RR No Thread	19631.82	3358.45	19047.50	52940.20
ROS 2	21792.21	3442.51	19530.90	34633.60

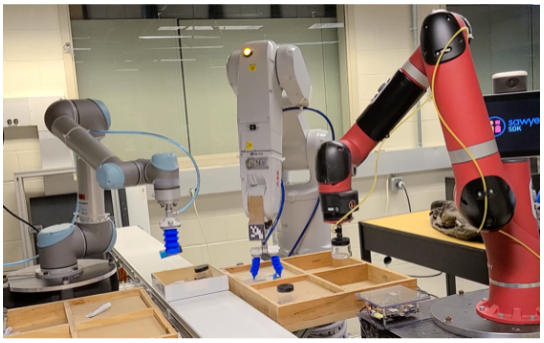

Fig. 3: Robot Raconteur Box Packing Example [17]

Fig. 4: Open-Source Teach Pendant Prototype [17]

objects into trays passing on a conveyor. These robots used RR drivers implementing the standard robot type. The client for each robot was identical, allowing for the robots to be switched into different positions without changing the client software, other than calibration information. A real-time quadratic programming collision avoidance algorithm was used to prevent the robots from impacting each other. For this example system, an ABB IRB1200, an Universal Robtots UR5, and a Rethink Sawyer were used. A Cognex camera was used for sensing the location of the trays. Figure 3 shows a picture of the system used for this demonstration. A video is also available [33].

B. Open Source Teach Pendant

The ARM project "Open Source Teach Pendant Programming Environment" [34] focused on the development of an Open-Source Teach Pendant based on RR and the standard device drivers. The Open-Source Teach Pendant uses a restricted Python dialect called PyRI (Python Restricted Industrial) and Blockly for industrial programming. The objective of the teach pendant is to provide a high-level user interface and programming environment to advanced open-source technologies that are currently only accessible to highly skilled developers. Figure 4 shows a picture of the prototype physical teach pendant. A video is also available [35].

C. Assistive Robotics

RR was used to integrate an assistive robotic system [36]. A Rethink Baxter was mounted on a wheelchair, and configured to complete daily activities. The user commanded the robot using either a sip-puff device or a joystick. Figure 5 shows a picture of the assistive robot setup.

Fig. 5: Assistive robot system integrated using Robot Raconteur [36]

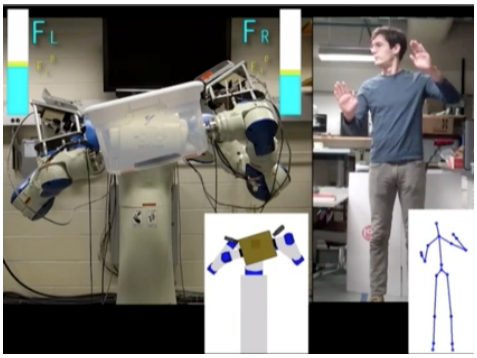

Fig. 6: Human guided robot system integrated using Robot Raconteur [37]

D. Human Guided Dual-Arm Manipulation

RR was used to integrate a system demonstrating telerobotic control of a dual-arm robot using gestures to control the robot [37]. A Microsoft Kinect was used to sense the hand gestures of the operator. An MotoMan SDA 10 robot with ATI force sensors tracked the commands. Figure 6 shows a picture of the human guided robot system.

E. Algorithm Development for Space Manipulation

RR was used to integrate testbeds for the development of algorithms to handle massive objects in space using flexible joint manipulators [38] [39]. These experiments used Rethink Baxter robots, cameras, and custom hardware using Raspberry Pi embedded computers.

F. Robot Raconteur Arduino Uno Demonstration

A minimal service was implemented on an Arduino Uno using an Ethernet Shield for communication. The Arduino Uno has very limited resources at 32 KB of flash storage and 2 KB of RAM. A custom RR implementation was devised that implemented a minimal service representative of a light dimmer switch. The service is capable of turning an LED on, off, and modifying its PWM intensity. The minimal service is fully featured: It supports plug-and-play client connections, standard RR messages, incoming WebSocket connections, and service discovery. This demonstration is important because it shows that even though RR is a powerful and at times complex system, it is still simple and flexible enough to be implemented on very limited hardware.

IX. Conclusion

Robot Raconteur is a powerful, flexible, and user friendly middleware for robotics and automation. This paper has

discussed the development of the framework, the expansion of the RR ecosystem, and presented example applications using RR. Future work on Robot Raconteur will include:

- Version 1.0 release of Robot Raconteur Core with ROS Quality Level 2
- Completion of Robot Raconteur Lite and Robot Raconteur Web
- Continuing development of Robot Raconteur drivers and ecosystem
- Improved documentation and training materials
- Expansion of supported platforms and programming languages
- Publicizing and marketing of framework to gain more users
- Hard real-time support

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] J. D. Wason, "Robot raconteur® version 0.8: An updated communication system for robotics, automation, building control, and the internet of things," in *IEEE International Conference on Automation Science and Engineering (CASE)*, 2016, pp. 595–602.

[3] "System and method for implementing augmented object members for remote procedure call," U.S. Patent 10 536 560B2, Jan. 14, 2022.

[4] D. Jansen and H. Buttner, "Real-time ethernet: the ethercat solution," *Computing and Control Engineering*, vol. 15, no. 1, pp. 16–21, 2004.

[5] P. Brooks, "Ethernet/ip-industrial protocol," in *IEEE 8th International Conference on Emerging Technologies and Factory Automation*, vol. 2, 2001, pp. 505–514.

[6] *Modbus application protocol specification v1.1b3*, Modbus Organization, Inc. Std.

[7] S.-H. Leitner and W. Mahnke, "Opc ua–service-oriented architecture for industrial applications," *ABB Corporate Research Center*, vol. 48, no. 61-66, p. 22, 2006.

[8] 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.

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

[10] T. M. Inc., "Matlab version: 9.13.0 (r2022b)," Natick, Massachusetts, United States, 2022. [Online]. Available: https://www.mathworks.com (accessed Mar. 15, 2023)

[11] R. Bitter, T. Mohiuddin, and M. Nawrocki, *LabVIEW: Advanced programming techniques*. Crc Press, 2006.

[12] 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.

[13] S. Macenski, T. Foote, B. Gerkey, C. Lalancette, and W. Woodall, "Robot operating system 2: Design, architecture, and uses in the wild," *Science Robotics*, vol. 7, no. 66, p. eabm6074, 2022.

[14] Robot raconteur vs ros. [Online]. Available: https://github.com/robotraconteur/robotraconteur/wiki/Robot-Raconteur-vs-ROS (accessed Mar. 15, 2023)

[15] Robot raconteur directory. [Online]. Available: https://github.com/robotraconteur/robotraconteur-directory (accessed Mar. 15, 2023)

[18] *QUIC: A UDP-Based Multiplexed and Secure Transport*, IETF Std. RFC 9000, 2022.

[19] *WebRTC Data Channels*, IETF Std. RFC 8831, 2021.

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

[16] Wikipedia contributors. (2023) Interface description language — Wikipedia, the free encyclopedia. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Interface_description_language&oldid=1138270495 (accessed Mar. 15, 2023)

[17] H. He, B. Aksoy, J. Wason, and J. T. Wen, "Plug-and-play software architecture for coordinating multiple industrial robots and sensors from multiple vendors," in *Submitted to the IEEE International Conference on Automation Science and Engineering (CASE)*, 2023.

[21] *The Transport Layer Security (TLS) Protocol Version 1.2*, IETF Std. RFC 5246, 2008.

[22] Web server implementations in ASP.NET Core. [Online]. Available: https://learn.microsoft.com/en-us/aspnet/core/fundamentals/servers/?view=aspnetcore-7.0&tabs=windows (accessed Mar. 15, 2023)

[23] W. Anggoro and J. Torjo, *Boost. Asio C++ Network Programming*. Packt Publishing Ltd, 2015.

[24] Pyodide: Python with the scientific stack, compiled to WebAssembly. [Online]. Available: https://pyodide.org/en/stable/ (accessed Mar. 15, 2023)

[25] "Package quality categories," REP 2004, ROS, Dec. 2019. [Online]. Available: https://ros.org/reps/rep-2004.html (accessed Mar. 15, 2023)

[26] Wikipedia contributors. (2022) Passive data structure — Wikipedia, the free encyclopedia. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Passive_data_structure&oldid=1117381999 (accessed Mar. 15, 2023)

[27] Python, generators. [Online]. Available: https://wiki.python.org/moin/Generators (accessed Mar. 15, 2023)

[28] Robot raconteur (RR): An interoperable middleware for robotics. [Online]. Available: https://arminstitute.org/projects/robot-raconteur-rr-an-interoperable-middleware-for-robotics/ (accessed Mar. 15, 2023)

[29] H. He, C.-L. Lu, Y. Wen, G. Saunders, P. Yang, J. Schoonover, J. D. Wason, A. Julius, and J. T. Wen, "High-speed high-accuracy spatial curve tracking using motion primitives in industrial robots," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2023.

[30] Optimized robot motion program for tracking complex geometric paths. [Online]. Available: https://arminstitute.org/projects/optimized-robot-motion-program-for-tracking-complex... (accessed Mar. 15, 2023)

[31] Arm institute funds eleven new technology projects. [Online]. Available: https://arminstitute.org/news/new-tech-projects-2023/ (accessed Mar. 15, 2023)

[32] rr_ros_latency_tests. [Online]. Available: https://github.com/johnwason/rr_ros_latency_tests (accessed Mar. 15, 2023)

[33] Plug-n-play robot software coordinating robots and sensor from multiple vendors. [Online]. Available: https://www.youtube.com/watch?v=3jhDXIRUiQY (accessed Mar. 15, 2023)

[34] Open source teach pendant programming environment. [Online]. Available: https://arminstitute.org/projects/open-source-teach-pendant-programming-environment/ (accessed Mar. 15, 2023)

[35] Open source teach pendant jog, save, playback. [Online]. Available: https://www.youtube.com/watch?v=9KSYgGpG8mk (accessed Mar. 15, 2023)

[36] L. Lu and J. Wen, "Human-directed coordinated control of assistive mobile manipulator," *International Journal of Intelligent Robotics and Applications*, vol. 1, no. 1, pp. 104–120, Feb. 2017.

[37] D. Kruse, J. Wen, and R. Radke, "Sensor-based dual-arm tele-robotic system," *IEEE Transaction on Automation Science and Engineering*, vol. 12, no. 1, pp. 4–18, Jan. 2015.

[38] D. Carabis, K. Oakes, and J. T. Wen, "Manipulation of massive objects in space using flexible joint manipulators," *AIAA Journal on Guidance, Dynamics, and Control*, 2021.

[39] D. S. Carabis and J. T. Wen, "Trajectory generation for flexible-joint space manipulators," *Journal on Frontiers in Robotics and AI, section Space Robotics*, vol. 9, Mar. 2022.