Remote Interoperability Protocol
for
online laboratories

Prepared by Luis de la Torre, Jesus Chacon and Dictino Chaos

Universidad Nacional de Educacion a Distancia (UNED)
ETS. Ingenieria Informatica
Madrid
Spain

18th November 2018

Contents

Revision History

Revision	Date	Author(s)	Description
0.1	18.11.2018	L. de la Torre	Chapter 1 - Introduction and document structure
0.11	24.11.2018	L. de la Torre	Chapter 2 - Overall Description (Sections 2.1-2.5)

Chapter 1

Introduction

1.1 Purpose

This document is a specification of the Remote Interoperability Protocol (RIP), which was conceived at UNED for the remote operation of online laboratories (OLs). Instructions on how to correctly implement both a server and a client that talk RIP are also given.

1.2 Document Conventions

For the purpose of this document, we consider that an OL can either be a virtual laboratory (VL) or a remote laboratory (RL).

VLs are simulations and offer experimentation possibilities based on mathematical models.

RLs use lab equipment and perform the experiments in real life, just remotely.

Simulation models are understood as software that include mathematical models that simulate a system for virtual experimentation purposes.

Control program is a term used in this document to refer to the software in charge of controlling and monitoring lab equipment.

In this sense, we consider that a VL always has an associated *simulation model* that is hosted and run in some computer, while a RL always has an associated *control program* that is also hosted and run in some computer.

Finally, we define an *experience* as each of the lab activities that can be carried out with an OL implementation, either through a RL or through a VL.

1.3 Intended Audience and Reading Suggestions

Audiences that may be interested in this document are educators, researchers and industry stakeholders that want or need to remotely communicate either with hardware devices or mathematical models from a web application.

More specifically, this document aims at anyone who is interested in one or more of the following points:

1. Implementing a RIP server and/or a RIP client to use RIP as the communication protocol for operating OLs.
2. Using or modifying an existing RIP server and/or RIP client implementation.

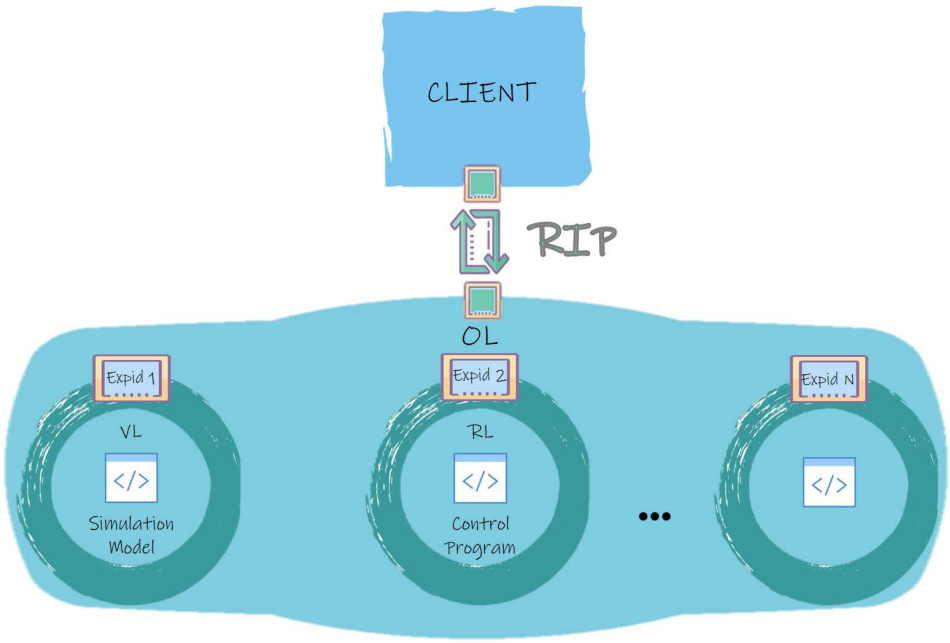

Figure 1.1: RIP Scope

3. Making modifications on the RIP protocol itself.

In any of the above cases, it is advised to read the present document in order. Before reading this document, it is reccommended to have some notions about TCP [ISI81], HTTP [FGM+99], SSE [sse09] and JSON-RPC [Gro10].

1.4 Product Scope

The objective of RIP is to offer a simple, yet powerful, communication solution usable from web clients. As such, RIP only uses pure HTTP standard protocols, supported by all major web browsers.

RIP is designed to communicate web clients with OLs; either VLs or RLs. When used to communicate with a VL, RIP exposes meta-data and input and output methods and variables related to a simulation model that is hosted and runs on a computer (usually, a remote server). When used to communicate with a RL, RIP does the same thing with a *control program* defined in a computer (usually, a remote server) to monitor and manipulate the lab equipment.

Figure 1.1 shows the usage of the RIP protocol implemented in a RIP client and a RIP server to communicate a web client with an OL. The figure represents the OL can implement either a VL, a RL, or any combination of both, each one defined as an independent *experience*, referenced through a certain *expid* parameter.

Chapter 2

Overall Description

2.1 Protocol Perspective

The protocol is an open source, under the GNU general Public License. It is a communication protocol to be used in the client-server model, especially designed for OLs in which the client runs within a web browser. RIP provides a simple mechanism for users and client machines/programs to acquire information about the lab *experiences* defined in the server and about each *experience's* inputs and outputs. The protocol also defines mechanisms for reading and writing the values of these inputs and outputs, respectively.

The main features of RIP are the following:

1. Defining *experiences* on the OL.
2. Obtaining meta-data related to each defined *experience*.
3. Obtaining a list of readable and writable variables for each *experience*.
4. Obtaining a list of methods to read and write variables in each *experience*.
5. Defining server events to send data either periodically or based on any other triggering condition defined in an *experience*.
6. Subscribing a client to any server event declared in an *experience*.

2.1.1 Server Implementation Perspective

Figure 2.1 depicts the architecture of a *RIP Server* that implements the RIP protocol. To sum it up, there are three functional subsystems: the *Web Server*, that handles client connections, sessions, etc., the *command interpreter*, that speaks the RIP protocol, and the *executor*, that controls the execution of the laboratory *control programs*.

The Web Server admits (and handles in different ways) three types of requests: GET (used to retrieve *experiences'* meta-data), SSE (used to get server-to-client data updates) and POST (used to send client-to-server updates). These different methods are each associated with the three basic cases of use, namely:

- *Meta-data* - A client, wanting to obtain information about the laboratory, launches an HTTP GET request to the URL associated with the laboratory. The laboratory responds with a JSON-RPC structure that informs the client, depending on the request's parameters, with one of the following:
    1. General information about the OL: what are the *experiences* defined and how they can be accessed.

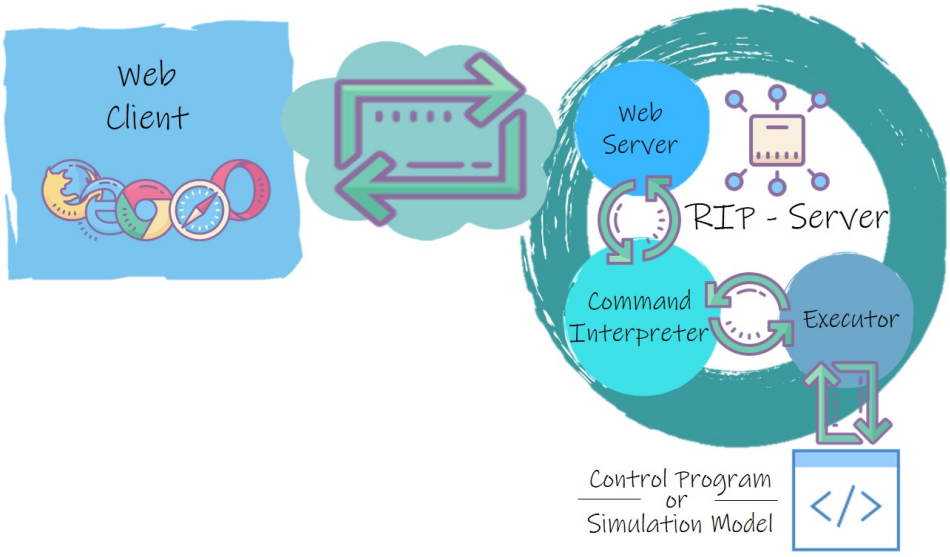

Figure 2.1: Architectural view of a RIP Server

2. Detailed information of a particular *experience* (when the request includes the experience id as a parameter).

- *Observer* - A client, that desires to receive updates on the state of the plant, subscribes to an SSE event stream associated to the *experience* of interest.
- *Controller* - A client, wanting to act over the OL, sends a POST request with the command codified as a RIP-JSON-RPC structure.

An *experience* represents a lab activity associated with an OL. In the case of RLs, each *experience* is implemented as a *control program*, which in general is responsible of managing the physical connections with the hardware, safe measures, and any other functionality the lab designer has considered appropriate to include. In the case of VLs, each *experience* is implemented as a *simulation model* that represents a real system. The *experience* abstraction is useful for two purposes: 1) to publish information about OLs in a standard and structured way and 2) to allow for hosting and running several different *control programs* or *simulation models* in the same computer.

2.2 Protocol Functions

The functions defined and used in the protocol are divided in two types: internal and external.

The internal functions are private functions, not exposed to the web clients. These methods are used internally by RIP server implementations to communicate either with the *simulation model* or the *control program* used in the OL.

The external functions are exposed through web services as public functions, and so, they are callable by the web clients. These methods are used by the clients to both get meta-data about the defined *experiences* and to send and retrieve data to and from the OL.

2.2.1 Internal Functions

The list of existing internal functions a RIP server must implement is:

1. *readablelist*, *writablelist* = **open**(*expid*): Returns the list of readable and writable variables (in two different variables) defined in the *experience* associated to the input *expid* parameter.

2. **close**(*expid*): Closes the control program (if it is a RL) or the simulation model (if it is a VL) of the OL *experience* defined by the input *expid* parameter.

2.2.2 External Functions

The list of existing external functions a RIP server must implement so that a RIP client may use is:

1. **info**(*expid*): Retrieves the meta-data information about the *experience* defined by the input *expid* parameter.
2. **start**(*expid*): Starts the execution of the control program (if it is a RL) or the simulation model (if it is a VL) of the OL *experience* associated to the input *expid* parameter.
3. **stop**(*expid*): Stops the execution of the control program (if it is a RL) or the simulation model (if it is a VL) of the OL *experience* defined by the input *expid* parameter.
4. *readvariablenames*, *readvariablevalues* = **get**(*expid*, *variablenames*): Retrieves the current values of the variables (*readvariablevalues*) specified by the *variablenames* input parameter. For this to happen, these variables must exist in the *control program* or *simulation model* associated to the *experience* defined by the *expid* input parameter. When the **get()** method is called, *readvariablenames* contains only the names of the variables that were successfully read, not all requested ones in *variablenamess*.
5. **set**(*expid*, *variablenames*, *variablevalues*): Writes the received values (*variablevalues*) in the specified variables (*variablenames*) of the appropiate OL *experience*. For this to happen, these variables must exist in the *control program* or *simulation model* associated to the (*expid*) input parameter.

Where:

Variables *variablenames* and *variablevalues* are arrays of text. For example: *variablenamess* = ["x", "y", ...], *variablevalues* = ["10", "a", ...].

Each method returns an error indicator when the operation is not completed successfully.

At the moment, only numbers, text and booleans are supported in the **set**() and **get**() methods.

2.3 Operating Environment

RIP uses only HTTP methods for the communication. Therefore, it works in any major web browser. However, RIP also relies on the use of SSE, which, up to date, are not supported by Microsoft Internet Explorer nor Microsoft Edge. Nevertheless, there are numerous poly-fill solutions for implementing SSE so that they work on these browsers that do not support them natively.

2.4 Design and Implementation Constraints

RIP is designed to only use pure HTTP methods on purpose, with the aim of guaranteeing its correct functioning from web browsers.

2.5 Assumptions and Dependencies

RIP depends solely on the use of HTTP POST and GET methods and on the JSON format for exchanging data.

2.6 User Documentation

TODO

Chapter 3

External Interface Requirements

3.1 User Interfaces

TODO

3.2 Software Interfaces

TODO

3.3 Communications Interfaces

TODO

Chapter 4

Protocol Features

4.1 Defining Experiences

TODO

4.2 Obtaining Meta-data

TODO

4.3 Obtaining Readable and Writable Variables

TODO

4.4 Obtaining and Calling Interface Methods

TODO

4.5 Defining Server Events

TODO

4.6 Subscribing to Server Events

TODO

Appendices

Appendix A

Glossary

Control program - A software program which purpose is to control and monitor the hardware equipment used to in a laboratory activity.

Experience - Any lab activity that can be performed in an OL.

OL - Online laboratory.

RIP - Remote Interoperability Protocol.

RL - Remote laboratory.

Simulation model - A mathematical simulation that models a certain system with which laboratory activities can be carried out.

SSE - Server-sent events.

VL - Virtual laboratory.

Appendix B

Analysis Models

TODO

References

[FGM+99] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Http 1.1 specification, 1999.

[Gro10] JSON-RPC Working Group. Json-rpc 2.0 specification, 2010.

[ISI81] University of Southern California Information Sciences Institute. Transmission control protocol specification, 1981.

[sse09] Server-sent events specification, 2009.