EAI Endorsed Transactions
on Cloud Systems
Research Article EAI.EU

Specification of REST API Services for Modbus Protocol using Formal technique

Jayasankar. M, Anandi Giridharan

Jayasankar. M is with Quest-Global, e-mail: (jayasankar.m@gmail.com)
Anandi Giridharan, Principal Research Scientist was with ECE Department, Indian Institute of Science, Bangalore. (e-mail: anandi@iisc.ac.in).

Abstract

With the advancements in technologies, there has been a growing trend to move from desktop applications towards web and mobile applications. This move was made possible through introduction of the RESTful Web Services. The benefits of such a move include minimal installation costs, automated upgrading for all users, and increased interactivity and universal access.
Modbus is a serial communications protocol originally published by Modicon (now Schneider Electric) in 1979 for use with its programmable logic controllers (PLCs). Simple and robust, it has since become a de facto standard communication protocol, and it is now a commonly available means of connecting industrial electronic devices.
This paper, demonstrates the advantages of providing RESTful interface to modbus protocol. We have used formal SDL (Specification and Description Language) to specify this framework and study their performance evaluation.

Keywords: HTTP, Modbus, REST, SDL, TCP, MSC

Received on 10 October 2017, accepted on 28 November 2017, published on 20 December 2017

Copyright © 2017 Jayasankar. M and Anandi Giridharan, licensed to EAI. This is an open access article distributed under the terms of the Creative Commons Attribution licence (http://creativecommons.org/licenses/by/3.0/), which permits unlimited use, distribution and reproduction in any medium so long as the original work is properly cited.*

doi: 10.4108/eai.20-12-2017.153492

1. Introduction

Representational State Transfer (REST) has gained widespread acceptance across the Web as a simpler alternative to SOAP and Web Services Description Language (WSDL) based Web services. Key evidence of this shift in interface design is the adoption of REST by mainstream Web 2.0 service providers including Yahoo, Google, and Facebook, who have passed on SOAP and WSDL-based interfaces in favor of an easier-to-use, resource-oriented model to expose their services.

Statelessness is the key of REST. Necessary state to handle the request is contained within the request itself, whether as part of the URI, query-string parameters, body or headers. The URI uniquely identifies the resource and body contains the state of that resource. Then after the server does it's processing, the appropriate state, or the piece of state that matter are communicated back to the client via headers, status and response body.
REST is any interface between systems using HTTP to obtain data and generate operations on those data in all possible formats, such as XML and JSON. This is an increasingly popular alternative to other standard data exchange protocols such as SOAP (Simple Object Access Protocol), which have a high capacity but are also very complex. Sometimes it's preferable to use a simpler data-processing solution such as REST.

2. RESTFUL SERVICES

REST defines a set of architectural principles by which one can design Web services that focus on a system's resources, including how resource states are addressed and transferred over HTTP by a wide range of clients written in different languages. If measured by the number of Web services that use it, REST has emerged in the last few years alone as a predominant Web service design model. In fact, REST has had such a large impact on the Web that it has mostly displaced SOAP and WSDL based interface design because it's a considerably simpler style to use. A concrete implementation of REST Web service follows four basic design principles:

1. Use HTTP methods explicitly.
2. Be stateless.
3. Expose directory structure-like URIs.

4. Transfer XML, JavaScript Object Notation (JSON), or both.

In this work, we propose a framework for the formal specification of REST API Services for Modbus Protocol using SDL based system. The purpose of this framework is to provide a formal basis for their performance evaluation and behavioral study. The rest of the paper is organized as follows. Section 4. illustrates specification design of RESTAPI Services for Modbus Protocol using formal SDL language. Section 5 describes of verification and validation of the protocol to rule out design errors. Section 5 illustrates experimental simulations with results and Section 6. provides our conclusion.

3. Modbus protocol

Modbus protocol is an application layer messaging protocol at Level 7 of the OSI Model that provides client/server communication between devices connected on different types of buses or networks. The Modbus messaging structure was developed by Modicon in 1979. Different versions of Modbus used today include Modbus RTU (based on serial communication like RS485 and RS232), Modbus ASCII and Modbus TCP, which is the Modbus RTU protocol embedded into TCP packets.

4. Specification of the framework using formal language SDL

Specification using formal SDL (Specification and Description Language) to specify this framework and study their performance evaluation was proposed.

Figure 1 and 2 shows the package RMB and the system RESTMB. The package RMB contains declarations of all the signals used in the system RESTMB

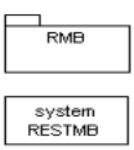

Figure 1: Specification Package RMB

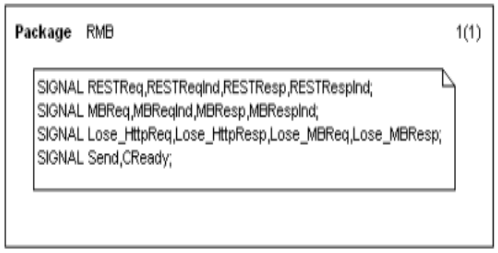

Figure 2: Declaration of all signals used

4.1 Possible Communication losses:

Figure 3 shows the possible communication losses that can happen in the system. There are 2 types of losses that can happen viz. http request/response losses and Modbus request/response losses.

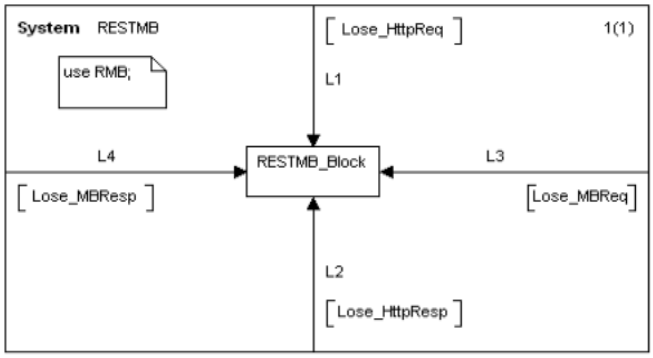

Figure 3: RESTMB system specification

Figure 4. depicts the RESTMB block which shows how the system works.

1.User sends a http request to a REST server (RESTServerAndModbusMaster in the figure) using a RestClient like postman and waits for response.

2.The REST server(which also acts as Modbus master) on receiving a request, processes it and sends a modbus request to modbus slave (slave simulator in this case) and waits for response.

3.The modbus slave sends back modbus response to the Modbus master.

4.The modbus response is processed and send as http response back to the RestClient.

5.The losses that can happen in the http and modbus channels are also depicted in the block.

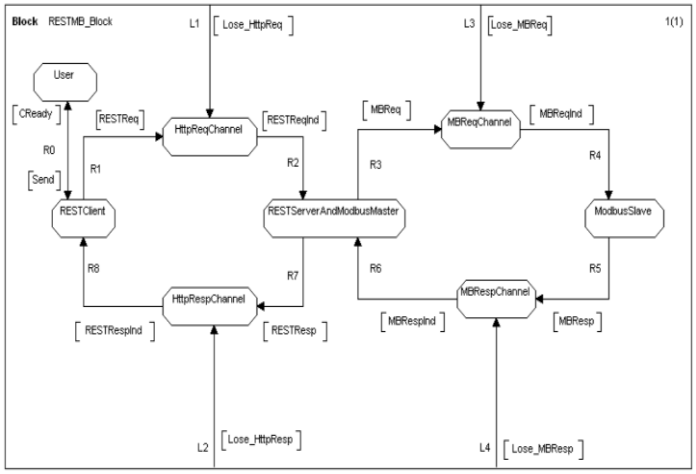

Figure 4: RESTMB block specification

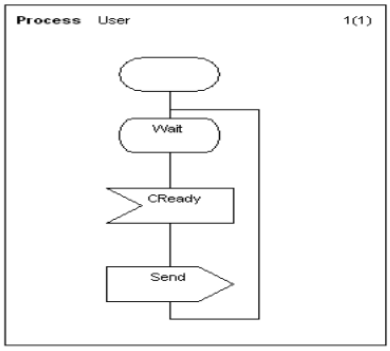

Figure 5: User Process specification

Figure 4 and 5 shows the User process and RestClient process respectively.

1.User initiates a rest request through the RestClient once the RestClient is ready.

2. The RestClient after sending the request starts a timer and waits for a specified time for the response.

3. if response is not received with in specified time, timeout happens and an error message is as indicated in the process diagram.

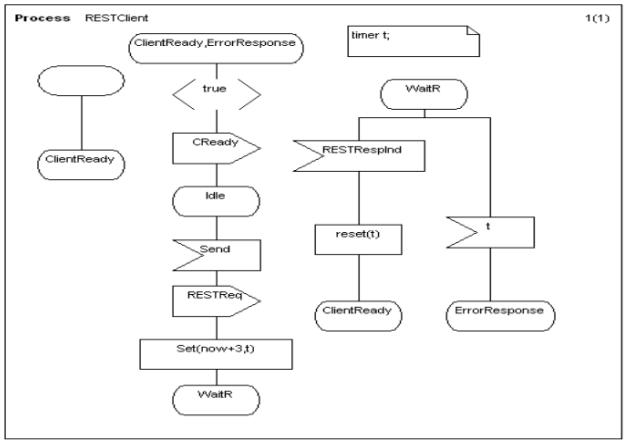

Figure 6 : RestClient process specification

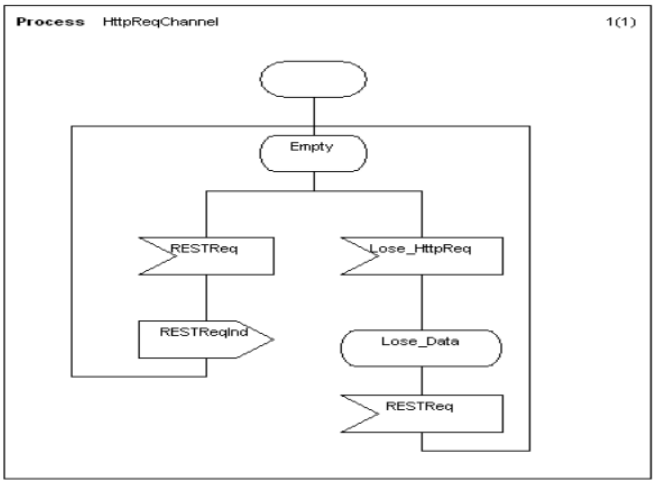

Figure 7: RestClient Process

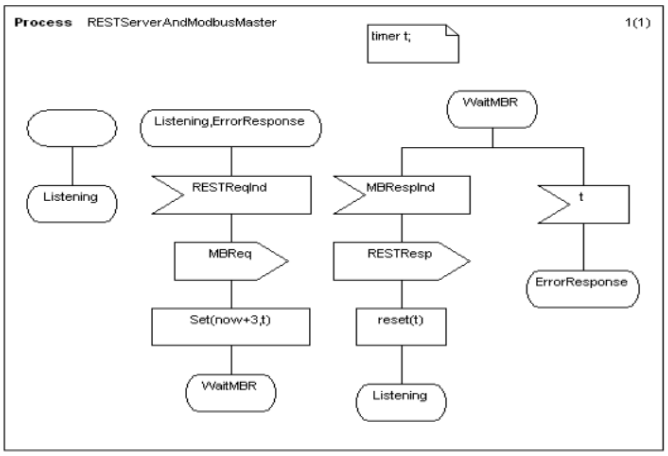

Figure 8: RESTServerAndModbusMaster Process

Figure 7 and 12 shows http request and response channels respectively which acts as channel for sending and receiving REST request and response.

Figure 8 shows the RESTServerAndModbusMaster. It processes and converts http (ReST) request from RestClient to modbus request and modbus response from ModbusSlave to http(ReST) response.

Figure 9 and 11 shows modbus request and response channels respectively which acts as channel for sending and receiving modbus request and response.

Figure 10 shows Modbus Slave process which waits and listens for modbus request from the Modbus master (RESTServerAndModbusMaster). On receiving a request, it sends back a modbus response.

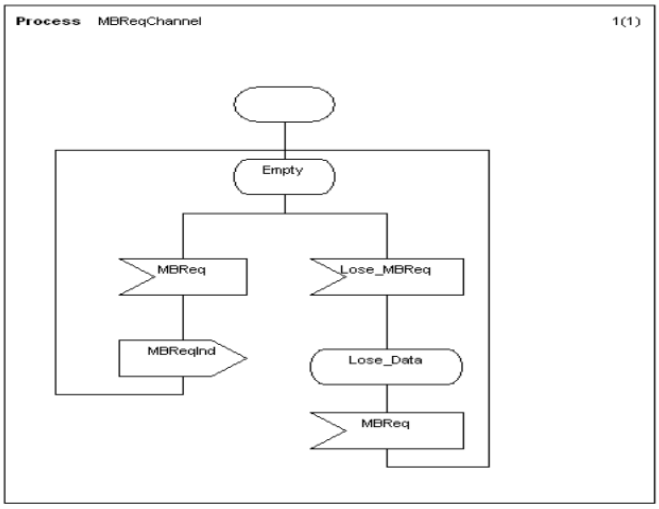

Figure 9:MBReqChannel Process

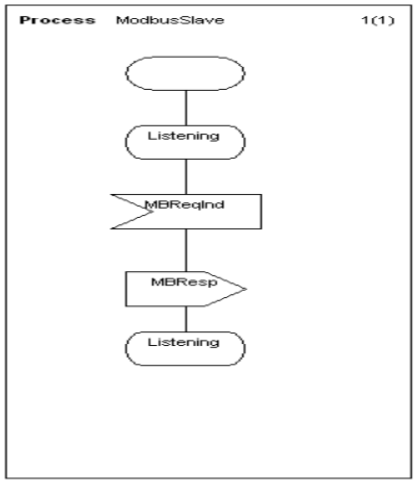

Figure 10:ModbusSlave Process

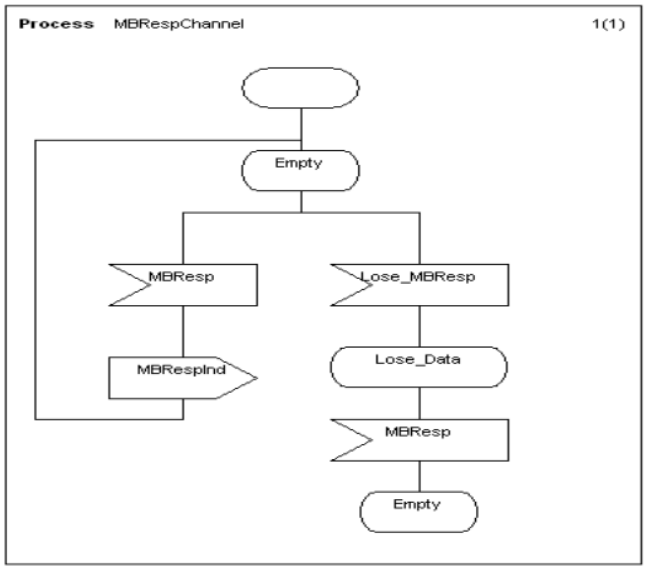

Figure 11:MBrespChannel Process

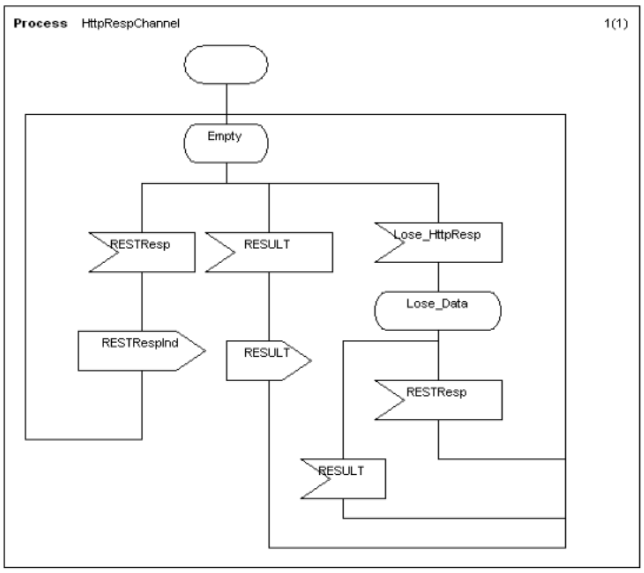

Figure 12:httpRespChannel Process

5.Verification and Validation

The design of the framework must ensure its ability to operate under increasing load, increasing complexity of requests and increasing size of resulting composite services. Hence verification and validation is done to check for correctness of protocol specification and check the protocol for liveness property and safety properties. Protocol validation can be used to increase the quality of protocol design by verifying the system against the requirement.

Validation ensures that the protocol specifications will not get into protocol design errors. (Deadlock, unspecified reception etc).
The message sequence chart in figure 13, shows interaction between entities and complete service discovery process.

Safety Property:
1. Ensures non-violation of assertions.
2. System ensures that proper data is send from "RestClient" and "RESTServerAndModbusMaster", although it may get lost in the channel

Liveness Property:
1. Ensures proper termination of protocol.
2. Even if the data send from RestClient" / "RESTServerAndModbusMaster" gets lost in the channel,it is ensured that the protocol terminates correctly.

Validation
1. From the message chart diagram, it is seen that the proper response is obtained for valid requests.

2. It is also ensured that incase of lossy channel/data loss,the system does not go to a deadlock.
3. The timers in the “RestClient” and “RESTServerAndModbusMaster” ensures that if the response is not obtained within a timelimit,an error response is generated and the system again is ready to accept new requests.

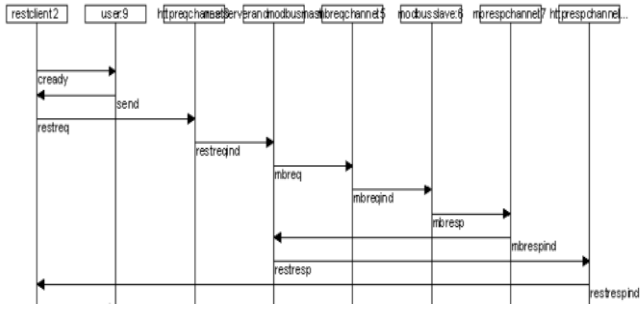

Figure 13 : MSC chart generated using SDL

The Message Sequence chart generated for the system for normal flow is shown in figure 13.

1. The RestClient sends a http request “restreq” through http request channel once the user initiates it.
2. The http request is converted to modbus request “mbreq” by the server and send to the modbus slave through modbus channel.
3. On receiving a modbus request, the modbus slave sends back the modbus response “mbresp” to the server.
4. The server processes the modbus response and generates the http response “restresp” which is send back to the rest client through http channel

5.Implementation

Following tools were used for implementation of the system

5.1 Slave Simulator

A slave simulator is used for simulating a device which has modbus capability.Basically it listens for modbus requests and responds based on the modbus command received. “Diagslave”, a freely available slave simulator for modbus was used for the project, is shown in figure 14.

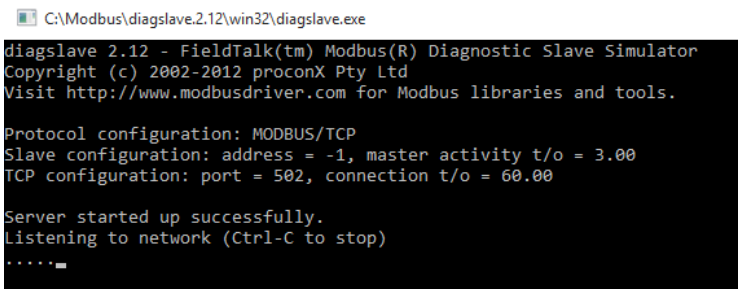

Figure14: Diagslave server

5.2 RestServer

This is required for performing REST operations, and to obtain the appropriate “REST” response when a query was given by user.
“EVE”, a freely available python library was used to perform necessary REST related operations and to generate appropriate REST response.Figure15 shows documentation of Eve framework.

Figure 15 – Eve python ReST API framework

5.3 Rest Client

A Rest Client is required for the user to send “REST” query to rest server. “POSTMAN” extension of chrome was used in the project to act as RestClient.Figure16 shows UI interface of POSTMAN tool.

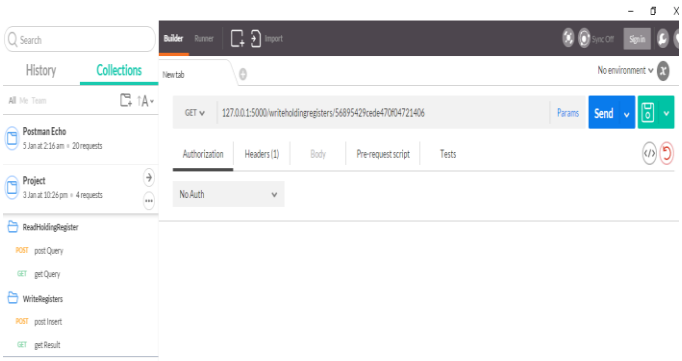

Figure 16: Postman extension for chrome

5.4 Modbus Master

A modbus master is required to send modbus command to the slave simulator and obtain the modbus response. “Pymodbus” a freely available python library for modbus was used in the project.Figure 17 shows Pymodbus documentation.

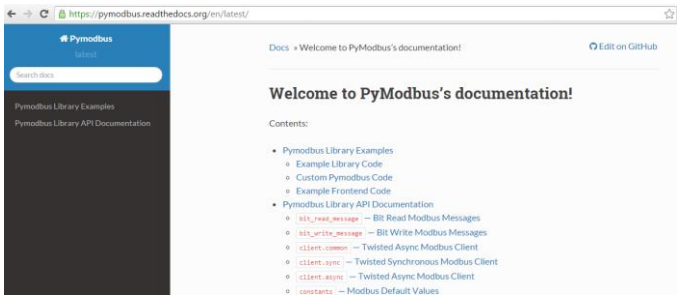

Figure 17:Pymodbus module

1.Using “Eve” Rest framework,the rest services for the Modbus function codes “Read holding registers” and “Write holding registers” were implemented.This can be extended to other function codes as well.

2.User can send modbus requests(function codes) by embedding the required data in the request made to the server.On receiving the request,the eve server processes the request and using the “pymodbus” makes a modbus request to the slave simulator and receives back the modbus response.

3.The Modbus response thus obtained can be send back to the user by embedding it in the response data as a response to a rest request.

6.Conclusion

Formal SDL (Specification and Description Language) to specify framework and study their performance evaluation was used. Any web/mobile application which requires interaction with devices can directly call the REST API for modbus and communicate with device.

References

1. http://python-eve.org
2. https://pymodbus.readthedocs.io/en/latest/
3. http://www.modbusdriver.com/diagslave.html
4. https://chrome.google.com/webstore/detail/postman/fhbjgbiflinjbdggehcddcbncdddomop
5. https://en.wikipedia.org/wiki/Representational_state_transfer
6. http://www.ibm.com/developerworks/library/ws-restful/
7. https://en.wikipedia.org/wiki/Modbus