Contents

Introduction:

The XML-RPC protocol is a widely used protocol for client-server communication over the internet. This protocol allows a client to invoke methods on a server, and receive the result of that method invocation in XML format. In this project, we have implemented an XML-RPC application that uses socket communication to transfer XML messages between the client and the server. The server has several methods that can be invoked by the client, including methods for computing the square of a given number, computing the sum of a given list of integers, reversing a given string, sorting a list of integers, solving mathematical equations, and finding the shortest path in a graph.

Methodology:

We have used Python programming language to implement the XML-RPC application. The socket module is used for socket communication between the client and the server. The xml module is used for decoding XML messages, and the OS module is used for performing OS tasks like invoking methods. The server has a Python script (command: python server.py) that receives an XML message from the client, decodes this message to perform method invocation, and then returns the result (as an XML message) to the client. The client has a Python script (command: python client.py) that sends an XML message to the server, then receives the response from the server (also as an XML message), decodes this message, and prints the result on the screen.

Note: Always start server first. Then start client (command: python client.py). Client will send you different option work according to it.

Logic Flow:

1. The client sends an XML message (in the form of a method invocation) to the server.
2. The server receives the XML message from the client, and decodes it to determine which method to invoke.
3. The server invokes the requested method, passing any necessary arguments.
4. The server computes the result of the method invocation.
5. The server encodes the result as an XML message, and sends it back to the client.
6. The client receives the XML message from the server.
7. The client decodes the XML message to extract the result of the method invocation.
8. The client prints the result on the screen.

Program Flow:

1. The server script (server.py) starts by creating a socket and binding it to a specific port.
2. The server script listens for incoming connections from clients on this port.
3. When a client connects to the server, the server script creates a new thread to handle the client's request.
4. The server script reads the XML message from the client, and decodes it to determine which method to invoke.
5. The server script invokes the requested method, passing any necessary arguments.
6. The server script computes the result of the method invocation.
7. The server script encodes the result as an XML message, and sends it back to the client.
8. The client script (client.py) starts by creating a socket and connecting to the server on a specific port.
9. The client script sends XML message to the server.
10. The client script waits for a response from the server.
11. The client script receives the response from the server, and decodes it to extract the result of the method invocation.
12. The client script prints the result on the screen.

Backend Working of XML-RPC:

The backend of the XML-RPC application consists of the server-side code that receives incoming socket connections from clients, parses the XML requests, invokes the appropriate method based on the request, and sends the result back to the client in an XML response.

The backend working of the application can be explained in the following steps:

1. Setting up the server: The server is set up to listen for incoming socket connections from clients. The server can be set up on any machine with a network connection and a Python environment.

2. Handling incoming socket connections: When a client establishes a socket connection with the server, the server accepts the connection and starts receiving the XML request message from the client.

3. Parsing the XML request: The server parses the XML request message to extract the method name and the input parameters. The xml module in Python can be used for parsing the XML messages.

4. Invoking the method: Based on the method name in the XML request message, the server invokes the appropriate method with the input parameters. The method can be any python file at server side that takes input parameters and returns a result.

5. Creating the XML response: The server creates an XML response message with the result of the method invocation.

6. Sending the XML response: The server sends the XML response message back to the client over the socket connection.

Program Execution:

- Navigate to directory of server in command line.
- Execute command (python server.py).

```
\Client_server(xml)\server> python server.py
```

- Server is started and listening for connection from client
- Now open another terminal and navigate to directory of client in command line.
- Now execute command (python client.py). Note that server is already running on another terminal.

```
Client_server(xml)\Client> python .\client.py
```

- After you execute command. Client will show you following menu on terminal:

```
Select any of following method :
1. Select xml File.
2. Generate your own xml
3. Exit

CHOICE(In Number) :
```

- If you click option ‘1’ it will show you pre-generated XML files.

```
CHOICE(In Number) :1

Select Filename From availaible Filenames:
----------------------------------------

data.xml
equation1.xml
path1.xml
result.xml
reverse1.xml
reverse2.xml
sort1.xml
square1.xml
square2.xml
sum1.xml
sum2.xml

----------------------------------------
Filename:
```

- Now select pre-generated xml file. For example : sum1.xml

```
----------------------------------------
Filename: sum1.xml

[+] received a data from -> ('127.0.0.1', 61910)

ANSWER :161

----------------------------------------
```

Now see server side.

```
 ---------------------------------


[+] received a data from -> ('127.0.0.1', 61909)
method requested = sum
server executing sum.py
Result = 161
Encoding Result in XML
sending result back to client

 ---------------------------------
```

- Note in this way you can select any XML file which are pre-generated.
- XML file handles all methods such as square, sum, equation calculator, shortest path, reverse string, sorting. In the screenshot I have shown only sum method but you all options.
- If you want to generate your own XML file. Select option ‘2’.

```
Select any of following method :
1. Select xml File.
2. Generate your own xml
3. Exit

CHOICE(In Number) :2
```

- You will have this menu generated.

```
CHOICE(In Number) :2
select your method from availiable methods
-----------------------------------------

sum
sort
path
equation
reverse
Square

-----------------------------------------

Method :
```

- Now select any method for which you wanted to generate xml.

For method sum:

- Write method: “sum”. Then enter required numbers according to your requirement.

```
Method :sum
Total no.of elment to sum :
```

```
Method :sum
Total no.of elment to sum :3
enter number 1 :2
enter number 2 :1
enter number 3 :3

[+] received a data from -> ('127.0.0.1', 61937)

ANSWER :6
```

Server side below:

```
[+] received a data from -> ('127.0.0.1', 61936)
method requested = sum
server executing sum.py
Result = 6
Encoding Result in XML
sending result back to client
```

For Method sort:

- Write method: “sort”. Then enter required numbers according to your requirement. See below screenshots for help:

```
Method :sort
Total no.of elment to sort :3
enter number 1 :3
enter number 2 :1
enter number 3 :2

[+] received a data from -> ('127.0.0.1', 61944)

ANSWER :1,2,3
```

Server side below:

```
[+] received a data from -> ('127.0.0.1', 61943)
method requested = sort
server executing sort.py
Result = 1,2,3
Encoding Result in XML
sending result back to client
```

For method equation:

- Write method: “equation”. Then enter required equation according to your requirement. See below screenshots for help:

```
Method :equation
Enter Equation :5+3-(3*2)

[+] received a data from -> ('127.0.0.1', 61949)

ANSWER :2.0
```

For method shortest path:

- Write method: “path”. See below screenshots for help:
- Consider simple connected graph. A->B->C->A (A is connected to B and C. B to C and A. C to A and B).
- Now implement this graph in following way as screenshot :

```
Method :path
Enter no.of nodes :3
Enter node (such as 'A'): A
Enter connected nodes of node A (Format : B,E,D):B,C
Enter node (such as 'A'): B
Enter connected nodes of node B (Format : B,E,D):C,A
Enter node (such as 'A'): C
Enter connected nodes of node C (Format : B,E,D):A,B
----------------------------------------

Enter start node :A
 enter destination node :C

[+] received a data from -> ('127.0.0.1', 61963)

ANSWER :A C
```

For method Reverse:

```
Method :reverse
enter String to Reverse :tea

[+] received a data from -> ('127.0.0.1', 62092)

ANSWER :aet
```

Conclusion:

In this project, we have implemented an XML-RPC application that uses socket communication to transfer XML messages between the client and the server. We have shown how to implement several different methods on the server, including methods for computing the square of a given number, computing the sum of a given list of integers, reversing a given string, sorting a list of integers, solving mathematical equations, and finding the shortest path in a graph. We have demonstrated how the client can invoke these methods by sending XML messages to the server, and how the server responds back to the client.