1. **NETCONF Client**: Establish a connection and fetch configuration data.
2. **RESTCONF Client**: Interact with RESTCONF API to fetch and apply configuration data.
3. **Unit Testing**: Basic tests to ensure the functionality of the clients.

**Project Structure**

css

Copy code

network-config-manager

├── pom.xml

└── src

└── main

└── java

├── NetconfClientManager.java

├── RestconfClientManager.java

└── test

└── java

├── NetconfClientManagerTest.java

├── RestconfClientManagerTest.java

**pom.xml**

Include dependencies for Netconf4J and Apache HttpClient:

xml

Copy code

<project xmlns="http://maven.apache.org/POM/4.0.0"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.example</groupId>

<artifactId>network-config-manager</artifactId>

<version>1.0-SNAPSHOT</version>

<dependencies>

<dependency>

<groupId>org.opendaylight.netconf</groupId>

<artifactId>netconf4j</artifactId>

<version>1.2.0</version>

</dependency>

<dependency>

<groupId>org.apache.httpcomponents</groupId>

<artifactId>httpclient</artifactId>

<version>4.5.13</version>

</dependency>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.13.2</version>

<scope>test</scope>

</dependency>

</dependencies>

</project>

**NetconfClientManager.java**

java

Copy code

import org.opendaylight.netconf.client.NetconfClient;

import org.opendaylight.netconf.client.NetconfClientSession;

import org.opendaylight.netconf.client.NetconfClientConfiguration;

import org.opendaylight.netconf.client.mina.NetconfClientMinaImpl;

public class NetconfClientManager {

private NetconfClientSession session;

public void connect(NetconfClientConfiguration clientConfig) throws Exception {

NetconfClient client = new NetconfClientMinaImpl();

this.session = client.connect(clientConfig).get();

}

public String getConfig() throws Exception {

String getConfigRequest = "<rpc message-id=\"1\"><get-config><source><running/></source></get-config></rpc>";

return session.sendRpc(getConfigRequest);

}

public String editConfig(String config) throws Exception {

String editConfigRequest = "<rpc message-id=\"2\"><edit-config><target><running/></target><config>" + config + "</config></edit-config></rpc>";

return session.sendRpc(editConfigRequest);

}

public void close() throws Exception {

if (session != null) {

session.close();

}

}

}

**RestconfClientManager.java**

java

Copy code

import org.apache.http.client.methods.HttpGet;

import org.apache.http.client.methods.HttpPost;

import org.apache.http.entity.StringEntity;

import org.apache.http.impl.client.CloseableHttpClient;

import org.apache.http.impl.client.HttpClients;

import org.apache.http.util.EntityUtils;

public class RestconfClientManager {

private CloseableHttpClient httpClient;

public RestconfClientManager() {

this.httpClient = HttpClients.createDefault();

}

public String getConfig(String url) throws Exception {

HttpGet request = new HttpGet(url);

request.addHeader("Accept", "application/yang-data+json");

return httpClient.execute(request, response -> {

int status = response.getStatusLine().getStatusCode();

return status == 200 ? EntityUtils.toString(response.getEntity()) : null;

});

}

public void editConfig(String url, String jsonConfig) throws Exception {

HttpPost request = new HttpPost(url);

request.addHeader("Content-Type", "application/yang-data+json");

request.setEntity(new StringEntity(jsonConfig));

httpClient.execute(request);

}

public void close() throws Exception {

if (httpClient != null) {

httpClient.close();

}

}

}

**NetconfClientManagerTest.java**

java

Copy code

import org.junit.Test;

import static org.junit.Assert.\*;

import org.opendaylight.netconf.client.NetconfClientConfiguration;

public class NetconfClientManagerTest {

@Test

public void testGetConfig() throws Exception {

NetconfClientConfiguration clientConfig = ...; // configure client settings

NetconfClientManager clientManager = new NetconfClientManager();

clientManager.connect(clientConfig);

String response = clientManager.getConfig();

assertNotNull(response);

clientManager.close();

}

}

**RestconfClientManagerTest.java**

java

Copy code

import org.junit.Test;

import static org.junit.Assert.\*;

public class RestconfClientManagerTest {

@Test

public void testGetConfig() throws Exception {

String url = "http://<device-ip>/restconf/data/ietf-interfaces:interfaces";

RestconfClientManager clientManager = new RestconfClientManager();

String response = clientManager.getConfig(url);

assertNotNull(response);

clientManager.close();

}

}

**Instructions**

1. **Set up the project**:
   * Create the directory structure as outlined above.
   * Add the pom.xml file to the root directory.
   * Create Java files in the respective directories.
2. **Implement the clients**:
   * Fill in the configuration details for NetconfClientConfiguration in NetconfClientManagerTest.java.
   * Replace <device-ip> with the actual IP address of the network device in RestconfClientManagerTest.java.
3. **Run the tests**:
   * Use a Maven command mvn test to run the unit tests and ensure everything works as expected.

**NetconfClientManager.java**

This class handles the NETCONF operations using the Netconf4J library.

**Key Components**

1. **Class Definition and Variables**:

java

Copy code

public class NetconfClientManager {

private NetconfClientSession session;

* + NetconfClientSession session: This variable holds the session object which is used to communicate with the NETCONF server.

1. **Connecting to the NETCONF Server**:

java

Copy code

public void connect(NetconfClientConfiguration clientConfig) throws Exception {

NetconfClient client = new NetconfClientMinaImpl();

this.session = client.connect(clientConfig).get();

}

* + NetconfClientConfiguration clientConfig: Configuration object containing the settings needed to connect to the NETCONF server.
  + NetconfClient client = new NetconfClientMinaImpl(): Creates a new NETCONF client using the Mina implementation.
  + this.session = client.connect(clientConfig).get(): Establishes a connection to the NETCONF server and stores the session.

1. **Fetching Configuration Data**:

java

Copy code

public String getConfig() throws Exception {

String getConfigRequest = "<rpc message-id=\"1\"><get-config><source><running/></source></get-config></rpc>";

return session.sendRpc(getConfigRequest);

}

* + getConfigRequest: A string representing the XML RPC request to get the running configuration from the NETCONF server.
  + return session.sendRpc(getConfigRequest): Sends the RPC request and returns the response as a string.

1. **Applying Configuration Changes**:

java

Copy code

public String editConfig(String config) throws Exception {

String editConfigRequest = "<rpc message-id=\"2\"><edit-config><target><running/></target><config>" + config + "</config></edit-config></rpc>";

return session.sendRpc(editConfigRequest);

}

* + config: The configuration data to be applied.
  + editConfigRequest: A string representing the XML RPC request to edit the running configuration.
  + return session.sendRpc(editConfigRequest): Sends the RPC request to apply the configuration changes and returns the response.

1. **Closing the Session**:

java

Copy code

public void close() throws Exception {

if (session != null) {

session.close();

}

}

* + This method closes the NETCONF session if it is not already closed.

**RestconfClientManager.java**

This class handles RESTCONF operations using the Apache HttpClient library.

**Key Components**

1. **Class Definition and Variables**:

java

Copy code

public class RestconfClientManager {

private CloseableHttpClient httpClient;

public RestconfClientManager() {

this.httpClient = HttpClients.createDefault();

}

* + CloseableHttpClient httpClient: The HTTP client used to make requests to the RESTCONF API.
  + RestconfClientManager(): Constructor that initializes the HTTP client.

1. **Fetching Configuration Data**:

java

Copy code

public String getConfig(String url) throws Exception {

HttpGet request = new HttpGet(url);

request.addHeader("Accept", "application/yang-data+json");

return httpClient.execute(request, response -> {

int status = response.getStatusLine().getStatusCode();

return status == 200 ? EntityUtils.toString(response.getEntity()) : null;

});

}

* + String url: The URL of the RESTCONF endpoint.
  + HttpGet request = new HttpGet(url): Creates an HTTP GET request to the specified URL.
  + request.addHeader("Accept", "application/yang-data+json"): Adds a header to the request to specify the expected response format.
  + return httpClient.execute(request, response -> { ... }): Executes the request and processes the response. If the status code is 200, it returns the response body as a string.

1. **Applying Configuration Changes**:

java

Copy code

public void editConfig(String url, String jsonConfig) throws Exception {

HttpPost request = new HttpPost(url);

request.addHeader("Content-Type", "application/yang-data+json");

request.setEntity(new StringEntity(jsonConfig));

httpClient.execute(request);

}

* + String url: The URL of the RESTCONF endpoint.
  + String jsonConfig: The JSON formatted configuration data to be applied.
  + HttpPost request = new HttpPost(url): Creates an HTTP POST request to the specified URL.
  + request.addHeader("Content-Type", "application/yang-data+json"): Adds a header to the request to specify the content type of the request body.
  + request.setEntity(new StringEntity(jsonConfig)): Sets the request body to the JSON configuration data.
  + httpClient.execute(request): Executes the request to apply the configuration changes.

1. **Closing the HTTP Client**:

java

Copy code

public void close() throws Exception {

if (httpClient != null) {

httpClient.close();

}

}

* + This method closes the HTTP client if it is not already closed.

**Summary**

* **NetconfClientManager**: Manages NETCONF operations to fetch and apply configurations using XML RPC requests.
* **RestconfClientManager**: Manages RESTCONF operations to fetch and apply configurations using HTTP requests with JSON payloads.
* **Unit Tests**: Basic tests to ensure the functionality of both clients.

Test in online complier and implement.

import java.io.BufferedReader;

import java.io.InputStreamReader;

import java.net.HttpURLConnection;

import java.net.URL;

public class Main {

public static void main(String[] args) {

try {

String url = "http://example.com/restconf/data/ietf-interfaces:interfaces"; // Replace with actual URL

Main restconfClientManager = new Main();

String response = restconfClientManager.getConfig(url);

System.out.println("Response: " + response);

} catch (Exception e) {

e.printStackTrace();

}

}

public String getConfig(String urlString) throws Exception {

URL url = new URL(urlString);

HttpURLConnection conn = (HttpURLConnection) url.openConnection();

conn.setRequestMethod("GET");

conn.setRequestProperty("Accept", "application/yang-data+json");

if (conn.getResponseCode() != 200) {

throw new RuntimeException("Failed : HTTP error code : " + conn.getResponseCode());

}

BufferedReader br = new BufferedReader(new InputStreamReader((conn.getInputStream())));

StringBuilder response = new StringBuilder();

String output;

while ((output = br.readLine()) != null) {

response.append(output);

}

conn.disconnect();

return response.toString();

}

}