Group 2: The Remote Automobile Utility Control Protocol Implementation Requirements

CS544 Spring 2017, Drexel University

Lewis Cannalongo	Max Mattes	Ismail Kuru
lrc74@drexel.edu	mm3888@drexel.edu	ik335@drexel.edu

June 9, 2017

This document specifies the packages and files in the RAUC protocol implementation code base with explanation of the following requirements

- `STATEFUL`
- `CONCURRENT`
- `SERVICE`
- `CLIENT`
- `UI`

The following sections include links to the explanations of the requirements in files at code base in git-hub repository.

1 Client

1.1 Client.java

Purpose of this file is to provide functionality to initialize a connection to the server. It uses GUI functionality provided by ClientGUI.java to control the client side capabilities.

- `CLIENT` : This requirement is satisfied by Client.java in a way that it provides instantiation of Client object with Server port number and Server address.
- `CONCURRENT` : This requirement is satisfied by Client.java in a way that it spawns a Listener thread to handle Servers responses right after establishing connection with server through SSL sockets.

1.2 ClientGUI.java

Client.java driver handles interactions with user through ClientGUI.java object handle. ClientGUI.java uses javax.swing GUI objects to handle user interaction and inputs.

- `UI` : A GUI based on javax.swing panels is provided to user for taking inputs which are used as arguments for building messages to be sent to Server. In addition, user authentication is provided with the inputs through Login-GUI.
- `CLIENT` : This requirement is relavant to CLientGUI.java because GUI for Client enables you to indicate Server address and Server port number.

2 Server

2.1 Server.java

This file is main driver for performing server functionalities.

- `SERVICE` : Main class for starting a server. The server is hardcoded to port 1500. The server keeps client connections in a map of identifier and connection.
- `CONCURRENT` : The server initializes a connection listener which handles incoming connections. It spawns a new thread, ConnectionThread , for handling each different client connection. This makes Server to handle multiple connections at a time.

2.2 ServerGUI.java

This file includes implementation of GUI which aims to provide server's logs.

- `UI` : Entire file's relevant requirement is satisfying UI.

2.3 ConnectionThread.java

This files includes the thread implementation aiming to handle server communication to the client. Main server drivers Server.java spawns ConnectionThread for each client communication.

- `SERVICE` : Main thread to handle server communication to the client, parse and respond to the client messages etc.

3 DFAs

3.1 ClientDFASpec.java

This file includes the definitions for specification of the Client side functionality. It is sub type of DFASpec.java.

- `CLIENT` : Client side functionality is specified in this file.
- `STATEFUL` : Entire file defines valid client side states and admissible valid transitions between these states. All functions([ref ClientSpecDFA sendInit, receiveAuth, sendEstablished, receiveAuth, receiveWaitcmd, receiveWaitqry]) inherited from DFASpec.java defines the valid transition relation between client side states, setting state of client dfa in each of these functions, triggered by Message m.

3.2 ServerDFASpec.java

This file includes the definitions for specification of the Server side functionality. It is sub type of DFASpec.java.

- `SERVICE` : Server side functionality is specified in this file.
- `STATEFUL` : Entire file defines valid client side states and admissible valid transitions between these states. All functions([ref ServerSpecDFA sendAuth, sendWaitcmd, sendWaitqry, receiveInit, receiveEstablished]) inherited from DFASpec.java defines the valid transition relation between server side states, setting state of server dfa in each of these functions, triggered by Message m.

3.3 DFASpec.java

This is an abstract class which defines methods to be implemented by the client and server for to satisfy the specification of the protocol in both end.

- `STATEFUL` : It includes all abstractions for representing states/transitions of DFAs.

4 PDU

All files under this package represents the message implementation used in protocol. They are received and sent accordingly with the Client/Server's state.

4.1 Message.java

Entire file includes definitions of all protocol messages. All messages passed to/from by Client/Server are encapsulated in Message objects.

- `SERVICE`: Messages are core components of the protocol service in a way that they are used in initiating a connection, initializing states of components of the automobile at the client/server side ex: Command / ACK etc.
- `STATEFUL` : The Message objects are the arrows in the DFA, i.e. the operations by which state transitions in the protocol DFA are applied.

4.1.1 AckMessage.java

This file includes implementation of the message which serves for acknowledgment of the Client for success authentication.

- `SERVICE` : Successful Acknowledgement Service
- `STATEFUL` : Server generates [SUCCESSFUL] messages and Client acts accordingly to successful acknowledgement. Details can be found in entire ServerDFASpec.java, ClientDFASpec.java and AckMessage.java.

4.1.2 PermanentErrorMessage.java

This file includes implementation of the message which serves for acknowledgment of the Client for unsuccessful communication.

- `SERVICE` : Error-Acknowledgement.
- `STATEFUL` : Permanent errors are sent by the server to the client and instructs the client that the same message sent at the current protocol state will never be accepted. Details can be found in entire ServerDFASpec.java, ClientDFASpec.java and PermanentErrorMessage.java.

4.1.3 QueryResultMessage.java

Query result is sent by the server in response to a Utility State Query. Query Result messages contain a header chunk and only one content chunk: the result list of a successful query.

- `SERVICE` : Service for Retrieval of Components' States

- `STATEFUL` : It is generated by Server when it is in state of WAITQRY and when Client is in ESTABLISHED state. Details can be found in entire ServerDFASpec.java, ClientDFASpec.java and QueryResultMessage.java.

4.1.4 RequestReceivedMessage.java

Request received message is sent to the client when the server successfully receives a Utility Control Request command.

- `SERVICE` : Acknowledgement of commands validity/arrival to Client by Server.
- `STATEFUL` : Server is in WAITCMD state and Client is already authenticated and in ESTABLISHED state. Details can be found in entire ServerDFASpec.java, ClientDFASpec.java and RequestReceivedMessage.java.

4.1.5 TemporaryErrorMessage.java

Temporary error messages are sent by the server when a command is not received by the server because of a temporal machine state problem.

- `SERVICE` : TemporaryError-Acknowledgement Service: Temporary errors indicate that the client sent a valid command which can be tried again at a later time without modification.
- `STATEFUL` : Details can be found in entire ServerDFASpec.java, ClientDFASpec.java and TemporaryErrorMessage.java.

4.1.6 TerminationMessage.java

- `SERVICE` : Communication Termination Service
- `STATEFUL` : Client is in ESTABLISHED state. Details can be found in entire ServerDFASpec.java, ClientDFASpec.java and TerminationMessage.java.

4.1.7 UserAuthenMessage.java

This file includes implementation of the message which serves for user to authenticate itself in the system.

- `SERVICE` : Authentication Service
- `STATEFUL` : Client is in INIT state and Server is in AUTH state. Details can be found in entire ServerDFASpec.java, ClientDFASpec.java and UserAuthenMessage.java.

4.1.8 UtilityControlReqMessage.java

This file includes implementation of the message which serves to set/change the states of components' attributes associated with an automobile.

- `SERVICE` : Component Control Service
- `STATEFUL` : Client is in ESTABLISHED state and Server is in WAITCMD state. Details can be found in entire ServerDFASpec.java, ClientDFASpec.java and UtilityControlReqMessage.java.

4.1.9 UtilityStateQueryMessage.java

This file includes implementation of the message which serves to query the states of components' attributes associated with an automobile.

- `SERVICE` : Component State Query Service
- `STATEFUL` : Client is in ESTABLISHED state and Server is in WAITQRY state. Details can be found in entire ServerDFASpec.java, ClientDFASpec.java and UtilityStateQueryMessage.java.

4.2 MessageFactory.java

This file includes implementation of factory methods which generates Message objects. [Deprecated]

- `SERVICE` : Messages are core components of the protocol service.

4.3 Chunk.java

This file includes abstract definitions for the most basic unit of Message which is core components of protocol's service.

- `SERVICE` : Entire file is related with `SERVICE` requirement.

4.3.1 ContentChunk.java

This file includes definitions for one type of basic unit of a message of which content of the message comprises.

- `SERVICE` : The relevant requirement the entire file satisfies is partly `SERVICE` because it is one type of basic units of Messages.

4.3.2 HeaderChunk.java

This file includes definitions for one type of basic unit of a message. Header chunk is a tuple where first element of the tuple denotes type of operation and second element of the tuple denots the number of the chunks which are going to follow header chunk.

- `SERVICE` : Entire file includes definitions for one type of basic unit of a message which represents header of the file.
- `STATEFUL` : This file defines the Message Type which is partly a part of being statefulness. The reason is that Message Type is tightly coupled with Client/Server DFAs' state when they are processing the message taken from other end. Details can be found in entire ServerDFASpec.java, ClientDFASpec.java [details in `switch case` branching on message opcodes/types OP_X].

5 Components

All files under this package serve for the of representation of an automobile, components of the automobile and commands that are applied on the status of these components. All of them satisfy the `SERVICE` requirement.

- Automobile.java : This file includes the representation of the Automobile object.

- Command.java : Commands are operations applied on states of components of an automobile. Command objects are generated from Message objects, [details annotated in createCommand method in Command.java] and applied on components of an automobile. It is part of the protocol's service.
- Component.java : This file includes abstract class for a component that can be part of an automobile controlled by the protocol. It is part of the protocol's service because Component.java specifies that each component sub type, ex:AC.java, needs to implement applyCommand, [details in Component.java and all its sub-type implementations], which enables applying a command to change component's state of attributes.
- Factory.java : This file includes instantiation of couple of objects (User, Automobile, Components etc.) to demonstrate that protocol is servicing properly.
- Query.java : This file includes implementation of read-only operation to check the status of an automobile's components. This is part of the the protocol's service.
- AC.java-Radio.java-Door.java : This is representation of one of the components that an automobile can have. It is sub type of the Component.java
- ComponentFactory.java : Service similar to Factory.java is provided for Component(AC, Door, Radio) generation with this file.
- Attribute.java : This file includes implementation of representation of attributes of components such as power, lock, temperature. It is part of the components which are part of data service of protocol.

6 QueryExecutor.java

This file includes implementation of read-only commands. QueryExecutor is run by Server when it gets UtilityStateQuery message.

- `STATEFUL` : Server needs to be in WAITQRY state when it gets executes query from Client. Details are in ServerDFASpec.java.
- `SERVICE` : It includes the implementation which is one of the backbones and provides Read-Only-Query of the data resides in ACS(Server).

7 Users

7.1 User.java

This file includes representation of user which is one of the core parts of the protocol service.

- `SERVICE` : Authentication of an user with user name and password, [details in User.java], is one important part of the protocol's service. Each Client object is created with its user name and password, [details annotated in Client.java] and creates/sends authenticate message, UserAuthMessage.java [details annotated in Client.java],

Appendix A Comment-Out-Structure

We incorporate the analysis of these requirements to each file in code base as header explanations. The structure of the header is :

```
/* ============================================================================
* CS544 - Computer Networks - Drexel University, Spring 2017
* Protocol Implementation: Remote Automobile Utility Control
* Group 2:
* - Ismail Kuru
* - Max Mattes
* - Lewis Cannalongo
**********************************************
* File name: Message.java
* ********************************************
* Definition: Definitions of all protocol messages. All messages passed
* from one end to another are encapsulated in a Message object.
* ******************************************************
* Requirements:
* - STATEFUL : the Message objects are the arrows in the DFA, i.e. the operations
*   by which state transitions in the protocol DFA are applied.
* - SERVICE : the messages are an important part in the protocol service
*   definition: initiating a connection, initializing states of components of the
*   automobile at the client/server side ex: Command / ACK etc.
*
* ============================================================================
*/
```

In addition, If specific code segments address relates the implementation to a specific requirement, a comment-out section including the explanation of this relation is added.

```
/* ============================================================================
* Requirement:
* - STATEFUL :  Comment on next line in terms of its relation to being stateful
* ============================================================================
*/
```