Low Level Design Document

for

AGORA

Version 1.1

A Versatile Environment for the Development of IntelliDrive Applications

(Visual … Extensible … Rule-Based)

Department of Computer Science, Western Michigan University



Table of Contents

Table of Contents ..... **ii**
Revision History ..... **ii**
1. Introduction **1**
1.1 Purpose 1
1.2 Document Conventions 1
1.3 Intended Audience and Reading Suggestions 1
1.4 References 1
2. System Use Cases **2**
3. Detailed System Design **7**
3.1 Super Nodes 7
3.2 Regional Manager 14
3.3 VIS 21
3.4 TMC 30
3.5 A3/Gov 2.0 34
Appendix A: Glossary **38**

Revision History

Name	Date	Reason For Changes	Version
Vinay B Gavirangaswamy	02/14/2010	Initial Draft	1.0
Vinay B Gavirangaswamy	02/16/2010	Changed project name from VII to AGORA, changed paragraph formatting, and reformatted cover page.	1.1

1. Introduction

1.1 Purpose

AGORA refers to a set of application and infrastructure which constitutes intellidrive environment that is being developed at Computer Science Department, Western Michigan University. This infrastructure consists of On-Board Equipment (OBE), Road Side Equipment (RSE), and Traffic Management Centers (TMC), which work together to increase the safety and efficiency of the transportation network. Current system that is deployed across the university consists of initial versions RSE's, OBE's and VIS. And Current scope of work involves enhancement of VIS to include more features and functionality through incorporating different capability/applications. And it will also include development of TMC (Traffic Management Center). However the goal of this program is to facilitate MDOT in enforcing a safe and an efficient transportation system.
This document details different components that each system is composed of and their technical design, implementation strategies.

1.2 Document Conventions

TBD

1.3 Intended Audience and Reading Suggestions

Document is primarily intended for members of MDOT team which consists of graduate students working under the guidance of Dr. Ala Al-Fuqaha and Dr. Dionysios Kountanis.

1.4 References

- Project Proposal Document
- System Specification for AGORA
- System Architecture and High Level Design for AGORA

2. System Use Cases

Name:	SuperNode		
Number:	1.0		
Author:	Mrinal Khanvilkar	**Date:**	July 14, 2009
Description:	1. For other super nodes: It checks whether the centroid set requested by another supernode is already blocked by itself or by another supernode. If it is not blocked, it means the centroid set can be allocated. 2. For regional managers: It checks with itself and other super nodes whether the centroid set is blocked. If it is not blocked then return the centroid set to the regional manager after blocking it. 3. For vehicles: It returns the regional manager associated with the closest centroid to the vehicles location.		
Actors:	SuperNode, Regional Manager, Car		
Pre-conditions:	None		
Course of Events:	1. Load up neighboring supernodes from a file (which are static). Upon receiving a request from another supernode. a) Parse the requested centroid set and course of action from the request. i) If action == "lock" then check whether the centroid set is free 1) If it is free return a positive answer. 2) if it is locked already then return a negative answer. ii) If action == "unlock" then unlock the centroid set which was previously locked. iii) if action == "delete" then delete the associated regional manager (from the request) from the regional manager database. iv) if action == "update" then update the associated regional manager's time (from the request) in the regional manager database. Upon receiving a request from a regional manager. a) Check if regional manger already has a centroid. If yes then update its time in the regional manager table and send "update" to other supernodes with the regional manager's information (IP and portNO), else continue with b). b) Check if any free centroids. v) Pick a free centroid. vi) Contact other supernodes about this centroid set and ask them to lock it if it is free. vii) If the response from all the supernodes is positive then grant the centroid set to the requesting regional manager. viii) If response from any of the supernodes if negative, contact the rest to unlock the centroid set and start with a). c) If no free centroids left, ignore the requesting regional manager. Upon receiving a request from a vehicle.		

	a) Parse the vehicles location from the request. b) Get the closest centroid set to the vehicles location. c) Lookup the centroid set and the corresponding regional manager and return to the vehicle. Garbage Collection: a) Remove the regional managers from the regional manager database whose update hasn't occurred from a given time interval, and forward the request to delete the same to the other supernodes.
Post-conditions:	
Alternatives/ Exceptions:	
Notes:	

Name:	Regional Manager		
Number:	1.0		
Author:	Mrinal Khanvilkar	**Date:**	July 14, 2009
Description:	1. For super nodes: It keeps on requesting a centroid set from a randomly selected supernode. The requesting takes place even if it has a centroid set associated with it. This enables the update of the regional managers. 2. For vehicles: It returns the other vehicles currently registered with this regional manager. This information includes (VIN, IP, portNO, latitude, longitude, passkey (for encryption)).		
Actors:	SuperNode, Regional Manager, Car		
Pre-conditions:	None		
Course of Events:	1. Load up supernodes from a file (which are static). 2. Select a random supernode. 3. Request the selected supernode for a centroid set. If selected supernode is down goto step 2. a) If the supernode returns a centroid set ID, the vehicles in the region will register with it later. b) If the supernode returns NULL, then step c) c) Repeat from 3 after a given time interval. Upon receiving a request from a vehicle. a) Parse the vehicles information (VIN, IP, portNO, latitude, longitude, passkey (for encryption)) from the request and register the vehicle in its vehicle database, if it's not present. Else update the vehicles time in the vehicle database. b) Send the other vehicles registered with it to the requesting vehicle. Garbage Collection: b) Remove the vehicles from the vehicles database whose update hasn't occurred from a given time interval.		
Post-conditions:			
Alternatives/ Exceptions:			
Notes:			

Name:	VIS		
Number:	1.0		
Author:	Mrinal Khanvilkar, Jamie Groos	**Date:**	May 14, 2009
Description:	1. For super nodes: It keeps on requesting a regional manager from a randomly selected supernode. The requesting takes place even if it has an associated regional manager. This enables the vehicle to get regional managers according to its current location. 2. For vehicles: It keeps on forwarding its own information to the vehicles returned by the associated regional manager. Rules are setup beforehand, and the fire according to the context of the this vehicle (i.e. the vehicle itself and the other vehicles in its neighborhood database).		
Actors:	SuperNode, Regional Manager, Car		
Pre-conditions:	None		
Course of Events:	1. Load up supernodes from a file (which are static). 2. Select a random supernode. 3. Request the selected supernode for a regional manager. If selected supernode is down goto step 2. a) The supernode will return a regional manager (IP portNO). Send a request to the regional manager to register along with (VIN, IP, portNO, latitude, longitude, passkey (for encryption)). b) If the supernode doesn't return a regional manager, do not overwrite the previous regional manager if present. c) The regional manager will send the other vehicles information (VIN, IP, portNO, latitude, longitude, passkey (for encryption)). Overwrite this information in the neighborhood database. The core values with (VIN, IP, portNO, latitude, longitude, passkey (for encryption)) + other values which are configurable by the user. d) Repeat from 3 after a given time interval. Sending information to another vehicle. a) If the neighborhood database is not empty. Send the encrypted (using the individual passkeys) core information to the other vehicles over UDP. Upon receiving a request from a vehicle. a) Parse the vehicles information from the request and register it in the neighborhood database.		
Post-conditions:			
Alternatives/ Exceptions:			
Notes:			

Name:	TMC		
Number:	1.0		
Author:	Vinay B Gavirangaswamy	**Date:**	Feb 14, 2010
Description:	TMC's are designed to serve number of purposes while integrating with other architecture building block and system building blocks. One of the functionality that TMC does is CRUD operation on hazard data. This is exposed as web services.		
Actors:	TMC, VIS, A3		
Pre-conditions:	Client web service request to getAllHazard()		
Course of Events:	▪ Receive client request to list all hazards around given centroid ▪ TMCServiceSEI handles the request and fetches all hazards that are within 2 mile radius of given location. ▪ Fetch all hazards within 2 mile radius of given location and populate the response.		
Post-conditions:	None		
Alternatives/ Exceptions:	Exception is thrown if no hazard is available around given location		
Notes:			

Name:	TMC		
Number:	1.0		
Author:	Vinay B Gavirangaswamy	**Date:**	Feb 14, 2010
Description:	TMC's are designed to serve number of purposes while integrating with other architecture building block and system building blocks. One of the functionality that TMC does is CRUD operation on hazard data. This is exposed as web services.		
Actors:	TMC, VIS, Gov 2.0		
Pre-conditions:	Client web service request to postHazard()		
Course of Events:	▪ Receive client request to store hazard around at centroid ▪ TMCServiceSEI handles the request and store hazard with its corresponding hazard code. ▪ Return success if hazard was stored successfully or false otherwise.		
Post-conditions:	None		
Alternatives/ Exceptions:	Exception is thrown in case of error.		
Notes:			

3. Detailed System Design

3.1 Super Nodes

3.1.1 Design Description

This service keeps an inventory of the set of centroids which are used in the system. The centroids are locations with co-ordinates. There might be more than one centroid in a set. A centroid set is typically allocated to a region. The set with more than two centroids can be locked for regions with high traffic density. These centroid sets have to allocate to the regional mangers for them to be active. The supernodes all maintain the same inventory of centroid sets (global view). i.e. at any given point of time they have the same inventory throughout. The global view of the used centroids is maintained as follows:

1. The regional manager pings any supernode which is randomly selected.
2. Now the supernode checks whether the regional manager already has the centroid set allocated to it. If not, the supernode contacts the other supernodes with the regional manager ID and the any unused centroid ID. The exact message is given as:

   "{CentroidSetID}&{RegionalManagerHostName}&{RegionalManagerPortNo}&{RegionalManagerIPAddress}&{SuperNodeHostName}"

   If the regional manager already has the centroid set allocated to it, the given supernode updates its used centroid set list to reflect the time of update. This is crucial since if there is no update from the regional manager, then its means that the regional manager has failed. The supernode uses the same message to communicate with its peers so that they too can update their used centroid lists with the time update from the regional manager thereby updating the entire global view.
3. Upon receiving the request from the original supernode, it's peer checks if that particular centroid set ({CentroidSetID}) is still in its unused list. If it is, it is marked as taken by the given regional manager({RegionalManagerIPAddress}). If it is used, then it checks if it was used by the same regional manager, upon which it simply updates the update time of the regional manager thereby updating the global view. But if the used centroid is taken by some different regional manager then it contacts the originating regional manager with the current Regional managers information and basically says,"hey, this centroid is already taken". The exact return message is given by "yes" (which means centroid taken).

   The originating supernode upon receiving this return message, contacts the rest of its peers with the following message:

   "{CentroidSetID}&{RegionalManagerHostName}&{SuperNodeHostName}"

   If you notice the message in point 2, has more than 3 fields and that is the distinguishing factor. The peers upon receiving this message know that they have to remove the regional manager allotment from their used centroid list (if they have previously allotted to it).
4. To sum it up, when the supernode receives a ping from the regional manager, if there is an unused centroid with it, it allocates the centroid set to the regional manager and then contacts its peers to allocate the same to the same regional manager, unless one peer replies with "centroid taken {yes}" in which case the originating supernode has to rollback the allotment from itself and the rest of the peers used centroid list. Also when it receives a ping from a regional manager and a centroid set is already allocated to it, the it updates it's

own used centroid list with the “update time” and make sure that its peers too update the time in their used centroid list.

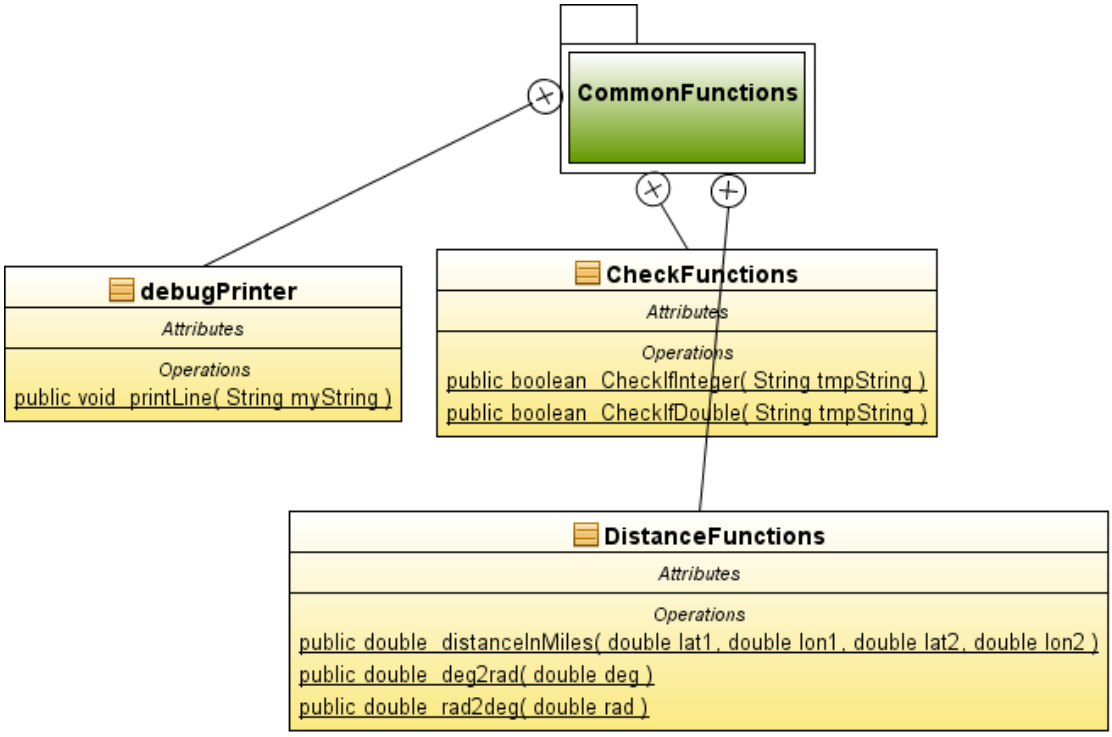
5. Consider a scenario if the regional manager doesn’t ping a supernode for a long time. There is a garbage collection running on every supernode. Since a global view is maintained on all the supoernodes, it will remove the regional manager from its used centroid list and pass the same message i.e.:

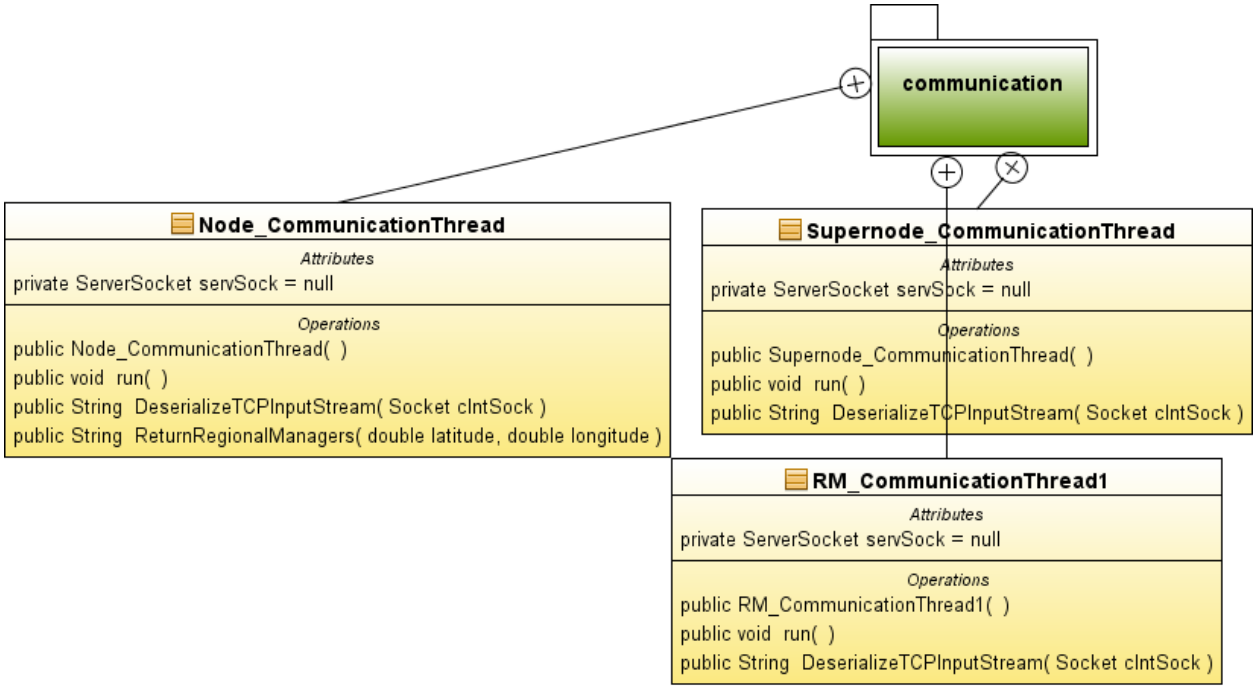
   “{CentroidSetID}&{RegionalManagerHostName}&{SuperNodeHostName}”

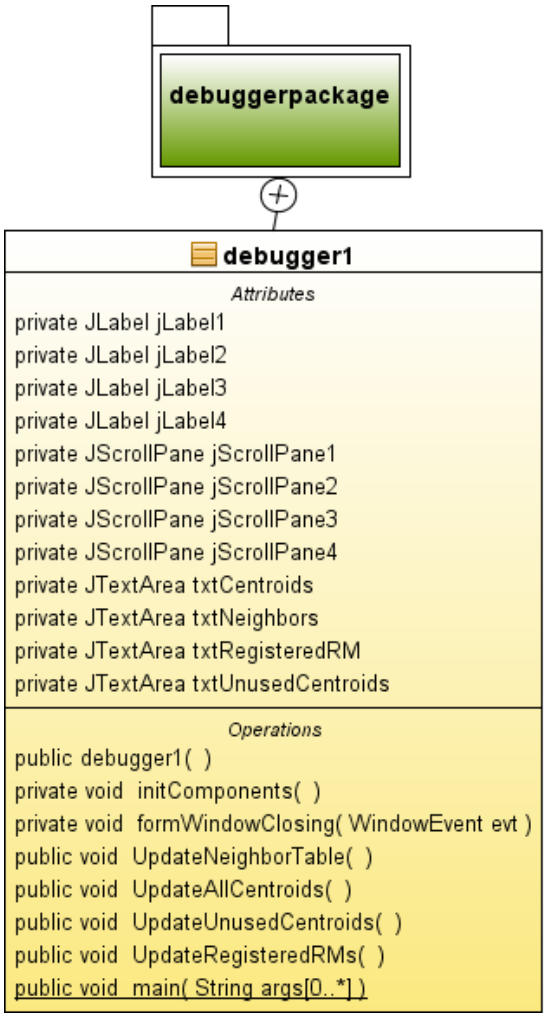
   to its peers in order to remove the particular supernode. So now the global view has been ridded of the regional manager who has not pinged for a long time (it may be because the regional manager has failed). In this case the used centroid allocated to that particular regional manager is returned back to the unused state and now it is free to be allocated to the next regional manager which doesn’t yet have a centroid allocated to it. All this together maintain the same global list of used centroid list across all supernodes.

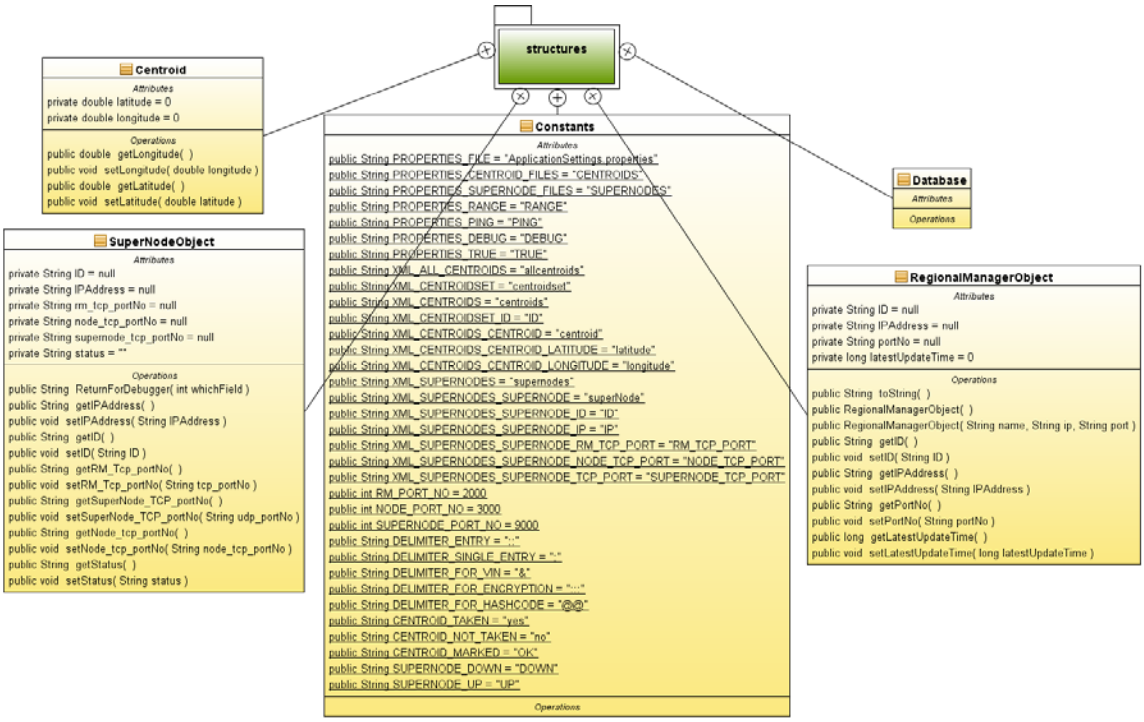
3.1.2 Design and Implementation Constraints

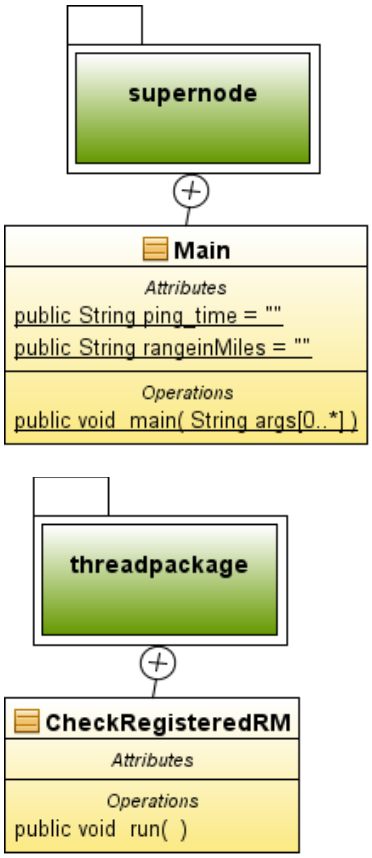
3.1.3 Class Diagram

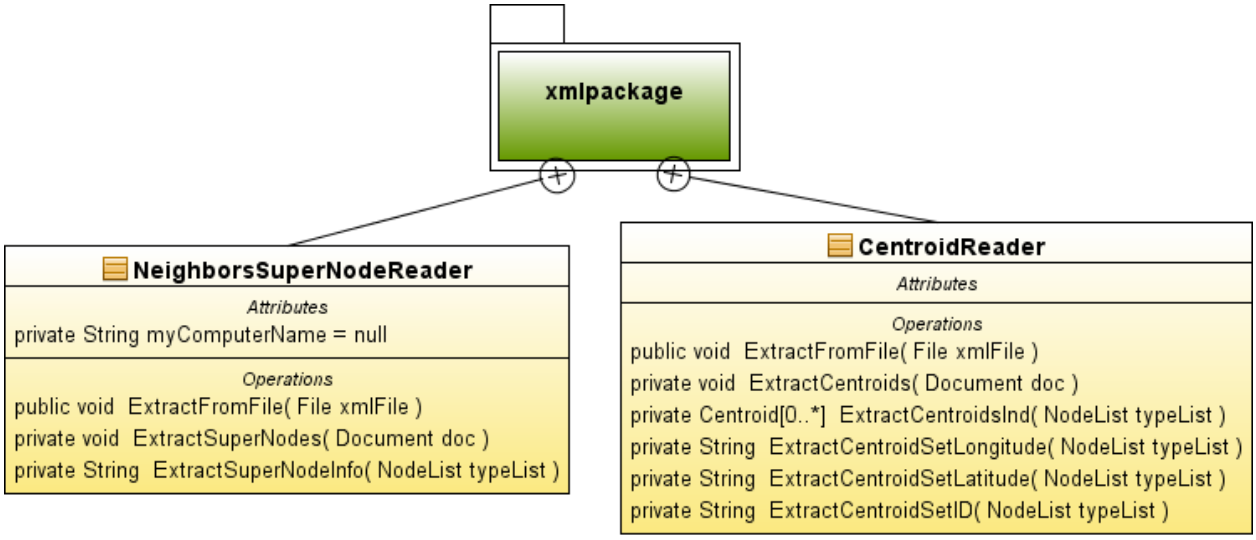



communication
Node_CommunicationThread
Attributes
private ServerSocket servSock = null
Operations
public Node_CommunicationThread( )
public void run( )
public String DeserializeTCPInputStream(Socket clntSock)
public String ReturnRegionalManagers(double latitude, double longitude)
Supernode_CommunicationThread
Attributes
private ServerSocket servSock = null
Operations
public Supernode_CommunicationThread( )
public void run( )
public String DeserializeTCPInputStream(Socket clntSock)
RM_CommunicationThread1
Attributes
private ServerSocket servSock = null
Operations
public RM_CommunicationThread1( )
public void run( )
public String DeserializeTCPInputStream(Socket clntSock)


debuggerpackage
debugger1
Attributes
private JLabel jLabel1
private JLabel jLabel2
private JLabel jLabel3
private JLabel jLabel4
private JScrollPane jScrollPane1
private JScrollPane jScrollPane2
private JScrollPane jScrollPane3
private JScrollPane jScrollPane4
private JTextArea txtCentroids
private JTextArea txtNeighbors
private JTextArea txtRegisteredRM
private JTextArea txtUnusedCentroids
Operations
public debugger1( )
private void initComponents( )
private void formWindowClosing(WindowEvent evt)
public void UpdateNeighborTable( )
public void UpdateAllCentroids( )
public void UpdateUnusedCentroids( )
public void UpdateRegisteredRMs( )
public void main(String args[0..*])


structures
Centroid
Attributes
private double latitude = 0
private double longitude = 0
Operations
public double getLongitude( )
public void setLongitude(double longitude)
public double getLatitude( )
public void setLatitude(double latitude)
SuperNodeObject
Attributes
private String ID = null
private String IPAddress = null
private String rm_tcp_portNo = null
private String node_tcp_portNo = null
private String supernode_tcp_portNo = null
private String status = ""
Operations
public String ReturnForDebugger(int whichField)
public String getIPAddress( )
public void setIPAddress(String IPAddress)
public String getID( )
public void setID(String ID)
public String getRM_Tcp_portNo( )
public void setRM_Tcp_portNo(String tcp_portNo)
public String getSuperNode_TCP_portNo( )
public void setSuperNode_TCP_portNo(String udp_portNo)
public String getNode_tcp_portNo( )
public void setNode_tcp_portNo(String node_tcp_portNo)
public String getStatus( )
public void setStatus(String status)
Constants
Attributes
public String PROPERTIES_FILE = "ApplicationSettings.properties"
public String PROPERTIES_CENTROID_FILES = "CENTROIDS"
public String PROPERTIES_SUPERNODE_FILES = "SUPERNODES"
public String PROPERTIES_RANGE = "RANGE"
public String PROPERTIES_PING = "PING"
public String PROPERTIES_DEBUG = "DEBUG"
public String PROPERTIES_TRUE = "TRUE"
public String XML_ALL_CENTROIDS = "allcentroids"
public String XML_CENTROIDSET = "centroidset"
public String XML_CENTROIDS = "centroids"
public String XML_CENTROIDSET_ID = "ID"
public String XML_CENTROIDS_CENTROID = "centroid"
public String XML_CENTROIDS_CENTROID_LATITUDE = "latitude"
public String XML_CENTROIDS_CENTROID_LONGITUDE = "longitude"
public String XML_SUPERNODES = "supernodes"
public String XML_SUPERNODES_SUPERNODE = "superNode"
public String XML_SUPERNODES_SUPERNODE_ID = "ID"
public String XML_SUPERNODES_SUPERNODE_IP = "IP"
public String XML_SUPERNODES_SUPERNODE_RM_TCP_PORT = "RM_TCP_PORT"
public String XML_SUPERNODES_SUPERNODE_NODE_TCP_PORT = "NODE_TCP_PORT"
public String XML_SUPERNODES_SUPERNODE_TCP_PORT = "SUPERNODE_TCP_PORT"
public int RM_PORT_NO = 2000
public int NODE_PORT_NO = 3000
public int SUPERNODE_PORT_NO = 9000
public String DELIMITER_ENTRY = ":.:"
public String DELIMITER_SINGLE_ENTRY = ":"
public String DELIMITER_FOR_VIN = "&"
public String DELIMITER_FOR_ENCRYPTION = ":::"
public String DELIMITER_FOR_HASHCODE = "@@"
public String CENTROID_TAKEN = "yes"
public String CENTROID_NOT_TAKEN = "no"
public String CENTROID_MARKED = "OK"
public String SUPERNODE_DOWN = "DOWN"
public String SUPERNODE_UP = "UP"
Operations
Database
Attributes
Operations
RegionalManagerObject
Attributes
private String ID = null
private String IPAddress = null
private String portNo = null
private long latestUpdateTime = 0
Operations
public String toString( )
public RegionalManagerObject( )
public RegionalManagerObject(String name, String ip, String port)
public String getID( )
public void setID(String ID)
public String getIPAddress( )
public void setIPAddress(String IPAddress)
public String getPortNo( )
public void setPortNo(String portNo)
public long getLatestUpdateTime( )
public void setLatestUpdateTime(long latestUpdateTime)


supernode
Main
Attributes
public String ping_time = ""
public String rangeinMiles = ""
Operations
public void main(String args[0..*])
threadpackage
CheckRegisteredRM
Attributes
Operations
public void run( )



Description:

Component Name	Description

3.1.4 Sequence Diagram

TBD

3.1.5 Activity Diagram

TBD

3.1.6 External Interface Requirements

TBD

3.1.6.1 User Interfaces

TBD

3.1.6.2 Hardware Interfaces

TBD

3.1.6.3 Software Interfaces

TBD

3.1.6.4 Communications Interfaces

TBD

3.2 Regional Manager

3.2.1 Design Description

The Regional managers maintain the database of the vehicles, roadside, point of interest for a particular set of centroids. When the regional manager boots up, it requests a centroid set from a random supernode. As discussed before, since all the supernodes maintain the same inventory (global view), the random selection of the supernode doesn't create discrepancies. If all the centroids sets are taken, then the regional manager keeps on requesting a centroid set.
The message from the regional manager to the supernode is given as follows:
"{RegionalManagerHostName}&{RegionalManagerPortNo}"
If the reply from the supernode is "yes",it means that all the centroids are already taken. But if the message from the supernode is "{CentroidID}", which means that centroid set with that particular ID is now allocated to it. It is now that the regional manager will be accepting registration requests from cars in the centroid set region.
The regional manager gets registration requests from cars as follows:
"{VIN}&{CAR_PortNO_Receive}&{CAR_Encryption_Password}&{CAR_Latitude}&{CAR_Longitude}"
Upon receiving this request, the regional manager has two options to deal with.

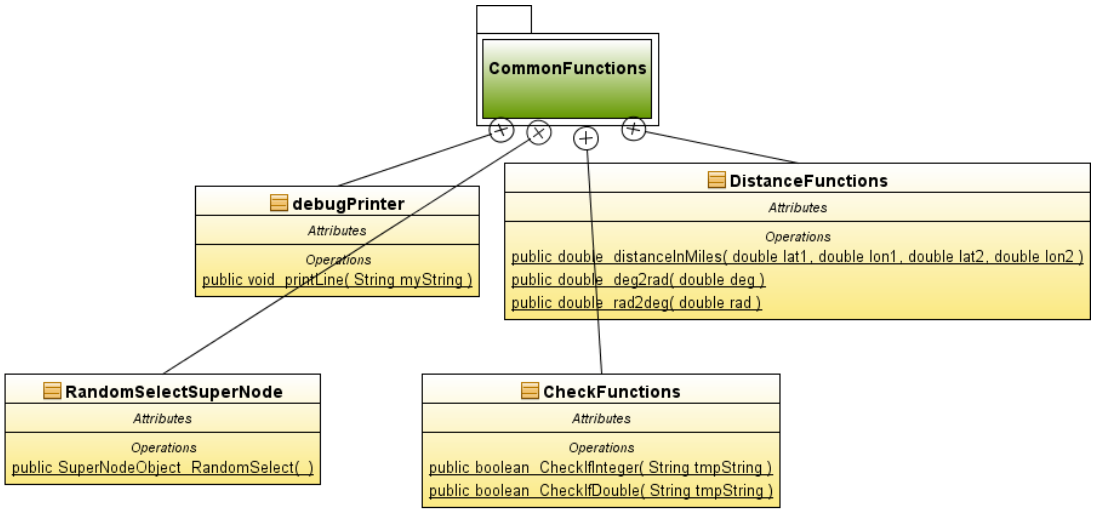
1. The car is already registered with it. In this case, it just updates the "time update" of the car. This is necessary since if that car hasn't pinged the regional manager for a long time, it implies that it has moved into a different region. It has to be removed so that the other cars don't get outdated information.
2. The car is not registered with it. In this case we registered the car in the regional manager's database. The database of the regional manager consists of only the core data of the elements like IP, latitude, longitude and password for encryption.

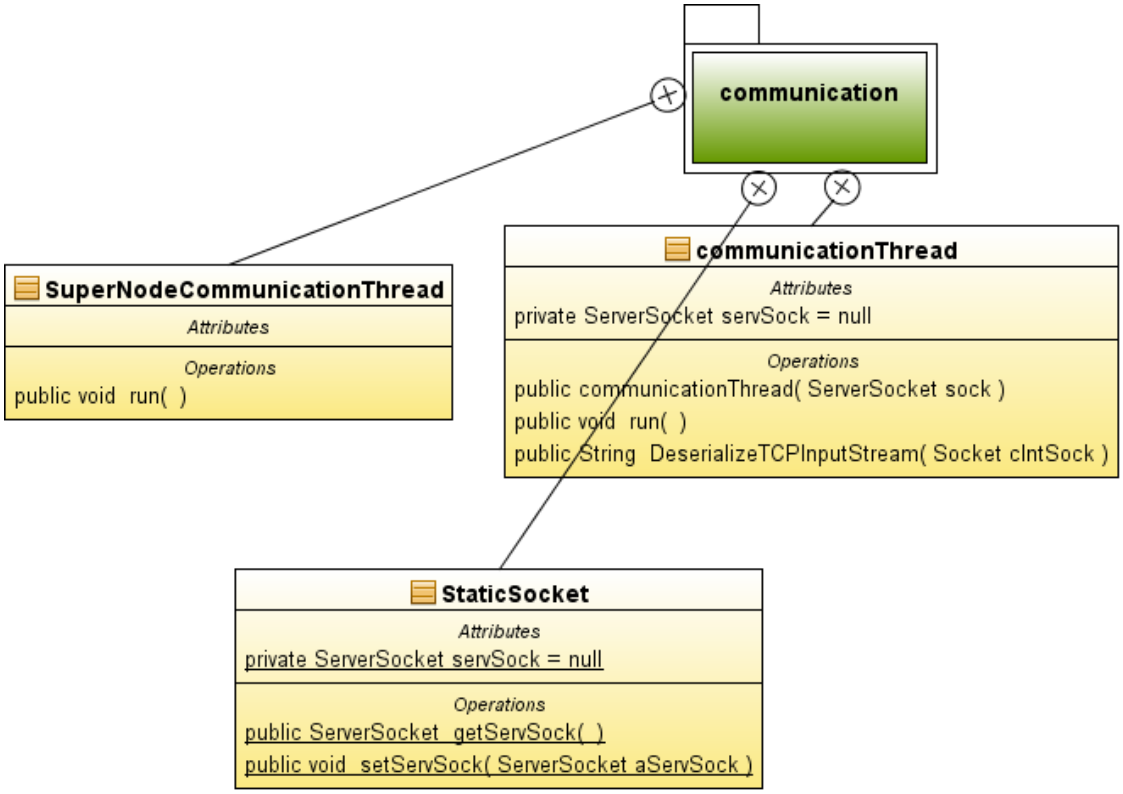
In both the cases, the return message from the regional manager to the car is given by:

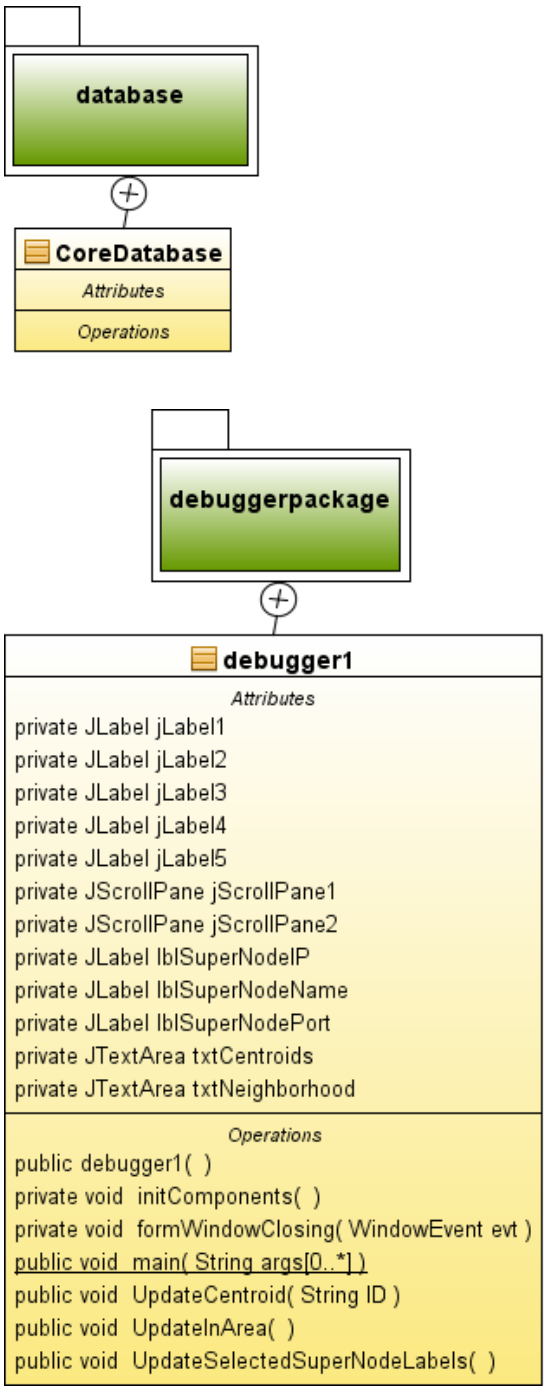
"{VIN_1}&{CAR_PortNO_IPAddress_1}&{CAR_PortNO_Receive_1}&{CAR_Encryption_Password_1}::{VIN_2}&{CAR_PortNO_IPAddress_2}&{CAR_PortNO_Receive_2}&{CAR_Encryption_Password_2}::…"
The return message contains the contact information about all the other cars in the regional manager's domain. In the above message example we have shown two vehicles, but in reality it can be more than that. Now the car has all the contact details of the other vehicles to which it contact and transfer information. Let's discuss that further in the next section.

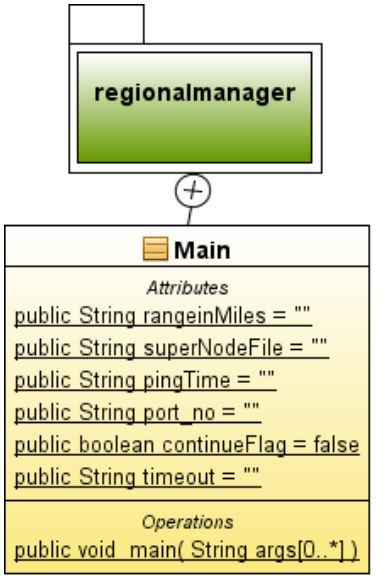
3.2.2 Design and Implementation Constraints

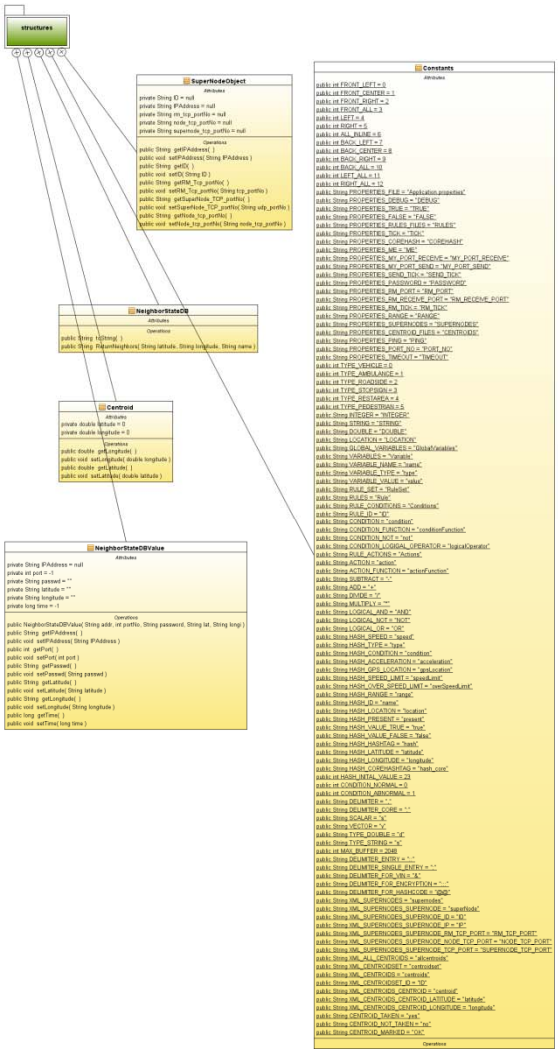
3.2.3 Class Diagram

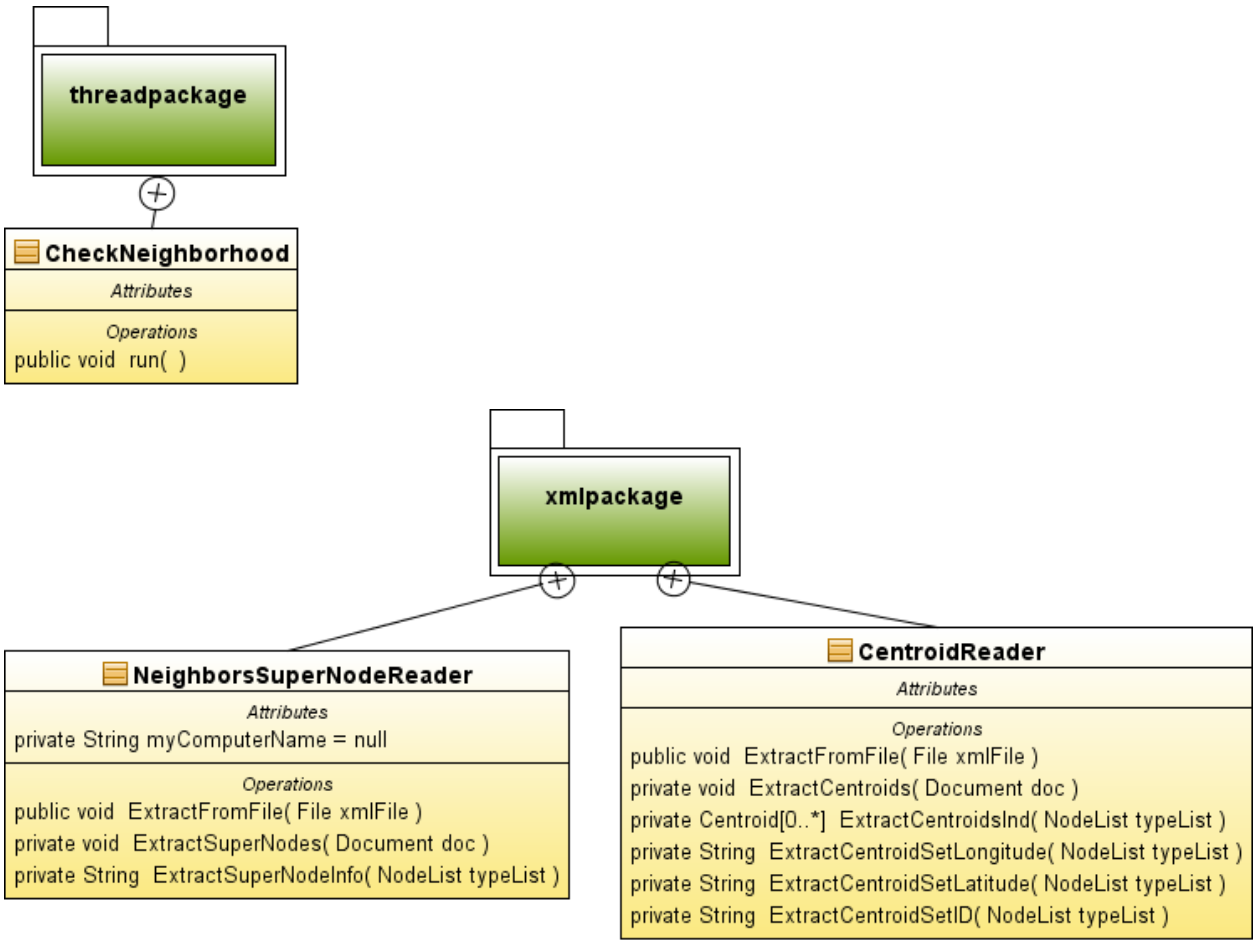



communication
SuperNodeCommunicationThread
Attributes
Operations
public void run( )
communicationThread
Attributes
private ServerSocket servSock = null
Operations
public communicationThread(ServerSocket sock)
public void run( )
public String DeserializeTCPInputStream(Socket clntSock)
StaticSocket
Attributes
private ServerSocket servSock = null
Operations
public ServerSocket getServSock( )
public void setServSock(ServerSocket aServSock)


database
CoreDatabase
Attributes
Operations
debuggerpackage
debugger1
Attributes
private JLabel jLabel1
private JLabel jLabel2
private JLabel jLabel3
private JLabel jLabel4
private JLabel jLabel5
private JScrollPane jScrollPane1
private JScrollPane jScrollPane2
private JLabel lblSuperNodeIP
private JLabel lblSuperNodeName
private JLabel lblSuperNodePort
private JTextArea txtCentroids
private JTextArea txtNeighborhood
Operations
public debugger1( )
private void initComponents( )
private void formWindowClosing(WindowEvent evt)
public void main(String args[0..*])
public void UpdateCentroid(String ID)
public void UpdateInArea( )
public void UpdateSelectedSuperNodeLabels( )


regionalmanager
Main
Attributes
public String rangeinMiles = ""
public String superNodeFile = ""
public String pingTime = ""
public String port_no = ""
public boolean continueFlag = false
public String timeout = ""
Operations
public void main(String args[0..*])

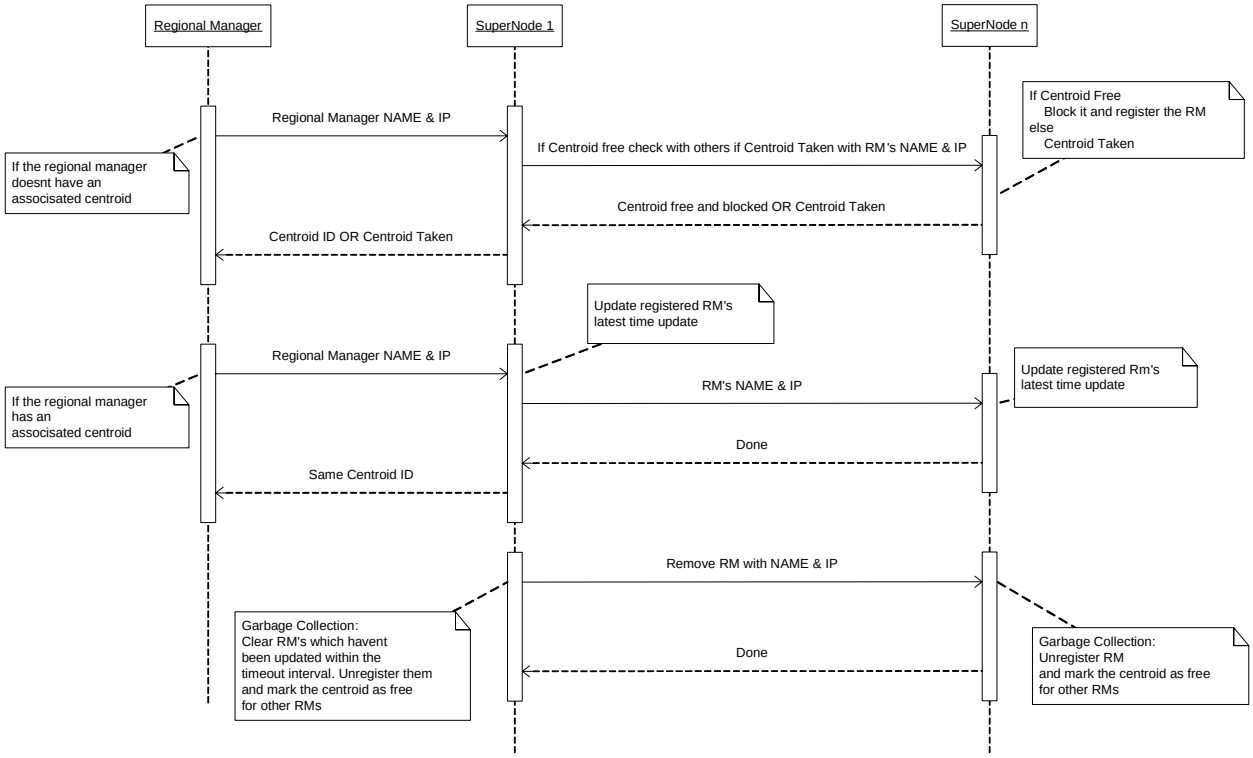

structures
SuperNodeObject
NeighborStateDB
Centroid
NeighborStateDBValue
Constants



Description:

Component Name	Description

3.2.4 Sequence Diagram



Description:

3.2.5 Activity Diagram

TBD

3.2.6 External Interface Requirements

3.2.6.1 User Interfaces

TBD

3.2.6.2 Hardware Interfaces

TBD

3.2.6.3 Software Interfaces

TBD

3.2.6.4 Communications Interfaces

TBD

3.3 VIS

The cars application, while booting up, contacts a random supernode (to facilitate load balancing) with its current latitude and longitude. Since all the supernodes have the same global view of all the used centroid set list, there would be no discrepancies. The exact message is gven by:

"{VIN}&{CAR_Latitude}&{CAR_Longitude}"

The supernode upon receiving this message looks up the centroid set and corresponding regional manager which is closest to this location and returns the regional managers contact details. The exact message is given by:

"{RegionalManagerHostName_1}&{RegionalManagerIPAddress_1}&{RegionalManagerPortNo_1}::
{RegionalManagerHostName_2}&{RegionalManagerIPAddress_2}&{RegionalManagerPortNo_2}::
…"

The car application on receiving this information contacts the regional manager and registers itself in the manager's database (as discussed in the previous section of regional managers. While registering the car gets the information about the other vehicles in the manager's database and saves it for future reference. Now the car, having this information, contacts the other vehicles and transmits its information using encryption over UDP. The exact message is given by:

"{VIN}&{CAR_Encryption_Password}&{CAR_PortNO_Receive}&{Hash_field_1};{scalar/vector(s/v)}
;{ string/double(s/d)};{value}::{Hash_field_2};{scalar/vector(s/v)};{string/double(s/d)};{value}::…"

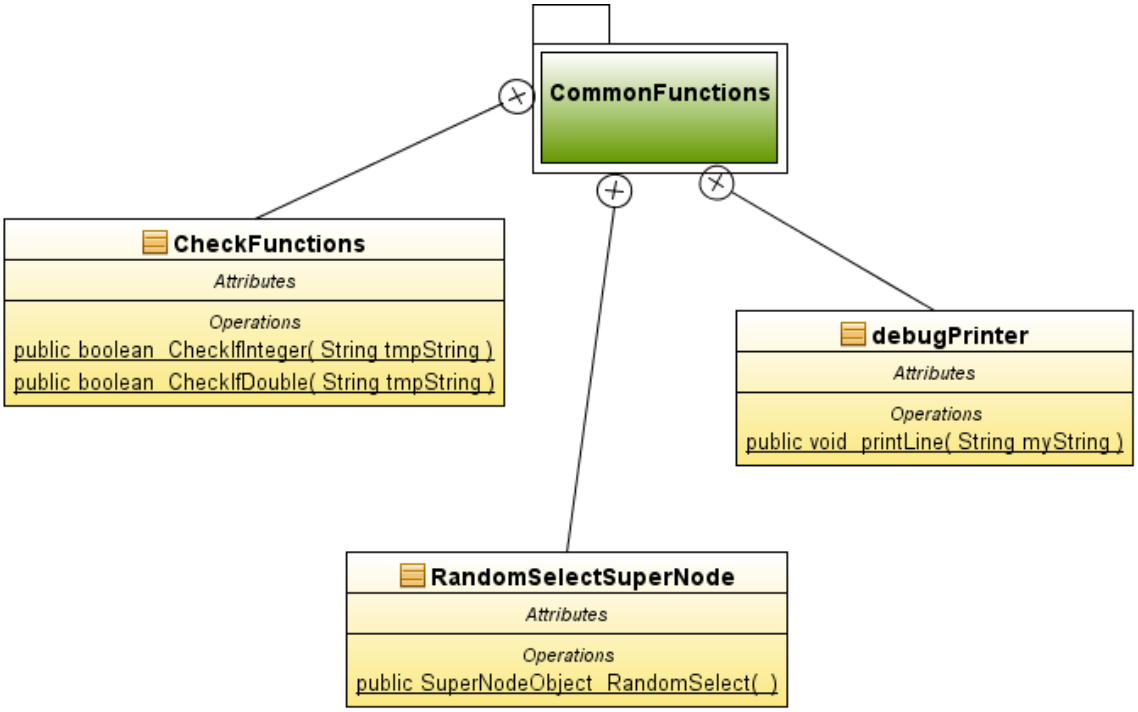
As we can see the first 3 fields are static in nature, whereas the rest of the fields depends upon the car context. The cars context is a hashtable with key/value pairs with keys being the fields and values being the objects/values (they can be scalar or vector of string and double).

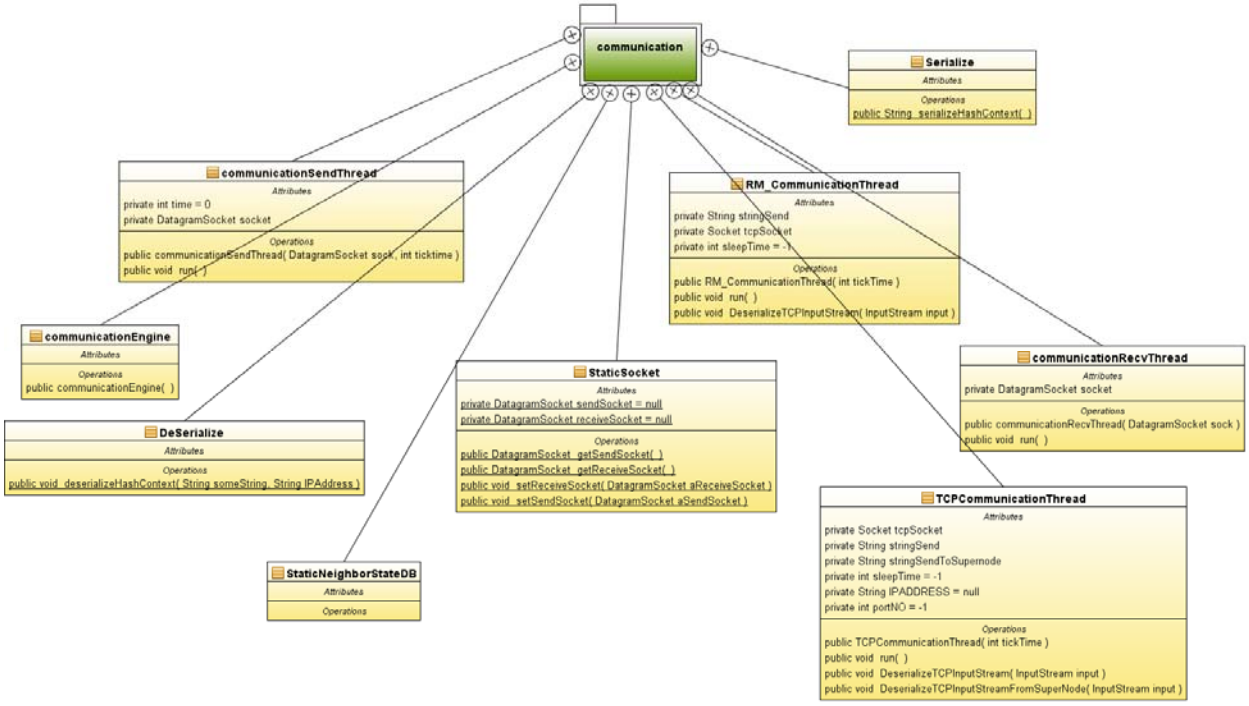
Since this communication is done through UDP, we don't wait for an acknowledgement from the other vehicles. The other vehicles upon receiving this message populate their own context with the details, thus enabling inter-vehicular communication.

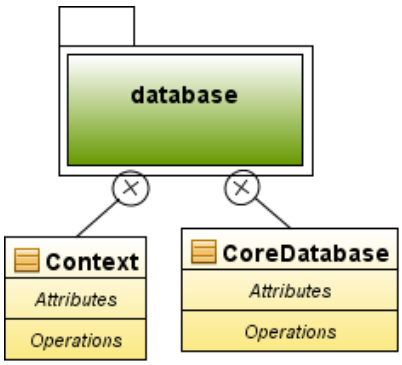
3.3.1 Design Description

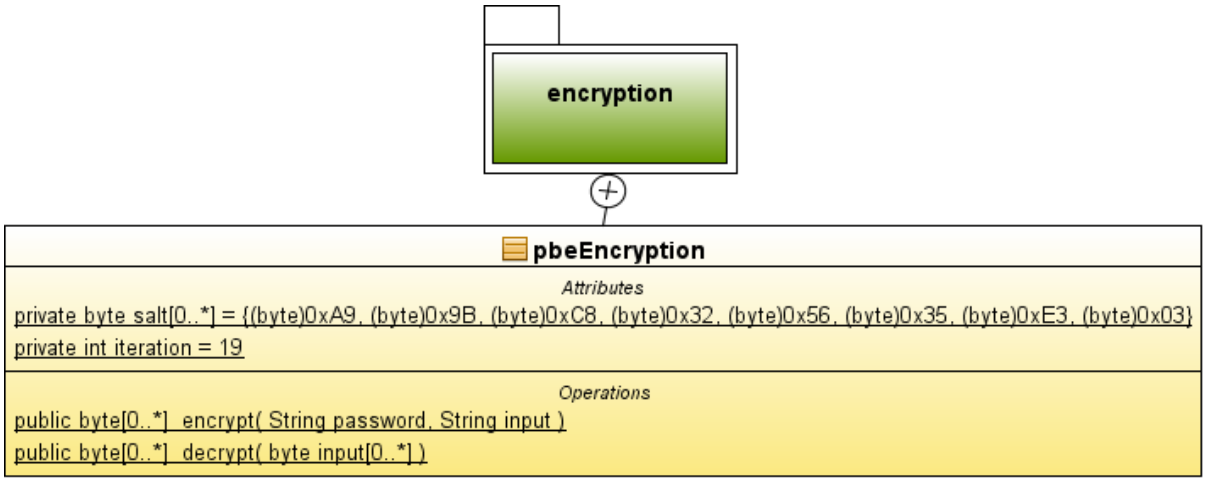
3.3.2 Design and Implementation Constraints

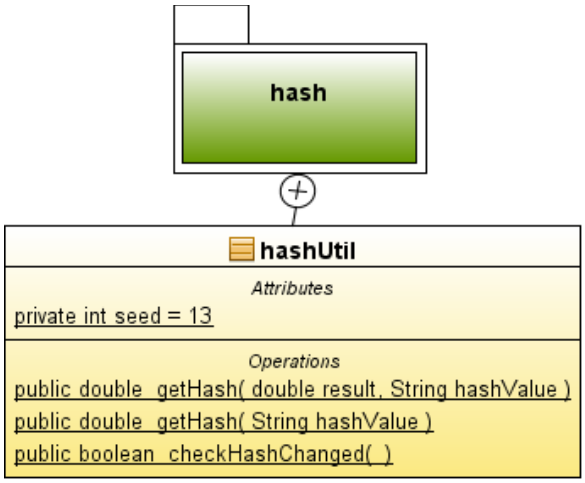
3.3.3 Class Diagram

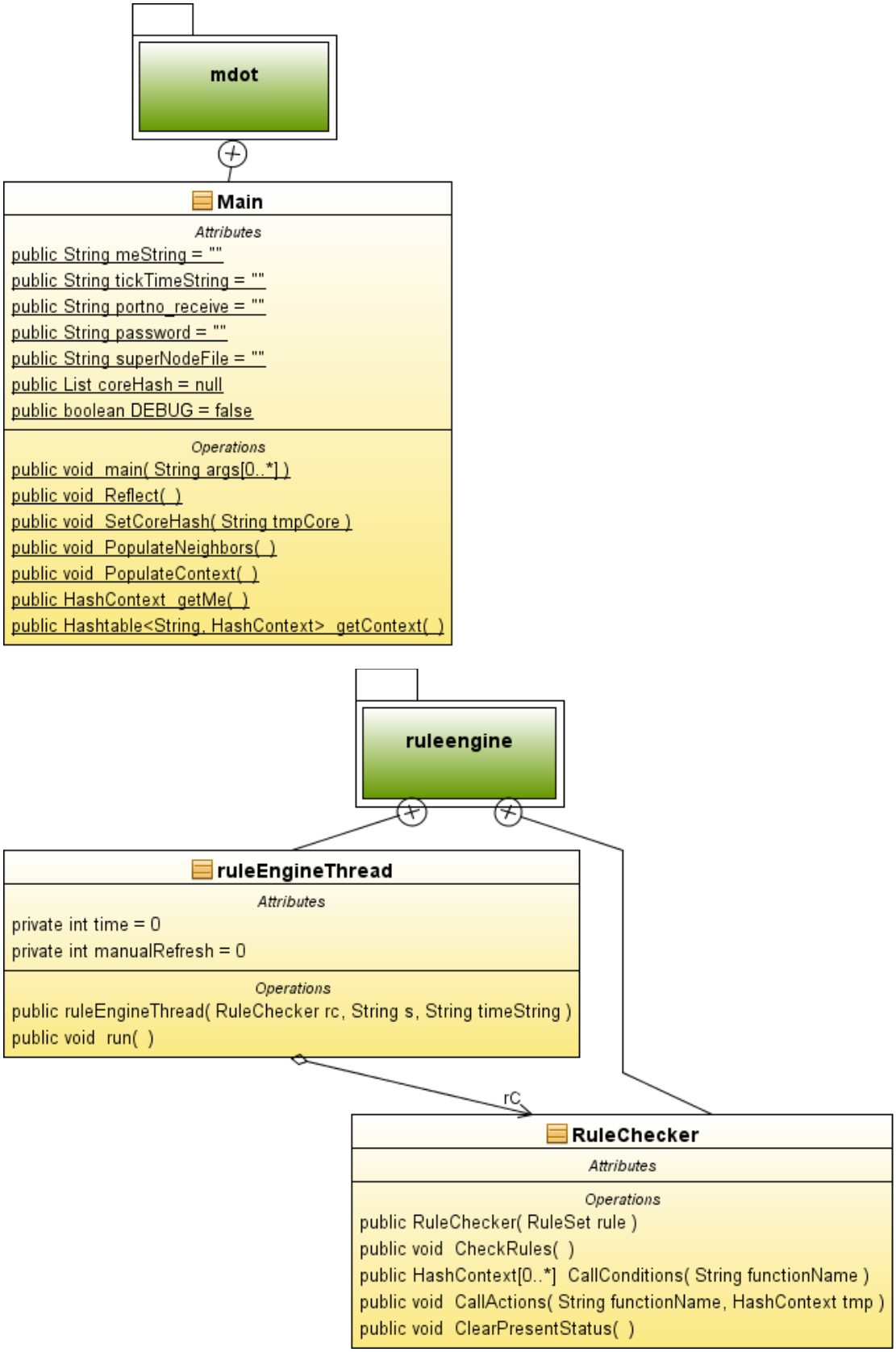


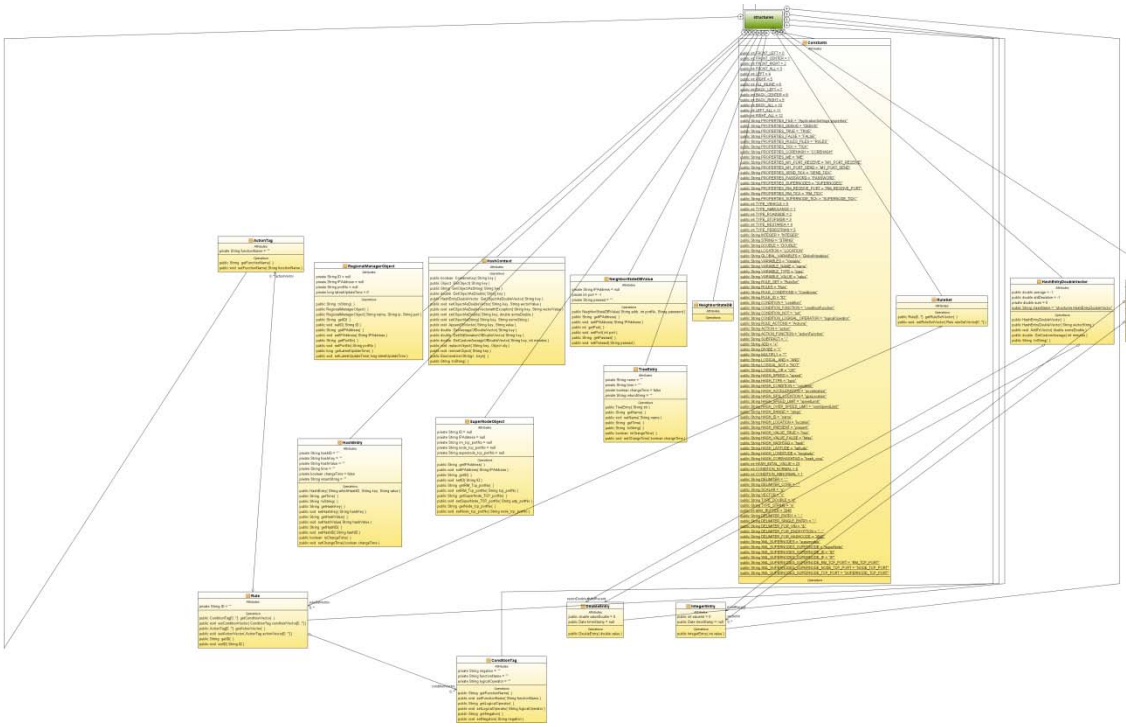

communication
Serialize
Attributes
Operations
public String serializeHashContext( )
communicationSendThread
Attributes
private int time = 0
private DatagramSocket socket
Operations
public communicationSendThread(DatagramSocket sock, int ticktime)
public void run( )
RM_CommunicationThread
Attributes
private String stringSend
private Socket tcpSocket
private int sleepTime = -1
Operations
public RM_CommunicationThread(int tickTime)
public void run( )
public void DeserializeTCPInputStream(InputStream input)
communicationEngine
Attributes
Operations
public communicationEngine( )
StaticSocket
Attributes
private DatagramSocket sendSocket = null
private DatagramSocket receiveSocket = null
Operations
public DatagramSocket getSendSocket( )
public DatagramSocket getReceiveSocket( )
public void setReceiveSocket(DatagramSocket aReceiveSocket)
public void setSendSocket(DatagramSocket aSendSocket)
communicationRecvThread
Attributes
private DatagramSocket socket
Operations
public communicationRecvThread(DatagramSocket sock)
public void run( )
DeSerialize
Attributes
Operations
public void deserializeHashContext(String someString, String IPAddress)
TCPCommunicationThread
Attributes
private Socket tcpSocket
private String stringSend
private String stringSendToSupernode
private int sleepTime = -1
private String IPADDRESS = null
private int portNO = -1
Operations
public TCPCommunicationThread(int tickTime)
public void run( )
public void DeserializeTCPInputStream(InputStream input)
public void DeserializeTCPInputStreamFromSuperNode(InputStream input)
StaticNeighborStateDB
Attributes
Operations

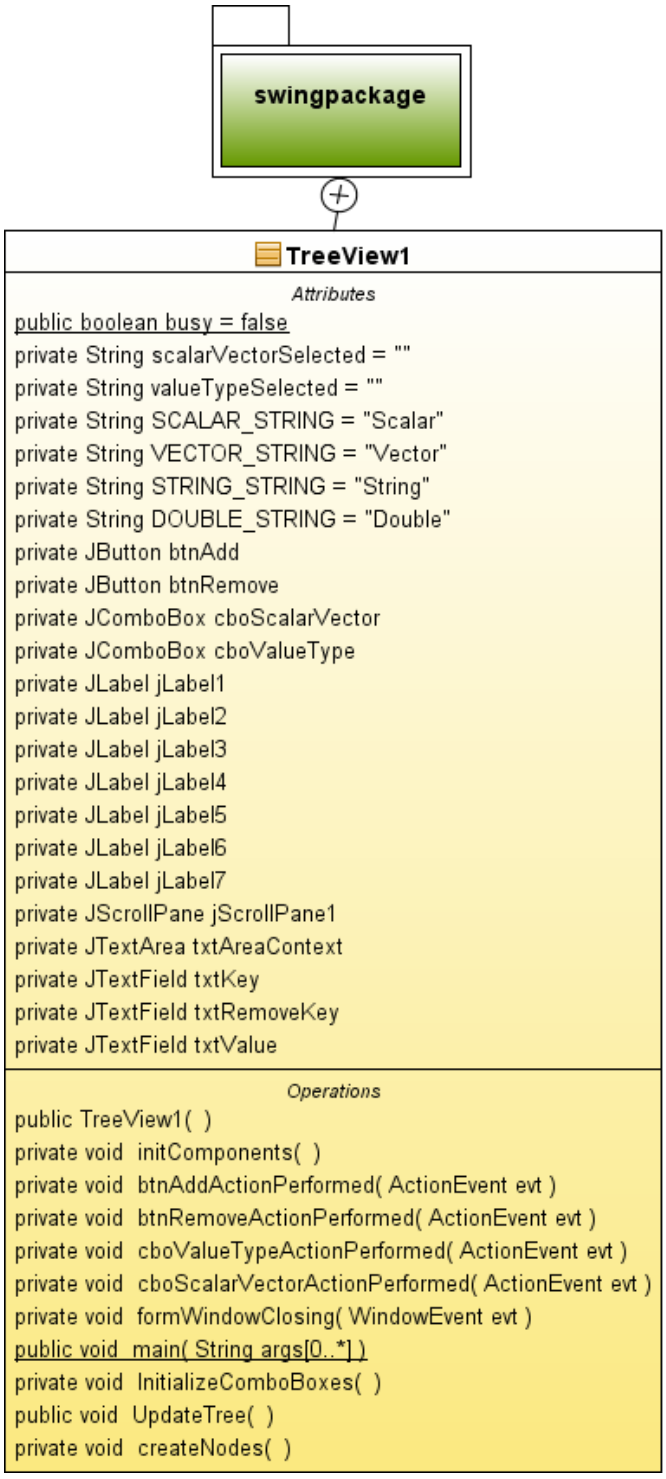

database
Context
Attributes
Operations
CoreDatabase
Attributes
Operations

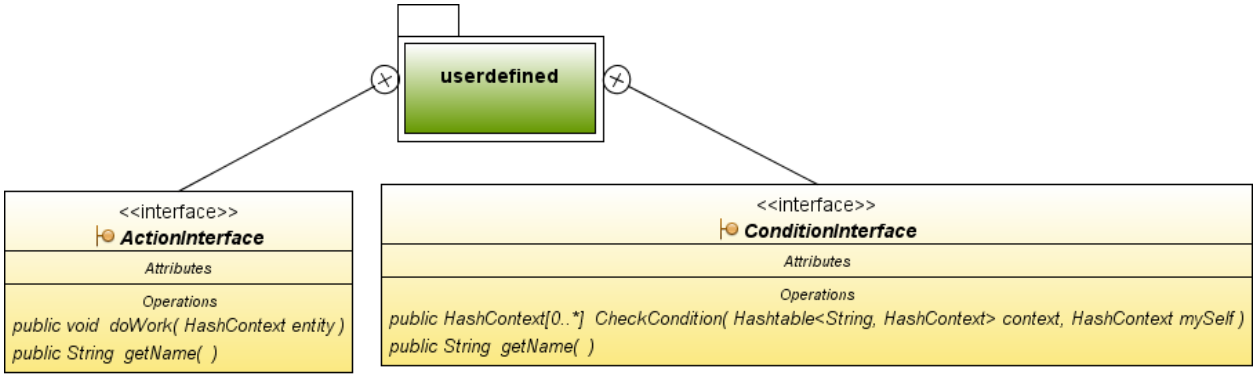

encryption
pbeEncryption
Attributes
private byte salt[0..*] = {(byte)0xA9, (byte)0x9B, (byte)0xC8, (byte)0x32, (byte)0x56, (byte)0x35, (byte)0xE3, (byte)0x03}
private int iteration = 19
Operations
public byte[0..*] encrypt(String password, String input)
public byte[0..*] decrypt(byte input[0..*])

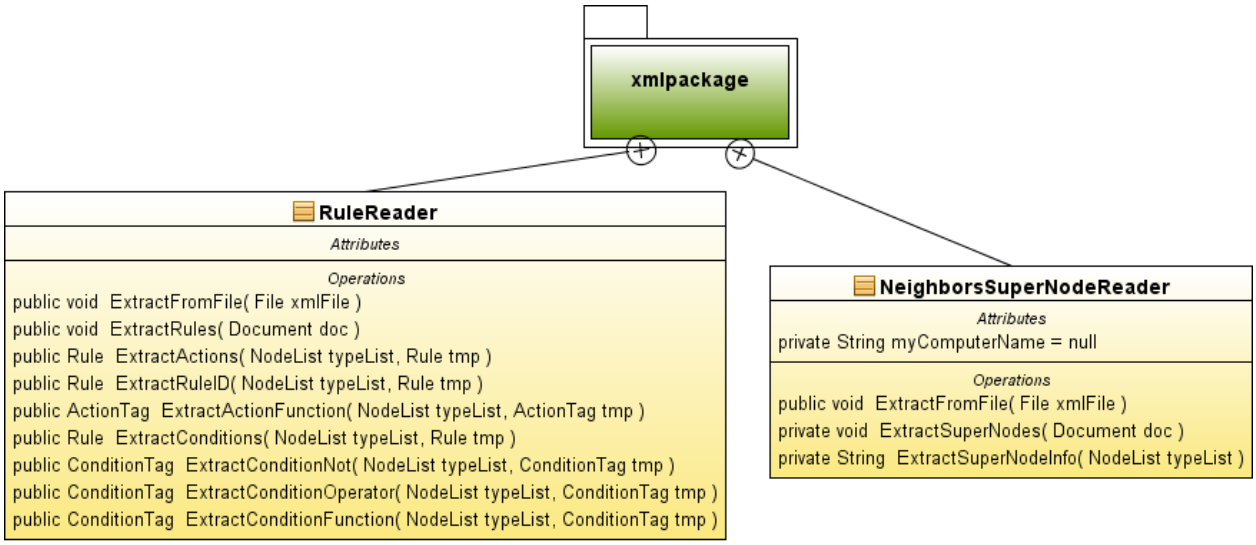

hash
hashUtil
Attributes
private int seed = 13
Operations
public double getHash(double result, String hashValue)
public double getHash(String hashValue)
public boolean checkHashChanged( )


mdot
Main
Attributes
public String meString = ""
public String tickTimeString = ""
public String portno_receive = ""
public String password = ""
public String superNodeFile = ""
public List coreHash = null
public boolean DEBUG = false
Operations
public void main(String args[0..*])
public void Reflect( )
public void SetCoreHash(String tmpCore)
public void PopulateNeighbors( )
public void PopulateContext( )
public HashContext getMe( )
public Hashtable<String, HashContext> getContext( )
ruleengine
ruleEngineThread
Attributes
private int time = 0
private int manualRefresh = 0
Operations
public ruleEngineThread(RuleChecker rc, String s, String timeString)
public void run( )
rC
RuleChecker
Attributes
Operations
public RuleChecker(RuleSet rule)
public void CheckRules( )
public HashContext[0..*] CallConditions(String functionName)
public void CallActions(String functionName, HashContext tmp)
public void ClearPresentStatus( )




swingpackage
TreeView1
Attributes
public boolean busy = false
private String scalarVectorSelected = ""
private String valueTypeSelected = ""
private String SCALAR_STRING = "Scalar"
private String VECTOR_STRING = "Vector"
private String STRING_STRING = "String"
private String DOUBLE_STRING = "Double"
private JButton btnAdd
private JButton btnRemove
private JComboBox cboScalarVector
private JComboBox cboValueType
private JLabel jLabel1
private JLabel jLabel2
private JLabel jLabel3
private JLabel jLabel4
private JLabel jLabel5
private JLabel jLabel6
private JLabel jLabel7
private JScrollPane jScrollPane1
private JTextArea txtAreaContext
private JTextField txtKey
private JTextField txtRemoveKey
private JTextField txtValue
Operations
public TreeView1( )
private void initComponents( )
private void btnAddActionPerformed(ActionEvent evt)
private void btnRemoveActionPerformed(ActionEvent evt)
private void cboValueTypeActionPerformed(ActionEvent evt)
private void cboScalarVectorActionPerformed(ActionEvent evt)
private void formWindowClosing(WindowEvent evt)
public void main(String args[0..*])
private void InitializeComboBoxes( )
public void UpdateTree( )
private void createNodes( )

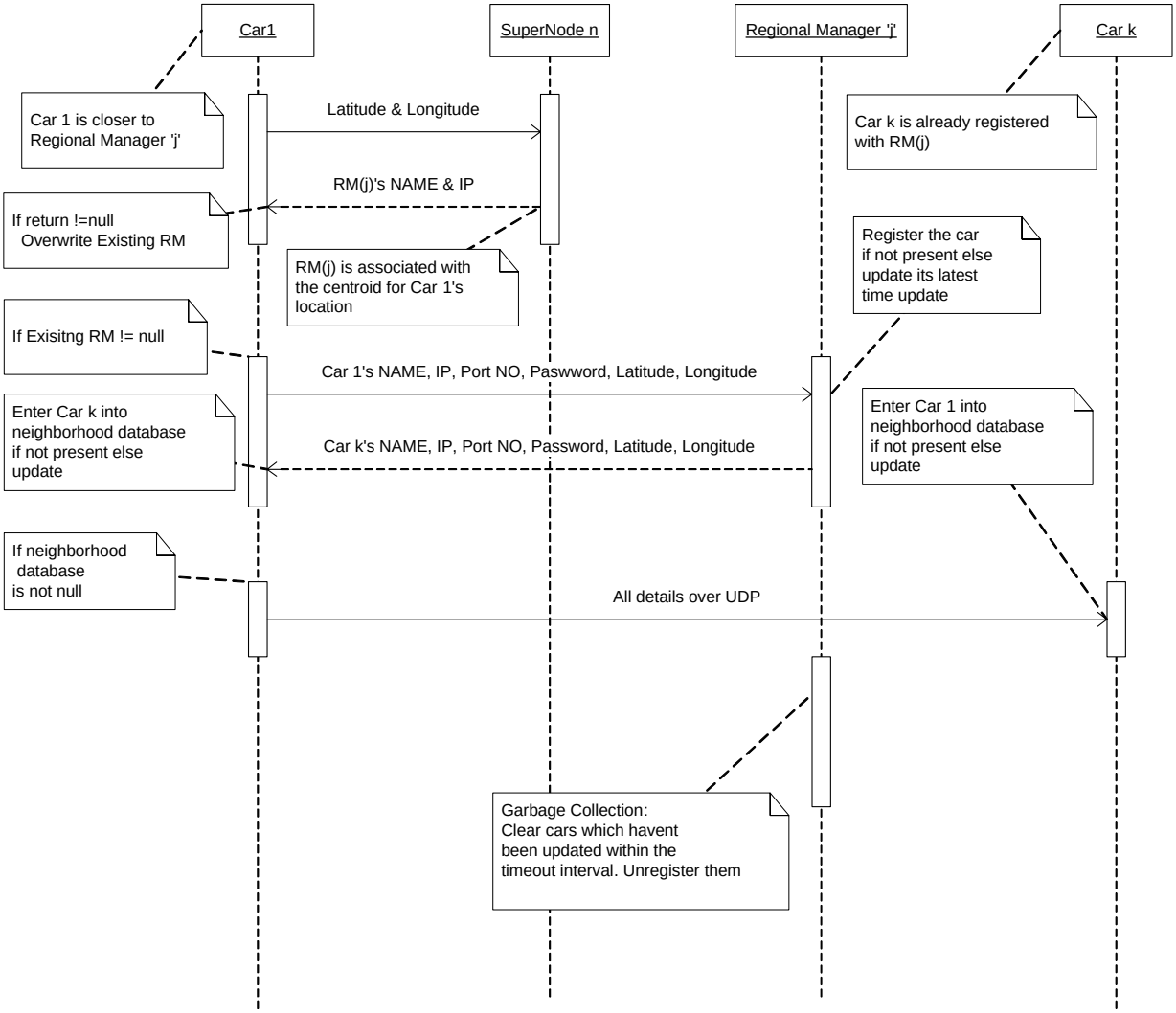




Description:

Component Name	Description

3.3.4 Sequence Diagram



Description:

3.3.5 Activity Diagram

TBD

3.3.6 External Interface Requirements

3.3.6.1 User Interfaces

TBD

3.3.6.2 Hardware Interfaces

TBD

3.3.6.3 Software Interfaces

TBD

3.3.6.4 Communications Interfaces

3.4 TMC

3.4.1 Design Description

Traffic Management Centers are visualized to be central monitoring stations. This integrates with different systems; to serve multiple purposes (refer to section 3.2.1 in AGORA Architecture and High Level Design Document).

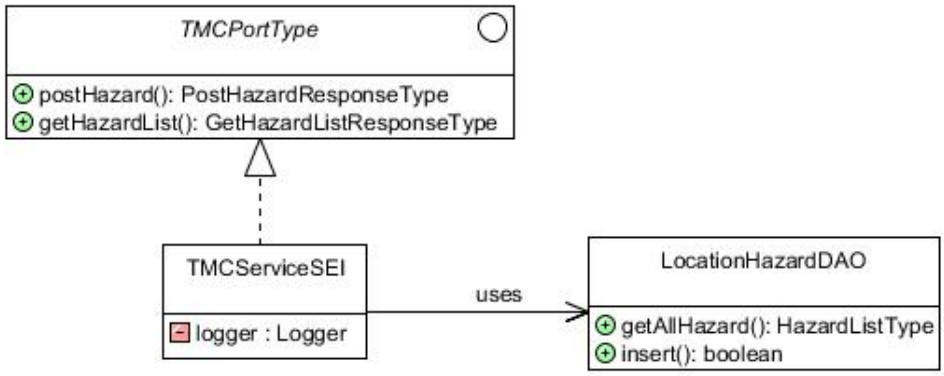
3.4.2 Design and Implementation Constraints

TMC integrates with multiples systems with resource constrained systems. One among them are

- VIS which integrates with every vehicle on the road
- Regional Managers
- Super Nodes

Because of the number of systems that TMC integrates, services exposed by TMC has strict SLA requirement, it should be highly scalable, and fault tolerant.

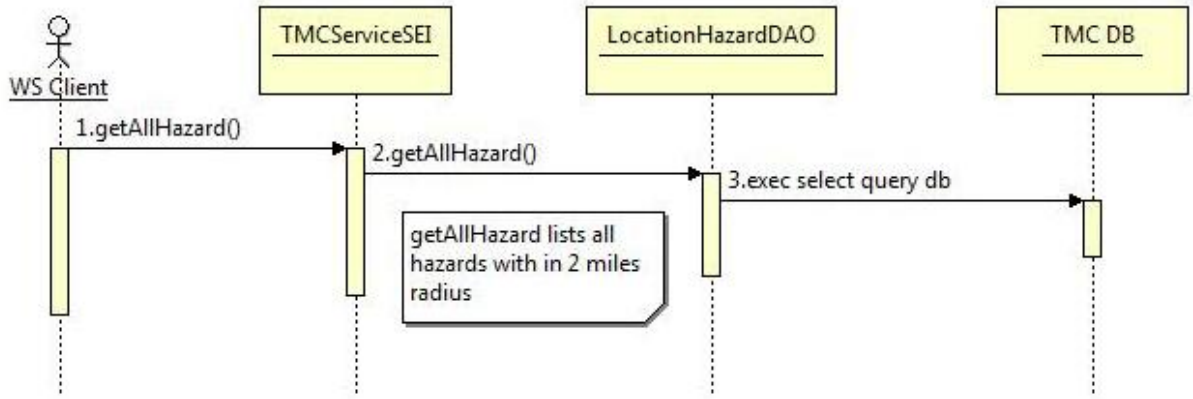
3.4.3 Class Diagram



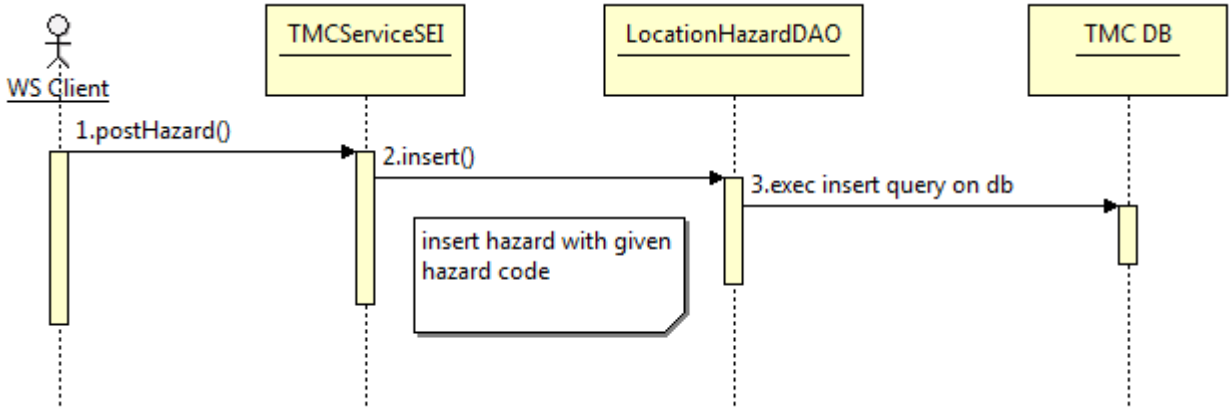
Description:

Component Name	Description
TMCPortType	Interface which is derived from wsdl contract
TMCServiceSEI	Concrete implementation of wsdl contract functions
LocationHazardDAO	Data access object which help in crud operation on hazards

3.4.4 Sequence Diagram



Sequence diagram to get all hazards

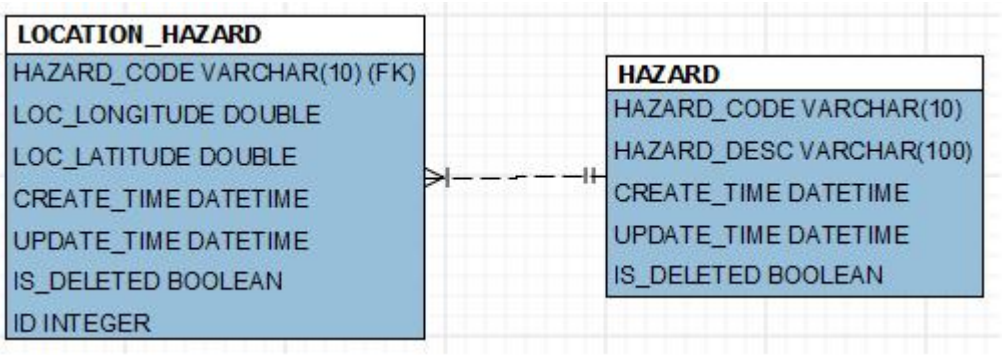


Sequence diagram to post hazard

Description:

Above sequence diagram shows how different components interact with each other to serve web service client requests.

3.4.5 ER Diagram



3.4.6 Activity Diagram

TBD

3.4.7 External Interface Requirements

3.4.7.1 User Interfaces

TBD

3.4.7.2 Hardware Interfaces

TBD

3.4.7.3 Software Interfaces

TBD

3.4.7.4 Communications Interfaces

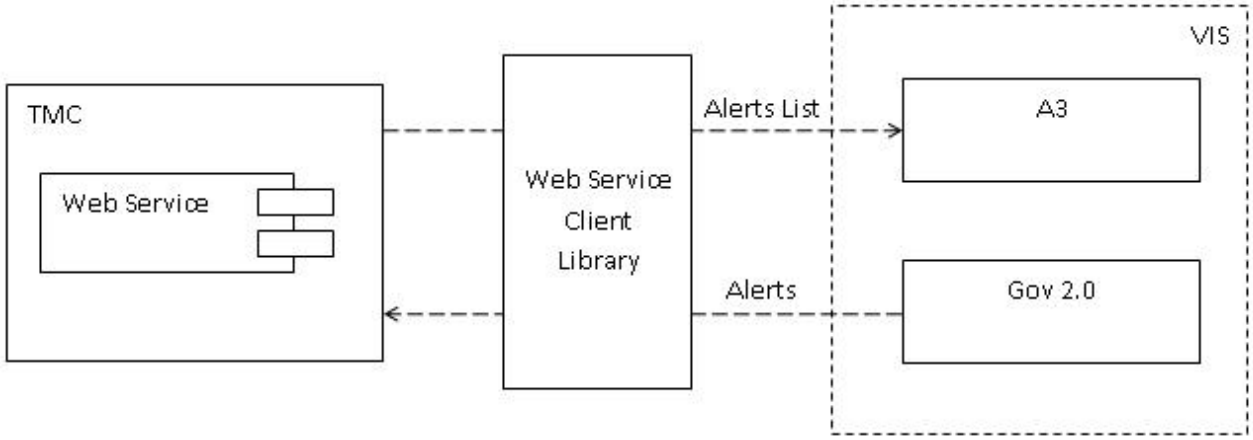
- WSDL
- SOAP
- XML

3.5 A3/Gov 2.0

3.5.1 Design Description

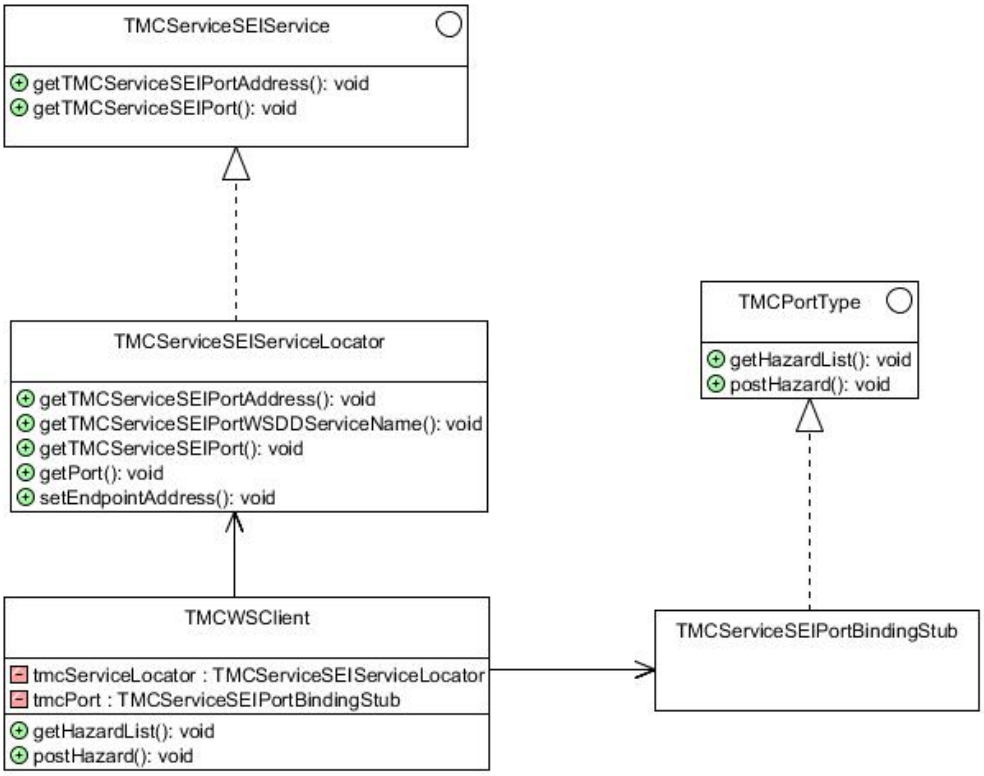
A3 is advisory alert application that integrates with VIS so it uses the entire infrastructure that has been developed for VIS. A3 also integrates with TMC through web services.

3.5.1 Design and Implementation Constraints



Description:

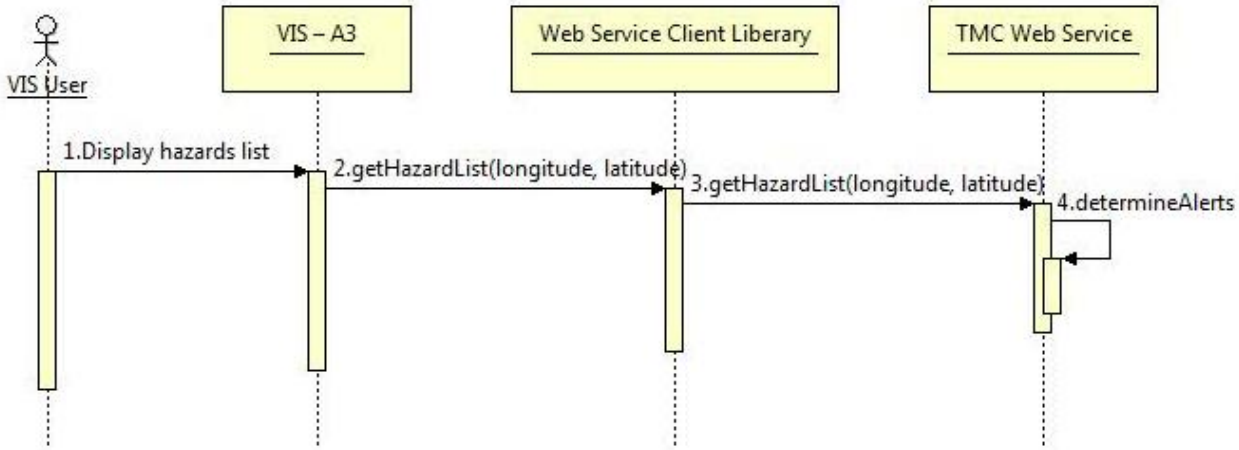
3.5.2 Class Diagram

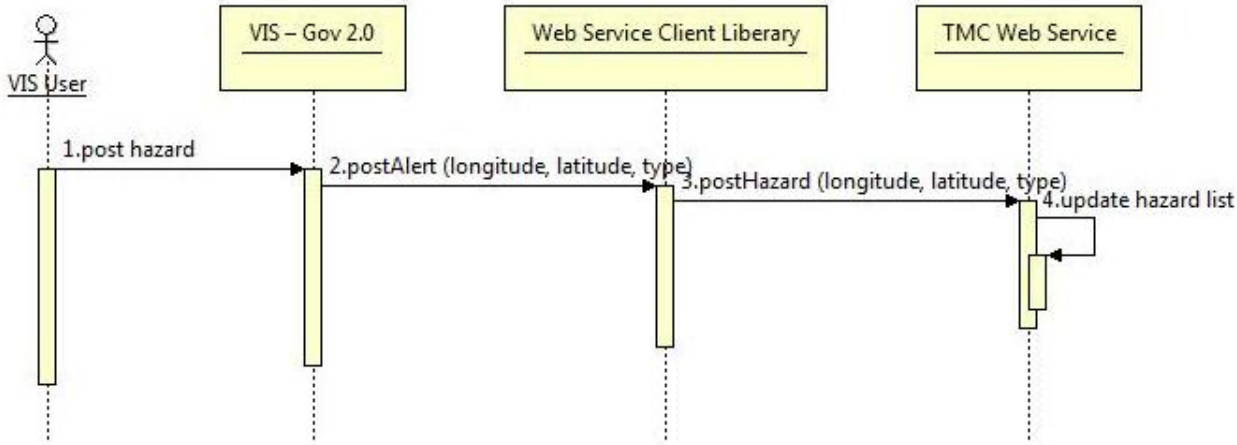


Description:

Component Name	Description
TMCServiceSEIService	TMC web service client service interface contract
TMCServiceSEIServiceLocator	Concrete implementation of TMCServiceSEIService
TMCPortType	Stub used to make web service call
TMCServiceSEIPortBindingStub	Binding class
TMCWSClient	Client which makes actual web service call

3.5.3 Sequence Diagram





Description:

NONE

3.5.4 Activity Diagram

TBD

3.5.5 External Interface Requirements

3.5.5.1 User Interfaces

- VIS – touch screen

3.5.5.2 Hardware Interfaces

None

3.5.5.3 Software Interfaces

- VIS

3.5.5.4 Communications Interfaces

- WSDL
- SOAP
- XML

Appendix A: Glossary

MDOT – Michigan Department of Transportation
OBE – On-Board Equipment
RSE – Road-Side Equipment
TMC – Traffic Management Centers
A3– Advisory Alert Application
LSA – Life Safety Application
VIS – Vehicle Integrated Software