* Original architecture is the best, doesn't require adaptations from smartGuard side;
* Check library licensing for javax.comm (RXTX, jSSC);
* Eclipse IDE with java, javascript, XML;
* The converter is considered as a central;
* One certificate should be enough for all the connections;
* GSM is a closed secure connection, possibility of simple encryption just in case;
* Phone numbers and message content are the weakest points in security (consider encryption).

[XML and Java](http://www.tutorialspoint.com/java_xml/java_dom_parser.htm)

[jSSC](https://code.google.com/p/java-simple-serial-connector/) (option to javax.comm) – Best option

[javax.comm](http://kishor15389.blogspot.de/2012/06/install-javaxcommor-rxtx-in-eclipse.html) (No longer supported) – Library to read/write serial ports. Unstable

[RXTX](http://rxtx.qbang.org/wiki/index.php/Main_Page) (option to javax.comm) Unstable

[SOAP](http://www.w3schools.com/webservices/ws_soap_intro.asp)

[Java read/write serial port](http://www.javaprogrammingforums.com/java-se-api-tutorials/5603-jssc-library-easy-work-serial-ports.html)

[java.awt](http://docs.oracle.com/javase/7/docs/api/java/awt/package-summary.html) (interface classes, might be needed)

**Software Architecture:**

Needs at least 3 different threads running simultaneously (maybe).

**Main** - Takes care of the interaction between the low level, high level and (maybe) user. Saves a log of messages every x minutes.

**LowLevel** - Receives and sends messages from/to the controllers. Message content example "ID Message".

**HighLevel** - Receives and sends messages from/to smartGuard, using the structures specified on the given XML schemes.

**Log** - Saving on RAM would decrease performance of the overall system and its not needed. Messages are saved in a file at specific events, easy fault check.

**Structures (temp):**

**Controller** - ID - Integer;

                    PhoneNumber - Integer;

                    Status - String;

                    Messages – Log.

                    Location(?).

**Message** - High - Based on XML scheme and source code;

                  Low - MessageID - Integer;

                            Send/Receive - Boolean/String;

                            Content - String;

**Log –** MessageQueue - List(String);

MessageCounter – Integer;

"I'm alive" message "ID PhoneNumber". Processed by Main, new controller is created and communications started.

Permanent (and intentional) disconnect message "ID Delete" Processed by Main, save last logs (including delete order), erase controller from the list.

Source code.

Test speed that messages can be sent/received.

Test GSM modem queue.

**Interface**

Text based. Receives commands (Start, Reset, Shutdown). Prints answers to provide information to the user.

After started prints a message when there is a connection to a new controller, when a controller receives a “delete” order. \*

Prints a list every hour with the status of all the connected controllers. \*

\*Adding to the saving all the messages sent and received by a controller.0

**Technical Specifications**

**Messages and Translation layer:**

Messages have similar structure (String) but don’t have the same content organization. So they need to be split in two structures.

Method to create empty messages. Public.

Method to transform messages from one type to another and destroy the original. Public.

Method destroy messages after they were used (sent or received and processed). Public.

Method to destroy a message. Private.

**Controllers:**

Organized on a list, identified by ID.

Controllers should be part of a subset of equipments (prepared for future integrations).

Receives/Sends messages.

Method to create a controller (normal status by default).

Method to destroy a controller.

Status change and general manipulation will be done within the Main structure or the equipment structure.

**Log:**

Will be kept for each controller. Name “ControllerID”.

Text will be “Date/time – Sent/Received – Message”.

Text file kept on the server to provide easy fault check.

[XMLParser](http://examples.javacodegeeks.com/core-java/xml/java-xml-parser-tutorial/)

[SOAP Java](http://docs.oracle.com/cd/A97630_01/appdev.920/a96616/arxml11.htm)

<https://docs.oracle.com/javase/tutorial/jaxp/intro/extensible.html>

<http://www.vogella.com/tutorials/JavaXML/article.html>

<https://en.wikipedia.org/wiki/Java_API_for_XML_Messaging>

<http://www.soapuser.com/basics1.html>

<http://www.java2blog.com/2013/03/soap-web-service-tutorial.html>

<http://www.java2blog.com/2013/03/jaxws-web-service-eclipse-tutorial.html>

<http://stackoverflow.com/questions/15940234/how-to-do-a-soap-web-service-call-from-java-class>