MGMT-CORE Progress Report

What's there and what is not?

1. **Versioning**
   a. **v1.0** (26/11/12) by **Baris Demiray** <baris.demiray@eurecom.fr>
      Initial version
   b. **v1.1** (27/11/12) by **Baris Demiray** <baris.demiray@eurecom.fr>
      3.b. Version Information and 3.c. Physical View of the Development Environment sections added
      4.a. Unit Tests and 4.b. Simulator sections added
   c. **v1.2** (27/11/12) by **Baris Demiray** <baris.demiray@eurecom.fr>
      10.a. Intelligent code to select a proper communication profile section has been updated according to code refactorization

2. **Goal**
   Goal of making this document is to keep track of what has been done and what is missing in the current SCOREF-MGMT (or MGMT-CORE) implementation. Every other document produced during the making of SCOREF-MGMT will be referred herein.

3. **Design & Development**
   a. **General Information**
      C++ has been used as the programming language along with STL and Boost libraries. Eclipse is the IDE and project files, coding style document (ready to be imported into Eclipse), path and library information (ready to be imported into Eclipse) are all ready in the Software/ directory of EURECOM SVN. Valgrind and Cppcheck have been utilized during the development and there are no major memory leaks or coding errors that compilers would not catch.

      Code documentation can be extracted using Doxygen (a .doxygen file is ready in the root directory of the project) yet further design documentation (especially class diagrams and sequence diagrams) will be produced.

   b. **Version Information**
      Current version information of the libraries and the software I have been using is as follows.

Software/Library	Version	Notes
Ubuntu	Precise 12.04.1 LTS	
Boost	1.48.0	Standard Ubuntu package
g++	4.6.3	Standard Ubuntu package
libstdc++	4.6	Standard Ubuntu package
Eclipse	Indigo Service Release 2	http://www.eclipse.org/

Doxygen	1.7.6.1	Standard Ubuntu package
Python	2.7.3	Standard Ubuntu package

c. Physical View of the Development Environment

Herein this section you will find how the software resides in the development environment and what is where.

i. EURECOM SVN Server

Following paths given in the table are relative to the root directory of the project, openair4G/trunk/openair3/SCOREF-MGMT/

Directory	Component	Notes
Binary/	Releases	Those versions released in the INRIA SVN are kept here
Documentation/	Presentations and manuals	All the presentations are kept here along with .odt versions
Software/	Root directory of the project	This is also the root directory to do "Makefile project with existing code" in Eclipse
Software/bin/	Makefile produced binaries	Makefile puts binaries here
Software/bin/configuration/	FACilities configuration files	
Software/doc/	Documentation	Doxygen will produce the source code documentation here
Software/doc/design/	Design	Currently empty, will have class and sequence diagrams
Software/src/	Root directory of the code	All the code is in here
Software/src/interface/	Interfaces	C++ interface classes
Software/src/packets/	GN and FAC packets	Classes where packets are handled
Software/src/util/	Utility methods	
Software/test/	Root directory of the test code	All the unit tests, simulator, etc. are here
Software/test/data/	Test data for unit tests	Do not edit this directory or unit tests will fail
Software/test/data/confFiles/	Test data for unit tests	
Software/test/simulation/	Root directory of the simulator	All the Python code resides here, see *4.b. Simulator*
Software/test/simulation/scenarios/	Simulator scenarios	
Software/test/unittest	Unit test code	See *4.a. Unit Tests*

ii. INRIA SVN Server

Following paths given in the table are relative to the root directory of the project, scoref/Software/SoftwareComponents/MGMT/CORE/

Directory	Component	Notes
Binary/	Releases	Versions released so far
Documentation/	Presentations and manuals	All the presentations are kept here only as .pdf

4. Testing

a. Unit Tests

Unit test code resides in Software/test/unittest/ and Google C++ Testing Framework has been used to develop this code. Please see the README file in this directory to see how the code is compiled and run.

b. Simulator

Simulator is a Python code that can process commands given in a scenario file. Please see the README file in test/simulation/ directory for the details about the commands. Currently following packets can be sent,

COMMUNICATION_PROFILE_REQUEST
NETWORK_STATE
CONFIGURATION_NOTIFICATION
GET_CONFIGURATION
LOCATION_UPDATE

5. Configuration Management

Configuration handling of SCOREF-MGMT is ready and working. Main configuration file (by default) named MGMT.conf is read upon start and all the parameters parsed are printed. GN configuration parameters are all integer, FAC-MGMT configuration parameters can have three data types: string, integer, and float.

There is also a directory called named configuration/ and there is FACilities configuration files inside. SCOREF-MGMT parses every configuration file it finds there and prints. See *SCOREF-MGMT_Configuration.pdf* for further details.

6. Socket Interface

Socket code is quite clear now. Boost's Asynchronous UDP socket has been utilized and everything about it is in ManagementServer class. All the packet exchange and repetitive sending (as in Wireless State Request packet) is handled in this class.

7. Threading

There are two threads, one main thread and one InquiryThread (as in MGMT-CORE's terminology) object. Main thread handles all the communication with all the clients and InquiryThread object handles transmission of repetitive packets, replies go to the main thread (whom spends almost all of its time in ManagementServer class) and handled there.

8. GN Interface

All the information sent and received by MGMT-CORE is written in hexadecimal notation.

a. LOCATION_UPDATE

This packet is implemented and is ready but it's not used since HITACHI provided a more practical solution by-passing MGMT-CORE so now it is as follows,

POTI -> Laurens (HITACHI) component -> GN

instead of

POTI -> FAC-MGMT -> CORE-MGMT -> GN

But if it's received from FAC-MGMT then it is going to be parsed and MIB will be updated.

b. LOCATION_TABLE_REQUEST

This packet is implemented and is in use. A LOCATION_TABLE_REQUEST packet is sent upon GN's connection to MGMT-CORE with the address 0xFFFFFFFFFFFFFFFF (in order to request all the location table entries).

c. LOCATION_TABLE_RESPONSE

This packet is implemented and is in use. Upon transmission of a LOCATION_TABLE_REQUEST to GN, GN replies back with a LOCATION_TABLE_RESPONSE and MGMT-CORE parses and updates MIB with extracted information. Currently only the EGO vehicle is returned by GN so Location Table has 1 entry after processing this message. All the information during and after the processing of this message and the location table can be found in the log records.

d. CONFIGURATION_UPDATE_AVAILABLE

This packet is implemented and is in use. Upon receipt of a CONFIGURATION_NOTIFICATION from FAC-MGMT, MGMT-CORE will notify GN with this packet. Even though FAC-MGMT is not ready yet a simulator has been used to test MGMT-CORE and following 5-step scenario seems working,

Event number	Event direction	Event Packet
1	FAC-MGMT -> MGMT-CORE	CONFIGURATION_NOTIFICATION
2	MGMT-CORE	Updating MIB

3	MGMT-CORE -> GN	CONFIGURATION_UPDATE_AVAILABLE
4	GN -> MGMT-CORE	CONFIGURATION_REQUEST
5	MGMT-CORE -> GN	CONFIGURATION_RESPONSE

Key count field (see CM-GN to MGMT Interface Description file) is always filled with the total number of ITS keys.

e. CONFIGURATION_REQUEST

This packet is implemented and is in use. Upon receipt of this packet MGMT-CORE prepares a CONFIGURATION_RESPONSE_BULK message (it's the only type that is asked by GN) and sends it to GN. This is the first packet MGMT-CORE receives from GN (and probably from other clients too).

f. CONFIGURATION_RESPONSE_CONTINUOUS

This packet is implemented but has never tested. GN doesn't ask for this but rather request a single message containing all the configuration items (CONFIGURATION_RESPONSE_BULK, see next title) so this is of no use now.

g. CONFIGURATION_RESPONSE_BULK

This packet is implemented and is in use. Upon receipt of a CONFIGURATION_REQUEST, MGMT-CORE prepares this packet and sends it to the sender. This packet carries GN, FAC, or all the configuration items according to the *Conf ID* parameter (see CM-GN to MGMT Interface Description file) of the CONFIGURATION_REQUEST packet.

h. COMMUNICATION_PROFILE_REQUEST

This packet is implemented and is in use, and interpretation of filters has been discussed and agreed upon with Andrea (HITACHI).

Communication profiles are placed towards the end of the configuration file MGMT.conf and MGMT-CORE parses them upon start. Upon receipt of this packet MGMT-CORE will prepare a COMMUNICATION_PROFILE_RESPONSE having the communication profiles matching the filter given in this packet (namely a copy of the communication profile table or a subset of it) and send it to the sender.

i. COMMUNICATION_PROFILE_RESPONSE

This packet is implemented and is in use. Upon receipt of a COMMUNICATION_PROFILE_REQ, MGMT-CORE will prepare this packet carrying all those communication profiles matching the filter given in the corresponding COMMUNICATION_PROFILE_REQUEST and send it to the sender.

j. WIRELESS_STATE_REQUEST

This packet is implemented and is in use. Yet there is no answer from GN thus far and furthermore in the logs of GN this is marked as an "invalid event type/subtype". So this packet should be regarded as *not tested*.

Upon start MGMT-CORE extracts the interval from the configuration file of how frequent it should send a WIRELES_STATE_REQUEST to GN and starts to send this packet upon connection of a GN and in every CONF_WIRELESS_STATE_UPDATE_INTERVAL (see MGMT.conf) seconds.

k. WIRELESS_STATE_RESPONSE

This packet is implemented but since WIRELESS_STATE_REQUEST is not implemented on GN side (see 5.j. WIRELESS_STATE_REQUEST) MGMT-CORE has never received this packet so this packet should be regarded as *not tested*.

l. NETWORK_STATE

This packet is supposed to be received on a regular basis from GN. But this feature will be available in the next release so implementation of this packet should be regarded as *not tested*.

9. FAC-MGMT Interface

All the information sent and received by MGMT-CORE is written in hexadecimal notation.

a. CONFIGURATION_UPDATE_AVAILABLE

This packet is implemented but is not yet in use since FAC-MGMT is not ready. It has been tested using the simulator and it seems okay (see 5.d. CONFIGURATION_UPDATE_AVAILABLE).

b. CONFIGURATION_REQUEST

This packet is implemented but is not yet in use since FAC-MGMT is not ready. This packet has been tested with GN, though.

c. CONFIGURATION_RESPONSE_CONTINUOUS

This packet is implemented but is not (and probably will not be) in use. FAC-MGMT is not ready yet but probably CONFIGURATION_RESPONSE_BULK packet will be utilized instead of this packet.

d. CONFIGURATION_RESPONSE_BULK

This packet is implemented but is not yet in use since FAC-MGMT is not ready. This packet has been tested with GN, though.

e. CONFIGURATION_NOTIFICATION

This packet is implemented but is not yet in use since FAC-MGMT is not ready. It has been tested using the simulator and worked for all ITS keys and data types (string, and integer) tested thus far.

f. COMMUNICATION_PROFILE_REQUEST

This packet is implemented but is not yet in use since FAC-MGMT is not ready. This packet has been tested with GN, though.

g. COMMUNICATION_PROFILE_RESPONSE

This packet is implemented but is not yet in use since FAC-MGMT is not ready. This packet has been tested with GN, though.

h. **COMMUNICATION_PROFILE_SELECTION_REQUEST**
This packet is implemented but has not yet been tested.

i. **COMMUNICATION_PROFILE_SELECTION_RESPONSE**
This packet is implemented but has not yet been tested. Besides, functionality of choosing a proper communication profile is missing.

10. Tasks

a. **Intelligent code to select a proper communication profile**
There is a stub (or empty) method named,

```
CommunicationProfileManager::selectProfile()
```

and the intelligent code should go into this method. It takes three parameters; namely latency, relevance, and reliability and returns the Communication Profile ID of type CommunicationProfileID. See mgmt_comm_prof_manager.{cpp|hpp} files for further details.

b. **Design documents has to be produced**
Especially class and sequence diagrams are important (although class diagrams can be extracted using Doxygen right now).

11. References

All the following files are placed in SCORE@F SVN.

a. Software/SoftwareComponents/MGMT/CORE/Documentation/**SCOREF-MGMT_Configuration.pdf**
b. Software/SoftwareComponents/MGMT/CORE/Documentation/**CM-GN Interface Description.pdf**
c. Software/SoftwareComponents/MGMT/CORE/Documentation/**FAC-CM Interface Description.pdf**