
# MASTER THESIS

TITLE: LAPI project
SUBTITLE: Application layer

MASTER DEGREE: Master in Science in Telecommunication Engineering & Management

AUTHOR: Josep Francesc Soler Vich

DIRECTOR: Lukáš Kencl

DATE: February 6 th 2009


ERICSSON
vodafone
RESEARCH AND DEVELOPMENT CENTRE

Títol: LAPI Project
Subtítol: Application layer

Autor: Josep Francesc Soler Vich

Director: Lukáš Kencl

Data: 6 de Febrer de 2009

**Resum**

Aquest projecte du a terme una part d'una nova aplicació de localització. Aquest servei consisteix en una aplicació basada en localització que es connecta a un servidor de localització. Aquest servidor proveeix la informació de localització necessària per córrer el servei. Llavors, es necessita d'una interfície entre l'aplicació basada en localització i el servidor de localització. Aquesta interfície pot ser una Interfície de Programació d'Aplicacions de Localització (LAPI). El desenvolupament d'aquesta LAPI és l'objectiu d'aquest projecte.

Title: LAPI Project
Subtitle: Application layer

Author: Josep Francesc Soler Vich

Director: Lukáš Kencl

Date: Febrer 6th 2009

**Overview**

This project carries out a part of a new location application. This service consists of location-based application that connects with a location server. That server provides the necessary location information to run the service. Then an interface between the location-based application and the location server is needed. That interface could be a Location Application Programming Interface (LAPI), and the development of one LAPI is the goal of this project.
.

I would like to thank my supervisor Petr Vláčil for his effort during the project, he always intended to help us. Thanks to my colleague and friend Luis for his effort. I would like to thank Dr.Lukáš Kencl and Ing.Robert Bešťák for their interest in we have the best possible stay. Thanks to all RDC for welcoming me so nice.

# TABLE OF CONTENTS

# ACRONYMS

| | |
|---|---|
| LAPI | Location Application Programming Interface |
| MLP | Mobile Location Protocol |
| OMA | Open Mobile Alliance |
| XML | Extensible Markup Language |
| HTTP | HyperText Transfer Protocol |
| SLI | Standard Location Immediate |
| LCS | Location Service |
| CGI | Cell Global Identity |
| LAC | Location Area Code |
| CI | Cell Identity |
| DTD | Document Type Definition |

# CHAPTER 1. INTRODUCTION

## 1.1. Purpose

The use of the mobile phone enhance every day. That is promoted by the mobile communications companies who release new mobile phone services. This project carries out a part of a new location application.

This service consists of location-based application that connects with a location server. That server provides the necessary location information to run the service. Then an interface between the location-based application and the location server is needed. That interface could be a Location Application Programming Interface (LAPI), and the development of one LAPI is the goal of this project.

## 1.2. Goals

The main goals of this project are:

- To develop a set of tools (API) to get the necessary information to start the location-service. Those tools have to be based on a location protocol.
- To develop a module who receives queries from internet and then response with the location-information.
- That module has to join the SS7box structure.

## 1.3. Structure of the thesis

A progressive structure has been followed to write and describe the project. The thesis starts explaining the technologies used and ends explaining the results obtained with the implementation of those technologies.

.

# CHAPTER 2. TECHNOLOGIES USED

## 2.1. Mobile Location Protocol

The MLP (1) was developed by Location Working Group of the Open Mobile Alliance (OMA). The MLP is an application-level protocol defined by an XML code and designed for querying the position of mobile stations independent of underlying network technology. There are different kinds of location services defined in three layers. Those layers are showed on Figure 1.

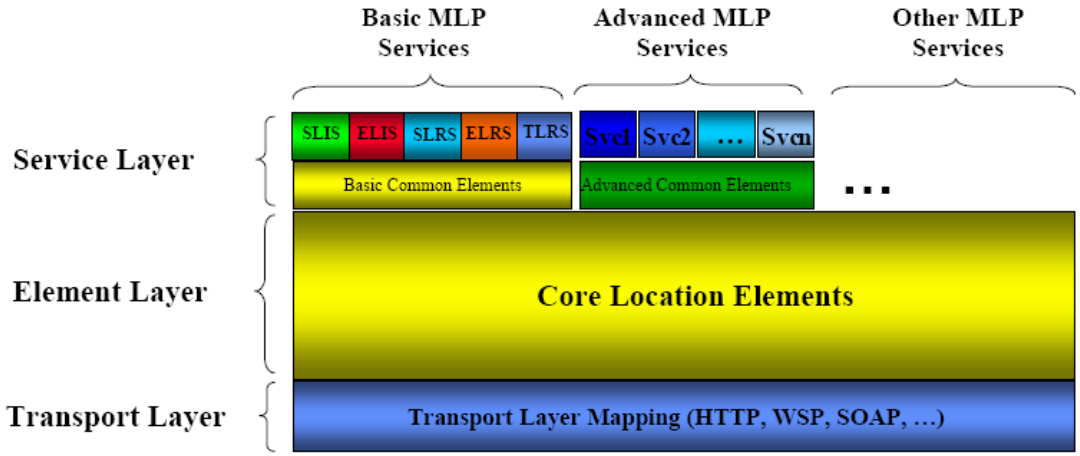



**Figure 1: MLP Structure**
**Source: OMA Mobile Location Protocol V 3.2 (1)**

### 2.1.1. The MLP structure

The MLP is defined by three layers. The lowest layer is called *Transport layer*. That layer defines how the XML code is transported. In this project the HTTP has been used but there are other possibilities like WSP, SOAP and others.

The other two layers define the MLP code and therefore the services supported. The *Element Layer* defines the common elements in each service. Those elements are defined on the documents type definition:

- MLP_ID.DTD defines the Identify Element Definitions.
- MLP_FUNC.DTD defines the Function Element Definitions.
- MLP_LOC.DTD defines the Location Element Definitions.
- MLP_RES.DTD defines the Result Element Definitions.
- MLP_SHAPE.DTD defines the Shape Element Definitions.
- MLP_QOP.DTD defines the Quality of Position Element Definitions.
- MLP_GSM_NET.DTD defines the GSM Network Parameters Element Definitions.

- MLP_CTXT.DTD defines the Context Element Definitions.

The *Service Layer* defines the services and it is the highest layer. This layer can be divided into two parts: the specific part of each service and the common part of the service. In the common part, there is defined the Header components which consists of the information that identifies the client.

### 2.1.2. The services

There are different types of location services. Each location server select the services that it needs to support. The services can be defined as:

- *Basic MLP services* are based on location services defined in 3GPP. These basic services are defined briefly on Table 1.
- *Advanced MLP services* or *Other MLP services* as well as it is defind in (1), they are additional services that will be specified in other specifications or forums.

| Service | Description |
|---|---|
| Standard Location Immediate Service | This service is used when a location response is needed immediately or the request may be answered by several asynchronous location responses. A timeout can be defined.<br>This service consists of the following messages:<br>• Standard Location Immediate Request.<br>• Standard Location Immediate Answer.<br>• Standard Location Immediate Report. |
| Emergency Location Immediate Service | This service is used for querying of the location of a mobile subscriber that has initiated an emergency call. The response is needed immediately, in a set time. The response can be served by several asynchronous location responses.<br>This service consists of the following messages:<br>• Emergency Location Immediate Request.<br>• Emergency Location Immediate Answer.<br>• Emergency Location Immediate Report. |
| Standard Location Reporting Service | This service is used when a client wants an LCS to receive the location of a MS location.<br>This service consists of the following messages:<br>• Standard Location Report. |

| | |
|---|---|
| | • Standard Location Report Answer. |
| Emergency Location Reporting Service | This service is used when the wireless network automatically initiates the positioning at an emergency call.<br>This service consists of the following messages:<br>• Emergency Location Report. |
| Triggered Location Reporting Service | This is a service used when the location of subscriber's mobile is served at a specific time interval or on the ocurrence of a specific event.<br>This service consists of the following messages:<br>• Triggered Location Reporting Request.<br>• Triggered Location Reporting Answer.<br>• Triggered Location Report.<br>• Triggered Location Reporting Stop Request.<br>• Triggered Location Reporting Stop Answer. |

**Table 1: MLP Services**

### 2.1.3. Standard Location Immediate Service

The Standard Location Immediate (SLI) service is the service implemented in this project. There is the only necessity to receive queries of location information of a MSISDN and after answer that location with the coordinates of the MSISDN. Therefore, the SLI fits with the necessity.

This is a standard service for requesting the location of one or more Mobile Subscribers. This service is used when a location response is required within a set time.

When more than one mobile subscriber position is requested, the network takes too much time to serve the answer. Then if the location server supports it, the answer can be sent at the same time or individually using one or more connections.

The message flow is made up by one standard location immediate request, one standard location immediate answer and if it is needed one or more standard location immediate report. Each report brings one or more mobile subscriber location. This message flow schema is showed in Figure 2.

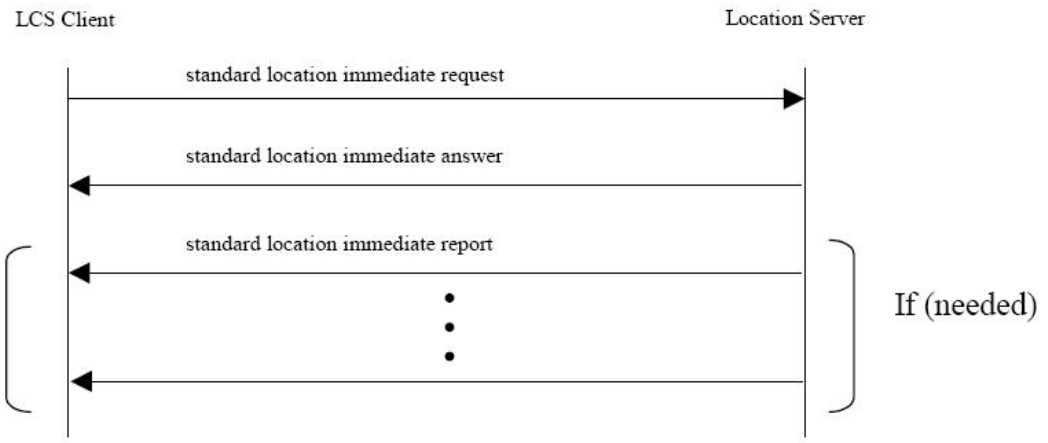



**Figure 2: SLI message flow.**
**Source: OMA Mobile Location Protocol V 3.2 (1)**

## 2.2. MySQL

The database used in this project has been MySQL. This is a relational database management system (RDBMS). It is developed by MySQL AB and it is available under terms of the GNU General Public License. This database uses the Structured Query Language that is a programming language for querying and modifying data and managing databases.

The MySQL used is the stable version 5.1.30. (2). The MySQL is provided with a lot of functionalities. The functionalities used in this project are showed in Table 2.

| Command | Description |
|---|---|
| `SELECT` | Used to get the information from the database. |
| `FROM` | to define the table where the information has to be searched. |
| `WHERE` | Used to specify which condition has to satisfy the information. |
| `select cast 0xff as unsigned` | Used to convert the hexadecimal number, in this case, ff to an integer. |

**Table 2: MySQL commands used.**

## 2.3. Programming language: C++

The main programming language used in this project has been C++. This is a powerful language wich introduce object-oriented features to C.

# CHAPTER 3. LAPI DESCRIPTION

## 3.1. Environment

The environment of this project is made up by two main parts. Those parts are showed on Figure 3 and are explained:

- *Location Server*: the server contains or can provide the location information. The location information is given by, in this case, the Vodafone network. The SS7Box, a server located in the server-room on RDC, works as the Location Server and the location information is provided by SS7Box's module called LBS module.
- *Location-based application*: that is a client who sends requests to the Location Server to get location information. That client is running a location service and needs the location information to respond the location service requests.

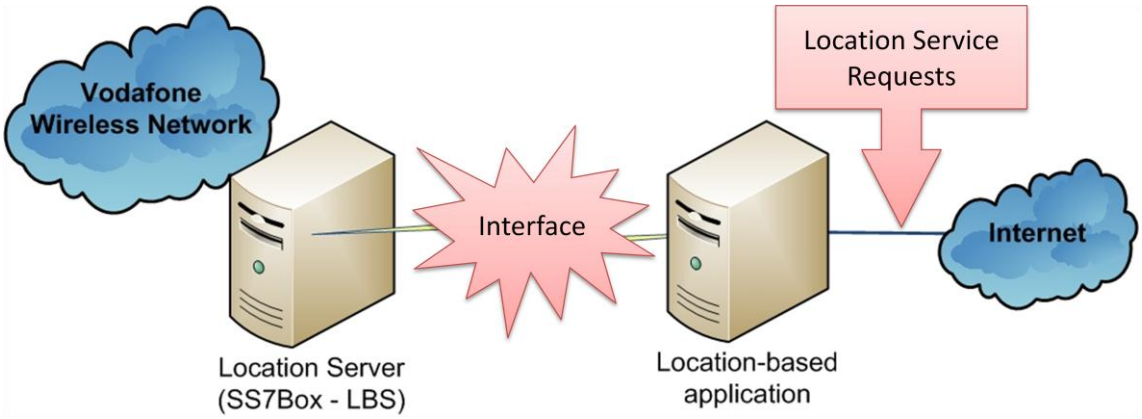



**Figure 3: Environment description.**

Then the LBS module of the SS7Box can provide the location information, but it does not have any interface where the clients from outside can connect and request that location information.

The solution proposed in this project to solve that problem is to develop an Application Programming Interface (API) in the SS7Box that the clients can use to get the location information. Then, this API is called Location Application Programming Interface (LAPI).

So the LAPI has to be a tool which the LCS clients can use to request the position of a mobile subscriber. That tool has to be able to reach from internet. On the other hand, the LAPI has to be perfectly integrated in the SS7Box and has to be able to communicate with the LBS module.

## 3.2. The LAPI

On the telecommunication market there are a lot of possibilities and technologies to make a LAPI, but not all the possibilities fit in our environment.

The LAPI implemented in this project has to:

- define a standard method for getting the position of mobile stations.
- be independent of the underlying network technology.
- be easy to use and implement.
- be able to serve location information to more than one location-based application.
- be a solid standard.

## 3.3. LAPI technology: MLP

Finally, MLP has been chosen as the base of the LAPI. Because of MLP:

- defines enough services, or ways, to serve the location information.
- is a application-level protocol, that means that it is independent of the underlying network technology.
- is based on XML, that is an established technology with a lot of tools to process it.
- can be transported by HTTP. The HTTP servers can serve multiples requests at the same time.
- is a established protocol. Its last version is the 3.2V from 2005.
- is used by a established companies in the telecommunication environment.

### 3.3.1. Examples of companies implementations

On the telecommunications market exits other companies that have implemented the MLP protocol to do their projects.

| Project name | Description |
|---|---|
| Location Studio 2.1 (3) | The Openwave has developed an interface for requesting and delivering of mobile terminal location information. This project follows the LIF MLP 3.0.0. |
| User Plane Location Gateway and Privacy Manager (4) | The UP-LGPM operates with the LocationLogic platform to serve location information. In this project just the SLI is implemented and not all the elements are supported as well. |
| TCS Mobile Location Protocol Hub (5) | This Hub serves as an aggregator for such location requests. This hubs acts as a proxy server to serve the location requests/responses required for LBS. |

# CHAPTER 4. LAPI IMPLEMENTATION

## 4.1. Introduction

To do the implementation of the LAPI based on MLP, there are some points that have to be carried out.

The MLP is based on the interchange of MLP messages transported, for example, on HTTP. So, the LAPI has to implement a server where the MLP request arrives and the corresponding MLP answer are sent. That is the way to use the LAPI tool from internet.

The LAPI has to be able to communicate with the LBS module. So, it has to be perfectly integrated in the SS7Box.

## 4.2. The Application module

As it was explained on 3.1, the LAPI is located on SS7Box. So it has been programmed following the SS7Box structure.

The SS7Box consists of three layers. The lowest layer is the DataKinetics layer which provides SS7 stack functionalities. The next is the Core layer which provides communication between DataKinetics and the highest layer, the Application layer. So the LAPI is implemented as an application module and is called MLP module.

The LAPI is the interface between the LBS, which provides the location information, and the clients from internet. So the MLP module has to communicate with LBS and has to wait requests from internet. This module has to do some other process functions. The functions done by the MLP Module are showed on Figure 4. The functions have been divided in layers:

- Transport layer: there are the functions that represent the communications from/to outside of SS7Box. That means through Internet.
- Parser layer: there are the functions to process the MLP messages. That functions code and decode MLP messages, getting the necessary information from a message received or making the response message from the information get it from LBS module.
- Application layer: there are the functions that do the communication with the LBS module, receiving or sending messages.

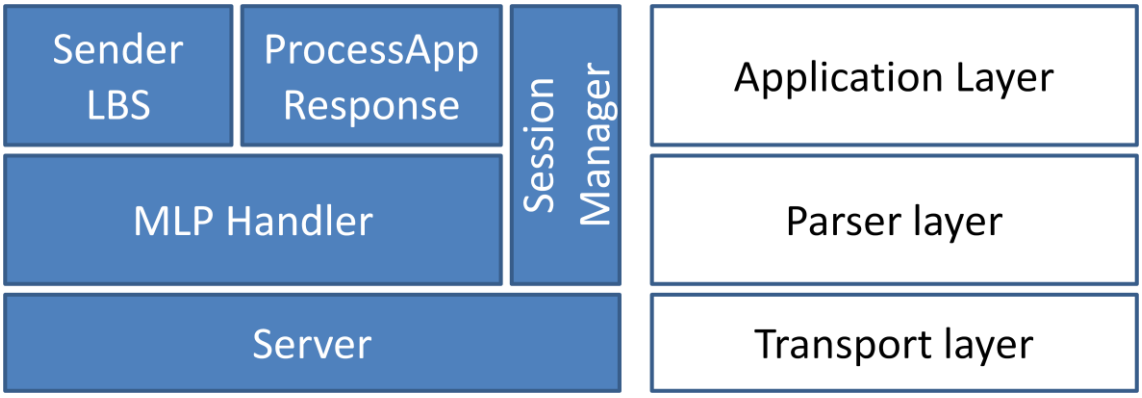



**Figure 4: the MLP Module functions**

## 4.3. MLP module

The MLP module includes all the functions showed on Figure 4. It manages all the functions to serve the location information correctly. In one word, it is the LAPI of the SS7Box. The location-service clients connect it to get location information.

The programming of the MLP module has been done following the definition of the application module in SS7Box. Therefore, the module has been programmed using a template called *startTemplate*. When the MLP module **starts**, the template initializes the queue of the module, where it receives the messages from other modules; the message-list, where the message has to be processed and creates an object of the *MlpModule* class. After that, the startTemplate starts the receiver, which reads the messages from the queue. Then the function *mainloop* of the *MlpModule* object is called and the module starts. After the module starts, the *mainloop* calls the *initialize* function. That is a virtual function, this means that is already declared (it is declared in *AppModule* which *MlpModule* inherits) but not defined. In the *initialize* function, the MLP module runs in a different thread the server. As an application module, the MLP module goes to the IDLE state.

As an application module, the MLP module can operate in the states:

- IDLE: the module is accessible only for configuration requests, state control messages and state request messages. All other messages are discarded.
- RUNNING: this is a fully functional mode – every interface of the module is working properly.
- SUSPEND: the module looks like RUNNING, but requests are not propagated further. Processing of requests is done autonomously.
- SHUTDOWN: this is a transition mode, as an alternative to immediate jump from RUNNING to IDLE mode. In this mode no new requests are accepted, but currently running requests are processed. This mode has a set-time in which it has to transit to IDLE mode, otherwise all current requests are discarded and module is turned to IDLE mode.

So, to **start** the MLP module, first the python script *mlp_create_queue.py* has to be executed. This script create the queue, it reserves the memory for it and define the

identifier of the queue. Then the command */mlp_local.sh mlpconfig.xml* has to be executed to **start** the MLP module. The *mlpconfig.xml* is a file where external parameters are defined. That file has to satisfy the *mlpconfig.dtd* . Finally to transition from IDLE mode to RUNNING mode and to **run**, it has to execute the python script *mlp_run.py* .

There is the possibility to define the behaviour of the MLP module in each mode, defining the functions *idle*, *running* and *suspend* which represent each mode and are located in *MlpModule*. That functions are declared in *AppModule* as virtual.

In the IDLE and SUSPEND modes there is no functionality defined or behaviour in their functions *idle* and *suspend* . One possible improvement could be shutdown the server when the module operates in those modes.

On the other hand, in the RUNNING mode there is defined the behaviour. In the *running* function the incoming messages are processed when the module is in that mode. So, if the module operates in RUNNING mode and message from LBS comes, the function *processAppResponse* is called. That is one of the functions showed in Figure 4.

## 4.4. processAppResponse

The goal of the *processAppResponse* is to process the application messages from the LBS module. This module only sends two kind of messages to the MLP module. An error message, that means that the location of the mobile subscriber has not been found, or a message with the Cell Global Identity (CGI).

Anyway, when the *processAppResponse* reads the message, it looks for the *SessionDataMLP* where all the session information of the mobile subscriber is saved. To find the *SessionDataMLP*, first it asks *SessionManager* who manages all the sessions.

The *processAppResponse* wants the *SessionDataMLP* object of the mobile subscriber to fill its *MLPServices* object with the information related with the MLP. This is explained in the next section.

So, if it is an error message, firts it looks the kind of error and then it fills the variable *resid* located on *MLPServices* with the error. One example is: the MLP module can receive an `USER_ERROR_numberChanged` error message, then the variable *resid* is filled with the value 4 that represents an UNKNOWN SUBSCRIBER. The kind of errors that the LBS module can sent are defined in *NetResult*. The kind of errors that the *resid* value can be are defined in (1) on the section 5.4 .

But if it is not an error message, we will get the CGI. That number is defined in (6). From CGI the Location Area Code (LAC) and the Cell Identity (CI). The CGI is a hexadecimal number made up by fourteen digits: *XXXXXXLLLLCCCC*, where the *L* digits represents the LAC and the *C* represents the CI. So, after getting the CGI, the

coordinates are not gotten yet. The coordinates are gotten from a DataBase where a query is sent using the LAC and CI. The query sent is:

```
SELECT longitude,latitude FROM cells WHERE LAC=(select cast(0xLAC as
unsigned)) AND CellID=(select cast(0xCI as unsigned))
```

The DataBase should return the coordinates of the mobile subscriber. Then the variable *X* that represents the latitude and the variable *Y* that represents the longitude. The DataBase can be located in other computer where the SS7Box can reach. The IP of the DataBase, the user name and the password have to be specified.

After that, a signal to the MLP Handler is sent. That function does the MLP message which is passed to the server and this sends the response with the MLP message.

## 4.5. Session Manager

The Session Manager is used to manage the sessions of each mobile subscriber. That is, when a request arrives at the server from LCS client requesting the position of one mobile subscriber, a *SessionDataMLP* object is created. That *SessionDataMLP* has an identifier different of the others *SessionDataMLP* already created. Then, the identifier and the pointer to the *SessionDataMLP* are saved in the *SessionManager.* That is saved in the *m_sessions* which is an object of the class *t_sessions*. That class is a kind of associative container called Map, formed by the combination of a key and a mapped value.

The three main functions in this class are:

- *insertSession*: it is used to add a new Session into the *SessionManager*. the SessionDataMLP object pointer and the only identifier have to be specified.
- *deleteSession*: it is used to delete an old Session of the *SessionManager*. The identifier has to be specified. Therefore, this function search in the *m_session* a key equals the identifier, after finding the key; it deletes the *SessionDataMLP* object associate.
- *getKeyData:* it is used to search the *SessionDataMLP* object. The identifier has to be specified. Then this function search in the *m_session* a key equals the identifier, after finding the key, it returns a pointer to the *SessionDataMLP* object associate.

### 4.5.1. SessionDataMLP

The *SessionDataMLP* is made up by:

- *cgi*: that is the Cell Global Identity, and it is the hexadecimal number wich the LBS module sents to the MLP module when the MSISDN has been found.
- *lac*: that is the Location Area Code
- *ci*: that is the Cell Identity
- *m_MLPServices*: that is a pointer to an object of the MLP implementation class.

- *mute*: that is the mutual exclusion.
- *condition_cond*: that is the condition which keep waiting or not the MLP Handler.

Among the *SessionDataMLP* function there are *WaitComplete* and *SignalComplete* function. The first function is called by the *Handler MLP* to keep waiting for the signal sent by the *processAppResponse* when it has filled the *SessionDataMLP* with the location information. That signal is sent whit the function *SignalComplete*.

The *SessionDataMLP* represents the share memory between the *Handler MLP* and the *processAppResponse*. These two parts do different works into the MLP module and they do not know nothing about the work which does the other.Both parts read and write from the same *SessionDataMLP* object to complete the MLP response.

### 4.5.2. MLPServices

The MLPServices is the class which represents the implementation of the MLP. In the section 2.1.1 the structure of the MLP was explained. The MLP services are defined by elements and each element has or has not attributes. The description of the elements and attributes is in the DTDs. At the same time, the elements are organized in layers, the Service and Element layers. So, the implementation of the MLP has been done following that rule.

There are two classes. The first class defines the Service layer, so the variables of the only service implemented, the SLI, can be found there. That class is called *MLPServices* and inherits from *ELayer*. This class defines the elements and attributes of the Element layer. So the elements implemented can be found as a structure. For example: there is one element called result from the Result Elements in the Element layer. That element has an attribute called resid. So, in the *ELayer* class the structure that represents this element is:

```
struct resultTag{
                    string resid;
                    string message;
             }fillResultTag;
```

Where message is the value of the *result* element. Then, a summary of the elements implemented is showed in Table 3.

| Element | Definition | Description |
|---|---|---|
| msid | Identity Elements Definitions | Represents an identifier of a mobile subscriber |
| loc_type | Function Elements Definitions | Defines the type of location Requested. |
| result | Result Elements Definitions | Indicates the result of a request. |
| pos | Location Element Definition | it contains the elements: msid, pd or poser, gsm_net_param(optional) and trans_id(optional). |
| pd | Location Element Definition | it contains the elements: time, shape and alt, alt_unc, speed, direction, lev |

| | | |
|---|---|---|
| | | conf and qos_not_met optionally. |
| shape | Shape Element Definition | it contains the elements: Point or LineString or Polygon or Box or CircularArea or CircularArcArea or EllipticalArea or MultiLineString or MultiPoint or MultiPolygonor LinearRing |
| point | Shape Element Definition | it contains the elment: coord |
| coord | Shape Element Definition | it contains the elements: X, Y (optional) and Z (optional). |
| X | Shape Element Definition | X is the first ordinate in a coordinate system, and denotes the latitude. |
| Y | Shape Element Definition | Y is the second ordinate in a coordinate system, this is optional in a linear coordinate system, and denotes the longitude. |
| Z | Shape Element Definition | third ordinate in a coordinate system which has at least three ordinates |
| poserr | Location Element Definition | it contains the elements: result, add_info(optional) and time. |
| hdr | Header DTD | it contains the elements: sessionid or client and sessionid (optional), subclient and requestor (optional) |
| client | Context Element Definition | it contains the elements: id, pwd(optional), serviced(optional) and requestmode(optional). |
| id | Context Element Definition | A string defining the name of a registered user performing a location request. In an answer the string represents the name of a location server |
| pwd | Context Element Definition | The password for the registered user performing a location request. In this answer the string represents the password for a location server |

**Table 3: Elements implemented**

With this implementation there are two kinds of MLP answer supported: the SLIA and the SLIA with error. That is showed in Table 4 and Table 5.

On the other hand, the decoder of the MLP requests is not done. But there are the variables to decode a message like the message showed in Table 6.

```
<?xml version = "1.0"?>
<!DOCTYPE slia SYSTEM "MLP_SLIA_300.DTD">
<slia ver='3.0.0'>
 <pos>
  <msid>msisdn</msid>
  <pd>
   <shape>
    <Point>
     <coord>
      <X>X</X>
      <Y>Y</Y>
     </coord>
    </Point>
   </shape>
  </pd>
  </pos>
 </slia>
```

**Table 4: SLIA implemented**

```
<?xml version = "1.0"?>
<!DOCTYPE slia SYSTEM "MLP_SLIA_300.DTD">
<slia ver='3.0.0'>
  <pos>
   <msid>msisdn</msid>
   <poserr>
    <result resid=error>message</result>
   </poserr>
  </pos>
 </slia>
```

**Table 5: SLIA with error implemented**

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE hdr SYSTEM 'MLP_HDR_300.DTD'>
<hdr ver='3.0.0'>
 <client>
  <id>testlst</id>
  <pwd>password</pwd>
 </client>
</hdr>
<slir ver='3.0.0' res_type='SYNC'>
 <msids>
  <msid type='MSISDN'>3035551003</msid>
 </msids>
 <loc_type type='CURRENT_OR_LAST'/>
</slir>
```

**Table 6: SLIR example**

## 4.6. MLP Handler

The *MLP Handler* does the processing of the messages incoming from internet and to internet. That is, when a request arrives at the server, this get the MLP message and

that message is passed to the MLP Handler. Then, it creates a *SessionDataMLP* with an only identify. Then the message is passed to the Sender LBS.

On the other hand, this function receives the Signal from the *processAppResponse.* Then, it starts to do the MLP response message which is passed to the server.

This function is not done in this project.

## 4.7. SenderLBS

This function sends the request to the LBS module. That is a request asking the position of a MSISDN.

This function is not done in this project.

## 4.8. Server

The MLP specify different kinds of transport layer. In this project the HTTP protocol has been chosen.

The design and implementation of this server is not done in this project.

# CHAPTER 5. MLP MODULE OPERATION

## 5.1. Block diagram

In this section the block diagram of the MLP module is showed in Figure 5: Block diagram Figure 5. In this diagram the communication between the functions are showed.

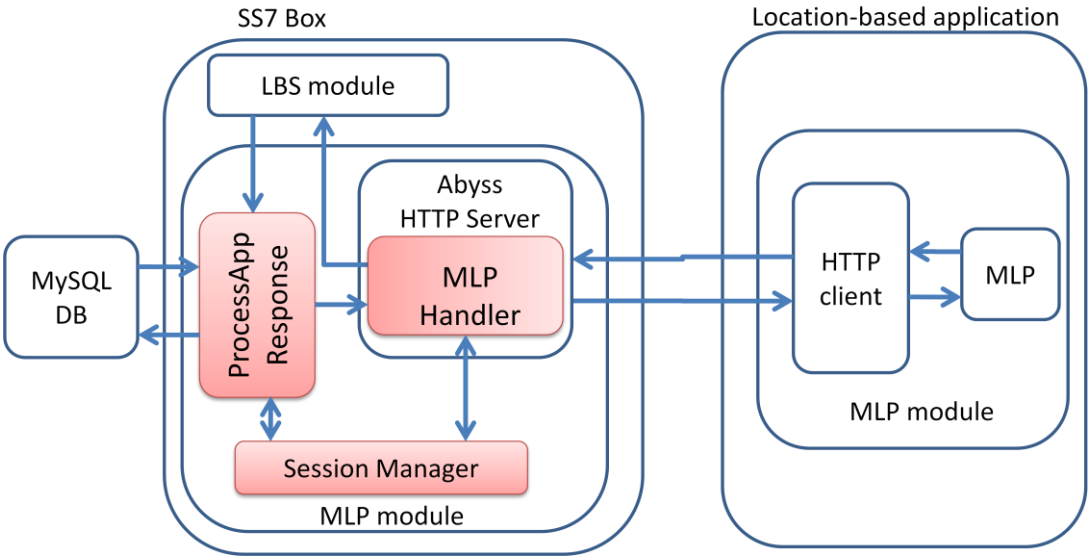



**Figure 5: Block diagram**

This diagram shows how a LCS client sends a request to the Server HTTP installed on SS7Box. The flow of the request and its processing is showed in the next section 5.2.

## 5.2. Flow diagram

In this diagram, showed on Figure 6, it explained how the MLP module works.

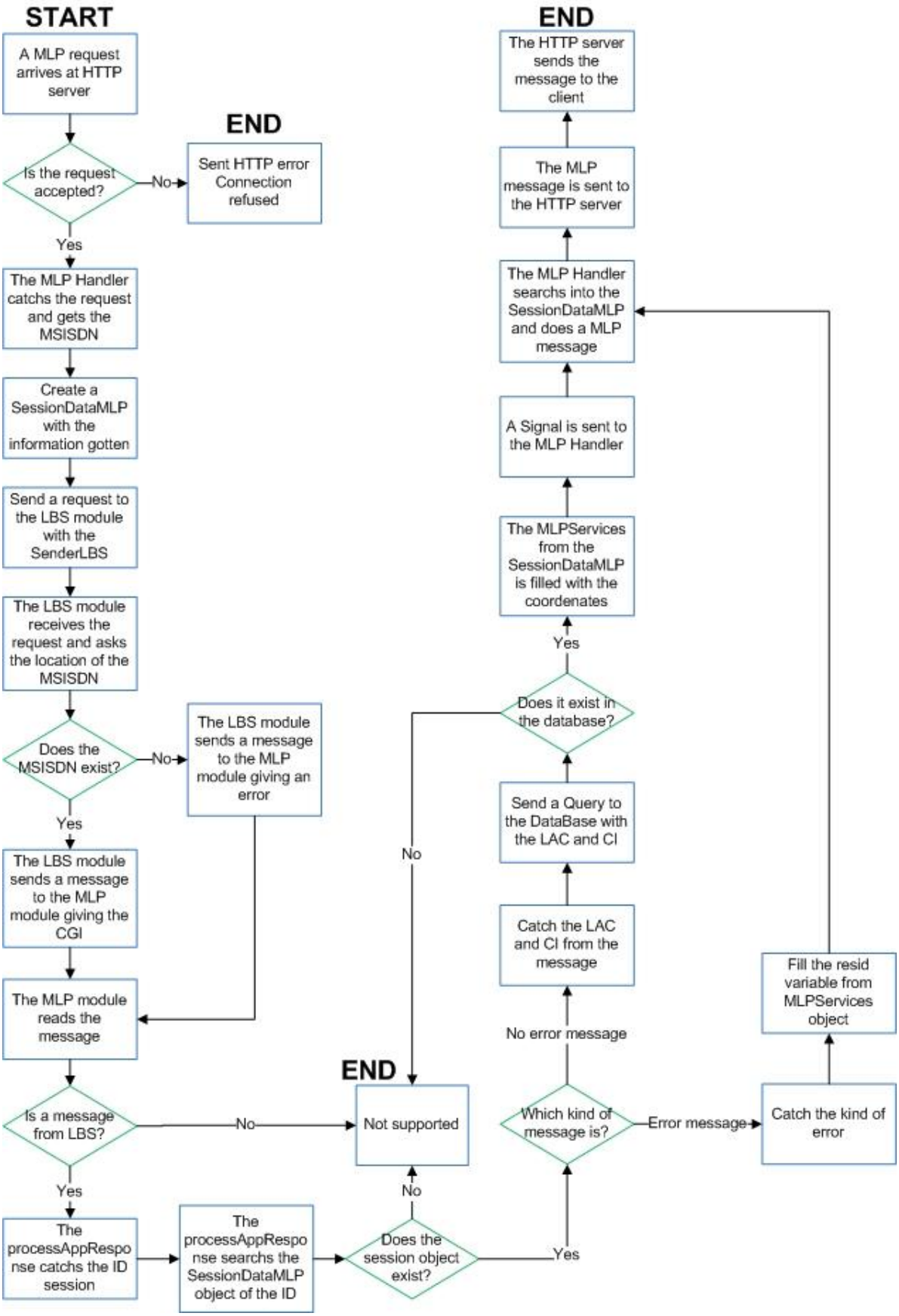


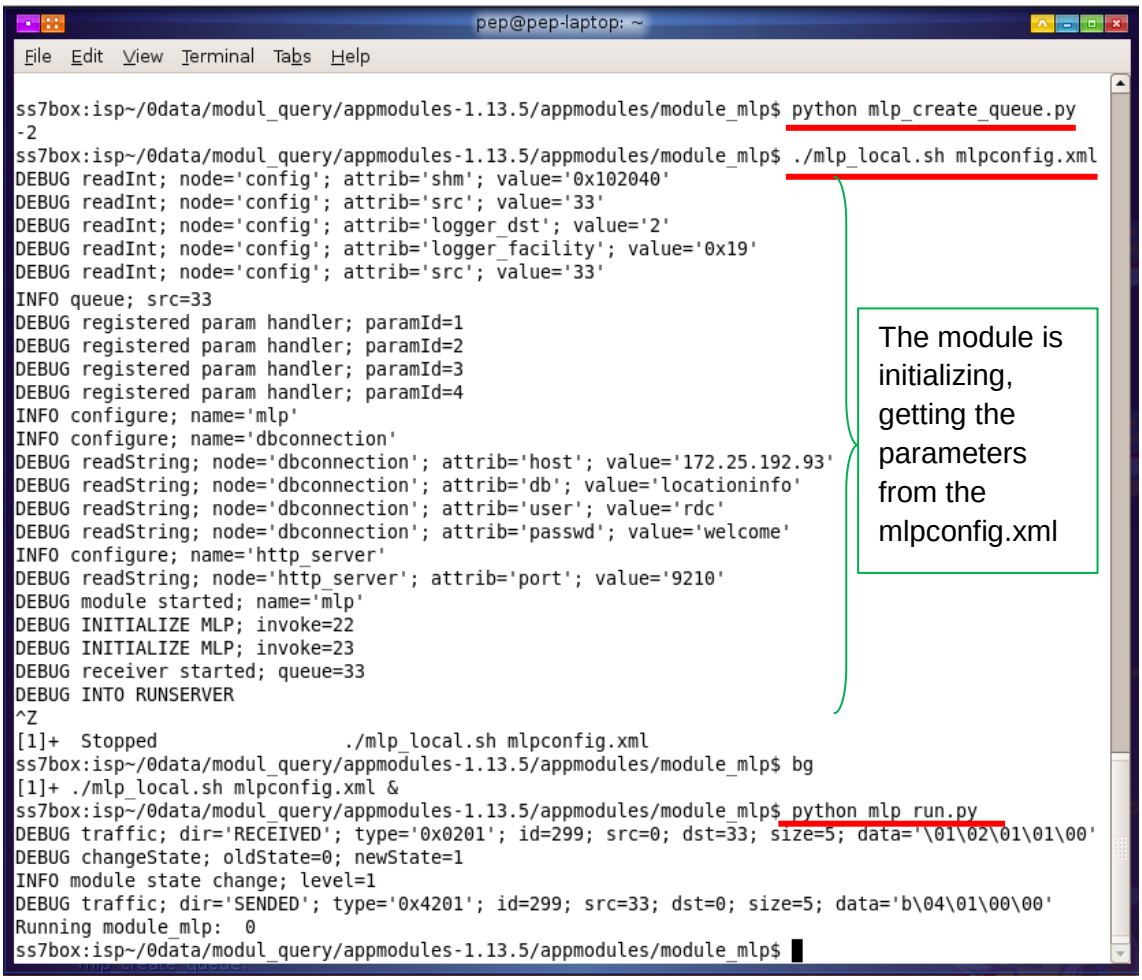

**Figure 6: flow diagram**

# CHAPTER 6. RESULT OBTAINED

## 6.1. Running the MLP module

As it was explained on 4.3, the commands to run the MLP module are:

1. python mlp_create_queue.py
    a. if the result is a -2, that means that the queue was already created
    b. if the result is a 0, that means that the queue was successfully created.
    c. other case, there is an error.
2. ./mlp_local.sh mlpconfig.xml
3. python mlp_run.py
    a. if the result is a 0, that means that the module has been successfully run.
    b. other case, there is an error.



## 6.2. Non-successful SLIA

In this section, the sending of a non-successful SLIA is explained. That is when there is a problem and the MSISDN has not been found by the LBS module and the MLP module has to send an error message to the client.

1. The MLP module is running, so the client can be connected.
2. The client is started. Figure 7

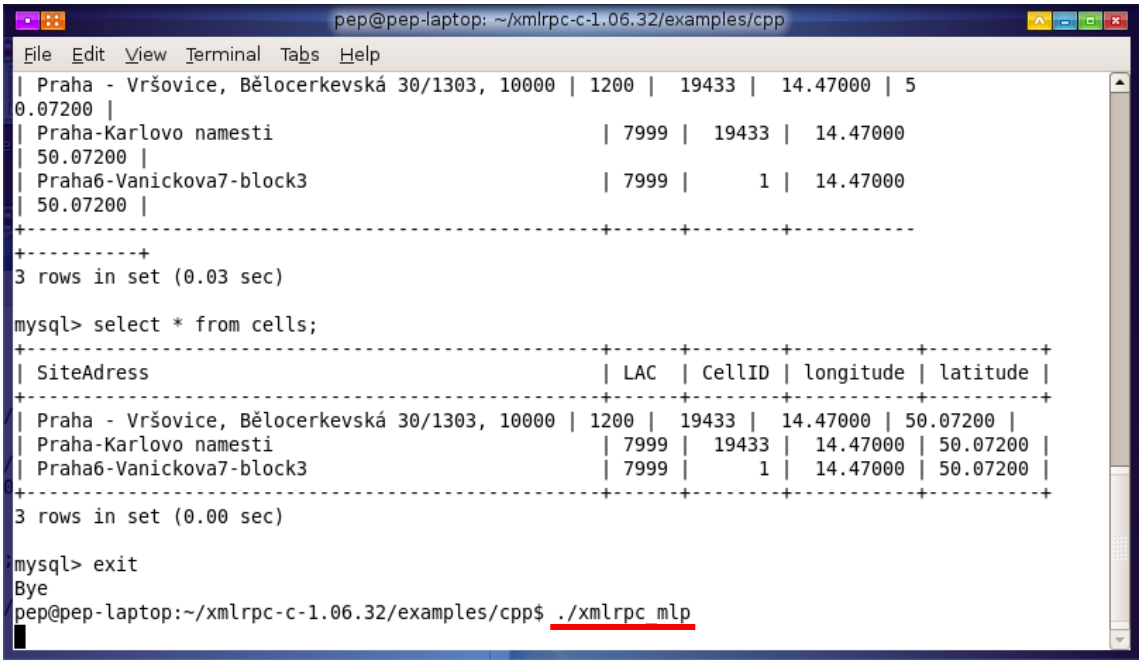



**Figure 7: starting the client.**

3. The server HTTP has received the request and it is processing the request. The command line shows all the process; from the server receive a request until the response is done. It is recommended see the flow diagram in Figure 6.

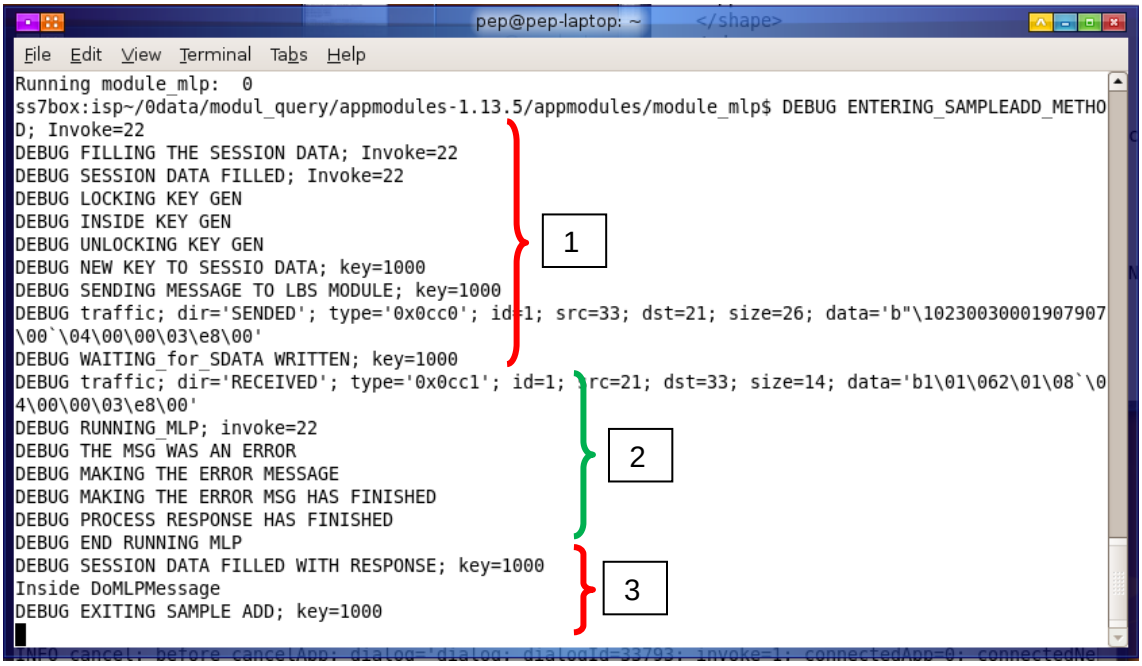



**Figure 8: Processing the request with an error**

The number 1 represents the process of the *MLP Handler*. This function creates a *SessionDataMLP* for the incoming request with an identifier (key) number 1000 and sends a request to the LBS module. Then the MLP Handler keeps waiting for the Signal sent by the *ProcessAppResponse*. At number 2, there is a incoming message to the MLP module. The incoming message is processed by processAppResponse. It realizes that it was an error message and fills the SessionDataMLP with the identifier (key) number 1000. Then it sends a signal to the *MLP Handler.* At number 3, the *MLP Handler* is woken up with the Signal sent by the processAppResponse. The handler continues the work and does the MLP message which is sent to the client. Figure 8

4. Finally the client receives the MLP message with the error, showed in Figure 9. This is one of the messages showed in 4.5.2.

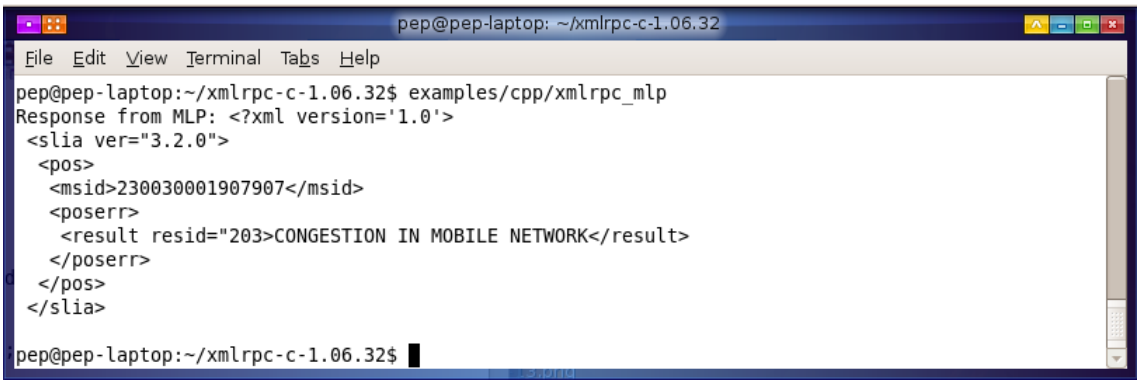



**Figure 9: the SLIA message with error**

## 6.3. Successful SLIA

In this section, the sending of a successful SLIA is explained. That is when the MSISDN has been found by the LBS module and the MLP module is able to send a MLP response with the coordinates to the client.

1. The MLP module is running, so the client can be connected.
2. The client is started. Figure 7
3. The server HTTP has received the request and is processing the request. The command line shows all the process; from the server receive a request until the response is done. It is recommended see the flow diagram in Figure 6.

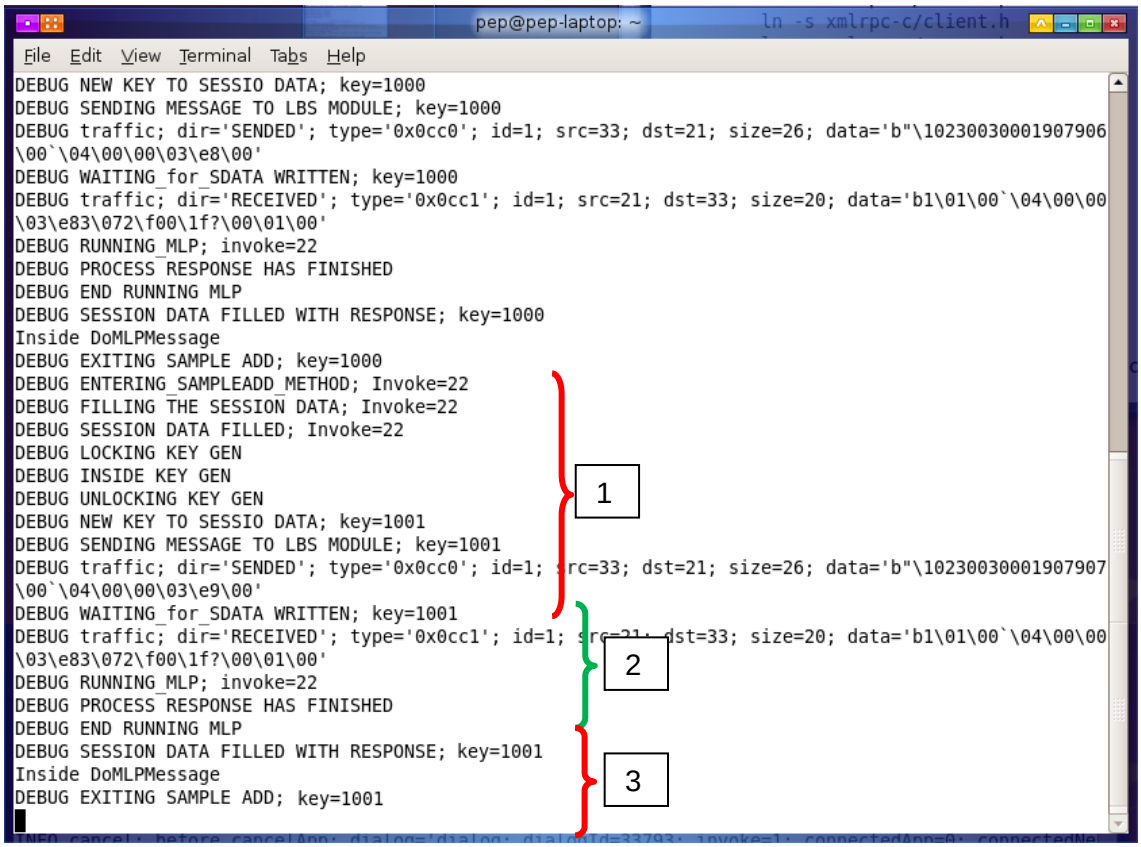



**Figure 10: Processing the request without error**

The number 1 represents the process of the *MLP Handler*. This function creates a *SessionDataMLP* for the incoming request with an identifier (key) number 1001 and sends a request to the LBS module. Then the MLP Handler keeps waiting for the Signal sent by the *ProcessAppResponse*. At number 2, there is a incoming message to the MLP module. The incoming message is processed by processAppResponse who fills the SessionDataMLP which has the identifier (key) number 1001. Then it sends a signal to the *MLP Handler.* At number 3, the *MLP Handler* is woken up with the Signal sent by the processAppResponse. The the handler continues the work and does the MLP message which is sent to the client.Figure 10.

Note that now the identifier is 1001. That is because in the last example, the identifier used was the 1000. This identify has to be different for each request.

4. Finally the client receives the MLP message with the coordinates, showed in Figure 11. This is one of the message supported explained in 4.5.2.

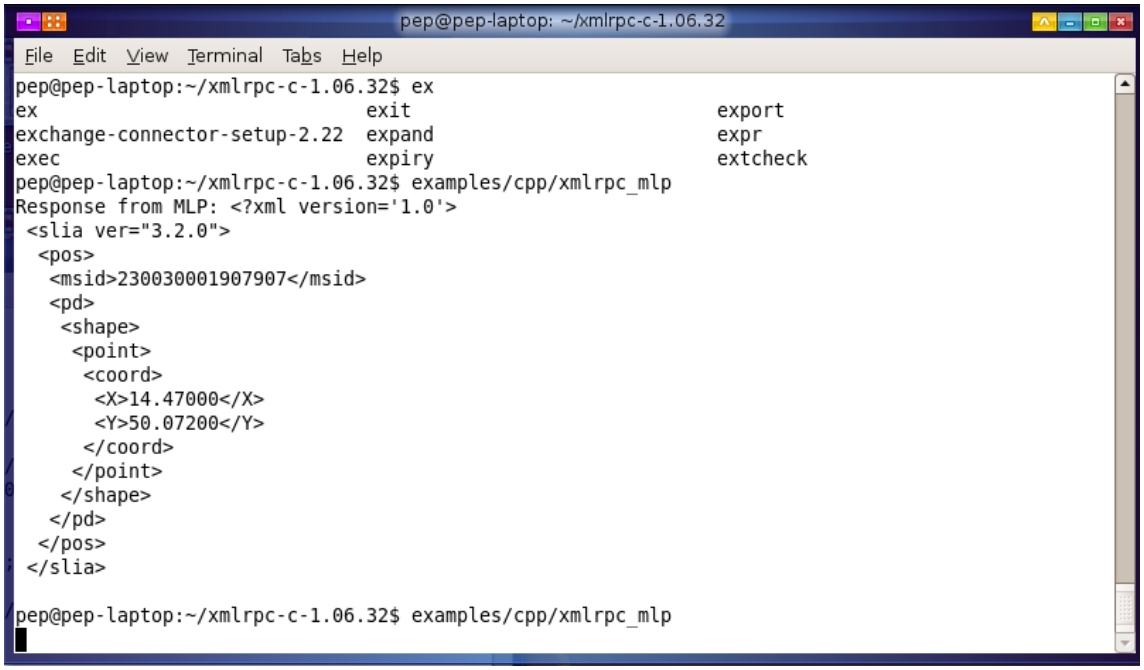



**Figure 11: the SLIA message with the coordinates.**

# CHAPTER 7. PROJECT REVISION

## 7.1. Improvements

Right now there are a lot of thing to improve yet. Some possible improvements and their explanation are showed in Table 7.

| Improvement | Description |
|---|---|
| Some states not supported | On the flow diagram in Figure 6 there are states with the tag “not supported”. To check the module are not necessary, but to be implemented in a real system they are really important. |
| Coordinates in Degrees Minutes and Seconds. | Right now the coordinates are given just in Degrees. A converter would have to be implemented. |
| Make a decode MLP | There is a code MLP implemented, but there is no decode MLP to read the messages from clients. |
| Timeouts | The time outs are not completely implemented. Right now the MLP Module receives timeouts messages but it does not do anything with them. |
| there are four services left | Each LCS server can chose which service wants to implement, but it could be fine to make up a server who supports all the services. |
| Support more than one MSISDN request | There is the possibility to request more than one MSISDN. Right now that is not supported in this implementation. |
| Improve the programming | Definitely the source-code can be improved. |

**Table 7: Improvements.**

## 7.2. Conclusions

The first conclusion is that there is a long way yet to do if the LAPI wants to be finished. There is a lot of thing to do yet. Anyway, the base of the LAPI has been done and the project has achieved some results. That means that a basic LAPI is working. Anyway we really think that this project can work in a real environment some day.

Another conclusion is that the implementation of a LAPI in a system already done is not easy. There are a lot of thing to learn about the LAPI and the system. Finally not all the system has been known, but the enough knowledge has been gotten to design the LAPI.

The last conclusion is that this project has been useful to get new knowledge about, tracking, complex-systems, new services, API, programming, etc.

# REFERENCES


1. **Alliance, Open Mobile.** *Mobile Location Protocol 3.2.* 2005.
2. **MySQL AB.** *MySQL 5.1 Reference Manual.* 2008.
3. **Openwave Systems, Inc.** *Location Studio 2.1 - MLP 3.0.0 Developer's guide.* 2004.
4. **Autodesk, Inc.** *Mobile Location Protocol Support.* 2005.
5. **TeleCommunication System, Inc.** *TCS Mobile Location Protocol Hub.* 2007.
6. **3rd Generation Paternship Project.** *3GPP TS 29.002 V8.1.0.* 2007.
7. **RDC.** *SS7Box - Architecture design version 1.1.*