iMIFARE – A RFID SOLUTION FOR TRACKING MUSEUM VISITS

Danilo Valvo, Luca Porcaro, Alberto Palazzo

DIEEI, University of Catania
Catania, Italy

{danilovalvo, alberto.palazzo1, lukylukep}@gmail.com

Abstract

Information technology has provided a great force to the communication of art galleries, offering a variety of channels through which they convey information flows: in recent years many technological solutions have been developed to improve the quality of interaction that will occur between the visitor and the items shown in the museum. This is the target of the project "Museum – Interactive – Working – Through – Rfid Elements", whose purpose is to provide not only a personal guide, but also an intelligent search system and a mechanism for tracking the visitor.

Unlike many other papers [1, 2] which discuss this topic using a PalmOS-like devices, in this one the system is fully working and implemented on common smartphone OS – like iOS.

I. INTRODUCTION

RFID (Radio Frequency IDentification) [3, 4] is a technology to identify things through a radio frequency transmission.

The identification involves assigning a unique identity to an object which allows to distinguish in an unambiguous way. The main aim is to take information on objects, animals or people by small radio frequency devices associated with them.

Gathering information relates to operations research, selection, tracking, identification, spatial localization.

The RFID diffusion has taken place the 90s, and currently there are numerous commercial solutions at low cost.

RFID technology consists of three basic elements:

– TAG: a little radio frequency transponder constituted by an integrated circuit (chip) with functions of simple logical control, equipped with a memory, connected to an antenna and inserted in a container or incorporated into a label, a smart card, a key or electronic devices (watches, phones, etc..). Allows the transmission of short-range data without physical contact;

– Reader: a transceiver controlled by a microprocessor and used to query and receive information in response to the TAG;

– Management System: when it exists it is networked with the Reader.

The TAG

Tags (or transponders) are distinct for the management by energy sources:

– *Passive:* they derive the energy for the operation by the signal from Reader, do not possess its own transmitter, but re-radiate the modulated signal transmitted from Reader and reflected by its own antenna;

– *Semi-passive*: with battery used to power the microchip or auxiliary devices (sensors), but not to power a transmitter; in transmission they behave as passive TAG;

– *Active:* powered by batteries. They incorporate receiver and transmitter as the Reader.

Passive TAG devices are typically low cost and small dimensions that allow to carry out many kinds of applications. They are generally made only by an antenna (typically printed) and a miniaturized integrated circuit, and can be included in credit cards, stickers, buttons and other small pieces of plastic, paper, notes and tickets, this creates real objects "talking". The TAG also can be either read-only or read-writable.

For active or semi-passive TAG, in addition to the greater amount of memory and the function of rewritable of the same, the technological evolution has allowed to add, in some cases, functions that exceed the pure identification. Are mentioned, for example, the functions of radiolocation (RTLS - Real Time Location System) or the measurement of environmental parameters through sensors (temperature, movement, etc.).

When the TAG passes through the electromagnetic field (EM) generated by a reader, it transmits to the reader its own information.

Typically, a passive tag that receives the signal from a reader, uses the same signal energy to power its internal circuitry and, consequently, "wake up" their functions. Once the TAG has correctly decoded as the signal of the Reader, it responds reflecting its antenna and modulating the field emitted by the reader.

The Reader

The Reader (also called "interrogator" or "controller") is the element that allows you to take the information contained in the TAG.

This is a real transceiver, governed by a control system and often connected to a network with management information systems to be able to extract information by the identifier transmitted by the TAG that, unlike bar codes, is unique.

The Reader for active TAG are controlled transceivers, they can use various techniques to RF energy. The active TAG, now, are only a small part covered by specific standards. The TAG Reader for passive (and semi-passive), instead, must emit RF signals of a particular type, able to provide to the TAG also the energy necessary for the response.

Applications

In a broader sense the RFID technology includes a wide range of micro-devices that are used to identify products. The electronic toll collection, tags implanted in animals for identification, access control (and card) which is used to read in proximity without physical or visual contact, central locking systems for motor vehicles. The "ski pass" are all forms of RFID systems.

The communication frequency between reader and tag depends on the nature of the TAG and the intended application, and it is standardized by international and national bodies.

The regulation, however, is divided into geographical regions with different laws from region to region.

Now, some frequency bands (typically in LF or HF) are accepted around the world. An example is the band of 13.56 MHz, used by many passive tag. The choice of the working frequency affects the range of operation of the system, the interference with other radio systems, the speed of data transfer and the size of the antenna.

In this paper we will present a system for visitor management into a museum which provides both a support to the visitor and to the museum.

The purpose of a museum visit is to obtain information on the exhibits. The RFID reader connected to a smartphone will detect the tag simply by approaching the artwork, and through a connection to the internal database of the museum, will provide the visitor its content (text or multimedia).

The project provides a mechanism for internal communication: When a user performs a reading of the tag, the smartphone uses the wi-fi system to communicate its position and access to remote data.

The information may include information regarding the historical period, author, etc. Using these notions will be possible to make specific searches.

From a management perspective, it is essential to track users movements between the artworks. It is useful to estimate the number of visits and the routes preferred by users (Figure 1).

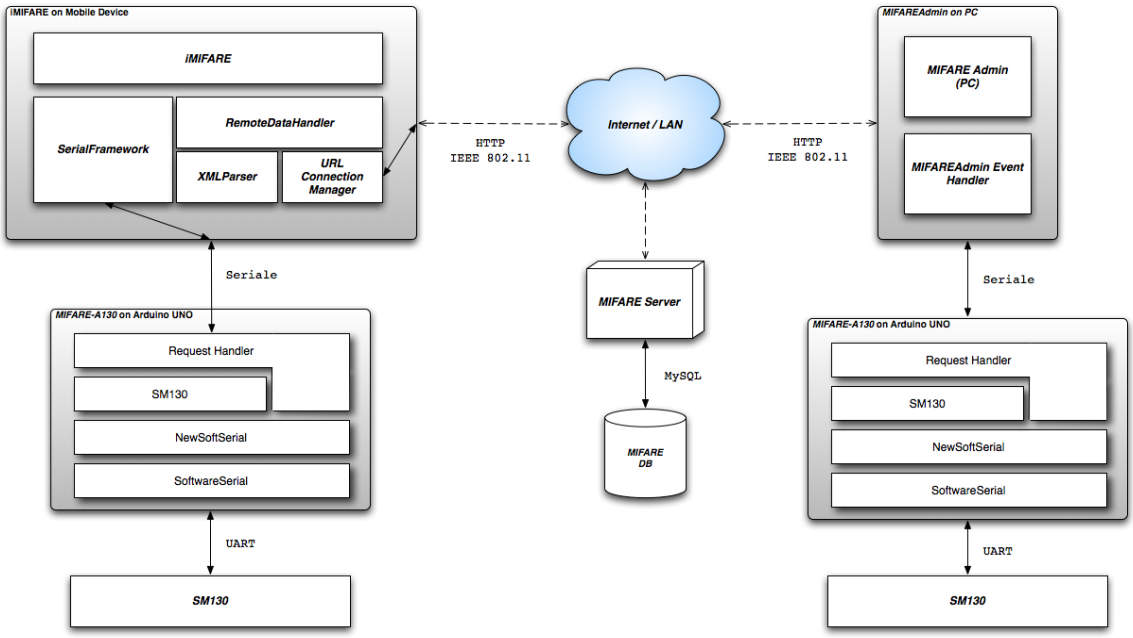

Fig. 1. Mifare Architectural scheme

II. HARDWARE

We will start first with a description of the components used in the project and later we will describe the software architecture of the system.

Arduino Uno

The Arduino [5] is a microcontroller based on the ATmega328 chip, made entirely in Italy and open source. It has 14 digital pins Input / Output, 6 analog inputs, a 16 MHz crystal oscillator, a USB connector, a power jack, an ICSP header, and a reset button. The Arduino can be powered by a USB cable, with an AC-DC or with a battery.

Each of the 14-pin digital on Arduino can be used as input or output, using the pinMode(), digitalWrite() and digitalRead().

The ATmega328 chip provides two methods of communication: UART and I2C.

The UART TTL serial communication is available on digital pins 0 (RX) and 1 (TX) and appears as a virtual COM port for your computer.

The Arduino software includes a serial monitor, which allows you to send and receive data flow to and from the card with simple text commands.

RFID Mifare Module SM130 – 13.56 MHz

SM130 [6] is a compact 28-pin DIP module that includes all necessary components for 13.56 MHz RFID Mifare System (Reader / Writer), except only PCB antenna.

The module performs all demodulation, decoding, encryption and decryption, has 2 inputs general features and 2 outputs for switches, relays, etc.

It's also equipped with two communication interfaces, UART and I2C. The controls are the same for both communication protocols, but the frames are different. Our choice fell on UART.

UART

Communication between the host and the module can be 9600bps, 19200bps, 38400bps, 57600bps or 115200bps.

The module communicates by default to 19200bps. Once the baud rate is changed, the communication will be successful only with the new baud rate.

The host will send the first command to the SM130 module that will perform the operation, then the module will send back a reply. The host will be able to analyse the response to see if the operation was successful or if an error occurred during the operation.

RFID Evaluation Shield – 13.56 MHz

This card is an evaluation platform for the SM130 module. Include a header XBee, a PCB antenna and a small prototyping area. This tab can also be used as an antenna for the RFID module SM130 and represents the natural link with the platform Arduino, in fact, its layout is designed to be used as a shield for that card.

RFID Tag - Transparent MIFARE 1K

It is an RFID tag, which follows the basic guidelines within the standard MIFARE 1K [7]. The tag has 1K of memory, and data can be written and read by a compatible device. Its label is transparent and has a diameter of 25mm and an overall thickness of about 0.7 mm.

Basically it is only a memory device, whose storage area is divided into 16 sectors with 4 blocks of 16 bytes each, with simple security mechanisms for controlling access. They are based on ASIC and have a limited computing power.

Mobile Device

The mobile device (*iPhone – iPod Touch – iPad*) must be able to communicate with the hardware already described (Arduino – SM130 – Evaluation Shield). The easiest way is a serial connection between iPhone and Arduino. The Arduino board has already integrated a serial interface accessible via pins 0 and 1 (respectively Reception and Transmission).

Each Apple mobile device is equipped with a connector in its lower part, called Dock Connector.

Dock can be accessed via the serial interface of the Apple device exposed from pins 12 and 13 (respectively Serial Serial Tx and Rx) of the Connector.

At this point of our work we found the following issues: individual connector pins are difficult to reach, the iOS operating system does not allow access to FileSystem and is impossible to manage the communication on the serial port.

For this reason it was used a small board called *iPodBreakOut* [8], consisting of a standard male connector for connecting to Apple's Dock and a PCB that "carries" the connections for each pin.

For correct operations have been connected the following pins:

- pin 12 from iPodBreakOut to Arduino RX;
- pin 13 from iPodBreakOut to Arduino to TX, via a resistor of 1 KΩ (so as to bring down the voltage from 5V to 3.3V);
- pin 16 from iPodBreakOut to GND.

III. MIFARE SOFTWARE

MIFARE Software is composed by four different component. In this chapter we start to analyze the communication and the interaction protocols between them:

- *MIFARE-A130*, Software designed to manage the HW components;
- *iMIFARE*, an user interface software for mobile devices;
- *MIFAREAdmin*, an admin interface;

– *MIFAREServer*, a PHP server used to help the user and the admin querying a remote database.

Communications

In order to have a working iMIFARE system, the components described above need to exchange information and messages of various kind and nature.

Arduino Uno & Module-SM130

During communications, the host (in our study case, iMIFARE software via Arduino) will send commands to SM130, which, analyze the request, handle it, process it and send a response frame back to the host.

The host will analyze this frame to check if there was errors.

Table 1.
Frame used to exchange information between Arduino and SM130

Header	Reserved	Length	Command	Data	CSUM
1Byte	1Byte	1Byte	1Byte	NByte	1Byte

Following, the request[/response] frame sent[/received) from the host[/received from the SM130]:

– *Header*, composed of a single byte, indicating the frame start. This value has to be always 0xFF;

– *Reserved*, a future use reserved byte. Currently must be equals to 0x00;

– *Length*, this byte is used to indicate the payload length (the command field plus the Data field);

– *Command*, a single byte to identify the host requested operation;

– *Data*, field of length equal to N bytes representing information to be sent or received from the reader / writer. Contains at most 16 bytes of information;

– *CSUM*, a checksum byte. It can be calculated by adding all the frame bytes excluding the header byte.

iMIFARE, MIFAREAdmin & Arduino Uno

The iMIFARE and MIFAREAdmin applications communicate with the Arduino UNO and, therefore, with the SM130 module via a serial connection realized respectively by iPodBreakout and USB cable. Very Small information packet will be sent, helping the host selecting the function requested and, eventually, a few bytes of payload. This kind of frame will be used either for iMIFARE request or the Arduino Board response. In the last case, the data field could contains a message error if the request operation went wrong.

Table 2.
Commands between iMIFARE/MIFAREAdmin and ArduinoUNO

Command	Function
T	Seek and Read Tag
S	Seek for Tag(s)
D	Read a Tag
W	Write to Tag
V	SM130 Version Number
A	Antenna ON / OFF
R	Reset

iMIFARE/MIFAREAdmin & MIFAREServer

The application for mobile devices and one for the administrator also require a communication to a web service that enables users to leverage the data stored within a remote database. To communicate with the server we are using the HTTP GET requests.

<URL>?
<PARAMETER_KEY>=<PARAMETER_VALUE>
[&…]

Both the applications send requests to MIFAREServer, but with different parameters:

– iMIFARE
  - START & TAG, notify to the server that user start viewing an artwork;
  - ID, requesting tag information;
  - END, notifying to server that an user has stopped viewing an artwork;
  - CLOSE, The user get out of the museum (or he just close the application);

– MIFAREAdmin
  - INS / MOD / DEL, the admin trying to modify one or more tag inside remote DB;
  - READ, read of a tag content;
  - VIEW, consulting all the tags inside DB;
  - ART, Consulting all the artworks of the DB;
  - PEOPLE, Consulting all the people currently inside musems;
  - HIST, analyzing people historical visit;
  - STAT, analyzing artwork historical values.

3.1 MIFARE A-130 – ArduinoUNO Application

The Arduino Board realize what commonly named Open Source Hardware. This board can be programmed, as the user likes, by programming it in a C++ similar language, using a set of libraries owned by Arduino Platform.

Our whole MIFARE A-130 Application is composed by two modules only:

– a file having .pde extension (tipically, Arduino source file extension, became .uno in last versione of Arduino Platform) that filters all the external requests;

– a C++ class, named SM130. Inside of it we can find functionality designed by us used to interact with SM130, connected with Arduino UNO.

Both these components use and reference to a common library inside Arduino framework used to implement serial communication also called SoftwareSerial.

SM130 Class

This class implements almost all the features exposed by SM130 RFID module.

```
class SM130 {
public:
     SM130();
     ~SM130();
     const char* reset();
     const char* getFirmwareVersion();
     boolean setAntennaPower(boolean on);
     const char* seekTag(void);
     const char* readTag(byte blockID);
     const char* readTagBlock(byte blockID);
     boolean writeTagBlock(byte* info, byte
blockID);
     […]
protected:
     void sendCommand(byte command, byte* data,
int dataLength);
     byte* receiveResponse(int *extLength );
     Tag selectTag();
     boolean authenticate(byte blockID);
     const char* readTagContent(byte blockID);
     boolean writeTagContent(byte* info, byte
blockID);
};
```

Send and Receive data

Sending and receiving frame from and to SM130 module via Arduino Board are implemented with two protected methods (they're not visibile from the outsider class, so not directly callable, but only using public interface methods):

- void sendCommand(byte command, byte* data, int dataLength);
- byte* receiveResponse(int extLength);

Tag Read e Write

Reading and writing tag content as the tag ID reading need to implement a specific procedure (figure 2).

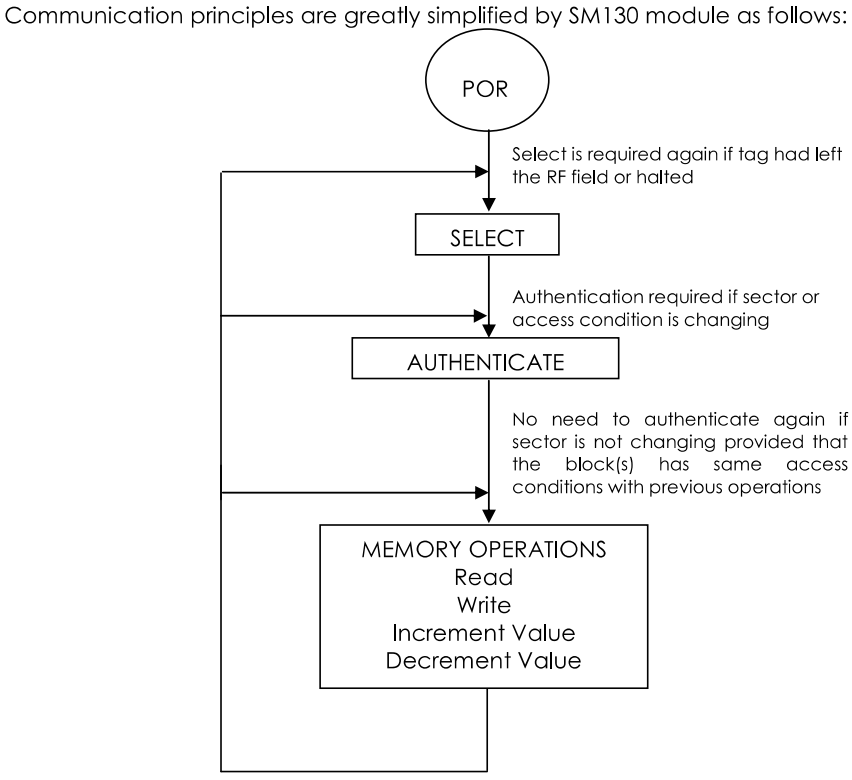

Fig. 2. Memory operation sequence

For consistency and security reason, before we can get tag informations and tag contents we have to execute this sequence of operations:

1. SELECT – Select the Tag, this step is necessary to ensure the presence of tag inside the reception field of the of RFID antenna reader. We have to notice that this function execute an anti-collision algorithm on a firmware level. This factor can be an advantage in the case you want to approach to implementate this kind of algorithm, but in our case has been a disadvantage because we have not been able to optimize or improve the mechanism of the collision (the firmware is not editable).
2. AUTHENTICATE, after a tag was selected among the "read" ones, it'll needs to use one of two keys (A Key and B Key) to indicate in which of the 16 memory block we want to access.

3. MEMORY OPERATIONS, from now on we can start to execute our memory access operation, in our study case represented by reading or writing information inside RFID Tag Memory (1KB).

3.2 iMIFARE – iPhone Application

The user interface is managed by iMIFARE application. This is designed to run on Apple's operating system devices, especially iPhone or iPod Touch.

The project can be divided into the following main sections:

- Menu;
- Map;
- List of Works;
- Guided-Mode;
- Settings.

There are also important components that allow you to "sync" with other local information contained in a remote database (figure 3).

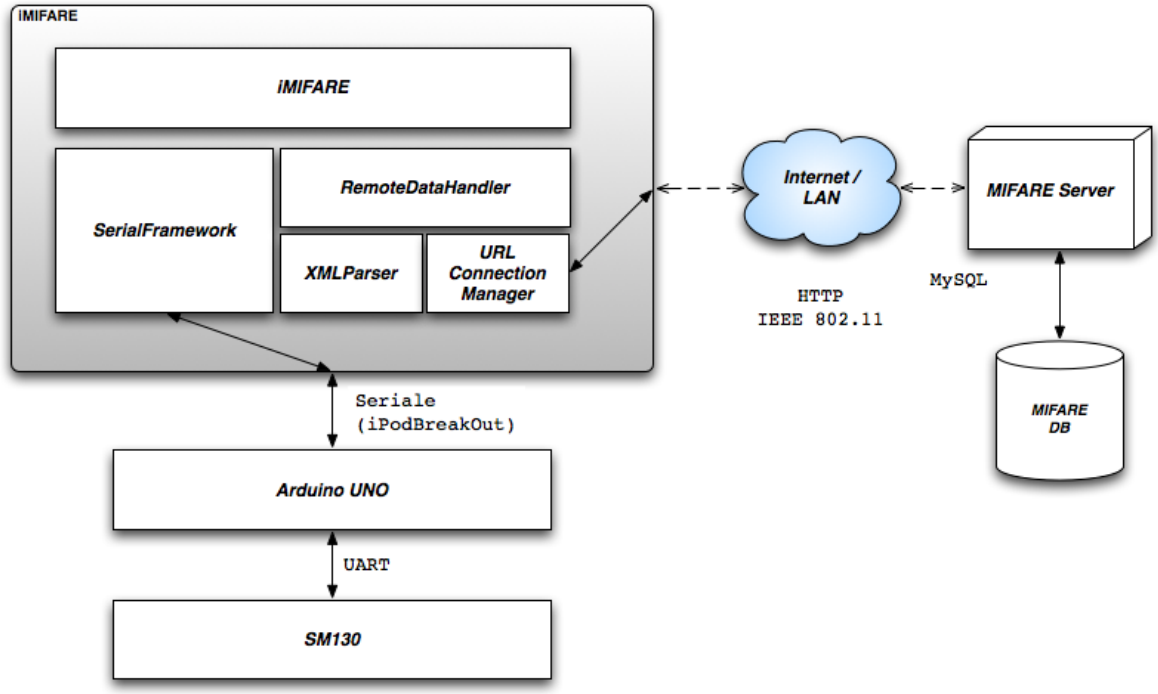

Fig. 3. Global scheme with details on mobile application

Guide Mode Section

The Guide Section is one that achieves the reason for which this software has been implemented. It allows a connection to the Arduino via the serial port. We will see in this chapter what are the classes and libraries used to perform this operation.

Logically, this components has to read a Tag – when this's reachable inside the action field of the reader – and show the related artwork details. In the described operation both the connection to Arduino and MIFAREServer connection are used.

Communication with Arduino is used to read information from the tag, especially, his ID, identifier assigned directly to the manufacturer, unique in the world. The system is already developed to also read the contents of the tag, even though in our case, no sensitive information is contained within it. Once the information is taken from the tag, it will be stored within an ADT (Abstract Data Type) represented by a class named "Tag".

We realize communication with the server, mainly, for one reason: to warn the server that a particular user – identified from a UUID (Universal Unique Identifier) – is displaying a given work – identified by a TAG ID. This information allows the server to start a timer, which serves as a further safety mechanism in the case where the user has moved viewing another artwork, but we lost his new request.

This request is called START request. Opposite to the START message, we can send another message called END, launched after the closing of Guide-Mode Function. Another similar message – CLOSE – allows you to close the logical session with the server, telling it that you just get out of the museum or the user has just exited from the application or he shutted down his device.

Inside Guide MVC (Model View Controller – Software design Pattern**)** has a fundamental importante a class called SerialDevice**.** This is a C++ class – which can be included without any compatibility problem inside an XCode project – which allow to start serial port operation with our own Apple device. This kind of operation are normally forbidden inside iOS mobile operative system, that's why we JailBroken our device. Neither the classes and library used for serial connection are available inside Cocoa Touch iOS Framework, so, we implemented them relying on an

external and open framework called openFramework which includes advanced and tested features to exchange information on a serial bus.

SerialDevice is also a singleton class which can be accessed by calling the class method: [SerialDevice sharedDevice]. This class allow the programmer to use fundamental features to managing connection and data exchange on a serial port, as you can see from the SerialDevice header file described below:

```
@interface SerialDevice : NSObject <NSCoding>
{
    NSString *path;
    long baudRate;
}

@property (nonatomic, copy) NSString *path;
@property (nonatomic, assign) long baudRate;


- (BOOL) connect;
- (BOOL) connectWithPath:(NSString *)path
andRate:(long)aRate;
- (void) disconnect;
- (void) flush;
- (BOOL) writeString:(NSString*)stringData;
- (NSInteger) available;
- (NSString*) readBytes:(NSInteger)length;

- (NSArray*) deviceList;

+ (SerialDevice*)sharedDevice;
- (void) saveValues;

@end
```

We proceed this chapter analyzing some of the method implemented inside this class:

– connect, allow to connect to a serial port represented by the default UNIX File located at path /dev/tty.iap having a baudrate equals to a 9600bps;

– connectWithPath:(NSString*)path andRate:(long)aRate, allow the programmer to start a new serial connection using an UNIX File located at a given "path" position and a given speed measured in bps;

– disconnect, close the opened file used to connect the Apple device to the serial media;

– flush, force to write into output serial buffer;

– available, returns the number of bytes inside input serial buffer;

– readBytes:(NSInteger)length, allow to read a number of bytes equals to the given value – length – from the serial port;

– writeString:(NSString*)stringData, writes "stringData" content on the already initialized serial file;

– deviceList, return a connected devices list – actually, a list of UNFIX FILE paths realted to a serial device;

– saveValues, is related to the implementations of NSCoding protocol methods. They allow to store, in a persistent way, all the parameters related to the serial connection (devidePath and baudRate). The saved file will have an iOS tipical extension .plist which is always related to a file where can be saved standard iOS framework data types (Foundation) but also custom object, inside an XML formatted file. In our case this file is saved into: Documents/SerialPreferences.plist inside the Application Main Bundle.

3.3 MIFAREAdmin – Administrator Interface

The system administrator has a GUI written in python language, which provides all the tools necessary to manage the museum and learn about the works and visitor statistics.

The management operations include insertion, modification and deletion of artwork from the db.

The statistics on visitors are divided into two types, the people currently in the museum and the history of all the people who have visited it. The operation is equivalent for both categories. Choosing the category will be displayed a number of persons identified by the serial number of the device that uses. Choosing the person, data will be printed on the artworks displayed.

The statistics regarding artworks, shown who is the time spent by people to each work.

For each work, will be calculated the time on the individual, then, will be added to all visitors in order to create aggregate statistics. You can also view the graph with the statistics. For the generation of graphs was used graphics library *matplotlib* with *numpy* python extension mathematics.

The communication with the central database is via *http request*, and a simple parser modelled on the structures congenial to us processes responses.

```
conn = httplib.HTTPConnection("www.site.it")
conn.request("GET","/mifare/admin.php?req=
             art&val="+query1+"&col="+row)
r1 = conn.getresponse()
#print r1.status, r1.reason
data2 = r1.read()
#print data2
conn.close()
```

3.4 MIFAREServer – The connection to the db

Informations exchanged during all steps of the system require a single control and a centralized repository accessible both from mobile devices both from the administration. In addition, management of real-time events, such as the number of visitors of the artworks, can be optimally only with a mechanism to keep track of information throughout the implementation period, thus discarding the possibility of using temporary sessions.

The repository is loaded from a mysql database on a remote server and organized into two tables: Museum and People (figure 4).

The museum table is designed in a perspective of future developments by inserting fields geo_x geo_y, that will be used to set and change the

position of the individual tags, improving the management of localization.

The people table uses a pair of values as the primary key: id and location. The first is the UUID of the iPhone that connects to the database and informs the system of its position, stored precisely in the second field. This allows to have a unique record that maintains the information on the single visit to be able to calculate the time.

Field	Type
TagID	varchar(8)
Name	text
Author	text
Description	text
Movement	text
Period	text
Position	text
Visits	text
Image	text
Audio	text
time_visit	text
geo_x	int(2)
geo_y	int(2)

Field	Type
id	varchar(12)
position	varchar(8)
time	text
start	text
exit	tinyint(1)

Fig. 4. Museum table and People table

In the case where the user forgets to close the tab of the work, doing then increase dramatically the time of access, a timeout of 15 minutes has been established, after which the mobile device sends the signal to stop automatically.

The choosen programming language on server side is PHP, for its qualities of flexibility and richness of features. The files in question are two: *index.php* and *admin.php*.

The index page manages the connection with the iPhone, called it, by passing the HTTP request parameters that do the mysql query; the answers to the device are provided as XML pages specially formatted according to the parser built into the software iMIFARE.

```
if(isset($_GET['id'])){
$q="SELECT * FROM Museum WHERE TagID=".$_GET[id]."
 $id=mysql_query($q);
 if(mysql_num_rows($id)){
  while($val=mysql_fetch_array($id)){
   echo '
    <?xml version="1.0" encoding="UTF-8"?>
    <TAGS>
     <TAG>
      <id>'.$_GET['id'].'</id>
      <nome>'.$val['Nome'].'</nome>
      [...]
     </TAG>
    </TAGS>
   ';
  }
```

The administrator request, sent by the python interface, are managed by the admin page and the answers are written on HTML pages.

```
if($_GET['req']=='ins'){
 $colonne = $_GET['col'];
 $getval = str_replace("_", " ", $_GET['val']);
 $val = split("$", $getval);
 for($i=0;$i<9;$i++){
  $v=substr($val[$i],0,1);
  if($v==" ") $val[$i]=substr($val[$i],1,strlen($val[$i]));
  if($i==8) $valori .= "\"http://www.sito.ext/".$val[$i]."\",";
  else $valori .= "\"".$val[$i]."\",";
 }
 $valori = substr($valori,0,strlen($valori)-1);
 $x="INSERT INTO museum ($colonne) VALUES ($valori) ";
 $q=mysql_query($x) or print("Error message");
 if($q){
  print("Tag was written correctly");
 }
}
```

A Practical Application

Let's describe now a complete scenario: the user arrives at the museum and begins his tour. The first thing to do is select the application iMifare on his iPhone, wait for loading of the initial settings and the connection to the database.

From the main screen you can select a user guide, a window to the system settings, the map of the museum and the list of works (figure 6).

Fig. 6. iMifare museum map

From the map you can see the position of the works, indicated by a red pin. The user can choose to click one of these and see which artwork is in the room of his choice, and if necessary request the information on the latter, clicking the blue arrow side.

The user sees a list of artworks sorted by author, with a thumbnail and a small description (figure 7).

Fig. 7. Artworks list sorted by author

With a touch on the right arrow brings up a detail screen with complete description.

At the bottom of the screen an arrow is visible, that allows you to update the list of works.

The ordering of works can be changed by the user through the appropriate button, and the options available are: Author, Period, Movement. In this way you can create custom routes, selecting only the topics that interest you.

These operations are connected to a timer that periodically sends its data to the centralized system. In this way it is possible to calculate the statistics for the display of the works.

The entry of a user in the museum causes its registration in the database, and this offers many possibilities, in addition to simple statistic as the UUID is, by definition, unique:

– you can know the position of a user in the room, and thus follow the movement;

– it is possible to trace the person in the case where it is necessary, such as a theft or damage of the works.

IV. CONCLUSIONS

This paper presents a new automated system for tourist attractions, especially, museum. All the "MIFARE" system components are based on open standards and/or open languages, allowing everyone to use and test all the implemented features.

The solution is also strictly related to mobile device and smartphone technologies using a mobile phone as a User Interface to let the user interact in an easy and practical way with our Applications.

The System Architecture can be set up just by using a simple Router, creating a LAN inside the Museum Building or, better, connecting the whole Museum communication system to the Internet.

Least but not last, this work would also promote the adoption and use of the RFID technology in other fields than the common "Stores Application", which represent the first kind of RFID scenario nowadays.

REFERENCES

[1] RFID-Based Guide Gives Museum Visitors More Freedom:http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5735592

[2] A Systematic RFID Application Platform with Integration Capability for Tour and Exhibition: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6068613

[3] RFID: A Technical Overview and Its Application to the Enterprise: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1490473

[4] RFId system: http://etd.adm.unipi.it/theses/available/etd-09172007-215049/unrestricted/Alessio_Conti_Finale.pdf

[5] Arduino Uno: http://arduino.cc/en/Main/arduino Board Uno

[6] SM130 13.56 Hz RFID Mifare Read / Write Module DATA SHEET: http://sigbed.seas.upenn.edu/archives/2011-12/ Keynote. Pdf

[7] RFID Evaluation Shield - 13.56MHz http://www.sparkfun.com/products/10162

[8] RFID Tag - Transparent MIFARE 1K (13.56 MHz) http://www.sparkfun.com/products/10128

[9] iPodBreakOut: http://www.kineteka.com/PodBreakout-v1.aspx