A client-server framework for the design of geo-location based augmented reality applications

Nicola Capece, Roberto Agatiello, Ugo Erra
Dipartimento di Matematica, Informatica ed Economia
Università della Basilicata
Potenza, Italy
Email: nicola.capece@unibas.it, roberto.agatiello@unibas.it, ugo.erra@unibas.it

***Abstract*—We present a client-server framework for the development of mobile applications that use Augmented Reality (AR) to visualize geolocated data. Geo-information displays allow users to understand and respond effectively to the context in which the application is deployed. We provide a scalable and flexible architecture for the development and management of the client, the server and the data that are used by the applications. This architecture is based on the display of connected layers that represent structured information. The approach has been implemented in two case studies: the management of failures in electrical power lines, and to support hydrogeological monitoring.**

***Keywords*-augmented reality; location Based AR; client-server framework;**

I. Introduction

Augmented Reality (AR) refers to an altered reality in which perceived normal reality is overlaid with artificial/virtual sensory information. In more formal terms, we define AR as: *the overlaying of virtual elements generated by computer onto the visual perception of physical reality, through a camera*. AR technology allows the user to see the real world augmented by virtual objects [1]. In other words, AR provides an environment in which virtual and real objects coexist.

Commercially, AR is mainly found in mobile applications and, in recent years, has attracted wider interest from business. It is a natural way to explore 3D objects and data, because it brings virtual objects into the real world [2]. The fields of application are potentially endless, and include: the display of information; navigation in real world environments; advertising; art; and games [2].

Technological support for daily activities is a common feature of mobile devices. Apps are used to send reminders about activities or events, and software enables users to orient themselves in unknown places. However, classic navigation systems are not user-friendly. In particular, when it comes to how people are usually presented with directions, it is easier to visualize information in relation to the real environment, rather than trying to interpret a 2D map. For this reason, many applications are based on geo-localized data, and focus more on usability and intuitive interactivity [3].

There are several frameworks that support the development of AR applications, the best-known include Metaio [4] and Wikitude [5]. Metaio is a modular framework that includes: *(i)* an acquisition component; *(ii)* a sensor interface component; *(iii)* a rendering component; *(iv)* a tracking component; *(v)* and the Metaio SDK interface. Wikitude is a very important framework, which has had considerable success in the development of AR applications. It offers two kinds of interfaces: the first is native to mobile platforms; the second is platform independent, based on HTML5 and JavaScript. It supports the display of geo-referenced data and allows the visualization of 3D models, images. etc. using the tracking image.

In this paper, we focus on applications that display geo-localized data as virtual objects. These applications can run on devices equipped with a webcam, and sensors such as a gyroscope, a compass and a GPS. We develop a software architecture that can manage different layers. These layers contain information related to the real environment that is displayed by the device. Information that enhances reality is identified by a geo-marker. This is a real object that is recognized by the system, and information (defined using geographic coordinates) is overlaid onto it. Our goal is to visualise these layers through virtual objects overlaid onto the real environment. Each layer corresponds to an information set called a Point Of Interest (POI), which must include information about 2D/3D objects to be displayed as text, 3D models, texture, images, billboards, etc.

The remainder of this paper is structured as follows. Section II, provides an overview of related works. Section III provides some background information about the software tools used to develop the framework. Section IV presents detailed information about the framework. Section V presents the application of the framework in two case studies. We end with some final remarks and future directions for our research in Section VI.

II. Related Work

There are many frameworks that support the development of different AR applications. Here, we focus on Location-based AR (LBAR) applications, which are now widely used.

The popularity of LBAR has increased with improvements in smartphone functionality and sensor technology. Our system is mainly based on the use of GPS and a digital compass that, with the support of a Geographical Information System (GIS), provides information about the POI. The user points the device's webcam at the surrounding environment then, using the inbuilt sensors, the system displays information about the POI via an overlaid geospatial tag environment. The user clicks on the tags to view pictures, video and other media associated with the POI.

One of the main problems of developing LBAR applications is the organisation of, and interaction with geospatial tags. This is especially true where there are many POIs close to the user's location. Choi [6] provides a way to organize geospatial tags within a scene.

Geiger [7] describes a method for developing an engine for LBAR applications giving profound insights into the design and implementation of such an advanced mobile application. Deli [8] provides a method based on LBAR that enhances the user's view of their surroundings with information about the land parcels that lie within the visual field of the device. Their aim is to use AR technology for educational purposes, through the development of instructional applications. Their approach highlights another way to use the LBAR through the real-time construction of virtual objects on the scene, using the Web Map Service (WMS) specification defined by the Open Geospatial Consortium [9].

Other software, such as Junaio [10] shows virtual objects such as images and labels, and a radar supports the search for POIs in the users environment. This system is typically used for route planning. Images and videos can also be saved in a database for other applications (e.g. uploading and sharing on social media). Layar [11] allows developers to create various features. Users can associate text messages with AR images, make calls that interact with virtual tags, send email and plan routes.

III. Background

We propose an approach based on a client-server architecture where:

- The client interacts with a server application (Web Service) through requests made via Representational State Transfer (REST) messages.
- Once the request is accepted, the Web Service queries the database and sends the retrieved information to the client as a JavaScript Object Notation (JSON) message.
- The client interprets the response and renders the information on the screen as virtual objects, overlaying them onto the real environment being viewed.

We use several specialized frameworks for different tasks. The REST architecture is used for client-server exchanges. Our software architecture [12] uses the HTTP protocol to transmit data by defining architectural constraints, but not the implementation of components. Network resources are used to communicate network components that exchange resource representations. A representation is composed of: *(i)* a sequence of bytes (the content); *(ii)* metadata that describe the content; and *(iii)* metadata describing the same metadata (e.g. hash sums) [13].

A connector (a client or server) mediates communication between components. It enables the application to interact with a resource, given its identifier and the action to be taken. The application interprets the answer and determines the representation of the information (e.g. an XML document, JSON).

Components are identified by their roles within the application, and can be classified into: *(i)* user agents: these use a client connector to instantiate a request, then receive the response (e.g. a Web Browser); *(ii)* origin servers: these use a connector server that receives the request and provides the representation of its resources (e.g. Apache Tomcat); *(iii)* intermediate components: these can operate as a client and as a server and support the translation of both requests and responses (e.g. Gateway).

One of the advantages of the REST architecture is that it provides a uniform interface that enables client-server separation. The client handles the user's state and the application interface, rather than the server. Consequently, both the client and the server are completely independent of the technology. Client requests have all of the information necessary to respond to the request; the session state is maintained by the client and can be transferred to the server through other services.

We use the REST architecture to handle the exchange of messages, specifically the Restlet [14] framework, version 2.3. This framework is composed of two main parts: *(i)* Restlet API: a neutral API that supports the principles of the REST architecture, facilitating call management within the client; *(ii)* the Restlet Engine: the implementation of the Restlet API. For client-side development, Restlet can easily interact with remote resources through its HTTP connector. The system analyzes the representation of resources (in JSON format) and extracts information that has to be saved in the model's objects. For server side development, we built additional http connector, listening on port 7080.

To represent the resource obtained through REST we used the Metaio framework. In the context of our approach, the implementation is encapsulated and the developer does not need to know about the details of acquisition, rendering, sensing and tracking. Metaio uses standard graphics' libraries for three-dimensional scenes and rendering, in particular OpenGL ES API. This is a subset of the OpenGL graphics library [15] that was designed for embedded devices such as smartphones.

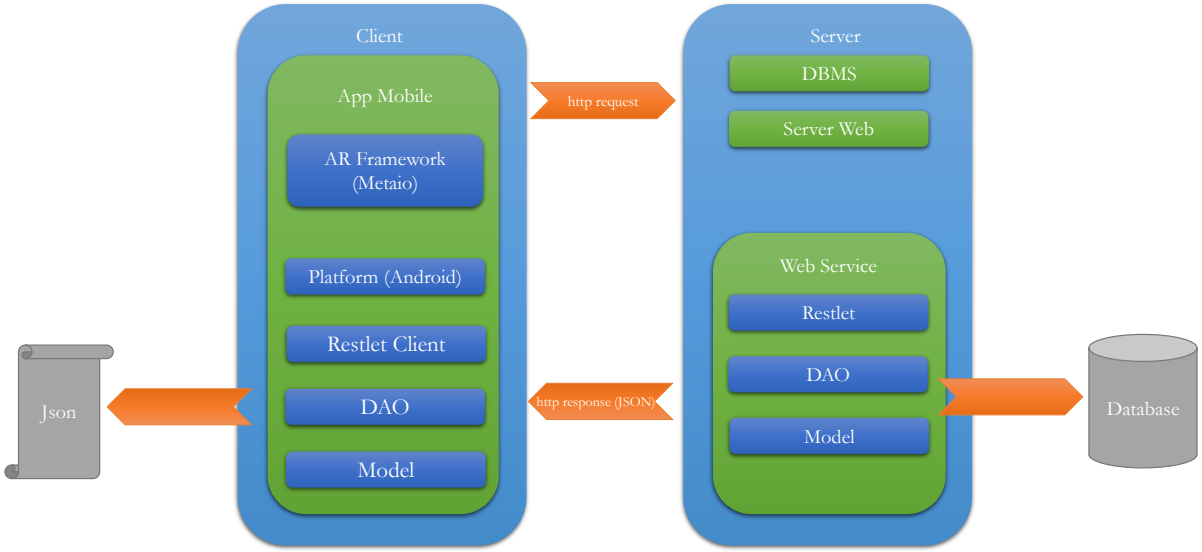

Figure 1. The architecture of the client-server framework.

IV. The Architecture

Our architecture can be used in different application contexts. Figure 1 shows the client-server framework:

- The client side allows the development of applications for mobile devices that display POIs using AR.
- The server side allows the development of Web applications (Web Services) that handle the requests from the client (Browser AR) and returns responses after querying a POI database.

The POI concept is central to our architecture. We need to be able to access information about POIs in situations where there is both good and poor network coverage. We use the concepts of online and offline modes to address these two cases. These three concepts are described below.

Point Of Interest (POI): A POI represents a specific point, geolocalized in the world, corresponding to locations such as a shop, a restaurant, a hotel, a road or a specific address. A POI can also be a location that is of particular interest to a specialised user, such as companies carrying out relief work, maintenance, etc. In this case the POI may be a critical area, or it may identify an at-risk area (e.g. from flooding). In our system, POIs are displayed on a mobile device using billboards that are overlaid onto a scene that is captured by the device's camera. The billboard is an icon that takes different shapes and colours based on the POI. Each billboard is associated with a label that displays information such as the distance between the device and the POI, etc.

Online Mode: In online mode the system queries a remote database (e.g. MySQL) directly, using the available network. The app sends a request to the web service and provides the user's location and the maximum range (in metres) that the user wants to see the POIs for. This distance serves as a filter to prevent the user from being overloaded by too many POIs. The filter is implemented by applying the Haversine formula [16].

It is also possible to apply additional filters (for example to focus on those POIs that are most relevant) by specifying the device's latitude and longitude, and a radius that represents the maximum distance to be used when locating POIs or, in the case the POI represents a status to monitor the precise condition of an object, a filter that shows only the most critical POIs (e.g. high temperature, water level dangerous).

The response is in JSON format. Whenever a user's location changes, the system sends a new request to the web service with the geographic coordinates of the current location of the device. Based on these parameters, the service returns all POIs that fall within the specified radius. When new POIs are found, the view is updated with the new information in real time. Additionally, as the user approaches a POI, the label displays the distance between it and the user, which is updated in real time. The web service receives the request and queries the database with the parameters given in the URL. The query is executed using a connector or drivers. Note that the database must initially contain at least one table recording POIs, and will eventually contain other tables relating to other types of information to be displayed.

The POI is characterized by three parameters: latitude, longitude and altitude. The database can be populated through the applications such as phpMyAdmin [17] for MySQL [18] databases as shown in Figure 2.

Offline Mode: In offline mode (e.g. when network coverage is poor) the system uses information that has previously been stored in a JSON file. The information to be displayed

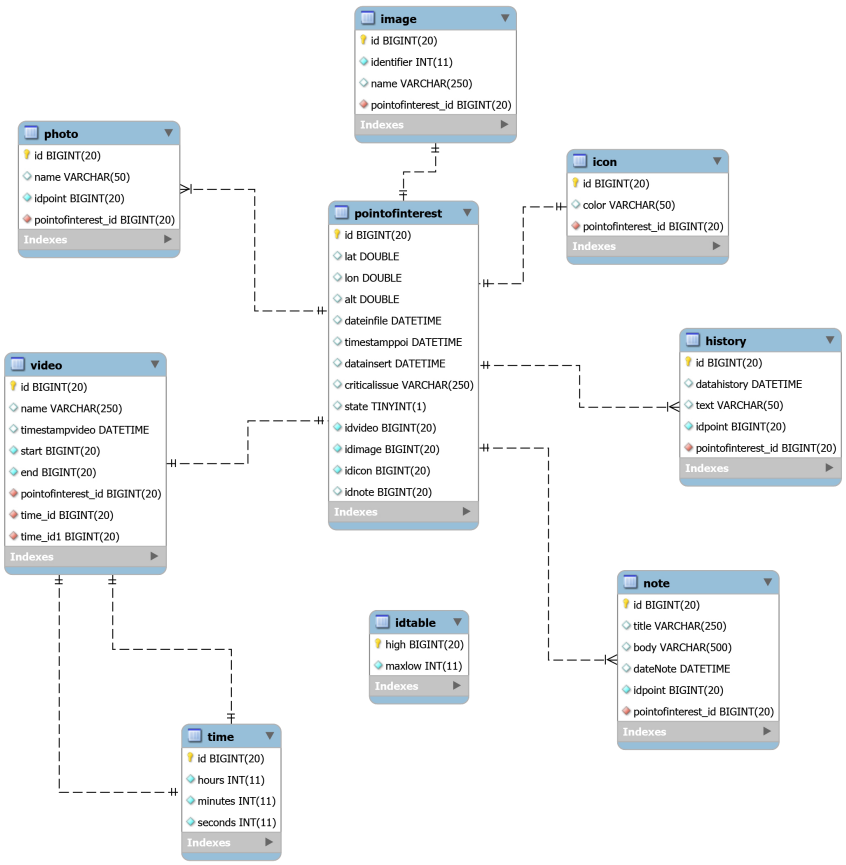

Figure 2. The database structure used to manage POIs.

is provided in advance. The JSON file is automatically generated and lists all POIs. The user can also choose what information to display according to their needs. This requires them to have previously downloaded POI information (when they had access to a mobile network). Once POIs have been selected and downloaded, they will also need to download additional media information (images, videos, etc.) associated with POIs. This data is saved locally in JSON format. The user can then view the available information without using the network and, when necessary, update the database by refreshing the information when a network becomes available.

V. Case studies

In this section, we describe two applications that were developed using our client-server framework.

The first application was designed to support the management of failures in electrical power lines. The idea was to provide information that is useful in locating high-voltage pylons needing maintenance. Such pylons are displayed in AR as billboards that are oriented towards them. The POI is the pylon, and information about the problem (e.g. an identification number or the date it was reported) and, most importantly, its location are is presented. Each POI can be associated with further information such images, videos and icons. The application was developed for the Android platform [19] using the Metaio SDK framework. Server side, we created a MySQL database that was managed using the phpMyAdmin application.

Figure 3. Electrical power lines AR application.

In Figure 3, each billboard has a label that summarises important information: the ID of the issue; and the distance from the users current location to the pylon. Metaio allowed us to deploy a radar display (top left) that shows the direction of the pylon and helps the use to rotate the device in its direction. The yellow circle represents the pylon's position and the blue circle represents the device. The application allows the user to interact with POIs by pressing on the billboards. A popup (Figure 4) shows the different operations that can performed: (*i*) View an image associated with the pylon, (*ii*) View a video associated with the pylon; (*iii*) View the pylon's history; (*iv*) Add notes.

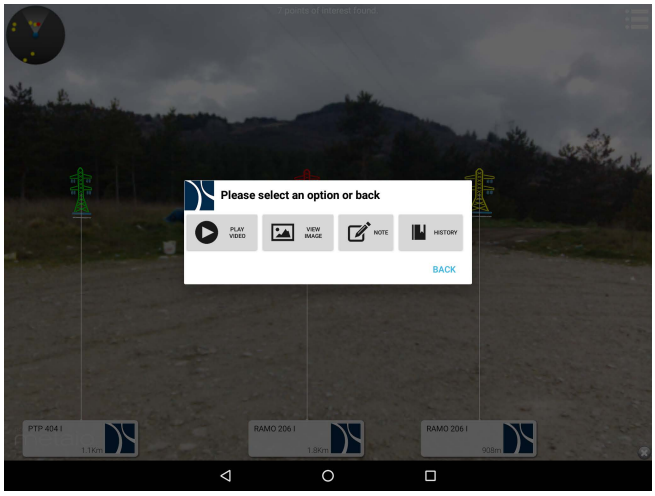

Figure 4. Options associated with the POI

Figure 5. The hydrometric AR application.

The second AR application was designed to support the collection and analysis of water level data that is measured by stations in southern Italy. The application can show the location of stations and their hydrometric levels. More importantly, it can graphically display daily hydrometric values in order to provide a quick preview of changes in water levels. The POIs in this context are hydrometric stations, and associated information relates to the water level, the date of the last reading, and a graph showing changes in water level.

As Figure 5 shows, the application also allows the user to also view static POI data, such as buildings, bridges and crossings as billboards. Whenever the water level changes, the color of the of billboard changes from red to green according to the status of the danger. Red is for severe risk of inundation and green is for low risk of inundation.

VI. Conclusions

The development of a client-server framework to manage and visualize geo information through AR evolved from the need for a flexible approach that could be adapted to different contexts. The proposed framework is generalised, and the infrastructure can provide solutions to problems from different domains [20]. Our case studies illustrate two issues: maanaging failures in electrical power lines and monitoring hydrogeological risks.

The main problem we faced was to develop a scalable software architecture with a clear division between client and server side technology. The server does not provide details of graphics, which can be customized independently according to the needs of the client. It only provides the POI information requested by the client, such as the distance between the client device and the POI, and the orientation of POIs.

The POI is an abstract concept in our architecture: for example, it can be a pylon, a tourist resort, or a measuring station. We intend to continue our work on the proposed infrastructure, notably by contextualizing the POI to the type of information the user would like to display. In this context, we are going to integrate the geographic information system inside our architecture. We are confident that this information system that integrates several types of geographic information can be easily brought in our architecture using the idea of the layers.

REFERENCES

[1] M. Hincapie, A. Caponio, H. Rios, and E. Mendivil, "An introduction to augmented reality with applications in aeronautical maintenance," in *Transparent Optical Networks (ICTON), 2011 13th International Conference on*, June 2011, pp. 1–4.

[2] V. Geroimenko, "Augmented reality technology and art: The analysis and visualization of evolving conceptual models," in *Information Visualisation (IV), 2012 16th International Conference on*, July 2012, pp. 445–453.

[3] J. Ma Luna, R. Hervás, J. Fontecha, and J. Bravo, *Ubiquitous Computing and Ambient Intelligence: 6th International Conference, UCAmI 2012, Vitoria-Gasteiz, Spain, December 3-5, 2012. Proceedings*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, ch. A Friendly Navigation-System Based on Points of Interest, Augmented Reality and Context-Awareness, pp. 137–144.

[4] "Metaio DOC doc description," http://dev.metaio.com/sdk/documentation/.

[5] "Wikitude DEV," http://www.wikitude.com/documentation/.

[6] J. Choi, B. Jang, and G. J. Kim, "Organizing and presenting geospatial tags in location-based augmented reality," *Personal and Ubiquitous Computing*, vol. 15, no. 6, pp. 641–647, 2010.

[7] P. Geiger, M. Schickler, R. Pryss, J. Schobel, and M. Reichert, "Location-based mobile augmented reality applications: Challenges, examples, lessons learned," in *10th Int'l Conference on Web Information Systems and Technologies (WEBIST 2014), Special Session on Business Apps*, April 2014, pp. 383–394.

[8] A. Deli, M. Domani, P. Vujevi, N. Drljevi, and I. Botiki, "Augeo: A geolocation-based augmented reality application for vocational geodesy education," in *ELMAR (ELMAR), 2014 56th International Symposium*, Sept 2014, pp. 1–4.

[9] "Open Geospatial Consortium," http://www.opengeospatial.org/.

[10] "Junaio, junaio description," https://my.metaio.com/dev/junaio/.

[11] "Layar, layar description," https://www.layar.com/.

[12] H. Li, "Restful web service frameworks in java," in *Signal Processing, Communications and Computing (ICSPCC), 2011 IEEE International Conference on*, Sept 2011, pp. 1–4.

[13] M. Jakl, "Rest representational state transfer," 2008.

[14] "Restlet," https://restlet.com/.

[15] "OpenGL," https://www.opengl.org/.

[16] J. E. Bell, S. E. Griffis, W. A. C. III, and J. A. Eberlan, "Location optimization of strategic alert sites for homeland defense," *Omega*, vol. 39, no. 2, pp. 151 – 158, 2011.

[17] "PhpMyAdmin," https://www.phpmyadmin.net/.

[18] "MySQL," http://www.mysql.com/.

[19] "Android DOC," http://developer.android.com/index.html.

[20] R. D. Chiara, V. D. Santo, U. Erra, and V. Scarano, "Real positioning in virtual environments using game engines," in *Eurographics Italian Chapter Conference 2007, Trento, Italy, 2007*, 2007, pp. 203–208.