A Service Oriented Mobile Augmented Reality Architecture for Media Content Visualization in Digital Heritage Experiences

By

Sasithorn Rattanarungrot

A thesis submitted in fulfillment of the requirements for the degree of

Doctor of Philosophy at the University of Sussex

School of Engineering and Informatics

Department of Informatics

University of Sussex

Brighton

BN1 9QT

April 2016

Declaration

The work described in this thesis, carried out in the school of Engineering and Informatics, I hereby declare that this thesis has not been and will not be submitted in whole or in part to another University for the award of any other degree.

Signed:__________________________

Sasithorn Rattanarungrot

Acknowledgements

This thesis has been accomplished by the supervision of Dr Martin White and Dr Paul Newbury. I really appreciate their time, support, and kindness. I also would like to thank Dr Ben Jackson who has given me some help and advice. Another group of people who have encouraged me during my PhD study is my colleagues at Walailak University, Thailand. I am grateful to have known these people. Nevertheless, I would like to thank the Royal Thai Government who supports me financially for the PhD study and the living in the United Kingdom.

I have been obliged and delightful to be a daughter of my parents (Boonsub Rattanarungrot, Pranee Rattanarungrot), a sister of my two brothers (Aroon Rattanarungrot, Arthid Rattanarungrot). I really appreciate their encouragement, love and understanding. Moreover, I would like to thank all of my lovely friends and family members who are always with me. Finally, I am really grateful to have known Chatsuda Knowthong, my dearest friend. I would like to thank her for moral support and good friendship.

UNIVERSITY OF SUSSEX

Sasithorn Rattanarungrot

Submitted for the degree of Doctor of Philosophy

A Service Oriented Mobile Augmented Reality Architecture for Media Content Visualization in Digital Heritage Experiences

Abstract

Mobile augmented reality has become an influential tool for digital content representation and visualization of media content in terms of enhancing users' experience and improving the adaptability and usability of typical augmented reality applications, such as in e-commerce shopping, virtual museum, or digital heritage scenarios. This research proposes a new Service Oriented Mobile AR Architecture called SOMARA, which includes a novel mobile AR client application.

SOMARA takes advantage the ability to integrate third party content through service-orientation. The SOMARA architecture enhances traditional standalone mobile AR applications with embedded media content by uniquely integrating a web service framework into an augmented reality client application to create more efficient and flexible mobile augmented reality applications that efficiently supports novel media content acquisition and visualization through appropriate access parameters.

The proposed architecture requires access to media content through specific media content service providers, e.g. a museum commissioning an augmented reality based museum interactive — predetermined media content, or any third party with their own service APIs, e.g. the Victoria and Albert Museum API — related external media content. This approach allows relevant third party media content to be 'mashed' via their public API with museums' augmented reality interactive's 'embedded' media content in the SOMARA mobile AR client. In this way novel mobile AR interactive applications, such as a museum augmented reality interactive, can be created based on particular museum environment scenarios that integrate a museum visitor's experience with the interactive's cultural objects.

Such experiences based on a SOMARA type museum augmented reality interactive can also be saved allowing visitors to take home their museum experience. SOMARA thus allows museum interactive experiences based on visualization of museums and third party media content physically located in the museum to be migrated to the visitor's home environment for further study, enjoyment and understanding. This unique feature, ability to effectively replay the experience at home, of the proposed system utilizes service-orientation to integrate third party media content, which is currently deficient from commercial augmented reality solutions.

List of Publications

Conference papers

1. S. Rattanarungrot, M. White, and P. Newbury, “A mobile service oriented multiple object tracing augmented reality architecture for education and learning experiences,” in *Proceedings of the International Conference on Mobile Learning*, IADIS Press, 2014, pp. 327–334.
2. S. Rattanarungrot, M. White, Z. Patoli, and T. Pascu, “The Application of Augmented Reality for Reanimating Cultural Heritage,” in *Virtual, Augmented and Mixed Reality. Applications of Virtual and Augmented Reality SE - 8*, vol. 8526, R. Shumaker and S. Lackey, Eds. Springer International Publishing, 2014, pp. 85–95.
3. S. Rattanarungrot and M. White, “A service-oriented mobile augmented reality architecture for personalized museum environments,” *Virtual Systems & Multimedia (VSMM), 2014 International Conference on*. pp. 277–284, 2014.
4. S. Rattanarungrot, M. White, and B. Jackson, “The Application of Service Orientation on a Mobile AR Platform - A Museum Scenario,” in *International Congress on Digital Heritage - Theme 2 - Computer Graphics And Interaction*, 2015.

Glossary

2D	Two Dimensions
3D	Three Dimensions
6DOF	Six Degrees of Freedom
AR	Augmented Reality
ARC 3D	Automatic Reconstruction Cloud Three Dimensions
API	Application Programming Interface
AREL	Augmented Reality Experience Language
AR-PDA	Augmented Reality Personal Digital Assistant
BBF	Best-Bin-First
BIM_WPVS	Building Information Models-Web Perspective View Service
CAD	Computer-Aided Design
CLONALG	Clonal Selection Algorithm
CV	Computer Vision
FERN	Fast Keypoint Recognition Algorithm
GIS	Geographical Information Systems
GPS	Global Positioning System
HTML	Hypertext
HTTP	Hypertext Transfer Protocol
IDE	Integrated Development Environment
JSON	JavaScript Object Notation
LM	Levenberg-Marquardt
MCaaS	Media Content as a Service
MLA	Museum Learning Activity
NFT	Natural Feature Tracking
OGC	Open Geospatial Consortium
OS	Operating System
PSO	Particle Swarm Optimization

RANSAC	Random Sample Consensus
RCH	Reanimating Cultural Heritage
REST	Representational State Transfer
RFID	Radio-Frequency Identification
RGB-D	Red Green Blue Depth
RSS	Rich Site Summary
SOA	Service Oriented Architecture
SDK	System Development Kit
SIFT	Scale Invariant Feature Transform
SLAM	Simultaneous Localization and Mapping
SOAP	Simple Object Access Protocol
SOMARA	Service Oriented Mobile Augmented Reality Architecture
UDDI	Universal Description, Discovery, and Integration
UI	User Interface
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
VM	Virtual Museum
VR	Virtual Reality
WSDL	Web Services Description Language
XML	Extensible Markup Language

Table of Content

Table of Figures

Chapter I

1 Introduction

Cultural heritage has been continuously improved so much in terms of assisted technologies, content representation and innovative applications supporting cultural heritage study, learning and digital content consumption. The European Work Program 2016-2017 has stated technologies that support access and utilization of digital cultural heritage in Virtual Museum (VM) that allow visitors to access content before, during and after a visit. Technologies that effectively enhance VM are Virtual Reality (VR) and Augmented Reality (AR) in terms of digital content representation and interpretation that can be applied in physical museum, online or heritage sites [1].

Mobile AR has become an influential tool for digital content representation and visualization of media content in terms of enhancing users' experience and improving the adaptability and usability of typical AR applications, such as in e-commerce shopping, virtual museum or digital heritage scenarios. This research proposes a new Service Oriented Mobile AR Architecture called SOMARA, which includes a novel mobile AR client application, which takes advantage the ability to integrate third party content through service-orientation (e.g. Media Content as a Service — MCaaS).

Recent evolutions in mobile technologies such as screen resolution, processing speed, embedded cameras and mobile network bandwidth have continuously led to an enormous improvement in mobile applications and implementations. Furthermore, advanced developments in mobile graphics processors, application programming interfaces (APIs) and displays have increased the processing potential in real time graphics tasks for instances of 3D rendering, visualization and interaction. Because of these advanced developments, many mobile applications today can provide graphic features and content, which hitherto where mostly found in desktop graphical user interface or in 3D rendering found in game development approaches.

Mobility, as in the ability to take application mobile or a users' penchant to use applications in a mobile context, has strongly influenced both the quality and performance of mobile technologies and applications in terms of greater potential for use of hardware (e.g. mobile devices) and development software (mobile applications), the quality of a mobile or wireless network, the scalability in application developments and service providers and finally, the usability and ubiquity of mobile services. Most mobile devices have a network connection module so mobile applications can be efficiently implemented in a client-server framework. This mobile network connectivity facilitates solutions for remote rendering on a mobile device. The most computational operations are processed on servers, which then send the output to mobile clients for visualization.

Nevertheless, cloud computing has currently been implemented in many client-server applications such as web service-based mobile applications [2] and service-oriented cloud computing [3]. Cloud computing is an enormous challenge adapting the traditional client-server architecture to cloud-based infrastructure on the Internet. The infrastructure relies on clients and services on the cloud computing architecture such as processing services and data services. For example, rendering 3D graphics on a cloud-based service infrastructure extremely improves the efficiency of the real-time rendering of 3D content [4][5]. Some 3D rendering services on the cloud infrastructure services are ArcGIS for Server [6] and 123D Catch on Autodesk® Cloud [7].

Advanced technologies utilized in hardware such as powerful mobile processors [8][9] and high quality smartphone cameras [10], software such as Swift [11] , hybrid and HTML5 development tools [12] combined with mobile network approaches such as cloud networking [13] and advanced mobile networks [14] for mobile computing that are now available on many platforms such as smart phones and tablet PCs have enabled mobile 3D computer graphics to be a part of an evolution and innovation in mobile applications. In the context of this thesis advanced interactive graphics development software is of particular interest. Currently, there are two state-of-the-art mobile graphics software architectures: OpenGL ES and M3G that both are configured as APIs for C/C++ and Java programming respectively [15] [16] that are very useful in developing mobile AR applications.

The following sub-sections discuss briefly the key concepts critical to the development of novel mobile AR applications, such as SOMARA, AR, mobile AR, MCaaS, museum environment scenarios, closed versus open AR platforms. These key concepts lead toward a consideration of the research question, potential solutions and a set of technology objectives. This chapter closes with a statement of the research contribution and organization of the thesis.

1.1 Augmented Reality

AR has recently been an influential technology utilized to efficiently enhance users' perception through computer-generated media content overlaid on real physical environments [17][18] . The types of media content that can be visualized on AR scenes are 3D models, labels, text, images, videos, etc. [19]. AR has become a powerful tool in many scenarios designed for particular approaches such as advertising, museum experiences, e-commerce, education and many other scenarios. In addition, AR applications are currently found in many platforms such as hand-held devices, PCs, wearable devices and even specially designed monitors and projectors for museum or advertising approaches [20][17][18]. In the context of this thesis the application scenarios discussed will focus on museum (digital heritage) experiences as described in the author's papers [21][22], but SOMARA could equally apply to other AR based interactive scenarios in different domains as described in the author's first paper [23].

AR application development requires important functions or AR tasks, including tracking, recognition and visualization, to enable AR applications to work simultaneously with cameras and computer screens. Traditional AR tracking is able to recognize markers and then overlay relevant media content on top of the markers. Marker AR is generally found in various scenarios, however using markers still has some restrictions in terms of flexibility and usability of AR application utilization. Applying marker-less AR is a solution that improves the functionality of AR applications by enable the applications to track and recognize marker-less such as images or 3D objects and then augment media content on the marker-less environment. This marker-less approach remarkably improves the implementation and experience of AR in various scenarios [24][19][25].

While AR scenarios can affect peoples' activities and lifestyles in positive ways, implementing mobile AR can prove to be a particular technological challenge.

Nevertheless, mobile AR has become the most recent trend in indoor and outdoor scenarios such as location-based AR, gaming or potential graphics applications, which allow users to experience static or interactive content on the real scene. Mobile AR systems require a tracking module including marker, marker-less and Global Positioning System (GPS) in order to recognize reference objects or identify current location of a user and the system will then augment related media content of a tracked object or user's current location [26].

Mobility is a powerful and influential structural concept that has been found in mobile technology and platforms including mobile devices, wireless networks, pervasive computing and ubiquitous networks. Current AR systems on mobile platforms enable mobile users to participate in virtual content and services at anytime and anywhere. Mobile devices are available on small and handheld devices (smartphone, tablets, etc.) that are lightweight and much more convenient to be carried. With such devices, wireless or mobile network connection will enable mobile devices to automatically connect to the Internet everywhere. A typical mobile's advanced hardware and software features can process interactive 3D graphic in real time enabling good human-computer interaction to be created. Because of these qualifications, mobile devices have become powerful tools that people can access and also utilize services and applications easily in the progressive and interesting ways. Mobile AR will potentially enhance users' new experiences in visualization and interaction with 3D models on mobile platforms by using embedded cameras, displays and graphical user interfaces — the basic 3 components needed for mobile AR scenarios embedded in a single mobile device.

Mobile AR has been implemented in various innovative applications such as gaming, shopping guides, advertising, edutainment, travel guides, museum guides, medical visualization, etc. However, these AR applications often provide limited flexibility because their media content are usually embedded within the application in advanced and they rely on marker-based approaches [27] [15]. Therefore, current AR applications can be enhanced by improving visualization, tracking, recognition, interaction, displays and user interface design of such AR applications both within the real and virtual environment.

AR applications work by superimposing or combining interactive virtual objects with real world environments in real time, and in this way AR application providers exploit software and hardware video (camera and tracking) techniques to represent computer generated models together with real objects to create visualization services based on interactive graphics representations. In addition, users can view and interact with virtual objects in a real environment through the AR application interface or through virtual interaction techniques such as hand tracking [28] or mixed-reality interfaces [29]. In such AR systems, cameras, displays and graphics libraries are currently the most important modules on mobile devices for implementing computer vision techniques such as 3D object tracking and recognition on AR environment [27]. This thesis proposes to add a new service-orientation module to this AR architecture to enable a new generation of mobile AR applications that can integrate MCaaS.

1.2 Media Content as a Service

MCaaS in the context of this thesis exploits the well-known Service-Oriented Architecture (SOA) approach, which is a web 2.0 technology that has been empowering sharing and acquiring digital content, data and services over open and interoperable networks. SOA and APIs often go hand-in-hand to provide flexible and efficient access for end-users to open data (e.g. media content) that is consumed in web applications. Currently, there are many open and interoperable networks and providers on various platforms, which interactively offer versatile data and content to other applications.

Open and interoperable network applications that consume or share data via service-orientation are usually based on web service architectures. They make data accessible over the Internet, perhaps as a mobile network, and are often configured client-server applications exploiting a private or public API. Examples of web service providers are Web Map Services, mash-up services, geospatial and social networking data [30][31][32].

This thesis investigates how to efficiently exploit open and interoperable networks (web services and APIs) to provide access to digital media content for mobile AR environments (using museum scenarios as an example). Through a web service approach, AR based media content are presented as virtual media or digital content on see-through AR browsers (on a mobile tablet or smartphone), which can be used instead of web browsers in outdoor AR environments or real scenes.

This SOA approach supports very well a client-server scheme over a mobile/wireless network. Thus, SOA and client-server architectures are well suited to developing mobile AR architectures, such as SOMARA. Additionally, integrating media content via an API allows the AR application to obtain more associated valuable content and significantly increase the usability and functionality of the proposed mobile AR application. So, a key features of SOMARA proposed in this thesis includes the integration of a web service framework into a mobile AR application.

This feature could be extensively implemented in indoor or outdoor AR scenarios, for which the AR browser is an application based on a web service framework to show media content in a real environment. The mobile AR client proposed in this thesis offers a key advantage (over current museum based AR systems) from being deployed on a service-oriented architecture. That is, it provides third party open services to access related digital media content from third party digital content providers [33][34].

1.3 Museum Environment Scenarios

Generally, museums basically allow visitors to physically view cultural objects, read details on small cards and take away some leaflets. Currently, some museums also provide a website that also exhibits digital cultural content and enable users to search catalogues online. Thus, the museum learning activities (MLAs) can be now performed online anytime and anywhere inside or outside the museum.

For the enjoyment of the casual or professional visitors, many museums have integrated technologies into their ‘collections environments’ (e.g. online or in a physical display) such as VR, AR and mobile technologies in order to revolutionize cultural object exhibits as well as enhance visitors’ experiences in participating the learning activities offered by the museum itself [35][36][37][38]. The next generation of museum exhibits could effectively be organized as AR exhibits on mobile platform exploiting SOA in order to create a tool or application on mobile devices that support effective learning scenarios where visitors will have opportunities to perform cultural objects study or view virtual exhibitions through a mobile device. These novel mobile AR applications will then be able to support content acquisition where associated cultural media content from any open services or repositories can be requested and the content will be presented to visitors on the AR environment.

Photogrammetry or image-based reconstruction is a technique that usefully supports content acquisition that could be beneficially applied in MLAs where visitors are able to obtain a 3D model of a preferred cultural object and utilize the acquired model in mobile AR environments. The photogrammetry could be integrated into the novel mobile AR applications in order to enhance typical museum learning where visitors are not allowed to take photographs or take away anything related to exhibited objects. The photogrammetry and its features are discussed further in section 1.3.2, and 2.6.

Additionally, the proposed mobile AR platform also supports multiple object tracking that enable the novel mobile AR applications to track and recognize several targeted objects individually. Multiple object tracking allows providers to create an AR exhibition of a group of cultural artifacts composed of reference cultural objects that will be tracked and related content of each object that will be visualized on top of the real environment. The multiple object tracking is discussed further in section 1.3.1, and Appendix F.

1.3.1 Multiple object tracking

Object tracking is one of the AR tasks that mobile AR applications firstly perform through the embedded-camera of a mobile device in order to detect and then recognize a reference object, which could be a marker or marker-less. Typical indoor mobile AR applications are particularly designed to track one single object and augment a related content on top of the real scene. For example, a mobile AR game allows users to download a marker or an image from which the application will augments an interactive game and mobile users can then have interaction with the active content on the screen. In order to create more functional and adaptable mobile AR applications, multiple object tracking could be another option in order to provide object augmentation to various reference objects in the same environment or scenario. This feature usefully enhances the concept of mobile AR in museum or learning scenarios that support content utilization in rich media environments where users are able to track a selection of reference objects and visualize associated content of each tracked object. In addition, mobile users are able to participate in the AR environments where they can create personal AR environments from the visualized content of preferred objects.

Multiple object tracking also enhances more flexible AR applications which is another level of improving the usability of mobile AR applications and providing various digital media content augmented from physical objects or images in AR environments. The new mobile AR applications can offer a broad range of reference objects and multiple related media content particularly specified for each object. The multiple object tracking can generate richer learning environments, which can be combined to other obtained media content, to mobile users so that they can efficiently view and utilize media content in AR environments. Typical mobile AR SDKs provide single object tracking and it can be improved to multiple object tracking by creating a tracking and content configuration file for reference objects. Thus, performing multiple object tracking and media content augmentation requires more preparation and processing compared to single object tracking.

1.3.2 Photogrammetry to acquire 3D content

Photogrammetry is an advanced technique for reconstructing 3D virtual objects from original objects. 3D model reconstruction tasks are mainly composed of matching and merging multiple images [39]. Photogrammetry requires photographs of a targeted object in order to create a virtual 3D model from the prepared photographs and the final outcome will be given away to the users. Image-based reconstruction or photogrammetry could be effectively used as a tool to perform 3D content acquisition in the novel mobile AR applications beneficially applied in MLA scenarios. Providing photogrammetry services to visitors is another technological strategy for physical museums that allow visitors to participate in selecting preferred objects, sending photogrammetry service requests and taking away obtained 3D models that can be done on a mobile device and the novel mobile AR application based on SOMARA. Photogrammetry applications currently are based on client-server architecture where users are able to upload photographs of a target objects to the server by using an application client on mobile devices, PCs or a web-based application.

Another platform of photogrammetry applications is the open photogrammetry services which expose service APIs and allow any applications to create connections with the services through a web service framework. Photogrammetry web services can then be integrated into mobile AR applications and the services can be requested directly from the client-side applications. As the result, the outcome of the service, which is virtual 3D models, is able to be instantly visualized in AR environments and taken away for presenting in other situations.

1.4 Closed AR Architectures

Most current mobile indoor and outdoor AR applications nowadays are implemented as ‘standalone’ or ‘closed platforms’ that provide users with a limited amount of embedded data or content on top of the real scene [5][40]. In this context a standalone platform means a single AR application not connected to any form of dynamic update, but can connect to the app store for a new version. On the other hand, a closed platform means that the AR application can be dynamically updated, for example, with commercial advertising, usually on a client-server framework. However, neither solution offer a communication channel to download or obtain dynamic content from other third party media that are related to the actual AR application’s media content in real-time. For example, if this was a museum app displaying cultural objects there is no way to add or integrate third party digital cultural object media content. This means the AR application is fixed and inflexible, i.e. cannot be dynamically updated with ‘meaningful’ media content. Therefore, these AR applications have no ability to provide content sharing between mobile end-users (e.g. museum visitors) and third party AR data providers because of the closed platform approach. Therefore, obtaining new content generally relies on application providers or application stores to update the AR applications.

For example, commercial outdoor AR applications such as Aurasma [41] and Layar [42] offer an interface or tool to create personal AR environments and upload media content to their server for use in the AR application. These tools then allocate an account to each individual mobile user and a channel to an AR environment so that they can visualize their own AR environments. However, dynamic advertising does take place in that commercial organizations (e.g. KFC, Tesco, whoever) can provide media content that are also displayed on the AR application (part of the revenue model for Aurasma, etc.). However, because the AR applications themselves are implemented on a closed platform where only their members (in thus case the application developer and any commercial advertiser) can place their preferred content (the developer, for example, could be a museum, and they also allow through the tool a channel for advertising media content that are displayed with their cultural content) on the AR environment, they are effectively inflexible and not truly dynamic, because they cannot exploit user-generated content.

Figure 1.1 illustrates the structure of typical AR applications on a closed platform and the process of updating the content that mobile users can only be done through the application store when the providers have launched an updated version of the application.

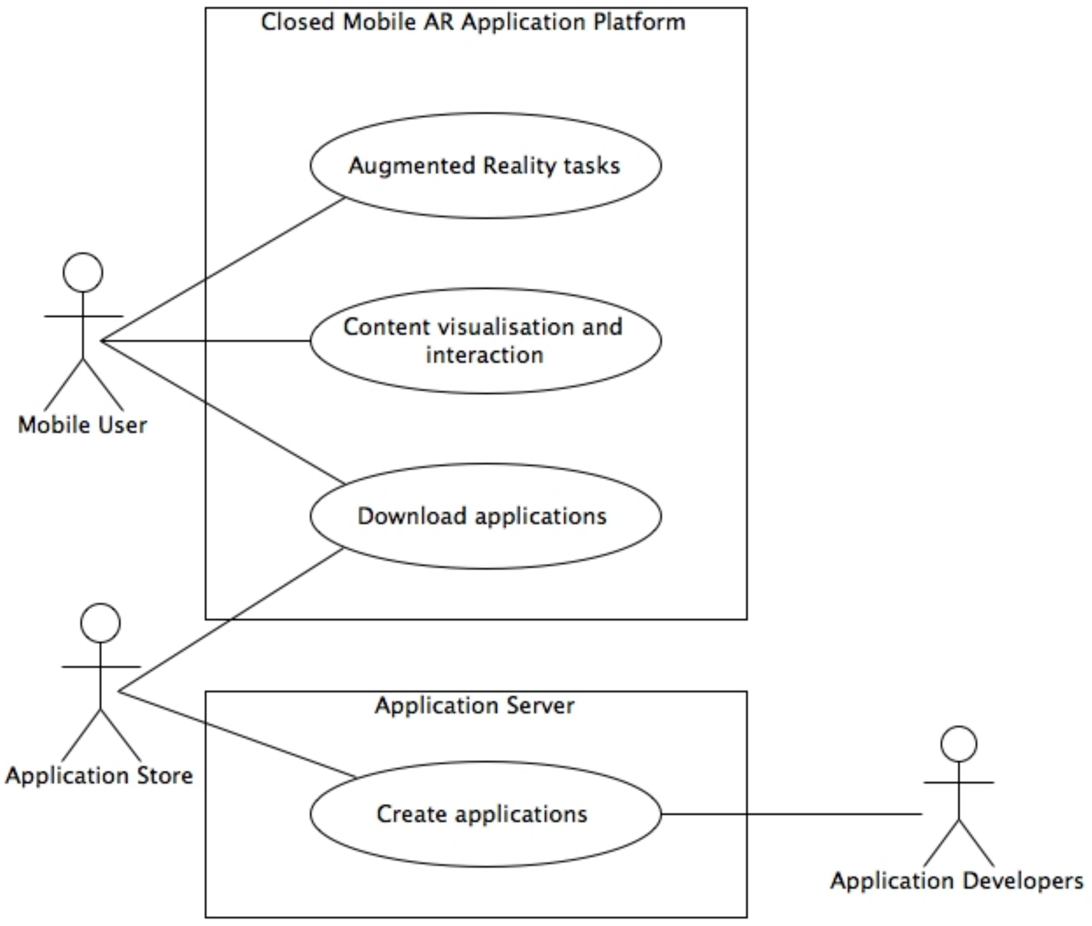


Figure 1.1 Closed mobile AR platform

AR applications lend themselves to the display of 3D objects. However for any mobile 3D graphic application, 3D virtual models still have to be created and designed in desktop computers and then transferred to mobile devices for running or rendering on applications just as they do for mobile games or other interactive media applications (e.g. web 3D apps). Although new generation mobile devices can generate good performance 3D graphics content, some complicated rendering tasks still require more processing power such as digital cultural heritage scenes, 3D virtual cities or complicated 3D models. Therefore, processing complicated tasks such as image-based reconstruction or 3D photogrammetry exploiting multiple images matching and 3D model building cannot be completely done on mobile devices because of the mobile device's limited resources. Unfortunately, closed architectures such as that described above preclude the ability to access photogrammetry (and other media) services to alleviate 3D content generation.

1.5 Open AR Architectures

In contrast, an open AR platform (open architecture) exploiting SOA, as illustrated in Figure 1.2, offers many more advantages, such as the ability to integrate third party or user-generated content. SOA is an open architecture that enables any application to participate with an open network where valuable content and services from potential providers can be accessed and requested. Thus, in an open architecture mobile AR applications are able to exploit a web service framework in order to create applications on an open platform, rather than a closed platform.

There are many advantages to utilizing an open architecture (or open platform). For example, it is relatively easy to integrate a web service framework (e.g. based on REST) into a mobile AR platform in order to create a novel open mobile AR application, as proposed in this thesis. Open architectures are able to apply service API calls, create enhanced interactions (between embedded and third party media content), request both local (e.g. British Museum AR exhibition content) and third party (e.g. Victoria and Albert Museum) media content and integrate them into the same mobile AR client application, and also share the results with other potential applications or service providers.

Additionally, mobile AR clients are capable of applying mash-up services, again through third party APIs, by requesting and presenting value-added media content such as geolocation and social networking data on AR or mixed reality environments. For example, an object from the British Museum (Mende Mask) and another object from the Victoria and Albert Museum (another Mende Mask) collocated in digital space (i.e. in the mobile client AR application) can be integrated with a Google map showing their shared location of origin (Sierra Leone).

The structure of SOMARA from a use case perspective is illustrated in Figure 1.2. Here, the critical components that comprise the open architecture or platform are composed of a mobile AR client, web service framework, and content or service providers. The mobile user is typically a museum visitor (or a shopper in a retail scenario) browsing augmented content.

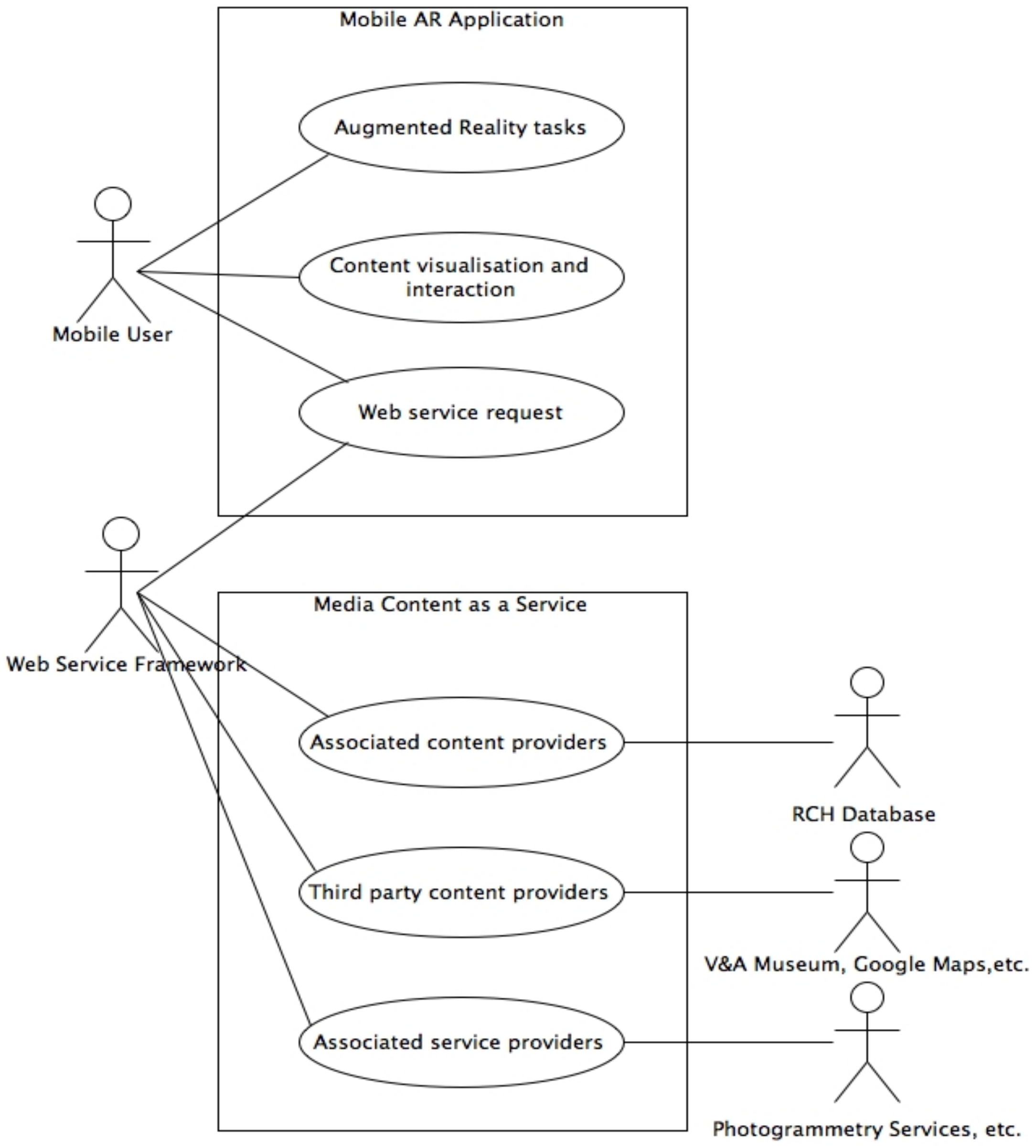

Mobile AR Application
Augmented Reality tasks
Content visualisation and interaction
Web service request
Mobile User
Web Service Framework
Media Content as a Service
Associated content providers
Third party content providers
Associated service providers
RCH Database
V&A Museum, Google Maps,etc.
Photogrammetry Services, etc.

Figure 1.2 Service Oriented Mobile AR Architecture

The mobile AR client performs AR tasks that support the interaction between a user and the novel mobile AR application in order to view augmented content of a targeted object on the screen. This new type of open mobile AR application effectively supports content acquisition where existing media content on the real scene can be augmented with media content from third party sources. Because, an open architecture can consume related third party media content, it will enhance the end-user experience further than is normally associated with closed platform AR — because there are richer media content in the AR environment that could lead to a greater understanding of the objects being studied.

Another valuable service that could be applied to a mobile AR platform is 3D model building from photographs or photogrammetry — alluded to in sections 1.3.2 and discussed further in section 2.6. Photogrammetry is an image-based 3D reconstruction method that requires high performance machines and plenty of resources to perform complex modeling, rendering and reconstruction tasks. At the moment, there are some closed photogrammetry applications that allow users to use their application tools for uploading a stream of photographs of a target object and then the systems will create a 3D virtual model and send it back to the user. This application will help users acquire 3D virtual models and utilize such models in their applications with time and cost saving. Typical examples of the photogrammetry applications are 123D Catch [7] and Autodesk ReCap [43]. Also, there are several potential European consortiums working on the concept of mobile photogrammetry for cultural heritage applications, in particular exploiting RGB-D cameras [44].

Another key feature required for enhancing mobile AR applications beyond so called marker-based AR is marker-less object tracking used to track and recognize physical targeted reference objects. This is a necessary requirement to build 'usable' AR applications such as those needed in shopping, museums and other environments where an end-user has the need to visualize associated media content collocated with the physical object (e.g. a museum artifact or a product in a shop, etc.). In this context, this thesis presents a mobile AR application based on marker-less AR by enhancing the Metaio AR SDK [45] i.e. with multiple object tracking, recognition and content visualization that demonstrates the efficacy of a new service-orientation approach [21][22][23][46].

Associated or related content can now be revealed on the real scene by augmenting the physical objects. In the mobile AR application proposed in this thesis, the tracking module is designed to perform marker-less tracking, which requires 3D object tracking (and image tracking) so that the system can recognize physical objects. Moreover, the proposed mobile AR system can augment various content of one reference object — as mentioned, this will lead to a richer AR environment in terms of media objects associated with the reference object. That is, multiple objects tracking has the potential to greatly enhance the interpretation of mobile AR scenarios and their environments where mobile users can obviously view a greater variety of media content from a reference object on the screen. Mobile AR applications can also offer some features for the users to manage and utilize those revealed content, e.g. saving an AR scenario for future use.

1.6 Research Question and Potential Benefits

Factors introduced above concerning open and closed AR architectures, scenarios of use, etc. lead to the consideration of an overarching research question posed in this thesis. This research question further leads towards consideration of potential benefits that will drive the design and development of the SOMARA framework in which key technology components (SOA, multiple object tracking, content acquisition and utilization) are implemented to demonstrate the efficacy of an open AR architecture.

Overall Research Question:

- How can a typical mobile AR application based on a closed architecture (or closed platform) be enhanced through service-orientation to create a more flexible open architecture that can acquire valuable MCaaS from participating open sources for museum (and other) interactive experiences?

This thesis proposes potential benefits to this research question**: Service Oriented Architecture on a Mobile AR platform (SOMARA)** that

- **Exploits an open architecture through service-orientation**: SOA is able to enhance typical mobile AR applications in order to obtain valuable associated content from participating open providers and extensively increase the adaptability and functionality of typical mobile AR applications on closed platforms (closed architecture). Therefore, integrating a web service framework into a closed mobile AR system efficiently improves the content acquisition and utilization of mobile AR system [40] [34] [47] [48].
- **Integrates a web services framework into the mobile AR client**: SOMARA is a unique implementation of a SOA on mobile AR platform where a web service framework is integrated into the AR application in order to perform interoperability tasks to acquire valuable media content that can be visualized on

the real scene and enhance the AR environment. The proposed service-oriented mobile AR application is composed of a mobile client, web service framework, and service providers.

- **Exploits APIs to access Media Content as a Service**: SOMARA's web service framework accesses media content via a set of APIs including an API developed to access the Reanimating Cultural Heritage (RCH) digital archive as a test bed. The mobile AR client also integrates other existing service APIs from third party content providers, e.g. Victoria and Albert Museum, in order to verify a service-orientation approach. Web service responses are processed and the obtained content are presented on top of a targeted object in AR environments.
- **Exploits APIs to access services**: Another benefit from working simultaneously on a web service framework and exploiting an open platform is that of obtaining value-added content from open services or third party such as photogrammetry or image-based reconstruction, which support 3D model acquisition from photographs. In addition, the mobile AR system also effectively support content utilization feature where mobile users are able to create personal AR environments by selecting preferred content on the active scene and save it for reviewing in other situations. This feature enhances the learning experiences and interpretation by transforming typical data or metadata into rich media content on AR environments. Another useful service in proposed museum learning scenarios is Google Maps where the user may wish to see a location origin of a cultural artifact. Google Maps APIs are applied by the mobile AR client that a visitor is able to see locations origin of a collection of preferred cultural artifacts saved from the museum on the map.
- **Implements a web service based museum scenario:** The application of SOMARA is validated through the implementation of a prototype mobile AR museum interactive application that integrates multiple object tracking. Multiple object tracking is required in the novel service-oriented mobile AR application in order to create a series of reference objects augmentation that expand the AR content representation and enrich the AR environment. The prototype allows MLA scenarios to be implemented where the mobile AR application is developed on a web service client framework [46][21] and mobile platform that related content of cultural objects are visualized on the mobile

screen. The novel mobile client developed in this prototype beneficially supports two related modes of use: museum-based and home-based AR visualizations.

1.7 Contribution to Knowledge

This new concept of developing an open AR architecture on mobile platform by implementing SOA will provide the adaptability and improvement of typical closed mobile AR applications that limit the exploitation of AR features and content acquisition that mobile AR applications should be able to obtain valuable media content from other sources such as third party. In order to prove the proposed concept, it requires a novel open mobile AR architecture and a mobile AR client that is applied to MLA scenarios where museum exhibits is transformed into media content on the real scene and it can be viewed through the mobile screen. The novel mobile AR application augments cultural objects by presenting related content in AR environments as well as requesting associated content and services from other participating museum content providers or third party. The following is the contribution to knowledge of this research, which is the accomplishment of developing SOA on mobile AR systems.

- A Service Oriented Mobile AR Architecture to support different mobile AR scenarios [23] — Conceptual Architecture
    - A new Service Oriented Mobile AR Architecture or SOMARA that is the development of a mobile AR framework based on SOA. A web service framework is applied into the architecture in order to create a mobile AR client that can access open service providers such as third party or local service providers and consume MCaaS. The connections allow the mobile AR client to send requests and receive responses, which is media content that will be visualized in AR environments.
- A prototype software framework based on iOS to demonstrate a SOMARA type application [21]— Specific Software Framework
    - A novel mobile AR software framework on iOS platform is developed in order to support the concept of SOMARA. A SOMARA application is built to validate the architectural concept and software framework. The SOMARA framework requires the embedded-camera of a mobile device, AR SDK and mobile/wireless connection in order to perform object tracking and interoperability tasks respectively. The SOMARA-based

    - application enables mobile users to interact with visualized media content and the application itself through the touch-screen mobile interface.

- A multiple object tracking algorithm exploiting the Metaio AR SDK to support [22] — Specific Algorithm
    - The Metaio AR SDK is applied as a framework performing AR tasks composed of object tracking, object recognition and content visualization. The Metaio AR SDK offers only single object tracking and single media content visualization. In order to create a mobile AR client that support rich media content augmented from different reference objects, the original object tracking has been extended in a unique way to afford multiple object tracking.
- Service-oriented content acquisition and utilization [46] — A SOA Approach
    - SOMARA encourages a new content acquisition approach that can be done through a web service framework, which is the advantage of SOA and open platform. Associated media content for a particular scenario could be requested from various sources such as third party (Google Maps, Flickers) or potential open services (Photogrammetry services). Obtained media content can be visualized on the screen and AR environment where mobile users are able to create interaction with active media content.

1.8 Thesis Organization

This research focuses on the design of SOMARA and the development of a novel mobile AR application based on SOMARA. The novel application is on open platform where SOA and web service framework is required and implemented into the application in order to create the mobile AR client that effectively perform AR tasks and interoperability tasks. The following is the thesis organization that explains the process of pursuing the research and creating useful SOMARA and its application in order to accomplish the contribution to knowledge.

Chapter 1 introduces the research proposed in this thesis, and provides detailed discussions of the research concepts, technological approaches taken, architectures and components designed, and solutions provided to achieve the research goals proposed.

Chapter 2 presents detailed background, concepts and applications of each component architectural proposed in the research. This information is incorporated with related technologies and techniques that were studied and implemented.

Chapter 3 provides the AR scenarios composed of architectural requirements and museum environment scenarios. The architectural requirements explain solutions and techniques that improve the usability of standalone mobile AR applications and create a mobile AR application on open platform. This chapter also presents the implementation of SOMARA and its application in the potential museum environment scenarios that the novel mobile AR application can enhance visitors' perception and experience in MLAs.

Chapter 4 explains the architectural design of SOMARA and its service orientation that generate a novel mobile AR application. The service orientation is composed of a mobile AR client, web service framework and service providers and each component is on client-server architecture. The processes of performing AR tasks, content acquisition and utilization are also described in this chapter.

Chapter 5 describes the implementation of SOMARA in MLA scenarios. The SOMARA-based application is developed on native development platform, Objective C, XCode and iPad mobile device. The novel mobile AR application requires an AR SDK in order to perform AR tasks and simultaneously work with the AR application and mobile interface to interact with mobile users and visualize content on the screen.

Chapter 6 explains the final mobile AR application based on SOMARA that support physical museum learning scenario and home-based museum learning scenario in testing and evaluation issues. The testing and evaluation shows how the designed SOMARA and its application can efficiently support content acquisition and utilization over a web service framework and AR platform. Therefore, the testing and evaluation will prove that the novel SOMARA is capable, useful and stable for applying in other scenarios.

Chapter 7 concludes the research accomplishment and gives some directions to future developments and implementations such as applying the system with multiple photogrammetry service providers or developing mobile client side with client-server architecture that will be able to enhance collaboration and interoperability in sharing digital AR content.

Chapter II

2 Service-Oriented Mobile AR

The aims of this research is to design a Service-Oriented Mobile Augmented Reality Architecture (SOMARA) and build proof of concepts and prototypes through a novel mobile AR application based on SOMARA in order to support content acquisition and utilization on mobile and AR platforms. There are three main SOMARA requirements that efficiently improve the usability and adaptability of standalone mobile AR applications including: 1) service orientation, 2) content acquisition and 3) consumption. This chapter explains the components of each requirement and the technology approaches in the literature survey including background, concepts, applications and implementation related to the significant components in a SOMARA.

2.1 Augmented Reality

Augmented Reality (AR) is a mixed reality technology that is a combination of real and virtual environments. Mixed reality consists of the real environment, AR, augmented virtuality and virtual reality (VR). Figure 2.1 presents mix reality and its elements in the scale of the continuum between reality and virtual reality.

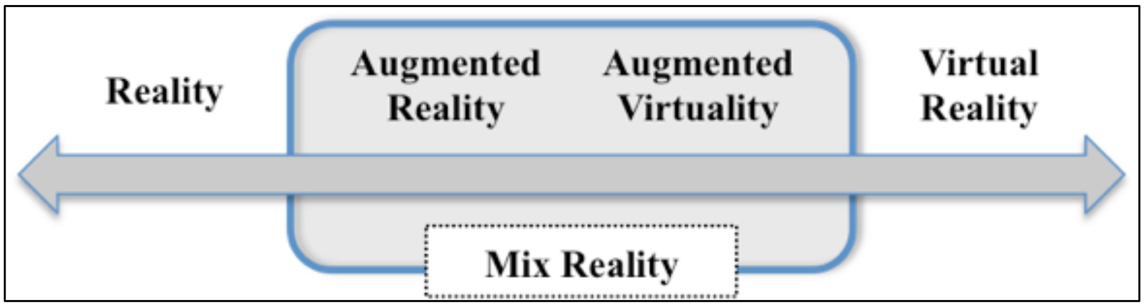


Figure 2.1 The continuum of reality and virtuality [49]

AR combines the real environment together with computer generated virtual objects and enables users to interact with virtual content as an element of the real environment. AR is different from VR in the experience of the realistic. VR is a technique to create a virtual environment and let users be a component of that environment. Consequently, users seem to be incorporated as part of the scene and can interact with other components. AR simultaneously provides users with a realistic and a virtual experience while VR allows users to participate only within the virtual environment. Applying VR requires display technologies such as a computer screen or a head-mounted display in order to present the virtual scene or content, whereas AR functionally uses embedded cameras to track targeted objects and related content is visualized on a screen. Moreover, AR can be applied for indoor or outdoor purposes using positioning systems such as GPS in order to present content based on the user's current location, a process, which is called location-based AR. These technologies enrich AR applications in terms of flexibility, adaptability and variety where AR has become a powerful tool in developing a broad range of applications on many platforms. Additionally, AR supports users in various activities such as marketing, education, maintenance, design, simulation, medical and personal assistance [17][18][50][51][52]. Examples of AR applications include developing an interactive coloring book [53], AR experiences for museum visitors [38][54], an AR shopping assistant [55][56] and developing AR to support mathematics and geometry education [57].

2.2 Mobile AR System

The mobility concept has become an innovative and powerful tool to create more advanced and usable applications implemented in many different scenarios. A current challenge in mobile technology is the implementation of AR on mobile platforms that focus on the convergence of AR, ubiquitous and wearable computing [27]. A mobile AR system is able to provide mobile users with media content e.g. 3D models, animations, images, videos and text visualized on a mobile screen and AR tasks can be done in both indoor and outdoor environments. Examples of mobile AR applications are:

- Virtual character-based application
- Cultural heritage
- Edutainment and games

- Navigation and path-finding
- Collaborative assembly and design
- Industrial maintenance and inspection

Examples of mobile AR applications are systems for remote collaboration [58], mobile AR for navigation and collaborative use [59] and for cultural heritage [60]. The motivation for this research focuses on designing a SOMARA, which is the implementation of a SOA on a mobile AR system, which addresses shortfalls in many of the current AR systems. For example, a SOMARA application is an open AR application on a mobile platform that is applied in MLA scenarios — current AR applications are closed to user-generated content. An open SOA allows novel mobile AR applications to be designed to accomplish AR tasks as well as interoperability tasks through a web service framework, where interaction between mobile users and the application can be done through the mobile interface. The AR tasks are composed of object tracking, recognition and content visualization. The following explains the notion of each AR tasks, which is one of the components in SOMARA.

- Object tracking is the process of detecting the real scene and finding a reference object pre-defined in the configuration file. Object tracking requires the embedded-camera of a mobile device in order to capture the video of the real scene and the tracking process will detect feature of objects in the scene.
- Object recognition is done after the system detect feature of an object in the real scene. The extracted feature will be compared with the pre-defined feature of a reference object stored in the content library. Once the reference is recognized, it will be augmented by presenting related content superimposed on top of the recognized object.
- Content visualization is augmenting a recognized object with related media content combined into the real scene. The novel mobile AR system is able to visualize various kinds of media content on the screen and users are also able to have interaction with the active content in the AR environment.

The tracking process requires an embedded camera on a mobile device that tracks reference objects and related content is visualized on the screen. The mobile AR application is applied in two MLA scenarios including a museum-based learning and a home-based learning. These two scenarios demand physical object tracking and image-based tracking. The object tracking requires a computer vision technique whereby the embedded camera on the mobile device is used to capture video streams and perform the tracking and recognition tasks. The following sections describe the literature survey and technologies for the SOMARA and the novel mobile AR application development. The components inside the SOMARA are mainly composed of AR tasks and interoperability tasks.

2.2.1 Computer vision

Typical AR applications use computer vision as a sensing technique to work with cameras for vision-based or optical tracking. Computer vision is a powerful feature in AR due to the fact that it depends on features that are presented naturally to cameras. Moreover, it does not need any adjustment of the environment. The system detects and tracks real world objects, calculates the virtual object position and orientation and then overlays the virtual objects onto the real scene [61]. Computer vision is an area of computer graphics used for image or video frame processing that has been implemented in many types of application and platform such as face detection, 3D model building (photogrammetry), medical imaging, motion capture, robot vision, image processing, fingerprint recognition and biometrics [39]. In mobile technologies, computer vision is a technique applied in mobile AR systems for vision-based tracking. Consequently, cameras and displays are the most important modules in mobile devices for tracking and visualization. The following sections describe the concept and application of vision-based tracking.

2.2.2 Object tracking

Object tracking is the most significant task in a real-time AR approach [62] in order to detect and calculate the position and orientation of objects such as shapes, 2D images or 3D objects. The system provides virtual objects combined with the real environment at the calculated position and pose. Additionally, tracking incorporates feature extraction techniques to extract features of these objects and perform object recognition tasks. Real-time accurate tracking is a key requirement of an AR system to superimpose 3D virtual objects onto a real world scene at the right location and time. The achievement of this process is a precise composite of real objects and virtual objects, which is called registration [63]. Tracking techniques in AR are composed of sensor-based, vision-based and hybrid tracking [24].

Sensor-based tracking

Sensor-based tracking utilizes sensor technologies such as ultrasound, infrared laser-diodes, optical gyroscopes and magnetic, acoustic and mechanical sensors for inside-out and outside-in tracking [50][64] that apply signal emitters and sensors on target and tracker equipment.

Vision-based tracking

Vision-based tacking integrates cameras with computer vision algorithms in order to capture video streams as an input and accomplish camera pose estimation of 3D virtual objects relevant to the real world objects [17] [65].

Hybrid tracking

Hybrid tracking requires two or more sensing techniques to work together for more robust and accurate tracking in outdoor environments. Examples of the sensing techniques are GPS and inertial and active transmitters and receivers (magnetic, optical, ultrasonic) and passive optical [66].

Each tracking technique or tracker has its own specific functions and operations. Although they are suitable for different types of application, the objectives of tracking techniques are to specify targets in the real world and to calculate the virtual objects' positions relative to real world objects with high accuracy and robustness. Implementing tracking methods in an AR approach are based on the type of AR application, such as mobile or static, and the environment in which the AR devices are applied [20], meaning they may be restricted to being indoor or outdoor. Therefore, a combination of vision-based tracking and robust tracking techniques such as GPS can provide more robust and advanced applications on mobile devices for outdoor environments.

In this research, the implementation of the SOMARA relies on a mobile device (e.g. an iPad) and uses embedded-camera and vision-based tracking. Such mobile devices are commonly available to the user in AR scenarios, such as a museum learning. The novel mobile AR application and mobile interface performs AR operations in order to visualize 3D virtual objects and overlay the virtual objects onto real targeted objects at the right pose in an indoor environment. Computer vision-based tracking is suitable for many indoor AR application platforms as they do not require any specific hardware or maintenance. The topics describe vision-based tracking that is typically composed of feature-based tracking and model-based tracking are in Appendix F.

2.3 Mobile AR SDKs

In section 2.2 this thesis described many computer vision, software methods and algorithms that are or can be applied to Mobile AR systems. In this section the thesis gives and overview of some of the more useful Mobile AR SDKs that can be exploited to implement novel AR applications.

For mobile application developers, there are AR Software Development Kits (SDKs) offering native development frameworks as a tool that can be integrated into the development framework including PC and mobile platforms. The AR SDKs offer AR tasks such as object tracking, object recognition, content rendering and visualization to developers so that they can integrate the provided application libraries or functions into the development framework. Applying a mobile AR SDK is a solution for any mobile AR application that does not require the development of AR tasks; utilizing existing AR SDKs can be the simplest and quickest way to develop a mobile AR application. The initial task of a mobile AR application is object tracking where the applied AR SDK controls the embedded camera and presents the camera view on a screen. The system then detects objects on the screen and recognizes these targeted objects by performing feature extraction and comparing them with the pre-defined features of objects in the system. The recognized object is augmented by superimposing pre-defined media content such as 3D models, images, videos, etc. on top of the tracked object. Each AR SDK exposes different functions and techniques in order to perform AR tasks. Examples of AR SDK are ARToolKit, Qualcomm and Metaio.

2.3.1 ARToolKit

ARToolKit is one of the first open-source software providing an AR SDK on all platforms including Mac OS X, PC, Linux, Android and iOS and a plug-in for Unity 3D. It supports Natural Feature Tracking (NFT) focusing on marker tracking, OpenCV for camera calibration and OpenSceneGraph for advanced rendering. The SDK also supports mobile focus including OpenGL ES, multi-platform mobile support and GPS and compass integration on the iOS platform [67] .

2.3.2 Qualcomm Vuforia

Qualcomm presents Vuforia SDK that supports AR application development with iOS, Android or Digital Eyewear including tracking and recognizing images, objects, text, markers and environment reconstruction. Vuforia also supports mixed reality: a combination of AR and VR in order to create immersive environments. Vuforia-based AR applications can be applied on optical see-through digital eyewear such as the Samsung Gear VR or R-7 in order to present mixed reality experiences to users. In addition, Vuforia offers its Cloud Recognition Service to recognize images and manage databases in the cloud [68].

2.3.3 Metaio

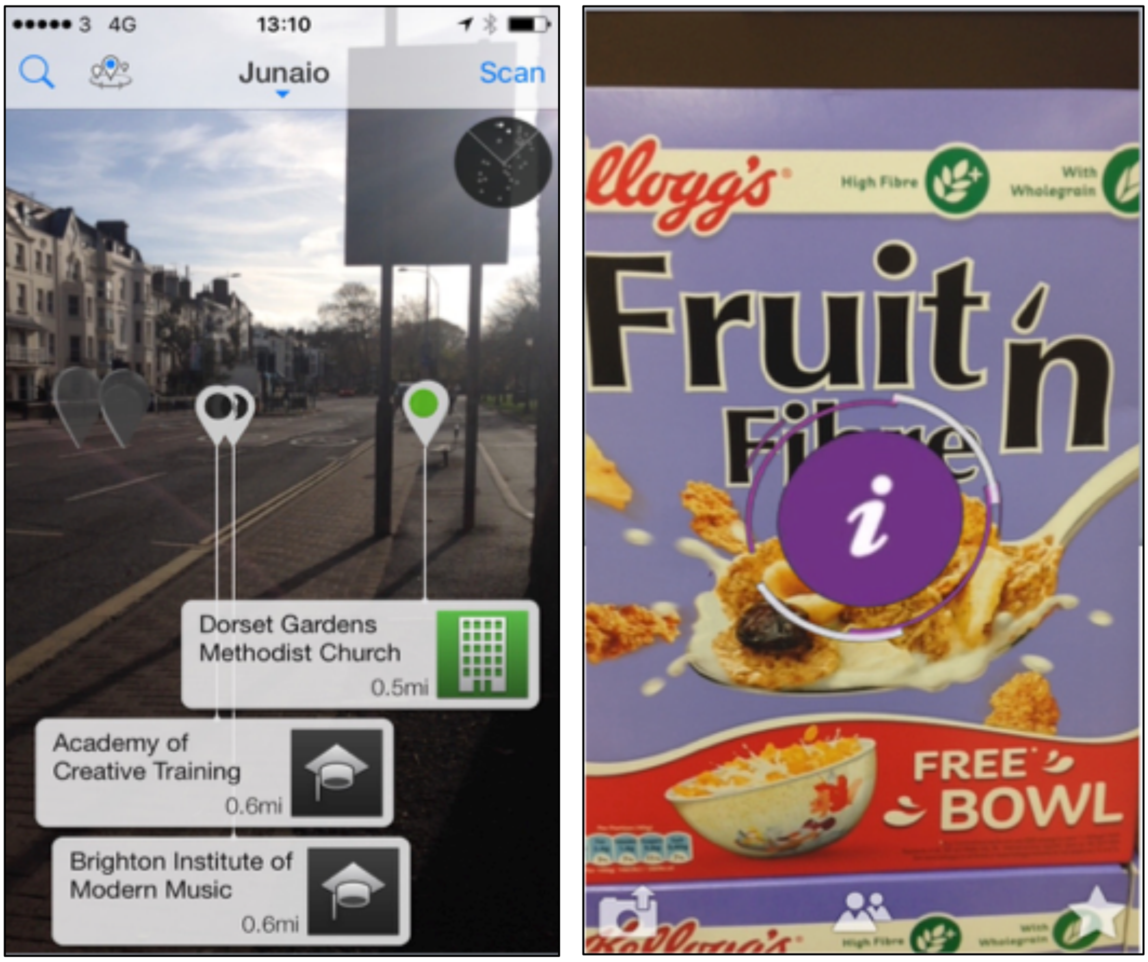
Metaio SDK supports GPS, Simultaneous Localization and Mapping (SLAM) tracking and location services and it can be applied to iOS, Android, Unity 3D and Windows frameworks. The SDK can perform location-based tracking, marker and marker-less tracking including images and physical objects. It also supports Augmented Reality Experience Language (AREL) templates that enable iOS developers to create AR applications with AREL using HTML5 and Javascript on UIWeb View instead of Objective C. In addition, Metaio provides a cloud plug-in so that AR applications are able to access the Metaio Cloud and utilize or upload content into the cloud. Metaio offers Junaio which is a mobile AR browser supporting location-based AR and image-based tracking. Junaio enables mobile users to create their own AR channels containing personal AR content, publish AR scenarios or access other existing channels [45][45].

Due to the continuously increasing requirements of SOA, cloud computing and mobile computing, applying a web services framework into mobile AR systems can be a beneficial solution for client-side applications in order to achieve mobile computing and digital content sharing with platform independence. In this research, SOA is integrated into a closed mobile AR platform to create a novel mobile AR application on an open platform that is able to accomplish content acquisition as well as content utilization. The following are the details and implementation of the SOA, the novel mobile AR application and its components.

2.4 Mobile Application Development Tools

In mobile application development frameworks there are existing tools for building mobile applications on different platforms. Mobile platforms that can effectively support and run AR applications are iOS and Android. Mobile AR applications on iOS devices can be developed on native development platforms by using XCode and Objective C. Moreover, applications can be created on hybrid platforms where HTML5 and JQuery are applied instead of the native platform, and this makes the development processes easier for HTML developers [69]. Android is another platform that also supports AR applications and their development tool is based on Android Studio and Java [70].

There is a solution for general mobile users or content providers who want to create and publish AR environments through an AR browser that can perform indoor and outdoor AR. The AR browser is an easy way to produce AR environments or provide AR content through personal channels. It performs object tracking and location-based tracking so the system accesses the user's channel and downloads personal content for visualisation on the browser. Examples of AR browser are Junaio [71], Aurasma [41], Layar [42]. Figure 2.2 shows a snapshot of Junaio performing location-based AR and Aursama performing logo tracking.


13:10
Junaio
Scan
Dorset Gardens Methodist Church 0.5mi
Academy of Creative Training 0.6mi
Brighton Institute of Modern Music 0.6mi
Kellogg's
Fruit 'n Fibre
FREE BOWL

(a) Junaio (b) Aurasma

Figure 2.2 Snapshots of Juanio and Aurasma AR browsers

2.5 Service-Oriented Architecture

SOA is an open architecture containing services for any application or other service in order to have connection and request data or processing [72][73]. Cloud computing has adopted SOA in order to create networks of cloud computing and services [3]. SOA has been integrated into many difference scenarios such as AR e-business systems [33], industrial systems [74], smart-home architecture [75] and co-production systems [76]. The SOA technology used to create connection to services is web services. The following is the details and components of web services and its framework that is applied into any client application.

2.5.1 Web services

A web service is a technology that relies on SOA. Web services can be implemented in client-side applications in order to provide service interfaces or a data service architecture for interoperability tasks in the form of APIs on server sides. A web service allows application clients or other servers to discover, connect and request operations or processing and acquire outputs from the servers via the Internet across hardware and software platforms. It is implemented in a variety of different approaches. The Web 2.0 technology that is popular at the moment allows users to share their own information over the Internet, such as in social networking. A mashup is an application platform in Web 2.0 technology that is the implementation of web services or web APIs. Users can download mashup APIs for inclusion in client-side applications and requesting services. Examples of mashup technologies are Google Maps and Rich Site Summary (RSS). Examples of web service methods are SOAP and REST.

SOAP

Simple Object Access Protocol (SOAP) is a web service standard and communication protocol that utilizes an Extensible Markup Language (XML) document format in operation and sends the SOAP messages via Hypertext Transfer Protocol (HTTP). Utilizing SOAP requires provision of a Web Services Description Language (WSDL) definition document for service descriptions such as input, output and procedure approaches for client sides. Developers can find WSDL documents and web service interface information from Universal Description, Discovery, and Integration (UDDI) registries that are maintained and presented on websites [77]. Examples of SOAP web service providers are TempConvert, StockQuote and Astronomy [78]. Figure 2.3 presents the structure of a service request and response through SOAP in application-to-application communications.

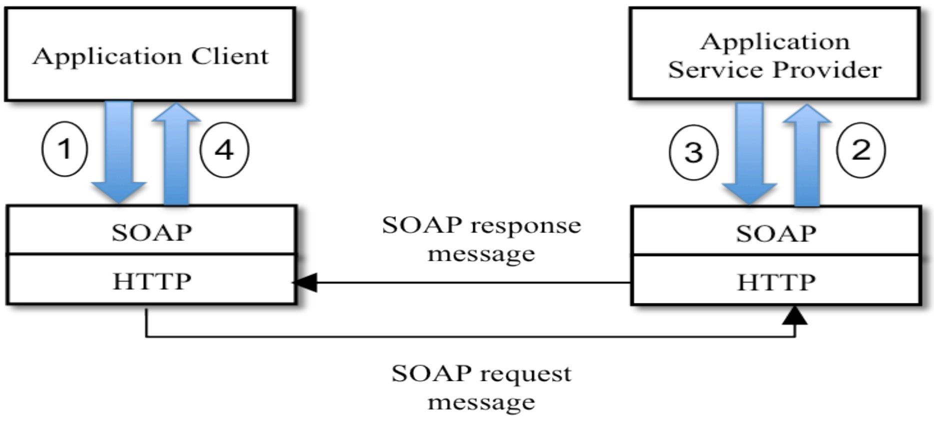
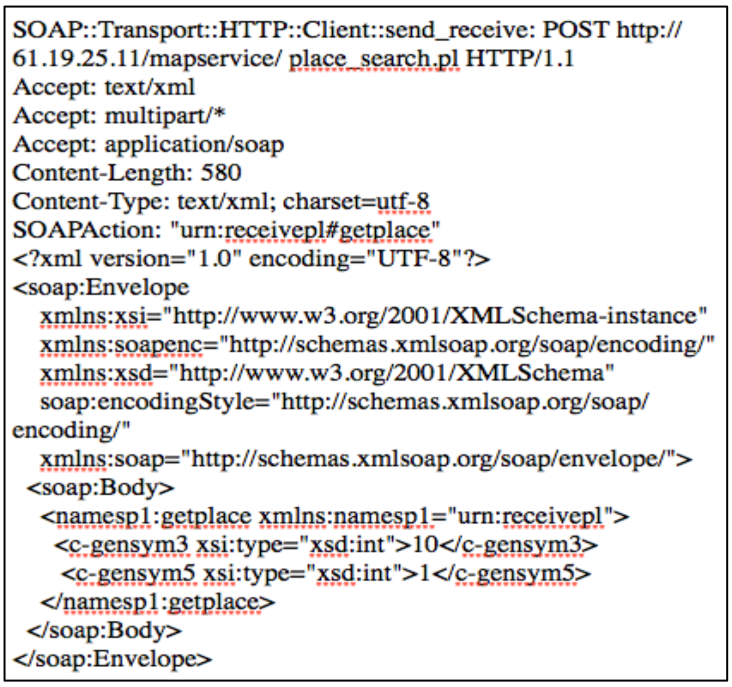


Figure 2.3 Service request and response process using SOAP

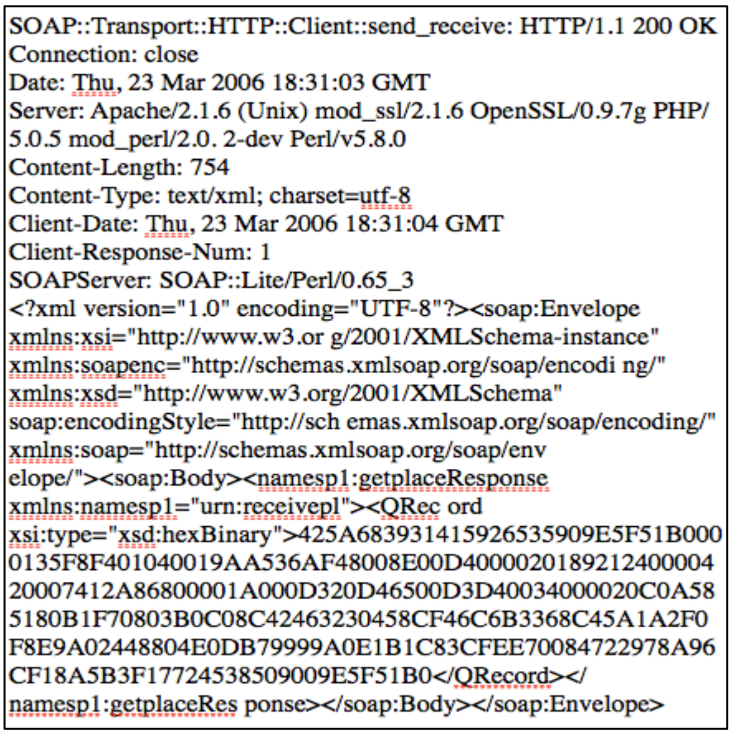
Figure 2.4 illustrates an example of a SOAP request message that is in XML document format. The SOAP request message in XML format consists of the method required, and parameters are sent to that method.



```
SOAP::Transport::HTTP::Client::send_receive: POST http://
61.19.25.11/mapservice/ place_search.pl HTTP/1.1
Accept: text/xml
Accept: multipart/*
Accept: application/soap
Content-Length: 580
Content-Type: text/xml; charset=utf-8
SOAPAction: "urn:receivepl#getplace"
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   soap:encodingStyle="http://schemas.xmlsoap.org/soap/
encoding/"
   xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
   <namesp1:getplace xmlns:namesp1="urn:receivepl">
     <c-gensym3 xsi:type="xsd:int">10</c-gensym3>
      <c-gensym5 xsi:type="xsd:int">1</c-gensym5>
   </namesp1:getplace>
  </soap:Body>
</soap:Envelope>
```

Figure 2.4 An example of a SOAP request message

Figure 2.5 presents a SOAP response message that is an XML document coved by a SOAP protocol message. The document contains output of the requested module.



```
SOAP::Transport::HTTP::Client::send_receive: HTTP/1.1 200 OK
Connection: close
Date: Thu, 23 Mar 2006 18:31:03 GMT
Server: Apache/2.1.6 (Unix) mod_ssl/2.1.6 OpenSSL/0.9.7g PHP/
5.0.5 mod_perl/2.0. 2-dev Perl/v5.8.0
Content-Length: 754
Content-Type: text/xml; charset=utf-8
Client-Date: Thu, 23 Mar 2006 18:31:04 GMT
Client-Response-Num: 1
SOAPServer: SOAP::Lite/Perl/0.65_3
<?xml version="1.0" encoding="UTF-8"?><soap:Envelope
xmlns:xsi="http://www.w3.or g/2001/XMLSchema-instance"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encodi ng/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
soap:encodingStyle="http://sch emas.xmlsoap.org/soap/encoding/"
xmlns:soap="http://schemas.xmlsoap.org/soap/env
elope/"><soap:Body><namesp1:getplaceResponse
xmlns:namesp1="urn:receivepl"><QRec ord
xsi:type="xsd:hexBinary">425A6839314159265359O9E5F51B000
0135F8F401040019AA536AF48008E00D4000020189212400004
20007412A86800001A000D320D46500D3D40034000020C0A58
5180B1F70803B0C08C42463230458CF46C6B3368C45A1A2F0
F8E9A02448804E0DB79999A0E1B1C83CFEE70084722978A96
CF18A5B3F17724538509009E5F51B0</QRecord></
namesp1:getplaceRes ponse></soap:Body></soap:Envelope>
```

Figure 2.5 An example of a SOAP response message

REST

Representational State Transfer (REST) is a new method for making web service connections and sending messages in client-server architecture. REST is an alternative to SOAP and uses service APIs to explain application resources in distributed systems. Consequently, REST is an option for implementing web services by using potential APIs from application servers. APIs represent application resources identified by a Uniform Resource Identifier (URI). REST is an architecture that transfers data over HTTP, composed of methods GET, POST, DELETE [79] [80]. Examples of REST web service providers are Twitter, Last.fm and Foursquare [81]. Figure 2.6 shows the REST architecture where the service request and response is sent through HTTP.

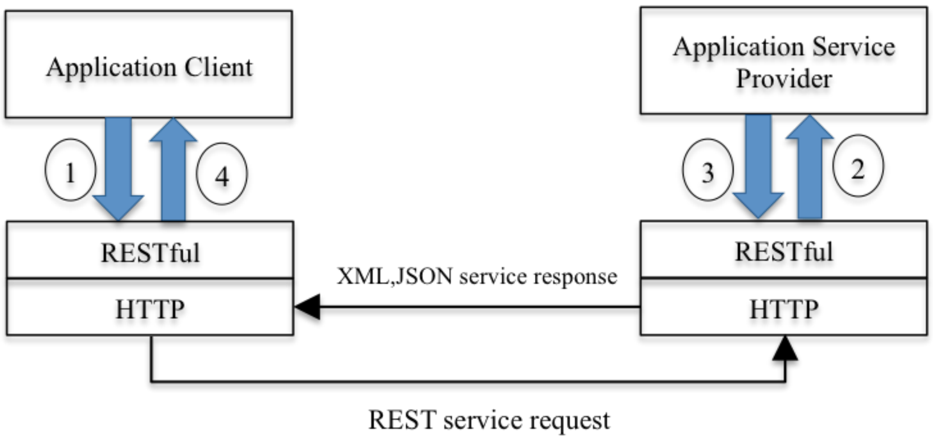
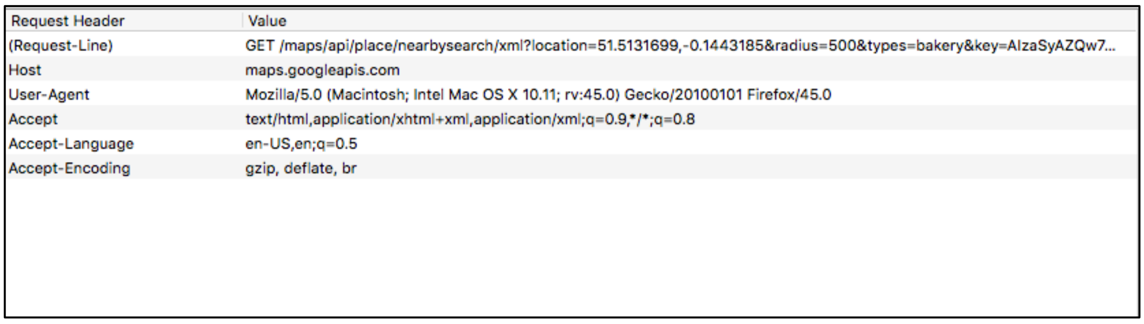


Figure 2.6 Service request and response process using REST

Figure 2.7 presents the structure of an HTTP service request by sending the REST service request in Uniform Resource Locator (URL) format:

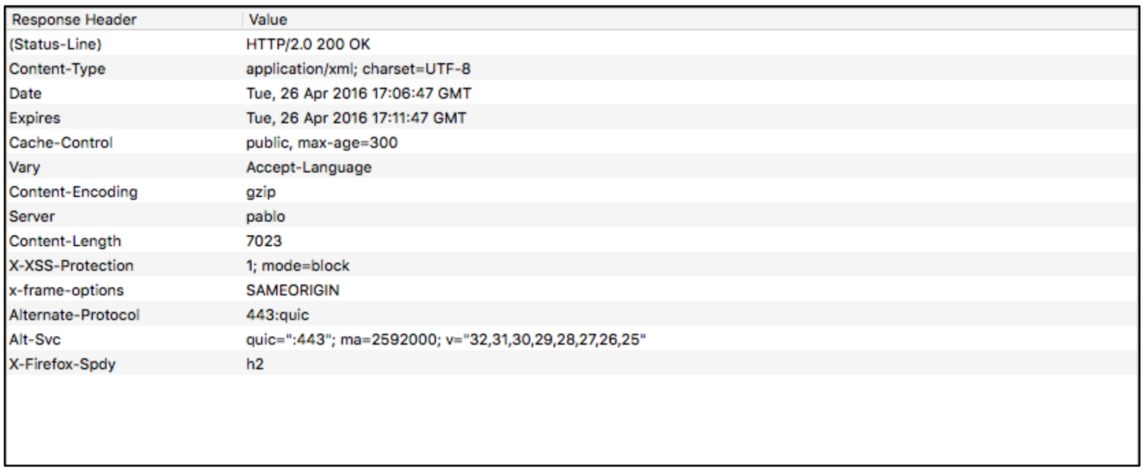
https://maps.googleapis.com/maps/api/place/nearbysearch/xml?location=51.5131699,-0.1443185&radius=500&types=bakery&key=AIzaSyAZQw7PEjmwNtsWRPphtw2D1omQCRjgw8I



Request Header	Value
(Request-Line)	GET /maps/api/place/nearbysearch/xml?location=51.5131699,-0.1443185&radius=500&types=bakery&key=AIzaSyAZQw7...
Host	maps.googleapis.com
User-Agent	Mozilla/5.0 (Macintosh; Intel Mac OS X 10.11; rv:45.0) Gecko/20100101 Firefox/45.0
Accept	text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language	en-US,en;q=0.5
Accept-Encoding	gzip, deflate, br

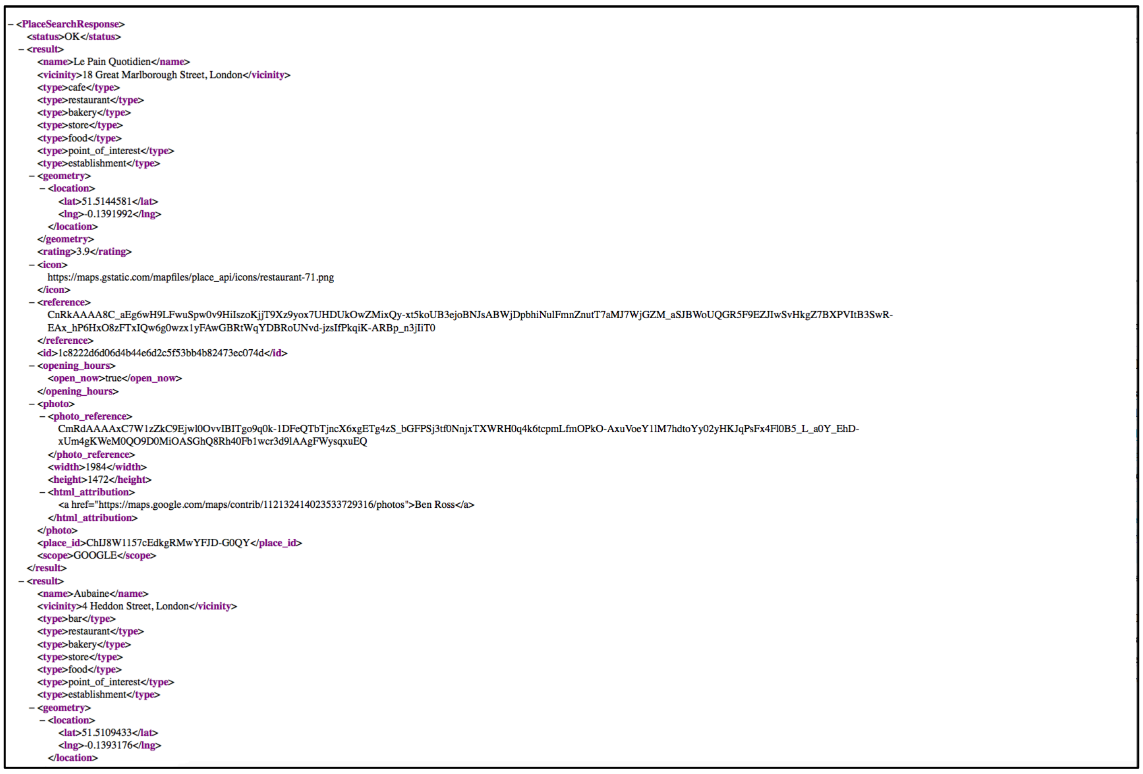
Figure 2.7 HTTP service request

Figure 2.8 shows the REST response header message via HTTP and Figure 2.9 shows the response content of the request in an XML document.



Response Header	Value
(Status-Line)	HTTP/2.0 200 OK
Content-Type	application/xml; charset=UTF-8
Date	Tue, 26 Apr 2016 17:06:47 GMT
Expires	Tue, 26 Apr 2016 17:11:47 GMT
Cache-Control	public, max-age=300
Vary	Accept-Language
Content-Encoding	gzip
Server	pablo
Content-Length	7023
X-XSS-Protection	1; mode=block
x-frame-options	SAMEORIGIN
Alternate-Protocol	443:quic
Alt-Svc	quic=":443"; ma=2592000; v="32,31,30,29,28,27,26,25"
X-Firefox-Spdy	h2

Figure 2.8 HTTP response header message



```
– <PlaceSearchResponse>
    <status>OK</status>
  – <result>
      <name>Le Pain Quotidien</name>
      <vicinity>18 Great Marlborough Street, London</vicinity>
      <type>cafe</type>
      <type>restaurant</type>
      <type>bakery</type>
      <type>store</type>
      <type>food</type>
      <type>point_of_interest</type>
      <type>establishment</type>
    – <geometry>
      – <location>
          <lat>51.5144581</lat>
          <lng>-0.1391992</lng>
        </location>
      </geometry>
      <rating>3.9</rating>
    – <icon>
        https://maps.gstatic.com/mapfiles/place_api/icons/restaurant-71.png
      </icon>
    – <reference>
        CnRkAAAA8C_aEg6wH9LFwuSpw0v9HiIszoKjjT9Xz9yox7UHDUkOwZMixQy-xt5koUB3ejoBNJsABWjDpbhiNulFmnZnutT7aMJ7WjGZM_aSJBWoUQGR5F9EZJIwSvHkgZ7BXPVItB3SwR-
        EAx_hP6HxO8zFTxIQw6g0wzx1yFAwGBRtWqYDBRoUNvd-jzsIfPkqiK-ARBp_n3jIiT0
      </reference>
      <id>1c8222d6d06d4b44e6d2c5f53bb4b82473ec074d</id>
    – <opening_hours>
        <open_now>true</open_now>
      </opening_hours>
    – <photo>
      – <photo_reference>
          CmRdAAAAxC7W1zZkC9Ejwl0OvvIBITgo9q0k-1DFeQTbTjncX6xgETg4zS_bGFPSj3tf0NnjxTXWRH0q4k6tcpmLfmOPkO-AxuVoeY1lM7hdtoYy02yHKJqPsFx4Fl0B5_L_a0Y_EhD-
          xUm4gKWeM0QO9D0MiOASGhQ8Rh40Fb1wcr3d9lAAgFWysqxuEQ
        </photo_reference>
        <width>1984</width>
        <height>1472</height>
      – <html_attribution>
          <a href="https://maps.google.com/maps/contrib/112132414023533729316/photos">Ben Ross</a>
        </html_attribution>
      </photo>
      <place_id>ChIJ8W1157cEdkgRMwYFJD-G0QY</place_id>
      <scope>GOOGLE</scope>
    </result>
  – <result>
      <name>Aubaine</name>
      <vicinity>4 Heddon Street, London</vicinity>
      <type>bar</type>
      <type>restaurant</type>
      <type>bakery</type>
      <type>store</type>
      <type>food</type>
      <type>point_of_interest</type>
      <type>establishment</type>
    – <geometry>
      – <location>
          <lat>51.5109433</lat>
          <lng>-0.1393176</lng>
        </location>
```

Figure 2.9 Response document in XML format

2.5.2 Web service mobile AR platform

Exploiting web service frameworks to create mobile AR and mixed reality systems as open AR architectures that are able to obtain content from open resources can be very useful for creating novel AR applications — SOMARA is a web service architecture designed to do just this. Such mobile AR clients based on a web service framework can access open services and support content acquisition on client-server architectures and the Internet. Integrating a web service framework into stand-alone (e.g. closed commercial systems like Aurasma, etc.) mobile AR applications can bring general data, user-generated content or commercial content to mixed reality systems, however there is little work addressing this need [48] [40]. Other examples of applications applying SOA into AR are E-business systems [33], development of an AR browser for an interoperable outdoor AR system [82] and creating a mixed reality mash-up system for mobile AR applications (Mixed Reality Web Service Platform) [83]. Most of the applications are based on third party content providers such as Google Maps, Yelp, Twitter, etc. This open and user-generated content is visualized on mobile AR browsers that require GPS or sensors in order to identify the user's location and get metadata depending on their location.

Building on this limited work, this thesis proposes SOMARA, which offers a service-orientation approach designed for supporting mobile AR client development in applications such as AR Museum scenarios, but also other scenarios. SOMARA's service-orientation is composed of a mobile client, web service framework and service providers. This novel mobile AR application can be classed as one of the next generation of mobile applications that combines a web service framework into an application on mobile devices such as an iPhone or iPad [2]. While SOMARA's mobile AR client is applied in museum environment scenarios, discussed in following chapter of this thesis, it can be applied to other commercial scenarios such as retail shopping. However, SOMARA's AR technology is demonstrated in this thesis with a scenario that presents virtual exhibitions of cultural objects to enhance visitors' experiences in MLAs. SOMARA's mobile AR application is able to perform typical AR tasks including object tracking, recognition and content visualization. Additionally, it performs interoperability tasks by creating connections to open providers in order to send service requests and receive responses. The web service framework is a tool that encourages content acquisition in the client-server architecture and the mobile AR client acquires associated media content from open museum content providers.

Another source of content that could be beneficial for museum environments and other scenarios are photogrammetry service providers. Currently, such service providers are sparse, but the potential is enormous and as such, SOMARA is laying the groundwork to integrate to such services. Photogrammetry or image-based reconstruction services can produce a virtual 3D model of a preferred object so that the obtained 3D model can be usefully consumed in other situations, i.e. MLA scenarios.

Given the enormous potential of photogrammetry services integrated with AR applications it is worth discussing such services in more detail.

2.6 Photogrammetry Services

Photogrammetry is an advanced technique for generating a 3D virtual object from multiple images of a target object or for performing image-based reconstruction. This technique makes the process of 3D object acquisition easier and more flexible. The processing steps of the reconstruction of photo-realistic 3D content consist of feature matching, structure from motion recovery, dense depth map estimation, 3D model building and texture map recovery [84]. In computer vision techniques, performing vision-based tracking captures natural features such as edges, lines and textures from a virtual environment. Tracking natural features can be used to develop more advanced computer vision and graphic tasks by implementing 3D modeling, rendering and reconstruction approaches. These approaches allow the applications to create 3D virtual models from real objects automatically by tracking edges or wireframes, creating textures and rendering virtual models of objects.

Photogrammetry providers offer 3D reconstruction services that are based on client-server architecture and web-based applications. Users or client sides can use tools that are based on the web application or desktop computer platform for uploading images to the server and editing the final 3D virtual models. Examples of photogrammetry applications are: Arc3D and MeshLab, and 123D Catch .

2.6.1 Arc3D and MeshLab

Arc3D and MeshLab constitute a web-based application for reconstructing 3D virtual models from multiple images. MeshLab is a mesh-processing tool that is composed of an image uploader, model viewer and editing function. MeshLab is a client-side application on PCs to work with Arc3D within client-server architecture [85][86]. The web-based service for 3D reconstruction requires photos of an object and an Internet connection to accomplish automatic 3D reconstruction processing by a cluster server. Therefore, users have to use specific tools such as web-based applications for uploading photos and manipulating 3D virtual models [87].

2.6.2 123D Catch

Autodesk® 123D Catch is a tool for generating 3D models. It is a web-based application on a client-server architecture that allows users to submit multiple digital photos of objects via an Internet connection and achieve realistic 3D models. Users can use a web-based application, PC or tablet PC tool for uploading pictures and editing 3D models [7].

Most of the photogrammetry applications for 3D reconstruction are created using a web-based and client-server approach rather than a web service architecture approach. In addition, the applications are in a close dependency platform so they do not provide any operational interface for client-side applications. Another example of a web-based application for geospatial data services is EuroSpec, which is an implementation of a web-based service and geospatial data infrastructure. EuroSpec provides reference geographic information that cross-border users can locate, select, access and download data sets from a distributed source. The communication is via the Internet, using a web-based approach and Open Geospatial Consortium (OGC) standards [88].

OGC offers open standards for interfaces or encoding on geospatial data and location services. This allow multiple and diverse organizations to acquire interoperability, information sharing and location-based services. Examples of the standard are Built Environment and 3D, CityGML and Web Map Service. These can support service interface design, distributed computing between web service architectures, service providers and application clients that are platform independent. Now, there are many web service-based providers on distributed geospatial processing [89].

2.7 Web Service-based Photogrammetry

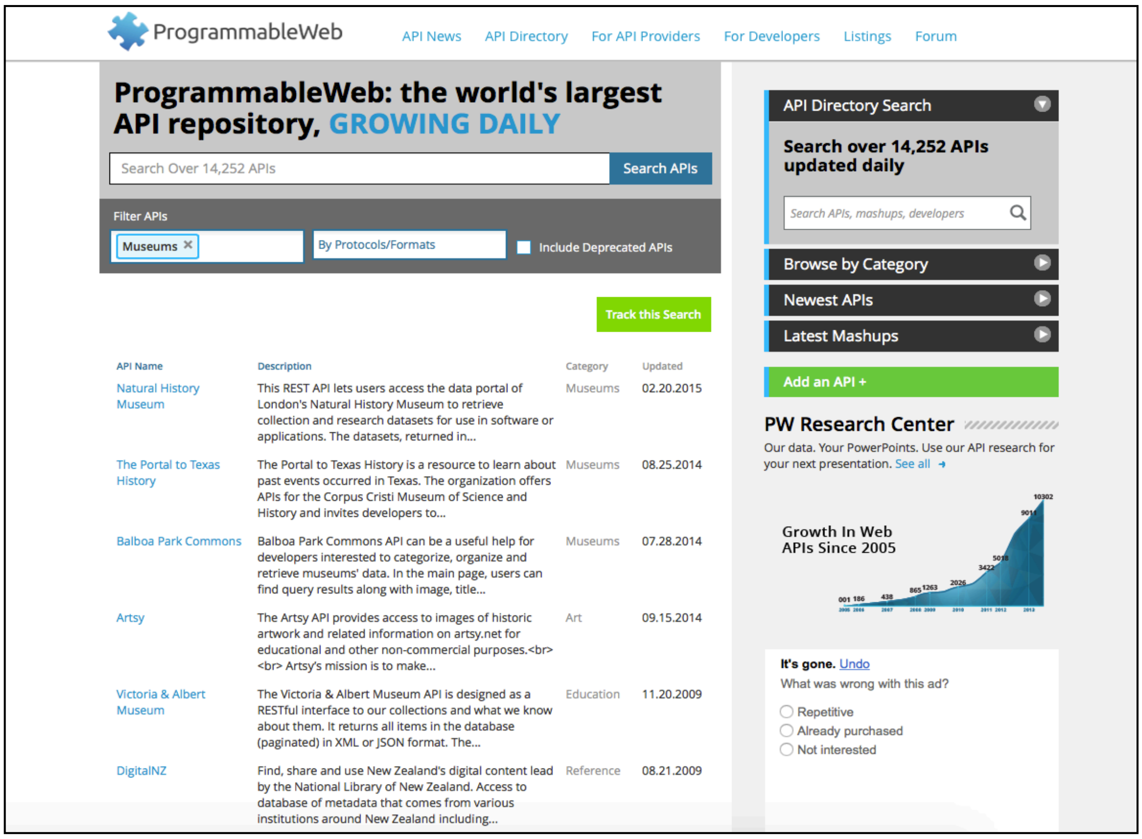
The SOMARA-based mobile AR application requires web service-based 3D photogrammetry providers for performing image-based reconstruction and accomplishing content acquisition on open mobile AR platforms. Therefore, integrating photogrammetry service interfaces into the web service framework could be beneficial because of the rapid communication, the increased requirements in interoperability tasks, information sharing, platform independence and effective server-side architecture. Furthermore, these features enhance the experience of mobile users and client-side applications in any platform to achieve both simple and complicated tasks such as data services, 3D reconstruction, geo-information and remote sensing by invoking the services over a communication network and the Internet.

There is some ongoing research on developing web service-based frameworks for geo-information services such as maps or Geographic Information Systems (GIS) processing. There is a web service architecture for GIS. The system provides GIS web services with the interoperability of data analysis and spatial data sharing from many organizations. The processed outcome of this architecture is a composite map from distributed mapping systems on the servers. The architecture is composed of web mapping servers, data sources and a web client. Each server offers an interface for the web clients to request the services [90]. Another application is a geo-information service named Building Information Models-Web Perspective View Service (BIM-WPVS). This service provides virtual 3D city models and an advanced 3D visualization web service on client-server architecture. The server opens a service interface for performing GIS and BIM data collection, 3D rendering and data visualization [91]. In addition, there is the design of a web service architecture for enhancing the proficiency of photogrammetry workstations. The architecture is composed of service customers, a portal server and service providers. The aim of this architecture is to allow users to request application services and digitized map images through a portal server that directly connects to the service providers [92].

Integrating a web service framework with photogrammetry services could be a solution for client-side applications to benefit from phenomenal mobile computing, distributed computing and AR content acquisition on mobile platforms. In this research, the mobile AR application requires web service-based photogrammetry providers that conceptually offer a photogrammetry or 3D reconstruction service on web service architecture by receiving a set of images of a target object from a mobile web service-based client. Then, a rendered 3D virtual object is sent back to the mobile client and presented in an AR environment.

2.8 Web Service Providers

One of the main components in a service-orientation architecture is a supply of web service providers. Web service providers offer platform-independent open services that any application can access through a web service framework. They present service interfaces such as WSDL and API documents so that client-side applications integrate potential service interfaces into their service request module. The service interfaces present data sources or methods that can be called in order to request data, metadata or content in open repositories. Currently, there are a large number of typical and specifically designed service interfaces from third parties in both SOAP and REST protocol format. Service APIs can be found on the Internet by using search engines. Figure 2.10 shows a web site containing APIs in many different areas that enables users or developers to search for potential APIs and get information and details of those targeted.



ProgrammableWeb: the world's largest API repository, GROWING DAILY

Search Over 14,252 APIs | Search APIs

Filter APIs: Museums ✕ | By Protocols/Formats | Include Deprecated APIs

Track this Search

API Name	Description	Category	Updated
Natural History Museum	This REST API lets users access the data portal of London's Natural History Museum to retrieve collection and research datasets for use in software or applications. The datasets, returned in...	Museums	02.20.2015
The Portal to Texas History	The Portal to Texas History is a resource to learn about past events occurred in Texas. The organization offers APIs for the Corpus Cristi Museum of Science and History and invites developers to...	Museums	08.25.2014
Balboa Park Commons	Balboa Park Commons API can be a useful help for developers interested to categorize, organize and retrieve museums' data. In the main page, users can find query results along with image, title...	Museums	07.28.2014
Artsy	The Artsy API provides access to images of historic artwork and related information on artsy.net for educational and other non-commercial purposes. Artsy's mission is to make...	Art	09.15.2014
Victoria & Albert Museum	The Victoria & Albert Museum API is designed as a RESTful interface to our collections and what we know about them. It returns all items in the database (paginated) in XML or JSON format. The...	Education	11.20.2009
DigitalNZ	Find, share and use New Zealand's digital content lead by the National Library of New Zealand. Access to database of metadata that comes from various institutions around New Zealand including...	Reference	08.21.2009

API Directory Search

Search over 14,252 APIs updated daily

Search APIs, mashups, developers

Browse by Category

Newest APIs

Latest Mashups

Add an API +

PW Research Center

Our data. Your PowerPoints. Use our API research for your next presentation. See all →

Growth In Web APIs Since 2005

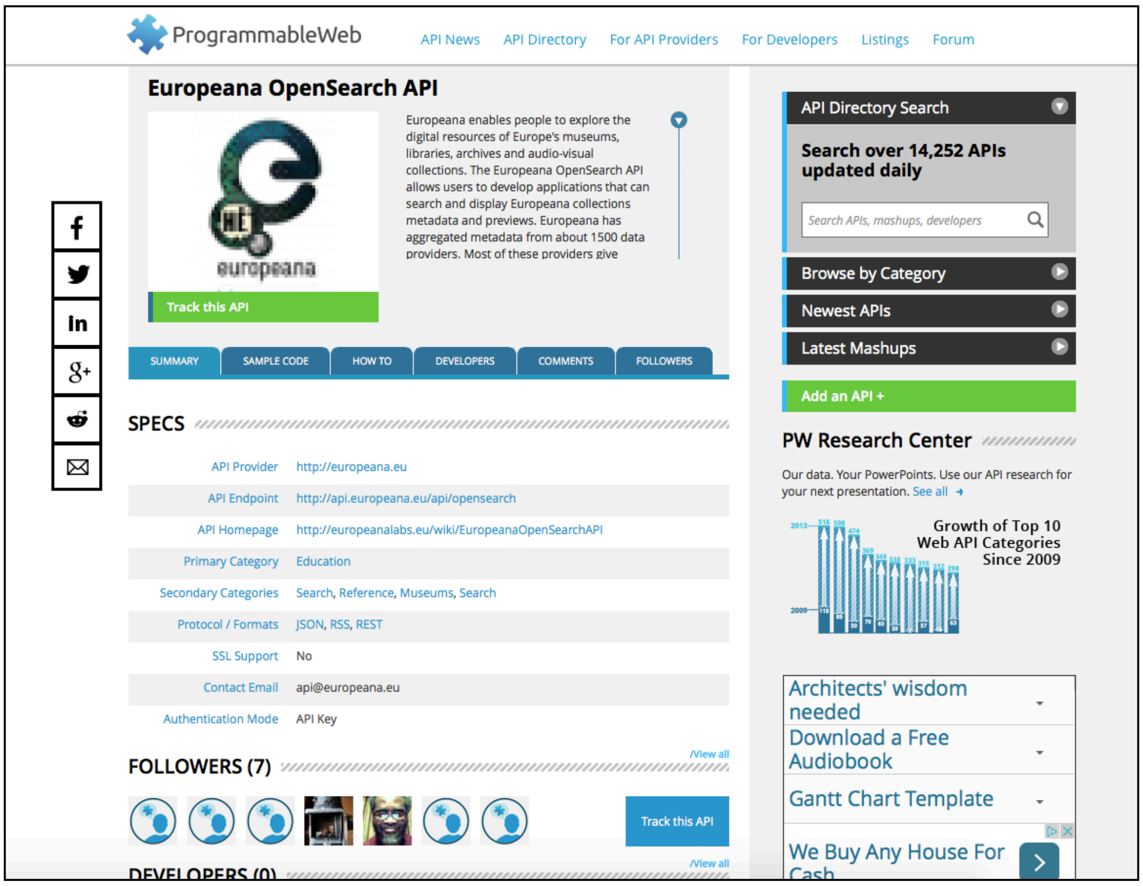
It's gone. Undo

What was wrong with this ad?

- Repetitive
- Already purchased
- Not interested

Figure 2.10 The API directory website [81]

The website allows developers to search APIs by identifying the types of API and protocol or format required. The list of APIs is displayed on the website as a result of searching in the repository. Figure 2.11 presents a selected API which is Europeana. The detail page shows information of the provider and the API as well as the protocol and format it supports. Users can then click the links that lead to the API homepage and see data about how to work with the API and the parameters it needs.



ProgrammableWeb

API News | API Directory | For API Providers | For Developers | Listings | Forum

Europeana OpenSearch API

Europeana enables people to explore the digital resources of Europe's museums, libraries, archives and audio-visual collections. The Europeana OpenSearch API allows users to develop applications that can search and display Europeana collections metadata and previews. Europeana has aggregated metadata from about 1500 data providers. Most of these providers give

Track this API

SUMMARY | SAMPLE CODE | HOW TO | DEVELOPERS | COMMENTS | FOLLOWERS

SPECS

API Provider	http://europeana.eu
API Endpoint	http://api.europeana.eu/api/opensearch
API Homepage	http://europeanalabs.eu/wiki/EuropeanaOpenSearchAPI
Primary Category	Education
Secondary Categories	Search, Reference, Museums, Search
Protocol / Formats	JSON, RSS, REST
SSL Support	No
Contact Email	api@europeana.eu
Authentication Mode	API Key

FOLLOWERS (7) /View all

Track this API

DEVELOPERS (0) /View all

API Directory Search

Search over 14,252 APIs updated daily

Search APIs, mashups, developers

Browse by Category

Newest APIs

Latest Mashups

Add an API +

PW Research Center

Our data. Your PowerPoints. Use our API research for your next presentation. See all →

Growth of Top 10 Web API Categories Since 2009

Architects' wisdom needed

Download a Free Audiobook

Gantt Chart Template

We Buy Any House For Cash

Figure 2.11 Europeana service API [93]

Developers are able to choose service providers that supply data relating to the scenario of the application. The content acquisition processes can be accomplished by an application client sending a request and receiving a response in XML or JavaScript Object Notation (JSON) format. The application client requires a parser in order to read through the received document and extract data. Some service interfaces are based on cloud computing and cloud APIs, such as Amazon Web Service APIs [94]. Furthermore, specifically designed applications also require particular content providers that return valuable content for client-side applications for a designed scenario.

2.9 Summary

Service-orientation, which is one of the components of SOMARA, enables the creation of a mobile AR application on open platform. The service-orientation is composed of a mobile AR client, a web service framework and web service providers. This chapter describes technologies and techniques required in developing the SOMARA-based mobile AR application based on the service-orientation. The literature survey leads to understanding each component of the service-orientation in more details and encouraging the application development by utilizing those techniques. It can also enhance knowledge and experiences that enable developers to innovate a novel technique by combining each technique altogether or improve an existing technique.

The SOMARA-based mobile AR application is developed by implementing the Metaio AR SDK to perform AR tasks. The Metaio AR SDK uses SLAM that is a marker-less feature-based tracking technique. Exploiting an existing AR SDK is the quickest way to develop an application but it still has limitation such as tracking environments. The Metaio AR SDK provides a good quality tracking technique and tool to perform physical object tracking but the tracking module still has to work under controlled environments that there is no reflection or shadow. This can enhance an idea of developing a new tracking and recognition tracking that uses or adapts the proposed algorithms and techniques above.

Chapter III

3 Mobile AR Scenarios

AR has become a data representation tool that effectively enhances users' perceptions and experiences in visualizing computer-generated content in a real environment. AR on a mobile platform can be applied to indoor and outdoor applications such as location-based AR. Multimedia content of location-based AR can be visualized in a real environment depending on the user's location and AR tasks can be performed at any time and any place. The majority of mobile AR applications are currently based on closed platforms where the content acquisition has been limited and there are no communication channels between the application and other applications or existing open service providers. Mobile users are only able to view pre-designed media content that has been installed into the system in advance. This chapter presents the main requirements that innovate functionality and usability of typical standalone mobile AR applications. These requirements efficiently strengthen the design of SOMARA and development of the open mobile AR application. This novel mobile AR application presents supporting tasks that take advantage of an open architecture where a web service framework performs interoperable tasks and acquires value-added content from participating open providers. This acquired content can be effectively consumed in AR environments that enhance content utilization and AR environment interpretation. The proposed architecture transforms general mobile AR systems into novel mobile AR learning systems where active AR environments and media content can be saved and taken away for further studies at any time or place.

3.1 Architectural Requirements

There are three main requirements that can be implemented into a typical mobile AR platform in order to improve the adaptability and functionality of typical closed-platform mobile AR application and enrich the usability of these applications, including service-orientation, open content acquisition and personalization. These requirements support the design of SOMARA enabling the development of a novel mobile AR application that demonstrate the SOMARA concept with its service-orientation, and support for content acquisition and utilization in AR environments.

3.1.1 Service-orientation

SOA is an interoperable service platform that supports any application in order to access a network of open service providers as well as provide service interfaces and APIs for the clients in order to create connections to any potential open providers [95] [96]. SOA-based applications can effectively present various related data from other sources or potential providers and share their data among other mobile users via mobile or wireless networks. As a result, SOA creates collaboration among applications in a client-server architecture that does not depend on any specific location or application platforms. Therefore, closed platform mobile applications that deliver limited data to users are able to implement SOAs, APIs and service interfaces that apparently transform them to applications on an open platform. This framework and its implementation expand the functionality of mobile devices that have limited resources such as a small display and restricted processing speed. Taking advantages of a SOA results in creating cost-effective applications on mobile platforms that additionally require processing and data from other web service providers.

In order to create an open mobile AR application, the mobile development architecture potentially requires service-orientation that is the collaboration of a mobile AR platform and SOA. The designed service-orientation is generally composed of a web service framework, mobile client and service APIs. The following explains the details of the service-orientation and the components inside that support the development of a mobile AR application on an open platform.

Web service framework

The web service framework is an important component in a SOA that efficiently facilitates generating the open framework and accessing open services for a novel mobile AR system that is versatile and can be implemented in various scenarios such as a shopping environment, cultural objects exhibitions and AR-based learning. The web service framework performs back-end interoperable tasks in order to create connections between web service providers and the service request module working inside the client-side applications. In this research, a novel mobile AR application applies REST as a web service client framework in the service request module that utilizes the iOS web service library applied by XCode in order to access and send requests to the participating services. The web service framework exploits a mobile client and service providers.

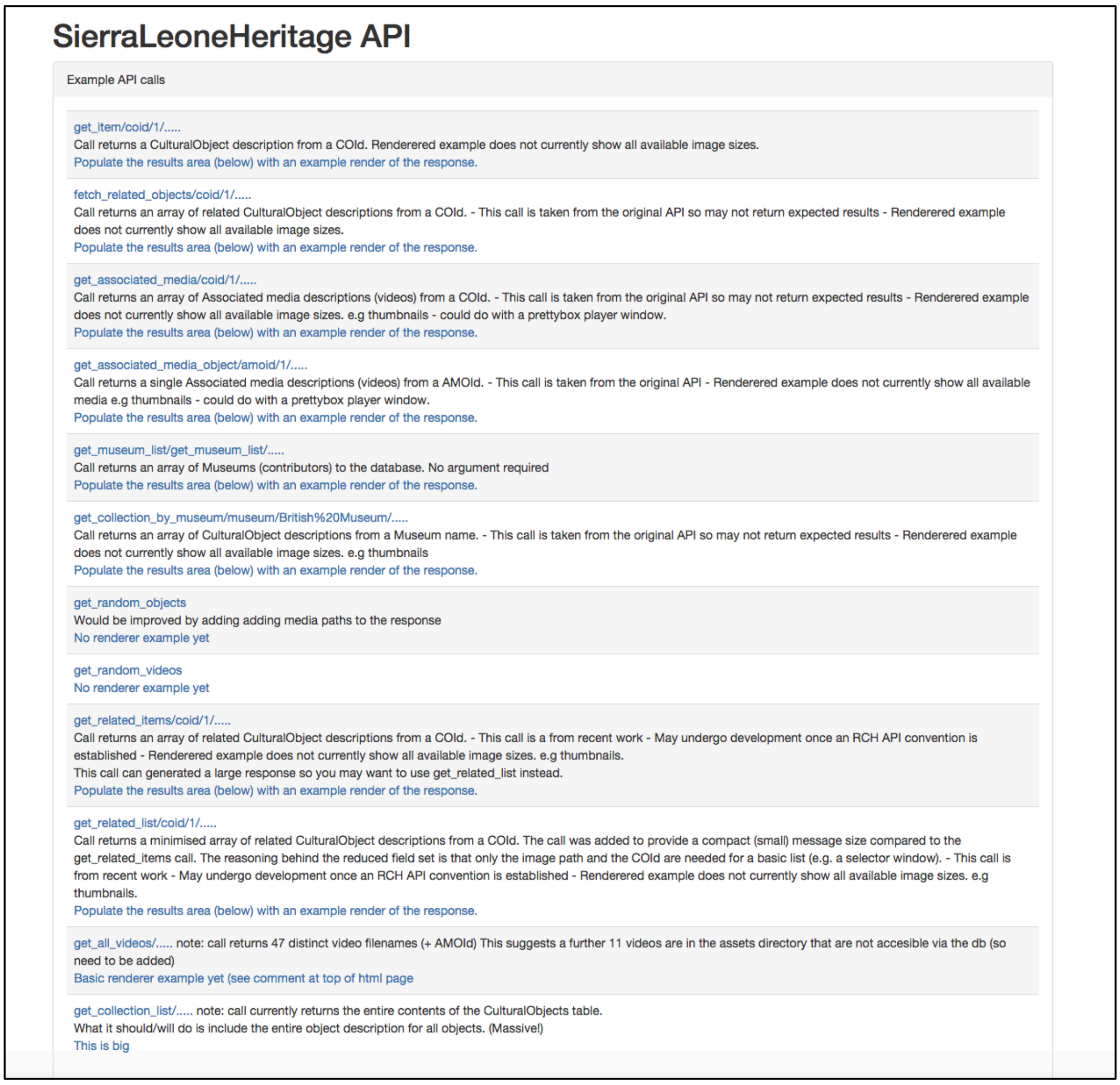
Mobile client

Client-side applications are created by implementing a web service framework into the mobile AR application. The application clients are able to access and interact with open services by applying potential service APIs into the service request module. They can then perform interoperable and content acquisition tasks by creating a connection to and from open service providers or participating servers. In addition, AR content can be shared via the open network and consumed by other AR applications as well [34][48]. Open mobile AR systems can also participate in distributed application development in order to perform pervasive computing and create distant AR services for mobile devices that require context and the location of the users [47][97]. Interoperable frameworks also offer a feature for location-based mobile AR, e.g. an AR browser that enables anywhere augmentation, a content-sharing framework and visualization in real-time of associated media content from AR providers on the see-through browser visualized in the real environment [82]. The web service framework has been used to work as a back-end system that creates connections for requesting services and receiving responses. The responses are passed to the AR application, processed and the data is then visualized and presented in the real scene. The web service framework essentially requires service APIs as service interfaces of participating providers that can be open services or data providers. The APIs present the URL of the data source along with parameters required for building a query in order to retrieve relevant data from the repository. The contributing APIs are applied into the service request modules in order to send a request along with necessarily required parameters to the providers. The potential service APIs for the client-side applications can be third party APIs such as Google Maps, Flickr, Facebook, etc. [81] or particular service APIs in the desired scenarios that are described in the following section.

The SOMARA-based application is an AR application on a mobile platform and web service framework specified to create connections to open services and perform interoperability tasks. This novel mobile AR application or mobile client is implemented on a camera-embedded mobile device that is able to perform AR tasks such as multiple object tracking, recognition and content visualization. In this research, the mobile client is prototyped on an iOS platform and run on an iPad where native development frameworks including XCode and Objective C are required. The mobile client offers a mobile interface for mobile users so that they can interact with active media content visualized on the screen as well as perform object tracking. The mobile client also contains an AR SDK that performs AR tasks in the AR application and simultaneously works with the mobile interface of the AR application as a front-end system for mobile users so that they can interact with the application and active content. In this research, the Metaio AR SDK [45] has been used to apply the mobile client to perform object tracking, recognition and content visualization.

Service providers

The mobile client requires service interfaces or APIs from participating open providers in SOA. Server-side applications offer data in their repository and services that can be requested through a web service framework and HTTP. In this research, the novel mobile AR application is applied in a MLA scenario and the application requires service APIs of the participating museum services and third parties. The application applies service APIs from existing museum content providers and a third party including the Reanimating Cultural Heritage (RCH) service API [98], the Victoria and Albert museum API [99] and the Google Maps APIs [100] as a proof of concept. These APIs are based on a REST platform and provide output in XML and JSON document format. Figure 3.1 presents the RCH service API and the details of services provided.



SierraLeoneHeritage API

Example API calls

get_item/coid/1/.....
Call returns a CulturalObject description from a COId. Renderered example does not currently show all available image sizes.
Populate the results area (below) with an example render of the response.

fetch_related_objects/coid/1/.....
Call returns an array of related CulturalObject descriptions from a COId. - This call is taken from the original API so may not return expected results - Renderered example does not currently show all available image sizes.
Populate the results area (below) with an example render of the response.

get_associated_media/coid/1/.....
Call returns an array of Associated media descriptions (videos) from a COId. - This call is taken from the original API so may not return expected results - Renderered example does not currently show all available image sizes. e.g thumbnails - could do with a prettybox player window.
Populate the results area (below) with an example render of the response.

get_associated_media_object/amoid/1/.....
Call returns a single Associated media descriptions (videos) from a AMOId. - This call is taken from the original API - Renderered example does not currently show all available media e.g thumbnails - could do with a prettybox player window.
Populate the results area (below) with an example render of the response.

get_museum_list/get_museum_list/.....
Call returns an array of Museums (contributors) to the database. No argument required
Populate the results area (below) with an example render of the response.

get_collection_by_museum/museum/British%20Museum/.....
Call returns an array of CulturalObject descriptions from a Museum name. - This call is taken from the original API so may not return expected results - Renderered example does not currently show all available image sizes. e.g thumbnails
Populate the results area (below) with an example render of the response.

get_random_objects
Would be improved by adding adding media paths to the response
No renderer example yet

get_random_videos
No renderer example yet

get_related_items/coid/1/.....
Call returns an array of related CulturalObject descriptions from a COId. - This call is a from recent work - May undergo development once an RCH API convention is established - Renderered example does not currently show all available image sizes. e.g thumbnails.
This call can generated a large response so you may want to use get_related_list instead.
Populate the results area (below) with an example render of the response.

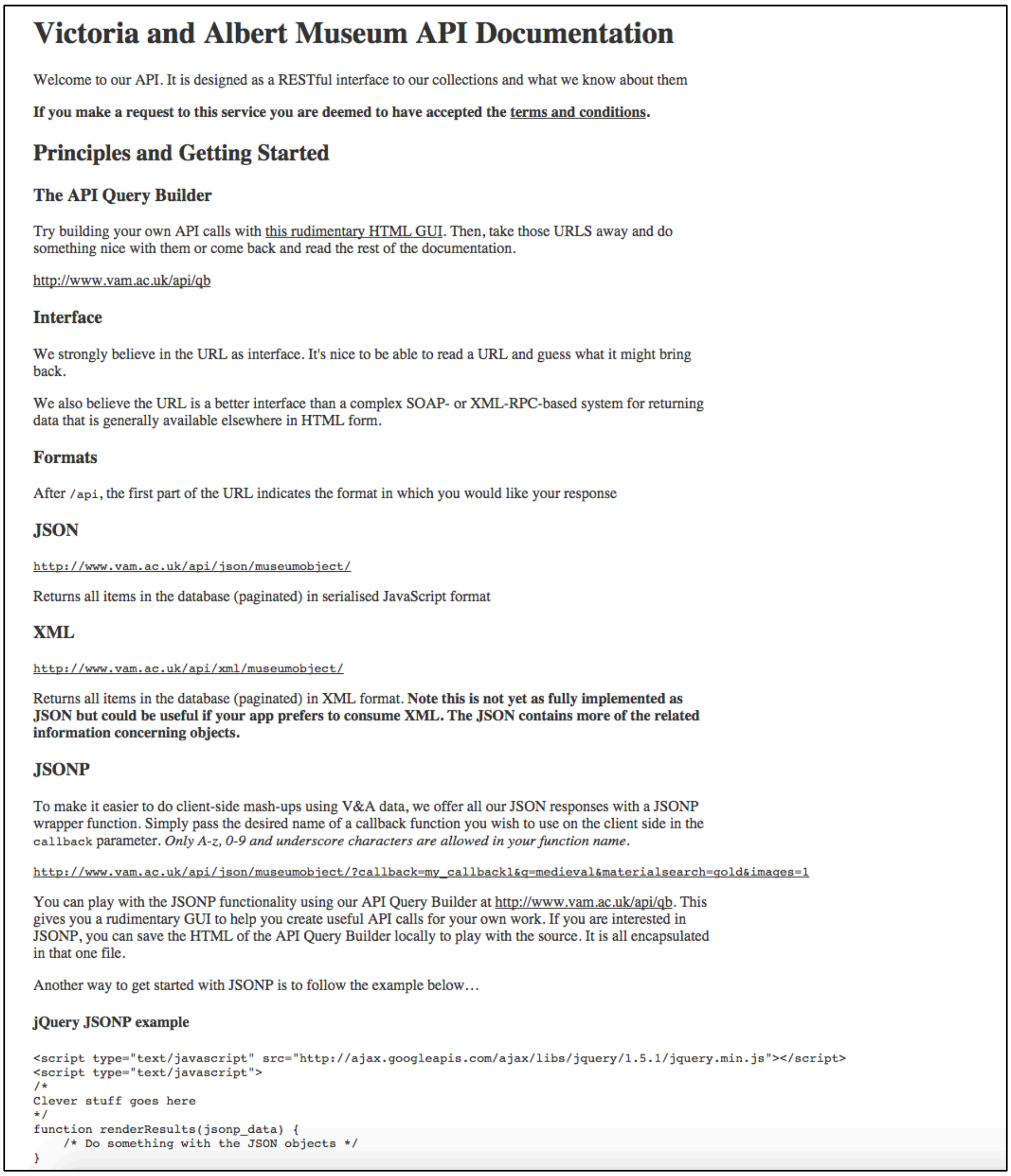
get_related_list/coid/1/.....
Call returns a minimised array of related CulturalObject descriptions from a COId. The call was added to provide a compact (small) message size compared to the get_related_items call. The reasoning behind the reduced field set is that only the image path and the COId are needed for a basic list (e.g. a selector window). - This call is from recent work - May undergo development once an RCH API convention is established - Renderered example does not currently show all available image sizes. e.g thumbnails.
Populate the results area (below) with an example render of the response.

get_all_videos/..... note: call returns 47 distinct video filenames (+ AMOId) This suggests a further 11 videos are in the assets directory that are not accesible via the db (so need to be added)
Basic renderer example yet (see comment at top of html page

get_collection_list/..... note: call currently returns the entire contents of the CulturalObjects table.
What it should/will do is include the entire object description for all objects. (Massive!)
This is big

Figure 3.1 SierraLeoneHeritage API [98]

Figure 3.2 shows the Victoria and Albert museum API documentation. The document explains how to start working with the API and the output formats that can be JSON or XML documents.



Victoria and Albert Museum API Documentation

Welcome to our API. It is designed as a RESTful interface to our collections and what we know about them

If you make a request to this service you are deemed to have accepted the terms and conditions.

Principles and Getting Started

The API Query Builder

Try building your own API calls with this rudimentary HTML GUI. Then, take those URLS away and do something nice with them or come back and read the rest of the documentation.

http://www.vam.ac.uk/api/qb

Interface

We strongly believe in the URL as interface. It's nice to be able to read a URL and guess what it might bring back.

We also believe the URL is a better interface than a complex SOAP- or XML-RPC-based system for returning data that is generally available elsewhere in HTML form.

Formats

After `/api`, the first part of the URL indicates the format in which you would like your response

JSON

`http://www.vam.ac.uk/api/json/museumobject/`

Returns all items in the database (paginated) in serialised JavaScript format

XML

`http://www.vam.ac.uk/api/xml/museumobject/`

Returns all items in the database (paginated) in XML format. **Note this is not yet as fully implemented as JSON but could be useful if your app prefers to consume XML. The JSON contains more of the related information concerning objects.**

JSONP

To make it easier to do client-side mash-ups using V&A data, we offer all our JSON responses with a JSONP wrapper function. Simply pass the desired name of a callback function you wish to use on the client side in the `callback` parameter. *Only A-z, 0-9 and underscore characters are allowed in your function name.*

`http://www.vam.ac.uk/api/json/museumobject/?callback=my_callback1&q=medieval&materialsearch=gold&images=1`

You can play with the JSONP functionality using our API Query Builder at http://www.vam.ac.uk/api/qb. This gives you a rudimentary GUI to help you create useful API calls for your own work. If you are interested in JSONP, you can save the HTML of the API Query Builder locally to play with the source. It is all encapsulated in that one file.

Another way to get started with JSONP is to follow the example below…

jQuery JSONP example

```
<script type="text/javascript" src="http://ajax.googleapis.com/ajax/libs/jquery/1.5.1/jquery.min.js"></script>
<script type="text/javascript">
/*
Clever stuff goes here
*/
function renderResults(jsonp_data) {
    /* Do something with the JSON objects */
}
```

Figure 3.2 Victoria and Albert Museum API [99]

Figure 3.3 presents the Google Maps APIs and the details of each service, which explains how to access and request location data.

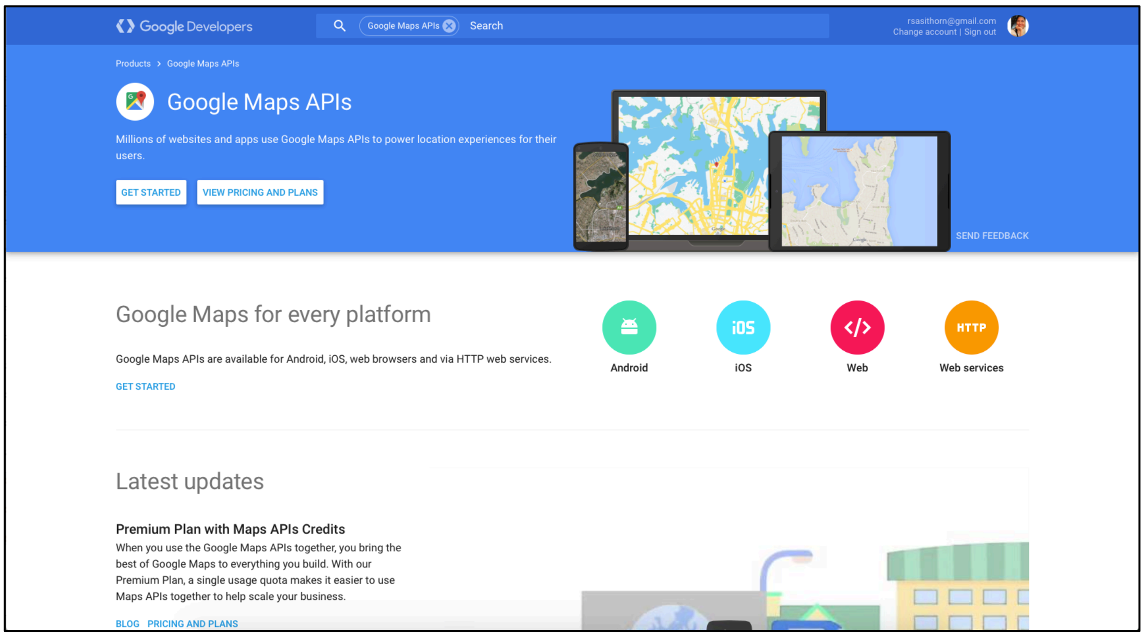

Google Developers
Google Maps APIs Search
rsasithorn@gmail.com
Change account | Sign out
Products > Google Maps APIs
Google Maps APIs
Millions of websites and apps use Google Maps APIs to power location experiences for their users.
GET STARTED VIEW PRICING AND PLANS
SEND FEEDBACK
Google Maps for every platform
Google Maps APIs are available for Android, iOS, web browsers and via HTTP web services.
GET STARTED
Android iOS Web Web services
Latest updates
Premium Plan with Maps APIs Credits
When you use the Google Maps APIs together, you bring the best of Google Maps to everything you build. With our Premium Plan, a single usage quota makes it easier to use Maps APIs together to help scale your business.
BLOG PRICING AND PLANS

Figure 3.3 Google Maps APIs [100]

3.1.2 Content acquisition

The key objective of SOA is allowing any application to access open services or applications and request processing or valuable data from these providers. Integrating SOA into a mobile AR platform results in acquiring new and valuable media content of a targeted object in real-time that could potentially be visualized in an AR environment. Open mobile AR applications are able to create connections and request associated content from existing potential providers who can be a third party or participating service providers in the anticipated scenarios. The content acquisition process can be done beneficially on a specific approach such as a photogrammetry service request where a mobile client is able to acquire a virtual 3D model of a preferred object from a web service-based photogrammetry provider. Photogrammetry or an image-based reconstruction service is a valuable technique for any mobile AR application that requires 3D models of preferred physical objects and then utilizes these obtained models in an AR environment. In this case, the acquisition process is accomplished by taking photographs of a targeted object and then directly sending a request along with the photographs through the web service framework to the photogrammetry provider. Currently, there are photogrammetry services on web-based platform such as ARC 3D [87] and CMP SfM [101] and on client-server platforms such as 123D Catch [7] and ReCap [43]. All of these services are closed platform applications and do not provide any APIs to other applications. Another approach is general content acquisition where the web service framework in a mobile AR client is effectively implemented in order to acquire valuable media content, e.g. 3D models, videos, audio, social networking data, etc. from existing open sources or third parties. The obtained content is then presented on the real scene along with existing content.

In this research, the SOMARA based mobile AR client is applied in an MLA scenario, and the content acquisition process is performed by creating connections and access to participating museum content providers. Moreover, the proposed photogrammetry service can be usefully applied in the mobile AR client in order to support 3D model reconstruction of a targeted object for an MLA. This feature is an advanced feature or strategy that could enhance a typical MLA and offer a content acquisition module to perform a function that provides some connection channels to open service providers and web service-based photogrammetry services. The mobile AR client is able to request associated media content from potential open providers and also has a function to support image-based reconstruction service request from photogrammetry service providers, although currently there is only one provider, Autodesk, named Recap API [102]. The Recap API is now on the beta version and does not allow the mobile AR client to use it.

3.1.3 AR environment personalization

Another aspect of the open mobile AR system that takes advantage of SOA and a mobile AR platform is AR environment personalization. Acquired content from the content acquisition function, web service framework and participating providers are not only visualized on the mobile device screen but also consumed in personal AR environments where mobile users can interact with the active content on the screen, such as selecting and saving preferred content relative to a targeted object so that the selected content can be taken away and revealed again in other situations. This feature supports AR content utilization that obtains or provides content overlaid on top of a real scene that can be saved and reused in personal AR environments where mobile users are able to review what they have experienced in other situations. This feature is adaptable and can effectively support learning as well as providing shopping experiences where active media content on the novel mobile AR application can be flexibly utilized and data representation can be presented differently by using personalized AR environments. Compared to general mobile AR applications on closed platforms where application providers do not allow mobile users to own their pre-designed media content, the web service-based mobile AR application is able to send a request for open content through the web service framework where the acquired content is transferred to the AR application and mobile interface. AR environment personalization is an advantageous strategy in mobile AR application development because mobile users are able to create their own AR environments by selecting content on the mobile device's screen and saving it as a personal museum exhibition. In the mobile AR client, after receiving associated content from open service providers or museum exhibition sources, the acquired content could be beneficially utilized by offering a personalization feature so that mobile visitors are able to generate personalized AR environments with this content.

Developing an open mobile AR application and supporting functions basically requires three important components: a mobile AR client, a web service framework and service providers. This research proposes SOMARA, composed of these three components incorporated on a mobile platform in the architecture. In SOMARA, the mobile client is an AR application on a mobile platform that integrates a web service framework into the system in order to perform AR tasks as well as simultaneously work with the mobile interface and AR application. The web service framework requires service interfaces or APIs of potential providers that can generate connections to the participating providers, send requests and receive responses. The responses are then processed and obtained content is sent back to be visualized on the mobile interface where mobile users are able to view and consume active content on the screen by selecting and creating personal AR environments.

3.2 Museum Learning Scenarios

SOMARA is an architecture based on a mobile platform and a web service framework. The application on SOMARA overcomes the limitation of closed platform architecture by creating interoperability between the application itself and other applications. The result of this is an open mobile AR application that is able to deliver collaborative components including content acquisition and consumption to mobile users as well as supporting the development of an open mobile AR system in many different scenarios, e.g. a shopping experience, mobile learning, MLAs, etc. In this research, SOMARA is applied in a museum scenario where mobile devices are effectively brought into the MLAs. SOMARA is exploited to develop novel open mobile AR applications that can enhance MLAs or cultural heritage environments and increase visitors' experiences and knowledge in cultural objects exhibited by augmenting a target object and presenting media content in the real scene. The application can create associations between participating museums and other potential open providers such as third parties through a web service framework and a mobile AR platform.

In general, museums exhibit cultural objects along with information that is illustrated or presented on a board, poster or monitor and visitors are allowed to read and view this information and content as well as taking leaflets provided. Most modern museums now offer interactive multimedia content on a touch-screen where visitors can experience or view information by interacting with media content through the user interfaces. Applying AR or mixed reality technology into MLAs is another level of improving the exhibition services that can enhance visitors' experiences in terms of improving the learning approaches in museum environments, drawing visitors' attention and enhancing understanding of the context of cultural objects as well as museum exhibits [38][103][104][105]. In addition, the web service-based mobile AR system improves participation between contributing museums and visitors are able to interact with active media content as well as obtain personal AR environments, including preferred cultural content of the exhibition that usefully encourages content acquisition, content utilization and MLAs in other situations. Figure 3.4 presents the use-case diagram of a general MLA where MLA activities are done within the physical museum and on the museum's website. Visitors are able to view physical objects, view content on the monitors and interact with content on the screen. In addition, MLAs can also be done through the museum's website where there are online catalogues of cultural artifacts that can be searched and viewed anytime by users.

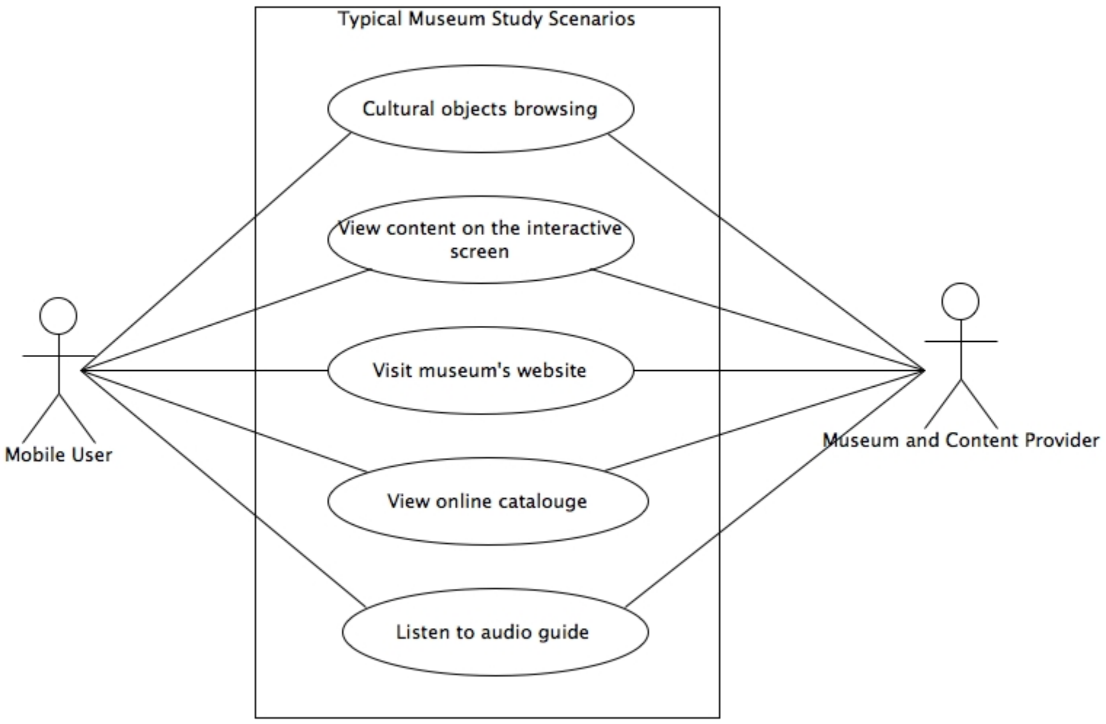

Typical Museum Study Scenarios
Cultural objects browsing
View content on the interactive screen
Visit museum's website
View online catalouge
Listen to audio guide
Mobile User
Museum and Content Provider

Figure 3.4 The use-case diagram of general museum learning

Currently, there are many museums that have developed a website providing an online catalogue of the cultural objects in their repository and it becomes another presentation channel so that visitors can also perform an MLA online. Each particular item can be searched and its information and metadata is shown on the web page. Examples of online museum exhibitions are the Victoria and Albert Museum [106], the British Museum [107] and Europeana [108]. Another example is a website that exhibits an online catalogue of cultural objects from Sierra Leone. The data repository of RCH [109] contains metadata of objects exhibited in many museums around the United Kingdom. These have shown that MLAs has already moved to another level of cultural data representation and communication. Moreover, the museums present an API or service interface in a SOAP or REST platform so that any client application is able to implement the API and then send a request along with parameters to the open cultural content service and receive a response in an XML or JSON document that can be read and presented on the application that requires the specific museum content as part of the outcome or the service interfaces. Therefore, applications on any platform are able to implement these APIs and request associated cultural content from the open museum repositories through the SOA. The acquired cultural content can then be presented and utilized on the application client with platform-independent features.

SOMARA supports content acquisition on a mobile AR platform and is applied in museum scenarios where associated museum content from open museum services is requested and overlaid on top of a real scene or AR environment. This architecture enables generating museum studies on a mobile AR platform using a mobile device. Cultural media content can be visualized on the screen in real environments by augmenting a real cultural object in a physical museum. Participating museums' APIs are required in this architecture in order to combine the APIs into the service request module and produce the output on the AR scene. In this research, SOMARA has been implemented in two associated MLA prototypes, which are museum-based and home-based learning. In the MLA scenarios, content acquisition modules in the mobile AR client are able to request associated content from particular museum exhibition sources or third party providers in order to visualize the acquired content on the museum visitors' screens. In addition, the content acquisition is not only from open museum repositories or museum services but SOMARA also supports service connections to useful open services that produce media content for MLAs in real-time. Photogrammetry or image-based reconstruction services benefit museum visitors by allowing them to request virtual 3D models of their preferred cultural objects. The visitors can then consume these acquired models by creating personal AR environments containing the models and other preferred media content. Figure 3.5 proposed the use-case diagram of SOMARA that supports MLAs.

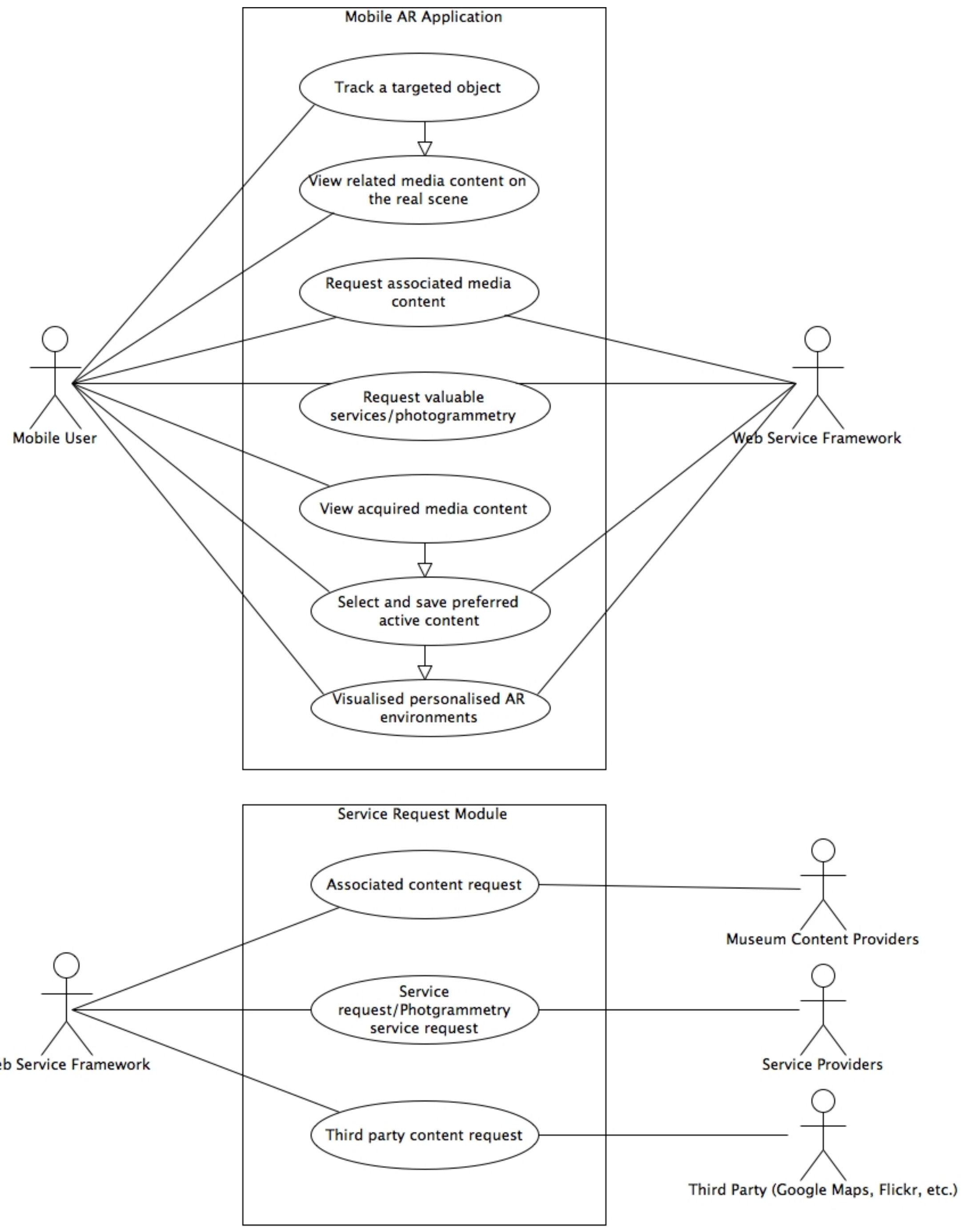

Mobile AR Application
Track a targeted object
View related media content on the real scene
Request associated media content
Request valuable services/photogrammetry
View acquired media content
Select and save preferred active content
Visualised personalised AR environments
Mobile User
Web Service Framework
Service Request Module
Associated content request
Service request/Photgrammetry service request
Third party content request
Web Service Framework
Museum Content Providers
Service Providers
Third Party (Google Maps, Flickr, etc.)

Figure 3.5 SOMARA use-case diagram

The following are the proposed museum scenarios applied by the SOMARA and its application. Both scenarios enhance typical museum studies in different situations by offering visitors a novel mobile AR application that supports content acquisition and utilization. As the result, MLAs can be done inside or outside the museum and cultural content can be acquired from open services.

3.2.1 Museum-based learning scenario

The proposed mobile AR system effectively enhances physical MLAs by offering an AR application on mobile devices, e.g. an iPad, to visitors in order to present related media content of a targeted cultural object that can be virtual text, images, videos, 3D models, etc. superimposed on top of a cultural object or a real scene. This content in an AR environment enhances the learning experience and encourages visitors to engage in museum-learning environments where supporting content can be seen through a mobile interface. In addition, the open mobile AR application also offers some encouraging features that take advantage of SOA and existing open providers. As a result, it benefits MLAs and visitors in terms of content acquisition and consumption where the museum itself can present associated content from other participating museums or third parties to visitors as well as offering valuable services such as image-based reconstruction that enables visitors to acquire 3D models of their preferred objects. The obtained content can then be utilized by saving them in a personal AR environment along with other selected content.

SOMARA and its application enhances typical museum studies where an exhibition can provide value-added services on mobile devices and AR environments that allow visitors to view and take away acquired content of preferred objects. The AR system is used as a data representation tool in order to visualize associated media content of a targeted object in a real scene and allows mobile users to send a request and interact with active content. The mobile AR application for MLA scenarios requires marker-less or object-based tracking where a cultural object exhibited is tracked, recognized and its related content is visualized on the mobile's screen. As a result, MLA enhances visitors' experiences as it offers valuable web service-based content and services as a strategy to support learning and content utilization through the SOA and its components. Additionally, the museum is able to provide a value-added mobile learning application that aggregates content from the museum itself, participating museums and other third parties.

3.2.2 Home-based learning scenario

Museum visitors mostly go to museums to learn about historical incidents and cultural objects exhibited. To experience such a different environment and knowledge, visitors have to go the museum themselves and browse the exhibitions inside. This can be done easily if the visitors go to a nearby or local museum. However, MLAs may limit other people who live far away and require resources but are unable to visit the museums physically. In addition, museums generally do not allow visitors to perform any other supporting learning activities such as taking photos of a preferred object or saving multimedia content presented on a monitor and taking them away for viewing in other situations. SOMARA usefully encourages media content utilization and supports MLAs outside the museum or in home-based MLAs. In the novel mobile AR application, there is a photogrammetry service request module that provides a content acquisition feature; museum visitors are able to request a model of a targeted object and then save it into a personal AR environment, which can then be revealed or presented in a home-based learning scenario. The home-based learning can be performed by a mobile user in other situations outside the museum. Consequently, museum learning and exhibition browsing do not need to be done in the physical museum as the mobile AR application offers a home-based learning function whereby mobile users are able to pursue the MLAs as well as view personal AR environments at home or school. The system requires marker-less or image-based tracking in order to track and recognize a trigger image and then augment cultural media content of the particular cultural object in the museum. The trigger images can be printed images or leaflets of targeted objects and they can be downloaded from the museum's website. As the result, home-based learning and its process can be done anywhere and at any time outside the museum, where mobile users are able to pursue remote MLAs via a mobile/wireless network. To do so, the system does not store any content of targeted objects and the mobile client requests related media content of a targeted object at the time the system recognizes its trigger image.

The system first starts working with a tracking configuration file that sets up the tracking and recognition process with an XML file containing targeted marker-less tracking objects. Then, the mobile AR application can send the request for relevant content of a tracked image or object to the participating server. In museum environment scenarios, the mobile client requires service APIs of open web service servers that offer cultural content belonging to a particular museum. Mobile users can then pursue MLAs by requesting related virtual content from the server and then visualize them on an AR scene by using a printed image or leaflet of a targeted artifact as a trigger object. Figure 3.6 illustrates the use-case diagram of home-based MLAs where mobile users are able to perform MLAs outside the museum by using the novel mobile AR application to augment trigger images instead of real physical objects. The related content is obtained from the open museum content provider and visualized on the real scene.

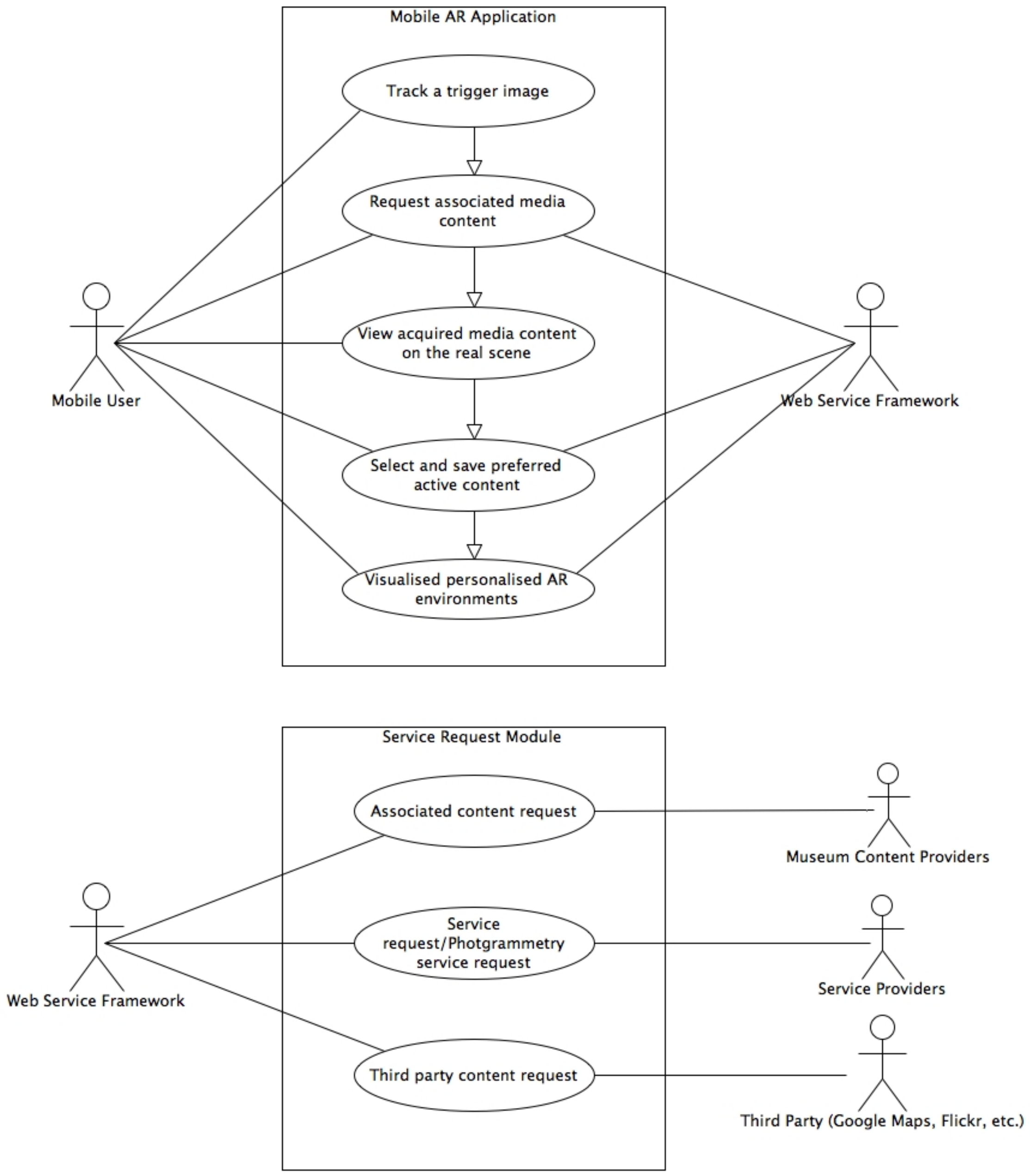

Mobile AR Application
Track a trigger image
Request associated media content
View acquired media content on the real scene
Select and save preferred active content
Visualised personalised AR environments
Mobile User
Web Service Framework
Service Request Module
Associated content request
Service request/Photgrammetry service request
Third party content request
Web Service Framework
Museum Content Providers
Service Providers
Third Party (Google Maps, Flickr, etc.)

Figure 3.6 Home-based museum learning use-case diagram

3.3 System Specifications

The set of system specifications are developed based on the research question and description of the potential benefits that strongly supports the creation of SOMARA and the service-orientation. These system specifications coupled with the background literature survey developed in Chapter 2 and the more detailed scenarios discussed in this section lead to the user requirements for SOMARA described in Chapter 4.

- This thesis proposes a mobile AR application developed to demonstrate the efficacy of SOA (in order to illustrate the improvements expected on an open platform based on mobile AR system when compared to closed AR platforms) by implementing a web service framework as a client-server system. This novel open mobile AR application is able to obtain more valuable content from open content providers or service providers to extensively enhance the usability and functionality of current mobile AR systems.
- A Service Oriented Mobile AR Architecture (SOMARA) is developed to demonstrate content acquisition via mobile or wireless networks and content utilization through an AR client-server environment. The associated technology features (SOA, mobile, wireless, client-server) are exploited on an open platform and AR framework to illustrate aggregation of valuable media content from other sources, where those acquired content are consumed in the AR platform by mobile users.
- The design of the SOMARA application in this thesis is based on an open mobile AR system that implements or integrates service APIs from participating providers. The novel mobile AR application is able to perform typical AR tasks and interoperability tasks such as multiple object tracking, sending a request and receiving a response for associated content of a targeted object and potential services that enhance the feature of the novel mobile AR client and bring supportive content onto mobile AR environments.
- The SOMARA application is designed to be adaptable and could effectively be applied in different scenarios such as learning, shopping and museum experiences. In this research, SOMARA is applied to 'museum interactive scenarios' to demonstrate that an open mobile AR application can efficiently support virtual exhibitions based on AR environments displaying museum content that can be requested from participating content providers and third party.
- A final technology objective is to demonstrate the scalability of SOMARA through service-orientation by providing example supportive content acquisition and utilization modules, e.g. a photogrammetry service request module and an

AR environment personalization module that are integrated with SOMARA and other novel mobile AR applications. The photogrammetry or image-based reconstruction service request module allows mobile users to simply acquire a 3D model of a targeted object and then create a personal AR environment of the model along with other preferred media content visualized on the screen. The personal AR environment can be taken away and revealed in other situations such as a 'take home museum exhibition'.

3.4 Summary

This chapter explains architectural requirements that encourage the development of an open mobile AR platform. Three main requirements that support the creation of SOMARA are service-orientation, content acquisition and AR environment personalization. A SOMARA-based mobile AR application is applied in MLAs where museum visitors are able to view cultural artifacts and request related content through mobile AR environments and a web service framework. The application supports two MLA scenarios including museum-based and home-based learning. These scenarios enhance learning activities that can be performed inside the museum or at home.

Chapter IV

4 Architectural Design

This chapter explains the structure of the Service-Oriented Mobile AR Architecture (SOMARA), its components and its service orientation that is specifically designed to support efficient content acquisition and utilization on mobile AR platforms. SOMARA improves on the usability and adaptability of general mobile AR applications for closed platforms, which limit the acquisition, consumption and visualization of digital media content, e.g. user generated content. Any novel mobile AR application based on SOMARA exists on an open platform and has a mobile web service framework as a back-end system. SOMARA specifically allows for the inclusion of user generated content from open service providers and their APIs. These could be third party or participating providers, depending on the desired scenarios and the APIs to be used in the AR application. Open mobile AR applications are able to connect to open providers via specifically implemented service interfaces (of these service providers). These interfaces are held in the service request module along with details of the parameters that they require. A service request will be sent through a web service framework to the service provider when mobile users or the system want to acquire media content from other sources. Application clients on any SOA are generally platform independent: any client on any platform can perform request-response tasks and data acquisition between service providers and application clients. Figure 4.1 presents SOMARA and its components as a client-server architecture. The architecture shows the web service framework working simultaneously with the AR application and other components in order to create service requests and receive responses. This is achieved by directly implementing the service APIs of participating service providers.

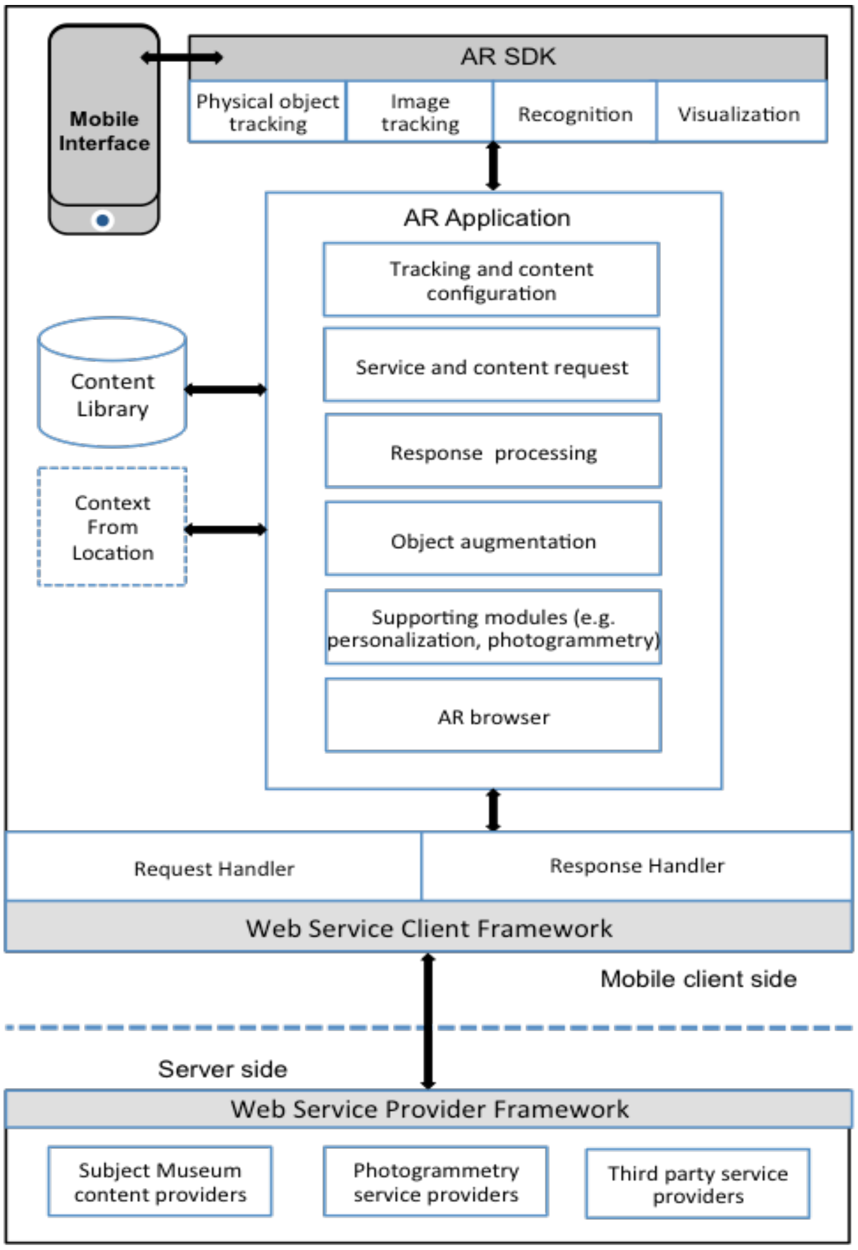

AR SDK
Mobile Interface
Physical object tracking
Image tracking
Recognition
Visualization
AR Application
Tracking and content configuration
Service and content request
Response processing
Object augmentation
Supporting modules (e.g. personalization, photogrammetry)
AR browser
Content Library
Context From Location
Request Handler
Response Handler
Web Service Client Framework
Mobile client side
Server side
Web Service Provider Framework
Subject Museum content providers
Photogrammetry service providers
Third party service providers

Figure 4.1 Service-Oriented Mobile AR Architecture

The SOMARA creates a service-orientation mainly composed of a mobile client, a web service framework and open service providers' APIs. It is based on a client-server architecture and the mobile/wireless framework. The following describes each significant component within SOMARA. These work together, in real-time, in order to accomplish AR tasks and interoperability tasks. In addition, there are some use-case diagrams illustrating the interaction between mobile users and the system through the service interface.

4.1 Mobile Client

The mobile client is an open AR application developed on a mobile platform and using a web service framework. The mobile client is able to perform interaction tasks, AR tasks and interoperability tasks in order to support users' interactions, content acquisitions, and content utilizations. The mobile AR client itself is composed of an AR application developed on top of a web service-based framework and service interfaces. These facilitate interoperable AR systems by simultaneously working with all the relevant open service providers in order to create connections and request content. The acquired media content will be visualized via an AR environment. Therefore, the mobile client is able to send requests for high-value content that can be visualized on the mobile AR system and superimposed on top of the real scene. The media content involved could be 3D objects, animations, images, maps, videos, tags, billboards, etc. These high-value content can very effectively extend and improve the quality of the experience provided by the mobile AR system in many different scenarios: shopping, MLAs or other scenarios such as personal assistance, or personalized AR systems.

The mobile client performs AR tasks that utilize and incorporate mobile device features such as embedded cameras and high quality touch-screens. These features are used to accomplish tasks such as, tracking, visualization and interaction between mobile users, references to physical objects (marker or marker-less) and computer-generated content. This mobile AR client framework is also designed for enhancing the data acquisition and collaboration features already present in conventional mobile AR applications - on closed platforms. Implementing a SOA on a mobile AR platform enables a mobile AR application to generate connections and acquire valuable digital content from open services or third party providers. The mobile AR client, in addition, offers features that support content acquisition and utilization facilities – the former including, for example, photogrammetry services and AR environment personalization. These features represent one of the benefits of using an open mobile AR platform. The mobile client can deliver valuable content to mobile AR environments as well as representing AR environment interpretations so that active content can be consumed and reused by mobile users. The mobile interface, mobile AR SDK, AR application are components of SOMARA mobile client that work concurrently to accomplish an AR presentation.

4.1.1 Mobile interface

A mobile AR system will usually offers just basic features for performing object tracking, recognition and visualization in relation to referenced objects. Related content of a tracked object will be rendered and visualized on the screen through the mobile interface. A mobile interface is a front-end system whereby mobile users can interact with the mobile AR application as well as with other mobile users and the system. The mobile AR application requires a touch-screen user interface with which mobile users can interact with the system, view visualized content and have interactions with those active content. In addition, the process of object tracking is done via an embedded camera on the mobile device. A targeted object can be viewed on the screen and, at the same time, the relevant content will be retrieved by the AR application and visualized by the AR SDK. The designated interactions between users and active content, which can be performed via the mobile interface, are selecting, and saving. This latter functionality can be performed on selected, predefined content or on content obtained from services - so that these content can be displayed again in other situations. The mobile AR application is developed on an iOS platform and the design is focused on the iPad device since this has a large screen as compared to the iPhone.

4.1.2 AR application

Mobile AR systems work with the embedded-camera, sensors, computer vision techniques and etc. in order to achieve object detection, tracking and content visualization. Using a built-in camera and/or GPS module in order to capture real environments and/or the user's location provides the AR system with the information necessary for performing object tracking, visualization and overlaying (virtual content on the real scene). The real-world scene could relate to either an indoor or an outdoor environment. For this research, the Metaio native AR SDK will be implemented within the AR system module to perform 3D tracking, recognition of target objects and visualization of relevant virtual content.

Metaio SDK is the native platform designed for mobile AR application development, and includes xCode/Objective-C for iOS and JavaScript for Android. The Metiao native SDK performs basic AR tasks including object tracking, recognition and visualization of objects (that the system will retrieve virtual content for, in order to present these to users). Metaio native SDK provides specific libraries, frameworks and APIs that developers can use to create mobile applications specific to their own platforms. These APIs fully support AR environments, in terms of hardware access, tracking processes, user interface design, content rendering and visualization.

For this research, the mobile AR application will be developed on a native iOS platform by implementing Metaio AR SDK for iOS into the AR application. Figure 4.2 presents the structure of the native Metaio AR SDK, which provides basic AR tasks to developers.

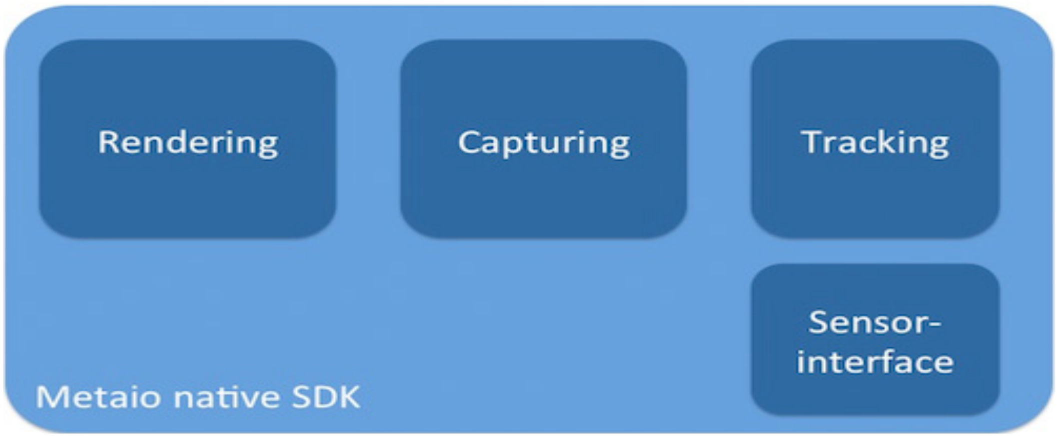


Figure 4.2 Metaio native AR SDK (www.metaio.com)

An AR application on the mobile AR client works with the AR SDK in order to simultaneously perform AR tasks and supporting tasks, thus creating interaction between mobile users and media content. In addition, the application will include web service connection modules, which will be able to request associated content or services. Moreover, the AR application will also include some features that will enhance AR content utilization, allowing mobile users to perform additional tasks that could be designed into the AR application so making it unique.

The novel AR application is built on top of a mobile and web service platform. Thus, some functions are available to it simply by connecting to existing providers through a web service. Therefore, the AR application strongly supports the mobile AR client to become an interactive open system that works in different ways with different special service providers in order to acquire virtual objects - e.g. image-based reconstructions or other such services. Standalone AR applications, on the other hand, present only a predefined and limited number of content to mobile users.

Interoperable frameworks and web service providers nowadays also support connections between mobile clients and provide data that can be presented and utilized on AR platforms, e.g. social network data, geospatial content, POIs data and mix reality content. As a result, mobile AR systems can become mobile web service clients by working with the web service client module in order to create connections with different service and content providers. Thus, an application can send very specific requests for additional content. The final outcome from these requests, which will be effectively overlaid on top of images of physical objects and real scenes, could be virtual objects, games, map data, videos, tags or billboards, etc. These can very effectively extend the AR system, whether it is related to shopping, or museum experiences, or is a personal assistance and personalized AR system.

Within the mobile AR application, there are a number of primary functions that will perform basic AR tasks. These simultaneously work with the Metaio AR SDK, and they include object tracking, object recognition and content visualization. In addition to this, the system also interacts with, and performs connection and content manipulation tasks for, mobile users in order for the application to provide its innovative features and to provide an interoperability framework in terms of the mobile platform and the AR environment.

The AR tasks are generally performed by the AR application whilst the AR SDK performs the task of recognizing reference objects; the system can thus present computer-generated content to accompany the objects. At the same time, the system also enables interaction between mobile users and active content so that users will be able to obtain possibly related information, which interrelates with content on the screen. Additionally, the system offers usefully featured functions that rely on the interoperability framework and the web service basis. These features work with users to quickly and smoothly accomplish connections to providers via the mobile network. The final outcomes from these services will be utilized on the mobile client. Mobile users will be able to personalize AR environments by selecting and saving preferred content on the screen.

In this research, a novel mobile AR application scenario exploiting service-orientation will be implemented in relation to museum experiences. This will be beneficial to its mobile users as a visual strategy to attract the user's attention with environmentally augmented hints, virtual exhibitions, augmented imagery, and other graphical information. Moreover, the system strongly engages mobile users. It does so by enabling the creation of AR preferences, such as personal AR environments, and by the presence of virtual experiences that can be launched and viewed in various situations. The following describes each component of this AR application. These components work concurrently together in order to perform AR tasks as well as interoperability tasks.

Tracking and content configuration

When a user wants to track a marker or marker-less object and the relevant camera view appears after she or he touches the tracking button, tracking configuration and content configuration will be performed, in concert, by the AR application itself and the AR SDK. These processes will identify all of the reference objects and their related content. These objects can then be tracked and recognized and the related content retrieved and visualized, overlaying the tracked objects on the image of the real-world scene. The mobile AR client supports physical object tracking as well as image-based tracking. This will be performed in two difference MLA-related scenarios.

For MLAs, the configuration process starts working when the system initiates the AR scene. The system has to classify all the reference objects that will be tracked and recognized via the tracking process. Each reference object has particular related media content, associated with it, which need to be identified in parallel with the tracking configuration. There are two methods of obtaining content related to reference objects: the cached and non-cached methods. For a mobile AR application that has a fixed number of related content prepared and cached in advance, a configuration file of the content is used to inform the system of each group of content that belong to each targeted object. The non-cached method is used when the system does not hold any related content itself, on the other hand, there is a data repository or open server that the application can refer to using a web service in order to obtain the related content for referenced objects. In this scenario, the configuration file will be composed of the service APIs of the relevant open content providers. For example, a museum has an open server maintaining data and meta-data relating to its cultural artifacts and presents these on a website. The mobile AR client is able to use its (the web-site's) APIs to acquire particular media content for a target object in the physical museum and then present the received content via the AR environment. Figure 4.3 illustrates the tracking and content configuration process performed by the AR application that will initially identify particular objects and retrieve their related content via this tracking and augmenting process.

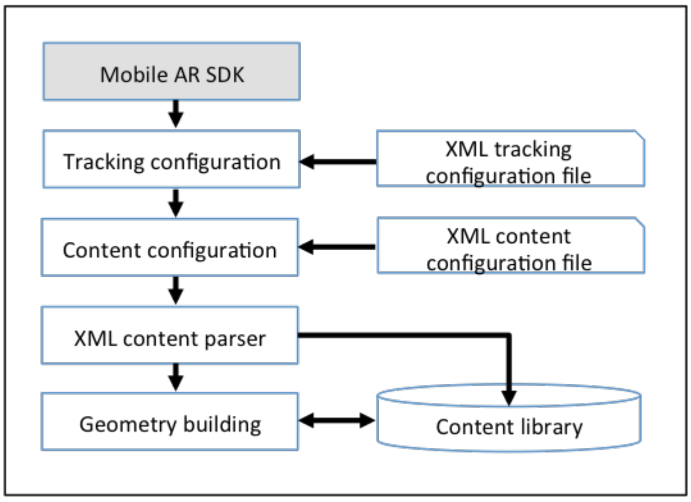


Figure 4.3 Tracking and content configuration process

Figure 4.4 presents the non-cached tracking configuration process. This, first, accesses the XML tracking configuration, and the related API configuration file so that the system can identify all of the reference objects and the participating APIs for each object. This latter is a specification of a URL and the parameters it requires, linking to the source of media content to each object.

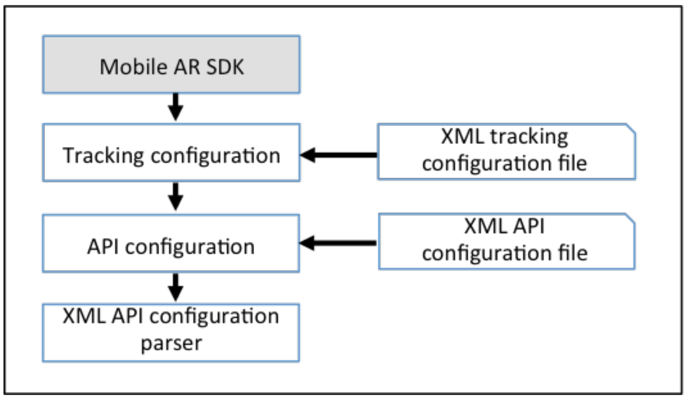

Mobile AR SDK
Tracking configuration
XML tracking configuration file
API configuration
XML API configuration file
XML API configuration parser

Figure 4.4 Tracking and API configuration process

The non-cached method could be applied in both museum and home-based scenarios. In the physical-museum MLA scenario, the architecture also offers the solution that the mobile client doesn't need to store the content that are related to the reference objects. It can, instead, request these content from participating content providers, such as museum subject content providers, depending on what is best for the circumstances. In the home-based MLA scenario, the mobile client performs image tracking after the camera view appears. In order to do this, the system requires the tracking configuration so that it can identify the reference marker-less or image objects, which must be tracked. It will also require the associated service APIs configuration that will identify the service API for each reference object. If the mobile client tracks and recognizes an image, the related APIs will be used and the associated content obtained and then visualized on the screen.

Service and content request

The mobile AR application supports both museum and home-based scenarios. Both of them can perform service and content requests in order to acquire related or associated content from participating open service or third party providers. The service and content request module works simultaneously with the web service framework in order to send requests and receive responses. In this module, the service interfaces or APIs of the providers are required to be present so that a web service request, which is in the format of a URL, can be sent along with its necessary parameters, to the participating providers via mobile/wireless network. In the physical-museum MLA scenario, the module is called when a visitor wishes to view the associated content for a targeted object. To do so, the system will access the participating API configuration file, which contains the associated APIs for each reference object, in order to retrieve the API for the object. Figure 4.5 shows a use case diagram of the associated content request process and the visualization of acquired content as performed by the mobile AR client for a museum visitor and through the mobile interface of the AR application.

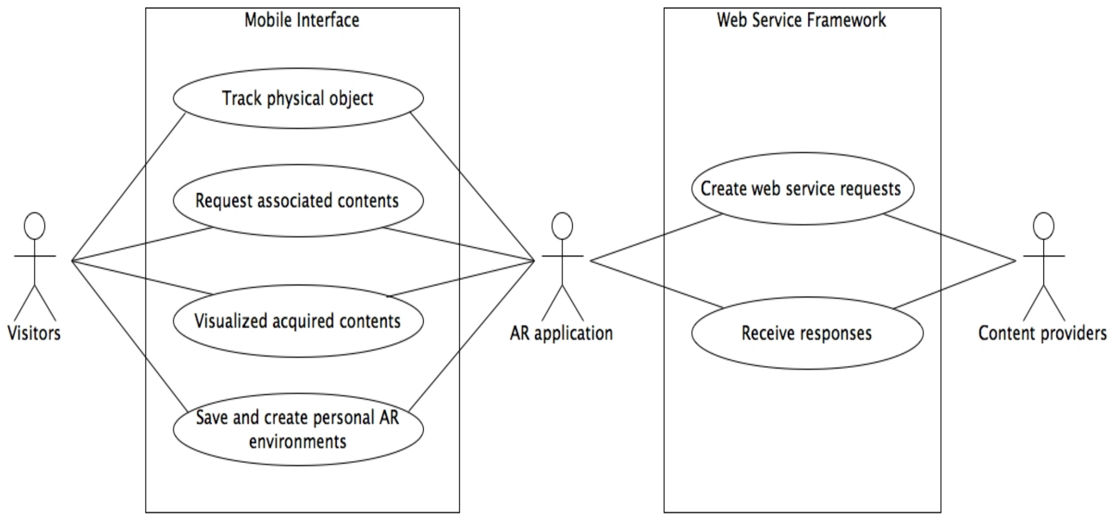

Mobile Interface
Track physical object
Request associated contents
Visualized acquired contents
Save and create personal AR environments
Visitors
AR application
Web Service Framework
Create web service requests
Receive responses
Content providers

Figure 4.5 The process of tracking and content acquisition in museum-based scenario

Acquiring virtual content from interoperable networks and open service providers can be achieved via the service request modules of the AR application. These modules include the service interfaces of the participating open providers. The providers could be museum content providers or third parties. The interface specifications include information concerning the parameters, which are required. In the museum-based scenario, a visitor is able to request associated content whenever they want by touching the sign or active object on the mobile interface. The application will then send the request along with its necessary parameters to the provider and receive the response in XML or JSON format. The received content will be visualized on the active AR environment. Figure 4.6 shows the process of service and content requests undertaken by the AR application when a mobile user requests associated content from an open server.

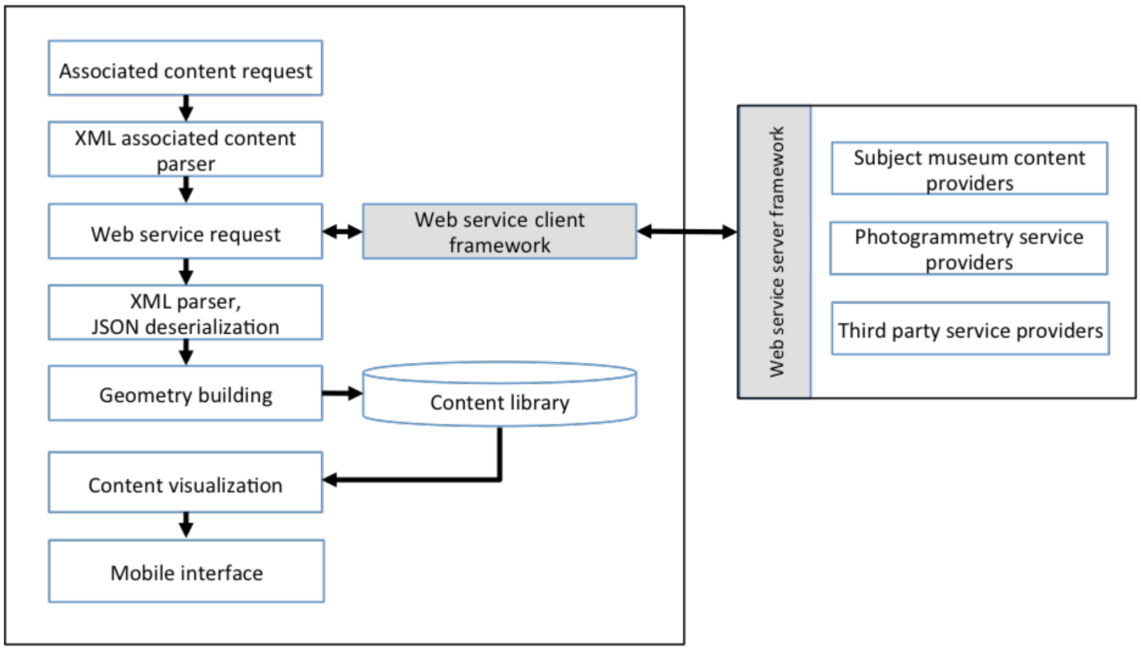

Associated content request
XML associated content parser
Web service request
Web service client framework
XML parser, JSON deserialization
Geometry building
Content library
Content visualization
Mobile interface
Web service server framework
Subject museum content providers
Photogrammetry service providers
Third party service providers

Figure 4.6 The process of service and content requests

In the home-based learning scenario, mobile users are able to perform MLAs outside the museum by tracking an image of a targeted object. The AR application will work on the non-cached system whereby the application has to obtain media content from participating open content providers by sending a request to the URL retrieved from the API configuration file. The content requests will be automatically performed when a mobile user tracks a trigger image and the system recognizes that image. The augmentation will be accomplished by the application sending a service request off to the provider through the web service framework and, of course, then receiving and processing the response. The related content of the trigger image will be visualized on the screen through the mobile interface. Figure 4.7 illustrates the use-case diagram of the home-based MLA scenario, and the content request process whereby a mobile user is able to track a trigger image in order to view relevant media content for the selected object.

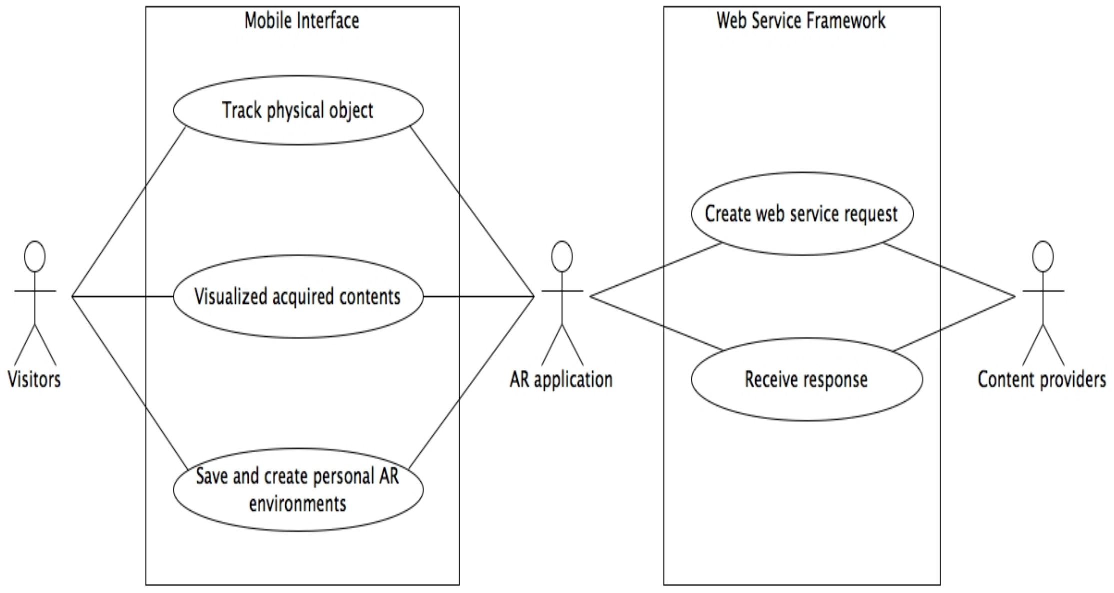


Figure 4.7 The process of tracking and content acquisition in home-based scenario

Both museum and home-based learning scenarios offer a feature allowing mobile users to effectively interact with active content on the screen by selecting and saving the content and thus creating personal AR environments via the mobile interface. The preferred museum content can be presented in other situations that enable users to conveniently review these AR images. The detail of the processes involved with content consumption is explained below in the supporting modules section.

Figure 4.8 presents an AR application performing content and service requests via the non-cached system applicable to home-based MLAs. After the object tracking and recognition phase the system will request related content in advance. This is to be ready for when user tracks the trigger image for the first time — otherwise there would be no geometry for the associated content stored in the content library.

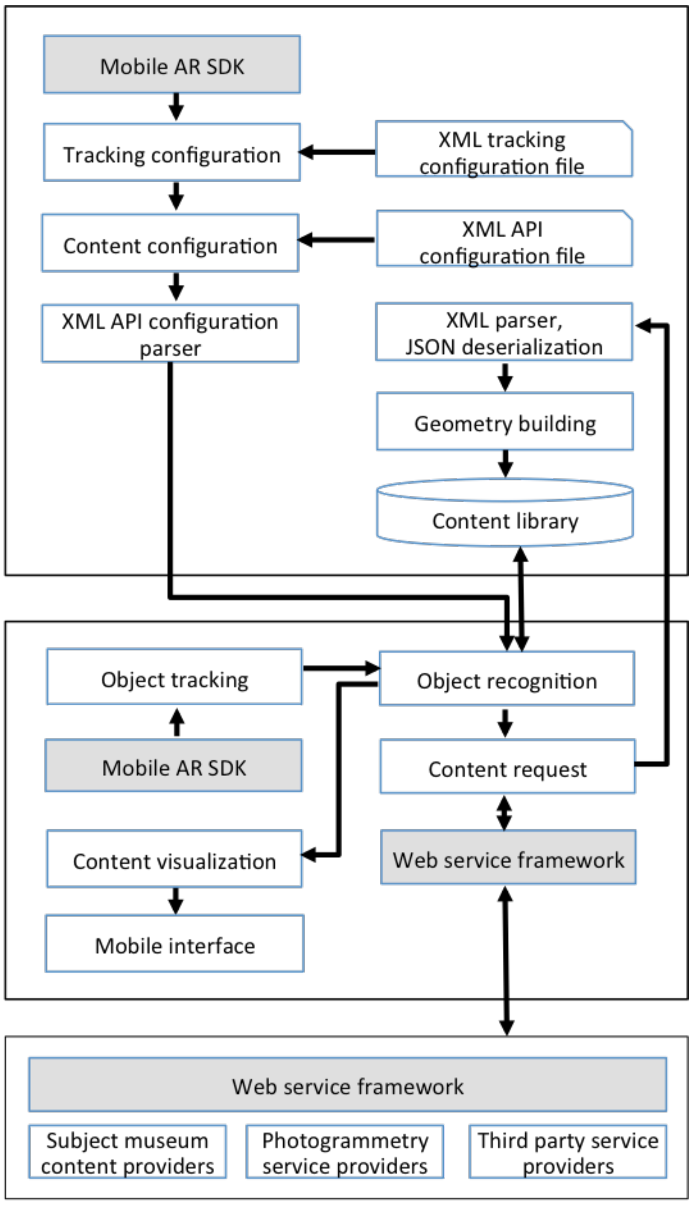


Figure 4.8 The service and content request in the home-based museum learning

Response processing

Service APIs provide the processing and return the outcome in relation to either XML or JSON document format. The service request URL relates to the source of the data. It is composed of the API, plus the parameters and output format dictated by the application client. The output document from the open service or other content provider is in either XML or JSON format and will be sent through the web service framework back to the mobile client. The AR application has to read and extract data from the response file. The native iOS development framework provides a function for developers to develop a reader for XML and JSON documents, which includes

NSXMLParser and NSJSONSerialization. Figure 4.9 presents the procedure for processing the JSON document response by utilizing the NSJSONSerialization function. The extracted data is transformed into objects and geometries, which will be kept in the content library of the AR application. The geometries will be retrieved and visualized on the AR scene when their reference object is tracked and recognized.

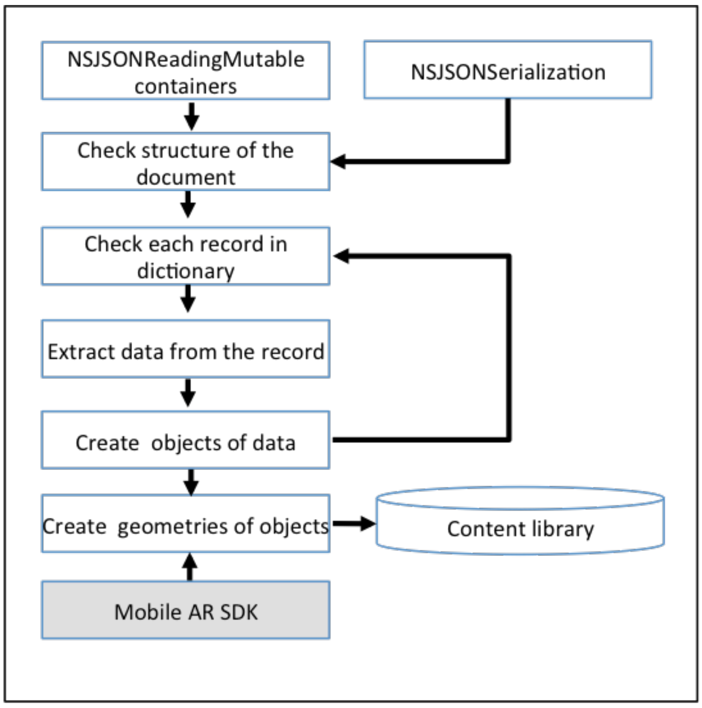

NSJSONReadingMutable containers
NSJSONSerialization
Check structure of the document
Check each record in dictionary
Extract data from the record
Create objects of data
Create geometries of objects
Content library
Mobile AR SDK

Figure 4.9 JSON document processing

Figure 4.10 shows the procedure for processing a response document in XML format by utilizing NSXMLParser. All of the elements contained in the XML document will be read through and the relevant data extracted from the document. The geometries for the data will be created and stored in the content library as for process related to NSJSONSerializaton.

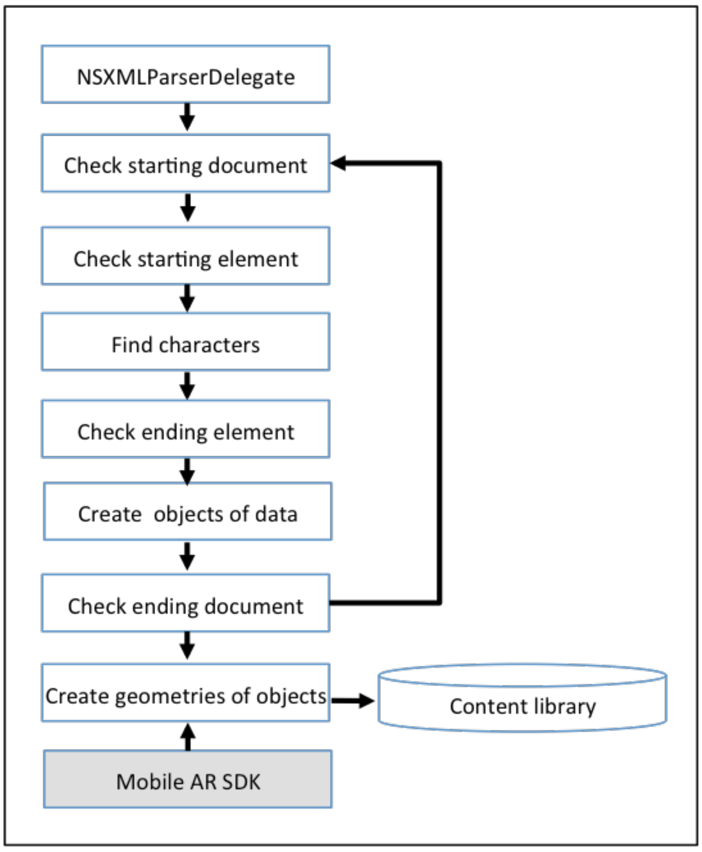


Figure 4.10 The process of reading the XML response document

Object augmentation

Object augmentation is the process of tracking objects and superimposing relevant media content associated with the targeted objects onto the real-world image. The mobile client starts tracking the real-world scene when the camera screen is presented and the mobile device user points a mobile client at the scene. When the system starts tracking the scene, the AR application will automatically work with the Metaio AR SDK in order to detect and recognize reference objects. Multiple content for a tracked object will be retrieved, aggregated and presented on top of the tracked object in the real-world scene/image. The proposed MLA scenarios mainly rely on this object augmentation in order to present media content - as geometries or objects, which can be seen on the screen.

In the museum-based learning scenario, the mobile AR client initially stores related content of reference objects in the content library that these can be retrieved directly (i.e. locally) once the system recognizes an object. The configuration process firstly identifies reference objects and the related content for each individual object so that the augmentation process can display related media content of the tracked objects, as required. Moreover, mobile users are able to request associated content by touching the object on the active screen; at which point, the system will check to see if there are any interactions with geometries on the screen. Once the user indicates that they want to see associated content, the system will read the associated content configuration file and extract data. The data is in the form of the predefined API and parameter specifications relevant to the participating provider. A service request will then be sent through the web service framework to the participating provider and a response document in the XML or JSON format will be sent back to the service request module.

In the home-based learning scenario, there are no predefined content stored in the content library and so all the content must be requested when the system tracks a trigger image. The object augmentation module will read through the reference configuration file and retrieve the URL, which is the API and associated parameters for the open content provider, related to the tracked image. The content request will then be sent through the web service framework to the provider and the response obtained will then be processed. The user can then view associated media content of the tracked trigger image, which represent the physical object in the museum. Figure 4.11 presents the process of object augmentation for museum-based and home-based learning scenario. The augmentation process starts working when the AR View starts and the related content will be visualized as soon as the system is able to recognize at least one reference object. The main functions of both scenarios are generally similar but the process of content acquisition in the home-based learning has to be done by tracking the trigger images. The acquired associated content from the provider will be presented on the screen.

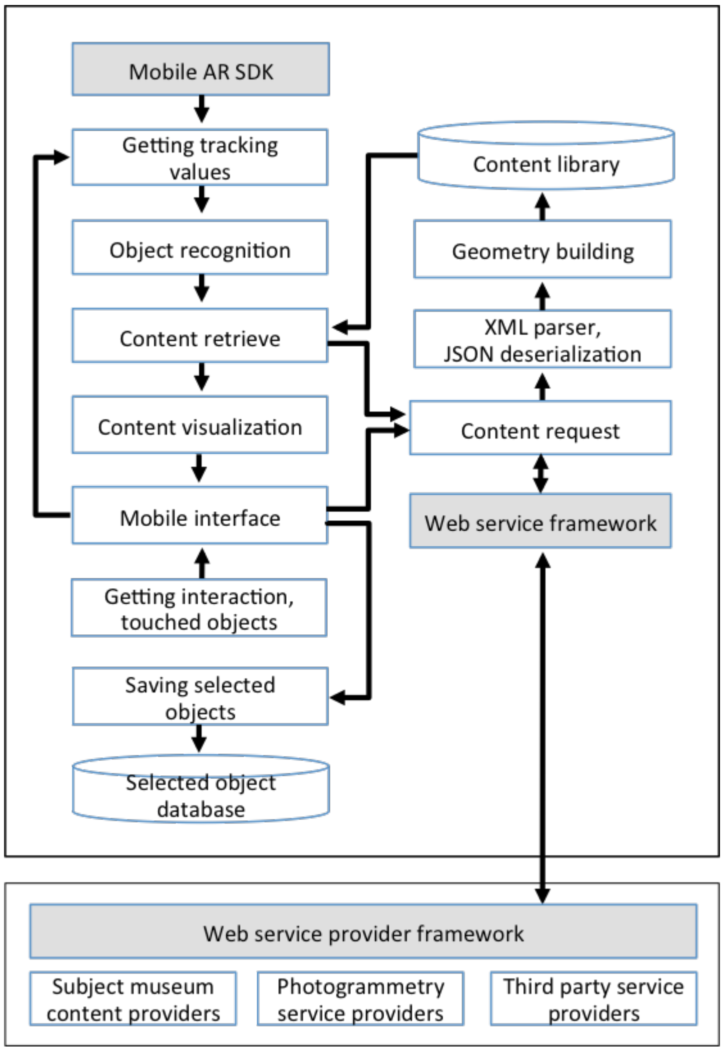


Figure 4.11 The object augmentation process in museum-based and home-based museum learning scenario

Supporting modules

This novel mobile AR application, or mobile client, is able to offer some supporting modules that take full advantage of it being on an open platform and, in addition, being an AR system. There are two approaches that could be usefully applied by the mobile client in relation to MLA scenarios, for content acquistion and utilisation. The following are details of each approach (these have been implemented into the mobile client and offered functions to users).

Content acquisition

Content acquistion is generally accomplished by sending a request to a participating content provider (which one depends on the scenario) and receiving a response through a web service framework. In addition, content acquistion can also be performed by sending a request to a possible service provider whose responses can usefully be presented in AR environments. One possible service provider that can, beneficially be used by the mobile client for MLA scenarios is a photogrammetry service. A photogrammetry service is an image-based reconstruction service whereby photographs of a physical object or landscape can be uploaded to it so that a virtual 3D model of the object is created. The resultant model will be sent back to the mobile client and visualized to the user. Photogrammetry is one of the content acquisition approaches that can be usefully integrated into a mobile client where this supports web-based content acquistion. In this case the content acquisition is carried out by sending a request along with parameters, which are a sequence of photographs, to the service provider. The service provider then responds with a 3D model that can be consumed in the AR environment. A photogrammetry service request module in the mobile client applies the service API of the provider and works with the embedded camera of the mobile device in order to obtain photographs of a selected object. It then makes an image-based reconstruction request and receives a response that is a virtual 3D model of the object. The final model will then be visualized and utilized on the scene. Figure 4.12 presents the process of requesting a photogrammetry service that requires photographs of a targeted object as parameters to be sent to the provider.

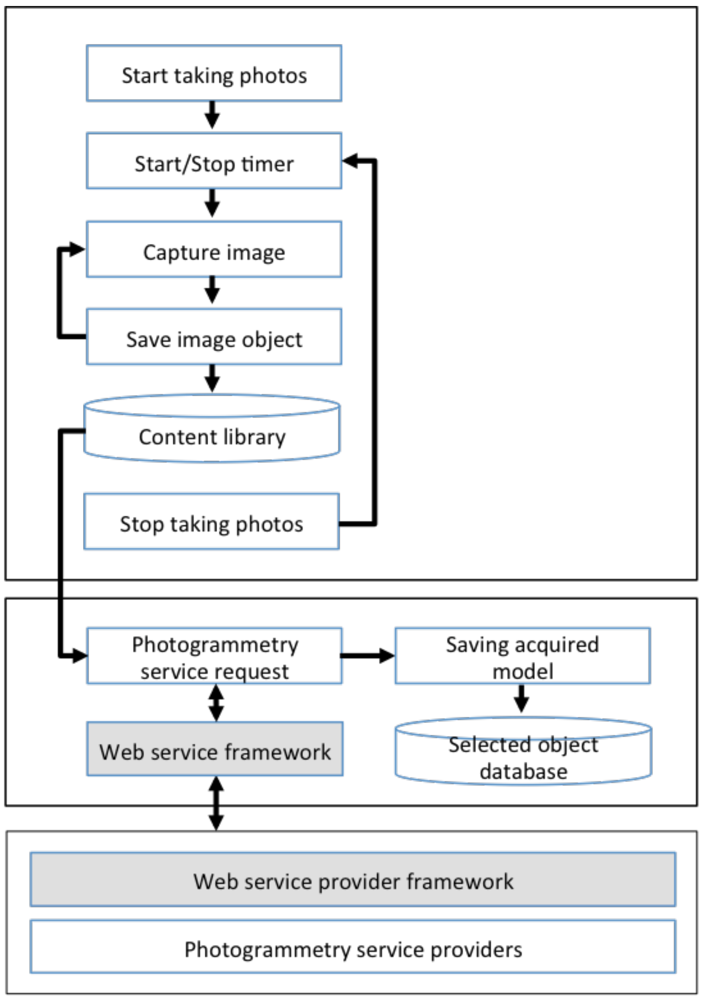


Figure 4.12 Photogrammetry service request module

The photogrammetry service request module is initiated when the mobile device user touches a button to start taking photographs. A timer will then be activated in order to automatically take further photographs. Each photograph will be transformed into an object and saved into the content library. The timer will be halted when the user touches the stop button (and so the system will stop taking photographs). The user can then decide to send a request for a virtual 3D model of the targeted object. Figure 4.13 shows the use-case diagram of a photogrammetry service request. The mobile user will participate by targeting an object, taking photographs of that object, and then sending a request for image-based reconstruction services to the relevant service provider.

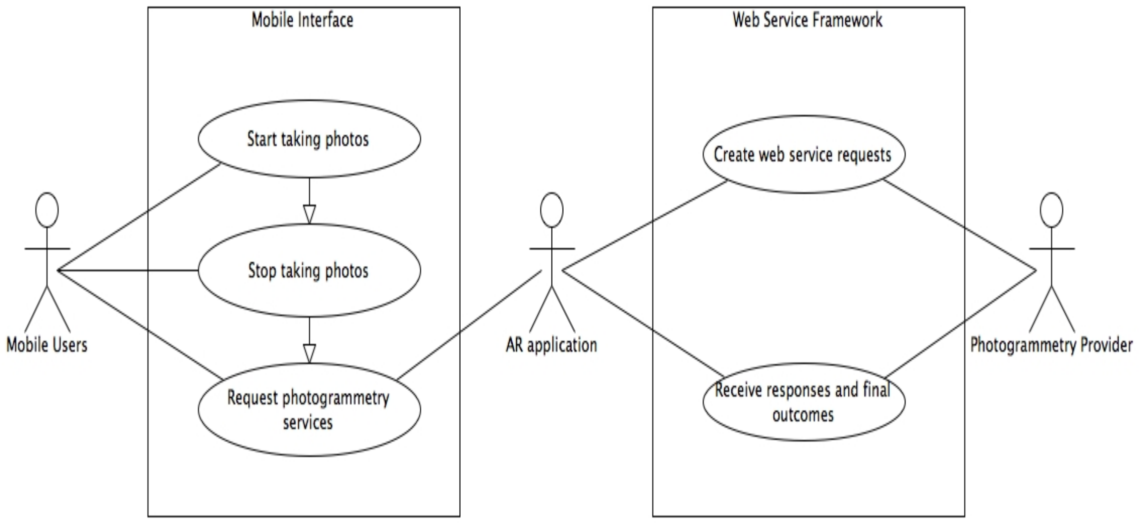


Figure 4.13 The process of photogrammetry service request done by mobile users

Content utilization

Content utilization is one of the features strongly supported by the mobile client in order to enable mobile users to consume acquired AR content. These content can be acquired from a mobile AR environment and then selected content can be used and/or displayed in other situations. The proposed content utilization module is an AR environment personalization system that allows mobile users to select and save content, which have been obtained and then visualized on the screen. This suggested AR environment personalization supports data representation using a mobile AR framework so that media content can be presented within AR environments. Personal AR environments can be selected for presentation elsewhere and presented by augmenting a trigger image. Figure 4.14 illustrates the process of AR environment personalization that mobile users are able to select and save their preferred AR content.

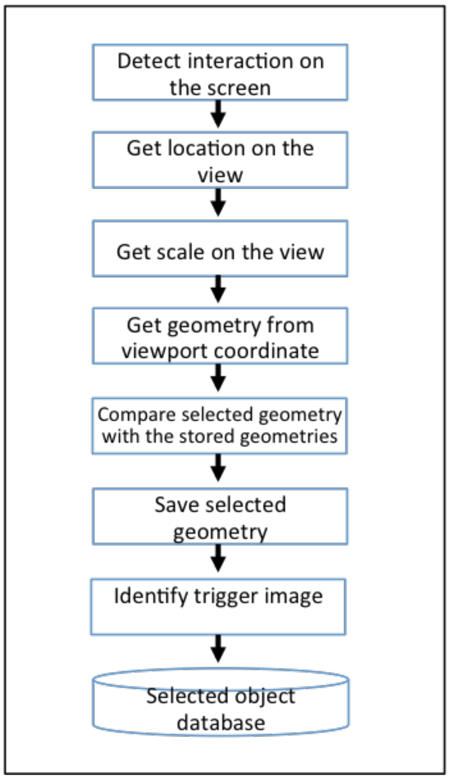

Detect interaction on the screen
Get location on the view
Get scale on the view
Get geometry from viewport coordinate
Compare selected geometry with the stored geometries
Save selected geometry
Identify trigger image
Selected object database

Figure 4.14 The process of creating personalized AR environments

Figure 4.15 is the use-case diagram for the AR environment personalization module, which can be accessed through the mobile interface. Mobile users are able to select and save preferred content. The saved content can then be visualized in an AR Browser that augments a trigger image by displaying or revealing selected related content.

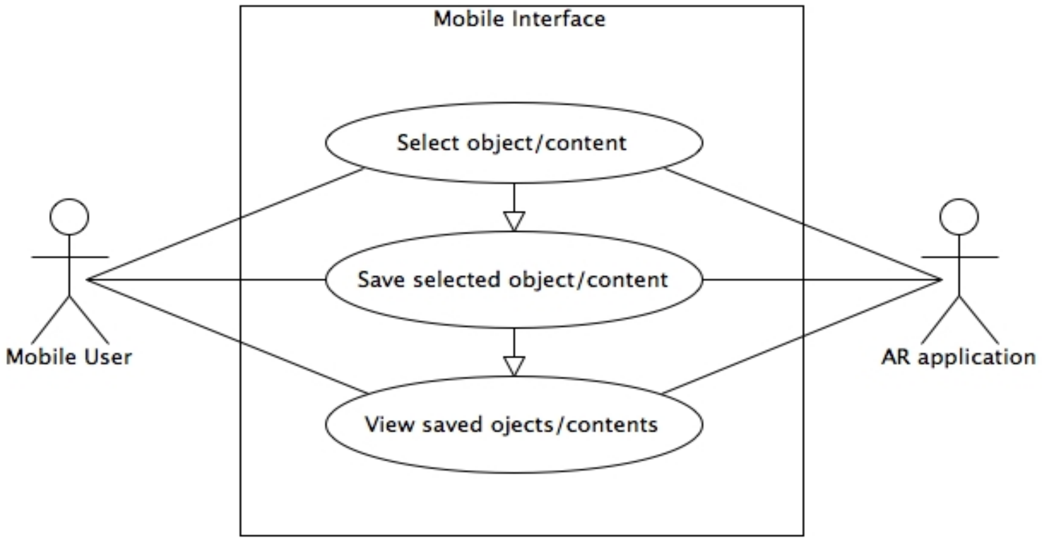


Figure 4.15 Use-case diagram of AR environment personalization

The AR Browser is used to reveal personalized AR environments composed of saved media content, including acquired content from participating providers and/or virtual 3D objects from photogrammetry services that have been selected and saved by the mobile user. This module will require the presence of an embedded-camera (readily provided by modern smartphones and tablets) in order to perform image-based tracking and so reveal user's preferences on the AR browser. Figure 4.16 illustrates the AR Browser whereby mobile users are able to reveal personalized AR content over-laid on the real scene by tracking a trigger image.

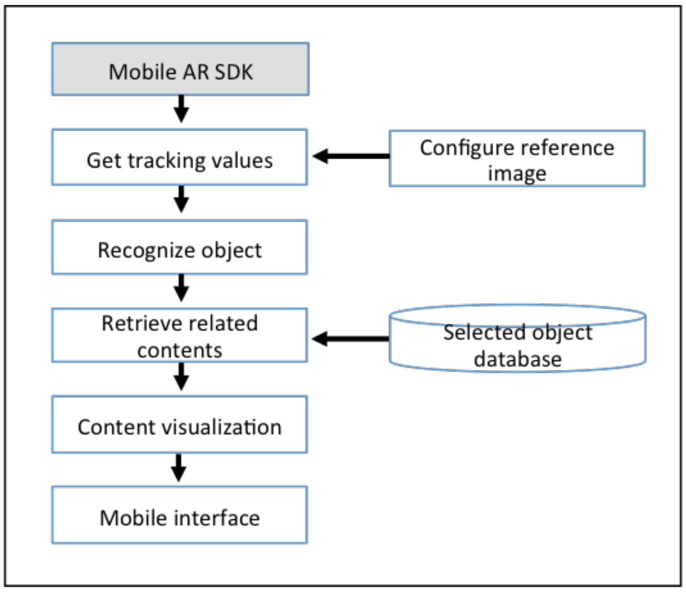


Figure 4.16 The process of visualizing personalized AR environments using AR Browser

Figure 4.17 shows the use-case diagram relating to using the AR Browser in order to view personalized AR environments in other situations (outside the museum).

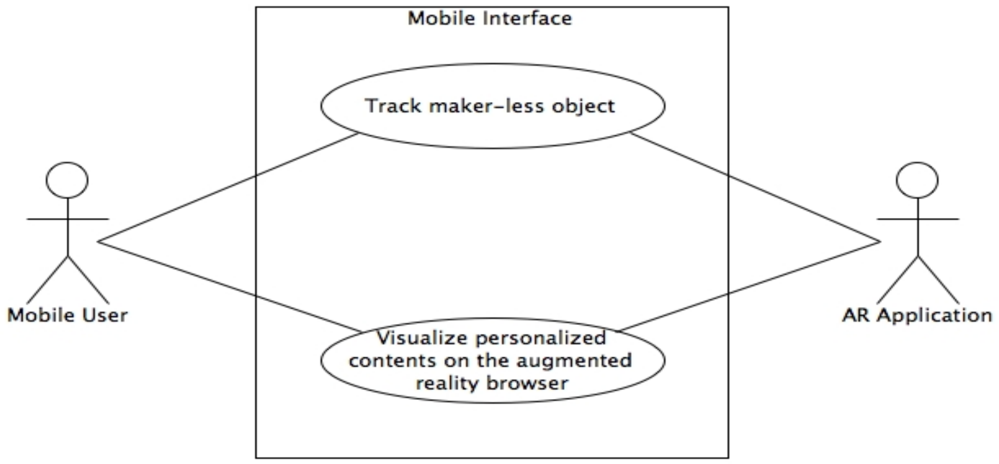
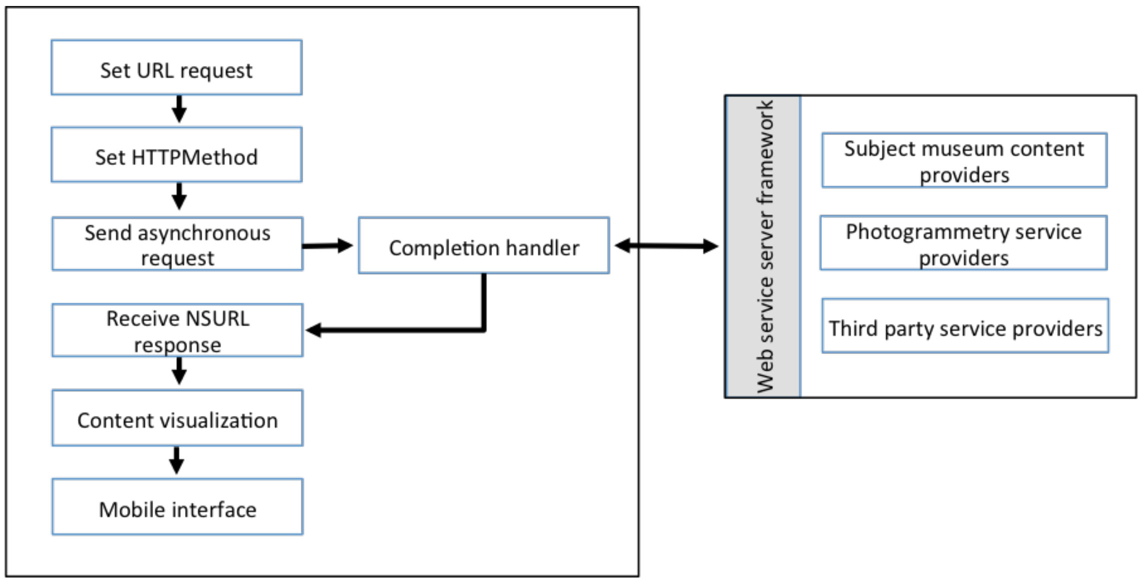

Mobile Interface
Track maker-less object
Visualize personalized contents on the augmented reality browser
Mobile User
AR Application

Figure 4.17 The use-case diagram of AR Browser

4.2 Web Service Framework

The web service framework is a technology used to create connections and interactions between services in a SOA. A web service framework can be applied to any system that requires open architecture. The system will then become a client-side application that is able to access open services in the SOA. In this research, the web service framework is implemented into the mobile AR application, making it a mobile client on open platform. The web service-based mobile AR application or mobile client will be able to perform interoperable tasks including sending requests and receiving responses as well as supporting the application client in acquiring associated content from open service or content providers. The web service framework in the native iOS development framework supports the REST communication protocol. This requires service APIs of participating providers to be in the form of a URL containing the source of content, the function and the parameters required, which will be processed in the server side. The web service framework works concurrently with the AR application as a back-end system for creating web service connections in order to send requests and receive responses between the mobile client and web service providers. The web service responses will need XMLParser or NSJSONSerialization function in order to process XML or JSON documents representing the final outcome from the web service providers. The content will be extracted from the received document and visualized on the screen. Figure 4.18 presents the structure of the web service framework of the mobile client using the REST architecture.



Set URL request
Set HTTPMethod
Send asynchronous request
Completion handler
Receive NSURL response
Content visualization
Mobile interface
Web service server framework
Subject museum content providers
Photogrammetry service providers
Third party service providers

Figure 4.18 Web service framework

4.3 Web Service Providers

SOMARA requires service APIs from all providers. These can be participating open providers that could be third party, potential content providers for the desired scenario or effective service providers who can provide processing services such as image-based reconstruction and so valuable content for AR environments such as 3D models, images, text etc. Open providers basically offer services for SOA so that any application is able to create connections and interaction with the participating providers through a web service framework. Web service providers process service requests and return responses to service requesters or application clients. Service providers offer service interfaces or APIs for clients and the service connection module inside that will apply the APIs and identify required parameters before sending requests to the providers. In this research, SOMARA is applied in museum scenarios for which the participating web service providers will be museum content providers that have exposition data concerning the objects in their repository. The proposed providers applied in SOMARA offer service APIs and the mobile client will apply those APIs in order to send service requests and receive responses. The participating service providers for MLAs chosen to integrate into the mobile client as sample content providers for SOMARA are the RCH Cultural Object Service and Victoria and Albert Museum open service mentioned in Chapter 3. Figure 4.19 illustrates the structure of the selected museum service providers that their API is applied into the service request module in order to obtain valuable content from the open sources.

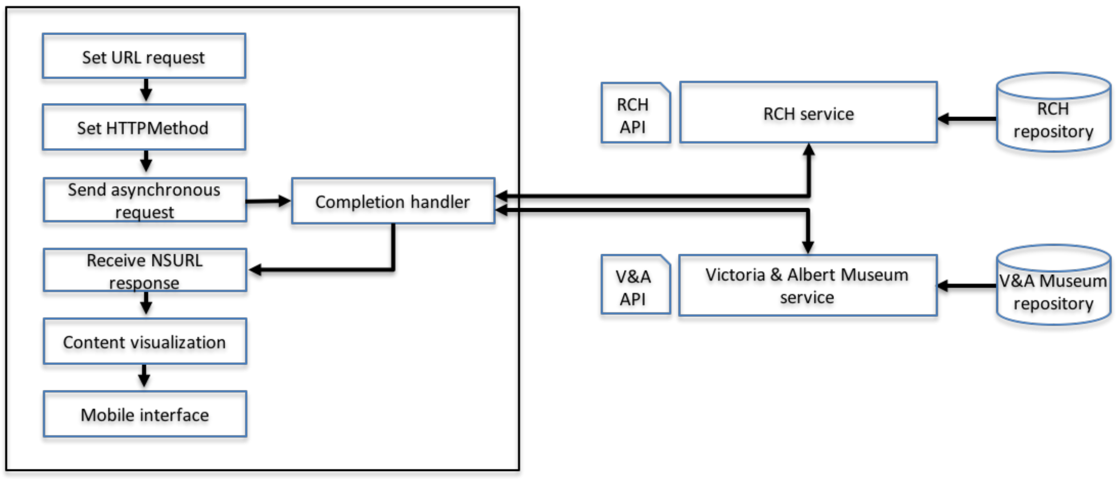


Figure 4.19 Web service framework and the implemented service providers

4.4 Summary

This chapter describes SOMARA and it's components that encourage the development of SOA on a mobile AR platform. SOMARA generates service-orientation enabling the creation of SOMARA-based mobile AR application that is composed of a mobile AR client, a web service framework and open service providers. The mobile AR client performs AR tasks that work simultaneously with the AR application and the mobile interface for tracking reference objects and visualizing related content on the real scene. The application also supports interoperability tasks where mobile users or the application itself are able to request associated content from web service providers through a web service framework.

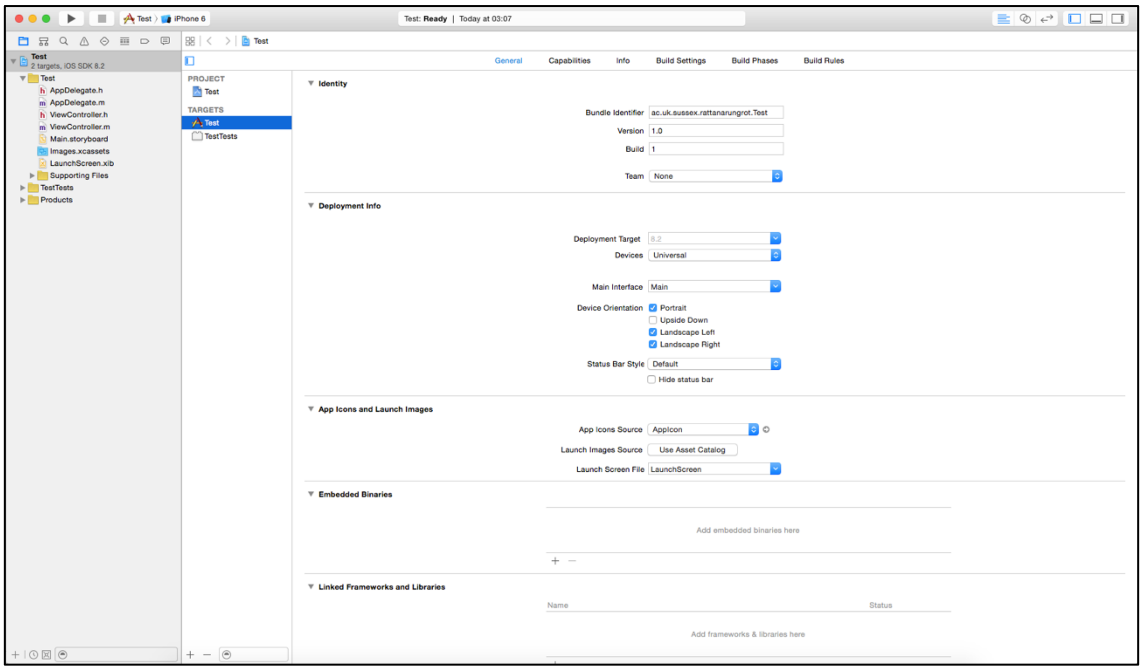
Chapter V

5 A SOMARA Application Development

This chapter explains the process of developing a novel mobile AR application based on the Service-Oriented Mobile Augmented Reality Architecture (SOMARA) — a SOMARA application. As such, this chapter provides the implementation of SOMARA based AR application that can be used as a guide for the development of other museum AR applications exploiting SOMARA — this particular implementation of a SOMARA application should not be regarded as a product; it serves to illustrate how build a typical SOMARA applications, and in this case using Apple technology. The development focuses on an implementation of SOMARA, which includes building the mobile AR client on an open platform that supports SOMARA, with a web service framework in order to work as a back-end system to perform interoperability tasks. The application is composed of functions in order to perform AR tasks including object tracking, recognition and content visualization and interoperability tasks including service connections and content acquisition. The novel mobile AR application supports museum-based and home-based learning scenarios. The mobile client is an AR application on a mobile platform and in this research the application is developed on Apple iOS SDK and XCode IDE. The following sections cover the processes of developing a SOMARA application that enhances the usability and functionality of typical standalone or closed-platform mobile AR applications. Each section shows the important functions and modules inside the open mobile AR application that effectively perform AR tasks, interoperability tasks and supporting tasks that encourage content acquisition and consumption.

5.1 Development Tools — XCode and iOS SDK

The development tools chosen for building mobile AR applications are XCode and Objective-C language because XCode IDE provides developer tools, a graphical interface and a debugger that effectively support application development. Developers have to apply for an Apple developer membership and register mobile devices that will be used to run the application. In addition, the personal certificate and profile on the developer portal needs to be downloaded and installed in the computer used to develop the application as well as in the mobile device that is the deployment target. Figure 5.1 presents the development framework of XCode IDE for the iOS platform and the components of the mobile application environment.



Identity

Bundle Identifier: ac.uk.sussex.rattanarungrot.Test
Version: 1.0
Build: 1
Team: None

Deployment Info

Deployment Target: 8.2
Devices: Universal
Main Interface: Main
Device Orientation: [✓ Portrait] Upside Down [✓ Landscape Left] [✓ Landscape Right]
Status Bar Style: Default
Hide status bar

App Icons and Launch Images

App Icons Source: AppIcon
Launch Images Source: Use Asset Catalog
Launch Screen File: LaunchScreen

Embedded Binaries

Add embedded binaries here

Linked Frameworks and Libraries

Name	Status

Add frameworks & libraries here

Figure 5.1 iOS and XCode development framework

The XCode IDE provides the Main Storyboard and View Controller as the start view for developers to create the first view of a mobile application. The developer can add more views into the application whereby each view contains a header file, implementation file and View Controller.

5.2 AR View

A SOMARA application has three AR Views for museum-based learning, home-based museum learning and an AR Browser. The AR Views have been created on top of the normal View Controller and simultaneously work with MetaioSDKViewController for connecting to the embedded camera and presenting a real scene so that mobile users can track a targeted object and see visualized content superimposed onto the tracked object. The View Controller contains modules that control and work with the AR View and MetaioSDKViewController in order to perform AR tasks as well as interoperability tasks, and mobile users are able to interact with the system and active content on the screen. Figure 5.2 shows the mobile AR View of the museum-based learning scenario that is run on an iPad. The AR View includes a mobile interface that allows mobile users to interact with the visualized content and system through the touch screen and buttons located on the screen.

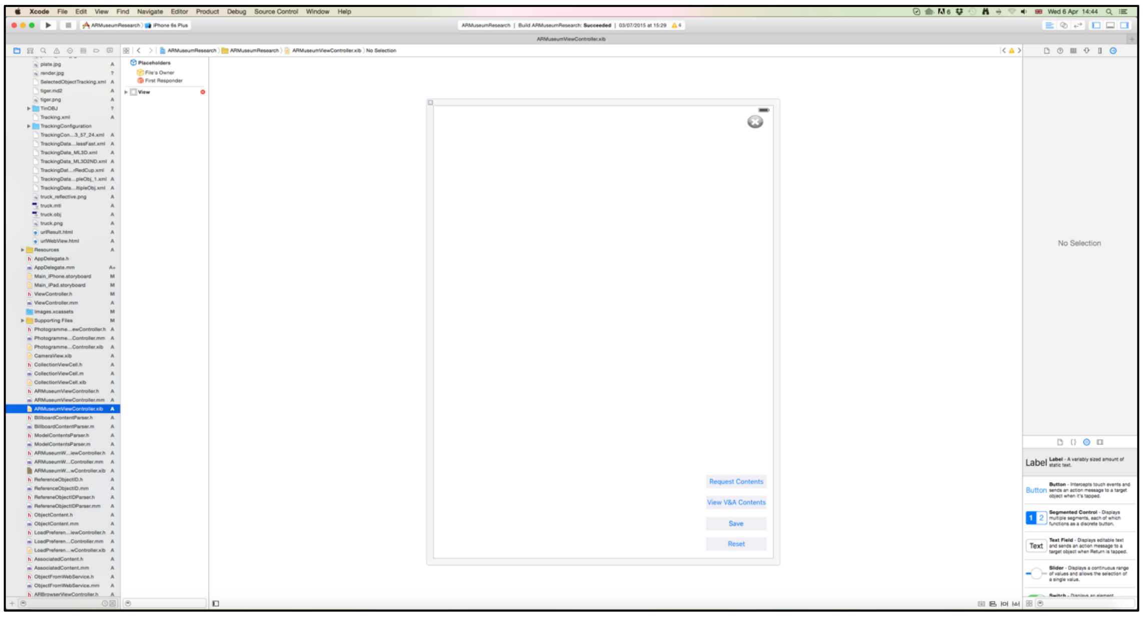


Figure 5.2 The mobile AR View of the museum-based learning scenario

Figure 5.3 presents the mobile AR View of the home-based learning scenario. In this scenario, an AR View is required for mobile users to perform image-based tracking and system interaction in order to view associated content acquired from the participating service provider on the screen.

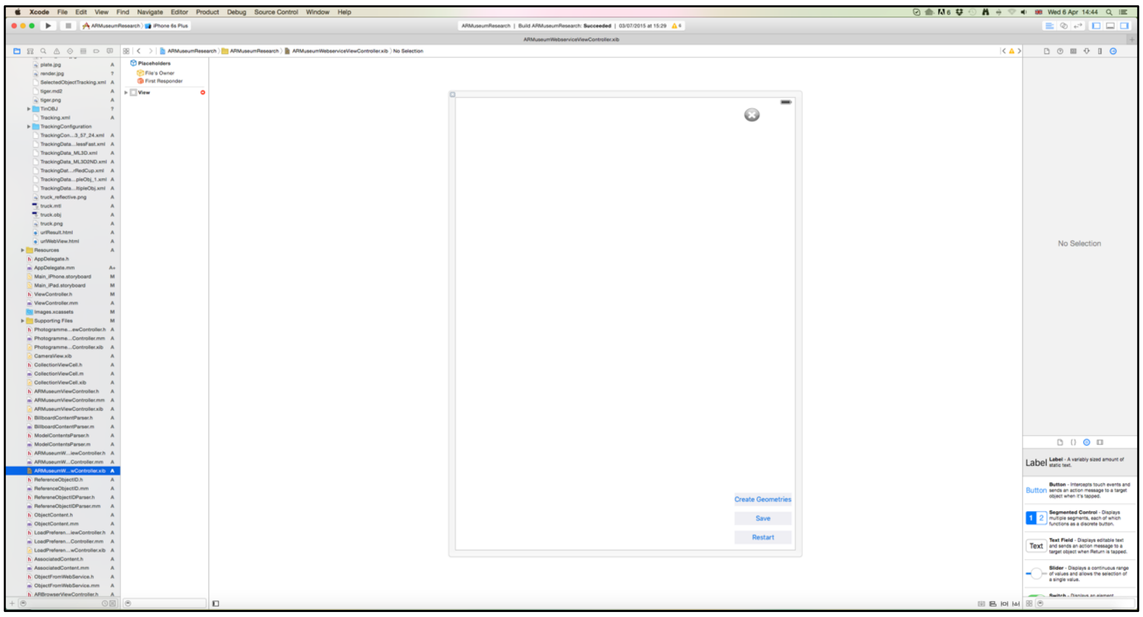


Figure 5.3 The mobile AR View of home-based museum learning scenario

Figure 5.4 shows the AR Browser that reveals personal AR environments created by mobile users. Preferred content from the museum-based learning scenario that has been stored in the database is retrieved and visualized on the AR Browser in other situations.

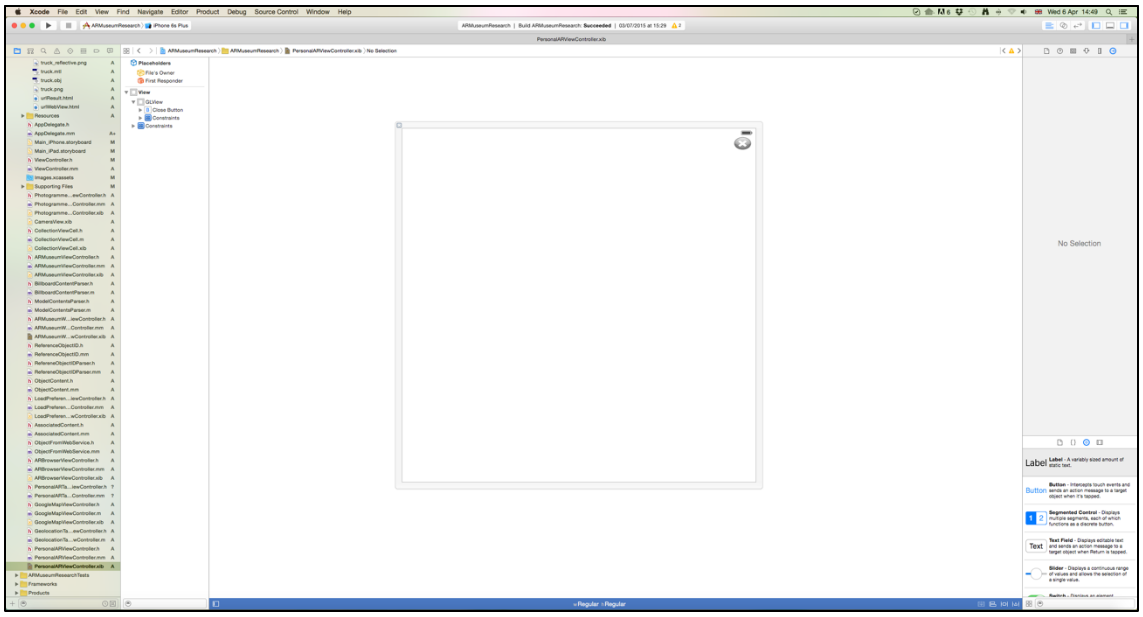


Figure 5.4 The AR Browser view

5.3 Tracking and Content Configuration

The tracking and content configuration starts working after the View Controller is loaded and the AR View appears on the screen. The tracking configuration process requires a tracking data file that is processed by the MetaioSDK to identify and recognize reference objects. In addition, a content configuration file is needed in order to provide related media content of each reference object that will be tracked. The following are the tracking processes applied in the museum-based learning and home-based museum learning scenarios that require different tracking and content configuration files.

5.3.1 Museum-based learning scenario

Museum-based learning performs physical object tracking and the system visualizes related content that has been pre-defined in the content library. The configuration process requires a tracking configuration file and content configuration file for initializing reference objects that are to be tracked and related content of each object that is to be revealed on the real scene. The following is the tracking and content configuration for the museum-based learning scenario.

Tracking configuration

The tracking configuration process is done when the UI View is loaded and the camera view is presented on the screen. The required tracking configuration file is in XML document format as shown below.

```
<?xml version="1.0" encoding="UTF-8"?>
<TrackingData>
  <Sensors>
    <Sensor Type="FeatureBasedSensorSource" Subtype="ML3D">
      <SensorID>FeatureBasedSensorSource_0</SensorID>
      <Parameters>
        <featureorientationassignment>gravity</featureorientationassignment>
        <MaxObjectsToDetectPerFrame>5</MaxObjectsToDetectPerFrame>
        <MaxObjectsToTrackInParallel>1</MaxObjectsToTrackInParallel>
      </Parameters>
      <SensorCOS>
        <SensorCosID>fb23be3415c00141a340d36b2766f610</SensorCosID>
        <parameters>
          <numextensiblefeatures>0</numextensiblefeatures>
          <mintriangulationangle>6</mintriangulationangle>
          <map>fb23be3415c00141a340d36b2766f610.f3b</map>
          <MinMatches>15</MinMatches>
          <NumExtensibleFeatures>250</NumExtensibleFeatures>
        </parameters>
      </SensorCOS>
    </Sensor>
  </Sensors>
  <Connections>
    <COS>
      <Name>cos1</Name>
      <Fuser Type="SmoothingFuser">
        <Parameters>
          …
        </Parameters>
      </Fuser>
      <SensorSource>
        <SensorID>FeatureBasedSensorSource_0</SensorID>
        <SensorCosID>fb23be3415c00141a340d36b2766f610</SensorCosID>
        <HandEyeCalibration>
          <TranslationOffset>
            …
          </TranslationOffset>
          <RotationOffset>
            …
          </RotationOffset>
        </HandEyeCalibration>
        <COSOffset>
          <TranslationOffset>
            …
          </TranslationOffset>
          <RotationOffset>
            …
          </RotationOffset>
        </COSOffset>
      </SensorSource>
    </COS>
  </Connections>
</TrackingData>
```

The XML document contains the metadata or details of reference objects specified by the developer, including a 3D tracking map of each reference object created by Metaio Toolbox, the SensorCOSID and COS name. The SensorCOS includes a group of details of each reference object composed of SensorCOSID and the parameters containing the 3D tracking map file and the default numbers. The 3D tracking map files have to be stored in the content library and the MetaioSDK uses the prepared map files in order to track and recognize a targeted object on the AR scene. The process of creating 3D map files of reference objects can be seen in Appendix D. The following code is the process of setting up the tracking configuration file in XCode and Objective-C.

```
NSString* trackingDataFile = [[NSBundle mainBundle] pathForResource:@"Tracking"
ofType:@"xml" inDirectory:@"Assets1"];
bool success = m_metaioSDK->setTrackingConfiguration([trackingDataFile
UTF8String]);
```

Content configuration

The next process is to perform content configuration that identifies related content to each reference object. The pre-defined content is organized in the XML document and it requires the XMLParser to read through the document and extract data in the document. The data are transformed into geometries of each object and visualized when the object is tracked and recognized.

```
<?xml version="1.0" encoding="UTF-8"?>
<Contentgroup>
   <Cos>
         <Name>cos1</Name>
         <Billboard>
                     <Title> Tea leaf tin </Title>
                     <Details>A Test 3D Object</Details>
         </Billboard>
         <Model>
                     <MFilename>metaioman</MFilename>
                     <MFiletype>md2</MFiletype>
         </Model>
         <Image>
                     <IFilename>crabtree</IFilename>
                     <IFiletype>png</IFiletype>
         </Image>
         <Image>
                     <IFilename>MoreDetails</IFilename>
                     <IFiletype>png</IFiletype>
         </Image>
   </Cos>
</Contentgroup>
```

The system can also support acquiring related content of a tracked object from open sources such as a local museum content provider. This can be done by sending a request for related content of a reference object to the provider and receiving a response. The developer does not need to prepare media content and store it in the library of the application in advance. In this option, the content configuration file contains service APIs or URLs and required parameters used to access the content provider and request media content of the reference objects. The following is an example of a content configuration file in XML format identified to each reference object for sending web service requests.

```
<?xml version="1.0" encoding="UTF-8"?>
<Contentgroup>
   <Cos>
      <Name>cos1</Name>
      <CID>http://www.vam.ac.uk/api/json/museumobject/O78523</CID>
   </Cos>
</Contentgroup>
```

The designed content configuration file has to be transferred to XMLParser in order to extract the specified content of each reference object and create geometries of the content. The pseudocode for configuring related content of each reference object and the process of calling XMLParser is displayed below.

```
identify content configuration file;
set NSData of the content configuration file;
create  an array of retrieved billboard content;
create the object of BillboardContentParser;
call function parseXMLFile in BillboardContentParser;
if(success)
{
        create billboard objects of acquired content in the array;
}
```

BillboardContentParser retrieves data of content stored in the content configuration file and it returns an array of data that is sent to the module for building geometries. The BillboardContentParser is used to read data in the tag <BillBoard> that are used to create billboard content presenting text on the AR screen. The following is the pseudocode of functions in BillboardContentParser class.

```
-(NSMutableArray *)parseXMLFile:(NSData *)xmlData
{
        initialize received XML data to NSXMLParser;
        set Delegate;
        parse XML document;
        return array;
}
-(void)parserDidStartDocument:(NSXMLParser *)parser
{
        create an object of array;
}
```

```
-(void)parser:(NSXMLParser *)parser didStartElement:
(NSString *)elementName namespaceURI:(NSString *)namespaceURI
qualifiedName:(NSString *)qName attributes:(NSDictionary *)attributeDict
{
    if(elementName is equal to "Cos")
   {
        create an object for billboards;
    }
}
-(void) parser:(NSXMLParser *)parser foundCharacters:(NSString *)string
{
        find and collect string;
}
-(void) parser:(NSXMLParser *)parser didEndElement:
(NSString *)elementName namespaceURI:(NSString *)namespaceURI
qualifiedName:(NSString *)qName
{
   if (elementName is equal to "Name")
   {
      store collected string in object;
   }
   else if (elementName is equal to "Title")
   {
     store collected string in object;
   }
   else if (elementName is equal to "Details")
   {
      store collected string in object;
   }
   if (elementName is equal to "Cos")
   {
       add object to the array;
   }
}
-(void) parserDidEndDocument:(NSXMLParser *)parser
{
}
```

The returned array of data extracted from the content configuration file is sent to the CreateBillboardARContent module, which is shown below, for building geometries of content in the obtained array.

```
- (void)CreateBillboardARContent:(NSMutableArray *)data
{
      create billboard object
      create array for billboards;
      read billboard data from array
      {
         if(data is the end of array)
```

```
            {
                  break;
            }
            else
            {
                  create billboard image from title;
                  create geometry of billboard image;
                  set visibility;
                  create billboard image from details;
                  create geometry of billboard image;
                  set visibility;
                  add geometry object to array;
            }
      }
}
```

5.3.2 Home-based museum learning scenario

Home-based museum learning performs image tracking that requires trigger images of real cultural objects. After recognizing the trigger image, the system sends a request to the participating open service and receives a response that is the associated content of the tracked image; this is visualized on the real scene. The following explains the tracking and content configuration process of the home-based museum learning scenario.

Tracking configuration

The tracking configuration process requires a tracking configuration file in the same way as the museum-based learning scenario. The following is the tracking configuration file in XML format used for tracking trigger images.

```
<?xml version="1.0"?>
<TrackingData>
   <Sensors>
      <Sensor Type="FeatureBasedSensorSource" Subtype="Fast">
        <SensorID>FeatureTracking1</SensorID>
        <Parameters>
              <FeatureDescriptorAlignment>regular</FeatureDescriptorAlignment>
              <MaxObjectsToDetectPerFrame>5</MaxObjectsToDetectPerFrame>
              <MaxObjectsToTrackInParallel>1</MaxObjectsToTrackInParallel>
              <SimilarityThreshold>0.7</SimilarityThreshold>
         </Parameters>
         <SensorCOS>
              <SensorCosID>Patch1</SensorCosID>
               <Parameters>
                  <ReferenceImage>Nomoli.jpg</ReferenceImage>
```

```
                <SimilarityThreshold>0.7</SimilarityThreshold>
             </Parameters>
         </SensorCOS>
      </Sensor>
   </Sensors>
   <Connections>
      <COS>
        <Name>cos1</Name>
          <Fuser Type="SmoothingFuser">
          <Parameters>
               …
         </Parameters>
         </Fuser>
         <SensorSource>
               <SensorID>FeatureTracking1</SensorID>
               <SensorCosID>Patch1</SensorCosID>
               <HandEyeCalibration>
                       <TranslationOffset>
                               …
                       </TranslationOffset>
                       <RotationOffset>
                               …
                       </RotationOffset>
               </HandEyeCalibration>
               <COSOffset>
                       <TranslationOffset>
                               …
                       </TranslationOffset>
                       <RotationOffset>
                               …
                       </RotationOffset>
               </COSOffset>
         </SensorSource>
        </COS>
   </Connections>
</TrackingData>
```

This XML configuration file is used to configure the tracking process by MetaioSDK for tracking planar surface objects such as a page in a magazine or images. The file contains details of each reference image including the SensorCosID, reference image file and COS name that is used to track and recognize a trigger image, and then the system reveals associated content obtained from open services such as third party or museum content providers. The process of setting up the configuration file is shown below.

```
NSString* trackingDataFile = [[NSBundle mainBundle]
pathForResource:@"MarkerlessTracking" ofType:@"xml" inDirectory:@"Assets1"];
```

```
bool success = m_metaioSDK->setTrackingConfiguration([trackingDataFile
UTF8String]);
```

Content configuration

The associated content is requested from the participating content providers on the client-server architecture; the content configuration process identifies the service API of each provider and parameters required for sending a request. The following is the content configuration file in XML format containing the URL and parameters of each reference image.

```
<?xml version="1.0" encoding="UTF-8"?>
<Contentgroup>
   <Cos>
      <Name>cos1</Name>
     <CID>http://www.sierraleoneheritage.com/api/search_service/fetch_item/coid/231
     1/format/xml</CID>
  </Cos>
  <Cos>
     <Name>cos2</Name>
     <CID>http://www.sierraleoneheritage.com/api/search_service/fetch_item/coid/220
     0/format/xml</CID>
   </Cos>
   <Cos>
     <Name>cos3</Name>
     <CID>http://www.sierraleoneheritage.com/api/search_service/fetch_item/coid/237
     0/format/xml</CID>
   </Cos>
   <Cos>
      <Name>null</Name>
   </Cos>
</Contentgroup>
```

The pseudocode for setting up the content configuration file is explained below. The file is sent to XMLParser in order to extract the URL of each trigger image. The extracted data are returned back to the module and stored in the array.

```
identify URL configuration file;
set NSData of the URL configuratrion file;
create the object of ReferenceObjectIDParser;
create an array of retrieved URLs;
call function parseXMLFile in ReferenceObjectIDParser;
```

The following is the pseudocode of functions in ReferenceObjectIDParser class that reads through the content configuration file, extracts data in the file and returns the array of data back to the module.

```
-(NSMutableArray *)parseXMLFile:(NSData *)xmlData
{
        initialize received XML data to NSXMLParser;
        set Delegate;
        parse XML document;
        return array;
}
-(void)parserDidStartDocument:(NSXMLParser *)parser
{
        create an object of array;
}
-(void)parser:(NSXMLParser *)parser didStartElement:(NSString *)elementName
namespaceURI:(NSString *)namespaceURI qualifiedName:(NSString *)qName
attributes:(NSDictionary *)attributeDict
{
   if (elementName is equal to "Cos")
   {
        create an object for URL reference objects;
   }
}
-(void) parser:(NSXMLParser *)parser foundCharacters:(NSString *)string
{
        find and collect string;
}
-(void) parser:(NSXMLParser *)parser didEndElement:(NSString *)elementName
namespaceURI:(NSString *)namespaceURI qualifiedName:(NSString *)qName
{
   if(elementName is equal to "Name")
   {
        store collected string in object;
   }
   else if (elementName is equal to "CID")
   {
        store collected string in object;
   }
   if (elementName is equal to "Cos")
   {
        add object to the array;
   }
}
-(void) parserDidEndDocument:(NSXMLParser *)parser
{
}
```

The tracking configuration files presented above in both museum-based and home-based learning scenarios support single object tracking where the system can track only one reference object. In order to perform multiple object tracking, the system requires a tracking configuration file that contains tracking information of all reference physical objects so that the SDK can process all of the information at once.

5.4 Object Augmentation

The process of tracking objects and augmenting related content on top of the real scene is described in this section. The tracking system starts working when the UI View is loaded, the AR View appears on the screen and the configuration is done by the MetaioSDK. The object augmentation module works recursively in order to get all tracking values in the real environment and recognize a reference object. In this module, mobile users can start tracking a targeted object seen on the AR View and the system augments related media content of the tracked object and visualizes them on the view. The code that performs object tracking or getting tracking values of objects in the real scene is shown below.

```
std::vector<metaio::TrackingValues> poses = m_metaioSDK->getTrackingValues();
```

The system augments related content on top of the real scene by checking whether the tracking finds and recognizes a reference object in the environment; then the system displays relevant content of the tracked object on the screen. This process can be done by getting the COSName of the tracked object and reading through the array of geometries for the related content that are visible to mobile users. The following pseucode performs object augmentation that visualizes billboards containing text on top of the tracked reference object.

```
if(found an object)
{
    create a sting of COS name from found object;
    read prepared object from array
   {
      if COS Name is equal to COS Name of the object
      {
              add object to the billboard group;
              set coordinateSystemID of the object;
              set visibility to true;
              break;
      }
   }
}
```

5.5 Service Request and Response

The content acquisition process in the SOMARA application is applied in both museum-based and home-based learning activity scenarios. Associated content of reference objects can be requested from participating open providers through the new SOMARA web service framework. The following is the code for sending service requests and receiving responses that can be in XML or JSON documents in both scenarios.

5.5.1 Museum-based learning scenario

Museum-based learning offers mobile users a module for requesting associated content of a targeted object when the user touches a button visualized on the screen. The AR View detects interaction from the mobile user and then performs a content request. The following is the pseucode for detecting touch by picking up selected geometries and checking whether the touched object is a service request one. The system performs content configuration in order to fetch the prepared URL and parameters, which is the API of the participating service provider, which could also be the museum itself exposing its collections via a web service API.

```
- (void) touchesBegan:(NSSet *)touches withEvent:(UIEvent *)event
{
        get interaction on screen;
        get location of the touch object;
        get geometry from location in ViewportCoordinates;
        if(found geometry)
        {
        search touched geometry in the moreDetialsBoardArray
        {
                if(cosName in object is equal to “null”)
                {
                        break;
                }
                if(selected geometry is moreDetailsObject)
                {
                        identify associated museum content configuration file;
                        set NSData of associated museum content configuration file;
                        create an object of ReferenceObjectIDParser;
                        create an array of URL reference objects;
                        call parseXMLFile in ReferenceObjectIDParser;
                        search object in the array
                        {
                                if(cName is equal to “null”)
                        {
                                break;
                        }
                        else if (cName is equal to the cosName of  selected
                                geometry)
                        {
                                connect web service by sending cid in the targeted
                                object;
                        }
                    }
                    break;
                }
            }
        }
```

This module requires the content configuration file presented below. The content configuration file contains APIs of service providers along with parameters in URL format that are read from the file and sent back to the module.

```
<?xml version="1.0" encoding="UTF-8"?>
<Contentgroup>
   <Cos>
      <Name>cos1</Name>
      <CID>http://www.vam.ac.uk/api/json/museumobject/?q=tea</CID>
   </Cos>
   <Cos>
      <Name>cos2</Name>
     <CID>http://www.sierraleoneheritage.com/api/search_service/fetch_item/coid/220
     0/format/xml</CID>
   </Cos>
   <Cos>
      <Name>null</Name>
   </Cos>
</Contentgroup>
```

The XML file above is sent to the XMLParser, which is ReferenceObjectIDParser. The ReferenceObjectIDParser is shown above in Section 5.3.2. The returned data are stored in an array and the prepared URL, which is the service API and parameters of the selected object, is retrieved and sent to the module in order to request associated content through the web service framework. The following is the web service connection module that sends a request and receives a response in JSON format. The response contains associated content that is transformed into geometries or objects and visualized on the screen in the AR environment.

```
- (void)connectWebservice:(NSString*)cid
{
        identify URL string to NSURL;
        create URL request with URL string;
        set TimoutInterval;
        set HTTP method to "GET"
        create NSOperationQueue object;
        send URL request
        if(receive response)
        {
            set JSON object with received data;
            if (JSON object is NSDictionary class)
            {
                set JSON object to NSDictionary class;
                call function for building geometries of received data;
            }
        }
}
```

5.5.2 Home-based museum learning scenario

All of the content visualized in the home-based learning is requested from participating content providers after the system track and recognition of trigger images. The web service connection and service request is performed in the AR View where the system gets tracking values of an object in the real scene and recognizes the reference objects. If there is no related content stored in the array, the system sends a request to the participating providers in order to acquire associated content of the recognized object. The pseudocode below is the web service request in the home-based museum learning.

```
if(cosName in object is equal to string "firstobject")
{
    connect web service by sending cos name of the tracked object;
    set cos name of first object in array of content to "serviceObject";
    break;
}
```

The web service connection module in the code is called connectWebServices and the CosName of the tracked image is sent to the module in order to send a request for associated content of a tracked object.

```
- (void)connectWebServices:(NSString *)cosName
{
   create an object of XMLMuseumContentParser;
   create an object of URL reference object;
   search URL reference object in reference object array
```

```
    {
        if(cName in the object is equal to “null”
        {
            break;
        }
        else if (cName in the object is equal to string cosName)
        {
            if(cid in the object is equal to string “null”)
            {
                create an object of MuseumContentElement
                set cName in the object to cosName;
                set elementName to “null”;
                set data to “null”;
                add object to array of content;
            }
            else
            {
                set cid of the object to URL string;
                identify URL string to NSURL;
                create URL request with URL string;
                set TimoutInterval;
                set HTTP method to “GET”
                create NSOperationQueue object;
                send URL request;
                if(received response)
                {
                    call and send received data to function parseXMLfile in
                    MuseumContentParser;
                }
            }
        }
    }
}
```

5.5.3 XMLParser

Open service providers return content in both XML and JSON document formats. In the home-based learning scenario, the selected document, which is the outcome of a service request, is in XML format and it requires the XMLParser in order to extract data from the response document. The following is the functions in XMLParser class that reads through the XML document, fetches the requested data in the document and returns them to the main module.

```
-(NSMutableArray *)parseXMLFile:(NSData *)xmlData
{
    initial NSXMLParser with received XML document
    create array object for content;
    set Delegate for parser;
    parse XML document;
    return array;
}
-(void)parserDidStartDocument:(NSXMLParser *)parser
{
}
-(void)parser:(NSXMLParser *)parser didStartElement:(NSString *)elementName
namespaceURI:(NSString *)namespaceURI qualifiedName:(NSString *)qName
attributes:(NSDictionary *)attributeDict
{
   if(elementName is equal to "Object")
   {
         store elementName;
   }
  else if(elementName is equal to "Description")
   {
         store elementName;
   }
   else if(elementName is equal to "media")
   {
         store elementName;
   }
   else if(elementName is equal to "museum")
   {
         store elementName;
   }
}
-(void) parser:(NSXMLParser *)parser foundCharacters:(NSString *)string
{
   if(elementName is equal to "Object" )
   {
         collect string;
   }
  else if(elementName is equal to "Description")
   {
         collect string;
   }
   else if(elementName is equal to "media")
   {
         collect string;
   }
   else if(elementName is equal to "museum")
   {
         collect string;
   }
}
-(void) parser:(NSXMLParser *)parser didEndElement:(NSString *)elementName
namespaceURI:(NSString *)namespaceURI qualifiedName:(NSString *)qName
{
   if(elementName is equal to “Object”)
   {
         create object for content;
         store elementName in object;
         store collected string in object
         add object to array;
    }
   else if(elementName is equal to “Description”)
   {
         create object for content;
         store elementName in object;
         store collected sting in object
         add object to array;
   }
   else if(elementName is equal to “media”)
   {
         create object for content;
         store elementName in object;
         store collected sting in object
         add object to array;
   }
   else if (elementName is equal to “museum”)
   {
         create object for content;
         store elementName in object;
         store collected sting in object
         add object to array;
   }
}
-(void) parserDidEndDocument:(NSXMLParser *)parse
{
    create object for content;
    store “null” in object;
    add object to array;
}
```

5.5.4 NSJSONSerialization

The system also has a module that de-serializes JSON documents from participating service providers. In the museum-based learning scenario, the acquired document from the Victoria and Albert Museum is in JSON format and is read through by the pseudocode shown below.

```
-(void) createDownloadedDataContent:(NSDictionary *)data
{
  get NSDictionary data in “record”;
  read data in each record
  {
     get NSDictionary data in “fields”;
```

```
        create an object of content;
        get object number data in “object_number”;
        get primary image ID data in “primary_image_id”;
        create URL string from primary image ID;
        set  URL sting to NSURL;
        get image data from URL string;
        create image object from image data;
        add image object to array;
    }
}
```

The structure of JSON documents that are the outcome of each service API is totally different and developers need specifically to create a de-serialization module for the selected service API.

5.6 Third Party Service Request

In the process of content acquisition, there can be content that is the outcome of service APIs such as the source of the cultural object or museum name. For example, the museum name can be used to acquire location (latitude, longitude) of the place as well as a map that shows the position of the museum. This content can be requested from a third party such as Google Maps API by sending a request composed of the name of the service, required parameters and user’s key to the provider. Figure 5.5 presents the UI View that shows a map image of a selected place processed by Google Maps API.

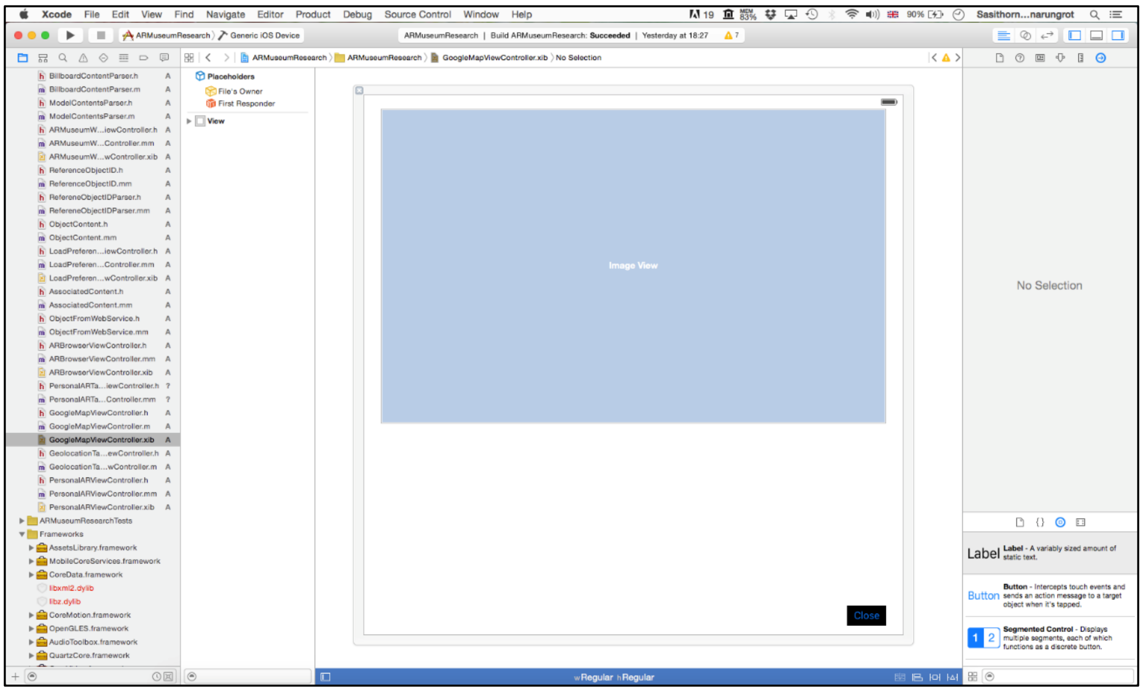

Xcode File Edit View Find Navigate Editor Product Debug Source Control Window Help
ARMuseumResearch | Build ARMuseumResearch: Succeeded | Yesterday at 18:27
ARMuseumResearch 〉 ARMuseumResearch 〉 GoogleMapViewController.xib 〉 No Selection
Placeholders
File's Owner
First Responder
View
GoogleMapViewController.xib
Image View
Close
No Selection
Label
Button
wRegular hRegular

Figure 5.5 The UI View showing a Google Map image

The pseudocode below performs fetching of a selected name on the table view that shows a list of museums or places related to preferred objects saved in the database. The location of a selected name is fetched from the database and is used to create a URL request that is sent to the Google Map Service. A response from the service is passed through to the UIView in Figure 5.5 for showing on the screen.

```
-(void)tableView:(UITableView *)tableView
accessoryButtonTappedForRowWithIndexPath:(NSIndexPath *)indexPath
{
        set fetchedResultsController to selected object;
        get content in object;
        get value from key “location”
        get value from key “latitude”;
        get value from key “longitude”;
        create GoogleMap URL string from fetched latitude and longitude;
        set URL string to NSURL;
        set URL request to NSURL;
        set TimeoutInterval to 30.0;
        set HTTPMethod to “GET”;
        set NSOperationQueue object;
        create web service connection;
        if(received data)
        {
                create image object of acquired data;
                create GoogleMapView object;
                set acquired image to GoogleMapView object;
                present googleMap View;
        }
}
```

The pseudocode below shows the process of sending a request for the location (latitude, longitude) to the Google Map provider by using Google Place API. The obtained location is stored in the database and used to request a map of the selected place.

```
-(void)getLocation:(NSString *)museum
{
        create URL string from acquired museum name;
        create NSMutableString from URL string;
        create NSURL from NSMutableString;
        create URL request from NSURL;
        set TimeoutInterval to 30.0;
        set HTTPMethod to "GET";
        create object of NSOperationQueue;
        create web service connection
        if(received data)
        {
          create JSONobject from received data;
          if (JSON object class is NSDictionary class)
          {
                get data from key "results";
                get data from key "geometry";
                get data from key "location";
                get data from key "lat";
                get data from key "lng";
                add acquired locations to array;
          }
        }
}
```

5.7 Interaction and Object Selection

Mobile users are able to interact with active content on their mobile's screen, such as selecting preferred content on the AR View that can be saved into the database and revealed in the AR Browser. The module named touchesBegan() receives users' interaction or touching on the AR View so that the system gets the position of a touched object in the AR View. After that, the system gets the geometry on the touched location and compares it to the media content geometries in the library. The pseudocode in section 5.5.1 searches if the touched geometry is the MoreDetail object. The system then sends a web service request to the participating content provider for associated content of the tracked object. The URL of the provider is stored in the

AssociatedMuseumContent.xml file and this file has to be read through by an XMLParser function called ReferenceObjectIDParser in order to extract the URL or service API of each targeted object. Users are able to select other active geometries on the AR View in order to create personal AR environments by saving the selected content into the database where the preferred content can be retrieved and revealed in other situations by tracking a trigger image. The following is the code in the touchesBegan() where the selected geometry is checked to see if it is active media content on the AR View.

```
- (void) touchesBegan:(NSSet *)touches withEvent:(UIEvent *)event
{
   get interaction on screen;
   get location of the touch object;
   get geometry from location in ViewportCoordinates;
   search touched geometry in the array of content
   {
         if(found touched geometry in the array)
         {
                 add selected geometry in the array of selected object;
         }
   }
}
```

5.8 Saving Selected Content

Mobile users are able to save preferred content that has been selected in the AR View by touching the Save button; the system shows a dialogue box asking the user to type in the name of their preference and then save the selected media content into the database. The following is the code for setting up the alert view containing buttons and a text field.

```
- (IBAction)saveSelectedGeometires:(id)sender
{
      MuseumObjects *object = [[MuseumObjects alloc]init];
      object.cosName = @"null";
      [selectedContent addObject:object];
       create AlertView object;
       identify messages and titles to objects on the AlertView;
       set AlertViewStyle;
       set TextField index;
       set KeyboardType;
       show AlertView;
}
```

The code below is to show the alertView in order to allow users either to save their preferred media content into the database or to cancel the saving process.

```
-(void) alertView:(UIAlertView *)alertView clickedButtonAtIndex:
(NSInteger)buttonIndex
{
      get index of the clicked button;
      if(button title is “Cancel”)
      {
        create new array object;
      }
      else if(button title is “OK”)
      {
        get text from TextField;
        save selected content and text into the database;
      }
}
```

After touching the OK button, the module sends the preference name to the saveToCoreDataDB module that saves the selected media content into the object-base database. The following is the code in the saveToCoreDataDB module called by the alertView() for setting up and working with the database.

```
-(void)saveToCoreDataDB:(NSString *)preferrenceName
{
    create new object for entity named “PreferredObjects”;
    if (found new object)
    {
        add selected COS name to object;
        add preferenceName to object;
        add date to object;
        add source of content to object;
        for (read selected object in array)
        {
                if (COS name in object is equal to “null”)
                {
                        break;
                }
                if (object type in object is equal to “media”)
                {
                        create new object for entity named “ImageBillboard”;
                        add imagePath to object;
                        add image to object;
                        add object to PreferredObjects object;
                }
                else if (object type in object is equal to “object” or “description”)
                {
                        create new object for entity named “TextBillboard”;
                        add text to object;
                        add billboard type to object;
                        add object to PreferredObjects object;
                    }
                    else if (object type in object is equal to “museum”)
                    {
                        preferredObjects.museum = savedObjects.data;
                        add museum data to PreferredObjects object;
                    }
            }
            save PreferredObjects object;
        }
    }
}
```

Figure 5.6 illustrates the structure of the Core Data that is the object-based databased operated by XCode. The Data Model contains entities used to store the details of selected content.

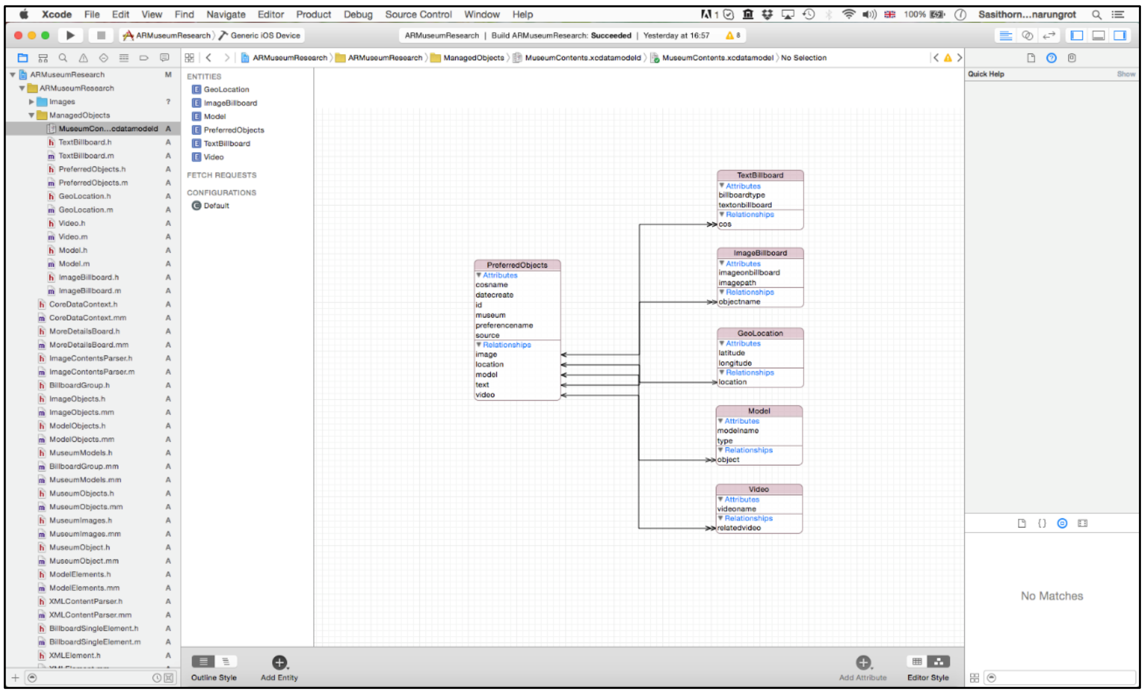


Figure 5.6 The Data Model of the application

The following is the class of the designed Core Data model representing the database for storing preferred content. The class is composed of components in the entity and subclasses that are used in order to store data in the database.

PreferredObjects.h

```
#import <Foundation/Foundation.h>
#import <CoreData/CoreData.h>
```

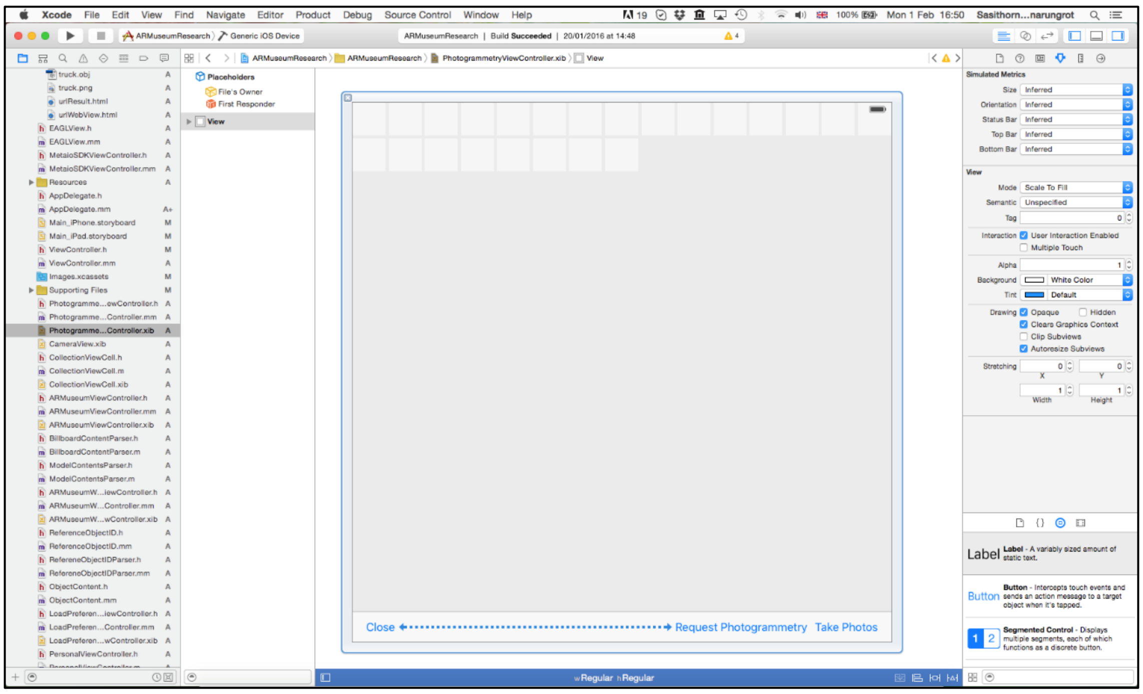
```
@class GeoLocation, ImageBillboard, Model, TextBillboard, Video;
@interface PreferredObjects : NSManagedObject
@property (nonatomic, retain) NSString * cosname;
@property (nonatomic, retain) NSDate * datecreate;
@property (nonatomic, retain) NSNumber * id;
@property (nonatomic, retain) NSString * museum;
@property (nonatomic, retain) NSString * preferencename;
@property (nonatomic, retain) NSString * source;
@property (nonatomic, retain) NSSet *image;
@property (nonatomic, retain) GeoLocation *location;
@property (nonatomic, retain) NSSet *model;
@property (nonatomic, retain) NSSet *text;
@property (nonatomic, retain) NSSet *video;
@end
@interface PreferredObjects (CoreDataGeneratedAccessors)
- (void)addImageObject:(ImageBillboard *)value;
- (void)removeImageObject:(ImageBillboard *)value;
- (void)addImage:(NSSet *)values;
- (void)removeImage:(NSSet *)values;
- (void)addModelObject:(Model *)value;
- (void)removeModelObject:(Model *)value;
- (void)addModel:(NSSet *)values;
- (void)removeModel:(NSSet *)values;
- (void)addTextObject:(TextBillboard *)value;
- (void)removeTextObject:(TextBillboard *)value;
- (void)addText:(NSSet *)values;
- (void)removeText:(NSSet *)values;
- (void)addVideoObject:(Video *)value;
- (void)removeVideoObject:(Video *)value;
- (void)addVideo:(NSSet *)values;
- (void)removeVideo:(NSSet *)values;
@end
```

PreferredObjects.m

```
#import "PreferredObjects.h"
#import "GeoLocation.h"
#import "ImageBillboard.h"
#import "Model.h"
#import "TextBillboard.h"
#import "Video.h"
@implementation PreferredObjects
@dynamic cosname;
@dynamic datecreate;
@dynamic id;
@dynamic museum;
@dynamic preferencename;
@dynamic source;
@dynamic image;
@dynamic location;
@dynamic model;
@dynamic text;
@dynamic video;
@end
```

5.9 Taking Photos and Photogrammetry Service Request

One of the features in museum-based learning scenarios applied through SOMARA is a photogrammetry service request that demonstrates the proposed content acquisition over SOA. The photogrammetry service request requires images of a targeted object that is sent through a web service framework to the service provider. In this example SOMARA application, there is a View Controller used to support a camera view that captures images of a preferred object in the museum environment. Figure 5.7 presents the photogrammetry View Controller in museum-based learning where it offers a feature for mobile users to take photographs of a targeted object and the photographs taken are shown on the View Controller.



Simulated Metrics
Size: Inferred
Orientation: Inferred
Status Bar: Inferred
Top Bar: Inferred
Bottom Bar: Inferred

View
Mode: Scale To Fill
Semantic: Unspecified
Tag: 0
Interaction: ✓ User Interaction Enabled; ☐ Multiple Touch
Alpha: 1
Background: White Color
Tint: Default
Drawing: ✓ Opaque; ☐ Hidden; ✓ Clears Graphics Context; ☐ Clip Subviews; ✓ Autoresize Subviews
Stretching: X 0, Y 0, Width 1, Height 1

Label - A variably sized amount of static text.
Button - Intercepts touch events and sends an action message to a target object when it's tapped.
Segmented Control - Displays multiple segments, each of which functions as a discrete button.

Figure 5.7 A View Controller supporting a Camera View

The View Controller contains a Collection View holding a Collection View Cell. The Collection View Cell is used to present a background image, which is a captured image of the targeted object. Figure 5.8 illustrates the Collection View Cell that has an Image View on the top in order to show a captured image on the Collection View.

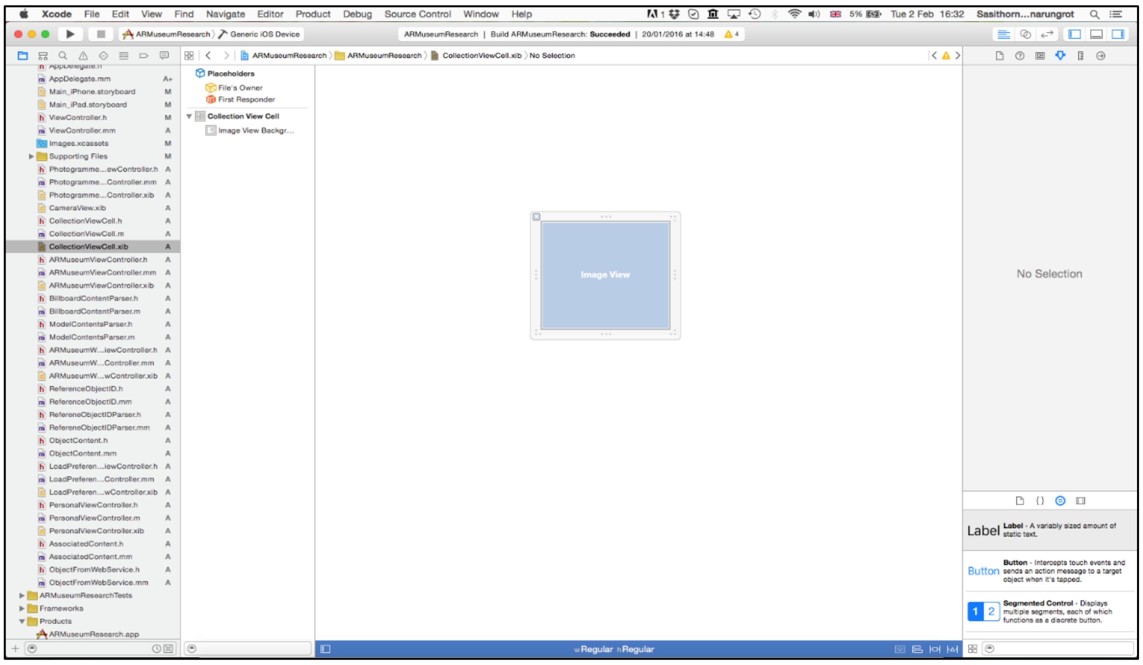

Xcode File Edit View Find Navigate Editor Product Debug Source Control Window Help
ARMuseumResearch | Build ARMuseumResearch: Succeeded | 20/01/2016 at 14:48
ARMuseumResearch > ARMuseumResearch > CollectionViewCell.xib > No Selection
Placeholders
File's Owner
First Responder
Collection View Cell
Image View Backgr...
Image View
No Selection
Label - A variably sized amount of static text.
Button - Intercepts touch events and sends an action message to a target object when it's tapped.
Segmented Control - Displays multiple segments, each of which functions as a discrete button.
wRegular hRegular

Figure 5.8 The Collection View Cell

On the View Controller, there is also a Camera View installed on the top in order to operate the embedded camera and present the current scene on the view so that mobile users can take photographs that are shown on the View Controller. Figure 5.9 presents the Camera View on top of the View Controller.

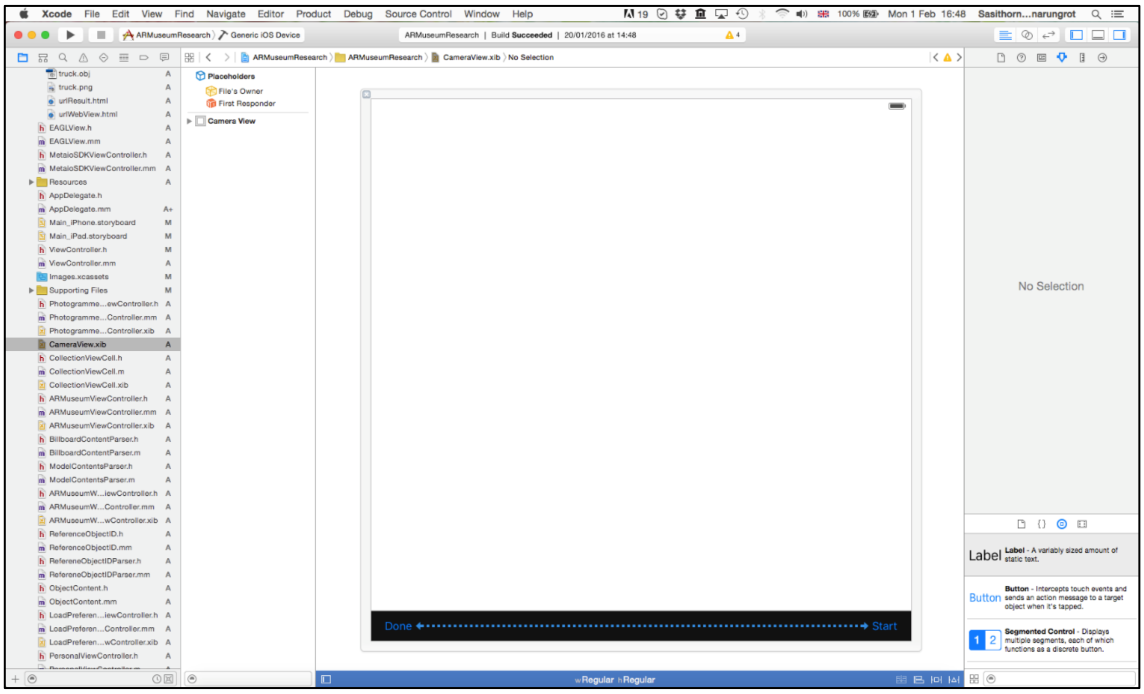

Xcode File Edit View Find Navigate Editor Product Debug Source Control Window Help
ARMuseumResearch | Build Succeeded | 20/01/2016 at 14:48
ARMuseumResearch > ARMuseumResearch > CameraView.xib > No Selection
Placeholders
File's Owner
First Responder
Camera View
Done
Start
No Selection
Label - A variably sized amount of static text.
Button - Intercepts touch events and sends an action message to a target object when it's tapped.
Segmented Control - Displays multiple segments, each of which functions as a discrete button.
wRegular hRegular

Figure 5.9 The Camera View

The camera view shows a real scene and is used to take photographs of a targeted object continuously after touching the Start button. Touching the Stop button ends taking photographs and shows all the taken photographs on the screen. The code below presents the Camera View after selecting Take Photos. The Camera View displays on the screen and allows users to take photographs of a targeted object.

```
- (IBAction)takePhotos:(id)sender
{
    show ImagePicker for camera;
}
- (void)showImagePickerForSourceType:(UIImagePickerControllerSourceType)source
Type
{
   create object of UIImagePickerController;
   set presentation style to object;
   set source type to object;
   set delegate to object;
   if(source type is camera)
   {
      set show camera controls
      load view named "CameraView";
      set cameraView frame to imagePickerController object;
   }
   present imagePickerController object;
}
```

After selecting Start on the Camera View, the system takes photographs of the object. Each photograph automatically taken is stored in an array that can be used to display on the Collection View or sent as a request to the photogrammetry service provider.

```
- (void)imagePickerController:(UIImagePickerController *)picker
didFinishPickingMediaWithInfo:(NSDictionary *)info
{
   set taken value to image object;
   add image object to array
   if(cameTimer)
   {
      return;
   }
  update the view of taken photos;
}
```

5.10 AR Browser

AR Browser is used to present personal AR environments as well as an obtained 3D model of a preferred object from a photogrammetry service. The personal AR environments are composed of preferred objects that are geometries visualized on the AR View. In the museum-based learning scenario, mobile users are able to select and save preferred content on the screen and use the AR Browser for revealing saved content at home or in other situations. In this project, AR Browser is a sub-view implemented on the UITabBar Controller used to contain and organize sub-views, which can be selected and viewed by mobile users. Figure 5.10 shows the UITabBar Controller that controls the tab bar of a UI View.

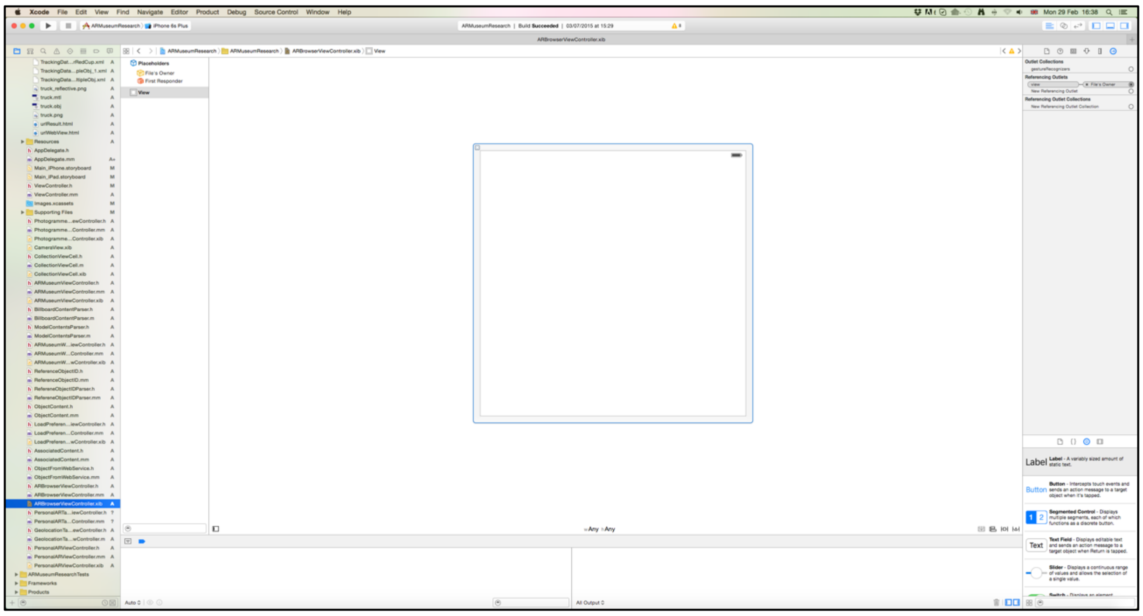


Figure 5.10 The UITabBar Controller of the Table views

The code below shows the UITabBar Controller class and its sub-views. Each sub-view is organized by UINavigation Controller in order to present each view by clicking a tab on the tab-bar as shown in the code below.

```
- (void)viewDidLoad
{
        load view;
        create tabBarController object;
        add tabBarController view to current view;
        create TableView object for preference names;
        create UINavigationController object with TableView;
        create TableView object for geolocations;
        create UINavigationController object with TableView;
        set all created objects to tabBarController object;
}
```

One of the sub-views in the UINavigation Controller is a view for revealing personal AR environments created by mobile users. The view is implemented by UITableView Controller that fetches data stored in the CoreData or database and presents it on the Table View so that mobile users can select each preference and view selected content in a real scene. The following is the code of a sub-view utilising UITableView Contoller and performing data retrieving, entity control and data presentation.

```
-(NSManagedObjectModel *)manageObjectModel
{
   if (found the object model)
   {
      return the object model;
   }
  create URL path for resource;
  initialize URL path to the object model;
  return object model;
}
-(NSPersistentStoreCoordinator *)persistentStoreCoordinator
{
   if (found the persistentStoreCoordinator)
   {
      return persistentStoreCoordinator;
   }
   create a string to store path to the database file;
   create URL from the store path string;
   identify objects and keys on the database;
   initialize presistentStoreCoordinator with the object model;
   return persistentStoreCoordinator;
}
-(NSFetchedResultsController *)fetchedResultsController
{
   if (found fetchedResultsController)
   {
      return fetchedResultsController;
   }
   create fetch request object;
   identify entity "PreferredObjects" in object context;
   set entity to fetch request object;
   set fetch batch size to 20;
   initialize sort descriptor object with key "datecreate";
   initialize array object with sort descriptor;
   set sort descriptor to fetch request;
   initialize FetchedResults Controller with fetch request in object context;
   set delegate to FetchedResultsController;
   return fetchedResultsController;
}
-(void)controllerWillChangeContent:(NSFetchedResultsController *)controller
{
    update Table View;
}
-(void)controllerDidChangeContent:(NSFetchedResultsController *)controller
{
   end updating Table View;
}
- (void)viewDidLoad
{
   set persistentStoreCoordinator;
   create context object;
   set PersistentStoreCoordinator to context object;
}
- (NSInteger)numberOfSectionsInTableView:(UITableView *)tableView
{
   count number of sections in Table View;
   return number;
}
- (NSInteger)tableView:(UITableView *)tableView numberOfRowsInSection:
  (NSInteger)section
{
    fetch object in each section;
    return number of objects;
}
- (UITableViewCell *)tableView:(UITableView *)tableView cellForRowAtIndexPath:
  (NSIndexPath *)indexPath
{
    identify cell idenifier to "ContentCell";
   dequeue cell to Table View cell;
   if (cell is empty)
   {
      initialize cell with default style;
   }
   configure cell at index path;
   return cell;
}
-(void)configureCell:(UITableViewCell *)cell atIndexPath:(NSIndexPath *)indexPath
{
   fetch object at index path;
   set text in textLabel from key "preferencename";
}
-(void)tableView:(UITableView *)tableView accessoryButtonTappedForRow
  WithIndexPath:(NSIndexPath *)indexPath
{
  fetch object at selected row;
  load PersonalARViewController;
  set selected content to PersonalARViewController;
  present PersonalARViewController;
 }
}
@end
```

The TableView lists all preference names of personal AR environments stored in the database and allows mobile users to select one in order to view the saved media content in the AR environment by tracking an image referred to a targeted cultural object. The AR Browser view of the application supporting content utilization is presented in Figure 5.10.

Another sub-view in the UINavigation Controller is a Table View showing a list of museums or places of selected objects fetched from the database. This Table View utilizes a Google Map API called Static Map for processing and providing a map image of the selected museum or place. The service request, URL, parameters and UI View can be seen in Section 5.6.

5.11 Summary

This chapter explains the development of the SOMARA-based mobile AR application using iOS platform and XCode as an example or reference application. The development is composed of front-end systems including views and interfaces, and back-end systems including a web service framework, AR tasks and interoperability tasks. The application is applied in MLAs composed of museum-based and home-based learning scenarios that require different functions.

Chapter VI

6 System and Unit Testing

This chapter describes system testing and evaluation of the SOMARA-based novel mobile AR application that performs AR tasks, content acquisition and content utilization. This example SOMARA application is a mobile AR client running on an iPad and it is developed on a web service framework exploiting service orientation, which requires mobile or wireless connection, in order to access open services and accomplish interoperability tasks. The mobile AR client is developed in MLA scenarios comprising museum-based and home-based learning. These scenarios require different tracking techniques and content acquisition schemes in order to encourage flexible museum exhibition through AR. The application has a mobile interface where mobile users are able to perform object tracking, visualize content and interact with the system. The web service framework is a back-end system that creates connections and accesses participating service providers (e.g. third party or the application's parent museum) by implementing service APIs. Example service APIs used in this mobile AR client in order to support MLAs and provide associated content to the sample reference objects are the Victoria and Albert Museum API and the Sierra Leone Heritage API. Figure 6.1 shows the main page of the SOMARA application on which mobile users are able to select museum learning or choose supporting functions in the system.

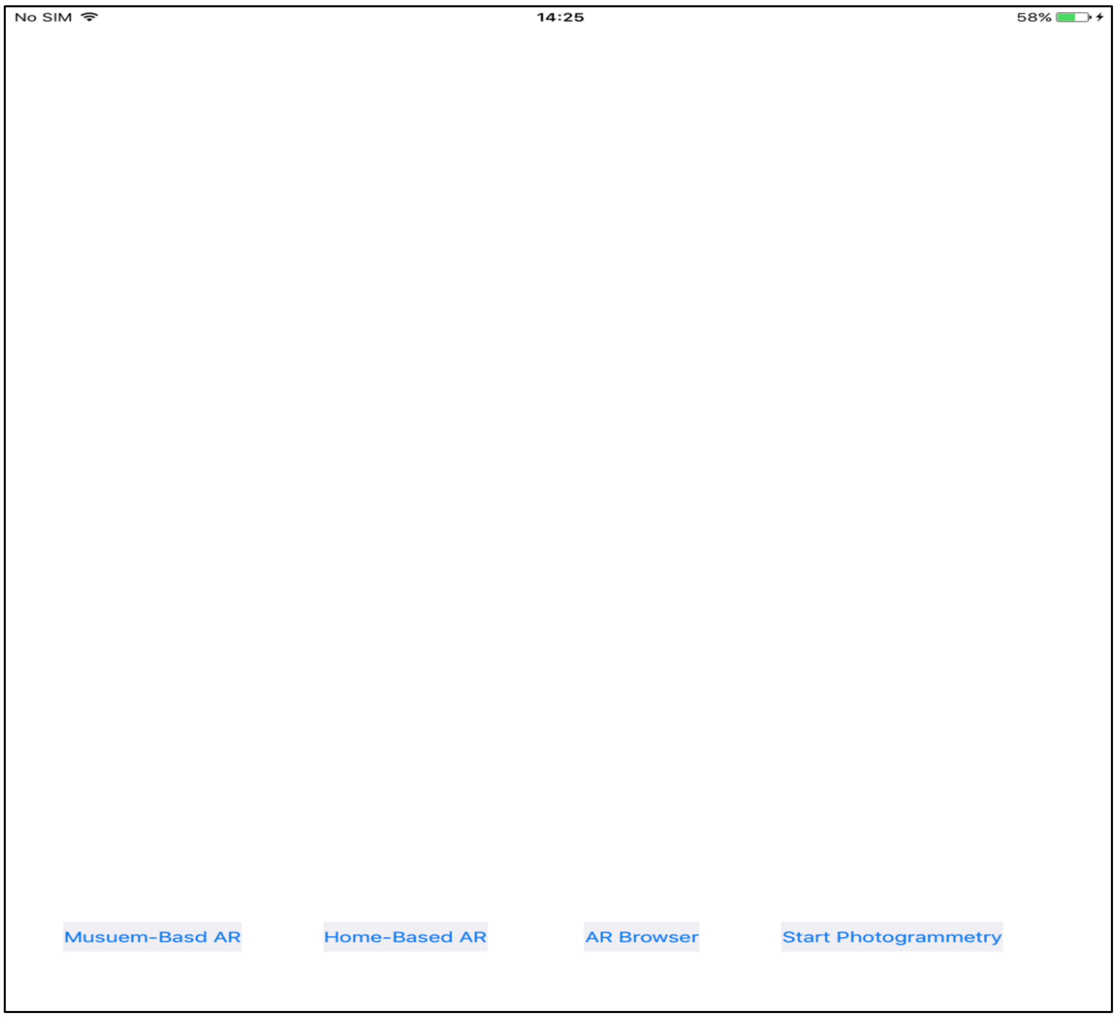


Figure 6.1 The main application page running on an iPad

The main page contains buttons including Museum-based AR, Home-based AR, AR Browser and Start Photogrammetry. The Museum-based AR is selected in order to perform museum-based learning where mobile users can perform object tracking, augmentation and can send web service requests for associated media content inside the museum. The following is a museum-based learning scenario of the novel mobile AR application that supports physical object tracking, content visualization and content acquisition.

6.1 Museum-based Learning Scenario

In this scenario, the system performs physical object tracking that requires cultural reference objects and pre-defined related media content stored in the library. Figure 6.2 presents the AR View of museum-based learning.

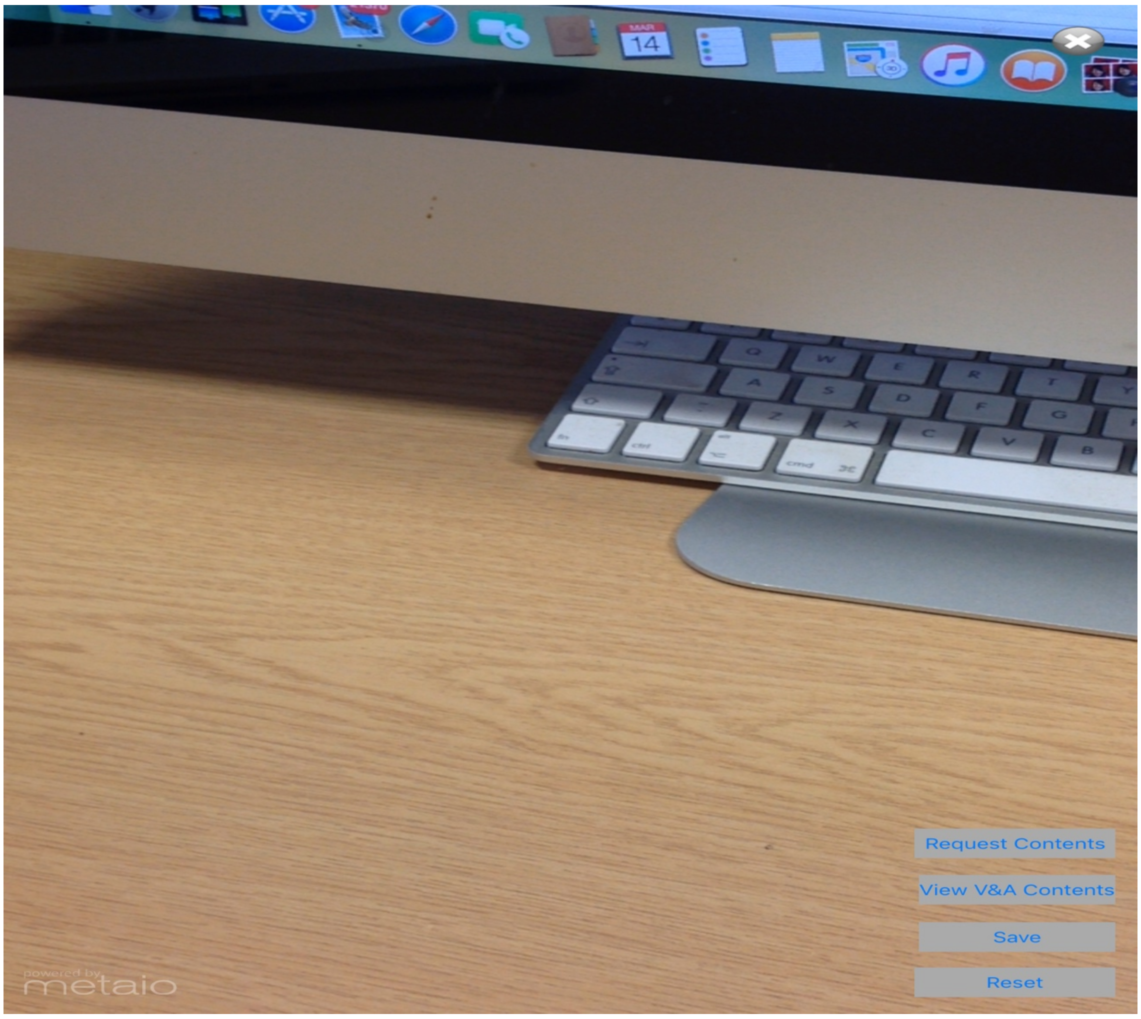


Figure 6.2 The AR View of museum-based learning scenario

When the system tracks and recognizes an object in the AR view, the system visualizes related media content of a targeted object. The following is an example of the outcome of object tracking in a museum-based learning scenario and the process of content acquisition that allows mobile users to send a request for associated media content of a targeted object.

6.1.1 Object tracking

In museum-based learning scenarios, object tracking can be done in a museum that exhibits cultural artifacts. Mobile users are able to track a reference object in order to view related media content visualized on the real scene. Figure 6.3 shows a tracked targeted object and its related content stored in the library of the system.

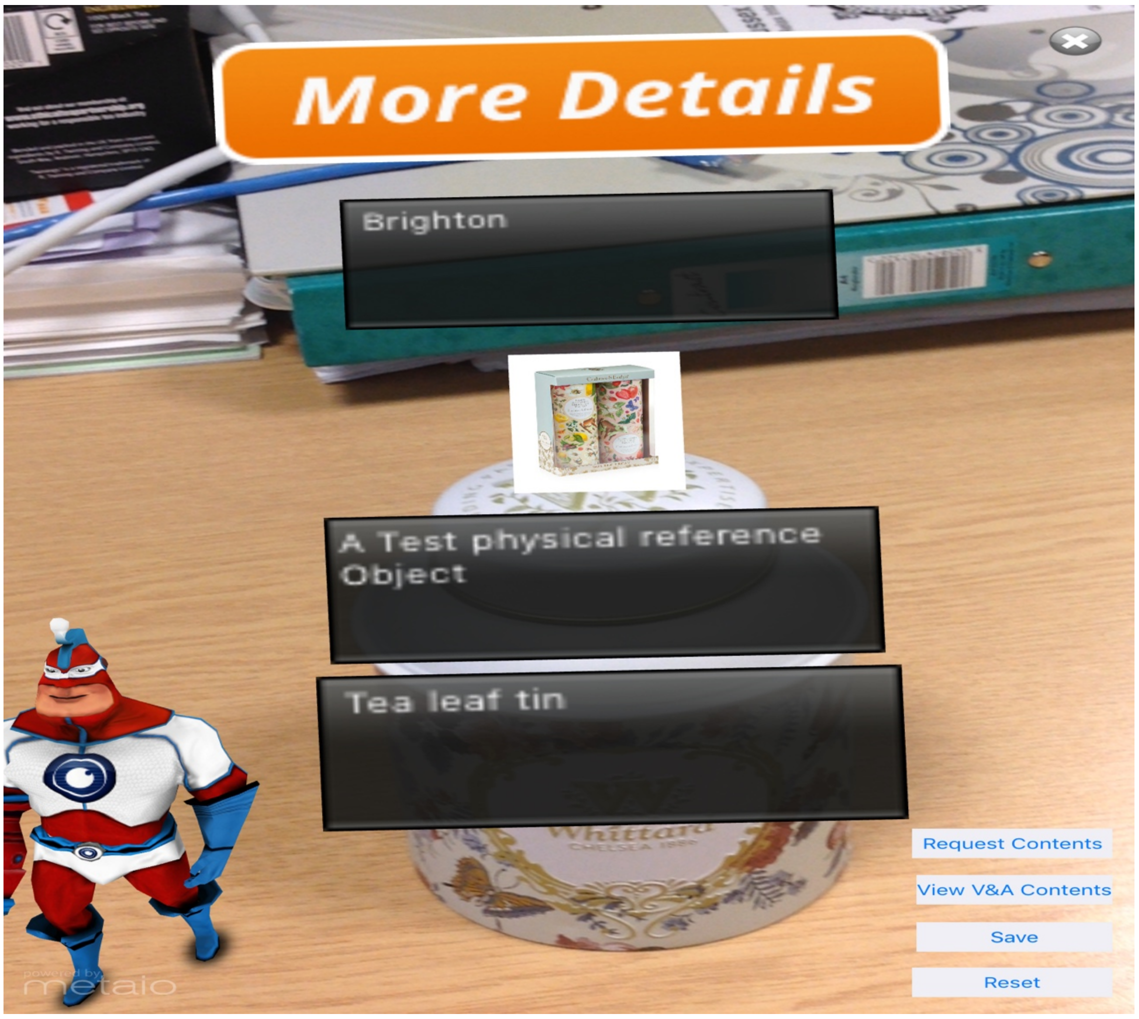


Figure 6.3 The related media content of a targeted object

Media content visualized in the AR view are text, images and a 3D model. Mobile users are able to acquire associated content of a targeted object from participating service providers by touching the More Details geometry; the system then sends a web service request to the open provider. The obtained content is visualized on the screen instead of the active content. Figure 6.4 presents the content acquisition of the associated content of a sample object from the Victoria and Albert Museum.

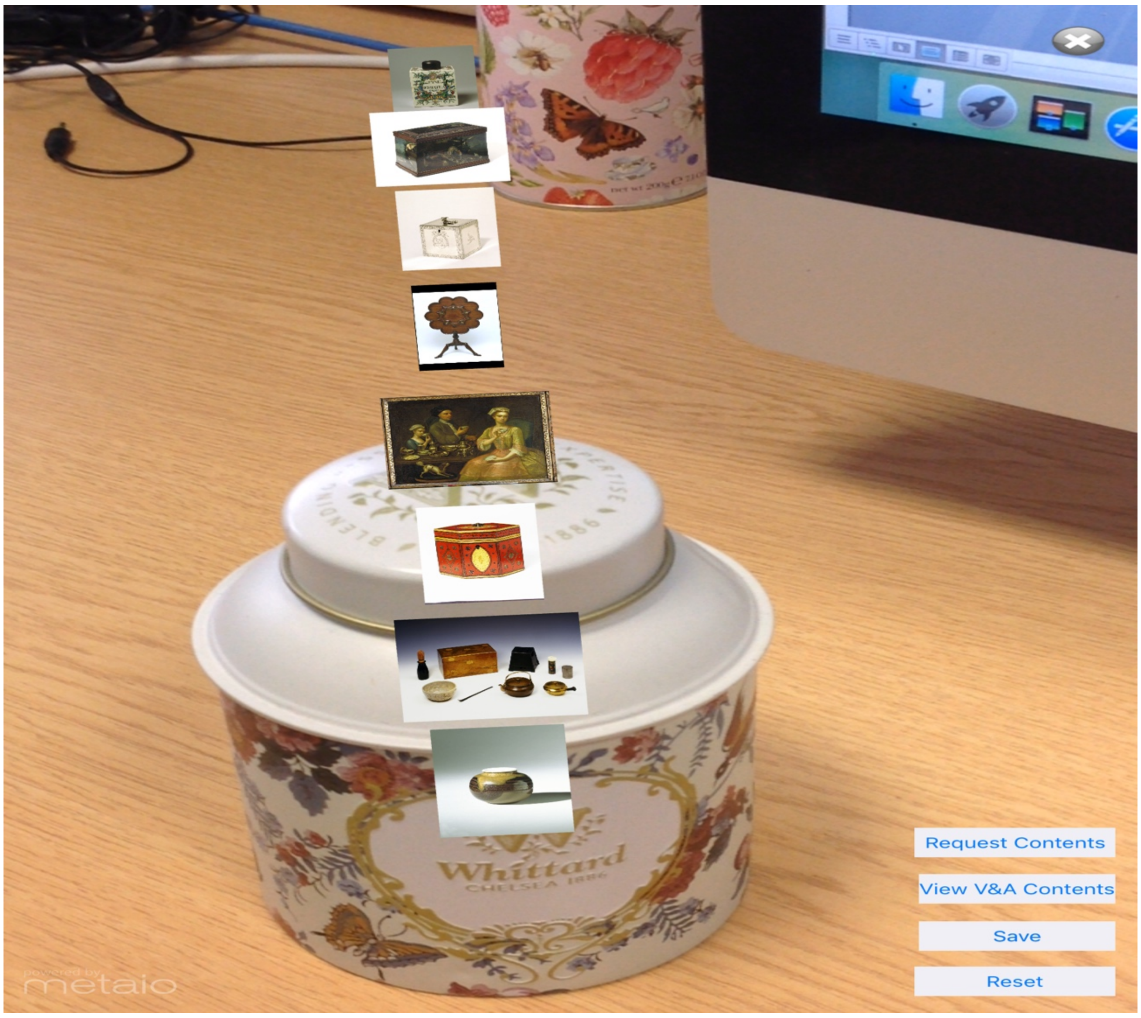
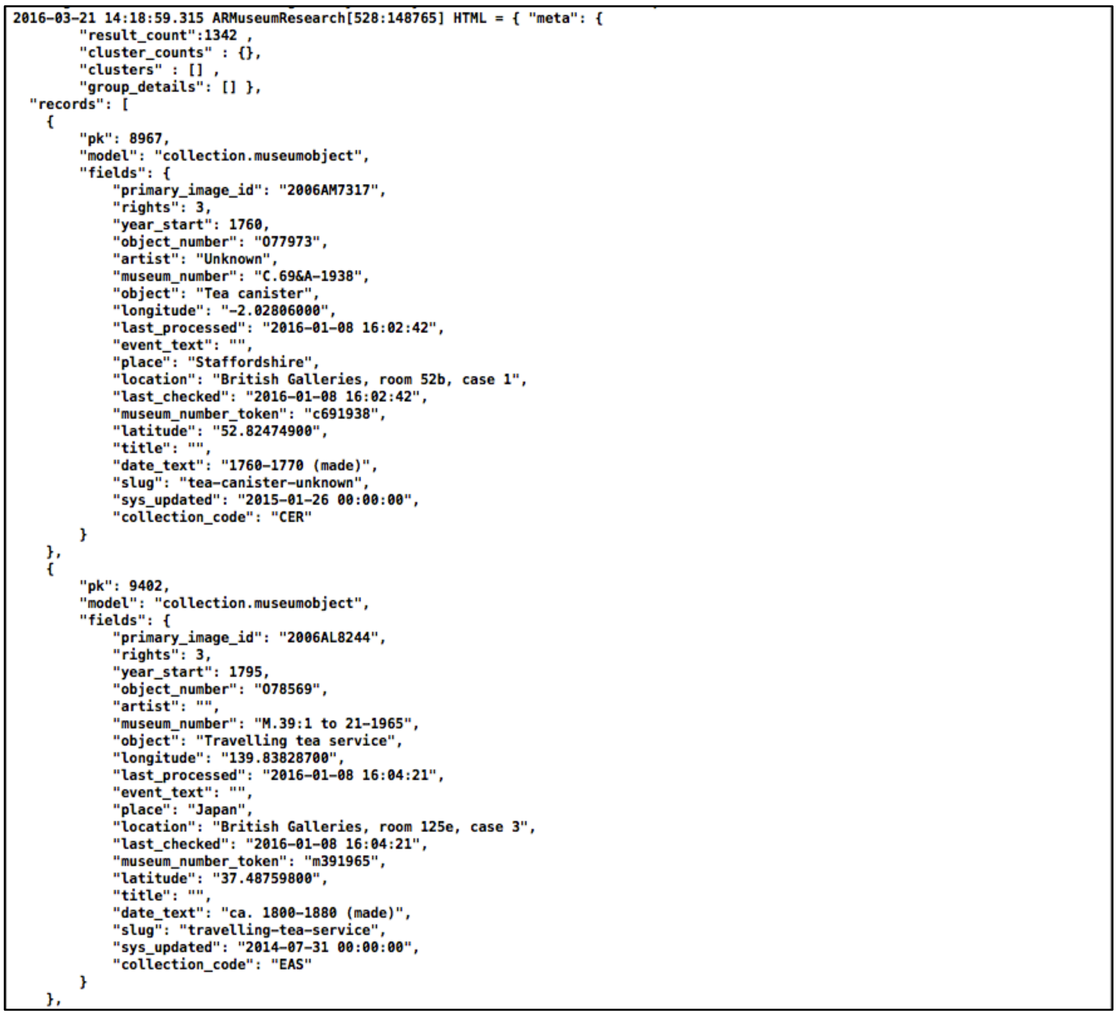


Figure 6.4 The obtained associated content of a tracked physical object

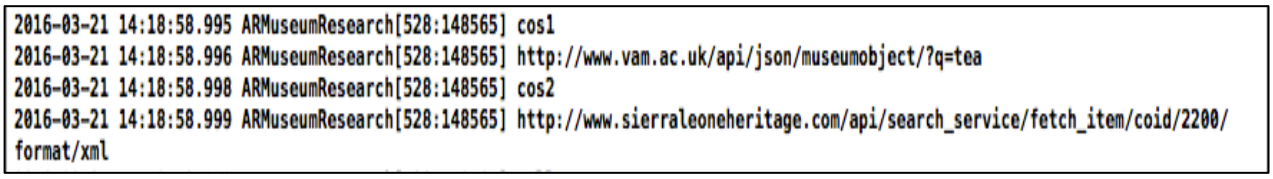
The associated content or images visualized on the AR scene in Figure 6.4 is the response from the Victoria and Albert Museum API that can be in the form of either a JSON or an XML document. Figure 6.5 shows the web service response of the Victoria and Albert API in JSON format whereby the required data are extracted and transformed into geometries that can be visualized and seen on the AR view.



```
2016-03-21 14:18:59.315 ARMuseumResearch[528:148765] HTML = { "meta": {
        "result_count":1342 ,
        "cluster_counts" : {},
        "clusters" : [] ,
        "group_details": [] },
  "records": [
    {
        "pk": 8967,
        "model": "collection.museumobject",
        "fields": {
            "primary_image_id": "2006AM7317",
            "rights": 3,
            "year_start": 1760,
            "object_number": "O77973",
            "artist": "Unknown",
            "museum_number": "C.69&A-1938",
            "object": "Tea canister",
            "longitude": "-2.02806000",
            "last_processed": "2016-01-08 16:02:42",
            "event_text": "",
            "place": "Staffordshire",
            "location": "British Galleries, room 52b, case 1",
            "last_checked": "2016-01-08 16:02:42",
            "museum_number_token": "c691938",
            "latitude": "52.82474900",
            "title": "",
            "date_text": "1760-1770 (made)",
            "slug": "tea-canister-unknown",
            "sys_updated": "2015-01-26 00:00:00",
            "collection_code": "CER"
        }
    },
    {
        "pk": 9402,
        "model": "collection.museumobject",
        "fields": {
            "primary_image_id": "2006AL8244",
            "rights": 3,
            "year_start": 1795,
            "object_number": "O78569",
            "artist": "",
            "museum_number": "M.39:1 to 21-1965",
            "object": "Travelling tea service",
            "longitude": "139.83828700",
            "last_processed": "2016-01-08 16:04:21",
            "event_text": "",
            "place": "Japan",
            "location": "British Galleries, room 125e, case 3",
            "last_checked": "2016-01-08 16:04:21",
            "museum_number_token": "m391965",
            "latitude": "37.48759800",
            "title": "",
            "date_text": "ca. 1800-1880 (made)",
            "slug": "travelling-tea-service",
            "sys_updated": "2014-07-31 00:00:00",
            "collection_code": "EAS"
        }
    },
```

Figure 6.5 The JSON response from the Victoria and Albert Museum API

The service API of each reference object is stored in a content configuration file in XML format. The content configuration file is sent to XMLParser for extracting data. Figure 6.6 shows the service APIs or URL of participating service providers related to each reference object.



```
2016-03-21 14:18:58.995 ARMuseumResearch[528:148565] cos1
2016-03-21 14:18:58.996 ARMuseumResearch[528:148565] http://www.vam.ac.uk/api/json/museumobject/?q=tea
2016-03-21 14:18:58.998 ARMuseumResearch[528:148565] cos2
2016-03-21 14:18:58.999 ARMuseumResearch[528:148565] http://www.sierraleoneheritage.com/api/search_service/fetch_item/coid/2200/
format/xml
```

Figure 6.6 The service APIs in a content configuration file

The media content of reference objects in a museum-based leaning scenario can be requested from an open server as well as storing media content in the library in advance. If the museum has digital collections or digital content stored in the server providing open services, the mobile AR client is then able to obtain related content of reference objects though a web service framework. Figure 6.7 presents the AR view that visualises acquired related content of a tracked physical object.

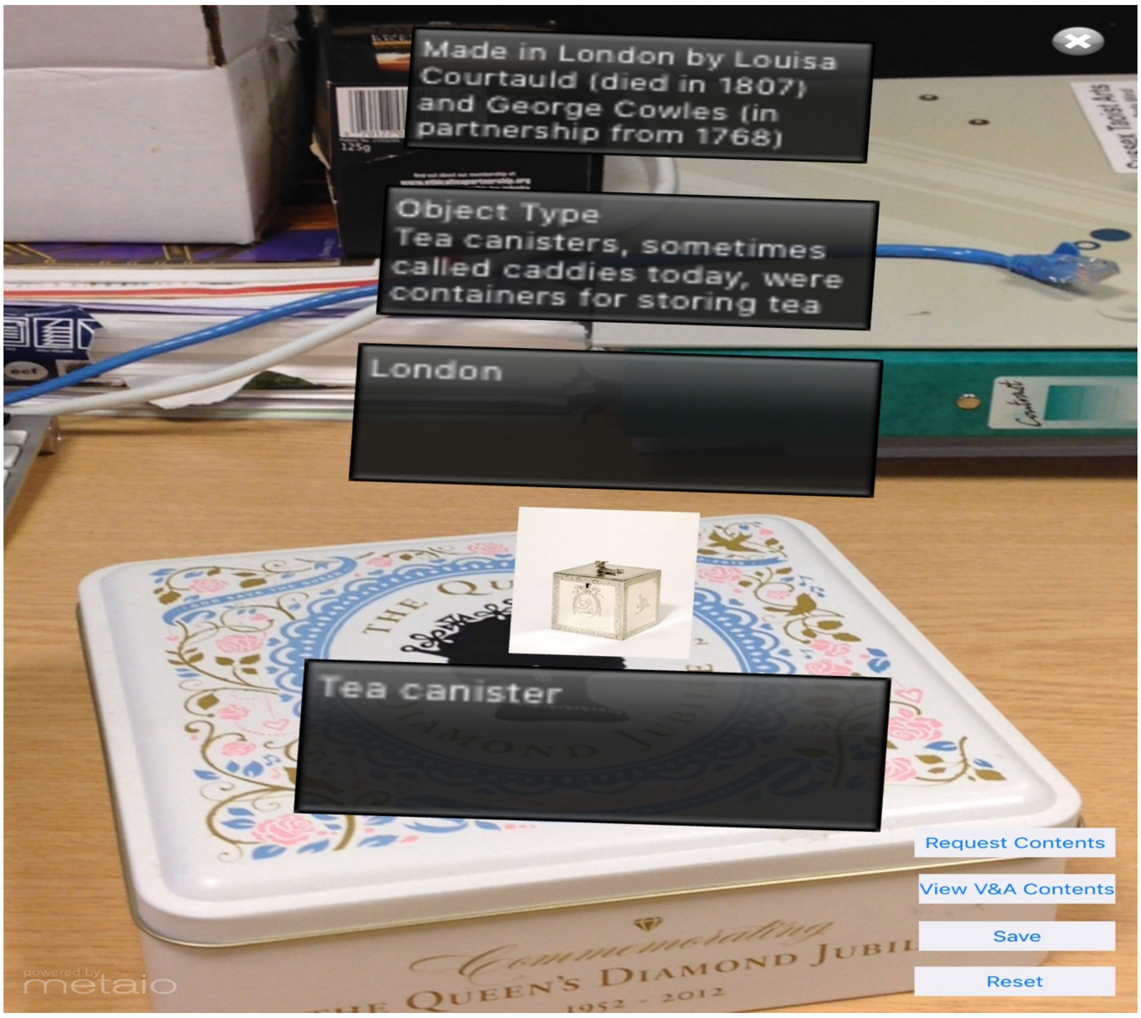
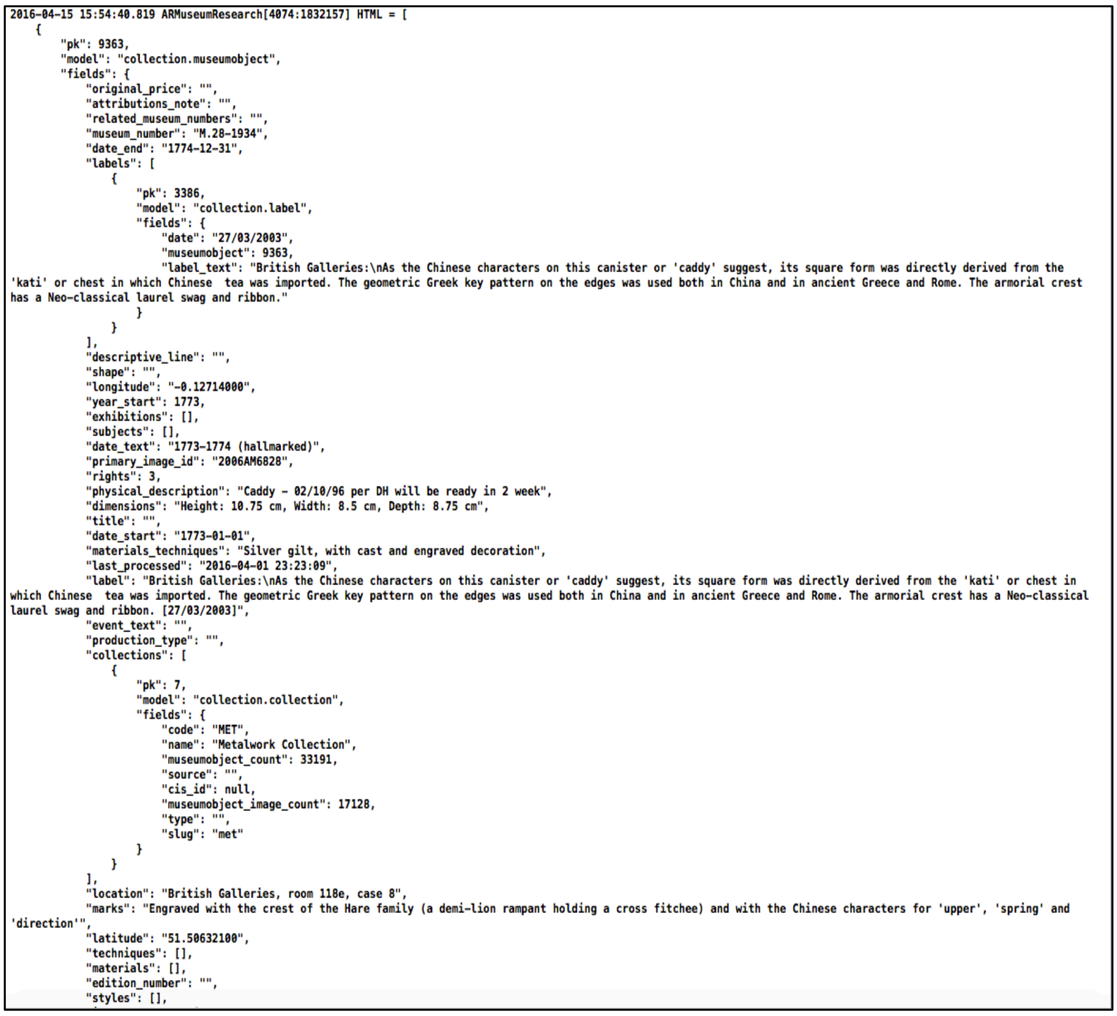

Made in London by Louisa Courtauld (died in 1807) and George Cowles (in partnership from 1768)
Object Type
Tea canisters, sometimes called caddies today, were containers for storing tea
London
Tea canister
Request Contents
View V&A Contents
Save
Reset
powered by metaio

Figure 6.7 The obtained related content of a tracked physical object

The response document from the Victoria and Albert Museum API containing content visualised in Figure 6.8 is in JSON format; it is read through and the data extracted. The obtained data are transformed to geometries and stored in the library. Figure 6.8 illustrates the web service response document in JSON format.



```
2016-04-15 15:54:40.819 ARMuseumResearch[4074:1832157] HTML = [
    {
        "pk": 9363,
        "model": "collection.museumobject",
        "fields": {
            "original_price": "",
            "attributions_note": "",
            "related_museum_numbers": "",
            "museum_number": "M.28-1934",
            "date_end": "1774-12-31",
            "labels": [
                {
                    "pk": 3386,
                    "model": "collection.label",
                    "fields": {
                        "date": "27/03/2003",
                        "museumobject": 9363,
                        "label_text": "British Galleries:\nAs the Chinese characters on this canister or 'caddy' suggest, its square form was directly derived from the
'kati' or chest in which Chinese  tea was imported. The geometric Greek key pattern on the edges was used both in China and in ancient Greece and Rome. The armorial crest
has a Neo-classical laurel swag and ribbon."
                    }
                }
            ],
            "descriptive_line": "",
            "shape": "",
            "longitude": "-0.12714000",
            "year_start": 1773,
            "exhibitions": [],
            "subjects": [],
            "date_text": "1773-1774 (hallmarked)",
            "primary_image_id": "2006AM6828",
            "rights": 3,
            "physical_description": "Caddy - 02/10/96 per DH will be ready in 2 week",
            "dimensions": "Height: 10.75 cm, Width: 8.5 cm, Depth: 8.75 cm",
            "title": "",
            "date_start": "1773-01-01",
            "materials_techniques": "Silver gilt, with cast and engraved decoration",
            "last_processed": "2016-04-01 23:23:09",
            "label": "British Galleries:\nAs the Chinese characters on this canister or 'caddy' suggest, its square form was directly derived from the 'kati' or chest in
which Chinese  tea was imported. The geometric Greek key pattern on the edges was used both in China and in ancient Greece and Rome. The armorial crest has a Neo-classical
laurel swag and ribbon. [27/03/2003]",
            "event_text": "",
            "production_type": "",
            "collections": [
                {
                    "pk": 7,
                    "model": "collection.collection",
                    "fields": {
                        "code": "MET",
                        "name": "Metalwork Collection",
                        "museumobject_count": 33191,
                        "source": "",
                        "cis_id": null,
                        "museumobject_image_count": 17128,
                        "type": "",
                        "slug": "met"
                    }
                }
            ],
            "location": "British Galleries, room 118e, case 8",
            "marks": "Engraved with the crest of the Hare family (a demi-lion rampant holding a cross fitchee) and with the Chinese characters for 'upper', 'spring' and
'direction'",
            "latitude": "51.50632100",
            "techniques": [],
            "materials": [],
            "edition_number": "",
            "styles": [],
```

Figure 6.8 The web service response document in JSON format

The associated objects of the targeted object can be requested from the Victoria and Albert Museum API by touching the active image on the AR View. The system then sends a request to the provider along with a parameter that is the name of the cultural object. A response is sent back in JSON document format. Figure 6.9 presents an image of associated objects of the targeted sample object. The JSON response document of an associated content request can be seen in Figure 6.5.

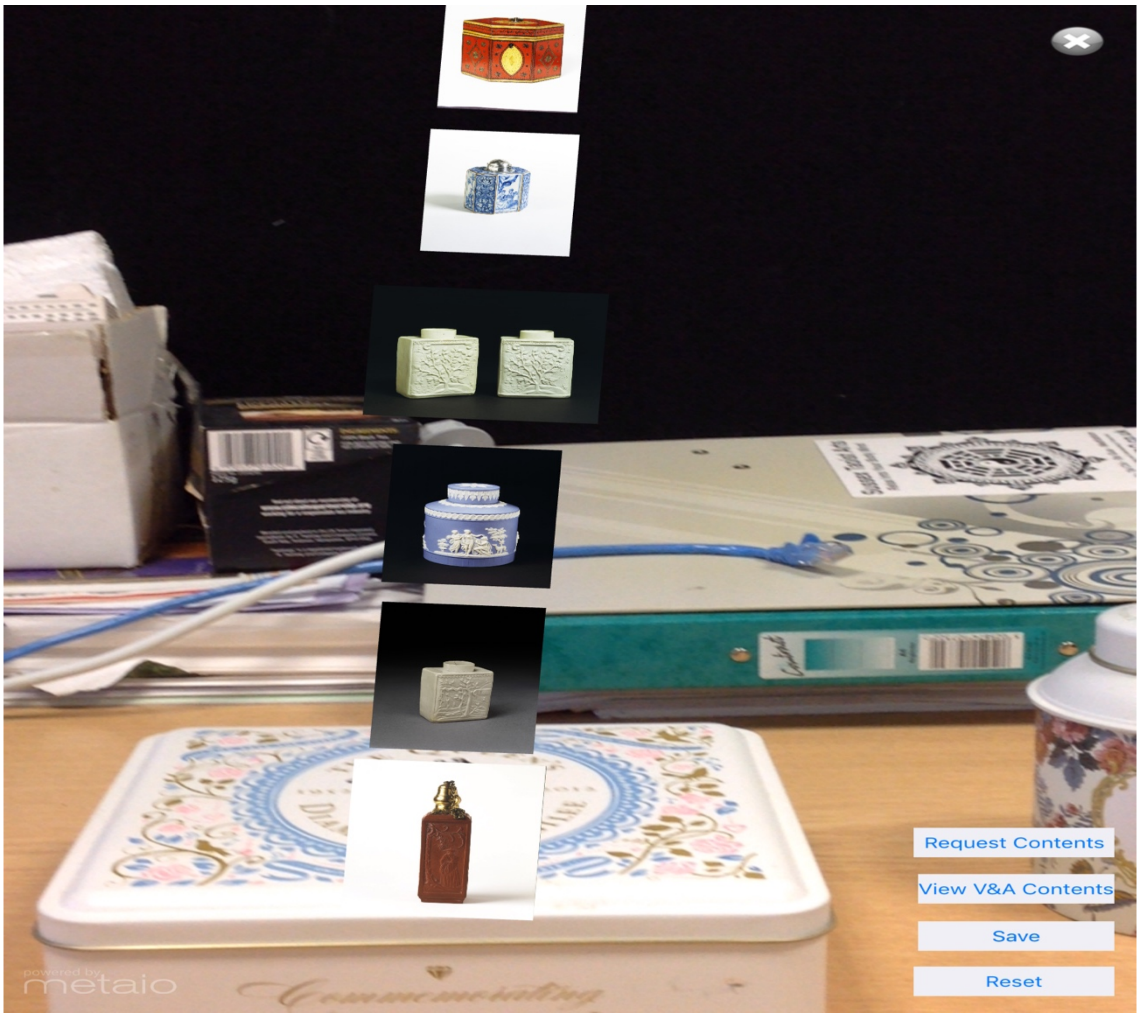


Figure 6.9 The associated content of a targeted sample object

6.2 Home-based Learning Scenario

A home-based learning scenario is another scenario that supports MLAs, one that can be done at any time and anywhere outside the museum. Home-based learning requires images of preferred objects instead of physical objects for the system to perform marker-less tracking and augment associated content requested from open providers. In this home-based learning scenario, the system accesses and utilizes content from the Sierra Leone Heritage API [98]. Figure 6.10 shows the AR view of a home-based learning scenario.

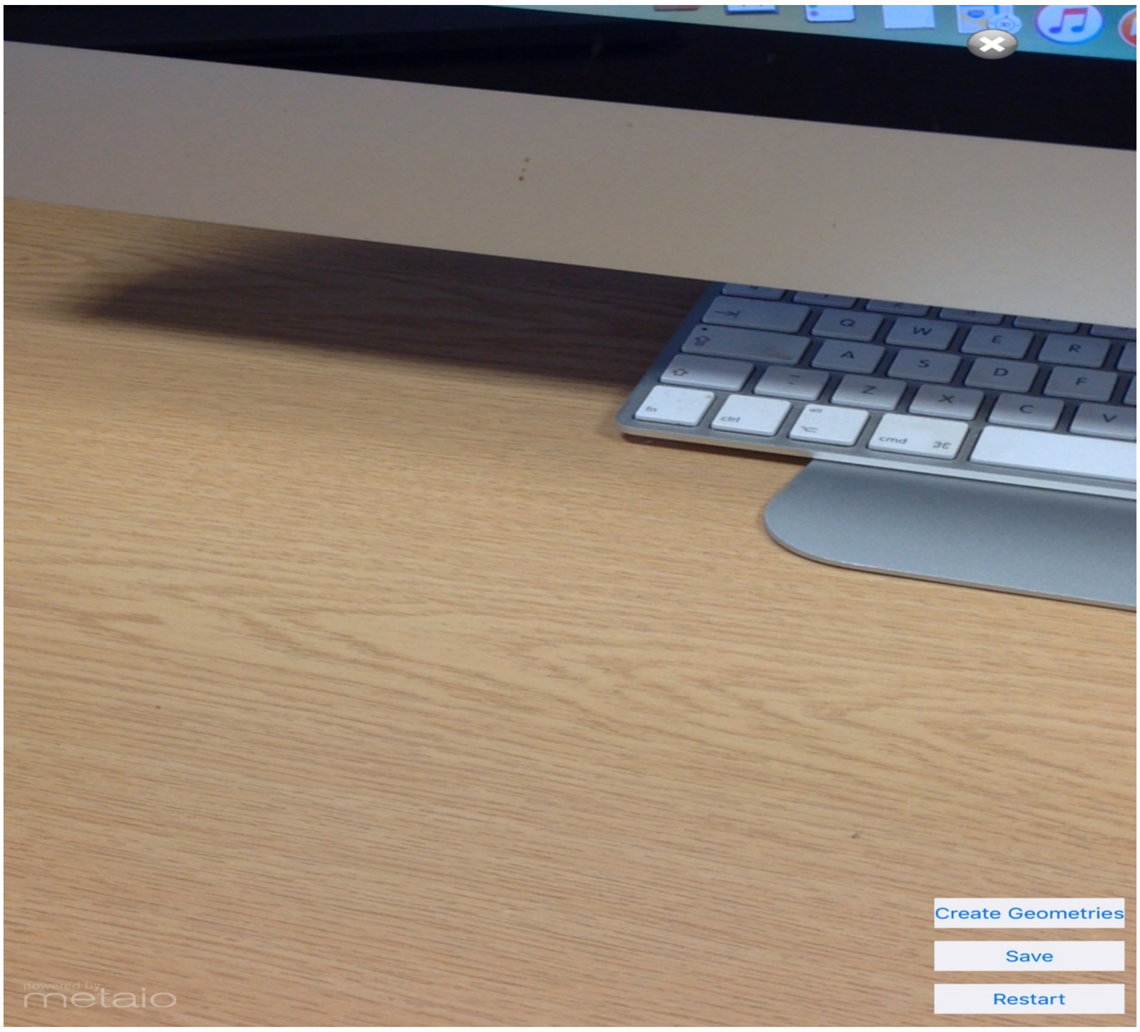
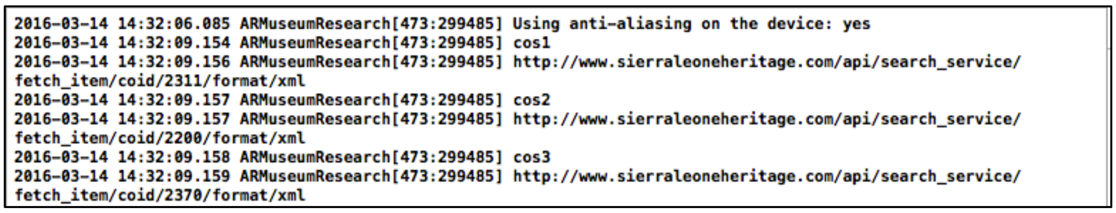


Figure 6.10 The AR view of home-based museum learning

When the system start loading the AR view, it also starts extracting data from a content configuration file in XML format composed of URLs or service APIs of participating content providers. Figure 6.11 presents extracted data from an XML file processed by XMLParser.



```
2016-03-14 14:32:06.085 ARMuseumResearch[473:299485] Using anti-aliasing on the device: yes
2016-03-14 14:32:09.154 ARMuseumResearch[473:299485] cos1
2016-03-14 14:32:09.156 ARMuseumResearch[473:299485] http://www.sierraleoneheritage.com/api/search_service/
fetch_item/coid/2311/format/xml
2016-03-14 14:32:09.157 ARMuseumResearch[473:299485] cos2
2016-03-14 14:32:09.157 ARMuseumResearch[473:299485] http://www.sierraleoneheritage.com/api/search_service/
fetch_item/coid/2200/format/xml
2016-03-14 14:32:09.158 ARMuseumResearch[473:299485] cos3
2016-03-14 14:32:09.159 ARMuseumResearch[473:299485] http://www.sierraleoneheritage.com/api/search_service/
fetch_item/coid/2370/format/xml
```

Figure 6.11 The URLs of participating content providers

6.2.1 Object tracking

Mobile users can now track a trigger image of a physical object, the system recognizes the tracked image and sends a request for associated content through the web service framework to the Sierra Leone Heritage API. Figure 6.12 presents the trigger image of a targeted physical object that the system tracks, automatically sending a web service request for associated content to the service provider.

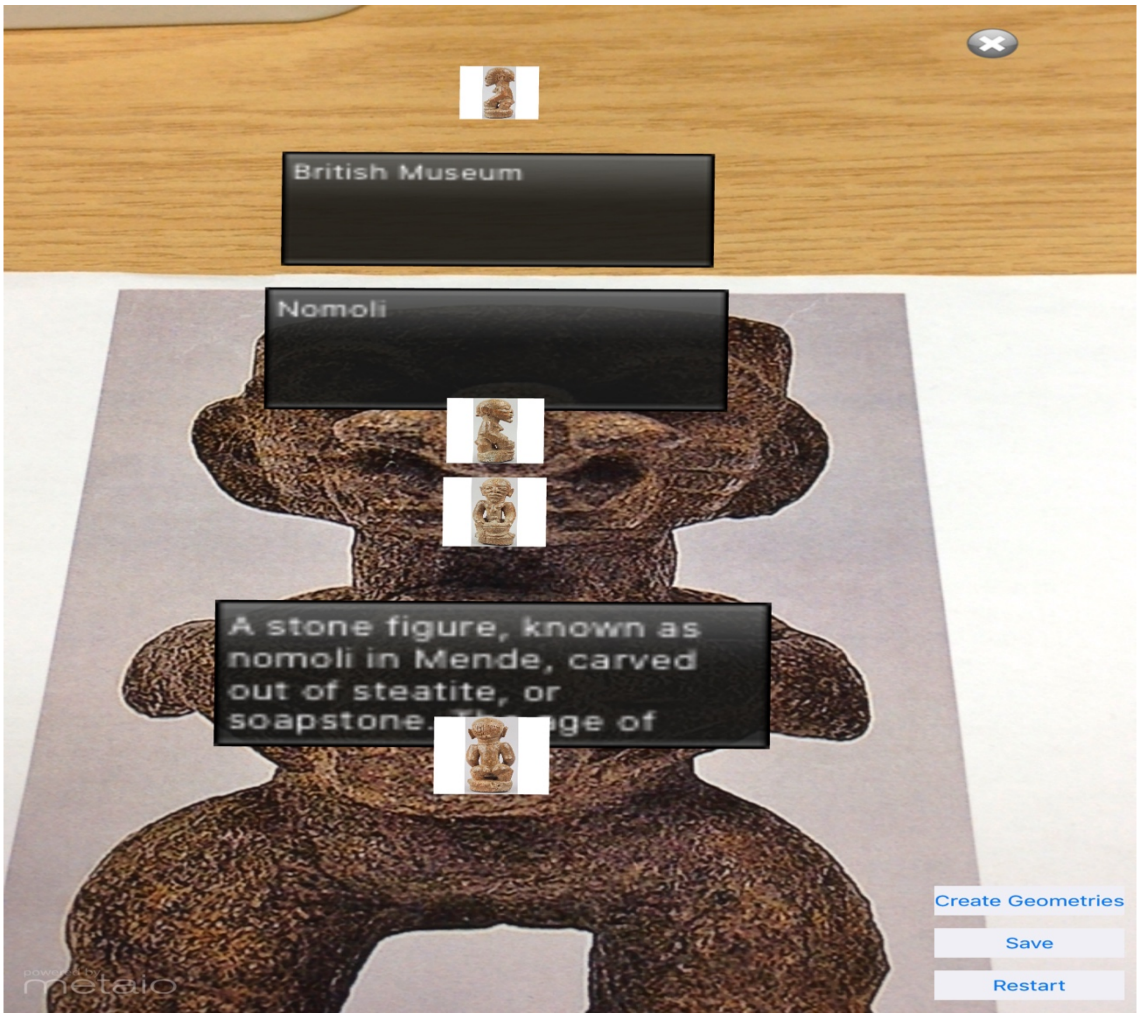

British Museum
Nomoli
A stone figure, known as nomoli in Mende, carved out of steatite, or soapstone. ...ge of
Create Geometries
Save
Restart
metaio

Figure 6.12 The related content of a trigger image representing a targeted cultural object

Figure 6.13 shows the related content of a trigger image composed of images, text and a 3D model of the physical cultural object that is added into the AR View in order to enhance the AR environment and increase more understanding about the targeted object.

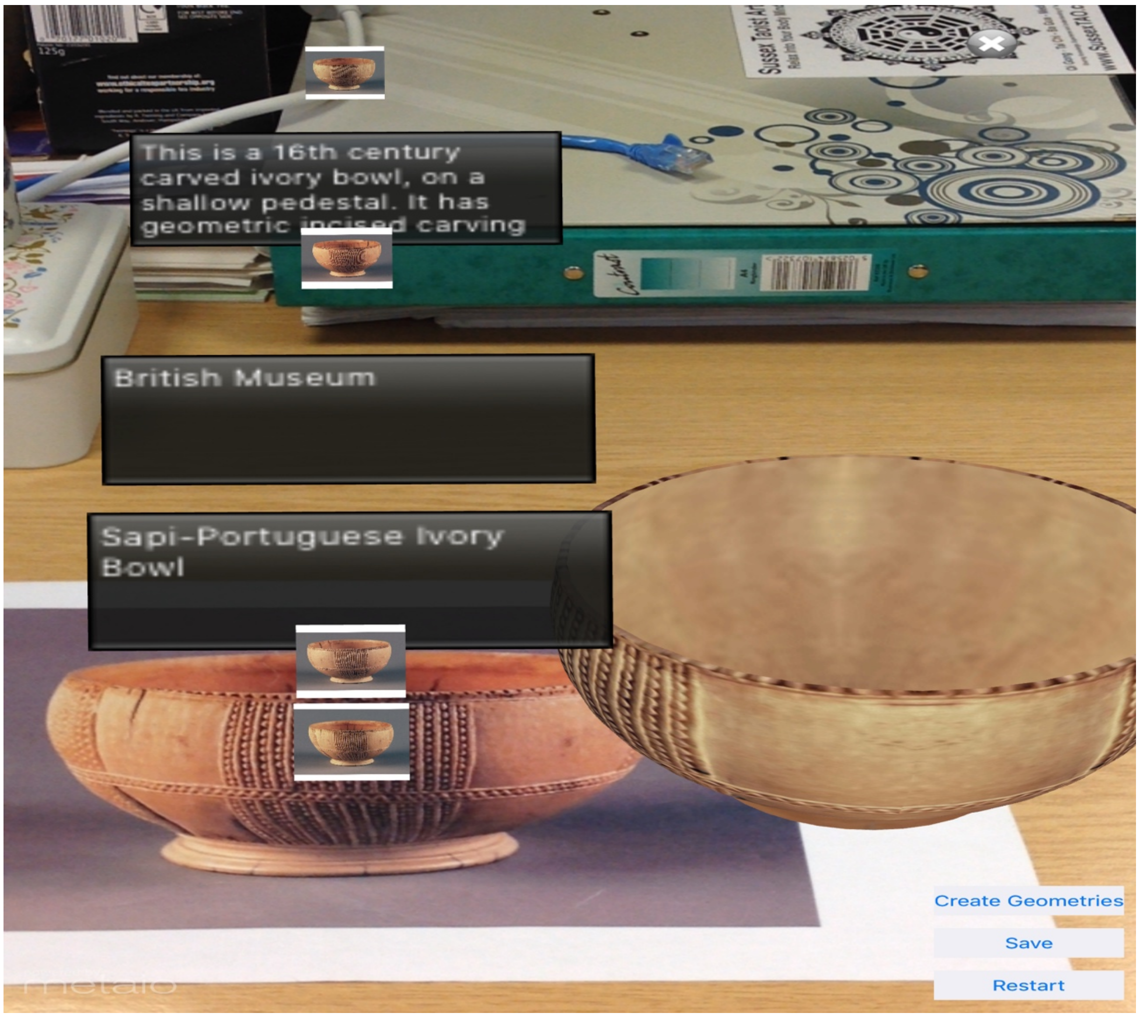
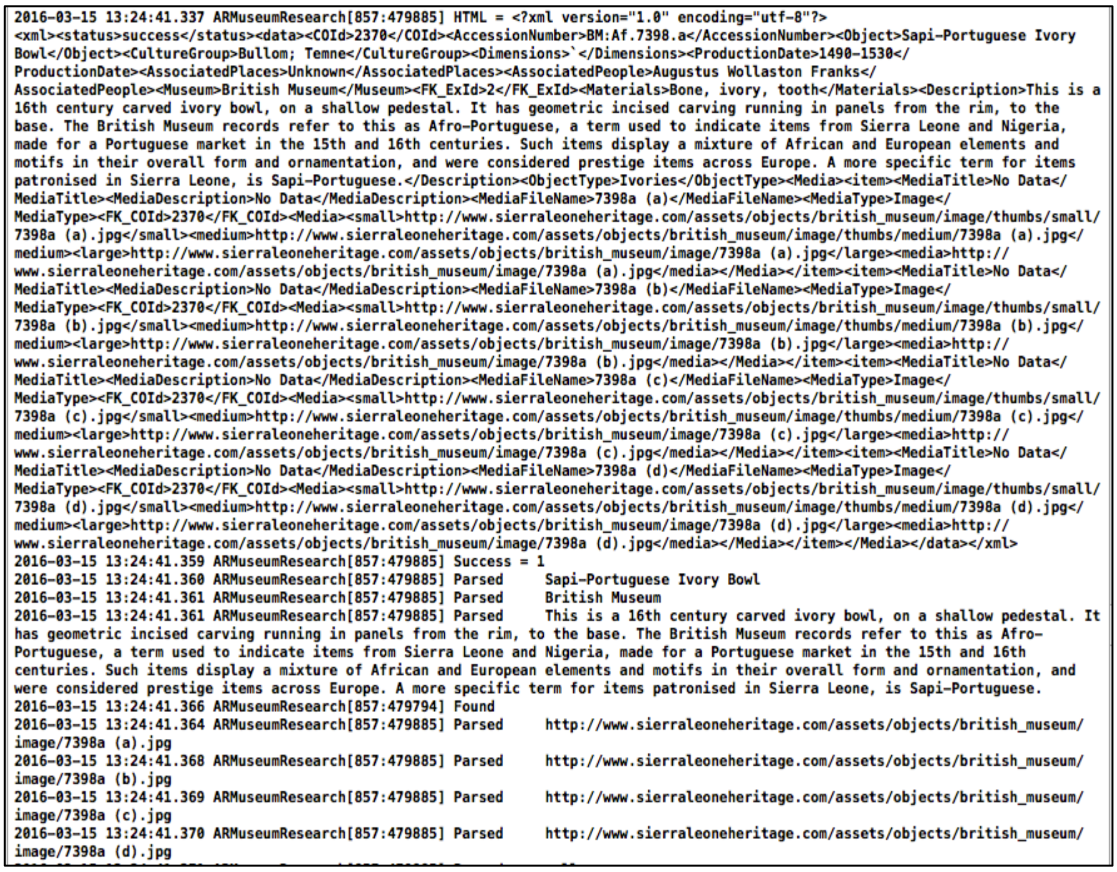

This is a 16th century carved ivory bowl, on a shallow pedestal. It has geometric incised carving
British Museum
Sapi-Portuguese Ivory Bowl
Create Geometries
Save
Restart

Figure 6.13 The tracked trigger image

After sending a request, the system receives a web service response in either XML or JSON format. The response is read through in order to extract required content that is visualized on the screen. Figure 6.14 shows the web service response in XML format and received content read through by XMLParser.



```
2016-03-15 13:24:41.337 ARMuseumResearch[857:479885] HTML = <?xml version="1.0" encoding="utf-8"?>
<xml><status>success</status><data><COId>2370</COId><AccessionNumber>BM:Af.7398.a</AccessionNumber><Object>Sapi-Portuguese Ivory
Bowl</Object><CultureGroup>Bullom; Temne</CultureGroup><Dimensions>`</Dimensions><ProductionDate>1490-1530</
ProductionDate><AssociatedPlaces>Unknown</AssociatedPlaces><AssociatedPeople>Augustus Wollaston Franks</
AssociatedPeople><Museum>British Museum</Museum><FK_ExId>2</FK_ExId><Materials>Bone, ivory, tooth</Materials><Description>This is a
16th century carved ivory bowl, on a shallow pedestal. It has geometric incised carving running in panels from the rim, to the
base. The British Museum records refer to this as Afro-Portuguese, a term used to indicate items from Sierra Leone and Nigeria,
made for a Portuguese market in the 15th and 16th centuries. Such items display a mixture of African and European elements and
motifs in their overall form and ornamentation, and were considered prestige items across Europe. A more specific term for items
patronised in Sierra Leone, is Sapi-Portuguese.</Description><ObjectType>Ivories</ObjectType><Media><item><MediaTitle>No Data</
MediaTitle><MediaDescription>No Data</MediaDescription><MediaFileName>7398a (a)</MediaFileName><MediaType>Image</
MediaType><FK_COId>2370</FK_COId><Media><small>http://www.sierraleoneheritage.com/assets/objects/british_museum/image/thumbs/small/
7398a (a).jpg</small><medium>http://www.sierraleoneheritage.com/assets/objects/british_museum/image/thumbs/medium/7398a (a).jpg</
medium><large>http://www.sierraleoneheritage.com/assets/objects/british_museum/image/7398a (a).jpg</large><media>http://
www.sierraleoneheritage.com/assets/objects/british_museum/image/7398a (a).jpg</media></Media></item><item><MediaTitle>No Data</
MediaTitle><MediaDescription>No Data</MediaDescription><MediaFileName>7398a (b)</MediaFileName><MediaType>Image</
MediaType><FK_COId>2370</FK_COId><Media><small>http://www.sierraleoneheritage.com/assets/objects/british_museum/image/thumbs/small/
7398a (b).jpg</small><medium>http://www.sierraleoneheritage.com/assets/objects/british_museum/image/thumbs/medium/7398a (b).jpg</
medium><large>http://www.sierraleoneheritage.com/assets/objects/british_museum/image/7398a (b).jpg</large><media>http://
www.sierraleoneheritage.com/assets/objects/british_museum/image/7398a (b).jpg</media></Media></item><item><MediaTitle>No Data</
MediaTitle><MediaDescription>No Data</MediaDescription><MediaFileName>7398a (c)</MediaFileName><MediaType>Image</
MediaType><FK_COId>2370</FK_COId><Media><small>http://www.sierraleoneheritage.com/assets/objects/british_museum/image/thumbs/small/
7398a (c).jpg</small><medium>http://www.sierraleoneheritage.com/assets/objects/british_museum/image/thumbs/medium/7398a (c).jpg</
medium><large>http://www.sierraleoneheritage.com/assets/objects/british_museum/image/7398a (c).jpg</large><media>http://
www.sierraleoneheritage.com/assets/objects/british_museum/image/7398a (c).jpg</media></Media></item><item><MediaTitle>No Data</
MediaTitle><MediaDescription>No Data</MediaDescription><MediaFileName>7398a (d)</MediaFileName><MediaType>Image</
MediaType><FK_COId>2370</FK_COId><Media><small>http://www.sierraleoneheritage.com/assets/objects/british_museum/image/thumbs/small/
7398a (d).jpg</small><medium>http://www.sierraleoneheritage.com/assets/objects/british_museum/image/thumbs/medium/7398a (d).jpg</
medium><large>http://www.sierraleoneheritage.com/assets/objects/british_museum/image/7398a (d).jpg</large><media>http://
www.sierraleoneheritage.com/assets/objects/british_museum/image/7398a (d).jpg</media></Media></item></Media></data></xml>
2016-03-15 13:24:41.359 ARMuseumResearch[857:479885] Success = 1
2016-03-15 13:24:41.360 ARMuseumResearch[857:479885] Parsed     Sapi-Portuguese Ivory Bowl
2016-03-15 13:24:41.361 ARMuseumResearch[857:479885] Parsed     British Museum
2016-03-15 13:24:41.361 ARMuseumResearch[857:479885] Parsed     This is a 16th century carved ivory bowl, on a shallow pedestal. It
has geometric incised carving running in panels from the rim, to the base. The British Museum records refer to this as Afro-
Portuguese, a term used to indicate items from Sierra Leone and Nigeria, made for a Portuguese market in the 15th and 16th
centuries. Such items display a mixture of African and European elements and motifs in their overall form and ornamentation, and
were considered prestige items across Europe. A more specific term for items patronised in Sierra Leone, is Sapi-Portuguese.
2016-03-15 13:24:41.366 ARMuseumResearch[857:479794] Found
2016-03-15 13:24:41.364 ARMuseumResearch[857:479885] Parsed     http://www.sierraleoneheritage.com/assets/objects/british_museum/
image/7398a (a).jpg
2016-03-15 13:24:41.368 ARMuseumResearch[857:479885] Parsed     http://www.sierraleoneheritage.com/assets/objects/british_museum/
image/7398a (b).jpg
2016-03-15 13:24:41.369 ARMuseumResearch[857:479885] Parsed     http://www.sierraleoneheritage.com/assets/objects/british_museum/
image/7398a (c).jpg
2016-03-15 13:24:41.370 ARMuseumResearch[857:479885] Parsed     http://www.sierraleoneheritage.com/assets/objects/british_museum/
image/7398a (d).jpg
```

Figure 6.14 The web service response and acquired content

The acquired content is stored in the library and the geometries of this content are created and visualized on the AR View in order to augment a trigger image of a real cultural object.

6.3 AR Environment Personalization

Mobile users are able to select active content on the screen and then save preferred content to a database. This can be done in both museum-based and home-based learning scenarios. After selecting preferred content, mobile users can touch the Save button and the Alert View appears on the screen asking the user to fill in a preference name. Touching the Save button saves selected content to the database. Figure 6.15 shows the Alert View after selecting content in a museum-based learning scenario.

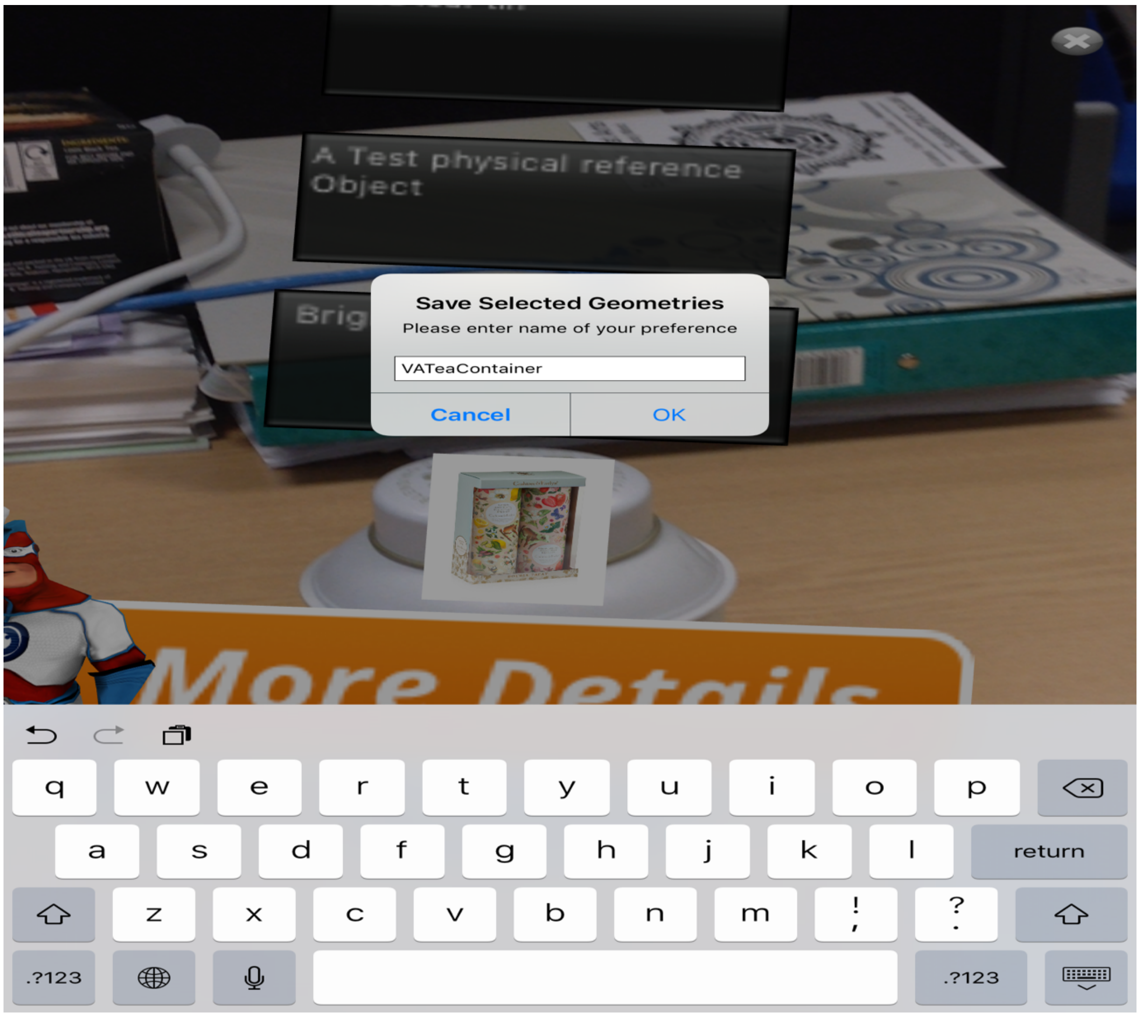


Figure 6.15 Selected content and the Alert View asking for a preference name

Media content on the AR View in home-based learning scenarios can be selected and saved just as in the museum-based learning scenario. Figure 6.16 presents the home-based learning AR environment and the Alert View asking for a preference name for the personal AR environment.

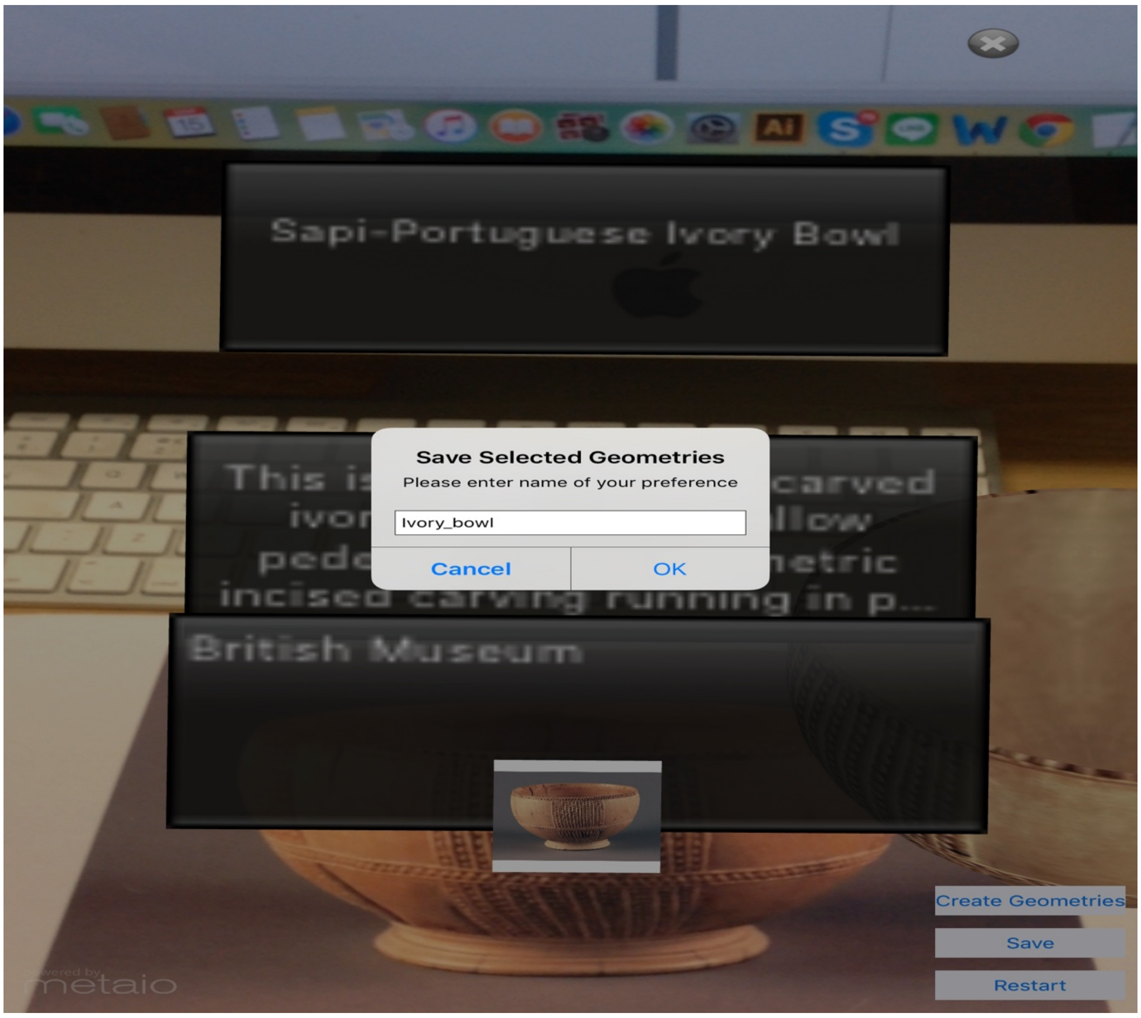
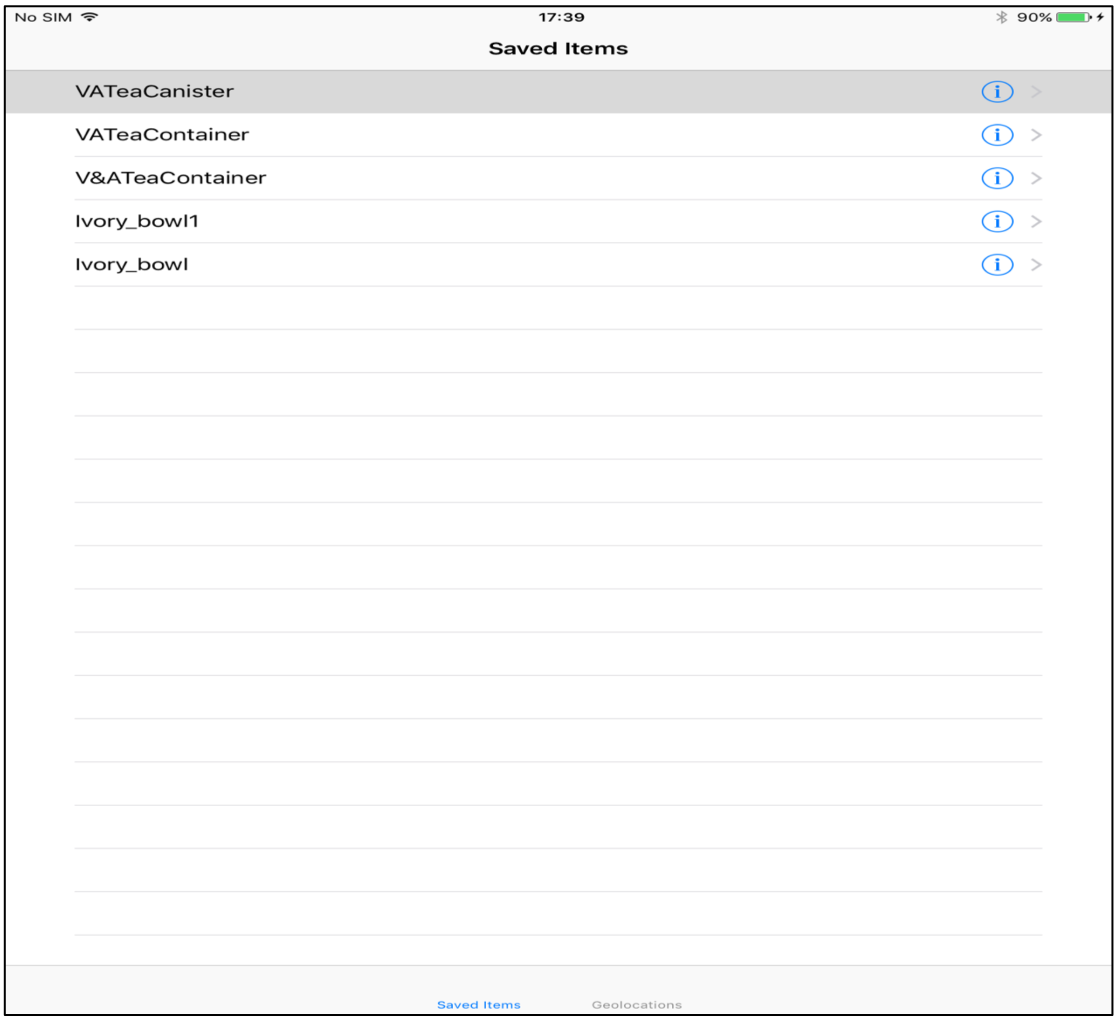

Sapi-Portuguese Ivory Bowl
This is a carved ivory bowl ... pedestal ... geometric incised carving running in p...
British Museum
Save Selected Geometries
Please enter name of your preference
Ivory_bowl
Cancel
OK
Create Geometries
Save
Restart
powered by metaio

Figure 6.16 A screen shows selected content and preference name

6.4 AR Browser

The AR browser is used to present saved AR environments stored in the database by listing preference names and museum names of preferred objects. Mobile users are able to choose a preference name in order to reveal its AR environment or content by tracking a trigger image. Figure 6.17 shows the Table View listing preference names of saved personal AR environments.



No SIM 17:39 90%

Saved Items

VATeaCanister

VATeaContainer

V&ATeaContainer

Ivory_bowl1

Ivory_bowl

Figure 6.17 User's preference names

After selecting a preference name, the system loads the AR view and creates geometries of the saved content in advance. When a trigger image is tracked, the system reveals the associated content as a personal AR environment on the real scene. Figure 6.18 illustrates the saved content in the database of a preferred object that is visualized on the screen. This content has been saved in a home-based learning scenario.

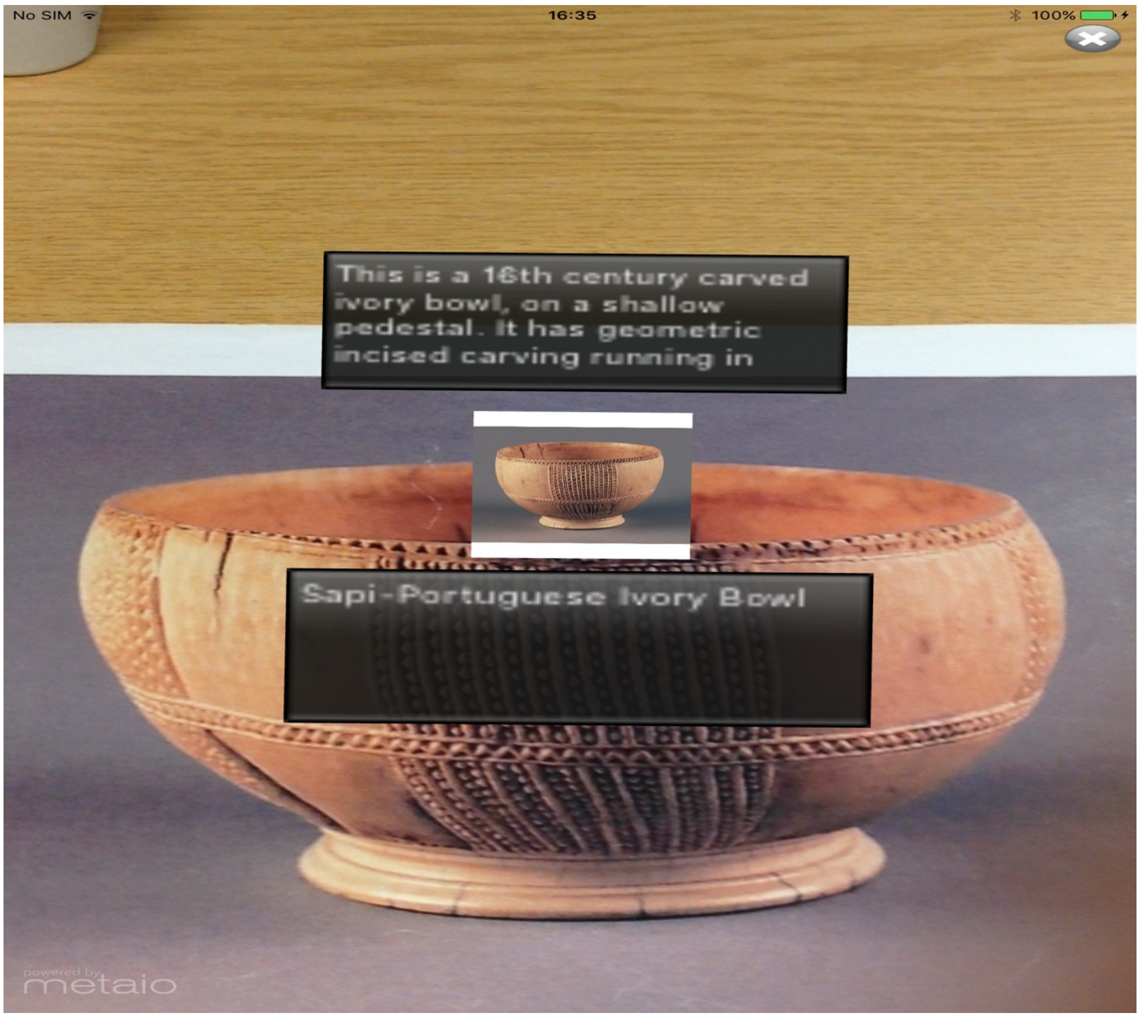

No SIM
16:35
100%
This is a 16th century carved ivory bowl, on a shallow pedestal. It has geometric incised carving running in
Sapi-Portuguese Ivory Bowl
powered by metaio

Figure 6.18 A personal AR environment

Figure 6.19 shows the AR Browser presenting saved content in a personal AR environment by augmenting a trigger image of the targeted physical object. This content is obtained in a museum-based learning scenario and can be revealed in other situations such as at home or school.

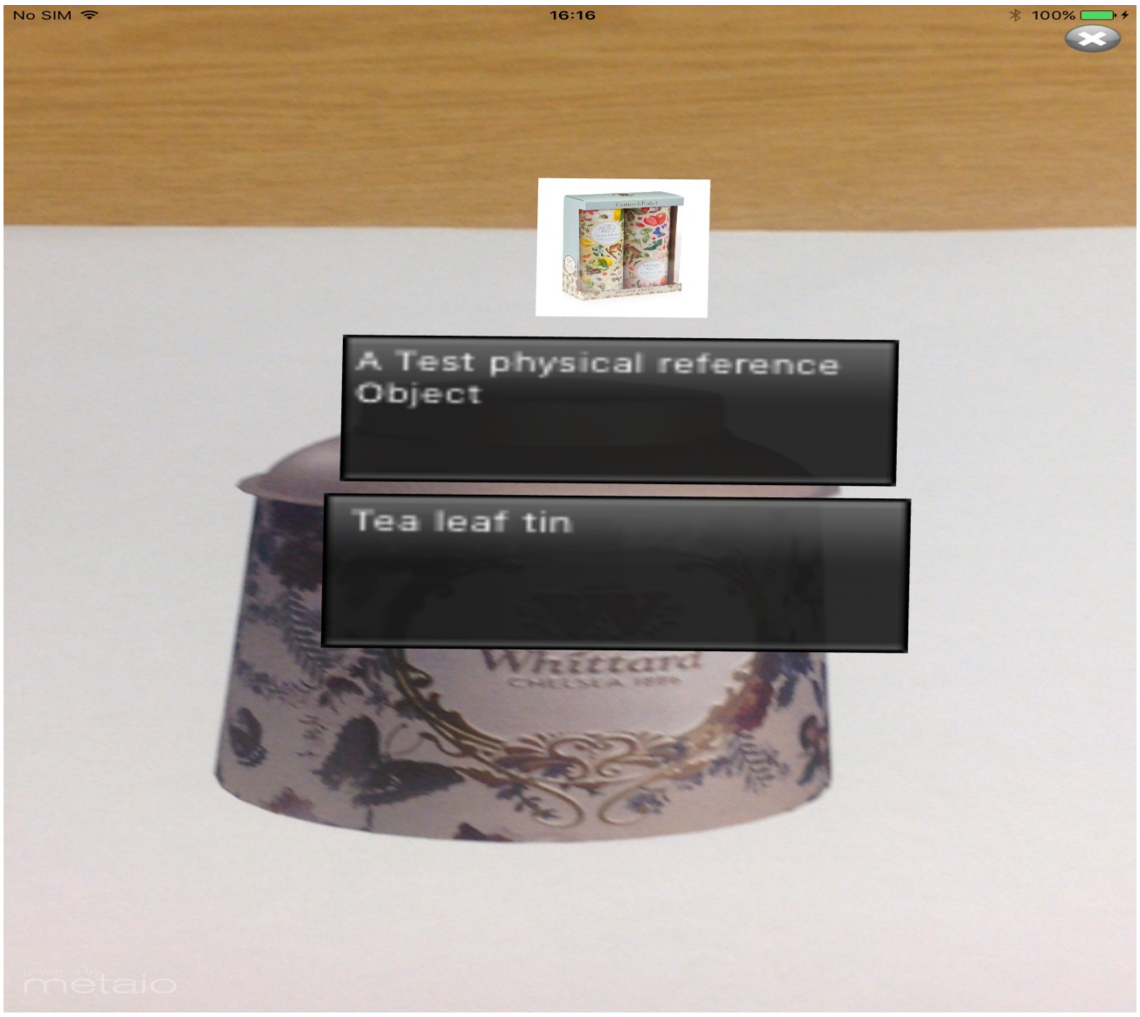
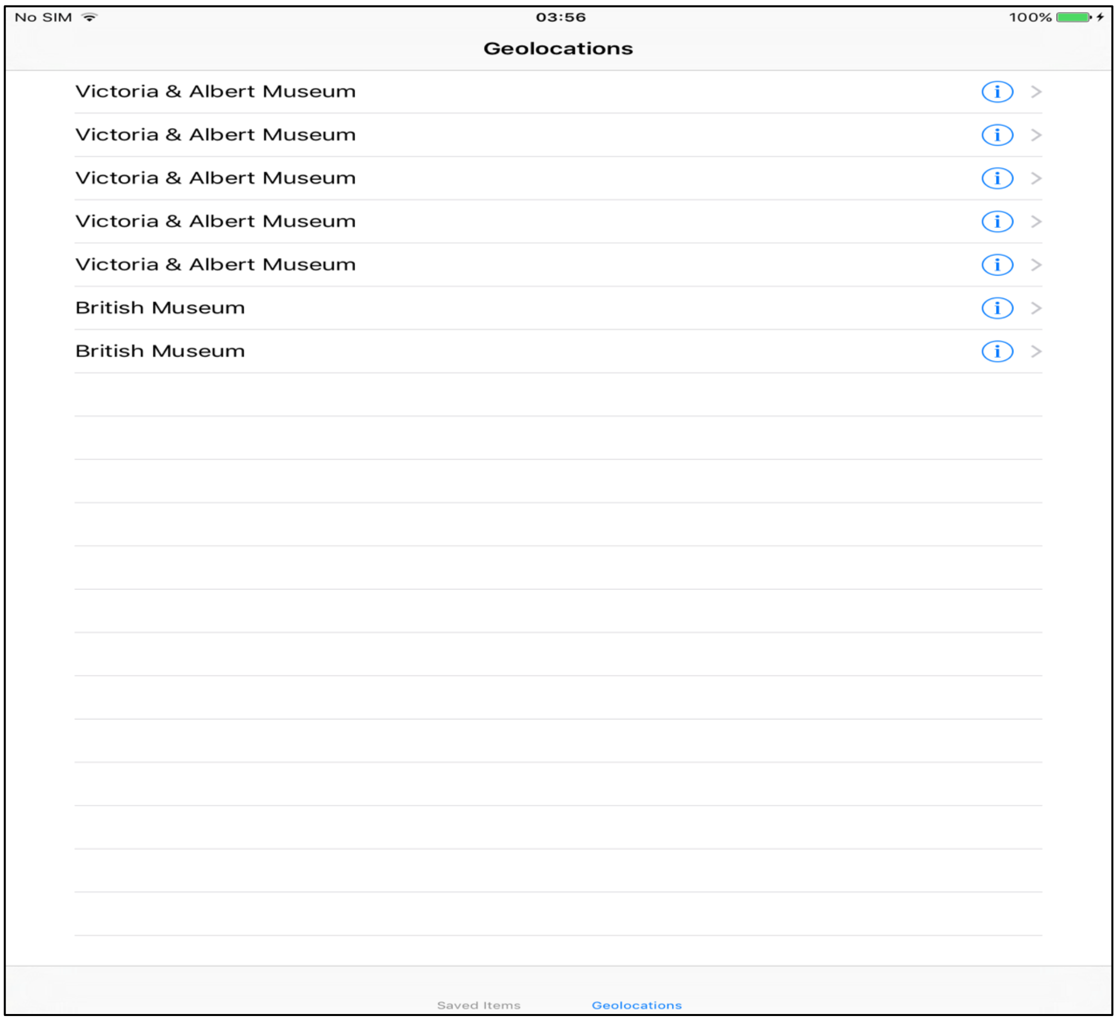


Figure 6.19 Saved content acquired from a museum-based learning scenario

The AR Browser also has a list of the names of museums that have been retrieved from the database. It allows mobile users to request a static map of the area where the museum is located. In this browser, Google Maps APIs, a third party, participates in the content acquisition process. Figure 6.20 shows the list of museums stored in the database. The list shows the name of the museum that owns the preferred objects and from whom mobile users have saved content.


No SIM 03:56 100%

Geolocations

Victoria & Albert Museum

Victoria & Albert Museum

Victoria & Albert Museum

Victoria & Albert Museum

Victoria & Albert Museum

British Museum

British Museum

Figure 6.20 The list of museums

The map of the area around the selected museum, supported by Google Maps APIs, is revealed on the screen as an image. Mobile users can also see the position of the museum or the origin of the preferred objects and other places nearby. Figure 6.21 presents the map of the selected British Museum that represents the saved content from a home-based learning scenario and the Sierra Leone Heritage API. The saved content belongs to a preferred object stored in the British Museum.

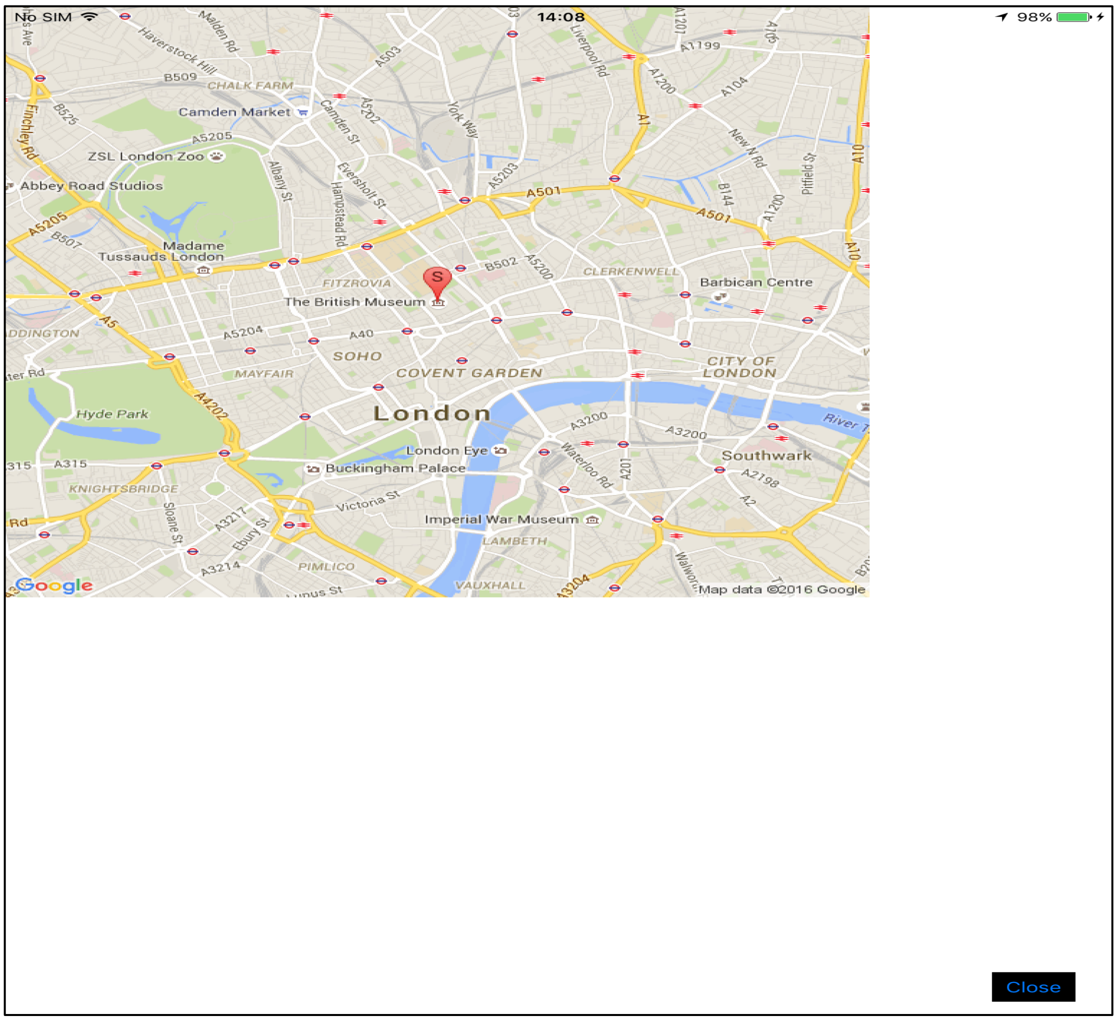


Figure 6.21 A map of the area around the British Museum

The system creates a service request in URL format that is the service API of Google Maps Web Service, which is called Google Static Maps API. The URL request is composed of the service API, parameters and user ID. A completed service request is transferred through the web service framework to the service provider. Figure 6.22 presents a service request for Google Maps Web Service for obtaining a static image presenting the position or the origin of a targeted object.

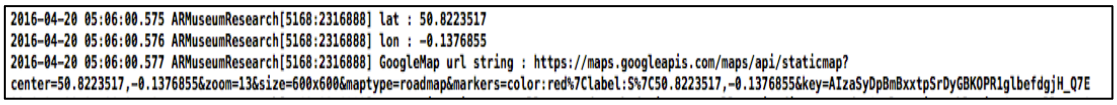

```
2016-04-20 05:06:00.575 ARMuseumResearch[5168:2316888] lat : 50.8223517
2016-04-20 05:06:00.576 ARMuseumResearch[5168:2316888] lon : -0.1376855
2016-04-20 05:06:00.577 ARMuseumResearch[5168:2316888] GoogleMap url string : https://maps.googleapis.com/maps/api/staticmap?
center=50.8223517,-0.1376855&zoom=13&size=600x600&maptype=roadmap&markers=color:red%7Clabel:S%7C50.8223517,-0.1376855&key=AIzaSyDpBmBxxtpSrDyGBKOPR1glbefdgjH_Q7E
```

Figure 6.22 A service request in URL format of Google Static Maps API

Figure 6.23 presents a map relating to a targeted object in the Victoria and Albert Museum that provides content in a museum-based learning scenario. The map shows the location of the origin of the targeted object that is stored in the Victoria and Albert Museum.

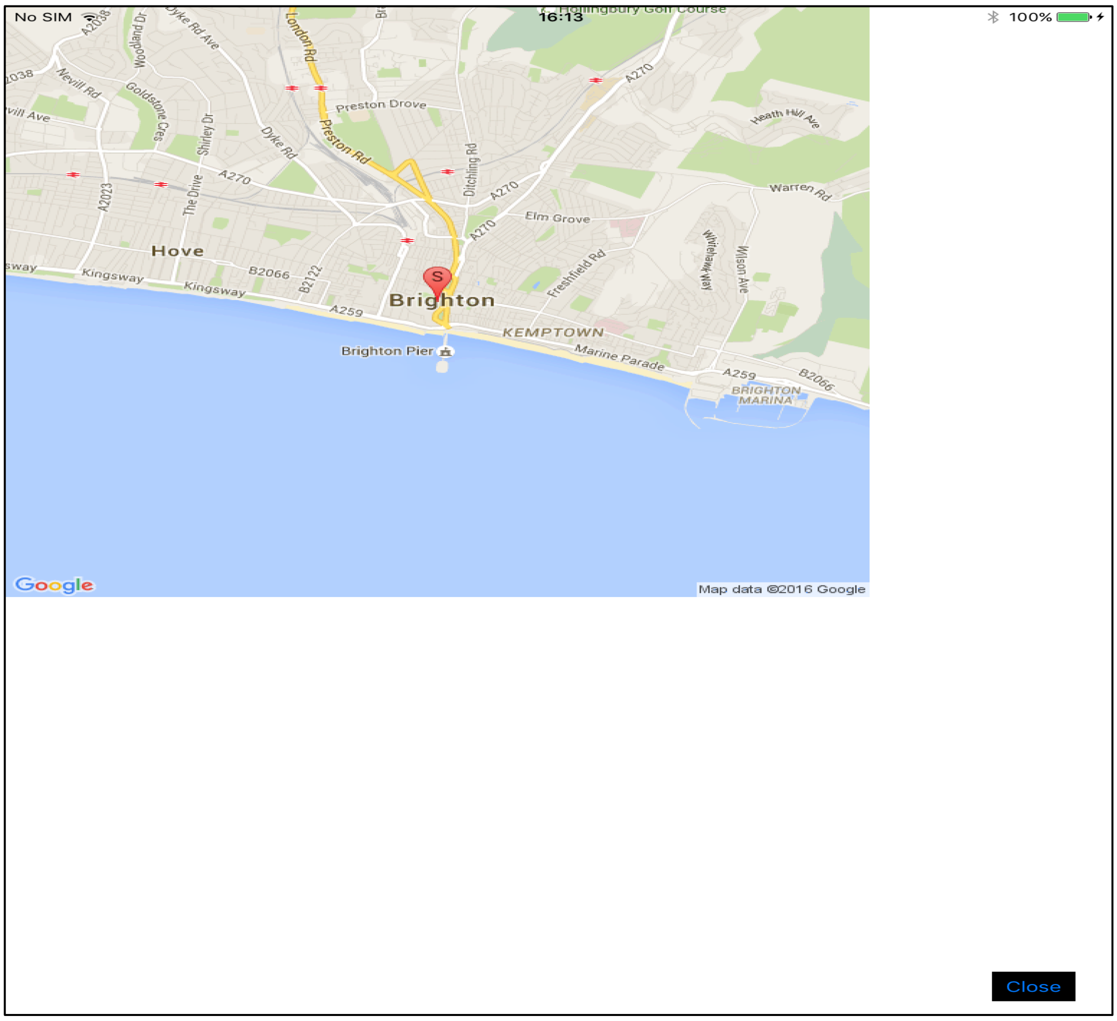


Figure 6.23 The map presenting the source of a preferred object stored in the Victoria and Albert Museum

6.5 Photogrammetry Service

An example content acquisition function is a photogrammetry service request that can be performed in a museum-based scenario. The SOMARA-based mobile AR application has a function to take photographs of a preferred object and send a request to a photogrammetry service provider. Figure 6.24 presents the Camera View that automatically takes photographs of a targeted object. Mobile users are able to move a mobile device around the object in order to capture photographs.

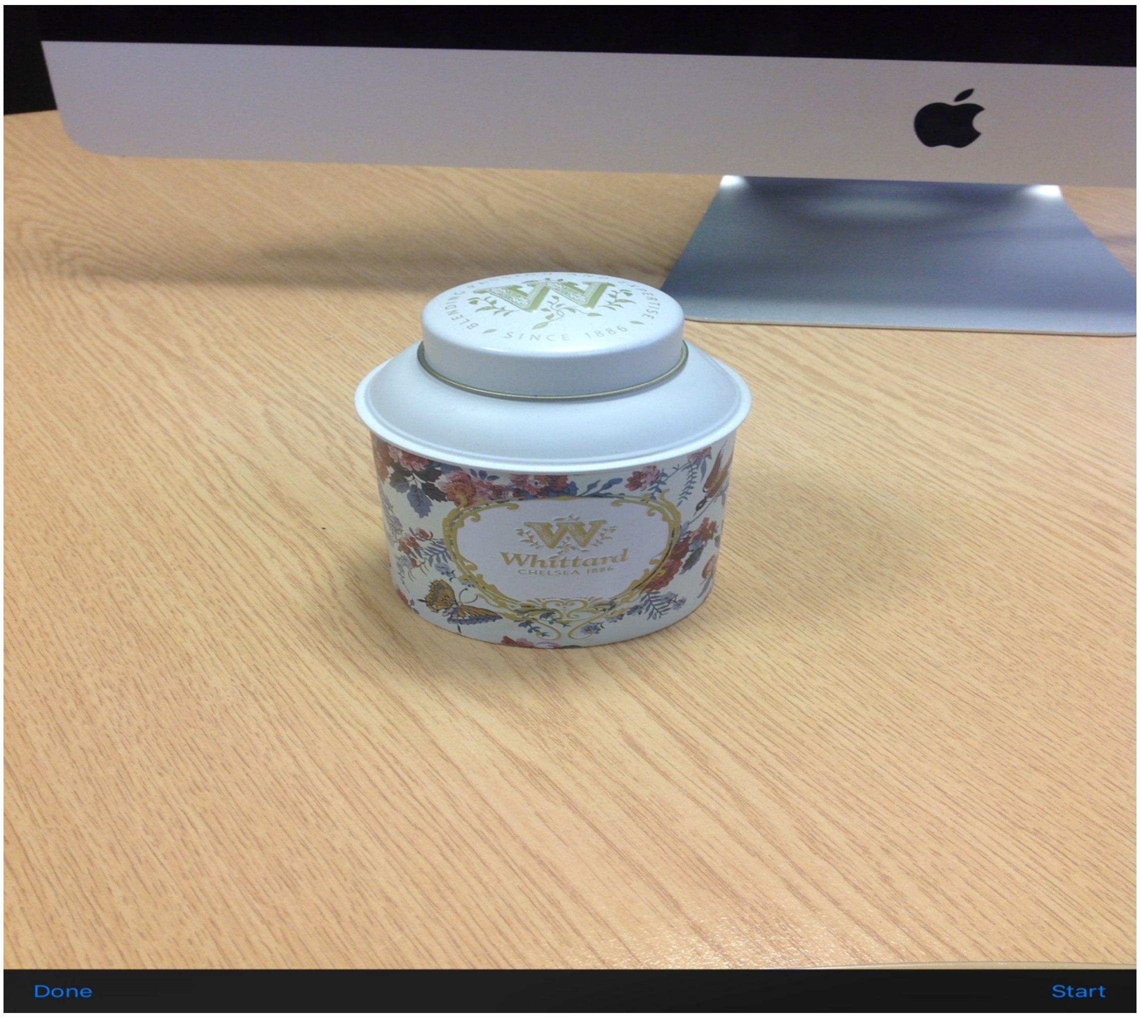


Figure 6.24 The Camera View

After the user touches the Start button, the system starts taking photographs of the sample object. The camera has to be moved around the object so that the system can take photographs from every angle. Figure 6.25 shows photographs taken in the Collection View. The Collection View appears after taking photographs by touching the Done button. Touching the Request Photogrammetry button sends a request through a web service framework to an open photogrammetry service.

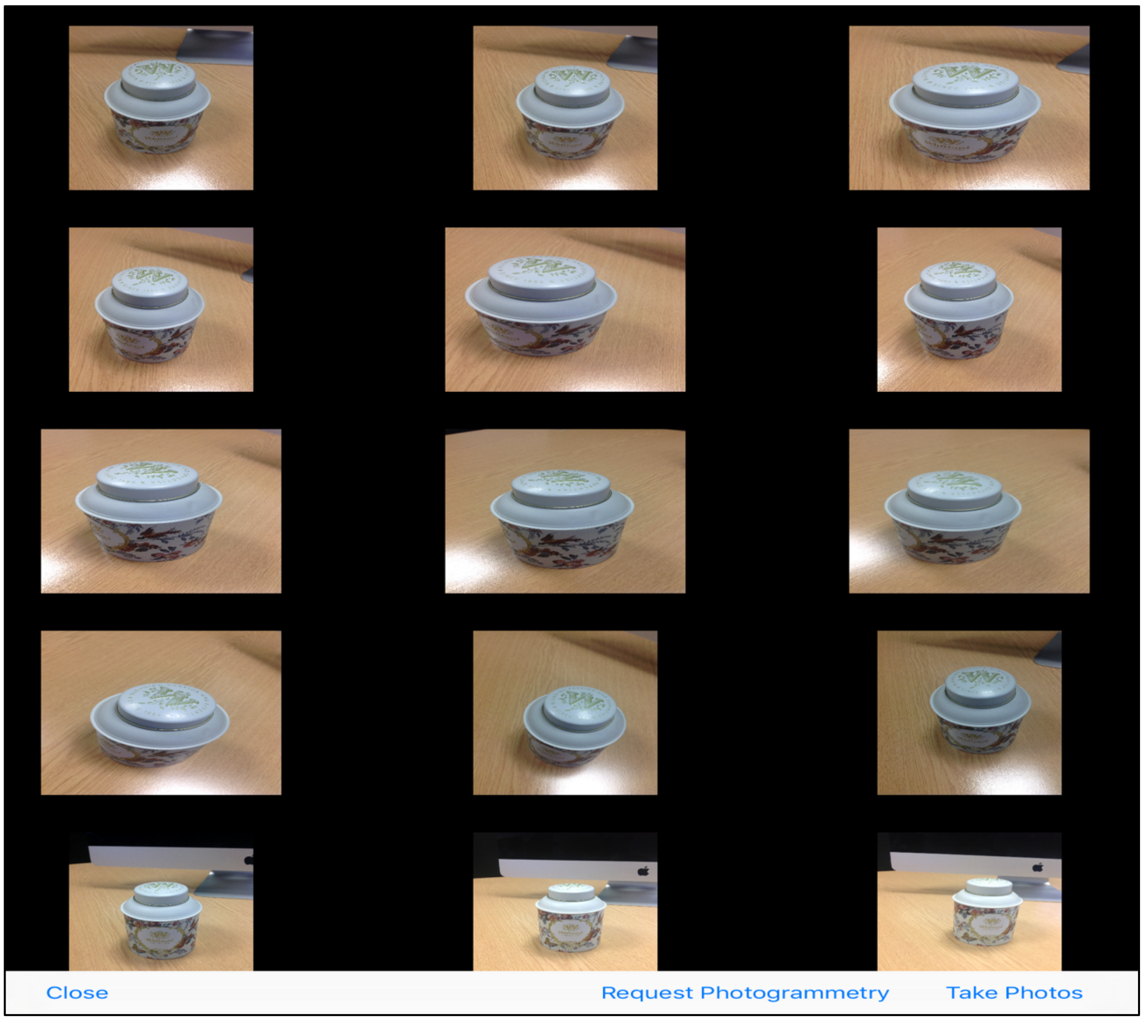


Figure 6.25 The Collection View displaying photographs taken of a sample object

6.6 Summary

This chapter illustrates a working example SOMARA-based mobile AR application running on a mobile device, which is an iPad. The application supports both museum-based and home-based learning scenarios that require different tracking techniques. Both scenarios acquire content from participating open service providers including the Victoria and Albert Museum API and the Sierra Leone Heritage API in order to augment a targeted object that could be a physical object in museum-based learning or a trigger image representing a physical object in a home-based learning scenario.

Chapter VII

7 Conclusion

A unique service oriented mobile augmented reality architecture (SOMARA) has been designed and developed as both a conceptual [21][23], prototype [22][46], example or reference implementation (chapter 5 and 6). The reference application is developed based on SOMARA that is an implementation of SOA on a mobile AR platform in order to create an open mobile AR architecture. The SOMARA application performs AR tasks as well as interoperability tasks in order to demonstrate content acquisition and content utilization through a web service framework. This chapter concludes the outcome of the research process that presents the accomplishment of developing SOMARA and implementing a mobile AR application on an open platform. The development also indicates its obvious limitations; it could be improved in order to create a more complete SOMARA-based mobile AR application as well as prepare an efficient open mobile AR framework for application to other scenarios.

7.1 The Research Accomplishment

The proposed SOMARA-based mobile AR application is the implementation of SOMARA in MLA scenarios as a proof of concepts and prototypes. The following are the research accomplishment that fulfills contribution to knowledge explained in chapter 1.

7.1.1 SOMARA

The designed SOMARA is composed of a mobile AR framework based on SOA that support an open mobile AR architecture. A web service framework is applied into the architecture as a back-end system to the mobile AR client in order to access open service providers such as third party or local service providers. The mobile AR client sends requests and receive responses though the web service framework. Acquired media content is visualized in AR environments.

7.1.2 The SOMARA-based mobile AR framework and application

The proposed SOMARA-based mobile AR application is able to perform AR tasks as well as interoperability tasks that require service APIs of participating service providers demonstrated through the inclusion of third party content from the Victoria and Albert Museum API, the Sierra Leone Heritage API and Google Maps API. These APIs offer media content for the application or mobile AR client applied in MLA scenarios comprising museum-based and home-based learning.

7.1.3 The Metaio AR SDK

The mobile AR client implements the Metaio AR SDK to perform AR tasks and use its physical object tracking function and content visualization to create multiple object tracking. The mobile AR client enhances AR environments by visualizing rich media contents augmented from different reference objects.

7.1.4 SOA-based content acquisition and utilization

The application supports content acquisition that is achieved through a web service framework as a back-end system. The application obtains associated content from open service providers and visualizes the acquired content on the screen. The application also encourages content utilization that allows mobile users to create personal AR environments from preferred content by selecting and saving it into the database that can be retrieved and visualized in the AR Browser. These features demonstrate the accomplishment of developing SOMARA and its application that contributes the implementation of an open mobile AR architecture that effectively apply a web service framework and existing valuable service APIs.

7.2 The Limitations and Problems

Developing a SOMARA-based mobile AR application requires many components that work together to produce an effective mobile AR client. The application development process has delivered some understanding and experiences in a mobile AR framework and open architecture that encourage content acquisition and utilization. The following are issues that have been found during the development process that can be improved to create a more sophisticated (or product oriented) SOMARA-based mobile AR application.

7.2.1 Metaio AR SDK

The SOMARA-based mobile AR application uses Metaio AR SDK to perform AR tasks including tracking, recognition and content visualization. Developing an open mobile AR application using Metaio AR SDK has limited some features of the application. The following are issues found in the development process that affect the outcome of the application.

Object tracking

The object tracking function of Metaio AR SDK is able to perform single object tracking, meaning that the system can track only one object and visualize one single content item in an AR environment. Extending object tracking in a mobile AR application using Metaio AR SDK required the tracking configuration process to be redesigned in order to create multiple object tracking. Each single object can be augmented by visualizing multiple related content superimposed on top of the tracked object. A content configuration file is needed to specify each reference object with its content. The content configuration file is in XML format and it requires an XMLParser to read through and extract data out of the file. The created multiple object tracking can track objects sequentially, which delivers different tracking experiences from multiple object tracking in parallel. However, the Metaio AR SDK limits the number of objects that can be tracked in an AR scene to one object, therefore it is possible to add more reference objects and manage multiple content for each object sequentially. It is possible to create a mobile AR application that performs multiple object tracking in parallel, but this is beyond the scope of this thesis with the mataio SDK. However, one mobile SDK that can be effectively implemented in this platform is Vuforia.

Content visualization

The obtained media content such as text and images are visualized on the AR view as billboards or geometries and there are multiple content visualized on the view once an object is tracked and augmented. The AR view on a mobile device such as an iPad has limited space to display multiple content at the same time and it is quite difficult to organize all of the related content overlaid on top of a tracked object on the AR view.

Metaio AR SDK provides a content visualization function, but it restricts the number of reference objects that are tracked in the same environment. The framework of the SDK supports single content representation that can cause difficulty when it is applied to multiple content visualization. The billboard geometries, including text and images, cannot be placed on a specific location but are placed on the current position of the camera, which is 0,0,0. The geometries are vertically aligned on the AR view by adding them into a BillboardGroup, which is a feature provided by Metaio mobile AR SDK for location-based AR.

7.2.2 Web service providers

The SOMARA-based mobile AR application uses third party service APIs including Google Maps API, the Victoria and Albert Museum API and specifically designed APIs including the Sierra Leone Heritage API as examples from which the mobile AR client acquires content. These APIs provide useful media content such as text and images and can be effectively applied in mobile AR environments. In a home-based museum learning scenario, mobile users are able to view museum content through AR environments where 3D models of targeted objects can be visualized on their trigger images. Displaying 3D models of cultural objects in AR environments usefully enhances MLAs by supporting home-based learning that allows museum visitors to perform MLAs in other situations through AR environments. SOMARA and the SOMARA-based mobile AR application have a function to visualize 3D models on the AR scene. None of the web service providers used in the novel mobile AR application provide 3D models of cultural objects and the application is mainly restricted to text and images from these providers. The application still needs a web service provider that provides valuable media content, including 3D models for MLAs.

7.2.3 Photogrammetry service

One of the proposed content acquisition features supported by SOMARA is a photogrammetry service that encourages mobile users to obtain a 3D model of a preferred object in a museum-based learning scenario. Currently, there are many existing photogrammetry applications available on many platforms such as stand-alone, mobile or client-server. These applications are based on closed architectures and only provide an application interface for users on their preferred platform. The SOMARA-based mobile AR application requires an open photogrammetry service that offers an image-based reconstruction service to the mobile AR client that encourages content acquisition over a mobile or wireless network. A final 3D model of the targeted object can be visualized on the AR view in order to demonstrate a preferred cultural object in a home-base learning scenario. The mobile AR client has a function to support a photogrammetry service request but it still requires a web service-based photogrammetry service to achieve the content acquisition and utilization in SOMARA.

7.3 Future Development

The SOMARA-based mobile AR application requires AR tasks to perform object tracking, recognition and content visualization. Implementing a mobile AR SDK into the application is a quick and convenient way to acquire AR tasks. All existing mobile AR SDKs offer a free version that is relatively easy to obtain. The Metaio mobile AR SDK used in the application has necessarily limited some features required in the application development. Developing in-house AR tasks would be a best solution in this framework in order to provide more suitable and flexible AR tasks to the application and produce a proper and manageable outcome on the AR view.

The mobile AR client utilizes a Google Maps API in order to acquire a map image of a selected place. The obtained map is a static map that can only be presented on the View. There is also a Google Maps SDK that can be implemented into the mobile AR client in order to present dynamic maps and locations of preferred cultural artifacts. The Google Maps SDK can improve the functionality and usability of SOMARA and its application by offering supporting features related to the addresses or locations of preferred objects.

The aim of developing SOMARA is to create an open architecture on a mobile AR platform that is adaptable to many different scenarios. The SOMARA-based mobile AR application in this research is applied in MLA scenarios that encourage content acquisition and utilization in a mobile AR environment. SOMARA can be effectively applied in other scenarios that require an open mobile AR framework in order to augment objects and visualize media content in the real environment. One of the proposed scenarios is a shopping environment where users can experience receiving supporting information for shopping through open mobile AR environments. Content acquisition and utilization techniques encourage users to receive more valuable media content that can enhance shopping experiences. Users are able to use a mobile device to track products in the shop and see related media content of a targeted product augmented and visualized on the screen. A web service framework will enable users to acquire associated content of the product and create personal preferences that could be a collection of preferred products and details offered by the shop itself or by a third party.

Another scenario that can be improved by implementing SOMARA is Mobile Learning (ML). ML supports learning through mobile AR systems where students can learn from media content visualized on a mobile device's screen or AR environments. This encourages learning activities that can be done on mobile AR platforms and outside the classroom. The system allows students to request more valuable content in order to enrich the AR learning environment as well as to interact with media content on the real scene. These can improve students' understanding of the topics they are learning.

References

[1] "Europe in a changing world - Inclusive, innovative and reflective societies - European Commission." [Online]. Available: https://ec.europa.eu/programmes/horizon2020/en/h2020-section/europe-changing-world-inclusive-innovative-and-reflective-societies. [Accessed: 05-Apr-2016].

[2] J. H. Christensen, "Using RESTful web-services and cloud computing to create next generation mobile applications," *Proc. 24th ACM SIGPLAN Conf. companion Object oriented Program. Syst. Lang. Appl.*, pp. 627–634, 2009.

[3] W.-T. Tsai, X. Sun, and J. Balasooriya, "Service-Oriented Cloud Computing Architecture," *2010 Seventh Int. Conf. Inf. Technol. New Gener.*, pp. 684–689, 2010.

[4] Ruby Annette, "A Service Broker Model for Cloud based Render Farm Selection," vol. 96, no. 24, pp. 11–14, 2014.

[5] X. Luo, "From Augmented Reality to Augmented Computing: A Look at Cloud-Mobile Convergence," *2009 Int. Symp. Ubiquitous Virtual Real.*, no. November 2007, pp. 29–32, Jul. 2009.

[6] "ArcGIS for Server | GIS Web Server Software | Web Map Server." [Online]. Available: http://www.esri.com/software/arcgis/arcgisserver. [Accessed: 07-Jan-2016].

[7] "Autodesk 123D Catch | Generate 3d model from photos." [Online]. Available: http://www.123dapp.com/catch. [Accessed: 16-Oct-2015].

[8] "Snapdragon Processors for Power and Efficiency | Qualcomm." [Online]. Available: https://www.qualcomm.com/products/snapdragon/processors. [Accessed: 09-Dec-2015].

[9] "The World's Fastest Mobile Processors | NVIDIA Tegra|NVIDIA." [Online]. Available: http://www.nvidia.com/object/tegra.html. [Accessed: 09-Dec-2015].

[10] "Sony Xperia Z5 Mobile review: Best mobile photo & video scores to date - DxOMark." [Online]. Available: http://www.dxomark.com/Mobiles/Sony-Xperia-Z5-Mobile-review-Best-mobile-photo-video-scores-to-date. [Accessed: 09-Dec-2015].

[11] "Swift - Overview - Apple Developer." [Online]. Available: https://developer.apple.com/swift/. [Accessed: 09-Dec-2015].

[12] "Native, HTML5, or Hybrid: Understanding Your Mobile Application Development Options - developer.force.com." [Online]. Available: https://developer.salesforce.com/page/Native,_HTML5,_or_Hybrid:_Understanding_Your_Mobile_Application_Development_Options. [Accessed: 09-Dec-2015].

[13] "Next Generation Network: Are You Ready?" [Online]. Available: https://www-

ssl.intel.com/content/www/uk/en/communications/next-generation-network-architecture-infographic.html. [Accessed: 09-Dec-2015].

[14] T. D. Duplex, "[WHITE] Timing and Synchronization for LTE-TDD and LTE-Advanced Mobile Networks," pp. 1–9, 2013.

[15] T. Capin, K. Pulli, and T. Akenine-Möller, "The state of the art in mobile graphics research," *IEEE Comput. Graph. Appl.*, vol. 28, no. 4, pp. 74–84, 2008.

[16] A. Henrysson and M. Ollila, "UMAR: Ubiquitous Mobile Augmented Reality," in *Proceedings of the 3rd International Conference on Mobile and Ubiquitous Multimedia*, 2004, pp. 41–45.

[17] D. W. F. Van Krevelen and R. Poelman, "A Survey of Augmented Reality Technologies , Applications and Limitations," *Int. J.*, vol. 9, no. 2, 2010.

[18] R. Azuma, Y. Baillot, R. Behringer, S. Feiner, S. Julier, and B. MacIntyre, "Recent advances in augmented reality," *Computer Graphics and Applications, IEEE*, vol. 21, no. 6. pp. 34–47, 2001.

[19] Y. Genc, S. Riedel, F. Souvannavong, C. Akinlar, and N. Navab, "Marker-less Tracking for AR: A Learning-Based Approach," in *Proceedings of the 1st International Symposium on Mixed and Augmented Reality*, 2002, p. 295–.

[20] J. Carmigniani, B. Furht, M. Anisetti, P. Ceravolo, E. Damiani, and M. Ivkovic, "Augmented reality technologies, systems and applications," *Multimed. Tools Appl.*, vol. 51, no. 1, pp. 341–377, Dec. 2010.

[21] S. Rattanarungrot, M. White, Z. Patoli, and T. Pascu, "The Application of Augmented Reality for Reanimating Cultural Heritage," in *Virtual, Augmented and Mixed Reality. Applications of Virtual and Augmented Reality SE - 8*, vol. 8526, R. Shumaker and S. Lackey, Eds. Springer International Publishing, 2014, pp. 85–95.

[22] S. Rattanarungrot and M. White, "A service-oriented mobile augmented reality architecture for personalized museum environments," *Virtual Systems & Multimedia (VSMM), 2014 International Conference on*. pp. 277–284, 2014.

[23] S. Rattanarungrot, M. White, and P. Newbury, "A Mobile Service Oriented Multiple Object Tracking Augmented Reality Architecture for Education and Learning Experiences," in *International Conference on Mobile Learning 2014*, 2014, pp. 327–331.

[24] F. Zhou, H. B.-L. Duh, and M. Billinghurst, "Trends in augmented reality tracking, interaction and display: A review of ten years of ISMAR," in *Mixed and Augmented Reality, 2008. ISMAR 2008. 7th IEEE/ACM International Symposium on*, 2008, pp. 193–202.

[25] R. Koch, K. Koeser, B. Streckel, and J.-F. Evers-Senne, "Markerless image-based 3d tracking for real-time augmented reality applications," in *The 7th Int. Workshop on Image analysis for multimedia interactive services, Montreux, Switzerland*, 2005.

[26] Springer, *Recent trends of mobile collaborative augmented reality systems.*

Springer, 2011.

[27] G. Papagiannakis, G. Singh, and N. Magnenat-Thalmann, “A survey of mobile and wireless technologies for augmented reality systems,” *Comput. Animat. Virtual Worlds*, vol. 19, no. 1, pp. 3–22, 2008.

[28] M. Schlattmann, T. N. Nakorn, and R. Klein, “3D Interaction Techniques for 6 DOF Markerless Hand-Tracking,” *J. or Proc. WSCG to Appear*, 2009.

[29] M. White, P. Petridis, F. Liarokapis, and D. Pletinckx, “Multimodal Mixed Reality Interfaces for Visualizing Digital Heritage,” no. April, 2007.

[30] G. Alonso and F. Casati, “Service-Oriented Architecture (SOA) and Web Services,” *21st Int. Conf. Data Eng. ICDE05*, no. Icde, pp. 1147–1147, 2005.

[31] “Service-Oriented Architecture (SOA) Definition.” [Online]. Available: http://www.service-architecture.com/articles/web-services/service-oriented_architecture_soa_definition.html. [Accessed: 14-Oct-2015].

[32] D. K. Senior, A. Ibm, and G. Business, “Service Oriented Architecture (SOA) SOA Guide,” *Architecture*, vol. 24, pp. 1–5, 2006.

[33] R. Wang and X. Wang, “Applying Service-Oriented Architecture into an Augmented Reality E-business System,” *2008 IEEE Int. Conf. E-bus. Eng.*, pp. 621–626, 2008.

[34] P. Selonen, P. Belimpasakis, Y. You, T. Pylvänäinen, and S. Uusitalo, “Mixed reality web service platform,” *Multimed. Syst.*, vol. 18, no. 3, pp. 215–230, 2012.

[35] R. Wojciechowski, K. Walczak, M. White, and W. Cellary, “Building Virtual and Augmented Reality Museum Exhibitions,” in *Proceedings of the Ninth International Conference on 3D Web Technology*, 2004, pp. 135–144.

[36] M. Hirose and T. Tanikawa, “Overview of the Digital Museum Project,” in *Proceedings of the 9th ACM SIGGRAPH Conference on Virtual-Reality Continuum and Its Applications in Industry*, 2010, pp. 11–16.

[37] A. Karoulis, S. Sylaiou, and M. White, “Usability Evaluation of a Virtual Museum Interface,” *Informatica*, vol. 17, no. 3, pp. 363–380, 2006.

[38] Y. Kolstee and W. Van Eck, “The augmented Van Gogh’s: Augmented reality experiences for museum visitors,” in *2011 IEEE International Symposium on Mixed and Augmented Reality - Arts, Media, and Humanities, ISMAR-AMH 2011*, 2011, pp. 49–52.

[39] R. Szeliski, “Computer Vision : Algorithms and Applications,” *Computer (Long. Beach. Calif).*, vol. 5, no. 3, p. 832, 2010.

[40] P. Belimpasakis, P. Selonen, and Y. You, “Bringing user-generated content from Internet services to mobile augmented reality clients,” *2010 Cloud-Mobile Converg. Virtual Real. Work. (CMCVR 2010) Proc.*, pp. 14–17, Mar. 2010.

[41] “AURASMA.” [Online]. Available: https://www.aurasma.com/. [Accessed: 15-Oct-2015].

[42] "Augmented Reality | Interactive Print | Layar." [Online]. Available: https://www.layar.com/. [Accessed: 15-Oct-2015].

[43] "ReCap Reality Capture for 3D Documentation | Autodesk ReCap," 2016. [Online]. Available: https://recap.autodesk.com/. [Accessed: 12-Nov-2015].

[44] P. Henry, M. Krainin, E. Herbst, X. Ren, and D. Fox, "RGB-D Mapping : Using Depth Cameras for Dense 3D Modeling of Indoor Environments," *Int. J. Rob. Res.*, vol. 31, no. 5, pp. 647–663, 2012.

[45] "Getting Started with the Metaio SDK." [Online]. Available: https://my.metaio.com/dev/sdk/index.html. [Accessed: 03-Nov-2015].

[46] S. Rattanarungrot, M. White, and B. Jackson, "The Application of Service Orientation on a Mobile AR Platform - A Museum Scenario," in *International Congress on Digital Heritage - Theme 2 - Computer Graphics And Interaction*, 2015.

[47] M. Chouiten, J.-Y. Didier, and M. Mallem, "A Framework for Service Based Composite Augmented Reality Applications," *2013 Int. Symp. Ubiquitous Virtual Real.*, pp. 19–22, Jul. 2013.

[48] P. Belimpasakis, P. Selonen, and Y. You, "A web service platform for building interoperable augmented reality solutions," in *International AR standards Workshop Oct 11-12 2010*, 2010, pp. 1–5.

[49] P. Milgram, H. Takemura, A. Utsumi, and F. Kishino, "Augmented reality: a class of displays on the reality-virtuality continuum," *Proc. SPIE*, vol. 2351. pp. 282–292, 1995.

[50] G. Sziebig, "Achieving total immersion : Technology trends behind Augmented Reality - A survey," *Optimization*.

[51] H.-K. Wu, S. W.-Y. Lee, H.-Y. Chang, and J.-C. Liang, "Current status, opportunities and challenges of augmented reality in education," *Comput. Educ.*, vol. 62, pp. 41–49, 2013.

[52] M. Kesim and Y. Ozarslan, "Augmented Reality in Education: Current Technologies and the Potential for Education," *Procedia - Soc. Behav. Sci.*, vol. 47, no. 222, pp. 297–302, Jan. 2012.

[53] A. Clark, A. Dünser, and R. Grasset, "An interactive augmented reality coloring book," *SIGGRAPH Asia 2011 Emerg. Technol. - SA '11*, pp. 1–1, 2011.

[54] T. Miyashita, P. Meier, T. Tachikawa, S. Orlic, T. Eble, V. Scholz, A. Gapel, O. Gerl, S. Arnaudov, and S. Lieberknecht, "An augmented reality museum guide," in *Proceedings - 7th IEEE International Symposium on Mixed and Augmented Reality 2008, ISMAR 2008*, 2008, pp. 103–106.

[55] Y. Lu and S. Smith, "Augmented Reality e-Commerce Assistant System: Trying While Shopping," in *Proceedings of the 12th International Conference on Human-computer Interaction: Interaction Platforms and Techniques*, 2007, pp. 643–652.

[56] W. Zhu, C. B. Owen, H. Li, and J.-H. Lee, “Design of the PromoPad: An Automated Augmented-Reality Shopping Assistant,” in *Computational Advancements in End-User Technologies: Emerging Models and Frameworks*, S. Clarke, Ed. Hershey, PA, USA: IGI Global, 2010, pp. 193–205.

[57] H. Kaufmann and D. Schmalstieg, “Mathematics and geometry education with collaborative augmented reality,” *Comput. Graph.*, vol. 27, no. 3, pp. 339–345, 2003.

[58] L. Alem, F. Tecchia, and W. Huang, “HandsOnVideo: Towards a Gesture based Mobile AR System for Remote Collaboration,” in *Recent Trends of Mobile Collaborative Augmented Reality Systems*, L. Alem and W. Huang, Eds. Springer New York, 2011, pp. 135–148.

[59] A. Morrison, A. Mulloni, S. Lemmelä, A. Oulasvirta, G. Jacucci, P. Peltonen, D. Schmalstieg, and H. Regenbrecht, “Collaborative use of mobile augmented reality with paper maps,” *Comput. Graph.*, vol. 35, no. 4, pp. 789–799, 2011.

[60] A. Angelopoulou, D. Economou, V. Bouki, A. Psarrou, L. Jin, C. Pritchard, and F. Kolyda, “Mobile augmented reality for cultural heritage,” Springer, 2012.

[61] V. Lepetit, “On Computer Vision for Augmented Reality,” in *Ubiquitous Virtual Reality, 2008. ISUVR 2008. International Symposium on*, 2008, pp. 13–16.

[62] A. I. Comport, É. Marchand, F. Chaumette, and C. De Beaulieu, “A real-time tracker for markerless augmented reality,” pp. 0–9, 2006.

[63] R. Azuma, J. W. Lee, B. Jiang, J. Park, S. You, and U. Neumann, “Tracking in unprepared environments for augmented reality systems,” vol. 23, pp. 787–793, 1999.

[64] J. P. Rolland, L. Davis, and Y. Baillot, “A survey of tracking technology for virtual environments,” *Fundam. wearable Comput. Augment. Real.*, vol. 8, no. September, pp. 1–48, 2001.

[65] D. Koller, G. Klinkerl, E. Rosel, D. Breen, R. Whit, and M. Tuceryan, “Real-time Vision-Based Camera Tracking for Augmented Reality Applications,” 1997.

[66] R. T. Azuma, B. R. Hoff, H. E. N. Iii, R. Sarfaty, M. J. Daily, G. Bishop, V. Chi, G. Welch, R. Nichols, and J. Cannon, “Making Augmented Reality Work Outdoors Requires Hybrid Tracking,” pp. 1–6, 1998.

[67] “Open Source Augmented Reality SDK | ARToolKit.org.” [Online]. Available: http://artoolkit.org/. [Accessed: 02-Nov-2015].

[68] “Getting Started View | Vuforia Library Prod.” [Online]. Available: https://developer.vuforia.com/library/getting-started. [Accessed: 03-Nov-2015].

[69] “Resources - Apple Developer.” [Online]. Available: https://developer.apple.com/resources/. [Accessed: 11-Nov-2015].

[70] “Download Android Studio and SDK Tools | Android Developers.” [Online]. Available: http://developer.android.com/sdk/index.html. [Accessed: 11-Nov-2015].

[71] "Junaio – Augmented Reality Browser." [Online]. Available: http://www.junaio.com/. [Accessed: 11-Nov-2015].

[72] D. Sprott and L. Wilkes, "Understanding service-oriented architecture," *Archit. J.*, vol. 1, pp. 10–17, 2004.

[73] M.-T. Schmidt, B. Hutchison, P. Lambros, and R. Phippen, "The Enterprise Service Bus: Making service-oriented architecture real," *IBM Syst. J.*, vol. 44, no. 4, pp. 781–797, 2005.

[74] N. Komoda, "Service oriented architecture (SOA) in industrial systems," *Ind. Informatics, 2006 IEEE Int. Conf.*, pp. 1–5, 2006.

[75] C. L. Wu, C. F. Liao, and L. C. Fu, "Service-oriented smart-home architecture based on OSGi and mobile-agent technology," *IEEE Trans. Syst. Man Cybern. Part C Appl. Rev.*, vol. 37, no. 2, pp. 193–205, 2007.

[76] A. Ordanini and P. Pasini, "Service co-production and value co-creation: The case for a service-oriented architecture (SOA)," *Eur. Manag. J.*, vol. 26, no. 5, pp. 289–297, 2008.

[77] "XML Soap." [Online]. Available: http://www.w3schools.com/xml/xml_soap.asp. [Accessed: 28-Oct-2015].

[78] "Directory of public SOAP Web Services." [Online]. Available: http://www.service-repository.com/?offset=0&max=10&sort=rating&order=desc. [Accessed: 29-Oct-2015].

[79] "Fielding Dissertation: CHAPTER 5: Representational State Transfer (REST)." [Online]. Available: http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm. [Accessed: 28-Oct-2015].

[80] "Create REST Request." [Online]. Available: https://docs.oracle.com/cd/E27515_01/common/tutorials/conversion_create_rest_request.html. [Accessed: 29-Oct-2015].

[81] "API Directory | ProgrammableWeb." [Online]. Available: http://www.programmableweb.com/apis/directory. [Accessed: 29-Oct-2015].

[82] R. Lee, D. Kitayama, Y.-J. Kwon, and K. Sumiya, "Interoperable augmented web browsing for exploring virtual media in real space," *Proc. 2nd Int. Work. Locat. Web LOCWEB 09*, pp. 1–4, 2009.

[83] B. Petros, Y. Yu, and S. Petri, "Enabling rapid creation of content for consumption in mobile Augmented Reality," *Proc. - NGMAST 2010 4th Int. Conf. Next Gener. Mob. Appl. Serv. Technol.*, pp. 1–6, 2010.

[84] M. Pollefeys, L. Van Gool, M. Vergauwen, K. Cornelis, F. Verbiest, and J. Tops, "Video-to-3D," pp. 1–12.

[85] P. Cignoni, M. Corsini, M. Dellepiane, G. Ranzuglia, M. Vergauven, and L. Van Gool, "MeshLab and Arc3D : Photo-Reconstruction and Processing 3D meshes,"

pp. 1–6, 2008.

[86] "ARC 3D." [Online]. Available: http://www.arc3d.be/. [Accessed: 04-Nov-2015].

[87] M. Vergauwen and L. Gool, "Web-based 3D Reconstruction Service," *Mach. Vis. Appl.*, vol. 17, no. 6, pp. 411–426, May 2006.

[88] C. Heipke, "Web-Based Photogrammetric Image and Geospatial Services – an Overview," no. Schiewe, pp. 1–7, 2005.

[89] "Welcome to the OGC | OGC." [Online]. Available: http://www.opengeospatial.org/. [Accessed: 05-Nov-2015].

[90] K. Sahin, "Service oriented architecture (SOA) based web services for geographic information systems," *XXIst ISPRS Congr. Beijing*, pp. 625–630, 2008.

[91] B. Hagedorn and J. Döllner, "High-level web service for 3D building information visualization and analysis," *Proc. 15th Annu. ACM Int. Symp. Adv. Geogr. Inf. Syst. - GIS '07*, p. 1, 2007.

[92] K. Park and A. Yilmaz, "A Design of Web Service for Digital Photogrammetry Workstation using Service Oriented Architecture."

[93] "Introduction to the Europeana APIs - Europeana Labs." [Online]. Available: http://labs.europeana.eu/api. [Accessed: 12-Nov-2015].

[94] "Amazon Web Services (AWS) - Cloud Computing Services." [Online]. Available: https://aws.amazon.com/. [Accessed: 06-Nov-2015].

[95] K. B. Laskey and K. Laskey, "Service oriented architecture," *Wiley Interdiscip. Rev. Comput. Stat.*, vol. 1, no. 1, pp. 101–105, 2009.

[96] T. Erl, *Service-Oriented Architecture: A Field Guide to Integrating XML and Web Services*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2004.

[97] G. Kunito, K. Sakamoto, N. Yamada, T. Takakashi, and S. Tanaka, "Architecture for Providing Services in the Ubiquitous Computing Environment," *26th IEEE Int. Conf. Distrib. Comput. Syst. Work. 2006 ICDCS Work. 2006*, pp. 60–60, 2006.

[98] "SierraLeoneHeritageApi examples." [Online]. Available: http://sierraleone.heritageinformatics.org/api/. [Accessed: 12-Nov-2015].

[99] "Victoria and Albert Museum API Documentation." [Online]. Available: http://www.vam.ac.uk/api/. [Accessed: 12-Nov-2015].

[100] "Google Maps APIs | Google Developers." [Online]. Available: https://developers.google.com/maps/. [Accessed: 21-Aug-2016].

[101] J. Heller, M. Havlena, M. Jancosek, A. Torii, and T. Pajdla, "3D reconstruction from photographs by CMP SfM web service," *Machine Vision Applications (MVA), 2015 14th IAPR International Conference on*. pp. 30–34, 2015.

[102] "Developer Autodesk ReCap API Samples." [Online]. Available: http://developer-recap-autodesk.github.io/. [Accessed: 12-Nov-2015].

[103] C. PortalÃ©s, J. L. Lerma, and C. PÃ©rez, "Photogrammetry and augmented reality for cultural heritage applications," *Photogramm. Rec.*, vol. 24, no. 128, pp. 316–331, Dec. 2009.

[104] A.-C. Haugstvedt and J. Krogstie, "Mobile augmented reality for cultural heritage: A technology acceptance study," in *2012 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, 2012, pp. 247–255.

[105] P. Petridis, M. White, N. Mourkousis, F. Liarokapis, M. Sifniotis, A. Basu, and C. Gatzidis, "Exploring and interacting with virtual museums," *Proc. Comput. Appl. Quant. Methods Archaeol.*, 2005.

[106] "V&A Home Page - Victoria and Albert Museum," Oct. 2015.

[107] "British Museum - Welcome to the British Museum." [Online]. Available: http://www.britishmuseum.org/. [Accessed: 12-Nov-2015].

[108] "Europeana - Homepage." [Online]. Available: http://www.europeana.eu/portal/. [Accessed: 12-Nov-2015].

[109] "SierraLeoneHeritage.org | Digital Heritage Resource." [Online]. Available: http://www.sierraleoneheritage.org/. [Accessed: 12-Nov-2015].

[110] J. P. Lima, F. Simões, L. Figueiredo, and J. Kelner, "Model Based Markerless 3D Tracking applied to Augmented Reality," *Evaluation*, vol. 1, pp. 2–15, 2010.

[111] M. Pressigout and E. Marchand, "Hybrid tracking algorithms for planar and non-planar structures subject to illumination changes," *2006 IEEE/ACM International Symposium on Mixed and Augmented Reality*. pp. 52–55, 2006.

[112] U. Neumann, "Virtual Reality Dynamic Registration Correction in Augmented Reality Systems," *IEEE Comput. Graph. Appl.*, no. September, pp. 52–60, 1995.

[113] U. Neumann and S. You, "Natural feature tracking for augmented reality," *IEEE Transactions on Multimedia*, vol. 1, no. 1. pp. 53–64, 1999.

[114] D. Wagner, T. Langlotz, and D. Schmalstieg, "Robust and unobtrusive marker tracking on mobile phones," *2008 7th IEEE/ACM Int. Symp. Mix. Augment. Real.*, pp. 121–124, Sep. 2008.

[115] H. Kato, K. Tachibana, M. Billinghurst, and M. Grafe, "A registration method based on texture tracking using ARToolKit," *IEE Rev.*, pp. 77–85.

[116] D. Wagner, G. Reitmayr, A. Mulloni, T. Drummond, and D. Schmalstieg, "Pose tracking from natural features on mobile phones," *2008 7th IEEE/ACM Int. Symp. Mix. Augment. Real.*, pp. 125–134, Sep. 2008.

[117] T. Lee and T. Höllerer, "Multithreaded hybrid feature tracking for markerless augmented reality.," *IEEE Trans. Vis. Comput. Graph.*, vol. 15, no. 3, pp. 355–68, 2009.

[118] R. Koch, K. Koeser, B. Streckel, and C. Kiel, "Markerless Image-based 3D

Tracking for Real-time Augmented Reality Applications.”

[119] N. Karlsson, E. Di Bernardo, J. Ostrowski, L. Goncalves, P. Pirjanian, and M. E. Munich, “The vSLAM Algorithm for Robust Localization and Mapping,” no. April, pp. 24–29, 2005.

[120] C. Oberhofer, J. Grubert, and G. Reitmayr, “Natural feature tracking in JavaScript,” *2012 IEEE Virtual Real.*, vol. 4, pp. 113–114, Mar. 2012.

[121] Z. Pan, Y. Li, M. Zhang, C. Sun, K. Guo, X. Tang, and S. Z. Zhou, “A Real-time Multi-cue Hand Tracking Algorithm Based on Computer Vision,” pp. 219–222.

[122] G. Reitmayr and T. Drummond, “Going out: robust model-based tracking for outdoor augmented reality,” *2006 IEEE/ACM Int. Symp. Mix. Augment. Real.*, pp. 109–118, Oct. 2006.

[123] L. Vacchetti, V. Lepetit, and P. Fua, “Stable real-time 3D tracking using online and offline information.,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 26, no. 10, pp. 1385–91, Oct. 2004.

[124] N. Krahnstoever and R. Sharma, “Appearance Management and Cue Fusion for 3D Model-Based Tracking,” in *Proc. IEEE Conf. Comp. Vision and Pattern Recognition (CPVR)*, 2003, pp. 249–256.

[125] I. Skrypnyk and D. G. Lowe, “Scene Modelling , Recognition and Tracking with Invariant Image Features,” no. Ismar, 2004.

[126] H. Wuest, F. Vial, and D. Stricker, “Adaptive Line Tracking with Multiple Hypotheses for Augmented Reality Department of Virtual and Augmented Reality,” pp. 0–7, 2005.

[127] C. Wiedemann, M. Ulrich, and C. Steger, “Recognition and tracking of 3D objects,” in *PATTERN RECOGNITION*, 2008, vol. 5096, pp. 132–141.

[128] M. Ozuysal, P. Fua, and V. Lepetit, “Fast Keypoint Recognition in Ten Lines of Code,” in *2007 IEEE Conference on Computer Vision and Pattern Recognition*, 2007, pp. 1–8.

[129] N. Hagbi, O. Bergig, J. El-Sana, and M. Billinghurst, “Shape recognition and pose estimation for mobile Augmented Reality.,” *IEEE Trans. Vis. Comput. Graph.*, vol. 17, no. 10, pp. 1369–79, Oct. 2011.

[130] Y. Kalantidis, L. G. Pueyo, M. Trevisiol, R. van Zwol, and Y. Avrithis, “Scalable Triangulation-based Logo Recognition,” in *Proceedings of the 1st ACM International Conference on Multimedia Retrieval*, 2011, pp. 20:1–20:7.

[131] P. J. Gausemeier, “Development of a Real Time Image Based Object Recognition Method for Mobile AR-Devices,” *Architecture*, vol. 1, no. 212, pp. 133–140, 2003.

[132] C. M. Cyr and B. B. Kimia, “3D object recognition using shape similiarity-based aspect graph,” *Computer Vision, 2001. ICCV 2001. Proceedings. Eighth IEEE International Conference on*, vol. 1. pp. 254–261 vol.1, 2001.

[133] I. Gordon and D. G. Lowe, "What and Where : 3D Object Recognition with Accurate Pose," *Lect. Notes Comput. Sci.*, 2004.

[134] S. Xu and Q. Peng, "3D object recognition using multiple features and neural network," *2008 IEEE Conf. Cybern. Intell. Syst.*, pp. 434–439, Sep. 2008.

[135] B. Scholz-reiter, H. Thamer, and C. Uriarte, "An Approach for 3D Object Recognition of Universal Goods," *Int. J.*, vol. 5, no. 2, pp. 218–225, 2011.

[136] H. Akbar, N. Suryana, and S. Sahib, "Training neural networks using Clonal Selection Algorithm and Particle Swarm Optimization: A comparisons for 3D object recognition," *2011 11th Int. Conf. Hybrid Intell. Syst.*, pp. 692–697, Dec. 2011.

[137] R. Freeman and a. Steed, "Interactive modelling and tracking for mixed and augmented reality," *Proc. ACM Symp. Virtual Real. Softw. Technol. - VRST '06*, p. 61, 2006.

Appendix

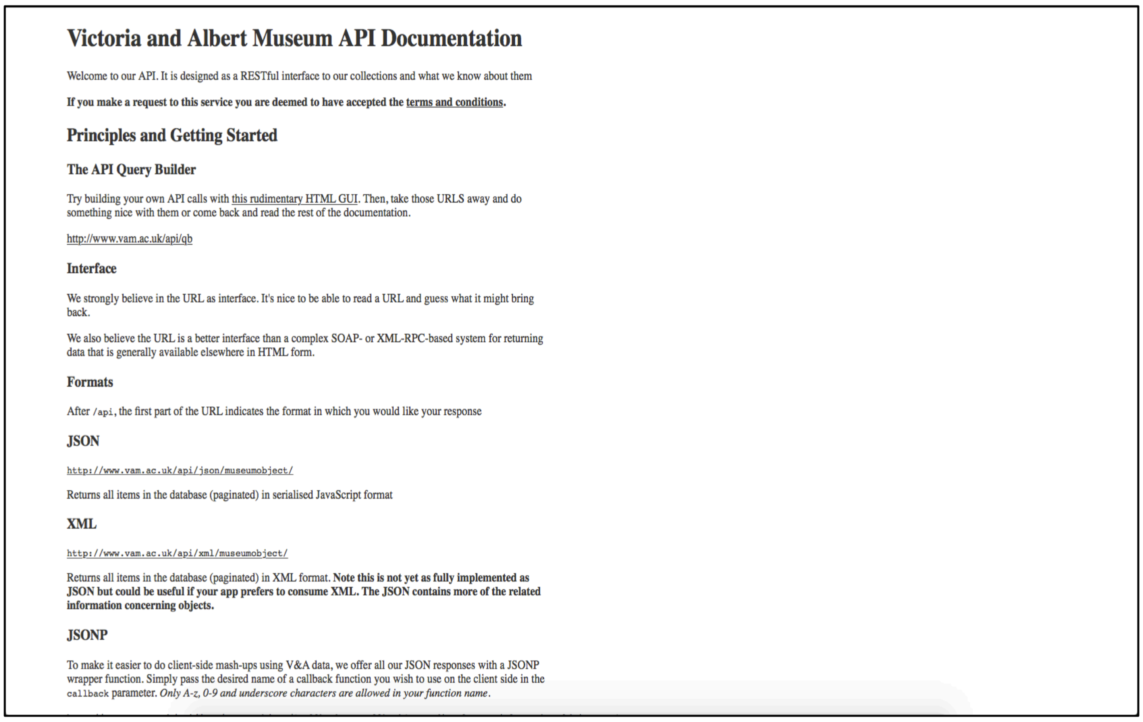
A. Web Service Provider APIs

Victoria and Albert Museum API

Victoria and Albert Museum provides a website (www.vam.ac.uk) that offers online collections and allow museum visitors to view and learn cultural artifacts on the website.

Figure A.1 Victoria and Albert Museum

The museum also expose an API called Victoria and Albert Museum API that provides service interfaces for any applications to access data and metadata of objects exhibited in the Victoria and Albert Museum.



Victoria and Albert Museum API Documentation

Welcome to our API. It is designed as a RESTful interface to our collections and what we know about them

If you make a request to this service you are deemed to have accepted the terms and conditions.

Principles and Getting Started

The API Query Builder

Try building your own API calls with this rudimentary HTML GUI. Then, take those URLS away and do something nice with them or come back and read the rest of the documentation.

http://www.vam.ac.uk/api/qb

Interface

We strongly believe in the URL as interface. It's nice to be able to read a URL and guess what it might bring back.

We also believe the URL is a better interface than a complex SOAP- or XML-RPC-based system for returning data that is generally available elsewhere in HTML form.

Formats

After `/api`, the first part of the URL indicates the format in which you would like your response

JSON

`http://www.vam.ac.uk/api/json/museumobject/`

Returns all items in the database (paginated) in serialised JavaScript format

XML

`http://www.vam.ac.uk/api/xml/museumobject/`

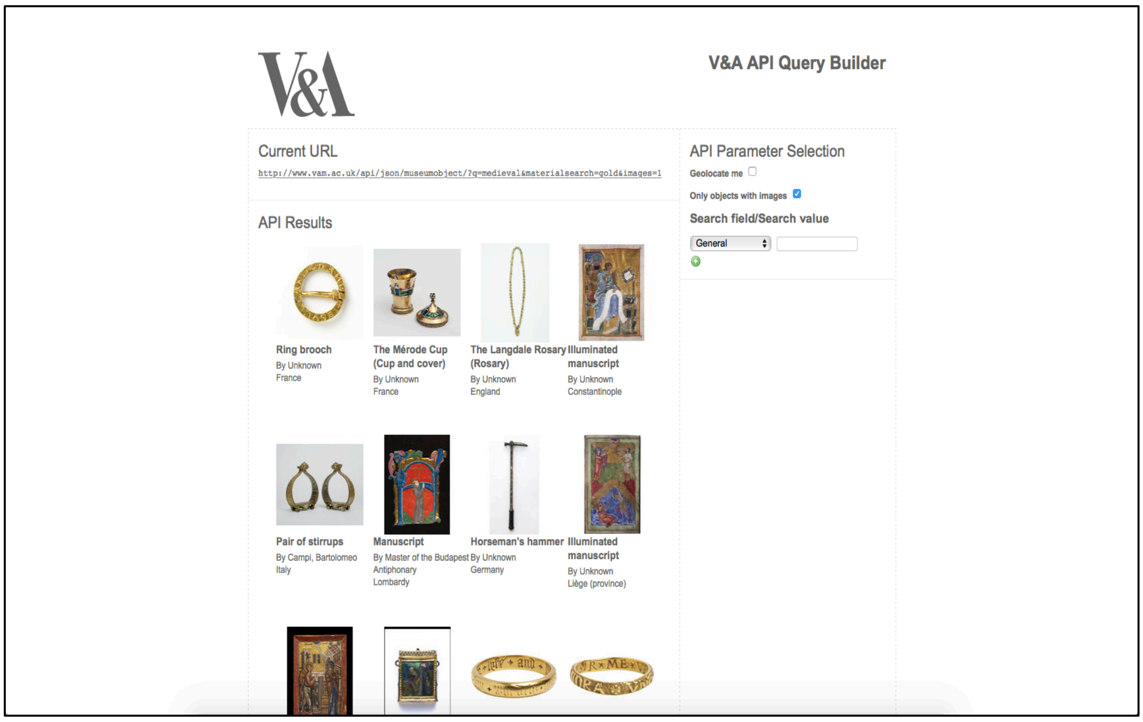
Returns all items in the database (paginated) in XML format. **Note this is not yet as fully implemented as JSON but could be useful if your app prefers to consume XML. The JSON contains more of the related information concerning objects.**

JSONP

To make it easier to do client-side mash-ups using V&A data, we offer all our JSON responses with a JSONP wrapper function. Simply pass the desired name of a callback function you wish to use on the client side in the `callback` parameter. *Only A-z, 0-9 and underscore characters are allowed in your function name.*

Figure A.2 Victoria and Albert Museum API

The API Query Builder enables users to create API calls that can be used in the web service request module in the application client.



V&A API Query Builder

Current URL

`http://www.vam.ac.uk/api/json/museumobject/?q=medieval&materialsearch=gold&images=1`

API Parameter Selection

Geolocate me ☐

Only objects with images ☑

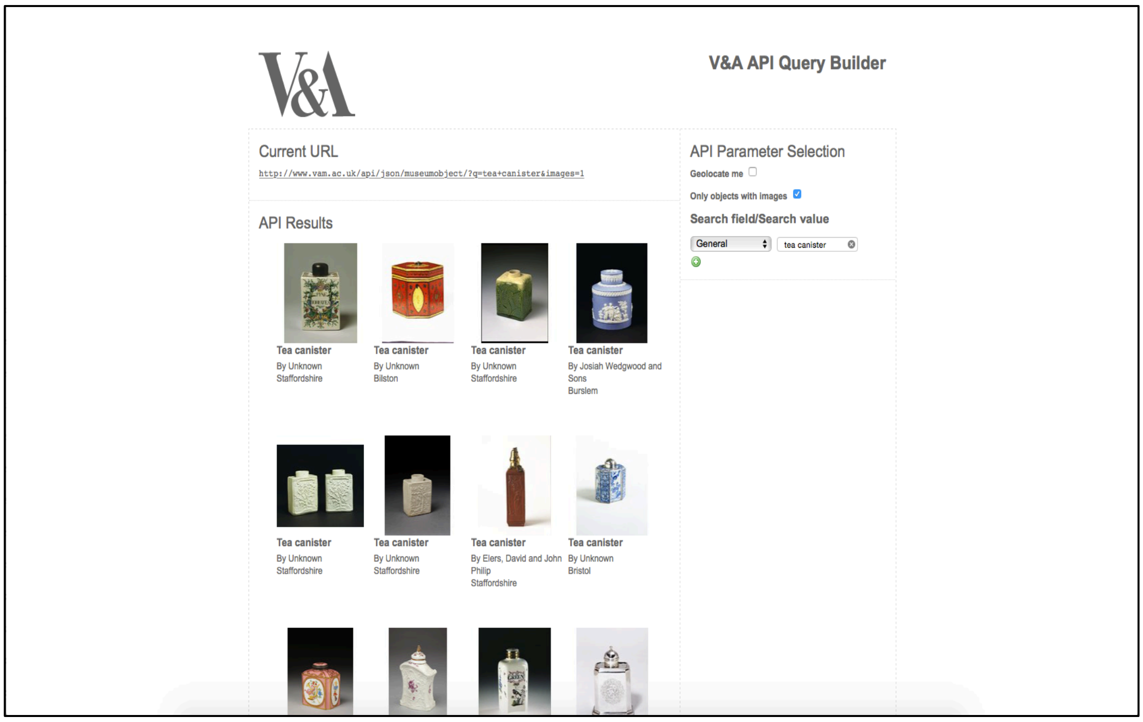
Search field/Search value

General

API Results

Figure A.3 The API Query Builder

The Builder creates an URL of a web service request composed of the source of content, the module requested, parameters and the output format. The outcome of a service request is presented on the Builder.


V&A API Query Builder

Current URL
http://www.vam.ac.uk/api/json/museumobject/?q=tea+canister&images=1

API Parameter Selection
Geolocate me ☐
Only objects with images ✓
Search field/Search value
General | tea canister

API Results

Tea canister
By Unknown
Staffordshire

Tea canister
By Unknown
Bilston

Tea canister
By Unknown
Staffordshire

Tea canister
By Josiah Wedgwood and Sons
Burslem

Tea canister
By Unknown
Staffordshire

Tea canister
By Unknown
Staffordshire

Tea canister
By Elers, David and John Philip
Staffordshire

Tea canister
By Unknown
Bristol

Figure A.4 The outcome of a service request

The following is a JSON response document of the web service request http://www.vam.ac.uk/api/json/museumobject/?q=tea+canister&images=1.

```
{ "meta": {
      "result_count":170 ,
      "cluster_counts" : {},
      "clusters" : [] ,
      "group_details": [] },
 "records": [
   {
      "pk": 8968,
      "model": "collection.museumobject",
      "fields": {
         "primary_image_id": "2006AM7317",
         "rights": 3,
         "year_start": 1760,
         "object_number": "O77973",
         "artist": "Unknown",
         "museum_number": "C.69&A-1938",
         "object": "Tea canister",
         "longitude": "-2.02806000",
         "last_processed": "2016-04-01 23:21:01",
      "event_text": "",
      "place": "Staffordshire",
      "location": "British Galleries, room 52b, case 1",
      "last_checked": "2016-04-01 23:21:01",
      "museum_number_token": "c691938",
      "latitude": "52.82474900",
      "title": "",
      "date_text": "1760-1770 (made)",
      "slug": "tea-canister-unknown",
      "sys_updated": "2015-01-26 00:00:00",
      "collection_code": "CER"
    }
  },
  {
    "pk": 9835,
    "model": "collection.museumobject",
    "fields": {
      "primary_image_id": "2006AM2645",
      "rights": 3,
      "year_start": 1780,
      "object_number": "O79070",
      "artist": "Unknown",
      "museum_number": "W.72-1919",
      "object": "Tea canister",
      "longitude": "-2.07449000",
      "last_processed": "2016-04-01 23:26:05",
      "event_text": "",
      "place": "Bilston",
      "location": "British Galleries, room 118e, case 1",
      "last_checked": "2016-04-01 23:26:05",
      "museum_number_token": "w721919",
      "latitude": "52.56663500",
      "title": "",
      "date_text": "1780-1800 (made)",
      "slug": "tea-canister-unknown",
      "sys_updated": "2014-08-04 00:00:00",
      "collection_code": "FWK"
    }
  },
  {
    "pk": 8858,
    "model": "collection.museumobject",
    "fields": {
      "primary_image_id": "2006AT4084",
      "rights": 3,
      "year_start": 1760,
      "object_number": "O77863",
      "artist": "Unknown",
      "museum_number": "C.16-1940",
      "object": "Tea canister",
      "longitude": "-2.02806000",
      "last_processed": "2016-04-01 23:20:26",
      "event_text": "",
      "place": "Staffordshire",
      "location": "British Galleries, room 118e, case 3",
        "last_checked": "2016-04-01 23:20:26",
        "museum_number_token": "c161940",
        "latitude": "52.82474900",
        "title": "",
        "date_text": "1760-1765 (made)",
        "slug": "tea-canister-unknown",
        "sys_updated": "2014-07-31 00:00:00",
        "collection_code": "CER"
    }
},
{
    "pk": 8832,
    "model": "collection.museumobject",
    "fields": {
        "primary_image_id": "2006AM3394",
        "rights": 3,
        "year_start": 1795,
        "object_number": "O77834",
        "artist": "Josiah Wedgwood and Sons",
        "museum_number": "C.112&A-1956",
        "object": "Tea canister",
        "longitude": "-2.18420000",
        "last_processed": "2016-04-01 23:20:18",
        "event_text": "",
        "place": "Burslem",
        "location": "British Galleries, room 118e, case 3",
        "last_checked": "2016-04-01 23:20:18",
        "museum_number_token": "c1121956",
        "latitude": "53.05424900",
        "title": "",
        "date_text": "1795-1800 (made)",
        "slug": "tea-canister-josiah-wedgwood-and",
        "sys_updated": "2015-06-24 00:00:00",
        "collection_code": "CER"
    }
},
{
    "pk": 13550,
    "model": "collection.museumobject",
    "fields": {
        "primary_image_id": "2006AM3440",
        "rights": 3,
        "year_start": 1745,
        "object_number": "O79779",
        "artist": "Unknown",
        "museum_number": "414:958-1885",
        "object": "Tea canister",
        "longitude": "-2.02806000",
        "last_processed": "2016-04-01 23:47:26",
        "event_text": "",
        "place": "Staffordshire",
        "location": "British Galleries, room 52b, case 1",
        "last_checked": "2016-04-01 23:47:26",
        "museum_number_token": "4149581885",
        "latitude": "52.82474900",
        "title": "",
        "date_text": "ca. 1750 (made)",
        "slug": "tea-canister-unknown",
        "sys_updated": "2014-11-19 00:00:00",
        "collection_code": "CER"
      }
    },
    {
      "pk": 154194,
      "model": "collection.museumobject",
      "fields": {
        "primary_image_id": "2009CT0498",
        "rights": 3,
        "year_start": 1745,
        "object_number": "O190483",
        "artist": "Unknown",
        "museum_number": "414:958/A-1885",
        "object": "Tea canister",
        "longitude": "-2.02806000",
        "last_processed": "2016-04-02 11:55:07",
        "event_text": "",
        "place": "Staffordshire",
        "location": "Ceramics Study Galleries, Britain, room 138, case 7, shelf 3",
        "last_checked": "2016-04-02 11:55:07",
        "museum_number_token": "414958a1885",
        "latitude": "52.82474900",
        "title": "",
        "date_text": "ca. 1750 (made)",
        "slug": "tea-canister-unknown",
        "sys_updated": "2015-05-26 00:00:00",
        "collection_code": "CER"
      }
    },
    {
      "pk": 8898,
      "model": "collection.museumobject",
      "fields": {
        "primary_image_id": "2006AM5304",
        "rights": 3,
        "year_start": 1690,
        "object_number": "O77903",
        "artist": "Elers, David and John Philip",
        "museum_number": "C.275-1921",
        "object": "Tea canister",
        "longitude": "-2.02806000",
        "last_processed": "2016-04-01 23:20:39",
        "event_text": "",
        "place": "Staffordshire",
        "location": "British Galleries, room 56c, case 4",
        "last_checked": "2016-04-01 23:20:39",
        "museum_number_token": "c2751921",
        "latitude": "52.82474900",
        "title": "",
        "date_text": "1690-1698 (made)",
        "slug": "tea-canister-elers-david-and",
      "sys_updated": "2015-07-17 00:00:00",
      "collection_code": "CER"
    }
  },
  {
    "pk": 22371,
    "model": "collection.museumobject",
    "fields": {
      "primary_image_id": "2006AJ5280",
      "rights": 3,
      "year_start": 1745,
      "object_number": "O21252",
      "artist": "Unknown",
      "museum_number": "C.13-1963",
      "object": "Tea canister",
      "longitude": "-2.59143000",
      "last_processed": "2016-04-02 00:37:43",
      "event_text": "",
      "place": "Bristol",
      "location": "British Galleries, room 52b, case 1",
      "last_checked": "2016-04-02 00:37:43",
      "museum_number_token": "c131963",
      "latitude": "51.45365900",
      "title": "",
      "date_text": "ca. 1750-1760 (made)",
      "slug": "tea-canister-unknown",
      "sys_updated": "2014-11-19 00:00:00",
      "collection_code": "CER"
    }
  },
  {
    "pk": 75427,
    "model": "collection.museumobject",
    "fields": {
      "primary_image_id": "2010EJ2514",
      "rights": 3,
      "year_start": 1756,
      "object_number": "O99172",
      "artist": "S\u00e8vres porcelain factory",
      "museum_number": "768A-1882",
      "object": "Tea canister",
      "longitude": "1.71832000",
      "last_processed": "2016-04-02 05:04:36",
      "event_text": "",
      "place": "France",
      "location": "Europe 1600-1815, lift lobby, case CA2",
      "last_checked": "2016-04-02 05:04:36",
      "museum_number_token": "768a1882",
      "latitude": "46.71244800",
      "title": "Boite a the",
      "date_text": "ca. 1761 (made)",
      "slug": "boite-a-the-tea-canister-sevres-porcelain-factory",
      "sys_updated": "2015-12-04 00:00:00",
      "collection_code": "CER"
    }
```

```
},
{
  "pk": 273307,
  "model": "collection.museumobject",
  "fields": {
    "primary_image_id": "2010ED2603",
    "rights": 3,
    "year_start": 1755,
    "object_number": "O334362",
    "artist": "F\u00fcrstenberg Porcelain",
    "museum_number": "C.50&A-1956",
    "object": "Tea canister and cover",
    "longitude": "9.66248000",
    "last_processed": "2016-04-02 19:00:12",
    "event_text": "",
    "place": "F\u00fcrstenberg",
    "location": "Europe 1600-1815, room 3, case CA3",
    "last_checked": "2016-04-02 19:00:12",
    "museum_number_token": "c501956",
    "latitude": "52.36421000",
    "title": "",
    "date_text": "ca. 1760 (made)",
    "slug": "tea-canister-and-furstenberg-porcelain",
    "sys_updated": "2015-12-04 00:00:00",
    "collection_code": "CER"
  }
},
{
  "pk": 8352,
  "model": "collection.museumobject",
  "fields": {
    "primary_image_id": "2006AM5978",
    "rights": 3,
    "year_start": 1755,
    "object_number": "O77657",
    "artist": "Swirled Flowers Painter",
    "museum_number": "5288&A-1901",
    "object": "Tea canister",
    "longitude": "-1.81541000",
    "last_processed": "2016-04-01 23:18:13",
    "event_text": "",
    "place": "West Midlands",
    "location": "British Galleries, room 52b, case 1",
    "last_checked": "2016-04-01 23:18:13",
    "museum_number_token": "52881901",
    "latitude": "52.50523000",
    "title": "",
    "date_text": "1755-1760 (made)",
    "slug": "tea-canister-swirled-flowers-painter",
    "sys_updated": "2014-11-19 00:00:00",
    "collection_code": "CER"
  }
},
{
  "pk": 9477,
    "model": "collection.museumobject",
    "fields": {
      "primary_image_id": "2006AM2247",
      "rights": 3,
      "year_start": 1717,
      "object_number": "O78644",
      "artist": "Farnell, John",
      "museum_number": "M.854:1, 2-1926",
      "object": "Tea canister",
      "longitude": "-0.12714000",
      "last_processed": "2016-04-01 23:23:47",
      "event_text": "",
      "place": "London",
      "location": "British Galleries, room 52b, case 1",
      "last_checked": "2016-04-01 23:23:47",
      "museum_number_token": "m8541926",
      "latitude": "51.50632100",
      "title": "",
      "date_text": "1717-1718 (made)",
      "slug": "tea-canister-farnell-john",
      "sys_updated": "2014-11-19 00:00:00",
      "collection_code": "MET"
    }
  },
  {
    "pk": 9363,
    "model": "collection.museumobject",
    "fields": {
      "primary_image_id": "2006AM6828",
      "rights": 3,
      "year_start": 1773,
      "object_number": "O78523",
      "artist": "Courtauld, Louisa",
      "museum_number": "M.28-1934",
      "object": "Tea canister",
      "longitude": "-0.12714000",
      "last_processed": "2016-04-01 23:23:09",
      "event_text": "",
      "place": "London",
      "location": "British Galleries, room 118e, case 8",
      "last_checked": "2016-04-01 23:23:09",
      "museum_number_token": "m281934",
      "latitude": "51.50632100",
      "title": "",
      "date_text": "1773-1774 (hallmarked)",
      "slug": "tea-canister-courtauld-louisa",
      "sys_updated": "2014-08-14 00:00:00",
      "collection_code": "MET"
    }
  },
  {
    "pk": 9830,
    "model": "collection.museumobject",
    "fields": {
      "primary_image_id": "2006AM6303",
            "rights": 3,
            "year_start": 1685,
            "object_number": "O79065",
            "artist": "Cradock, Marmaduke",
            "museum_number": "W.70-1919",
            "object": "Tea canister",
            "longitude": "-0.12714000",
            "last_processed": "2016-04-01 23:26:03",
            "event_text": "",
            "place": "London",
            "location": "British Galleries, room 56d, case 14",
            "last_checked": "2016-04-01 23:26:03",
            "museum_number_token": "w701919",
            "latitude": "51.50632100",
            "title": "",
            "date_text": "1685-1717 (made)",
            "slug": "tea-canister-cradock-marmaduke",
            "sys_updated": "2014-08-04 00:00:00",
            "collection_code": "FWK"
        }
    },
    {
        "pk": 13211,
        "model": "collection.museumobject",
        "fields": {
            "primary_image_id": "2006AM7382",
            "rights": 3,
            "year_start": 1722,
            "object_number": "O10916",
            "artist": "Nash, Bowles",
            "museum_number": "M.180A/1, 2-1919",
            "object": "Pair of tea canisters",
            "longitude": "-0.12714000",
            "last_processed": "2016-04-01 23:45:37",
            "event_text": "",
            "place": "London",
            "location": "British Galleries, room 52b, case 1",
            "last_checked": "2016-04-01 23:45:37",
            "museum_number_token": "m1801919",
            "latitude": "51.50632100",
            "title": "",
            "date_text": "1722-1723 (made)",
            "slug": "pair-of-tea-nash-bowles",
            "sys_updated": "2015-07-01 00:00:00",
            "collection_code": "MET"
        }
    }
]}
```

The image of each object presented on the Builder can be selected and the system will create a service quest of the selected image for its metadata. The web service request is : http://www.vam.ac.uk/api/json/museumobject/O79070.

```
[
   {
      "pk": 9835,
      "model": "collection.museumobject",
      "fields": {
         "original_price": "",
         "attributions_note": "",
         "related_museum_numbers": "",
         "museum_number": "W.72-1919",
         "date_end": "1800-12-31",
         "labels": [
            {
               "pk": 3830,
               "model": "collection.label",
               "fields": {
                  "date": "27/03/2003",
                  "museumobject": 9835,
                  "label_text": "British Galleries:\nThis canister combines bright-cut engraving, a
technique introduced to silver in the 1770s, with areas of paint and a final layer of varnish. The
tin appears yellow because of the varnish."
               }
            }
         ],
         "descriptive_line": "Tea canister of elongated hexagonal form, of tin, with incised
decoration, decortated with red japanning and lacquering imitating gilding",
         "shape": "",
         "longitude": "-2.07449000",
         "year_start": 1780,
         "exhibitions": [],
         "subjects": [],
         "date_text": "1780-1800 (made)",
         "primary_image_id": "2006AM2645",
         "rights": 3,
         "physical_description": "",
         "dimensions": "Height: 11.11 cm, Width: 13.97 cm, Depth: 8.57 cm",
         "title": "",
         "date_start": "1780-01-01",
         "materials_techniques": "Tin, bright-cut engraved and japanned, with brass handle",
         "last_processed": "2016-04-01 23:26:05",
         "label": "British Galleries:\nThis canister combines bright-cut engraving, a technique
introduced to silver in the 1770s, with areas of paint and a final layer of varnish. The tin appears
yellow because of the varnish. [27/03/2003]",
         "event_text": "",
         "production_type": "",
         "collections": [
            {
               "pk": 6,
               "model": "collection.collection",
               "fields": {
                  "code": "FWK",
                  "name": "Furniture and Woodwork Collection",
                  "museumobject_count": 10947,
                  "source": "",
                  "cis_id": null,
                  "museumobject_image_count": 6143,
            "type": "",
            "slug": "fwk"
          }
        }
      ],
      "location": "British Galleries, room 118e, case 1",
      "marks": "",
      "latitude": "52.56663500",
      "techniques": [],
      "materials": [],
      "edition_number": "",
      "styles": [],
      "inventory_set": [
        {
          "pk": 11995,
          "model": "collection.inventory",
          "extras": {
            "gallery_id": 160
          },
          "fields": {
            "box": "",
            "case": "1",
            "inventory_number": 528323,
            "room": "118E",
            "part_name": "",
            "museum_number": "W.72-1919",
            "museumobject": 9835,
            "shelf": "",
            "site": "VA",
            "on_display": true,
            "status": "",
            "location": "British Galleries, room 118e",
            "museum_number_token": "w721919",
            "gallery": 160
          }
        }
      ],
      "updated": null,
      "galleries": [
        {
          "pk": 160,
          "model": "collection.gallery",
          "fields": {
            "room_code": "118E",
            "site_name": "",
            "name": "British Galleries, room 118e",
            "site_code": "VA",
            "museumobject_count": 163,
            "source": "",
            "on_display": true,
            "cis_id": "THES49228",
            "museumobject_image_count": 161,
            "type": "",
            "slug": "118e-va"
          }
```

```
    }
],
"names": [
    {
        "pk": 3,
        "model": "collection.name",
        "fields": {
            "death_date": null,
            "surname": "",
            "name": "unknown",
            "gender": null,
            "museumobject_count": 164487,
            "death_year": null,
            "source": "object_production",
            "cis_id": "A1848",
            "museumobject_image_count": 83564,
            "forename": "",
            "birth_date": null,
            "nationality": "",
            "type": "person",
            "slug": "unknown",
            "birth_year": null
        }
    }
],
"placecontext_set": [
    {
        "pk": 10782,
        "model": "collection.placecontext",
        "extras": {
            "place_id": 572
        },
        "fields": {
            "part_name": "",
            "uncertainty": "or Wolverhampton",
            "museumobject": 9835,
            "role": "made",
            "place": 572,
            "order": 1
        }
    }
],
"original_currency": "",
"museum_number_token": "w721919",
"object": "Tea canister",
"categories": [
    {
        "pk": 13,
        "model": "collection.category",
        "fields": {
            "name": "Tea, Coffee & Chocolate wares",
            "museumobject_count": 2516,
            "source": "cis_category",
            "cis_id": null,
            "museumobject_image_count": 2080,
                "type": "",
                "slug": "tea-coffee-chocolate-wares"
            }
        },
        {
            "pk": 12,
            "model": "collection.category",
            "fields": {
                "name": "British Galleries",
                "museumobject_count": 1596,
                "source": "cis_category",
                "cis_id": null,
                "museumobject_image_count": 1476,
                "type": "",
                "slug": "british-galleries"
            }
        }
    ],
    "last_checked": "2016-04-01 23:26:05",
    "public_access_description": "Object Type\nTea canisters were used to store loose tea leaves.  They were called 'canisters' until about 1800, when the term caddy began to be used. They were placed on the table as tea was served and were therefore decorated in a variety of fashionable styles.\n\nOwnership & Use\nTea was a popular drink even in middle-income households in the later 18th century although the price was high owing to import restrictions and duties.  Tea canisters had locks to safeguard the valuable tea. All tea came from China until 1839 when Indian tea began to be imported. This canister has two compartments inside to keep different types of tea apart. Black teas were Bohea, Congou and Souchong, and the more expensive green teas Singlo or Hyson.  Blending the teas was an essential part of the tea-making ritual.\n\nDesign & Designing\nThis metal tea canister is hexagonal, a popular shape for canisters and the angled sides would have shown off the sparkling effect of the metal in the incised patterns and the gold  stars, border patterns and lozenge shapes. The inside is lined with metal foil.",
    "exhibition_history": "",
    "bibliography": "Jones, Yvonne, <i>Japanned Papier-M\u00e2ch\u00e9 and Tinware c. 1740-1940</i>. Woodbridge, Antique Collectors' Club, 2012 (ISBN 978 1 85149 686 0), p. 49, fig. 36",
    "vanda_exhibition_history": "",
    "slug": "tea-canister-unknown",
    "sys_updated": "2014-08-04 00:00:00",
    "image_set": [
        {
            "pk": 5124,
            "model": "collection.image",
            "fields": {
                "sys_updated": "2012-10-13 00:00:00",
                "last_processed": "2016-04-01 12:52:43",
                "priority": 0,
                "image_id": "2006AM2645",
                "sys_id": 112645,
                "last_checked": "2016-04-01 12:52:43",
                "size_bytes": null,
                "local": "collection_images/2006AM/2006AM2645.jpg"
            }
        }
    ],
        "places": [
          {
            "pk": 572,
            "model": "collection.place",
            "extras": {
              "parent_id": 546
            },
            "fields": {
              "name": "Bilston",
              "parent": 546,
              "country": "England",
              "museumobject_count": 3,
              "longitude": "-2.07449000",
              "source": "production",
              "cis_id": "x30593",
              "museumobject_image_count": 3,
              "latitude": "52.56663500",
              "type": "city/town",
              "slug": "bilston-england-x30593"
            }
          }
        ],
        "artist": "Unknown",
        "namecontext_set": [
          {
            "pk": 9714,
            "model": "collection.namecontext",
            "extras": {
              "name_id": 3
            },
            "fields": {
              "name": 3,
              "part_name": "",
              "uncertainty": "",
              "museumobject": 9835,
              "role": "production",
              "order": 1
            }
          }
        ],
        "historical_significance": "",
        "year_end": 1800,
        "object_number": "O79070",
        "events": [],
        "credit": "Given by Thomas Sutton, Esq., in memory of his wife",
        "history_note": "",
        "place": "Bilston",
        "production_note": "",
        "historical_context_note": "",
        "collection_code": "FWK"
      }
    }
]
```

SierraLeoneHeritage API

http://sierraleone.heritageinformatics.org provides online collections of Sierra Leonean artefacts exhibited in museums around the UK and the Sierra Leonean National Museum.

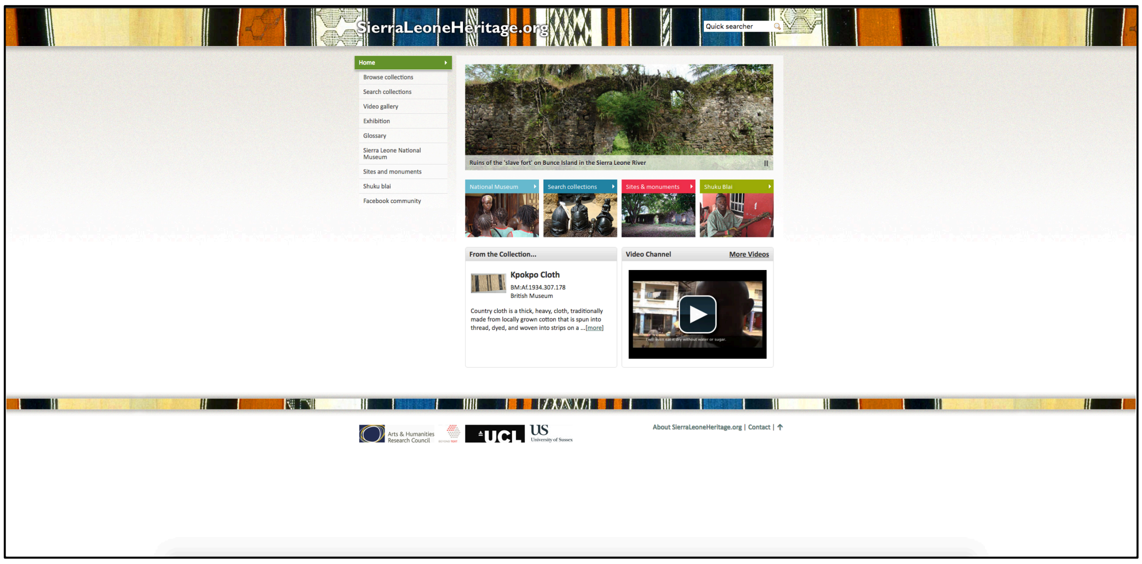
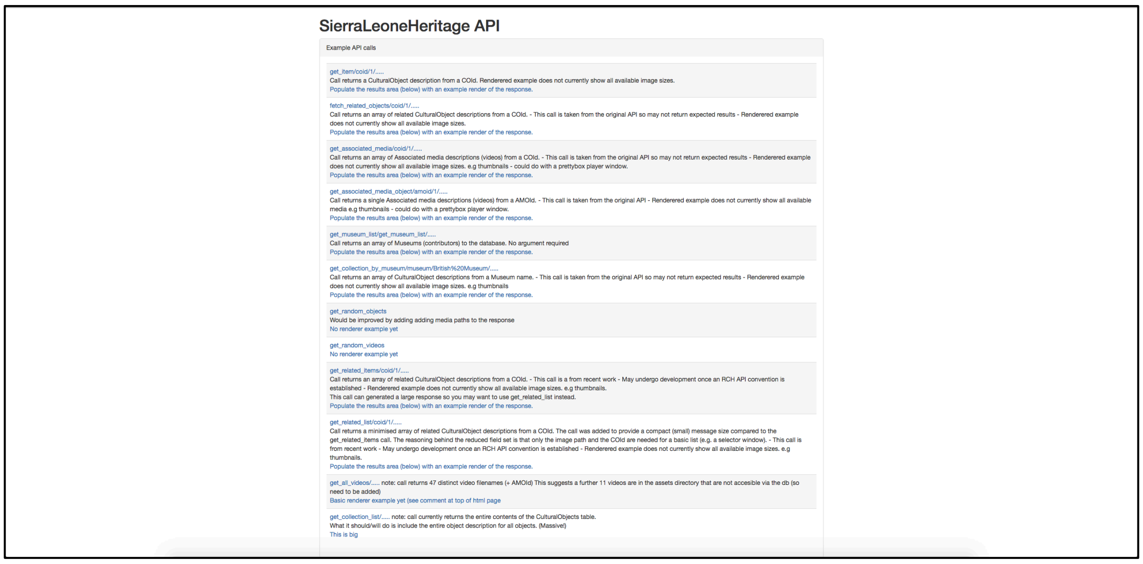


Figure A.5 SierraLeoneHeritage

SierraLeoneHeritage also offer an API called SierraLeoneHeritage API that exposes modules that allow developers to access data and metadata aggregated in the server.



SierraLeoneHeritage API

Example API calls

get_item/coid/1/.....
Call returns a CulturalObject description from a COId. Renderered example does not currently show all available image sizes.
Populate the results area (below) with an example render of the response.

fetch_related_objects/coid/1/.....
Call returns an array of related CulturalObject descriptions from a COId. - This call is taken from the original API so may not return expected results - Renderered example does not currently show all available image sizes.
Populate the results area (below) with an example render of the response.

get_associated_media/coid/1/.....
Call returns an array of Associated media descriptions (videos) from a COId. - This call is taken from the original API so may not return expected results - Renderered example does not currently show all available image sizes. e.g thumbnails - could do with a prettybox player window.
Populate the results area (below) with an example render of the response.

get_associated_media_object/amoid/1/.....
Call returns a single Associated media descriptions (videos) from a AMOId. - This call is taken from the original API - Renderered example does not currently show all available media e.g thumbnails - could do with a prettybox player window.
Populate the results area (below) with an example render of the response.

get_museum_list/get_museum_list/.....
Call returns an array of Museums (contributors) to the database. No argument required
Populate the results area (below) with an example render of the response.

get_collection_by_museum/museum/British%20Museum/.....
Call returns an array of CulturalObject descriptions from a Museum name. - This call is taken from the original API so may not return expected results - Renderered example does not currently show all available image sizes. e.g thumbnails
Populate the results area (below) with an example render of the response.

get_random_objects
Would be improved by adding adding media paths to the response
No renderer example yet

get_random_videos
No renderer example yet

get_related_items/coid/1/.....
Call returns an array of related CulturalObject descriptions from a COId. - This call is a from recent work - May undergo development once an RCH API convention is established - Renderered example does not currently show all available image sizes. e.g thumbnails.
This call can generated a large response so you may want to use get_related_list instead.
Populate the results area (below) with an example render of the response.

get_related_list/coid/1/.....
Call returns a minimised array of related CulturalObject descriptions from a COId. The call was added to provide a compact (small) message size compared to the get_related_items call. The reasoning behind the reduced field set is that only the image path and the COId are needed for a basic list (e.g. a selector window). - This call is from recent work - May undergo development once an RCH API convention is established - Renderered example does not currently show all available image sizes. e.g thumbnails.
Populate the results area (below) with an example render of the response.

get_all_videos/..... note: call returns 47 distinct video filenames (+ AMOId) This suggests a further 11 videos are in the assets directory that are not accesible via the db (so need to be added)
Basic renderer example yet (see comment at top of html page

get_collection_list/..... note: call currently returns the entire contents of the CulturalObjects table.
What it should/will do is include the entire object description for all objects. (Massive!)
This is big

Figure A.6 SierraLeoneHeritage API

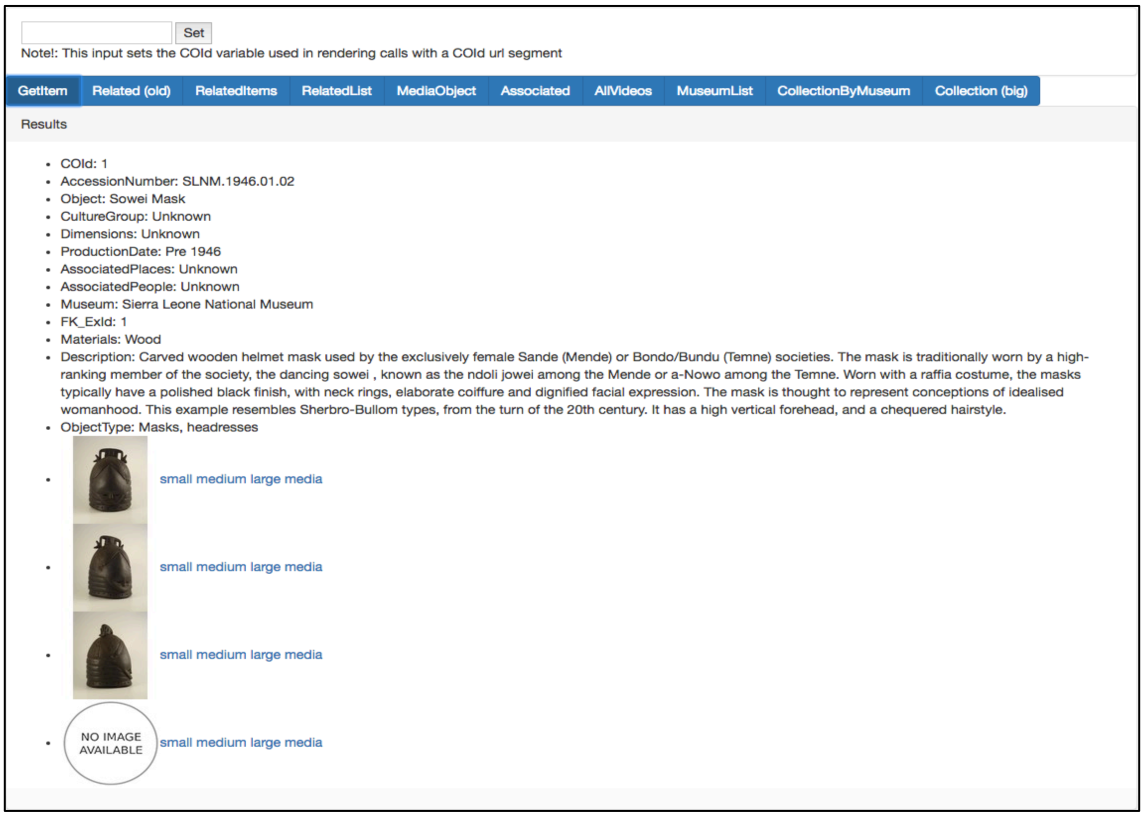
The API provides modules and their URL request that users can change the object ID and the response document format, which could be XML or JSON. An XML response of the get_item module is shown below.

```
<xml>
<status>success</status>
<data>
<COId>1</COId>
<AccessionNumber>SLNM.1946.01.02</AccessionNumber>
<Object>Sowei Mask</Object>
<CultureGroup>Unknown</CultureGroup>
<Dimensions>Unknown</Dimensions>
<ProductionDate>Pre 1946</ProductionDate>
<AssociatedPlaces>Unknown</AssociatedPlaces>
<AssociatedPeople>Unknown</AssociatedPeople>
<Museum>Sierra Leone National Museum</Museum>
<FK_ExId>1</FK_ExId>
<Materials>Wood</Materials>
<Description>
Carved wooden helmet mask used by the exclusively female Sande (Mende) or Bondo/Bundu
(Temne) societies. The mask is traditionally worn by a high-ranking member of the society, the
dancing sowei , known as the ndoli jowei among the Mende or a-Nowo among the Temne.
Worn with a raffia costume, the masks typically have a polished black finish, with neck rings,
elaborate coiffure and dignified facial expression. The mask is thought to represent conceptions
of idealised womanhood. This example resembles Sherbro-Bullom types, from the turn of the
20th century. It has a high vertical forehead, and a chequered hairstyle.
</Description>
<ObjectType>Masks, headresses</ObjectType>
<Media>
<Medium>
<MediaTitle>No Data</MediaTitle>
<MediaDescription>No Data</MediaDescription>
<MediaFileName>SLNM.1946.01.02.pic1</MediaFileName>
<MediaType>Image</MediaType>
<FK_COId>1</FK_COId>
<Media>
<small>
http://sierraleone.heritageinformatics.org/assets/objects/sierra_leone_national_museum/image/t
humbs/small/slnm.1946.01.02.pic1.jpg
</small>
<medium>
http://sierraleone.heritageinformatics.org/assets/objects/sierra_leone_national_museum/image/t
humbs/medium/slnm.1946.01.02.pic1.jpg
</medium>
<large>
http://sierraleone.heritageinformatics.org/assets/objects/sierra_leone_national_museum/image/sl
nm.1946.01.02.pic1.jpg
</large>
<media>
http://sierraleone.heritageinformatics.org/assets/objects/sierra_leone_national_museum/image/sl
nm.1946.01.02.pic1.jpg
</media>
</Media>
```

```
</Medium>
<Medium>
<MediaTitle>No Data</MediaTitle>
<MediaDescription>No Data</MediaDescription>
<MediaFileName>SLNM.1946.01.02.pic2</MediaFileName>
<MediaType>Image</MediaType>
<FK_COId>1</FK_COId>
<Media>
<small>
http://sierraleone.heritageinformatics.org/assets/objects/sierra_leone_national_museum/image/t
humbs/small/slnm.1946.01.02.pic2.jpg
</small>
<medium>
http://sierraleone.heritageinformatics.org/assets/objects/sierra_leone_national_museum/image/t
humbs/medium/slnm.1946.01.02.pic2.jpg
</medium>
<large>
http://sierraleone.heritageinformatics.org/assets/objects/sierra_leone_national_museum/image/sl
nm.1946.01.02.pic2.jpg
</large>
<media>
http://sierraleone.heritageinformatics.org/assets/objects/sierra_leone_national_museum/image/sl
nm.1946.01.02.pic2.jpg
</media>
</Media>
</Medium>
<Medium>
<MediaTitle>No Data</MediaTitle>
<MediaDescription>No Data</MediaDescription>
<MediaFileName>SLNM.1946.01.02.pic3</MediaFileName>
<MediaType>Image</MediaType>
<FK_COId>1</FK_COId>
<Media>
<small>
http://sierraleone.heritageinformatics.org/assets/objects/sierra_leone_national_museum/image/t
humbs/small/slnm.1946.01.02.pic3.jpg
</small>
<medium>
http://sierraleone.heritageinformatics.org/assets/objects/sierra_leone_national_museum/image/t
humbs/medium/slnm.1946.01.02.pic3.jpg
</medium>
<large>
http://sierraleone.heritageinformatics.org/assets/objects/sierra_leone_national_museum/image/sl
nm.1946.01.02.pic3.jpg
</large>
<media>
http://sierraleone.heritageinformatics.org/assets/objects/sierra_leone_national_museum/image/sl
nm.1946.01.02.pic3.jpg
</media>
</Media>
</Medium>
<Medium>
<MediaTitle>No Data</MediaTitle>
<MediaDescription>No Data</MediaDescription>
<MediaFileName>No Data</MediaFileName>
```

```
<MediaType>Image</MediaType>
<FK_COId>1</FK_COId>
<Media>
<small>
http://sierraleone.heritageinformatics.org/assets/objects/all/image/small/no-image.jpg
</small>
<medium>
http://sierraleone.heritageinformatics.org/assets/objects/all/image/medium/no-image.jpg
</medium>
<large>
http://sierraleone.heritageinformatics.org/assets/objects/all/image/large/no-image.jpg
</large>
<media>
http://sierraleone.heritageinformatics.org/assets/objects/all/image/large/no-image.jpg
</media>
</Media>
</Medium>
</Media>
</data>
</xml>
```

The API page also presents the results of the URL request and a web interface that users can see results of each open module.



Note!: This input sets the COId variable used in rendering calls with a COId url segment

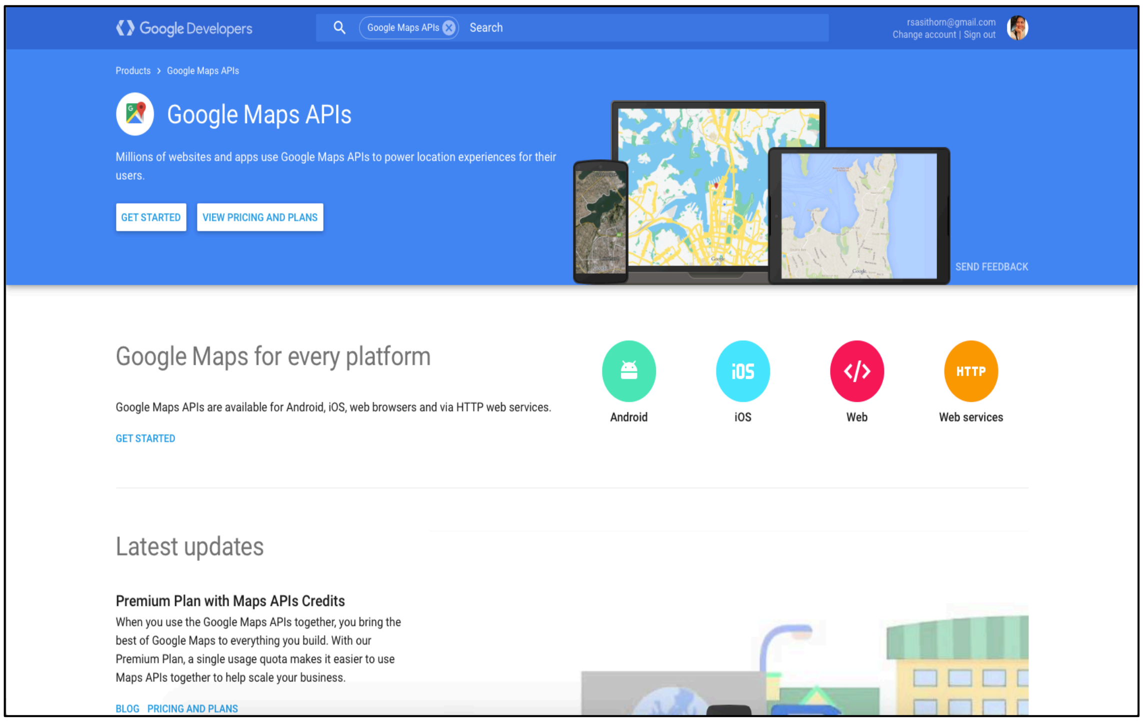
Results

- COId: 1
- AccessionNumber: SLNM.1946.01.02
- Object: Sowei Mask
- CultureGroup: Unknown
- Dimensions: Unknown
- ProductionDate: Pre 1946
- AssociatedPlaces: Unknown
- AssociatedPeople: Unknown
- Museum: Sierra Leone National Museum
- FK_ExId: 1
- Materials: Wood
- Description: Carved wooden helmet mask used by the exclusively female Sande (Mende) or Bondo/Bundu (Temne) societies. The mask is traditionally worn by a high-ranking member of the society, the dancing sowei , known as the ndoli jowei among the Mende or a-Nowo among the Temne. Worn with a raffia costume, the masks typically have a polished black finish, with neck rings, elaborate coiffure and dignified facial expression. The mask is thought to represent conceptions of idealised womanhood. This example resembles Sherbro-Bullom types, from the turn of the 20th century. It has a high vertical forehead, and a chequered hairstyle.
- ObjectType: Masks, headresses
- small medium large media
- small medium large media
- small medium large media
- small medium large media

Figure A.7 The SierraLeoneHeritage result page

Google Maps APIs

Google Maps APIs offer a broad range of service APIs for location, maps and places processing that support client application on every platform. Users have to create an account and register to get a key for a particular API.



Google Maps APIs

Millions of websites and apps use Google Maps APIs to power location experiences for their users.

Google Maps for every platform

Google Maps APIs are available for Android, iOS, web browsers and via HTTP web services.

Latest updates

Premium Plan with Maps APIs Credits

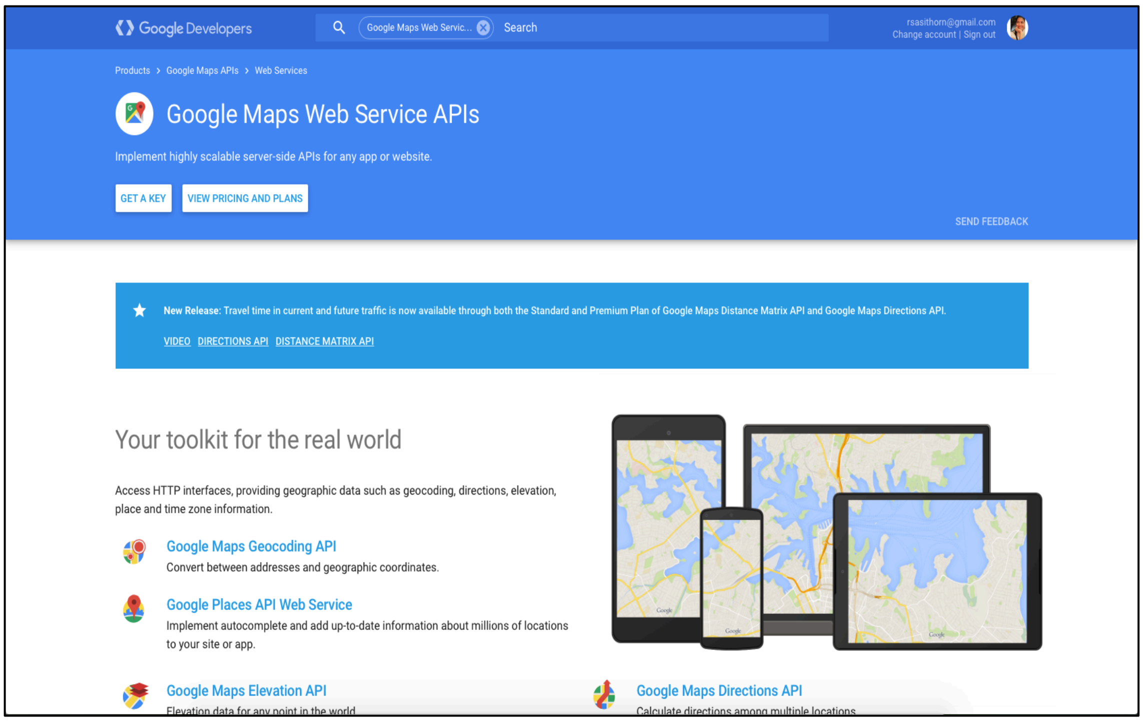
When you use the Google Maps APIs together, you bring the best of Google Maps to everything you build. With our Premium Plan, a single usage quota makes it easier to use Maps APIs together to help scale your business.

Figure A.8 Google Maps APIs

The following describes Google Maps APIs that have been implemented into the SOMARA-based mobile AR application.

Google Places API Web Service

After selecting the HTTP (Web services) button, the system presents a list and details of services available for developers.



Google Maps Web Service APIs

Implement highly scalable server-side APIs for any app or website.

New Release: Travel time in current and future traffic is now available through both the Standard and Premium Plan of Google Maps Distance Matrix API and Google Maps Directions API.

Your toolkit for the real world

Access HTTP interfaces, providing geographic data such as geocoding, directions, elevation, place and time zone information.

Google Maps Geocoding API
Convert between addresses and geographic coordinates.

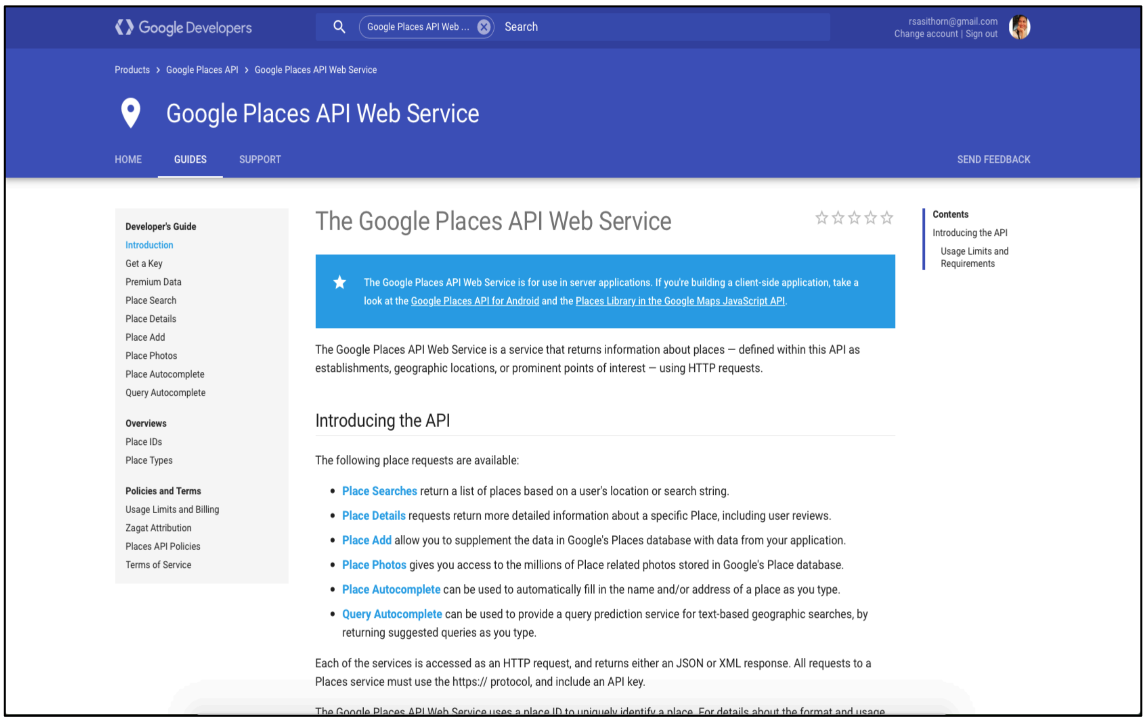
Google Places API Web Service
Implement autocomplete and add up-to-date information about millions of locations to your site or app.

Google Maps Elevation API
Elevation data for any point in the world

Google Maps Directions API
Calculate directions among multiple locations

Figure A.9 Google Maps Web Service APIs

Google Places API Web Service provides processing for application clients that require information of a specific place, a list of places based on a specific location, etc.



Google Places API Web Service

The Google Places API Web Service

The Google Places API Web Service is for use in server applications. If you're building a client-side application, take a look at the Google Places API for Android and the Places Library in the Google Maps JavaScript API.

The Google Places API Web Service is a service that returns information about places — defined within this API as establishments, geographic locations, or prominent points of interest — using HTTP requests.

Introducing the API

The following place requests are available:

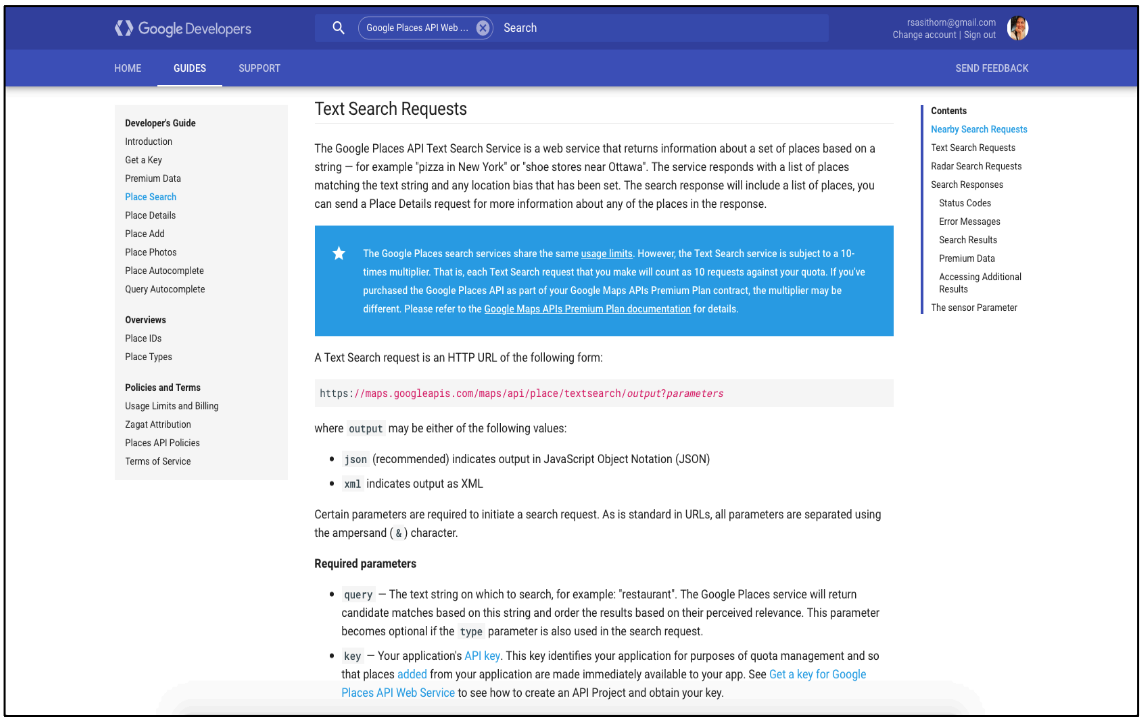
- Place Searches return a list of places based on a user's location or search string.
- Place Details requests return more detailed information about a specific Place, including user reviews.
- Place Add allow you to supplement the data in Google's Places database with data from your application.
- Place Photos gives you access to the millions of Place related photos stored in Google's Place database.
- Place Autocomplete can be used to automatically fill in the name and/or address of a place as you type.
- Query Autocomplete can be used to provide a query prediction service for text-based geographic searches, by returning suggested queries as you type.

Each of the services is accessed as an HTTP request, and returns either an JSON or XML response. All requests to a Places service must use the https:// protocol, and include an API key.

The Google Places API Web Service uses a place ID to uniquely identify a place. For details about the format and usage

Figure A.10 Google Places API Web Services

Place Searches has been implemented into the SOMARA-based mobile AR application in order to acquire the location (latitude, longitude) of a museum selected by a user. The module that provide detailed information of a specific place is Text Search Requests



Text Search Requests

The Google Places API Text Search Service is a web service that returns information about a set of places based on a string — for example "pizza in New York" or "shoe stores near Ottawa". The service responds with a list of places matching the text string and any location bias that has been set. The search response will include a list of places, you can send a Place Details request for more information about any of the places in the response.

The Google Places search services share the same usage limits. However, the Text Search service is subject to a 10-times multiplier. That is, each Text Search request that you make will count as 10 requests against your quota. If you've purchased the Google Places API as part of your Google Maps APIs Premium Plan contract, the multiplier may be different. Please refer to the Google Maps APIs Premium Plan documentation for details.

A Text Search request is an HTTP URL of the following form:

```
https://maps.googleapis.com/maps/api/place/textsearch/output?parameters
```

where `output` may be either of the following values:

- `json` (recommended) indicates output in JavaScript Object Notation (JSON)
- `xml` indicates output as XML

Certain parameters are required to initiate a search request. As is standard in URLs, all parameters are separated using the ampersand (`&`) character.

Required parameters

- `query` — The text string on which to search, for example: "restaurant". The Google Places service will return candidate matches based on this string and order the results based on their perceived relevance. This parameter becomes optional if the `type` parameter is also used in the search request.
- `key` — Your application's API key. This key identifies your application for purposes of quota management and so that places added from your application are made immediately available to your app. See Get a key for Google Places API Web Service to see how to create an API Project and obtain your key.

Figure A.11 Text Search Requests

The create URL used to acquire the location of British Museum is:

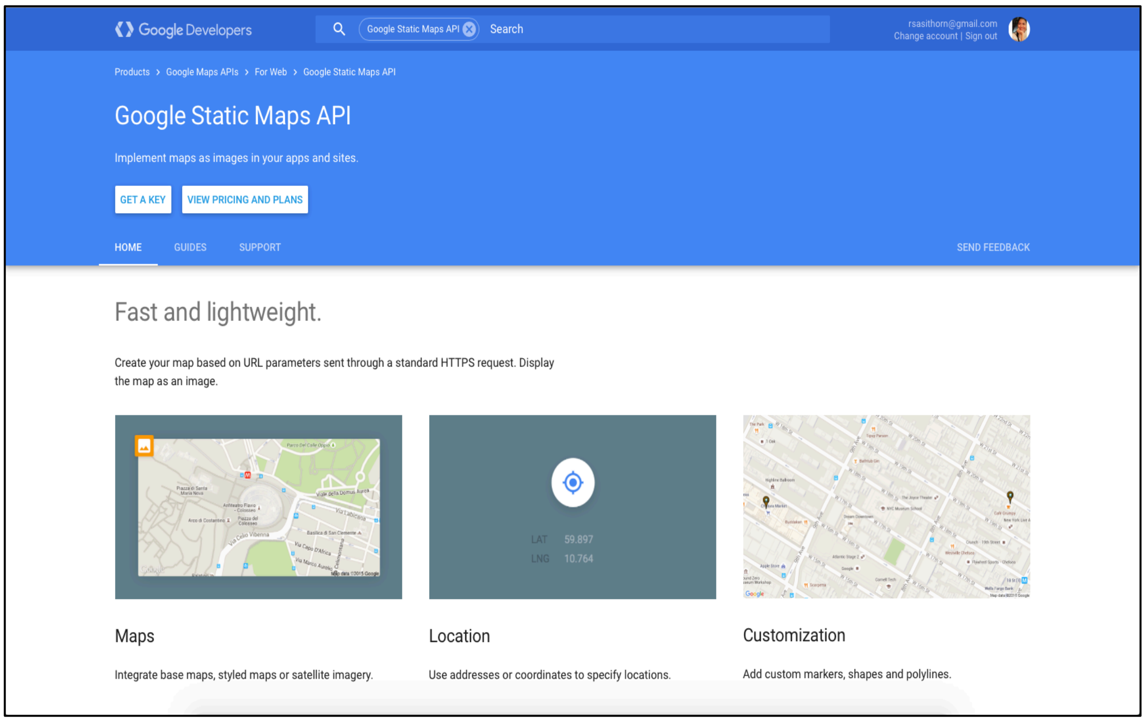
https://maps.googleapis.com/maps/api/place/textsearch/json?query=British%20Museum &key=AIzaSyC3GtYIqNylldA29Enfyg_YM5uSfjnHm94

A JSON response of the service quest above is presented below.

```
{
  "html_attributions" : [],
  "results" : [
    {
      "formatted_address" : "Great Russell St, London WC1B 3DG, United Kingdom",
      "geometry" : {
        "location" : {
          "lat" : 51.5194133,
          "lng" : -0.1269566
        }
      },
      "icon" : "https://maps.gstatic.com/mapfiles/place_api/icons/museum-71.png",
      "id" : "ad6aaec7b7b0fa2c97a127c24845d76135e760ae",
      "name" : "The British Museum",
      "opening_hours" : {
          "open_now" : false,
          "weekday_text" : []
        },
        "photos" : [
          {
            "height" : 1067,
            "html_attributions" : [
              "\u003ca
href=\"https://maps.google.com/maps/contrib/101073730345618386482/photos\"\u003eCarlos
López\u003c/a\u003e"
            ],
            "photo_reference" : "CmRdAAAANFms_W7GBoiodzoFq_j5-
3bK33hEnHBGgQpEJsQQX9UDem-MYfgjOrL8i4ZJNQl1-
ZULQQ_RyBHB5rZ8mix2L98KM6oDHA9DPLHHxkBrqs0tzFMfAzt4rZ9PfhpPQQf7EhAJD
Pj10P4iZ2p7NVWpB2bFGhTKjb93zBGL_kz6R6Fs05M8Lob-yQ",
            "width" : 1600
          }
        ],
        "place_id" : "ChIJB9OTMDIbdkgRp0JWbQGZsS8",
        "rating" : 4.6,
        "reference" : "CnRlAAAAWeVJlBpEKNEbdHjIJwAUVnmP7id9B3REOB-
LMXjWiGLIqx94YdaqfuevL3Sb2LL9oS6xqlAYTWZoZhmmndVBs6UiKJ3UlZ2_QSFxC8G
Xeb-4iuu-
bFJy2uy2Rwvt7Av5Tvp4_9_RttA3mBRxM8soaxIQPA8wPWHEG7Oe5Lp2JNNrGhoUOX7u
Q1ipJmMZaaHtUC0PnjvprfI",
        "types" : [ "museum", "point_of_interest", "establishment" ]
      }
   ],
   "status" : "OK"
}
```

Google Static Maps API

Google Static Maps API provides a static map of a location (latitude, longitude) that can be presented on a website or used by any application. The SOMARA-based Mobile AR application requires a static map to present the position of a selected museum or the origin of a selected object.



Google Static Maps API

Implement maps as images in your apps and sites.

Fast and lightweight.

Create your map based on URL parameters sent through a standard HTTPS request. Display the map as an image.

Maps

Integrate base maps, styled maps or satellite imagery.

Location

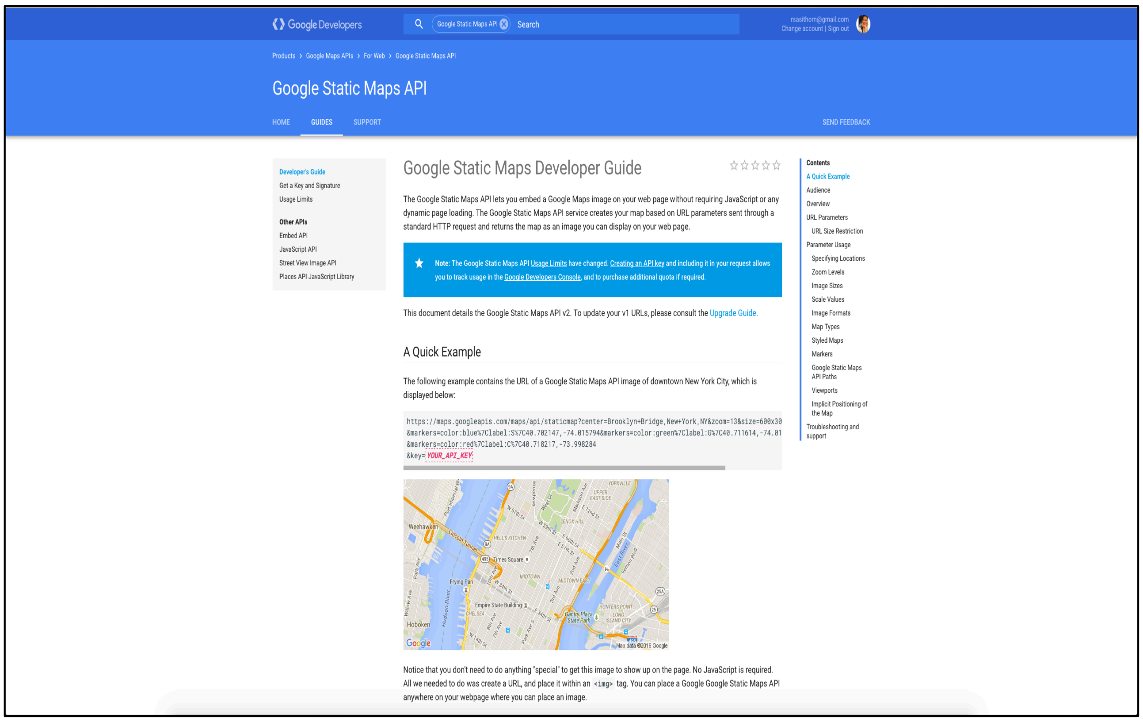
Use addresses or coordinates to specify locations.

Customization

Add custom markers, shapes and polylines.

Figure A.12 Google Static Maps API

The URL request includes the location of a particular place, the size of the map, the marker indicating the position of the place, etc.



Google Static Maps Developer Guide

The Google Static Maps API lets you embed a Google Maps image on your web page without requiring JavaScript or any dynamic page loading. The Google Static Maps API service creates your map based on URL parameters sent through a standard HTTP request and returns the map as an image you can display on your web page.

Note: The Google Static Maps API Usage Limits have changed. Creating an API key and including it in your request allows you to track usage in the Google Developers Console, and to purchase additional quota if required.

This document details the Google Static Maps API v2. To update your v1 URLs, please consult the Upgrade Guide.

A Quick Example

The following example contains the URL of a Google Static Maps API image of downtown New York City, which is displayed below:

```
https://maps.googleapis.com/maps/api/staticmap?center=Brooklyn+Bridge,New+York,NY&zoom=13&size=600x30
&markers=color:blue%7Clabel:S%7C40.702147,-74.015794&markers=color:green%7Clabel:G%7C40.711614,-74.01
&markers=color:red%7Clabel:C%7C40.718217,-73.998284
&key=YOUR_API_KEY
```

Notice that you don't need to do anything "special" to get this image to show up on the page. No JavaScript is required. All we needed to do was create a URL, and place it within an `<img>` tag. You can place a Google Google Static Maps API anywhere on your webpage where you can place an image.

Figure A.13 The Google Static Maps URL request

A service request for a static map that presents the location of British Museum in the center is:

https://maps.googleapis.com/maps/api/staticmap?center=51.5194133,-0.1269566&zoom=13&size=600x600&maptype=roadmap&markers=color:red|label:S|51.5194133,-0.1269566&key=AIzaSyDpBmBxxtpSrDyGBKOPR1glbefdgjH_Q7E

The location of British Museum is Latitude: 51.5194133, Longitude: -0.1269566.

B. Tracking Configuration for Home-based Learning Scenario

```
<?xml version="1.0" encoding="UTF-8"?>
<TrackingData>
  <Sensors>
    <Sensor Type="FeatureBasedSensorSource" Subtype="ML3D">
      <SensorID>FeatureBasedSensorSource_0</SensorID>
      <Parameters>
        <featureorientationassignment>gravity</featureorientationassignment>
        <MaxObjectsToDetectPerFrame>5</MaxObjectsToDetectPerFrame>
        <MaxObjectsToTrackInParallel>1</MaxObjectsToTrackInParallel>
      </Parameters>
      <SensorCOS>
        <SensorCosID>fb23be3415c00141a340d36b2766f610</SensorCosID>
        <parameters>
          <numextensiblefeatures>0</numextensiblefeatures>
          <mintriangulationangle>6</mintriangulationangle>
          <map>fb23be3415c00141a340d36b2766f610.f3b</map>
          <MinMatches>15</MinMatches>
          <NumExtensibleFeatures>250</NumExtensibleFeatures>
        </parameters>
      </SensorCOS>
      <SensorCOS>
        <SensorCosID>88320ad8a0b8d7fb9ccdd466217b1d69</SensorCosID>
        <parameters>
          <numextensiblefeatures>0</numextensiblefeatures>
          <mintriangulationangle>6</mintriangulationangle>
          <map>88320ad8a0b8d7fb9ccdd466217b1d69.f3b</map>
          <MinMatches>15</MinMatches>
          <NumExtensibleFeatures>250</NumExtensibleFeatures>
        </parameters>
      </SensorCOS>
    </Sensor>
  </Sensors>
  <Connections>
    <COS>
      <Name>cos1</Name>
      <Fuser Type="SmoothingFuser">
        <Parameters>
          <AlphaRotation>0.8</AlphaRotation>
          <AlphaTranslation>1.0</AlphaTranslation>
          <GammaRotation>0.8</GammaRotation>
          <GammaTranslation>1.0</GammaTranslation>
          <KeepPoseForNumberOfFrames>0</KeepPoseForNumberOfFrames>
        </Parameters>
      </Fuser>
      <SensorSource>
        <SensorID>FeatureBasedSensorSource_0</SensorID>
        <SensorCosID>fb23be3415c00141a340d36b2766f610</SensorCosID>
        <HandEyeCalibration>
          <TranslationOffset>
            <x>0.0</x>
            <y>0.0</y>
            <z>0.0</z>
          </TranslationOffset>
          <RotationOffset>
            <x>0.0</x>
            <y>0.0</y>
            <z>0.0</z>
            <w>1.0</w>
          </RotationOffset>
        </HandEyeCalibration>
        <COSOffset>
          <TranslationOffset>
            <x>0.0</x>
            <y>0.0</y>
            <z>0.0</z>
          </TranslationOffset>
          <RotationOffset>
            <x>0.0</x>
            <y>0.0</y>
            <z>0.0</z>
            <w>1.0</w>
          </RotationOffset>
        </COSOffset>
      </SensorSource>
    </COS>
    <COS>
      <Name>cos2</Name>
      <Fuser Type="SmoothingFuser">
        <Parameters>
          <AlphaRotation>0.8</AlphaRotation>
          <AlphaTranslation>1.0</AlphaTranslation>
          <GammaRotation>0.8</GammaRotation>
          <GammaTranslation>1.0</GammaTranslation>
          <KeepPoseForNumberOfFrames>0</KeepPoseForNumberOfFrames>
        </Parameters>
      </Fuser>
      <SensorSource>
        <SensorID>FeatureBasedSensorSource_0</SensorID>
        <SensorCosID>88320ad8a0b8d7fb9ccdd466217b1d69</SensorCosID>
        <HandEyeCalibration>
          <TranslationOffset>
            <x>0.0</x>
            <y>0.0</y>
            <z>0.0</z>
          </TranslationOffset>
          <RotationOffset>
            <x>0.0</x>
            <y>0.0</y>
            <z>0.0</z>
            <w>1.0</w>
          </RotationOffset>
        </HandEyeCalibration>
        <COSOffset>
          <TranslationOffset>
            <x>0.0</x>
            <y>0.0</y>
            <z>0.0</z>
          </TranslationOffset>
          <RotationOffset>
            <x>0.0</x>
```

```
                        <y>0.0</y>
                        <z>0.0</z>
                        <w>1.0</w>
                    </RotationOffset>
                </COSOffset>
            </SensorSource>
        </COS>
    </Connections>
</TrackingData>
```

C. Tracking Configuration for Home-based Learning Scenario

```
<?xml version="1.0"?>
<TrackingData>
   <Sensors>
          <Sensor Type="FeatureBasedSensorSource" Subtype="Fast">
                    <SensorID>FeatureTracking1</SensorID>
                    <Parameters>
            <FeatureDescriptorAlignment>regular</FeatureDescriptorAlignment>
            <MaxObjectsToDetectPerFrame>5</MaxObjectsToDetectPerFrame>
            <MaxObjectsToTrackInParallel>1</MaxObjectsToTrackInParallel>
            <SimilarityThreshold>0.7</SimilarityThreshold>
                    </Parameters>
                    <SensorCOS>
                              <SensorCosID>Patch1</SensorCosID>
                              <Parameters>
                                        <ReferenceImage>Nomoli.jpg</ReferenceImage>
                                        <SimilarityThreshold>0.7</SimilarityThreshold>
                              </Parameters>
                    </SensorCOS>
                    <SensorCOS>
                              <SensorCosID>Patch2</SensorCosID>
                              <Parameters>
                                        <ReferenceImage>Nomoli1.jpg</ReferenceImage>
                                        <SimilarityThreshold>0.8</SimilarityThreshold>
                              </Parameters>
                    </SensorCOS>
         <SensorCOS>
            <SensorCosID>Patch3</SensorCosID>
            <Parameters>
               <ReferenceImage>bowl1.jpg</ReferenceImage>
               <SimilarityThreshold>0.8</SimilarityThreshold>
            </Parameters>
         </SensorCOS>
         <SensorCOS>
            <SensorCosID>Patch4</SensorCosID>
            <Parameters>
               <ReferenceImage>render.jpg</ReferenceImage>
               <SimilarityThreshold>0.8</SimilarityThreshold>
            </Parameters>
         </SensorCOS>
      </Sensor>
        </Sensors>
   <Connections>
          <COS>
            <Name>cos1</Name>
            <Fuser Type="SmoothingFuser">
                    <Parameters>
                              <KeepPoseForNumberOfFrames>2</KeepPoseForNumberOfFrames>
                              <GravityAssistance></GravityAssistance>
                              <AlphaTranslation>0.8</AlphaTranslation>
                              <GammaTranslation>0.8</GammaTranslation>
                              <AlphaRotation>0.5</AlphaRotation>
                              <GammaRotation>0.5</GammaRotation>
        <ContinueLostTrackingWithOrientationSensor>false</ContinueLostTrackingWithOrient
         ationSensor>
                    </Parameters>
```

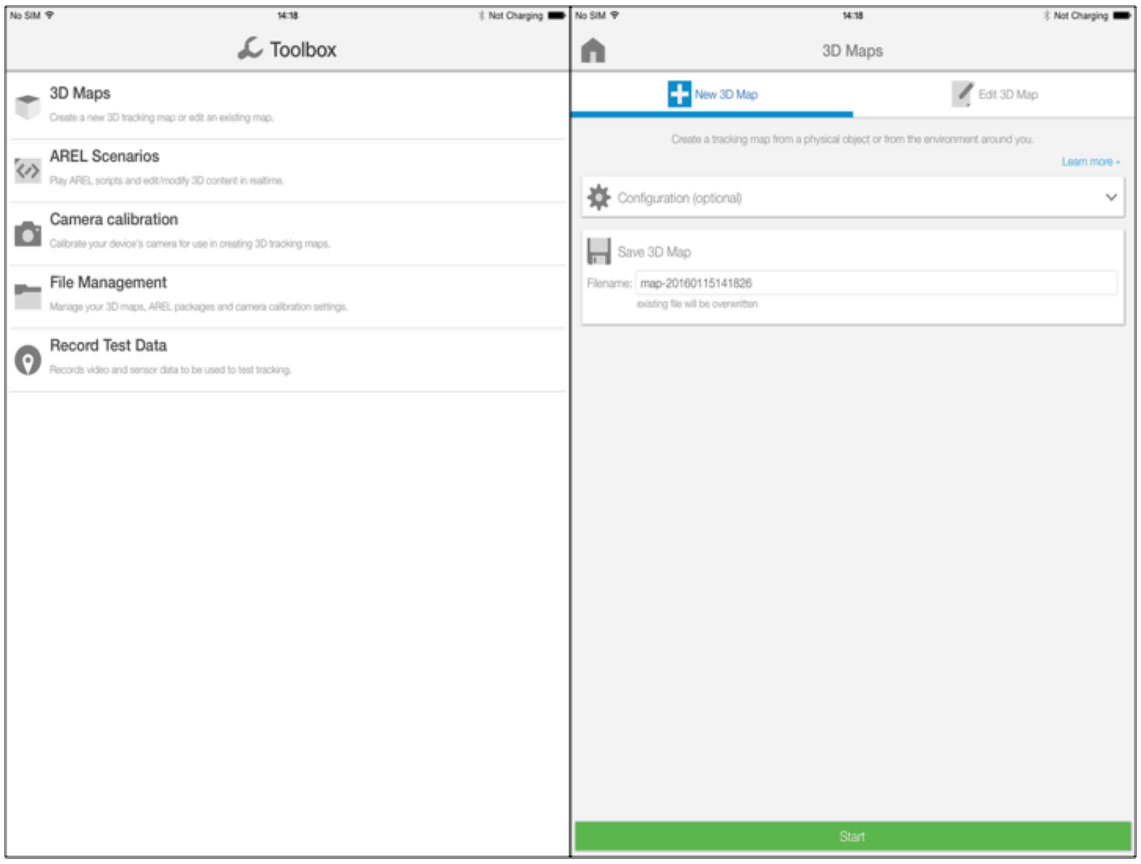
```
                </Fuser>
                <SensorSource>
                        <SensorID>FeatureTracking1</SensorID>
                        <SensorCosID>Patch1</SensorCosID>
                        <HandEyeCalibration>
                                <TranslationOffset>
                                        <X>0</X>
                                        <Y>0</Y>
                                        <Z>0</Z>
                                </TranslationOffset>
                                <RotationOffset>
                                        <X>0</X>
                                        <Y>0</Y>
                                        <Z>0</Z>
                                        <W>1</W>
                                </RotationOffset>
                        </HandEyeCalibration>
                        <COSOffset>
                                <TranslationOffset>
                                        <X>0</X>
                                        <Y>0</Y>
                                        <Z>0</Z>
                                </TranslationOffset>
                                <RotationOffset>
                                        <X>0</X>
                                        <Y>0</Y>
                                        <Z>0</Z>
                                        <W>1</W>
                                </RotationOffset>
                        </COSOffset>
                </SensorSource>
        </COS>
        <COS>
                <Name>cos2</Name>
                <Fuser Type="BestQualityFuser">
                        <Parameters>
                        <KeepPoseForNumberOfFrames>2</KeepPoseForNumberOfFrames>
                                <GravityAssistance></GravityAssistance>
                                <AlphaTranslation>0.8</AlphaTranslation>
                                <GammaTranslation>0.8</GammaTranslation>
                                <AlphaRotation>0.5</AlphaRotation>
                                <GammaRotation>0.5</GammaRotation>
<ContinueLostTrackingWithOrientationSensor>false</ContinueLostTrackingWithOrient
ationSensor>
                        </Parameters>
                </Fuser>
                <SensorSource>
                        <SensorID>FeatureTracking1</SensorID>
                        <SensorCosID>Patch2</SensorCosID>
                        <HandEyeCalibration>
                                <TranslationOffset>
                                        <X>0</X>
                                        <Y>0</Y>
                                        <Z>0</Z>
                                </TranslationOffset>
```

```
                                    <RotationOffset>
                                            <X>0</X>
                                            <Y>0</Y>
                                            <Z>0</Z>
                                            <W>1</W>
                                    </RotationOffset>
                            </HandEyeCalibration>
                            <COSOffset>
                                    <TranslationOffset>
                                            <X>0</X>
                                            <Y>0</Y>
                                            <Z>0</Z>
                                    </TranslationOffset>
                                    <RotationOffset>
                                            <X>0</X>
                                            <Y>0</Y>
                                            <Z>0</Z>
                                            <W>1</W>
                                    </RotationOffset>
                            </COSOffset>
                    </SensorSource>
         </COS>
         <COS>
        <Name>cos3</Name>
        <Fuser Type="BestQualityFuser">
           <Parameters>
              <KeepPoseForNumberOfFrames>2</KeepPoseForNumberOfFrames>
              <GravityAssistance></GravityAssistance>
              <AlphaTranslation>0.8</AlphaTranslation>
              <GammaTranslation>0.8</GammaTranslation>
              <AlphaRotation>0.5</AlphaRotation>
              <GammaRotation>0.5</GammaRotation>
<ContinueLostTrackingWithOrientationSensor>false</ContinueLostTrackingWithOrientationS
ensor>
           </Parameters>
        </Fuser>
        <SensorSource>
           <SensorID>FeatureTracking1</SensorID>
           <SensorCosID>Patch3</SensorCosID>
           <HandEyeCalibration>
              <TranslationOffset>
                 <X>0</X>
                 <Y>0</Y>
                 <Z>0</Z>
              </TranslationOffset>
              <RotationOffset>
                 <X>0</X>
                 <Y>0</Y>
                 <Z>0</Z>
                 <W>1</W>
              </RotationOffset>
           </HandEyeCalibration>
           <COSOffset>
              <TranslationOffset>
                 <X>0</X>
```

```
                  <Y>0</Y>
                  <Z>0</Z>
               </TranslationOffset>
               <RotationOffset>
                  <X>0</X>
                  <Y>0</Y>
                  <Z>0</Z>
                  <W>1</W>
               </RotationOffset>
            </COSOffset>
         </SensorSource>
      </COS>
      <COS>
         <Name>cos4</Name>
         <Fuser Type="BestQualityFuser">
            <Parameters>
               <KeepPoseForNumberOfFrames>2</KeepPoseForNumberOfFrames>
               <GravityAssistance></GravityAssistance>
               <AlphaTranslation>0.8</AlphaTranslation>
               <GammaTranslation>0.8</GammaTranslation>
               <AlphaRotation>0.5</AlphaRotation>
               <GammaRotation>0.5</GammaRotation>
<ContinueLostTrackingWithOrientationSensor>false</ContinueLostTrackingWithOrientationS
ensor>
            </Parameters>
         </Fuser>
         <SensorSource>
            <SensorID>FeatureTracking1</SensorID>
            <SensorCosID>Patch4</SensorCosID>
            <HandEyeCalibration>
               <TranslationOffset>
                  <X>0</X>
                  <Y>0</Y>
                  <Z>0</Z>
               </TranslationOffset>
               <RotationOffset>
                  <X>0</X>
                  <Y>0</Y>
                  <Z>0</Z>
                  <W>1</W>
               </RotationOffset>
            </HandEyeCalibration>
            <COSOffset>
               <TranslationOffset>
                  <X>0</X>
                  <Y>0</Y>
                  <Z>0</Z>
               </TranslationOffset>
               <RotationOffset>
                  <X>0</X>
                  <Y>0</Y>
                  <Z>0</Z>
                  <W>1</W>
               </RotationOffset>
            </COSOffset>
            </SensorSource>
        </COS>
          </Connections>
</TrackingData>
```

D. The 3D Maps of Reference Objects

Figure D.1 presents the Metaio Toolbox on iPad used to create a map of a physical reference object. Users select the 3D Maps icon in order to create a new 3D tracking map of an object.



3D Maps
Create a new 3D tracking map or edit an existing map.

AREL Scenarios
Play AREL scripts and edit/modify 3D content in realtime.

Camera calibration
Calibrate your device's camera for use in creating 3D tracking maps.

File Management
Manage your 3D maps, AREL packages and camera calibration settings.

Record Test Data
Records video and sensor data to be used to test tracking.

Create a tracking map from a physical object or from the environment around you.

Filename: map-20160115141826
existing file will be overwritten

Figure D.1 Metaio Toolbox on iOS

After the Start button is touched, the application shows a camera view that allow users to point the camera on the mobile device at the targeted object and move around in order to create a good quality map of the object. Figure D.2 illustrates the application creating a 3D map of a sample object. The application learns the environment and then tracks the features of the object that can be seen on the camera view.

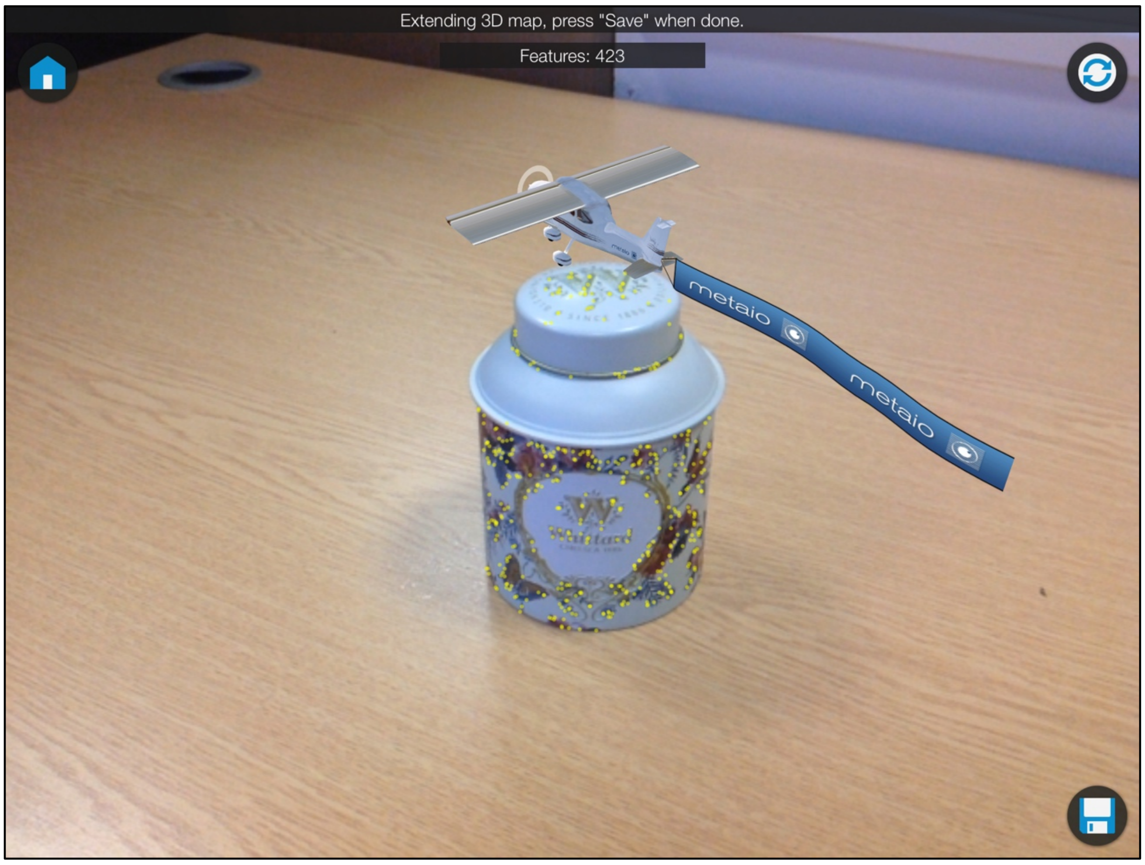
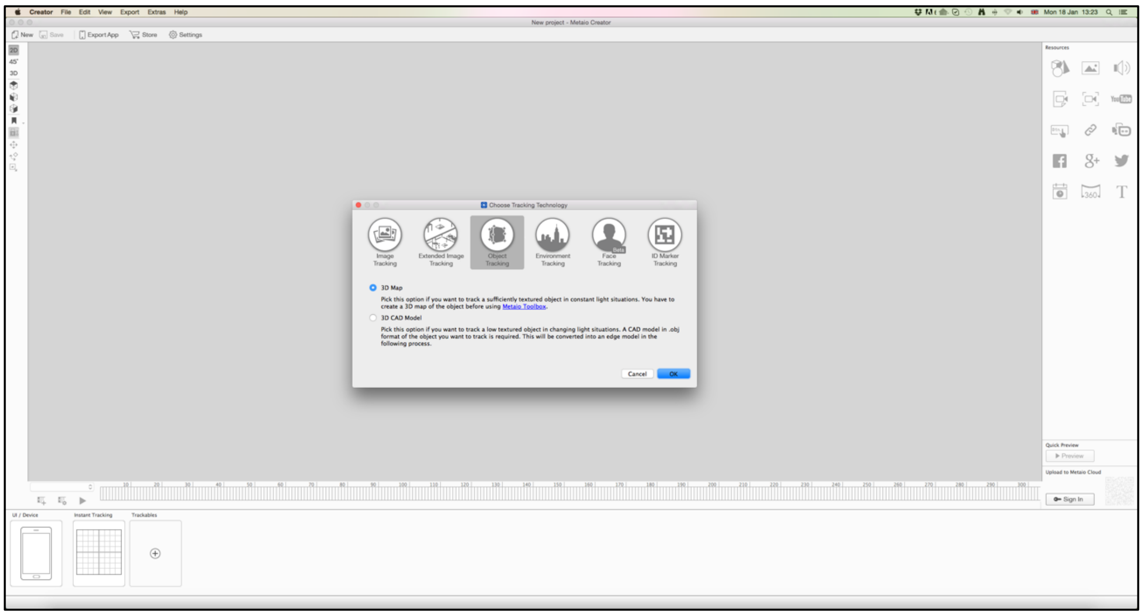


Figure D.2 The created map of a sample object

The 3D map can be saved to the mobile device and then transferred to Metaio Creator running on a PC or MAC where the map file is loaded and exported to two separate files including a map file and a tracking configuration file in XML format. The tracking configuration file and 3D map file are specifically created for a particular reference object and these need to be stored in the library of the novel mobile AR application. Figure D.3 shows Creator used to create an AR environment of a reference object that could be a physical object, image, environment, human face or marker. The process starts by selecting a 3D map file of a reference object. For physical object tracking, developers have to import or open a 3D map of the reference object that has been created by Metaio Toolbox.



[✓ 3D Map]

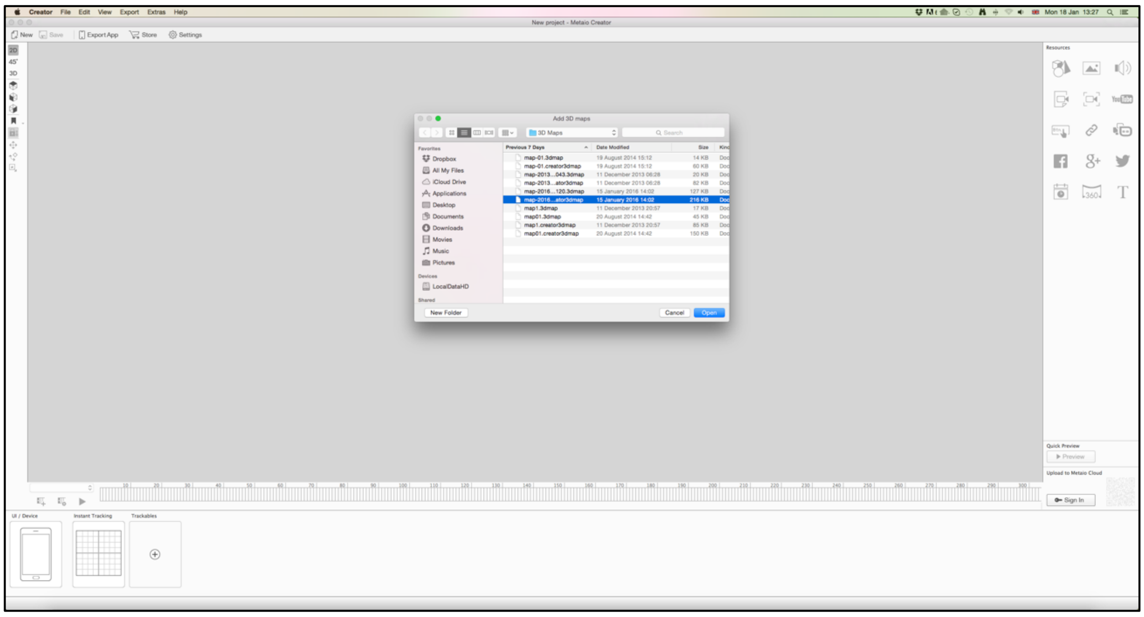
Pick this option if you want to track a sufficiently textured object in constant light situations. You have to create a 3D map of the object before using Metaio Toolbox.

3D CAD Model

Pick this option if you want to track a low textured object in changing light situations. A CAD model in .obj format of the object you want to track is required. This will be converted into an edge model in the following process.

Figure D.3 Metaio Creator on Mac

The saved 3D map file transferred from the mobile device is opened in Creator, presenting a 3D map of the physical object. Figure D.4 shows the selected 3D map file created and saved by Metaio Toolbox that is loaded into the Creator.



Previous 7 Days	Date Modified	Size	Kind
map-01.3dmap	19 August 2014 15:12	14 KB	Doc
map-01.creator3dmap	19 August 2014 15:12	60 KB	Doc
map-2013...043.3dmap	11 December 2013 06:28	20 KB	Doc
map-2013...ator3dmap	11 December 2013 06:28	82 KB	Doc
map-2016...120.3dmap	15 January 2016 14:02	127 KB	Doc
map-2016...ator3dmap	15 January 2016 14:02	216 KB	Doc
map1.3dmap	11 December 2013 20:57	17 KB	Doc
map01.3dmap	20 August 2014 14:42	45 KB	Doc
map1.creator3dmap	11 December 2013 20:57	85 KB	Doc
map01.creator3dmap	20 August 2014 14:42	150 KB	Doc

Figure D.4 A selected 3D map file of a sample object

Figure D.5 shows the 3D map of a sample object on the Creator. The 3D map is exported to a tracking configuration file and a tracking map.

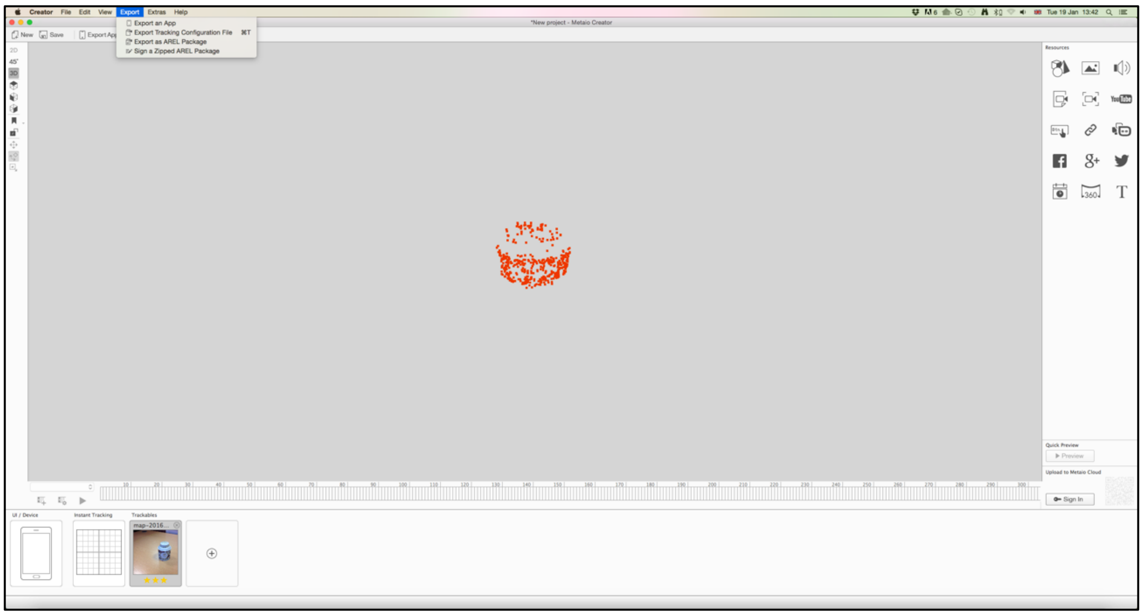


Figure D.5 The 3D map of a sample object

Figure D.6 presents the 3D map exported to a tracking configuration file and a 3D tracking map that is saved and copied into the iOS project for building marker-less tracking components.

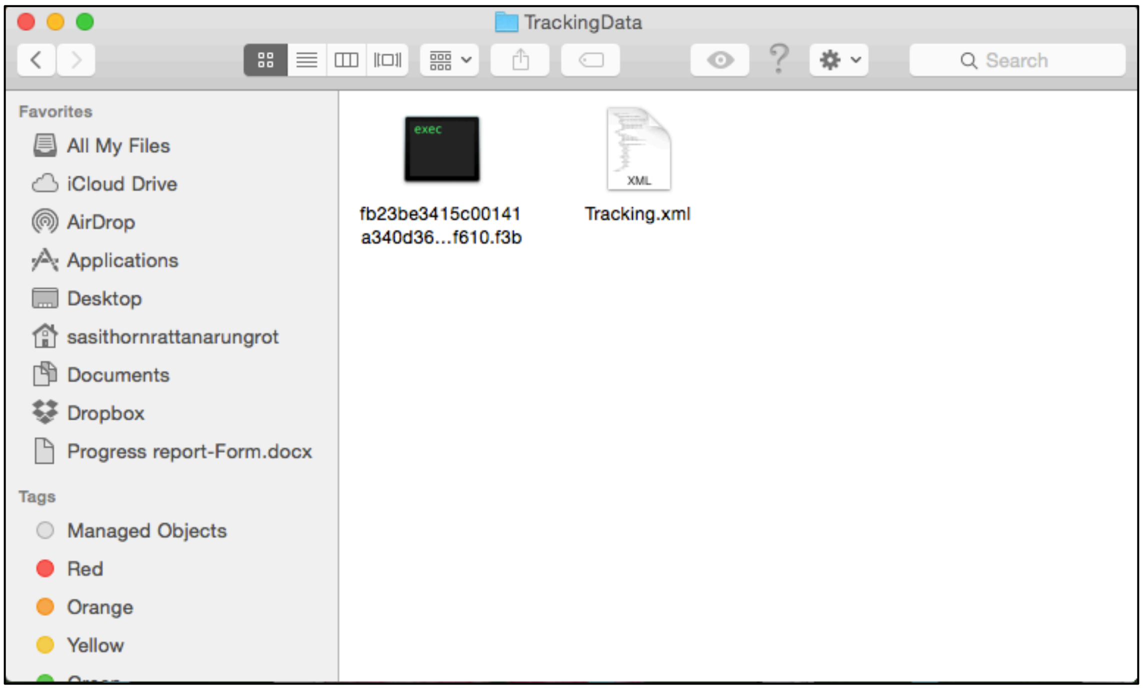


Figure D.6 The tracking configuration files exported from a 3D map

The figures below present the map file of another reference object installed into the mobile AR client for the museum-based learning scenario.

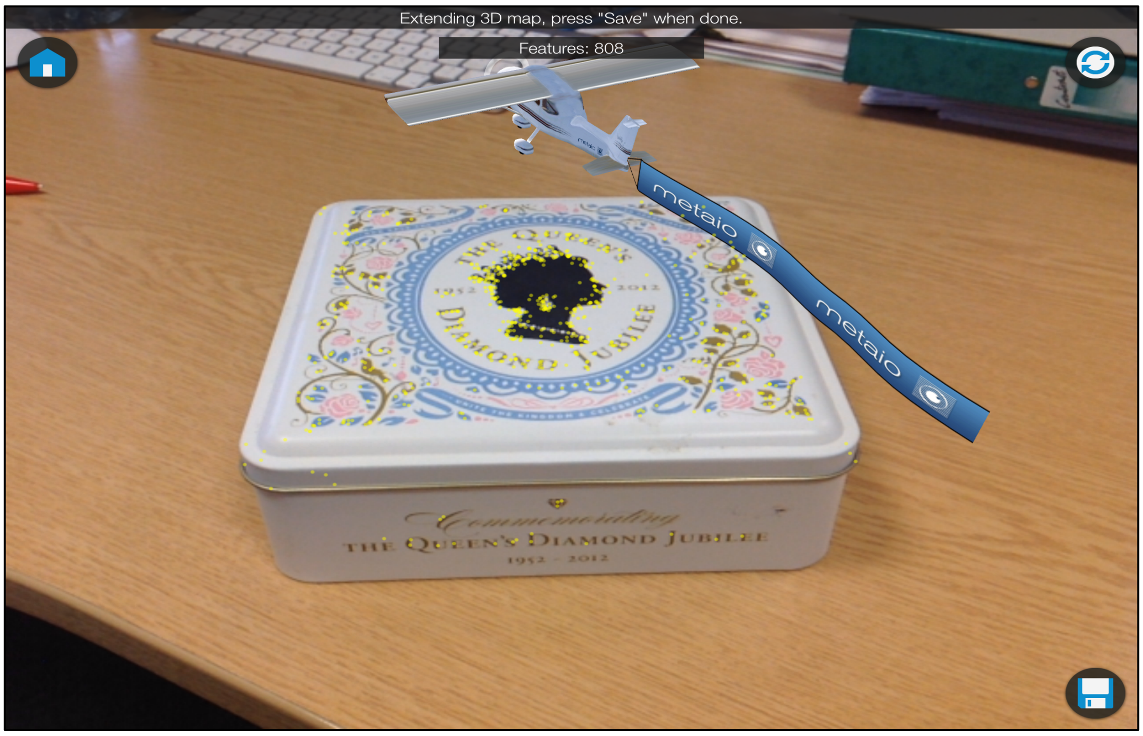


Figure D.7 The 3D map of a sample object

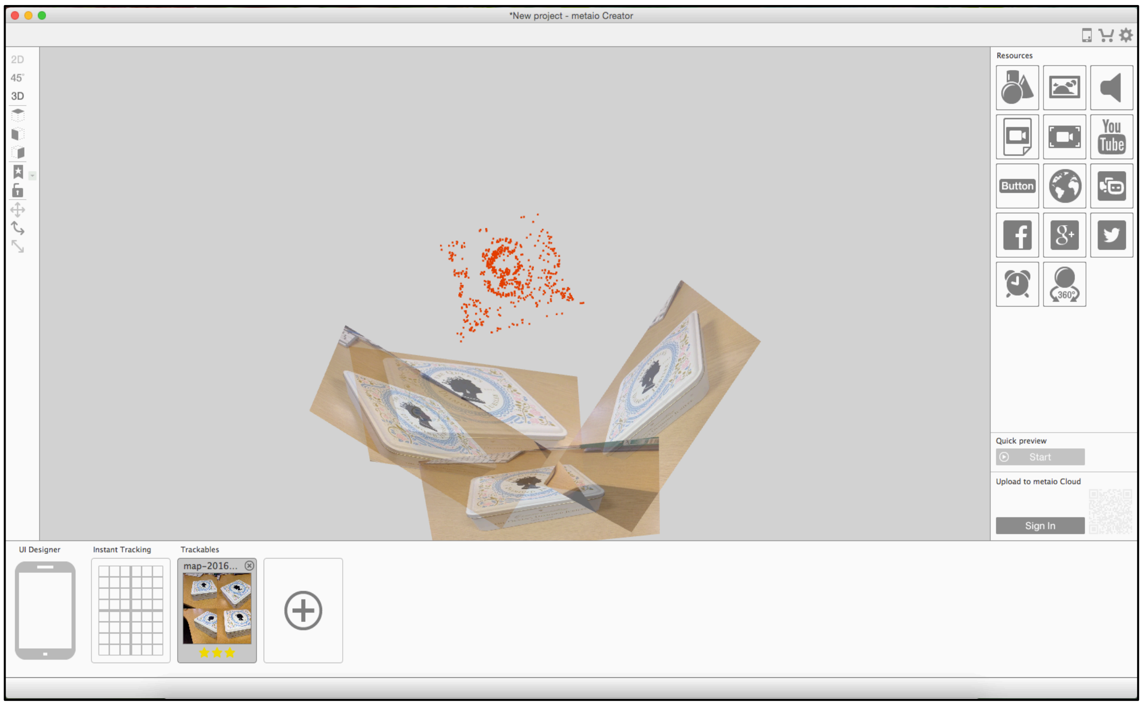


Figure D.8 The 3D map on Metaio Creator

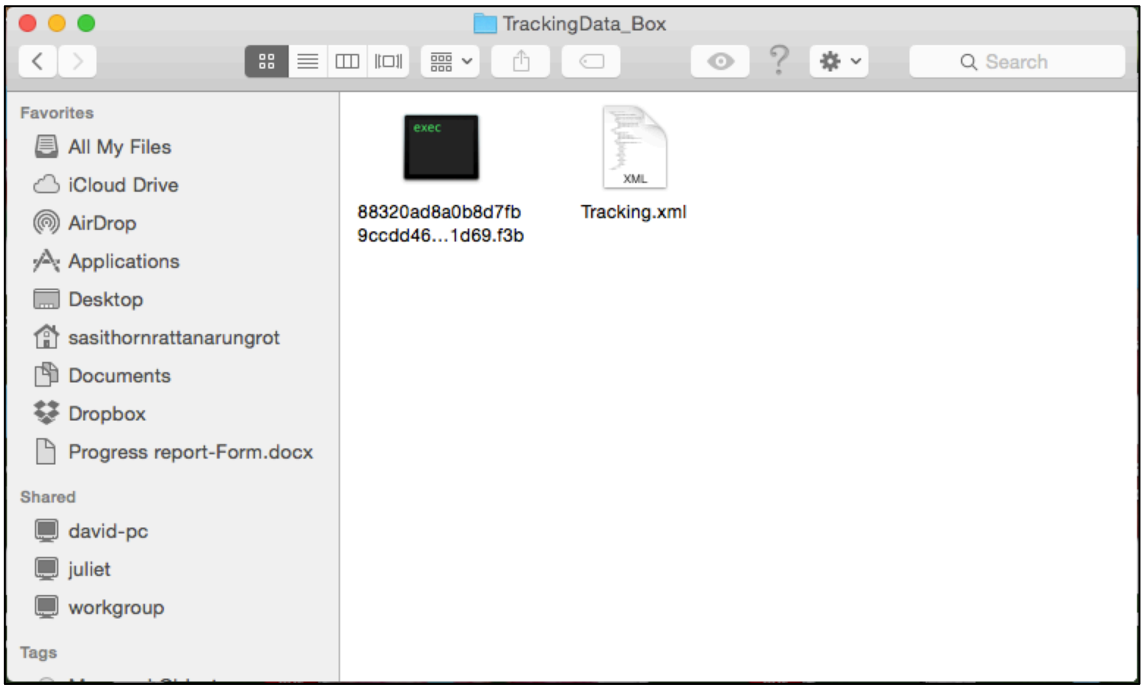


Figure D.9 The 3D map file and tracking configuration file

E. Installing the Metaio Augmented Reality SDK

In order to build a mobile AR application, the development framework requires an AR SDK for performing AR tasks including object tracking, recognition and content visualization. In this research, the mobile AR application uses Metaio's native AR SDK that can be downloaded from the Metaio website. After installing the SDK into the computer and opening the SDK folder, the developer sees the existing libraries for the development frameworks it supports. Figure E.1 presents the provided libraries and frameworks for developing environments including Android, iOS and Unity.

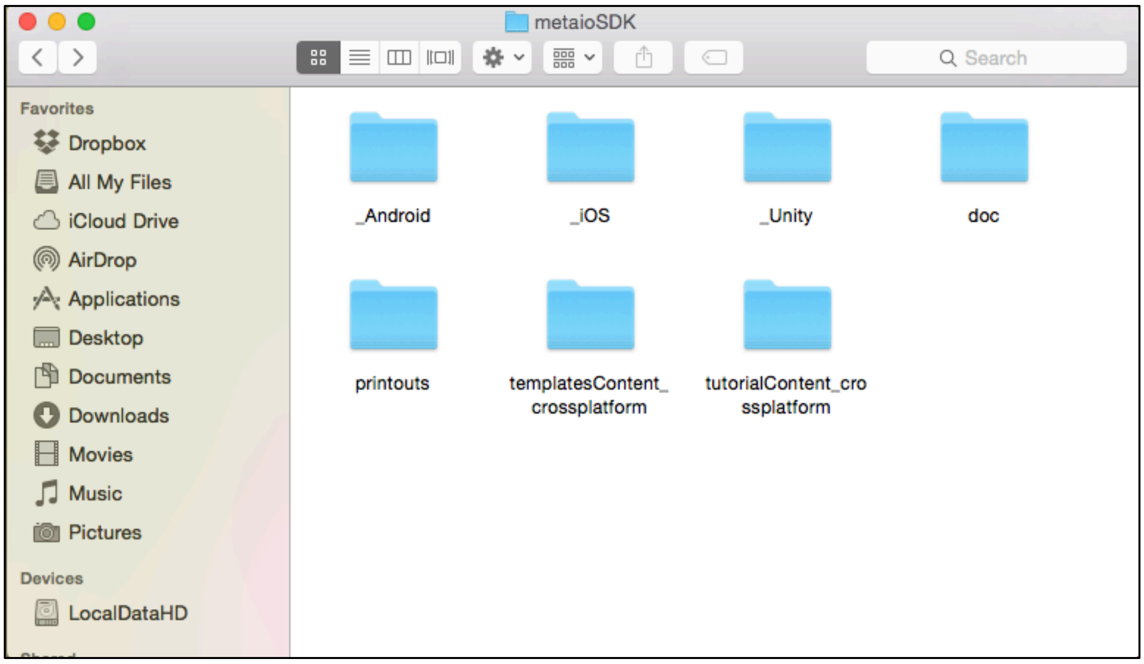


Figure E.1 The Metaio development frameworks

Developers are able to choose the SDK specifically created for each platform in order to build a mobile AR application based on their needs. Figure E.2 shows the components for developing a mobile AR application on iOS SDK.

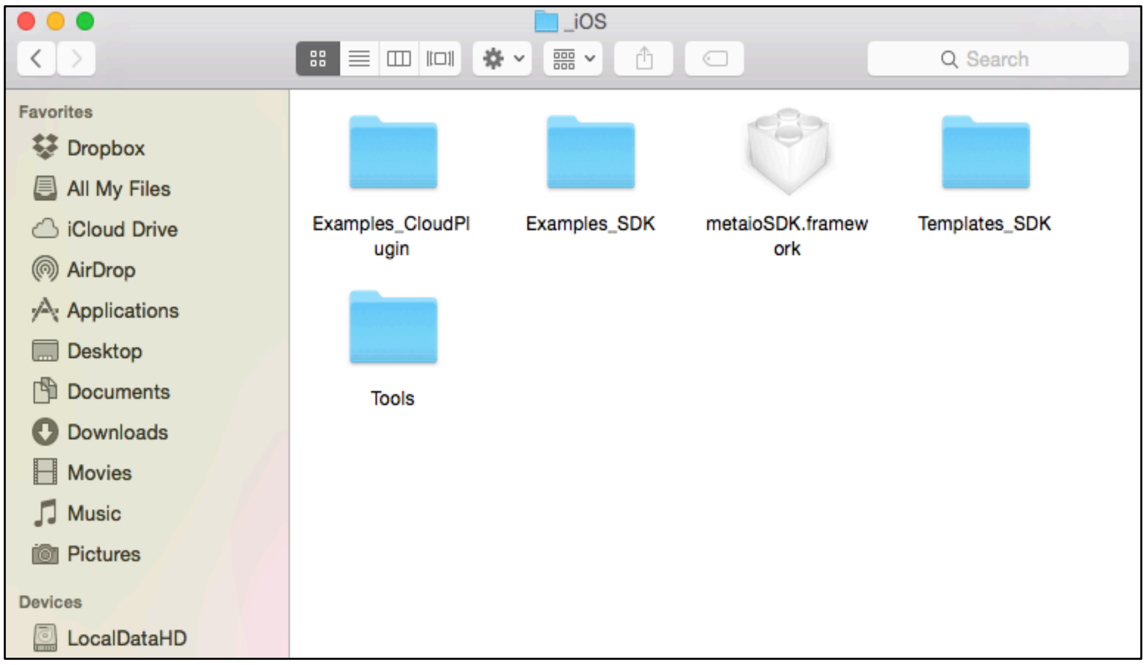


Figure E.2 iOS development framework

After creating a new project in XCode, the metaioSDK.framework file has to be copied and pasted into the folder of the created mobile AR application. Figure E.3 shows the folder of the application and development environment inside including the metaioSDK.framework.

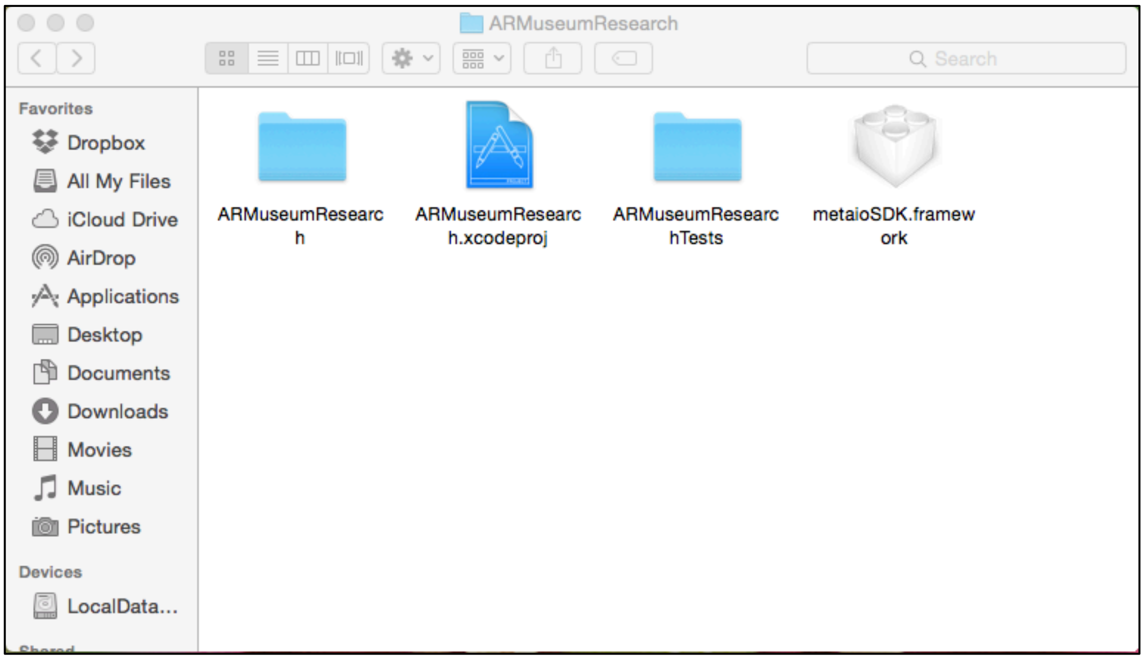
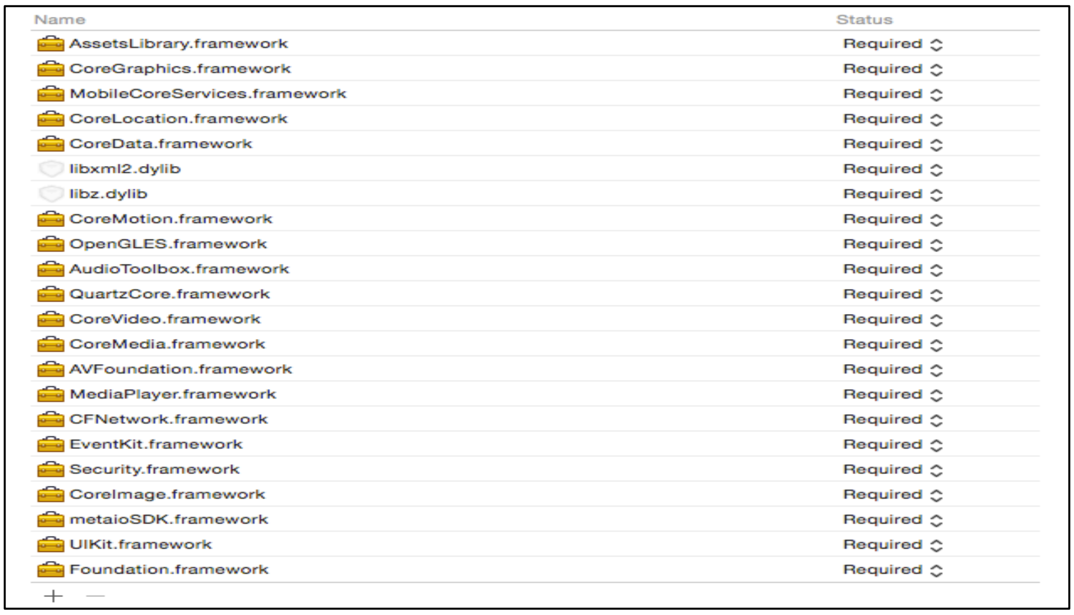


Figure E.3 The mobile AR application development environment

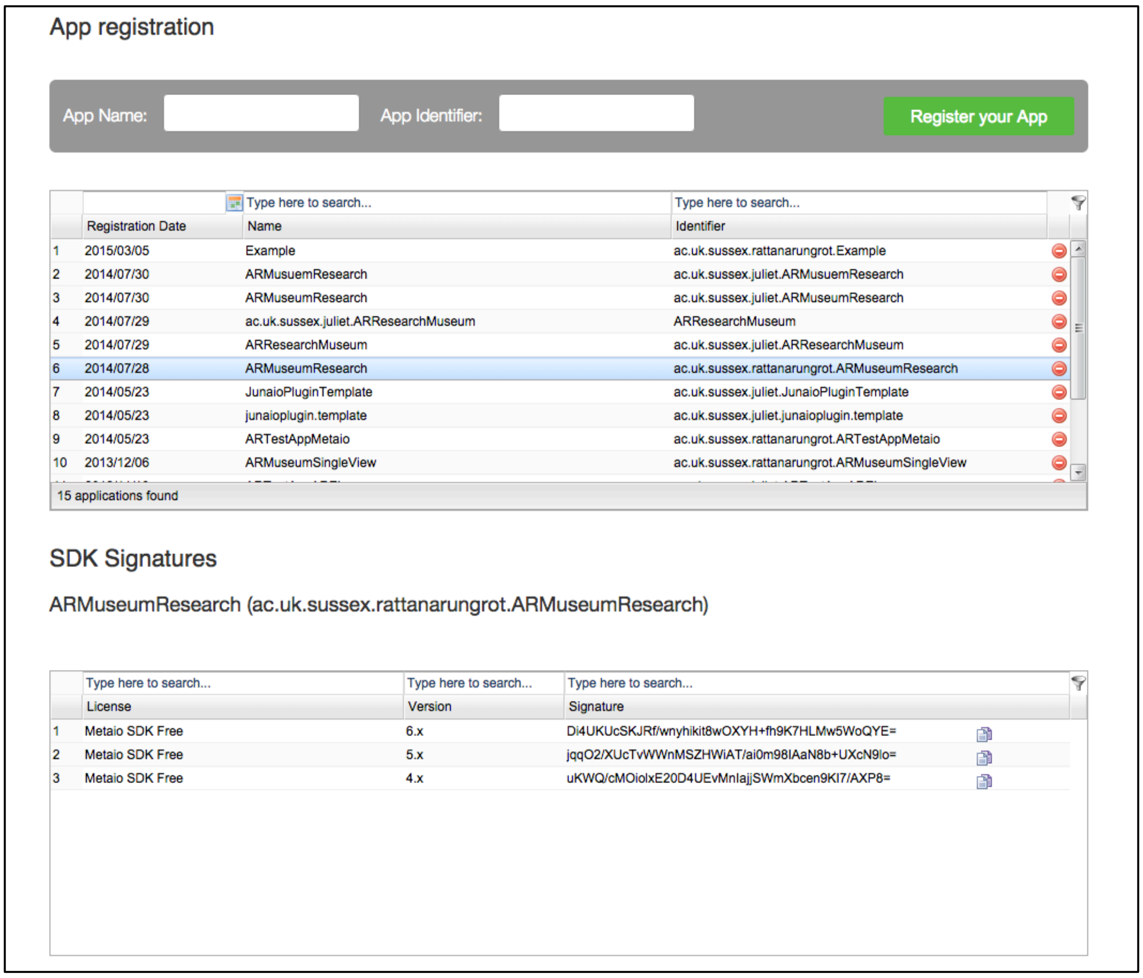
The developer opens the iOS project in the XCode IDE and adds the metaioSDK.framework and other necessary libraries into the Linked Frameworks and Libraries in the General Menu. Figure E.4 illustrates the framework and libraries required in the development environment.



Name	Status
AssetsLibrary.framework	Required
CoreGraphics.framework	Required
MobileCoreServices.framework	Required
CoreLocation.framework	Required
CoreData.framework	Required
libxml2.dylib	Required
libz.dylib	Required
CoreMotion.framework	Required
OpenGLES.framework	Required
AudioToolbox.framework	Required
QuartzCore.framework	Required
CoreVideo.framework	Required
CoreMedia.framework	Required
AVFoundation.framework	Required
MediaPlayer.framework	Required
CFNetwork.framework	Required
EventKit.framework	Required
Security.framework	Required
CoreImage.framework	Required
metaioSDK.framework	Required
UIKit.framework	Required
Foundation.framework	Required

Figure E.4 The frameworks and libraries in the development environment

The next step is to register the application in the Metaio Developer Portal; the system then creates an SDK signature for each application. The developer must copy the signature and pass it into the MetaioLicenseString in the Info menu. Figure E.5 shows the application registration in the portal that can be done after logging into the system.



App registration

App Name: | App Identifier: | Register your App

	Registration Date	Name	Identifier
1	2015/03/05	Example	ac.uk.sussex.rattanarungrot.Example
2	2014/07/30	ARMusuemResearch	ac.uk.sussex.juliet.ARMusuemResearch
3	2014/07/30	ARMuseumResearch	ac.uk.sussex.juliet.ARMuseumResearch
4	2014/07/29	ac.uk.sussex.juliet.ARResearchMuseum	ARResearchMuseum
5	2014/07/29	ARResearchMuseum	ac.uk.sussex.juliet.ARResearchMuseum
6	2014/07/28	ARMuseumResearch	ac.uk.sussex.rattanarungrot.ARMuseumResearch
7	2014/05/23	JunaioPluginTemplate	ac.uk.sussex.juliet.JunaioPluginTemplate
8	2014/05/23	junaioplugin.template	ac.uk.sussex.juliet.junaioplugin.template
9	2014/05/23	ARTestAppMetaio	ac.uk.sussex.rattanarungrot.ARTestAppMetaio
10	2013/12/06	ARMuseumSingleView	ac.uk.sussex.rattanarungrot.ARMuseumSingleView

15 applications found

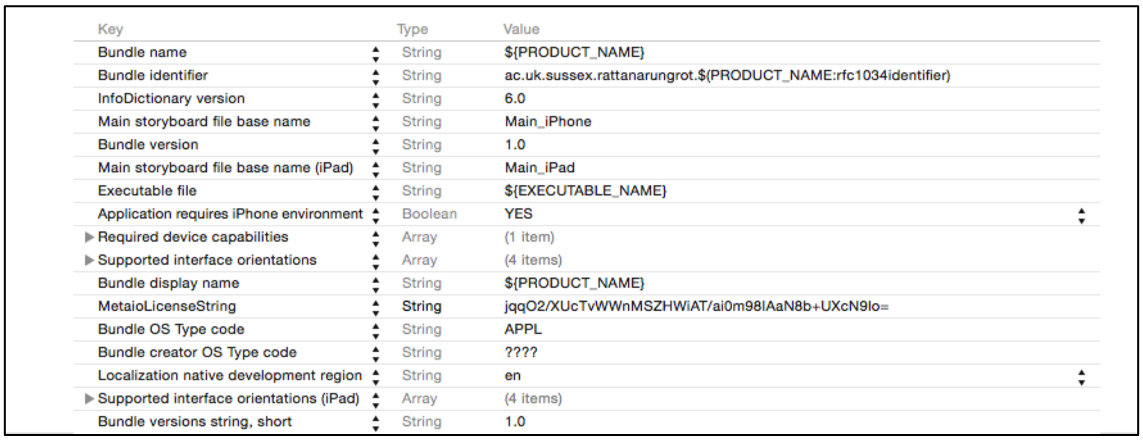
SDK Signatures

ARMuseumResearch (ac.uk.sussex.rattanarungrot.ARMuseumResearch)

	License	Version	Signature
1	Metaio SDK Free	6.x	Di4UKUcSKJRf/wnyhikit8wOXYH+fh9K7HLMw5WoQYE=
2	Metaio SDK Free	5.x	jqqO2/XUcTvWWnMSZHWiAT/ai0m98IAaN8b+UXcN9lo=
3	Metaio SDK Free	4.x	uKWQ/cMOiolxE20D4UEvMnIajjSWmXbcen9KI7/AXP8=

Figure E.5 The application registration page on the Metaio developer portal

The Metaio SDK version 5.X signature is copied and pasted into the XCode iOS Target Properties. Figure E.6 presents the SDK signature and other necessary keys in the development environment. The signature is used to identify the member or user of the SDK installed in the system.



Key	Type	Value
Bundle name	String	${PRODUCT_NAME}
Bundle identifier	String	ac.uk.sussex.rattanarungrot.${PRODUCT_NAME:rfc1034identifier}
InfoDictionary version	String	6.0
Main storyboard file base name	String	Main_iPhone
Bundle version	String	1.0
Main storyboard file base name (iPad)	String	Main_iPad
Executable file	String	${EXECUTABLE_NAME}
Application requires iPhone environment	Boolean	YES
▶ Required device capabilities	Array	(1 item)
▶ Supported interface orientations	Array	(4 items)
Bundle display name	String	${PRODUCT_NAME}
MetaioLicenseString	String	jqqO2/XUcTvWWnMSZHWiAT/ai0m98lAaN8b+UXcN9lo=
Bundle OS Type code	String	APPL
Bundle creator OS Type code	String	????
Localization native development region	String	en
▶ Supported interface orientations (iPad)	Array	(4 items)
Bundle versions string, short	String	1.0

Figure E.6 The MetaioLicenseString in the development environment

In the Metaio SDK folder, the developer must copy MetaioSDKViewController files and EAGLView files and paste them into the application development folder. Then these files have to be added into the project in the XCode. Figure E.7 presents the Metaio SDK folder containing the required files.

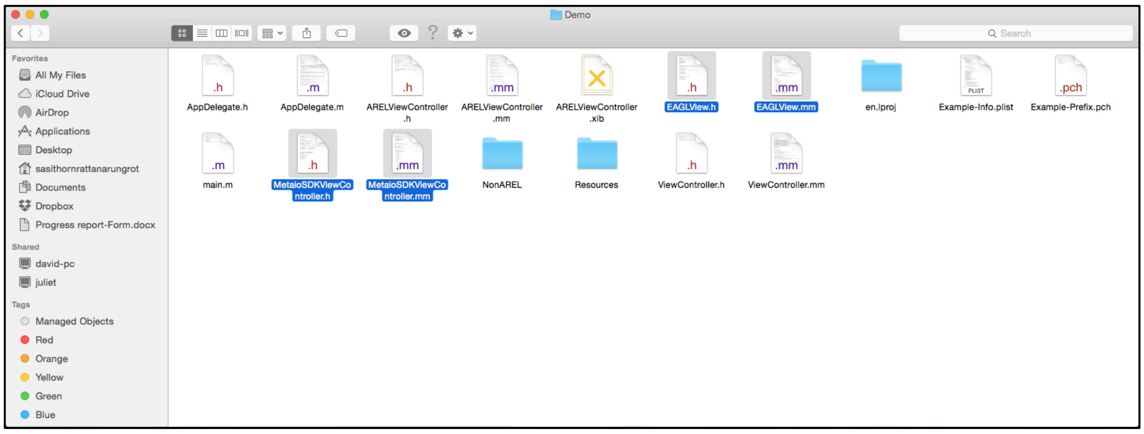
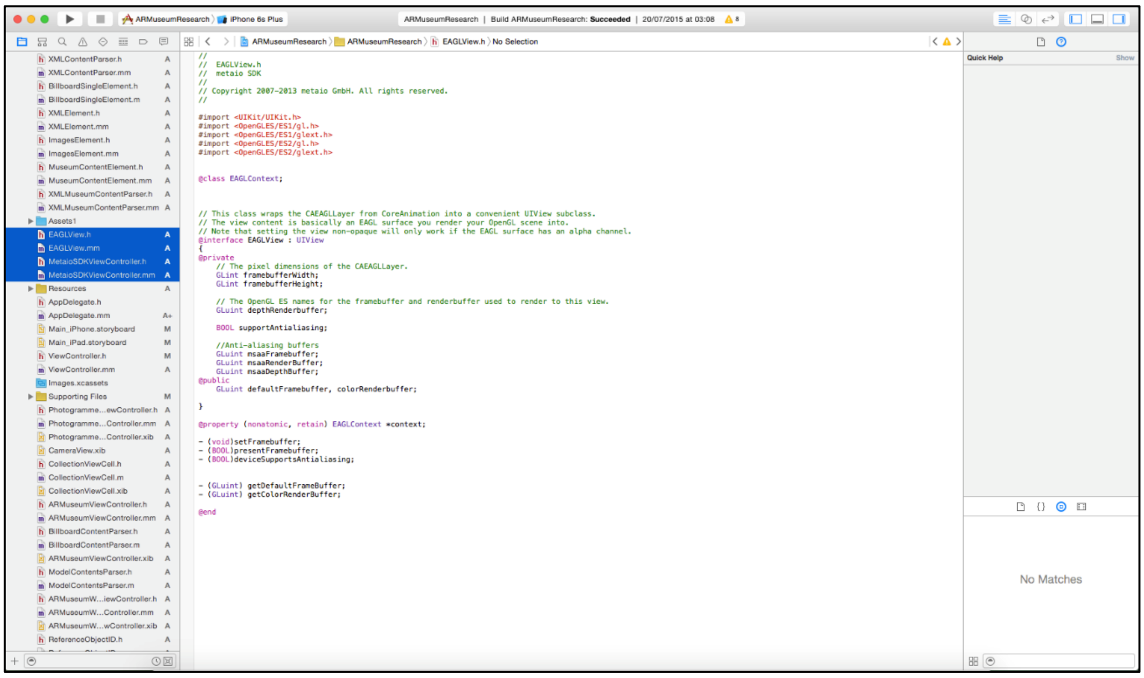


Figure E.7 The Metaio SDK folder containing the required files

The MetaioSDKViewController and EAGLView files utilise the MetaioSDK.framework in order to create a sensor interface and perform object tracking and rendering. Figure E.8 presents the project of the mobile AR application in XCode and the Metaio SDK's view controller and its supporting files.



```
//
//  EAGLView.h
//  metaio SDK
//
// Copyright 2007-2013 metaio GmbH. All rights reserved.
//

#import <UIKit/UIKit.h>
#import <OpenGLES/ES1/gl.h>
#import <OpenGLES/ES1/glext.h>
#import <OpenGLES/ES2/gl.h>
#import <OpenGLES/ES2/glext.h>


@class EAGLContext;



// This class wraps the CAEAGLLayer from CoreAnimation into a convenient UIView subclass.
// The view content is basically an EAGL surface you render your OpenGL scene into.
// Note that setting the view non-opaque will only work if the EAGL surface has an alpha channel.
@interface EAGLView : UIView
{
@private
    // The pixel dimensions of the CAEAGLLayer.
    GLint framebufferWidth;
    GLint framebufferHeight;

    // The OpenGL ES names for the framebuffer and renderbuffer used to render to this view.
    GLuint depthRenderbuffer;

    BOOL supportAntialiasing;

    //Anti-aliasing buffers
    GLuint msaaFramebuffer;
    GLuint msaaRenderBuffer;
    GLuint msaaDepthBuffer;
@public
    GLuint defaultFramebuffer, colorRenderbuffer;

}

@property (nonatomic, retain) EAGLContext *context;

- (void)setFramebuffer;
- (BOOL)presentFramebuffer;
- (BOOL)deviceSupportsAntialiasing;


- (GLuint) getDefaultFrameBuffer;
- (GLuint) getColorRenderBuffer;

@end
```

Figure E.8 The project of the mobile AR application and the MetaioSDKViewController

F. Object Tracking

Vision-based tracking

Real-time vision-based tracking is typically implemented in mobile AR systems because new generation mobile devices already have a high quality camera that can be used as a sensor. Vision-based tracking uses computer vision methods for processing video streams and images. Performing vision-based tracking on mobile AR results in low computational requirements, higher accuracy, flexibility, robustness and cost savings in operations when compared to other sensing techniques [19]. In addition, there are some advantages of vision-based tracking such as higher precision, less sensitivity to intervention and more compact systems [110]. Most of the currently available tracking techniques are composed of feature-based and model-based tracking [111].

For indoor AR applications, vision-based tracking is particularly suitable for implementation because of its hardware requirements, robustness and accuracy. Computer vision does not rely on the volume of sensitizers, unlike magnetic, mechanical or ultrasonic sensors [61]. In addition, only computer vision can guarantee an accurate alignment between real objects and virtual objects. Vision-based tracking requires cameras, which are equipped in mobile devices or connected to computers as its optical sensor. Registration in an AR system has to be accurate in order to maintain the user's perception of virtual objects related to real objects in the same environment. In addition, there is a close-loop AR system that can dynamically measure registration errors and correct the cause of any mis-registration such as camera orientation or position [112].

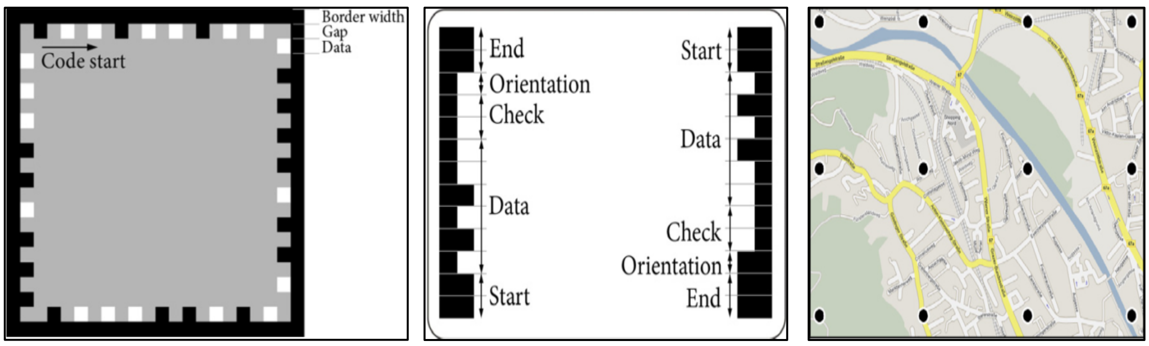
The closed-loop system can correct temporal errors in video-based AR systems that are analogous to vision-based tracking systems [24]. Nevertheless, registration is the most crucial component in AR systems. Virtual objects have to be adjusted with the 3D location and orientation of real objects when users move their viewpoint. Adjustment is based on precisely estimating a real-world viewing pose that is six degrees of freedom (6DOF): three degrees of freedom for position and three for orientation. The accuracy of tracked objects posed correctly defines the projection of 3D virtual objects into the real world scene[113].

Feature-based tracking

Feature-based tracking uses fiducials that are easily recognizable landmarks [27] or known artificial patterns [110] as a marker for tracking and overlaying with virtual objects using registration techniques or camera pose estimation. The fundamental characteristic of feature-based tracking is to discover 2D image features and the relationship with the 3D graphic objects or scenes relevant to their features. Another feature is camera pose calculation of 2D features and finding the 3D coordinates of markers for object registration. In addition, feature-based tracking can be divided into two groups: visual marker-based and visual marker-less tracking [24][27].

Marker-based tracking

Mobile AR applications typically use original printed markers that are in square or circular shapes. Markers or fiducial markers are more easily tracked and processed by cameras and computer vision approaches when they are placed on specific or general positions around the environment such as tabletops, books or walls. The fiducial markers offer feature points that are tracked and 2D image coordinates are identified and calculated for 3D object coordinates. There is a new marker technique that is less obtrusive than the usual black and white squares, called Studierstube Tracker. The Studierstube Tracker is a new marker-tracking library for mobile AR that supports frame markers, split markers and dot markers [114]. Figure F.1 illustrate a frame marker, split marker and dot marker that can be tracked by the Studierstube Tracker.



(a) Frame marker (b) Split marker (c) Dot marker

Figure F.1 Markers for Studierstube Tracker [114]

Marker-based tracking has been found in multiple AR applications. Fiducial markers are still useful as they can be tracked with high accuracy, robustness and low computational power. In contrast, printed markers have to be installed in the real environment where good visualization and interaction with users is not guaranteed. Due to the functionality in computer vision methods, mobile AR applications are not limited to use in indoor environments. In addition, applying marker-based AR in some environments may cause occlusion of markers by real objects; this will affect the quality of view and appearance. Applying fiducial markers is inconvenient because they have to be placed in the real environment and the system can track only those markers in the view. Moreover, markers have specific shapes and colors while image-processing algorithms have much more potential to track a variety of shapes, colors and textures found in the natural feature extraction of real objects [115]. Furthermore, markers are useless or inconvenient to present and track in some outdoor applications such as location-based AR, gaming or travel guides that offer graphics, media content and location-based user interfaces. Applying hybrid tracking such as GPS, Radio-Frequency Identification (RFID) and tracking sensors with marker-less techniques can be done effectively to improve the usability and functionality of the services.

The implementation of marker-less tracking is more complicated and consumes more processing power than marker-based tracking. However, marker-less tracking leads to further advanced developments in computer vision and image processing methods, allowing the system to track real objects in the real environment without the use of any fiducial markers.

Marker-less tracking

Trends in mobile AR applications have recently focused on marker-less tracking that uses computer vision for tracking the natural features of objects to create highly miscellaneous and flexible applications. Marker-less tracking improves usability and ubiquity in AR, especially in complex environments, because it does not require any markers to be placed on real world scenes. The systems can utilize natural features in the real environment and track any part of the scene as a marker [110]. Marker-less AR effectively enhances the user's experience in 3D visualization and interaction with the virtual objects in the real world scenes. Marker-less tracking extracts natural features of real objects such as points, lines, edges and textures [24] by using natural feature extraction methods. The system performs detecting and tracking in any part of the real environment and overlays 3D virtual models at the calculated pose by using optical sensors and registration techniques. These methods make the development of mobile AR applications much more flexible and scalable for indoor or outdoor use by implementing marker-less tracking with other sensing techniques.

The difference between marker-based and marker-less tracking is in the methods that are used to track and register the computer-generated virtual objects. In addition, performing tracking in complex environments such as those involving moving objects, occlusion of objects and illumination increases complexity in processing. Because of these factors, marker-less tracking is obviously more complicated and has greater computational requirements than marker-based tracking. Advanced applications in marker-less AR at the moment can be found in interactive prints, shopping guides, advertising, maintenance tools, AR magazines, travel guides, augmented cities, AR kiosks, etc.

Marker-less tracking applications enhance the AR experiences of users by augmenting computer-generated virtual objects from real objects in the real environment and presenting in different ways, rather than relying on fiducial markers. Figure F.2 presents the example of marker-less tracking by augmented a targeted object [45].

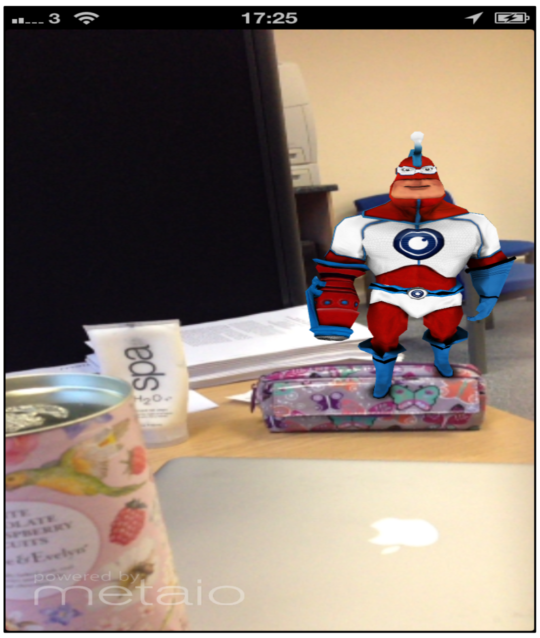


Figure F.2 Marker-less tracking

Real-time natural feature tracking has been studied and developed continuously for tracking or detection techniques [113][116][117][118][119] and implementation techniques [120]. The following research proposes advanced development and technologies in natural feature tracking.

One of the architectures for marker-less tracking is detecting and tracking natural features such as points and regions occurring in video images or unprepared environments by using a multistage tracking algorithm. The architecture is composed of natural feature detection and selection, multistage motion estimation and feedback evaluation. Additionally, the applications can perform image sequence annotation, pose stabilization and tracking range extension. The system is implemented in a closed-loop architecture that maintains performance and accuracy [113]. Figure F.3 shows the natural feature tracking that present tags of detected positions.

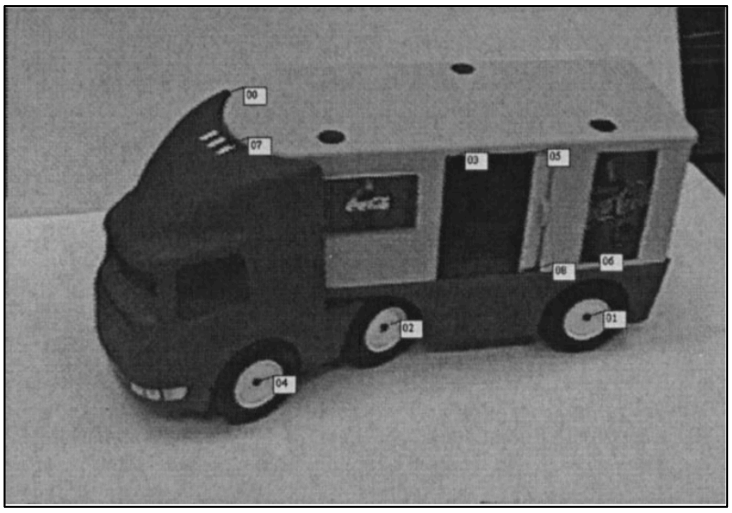


Figure F.3 Natural feature tracking [113]

Scale Invariant Feature Transform (SIFT) and Fast Keypoint Recognition Algorithm (FERN) are natural feature tracking techniques in which the main tasks of natural feature extraction are composed of feature detection, feature tracking, feature extraction and pose calculation in real-time. These techniques are modified to perform real-time 6DOF tracking and pose refinement on mobile phones. SIFT performs feature detection, feature tracking, descriptor creation, descriptor matching, outlier removal and target data acquisition, while FERN performs feature detection, classification, training, active search and outlier rejection [116]. In addition, there is another real-time architecture for feature tracking in which its operations are performed in a synchronized multithread framework composed of video frame capture, optical-flow feature tracking, distinctive feature detection and virtual object rendering. The system also performs recognition on different stored scenes by matching SIFT features [117]. Figure F.4 presents the Handy AR using feature-based maker-less tracking.

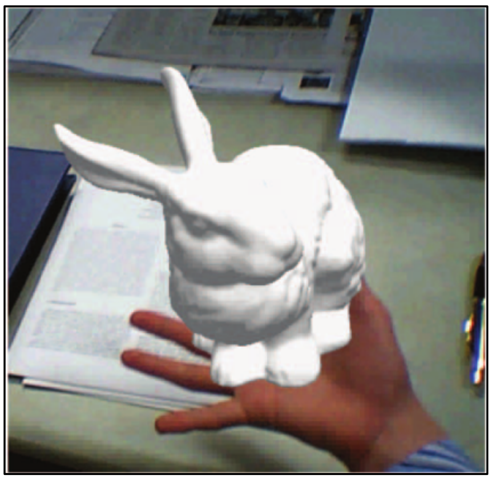


Figure F.4 Handy AR [117]

The structure-from-motion technique is a marker-less real-time tracking technique for AR applications using a fisheye lens on a camera for tracking 3D scene points and 2D feature points, and a tracker is used to track 2D image features. The system utilizes the structure-from-motion technique for tracking geometric data and estimating camera pose and 3D features online. Feature matching is also an efficient technique for achieving real-time feature tracking and object registration [25].

Natural feature tracking can be implemented on many different platforms as they are currently presented in mobile AR applications. There is a natural feature tracking pipeline on web browsers that is cross-platform with AR solutions. The system is implemented by JavaScript and embedded in a plugin-free web technology-based AR pipeline in which all tracking tasks such as video access, detection and tracking and 3D rendering are implemented in HTML5, JavaScript and WebGL platforms [120]. Moreover, marker-less tracking has recently been adapted to 6DOF hand-tracking techniques. Hand-tracking techniques track the position and orientation of a user's hands. Tracking determines the user's hand gestures such as grabbing as a virtual user interface for AR systems [28][121].

Model-based tracking

One current approach is model-based tracking, a trend in vision-based tracking that uses the natural features of real objects such as lines, edges and textures for tracking and calculating camera positions in real time. Model-based tracking methods use models of the extracted features of the objects such as a Computer-Aided Design (CAD) model or 2D template [24]. The models are usually created from features including edges and lines but edges are most frequently used because of the ease in finding them and their robustness to lighting changes. Textures are also useful features for creating models and they can be combined with edge information or feature points by performing edge detection or extraction to obtain more robustness and accuracy [122]. Figure F.5 presents the model-based tracking using 2D feature points.

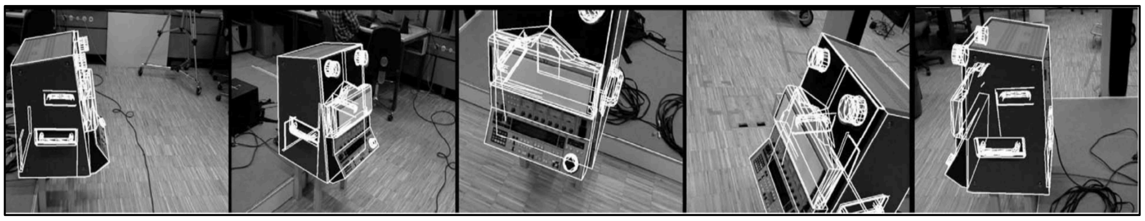


Figure F.5 Model-based tracking [123]

Model-based tracking is a marker-less tracking technique, which is the most recent technique in computer vision-based tracking. Lines, edges and textures are features that can be used to build models. Edges and textures are the most useful features and some applications integrate both to build models [24]. Model-based tracking is a powerful technique for determining 3D location, pose of objects and feature models of image sequences. However, tracking complex objects that have high DOF articulated models is still complicated because small components lack good visual features, include clutter and there may be occlusion between components. Possible solutions are to base feature models on multiple views or to increase the reliability of particular feature models [124]. The main advantage of model-based tracking is that knowledge about the scene or 3D information leads to improvements in robustness and performance by predicting hidden movements of the object, and actions to reduce the effects of outlier data can be found in the tracking process [62] .

Model-based tracking techniques consist of recursive tracking and tracking by detection on natural features of tracked objects, composed of edges, textures and optical flow. Therefore, when considering features of objects used for tracking including edge-based, texture-based and optical-flow based methods, model-based tracking can be categorized into two methods: tracking-by-detection and recursive tracking.

- Tracking-by-detection
  - View based (edge-based)
  - Keypoint based (texture-based)
- Recursive tracking
  - Point sampling (edge-based)
  - Optical flow based
  - Template matching (texture-based)
  - Interest point based (texture-based)

Tracking-by-detection

Tracking-by-detection calculates the current camera pose from the tracked objects without any previous pose estimation or knowledge of scene geometry [125]. Tracking by detection can be implemented to track features of objects and create models from edges and textures. This technique extracts feature points from video frames and matches them into a database of models and 3D locations of objects. A 3D pose is then estimated from the correspondences. The implementation of a tracking by detection approach creates specific difficulty because of the image database and the input frames that may be obtained from many different viewpoints [61]. Figure F.6 shows the result of the tracking-by-detection technique.

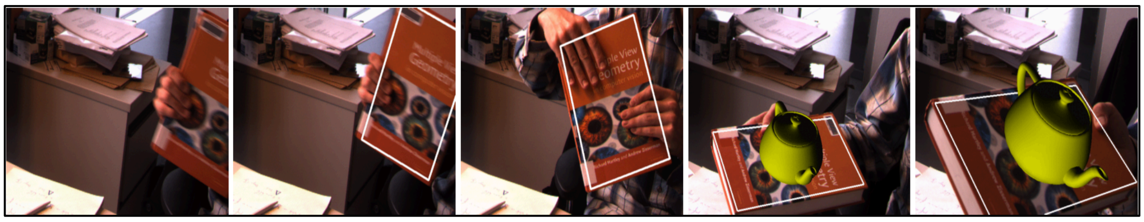


Figure F.6 Tracking-by-detection [61]

Edge-based tracking by detection techniques are developed and implemented in several approaches because edges are efficient and robust to detection. The edge-based tracking method extracts the edges of an object and matches the 3D object model to the database, then estimates the pose of the object. The main tasks are typically composed of tracking objects, generating models, recognition and calculation of poses and projecting virtual objects onto the scene [126]. Recent research on edge-based tracking-by-detection approaches mostly focus on developing algorithms for real-time tracking, recognition and matching object models [62][125][126][127].

One of the model-based tracking techniques is a real-time 3D model-based tracking algorithm for a video-see-through monocular vision system on a visual servoing system. The goal of visual servoing is to move a camera in order to detect an object at a defined position in the image. Visual servoing uses techniques composed of a moving edge tracker, feature matching, uncertainty propagation and M-estimator in order to achieve robust real-time tracking [62]. The system architecture for fully automated marker-less AR utilizes a sparse metric model and the SIFT algorithm in order to generate stable natural features of objects and determine the pose of a virtual object. The system performs in two stages. The first stage is extracting SIFT features for creating a metric model of objects, conducting model recognition and current camera pose computation. The second stage is matching detected features in the current video frames with the world model. The matching is used to compute the current camera pose [125].

There are some methods implemented using 3D object tracking and recognition for 3D object pose calculation. A 3D model-based tracking and model-based recognition technique can be done using a 3D CAD model of an object for training. The method performs training with models based on the geometry information of a 3D CAD model of an object and then processes an exhaustive search and hierarchical search to match the models and calculate a 3D object pose [127]. Another method performs a visibility test on visible edges of the model and extracts the sample points of visible 3D model lines, then minimizes the distances between projection of all sample points and the sample points in the image [126].

The tracking by detection technique still has some challenges in implementation and there is an approach that is efficient enough to improve tracking performance [61]. It can be achieved by using a classifier to recognize the feature points, which relies on a texture-based tracking method. The texture-based tracking method relies on texture information found on real objects [125]. A classification method is implemented by utilizing a FERN for generating models and estimating object poses [128]. This algorithm is based on a naive Bayesian classification framework for accuracy and robustness. The classifier is used to recognize the patches surrounding key points. Furthermore, the classifier uses hundreds of simple binary features and models class posterior probabilities. The system uses SIFT to compare computational times with the FERN algorithm.

For improving the robustness and accuracy in tracking by detection methods, edge-based and texture-based can be combined together to perform real-time model-based tracking and pose estimation. There is a real-time, robust and efficient 3D model-based tracking algorithm that utilizes a virtual visual servoing approach to estimate the pose between the camera and the object and to integrate texture information to edge-based extraction and pose computation [111].

Recursive tracking

Recursive tracking is used for tracking 3D objects with fast camera movements so the system uses the last camera pose to estimate the current pose. Edge-based recursive tracking uses a point sampling method that samples some control points along the edges of the 3D model and compares the projection with strong gradients in the image. In addition, texture-based recursive tracing is classified into two subcategories: template matching, which recovers object movement by applying a distortion model to a reference image, and interest point-based, which localizes features in the camera pose estimation. Each tracking technique is suitable for different scenarios and environments such as moving objects or moving cameras. Edge-based methods are suitable when tracked objects are polygonal or have strong contours. Optical flow-based techniques work with textured objects and a texture-based method is the best solution if the optical flow-based techniques do not solve the problem.

Model-based visual tracking has recently attracted attention in many areas such as robotics and AR where object recognition tasks are quite commonly combined with visual tracking [129]. Figure F.7 presents shape augmentation using recursive tracking.

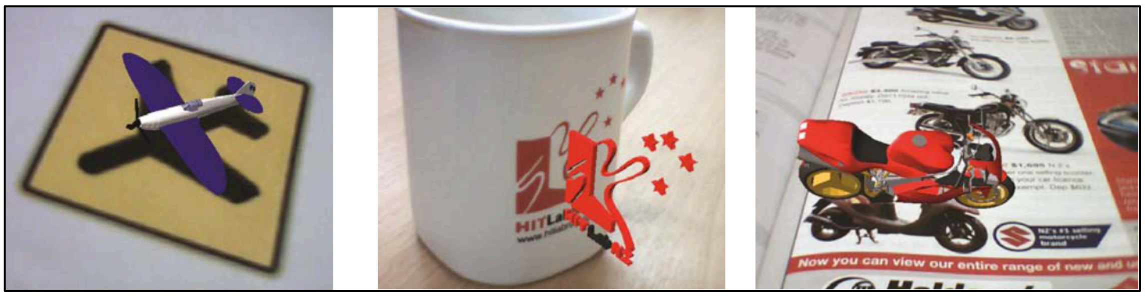


Figure F.7 Shape augmentation using recursive tracking [129]

Real-time 3D tracking is an important module in mobile AR for performing computer vision tasks in both indoor and outdoor environments. However, integrating 3D marker-less tracking with computer vision techniques is still a challenge to enhance the tracking mechanism in video frames for example, creating 3D models from tracked objects. Object recognition is a new technique incorporating maker-less tracking to improve the proficiency in tracking and pose estimation that allows the system to increase its perception in a real environment.

Object recognition

Vision-based tracking has now been incorporated effectively with object recognition to perform visual tasks on AR platforms. Real-time object recognition has recently been applied to perform object detection, object tracking, feature extraction, recognition and augmentation with real environments such as planar surfaces, images and 3D objects. Object recognition is a complicated task in computer vision because there are various kinds of real object in the environment. Each object has their own size, shape, pose and occlusion with other objects. Object recognition is a method in computer vision that can be effectively developed to improve advanced features in some AR applications and robot vision such as face, object and logo recognition. Most of the recent advances in computer vision have focused on machine learning techniques in object recognition [61]. The object recognition field has made significant progress and is often a solution to current limitations. Most AR applications today rely on working with recognition target objects such as sketches, images and video frames [130].

3D object recognition is a powerful method for identifying physical objects, because when a particular object is recognized it can trigger relevant services such as the display of associated virtual objects or loading of new graphical user interfaces to access more features. The tracking method in video streams is composed of two phases: learning and tracking for pose estimation [19]. The system can recognize and identify real objects after being trained with sample objects. 3D object recognition in AR works together with marker-less tracking in order to detect and track real objects and then performs real-time recognition and pose estimation. As an adaptive learning system, the first stage is training the system with new objects by using feature extraction and machine learning techniques. The system is then able to track real scenes and identify the objects in the scenes by using pattern or shape-matching techniques. Performing object recognition requires a database for maintaining the sample shapes or models that have been trained. The machine learning applied in object recognition techniques consist of neural networks, support vector machines and boosting. The following sections introduce the recognition technique for shapes, 2D images and 3D objects. These are described by research that has implemented object recognition with different techniques.

Planar surfaces recognition

Recognition systems for AR have recently been studied and developed for a variety of scenarios. One of the approaches used is real-time recognition and pose estimation for planar surfaces or 2D images. The research in this approach are Nestor [129] and Scalable Triangulation-based Logo recognition [130].

Nestor is a system for recognizing and tracking shape contours in real-time on a mobile phone. The system can track hand sketches and overlays 3D virtual content onto a real scene. The process starts training the system by presenting new shapes to the camera; the system analyses and stores the learned shapes in its library. Two main tasks in the system are model-based recognition and 3D pose estimation. For model-based recognition, the system performs shape recognition by analyzing the contour structures and then generating the projective invariant signatures from their concavities that are further used to accomplish feature extraction for pose estimation and tracking. In addition, the geometric invariant constructions are used to calculate the signatures for the shapes that can then be recognized across different viewpoints.

Another research in the shape recognition area is Scalable Triangulation-based Logo Recognition. The objective of this research is to recognize logos in a query image by applying a bag-of-words model because logos normally contain useful information such as text and geometric shapes for detection processes. Moreover, most logos emerge on planar surfaces that are used to facilitate the process of feature detection and extraction. The research uses a class database to collect the sample logos separated into each class or brand in the database. The local features of each sample are grouped by using multi-scale triangulation, then the local features are applied by a Delaunay triangulation that is directed by the scales from the feature detector. A signature from each triangle is extracted and the signatures of all samples in the class are used to represent each class. The signatures are extracted from a query image during the recognition process and then they are matched to the samples in the class database. Finally, an inverted index response is used to rank the class models and provide a sub-linear search.

3D object recognition

3D object recognition is the next advanced step in an object recognition system in an AR environment. Current research on recognition have focused on object detection, object tracking, feature extraction, pattern matching and generating virtual content relevant to real 3D objects and environments at the estimated pose. Research on 3D object recognition is mainly focused on face and object detection as well as recognition by implementing machine-learning techniques. Currently much research and applications concentrate on recognizing 3D objects in photographs or image sequences with samples or training input images such as personal photo collections, location recognition, intelligent photo editing and image search. These are developed from the area of recognition including face recognition, instance recognition, category recognition and context and scene understanding. The following are the techniques used to perform 3D object recognition.

Image-based recognition

An example of tracking and image-based recognition for mobile AR is AR-PDA. The AR-PDA project effectively provides services to customers by presenting AR technology on a high-end mobile phone or PDA. The goal of this project is to support customers by providing AR as an interactive tool for their domestic equipment, such as a virtual user manual and maintenance tasks. The project is developed on client-server architecture that relies on a mobile network. The server receives live video streams that are taken by an integrated camera and recognizes the objects by analyzing the images. The specific relevant information or multimedia is assigned to the video stream and sent back to the AR-PDA. The image-based recognition and tracking in this project uses 3D-geometrics of the objects and requires information created from a 3D-CAD file [131].

2D shape similarity metric

A 2D shape similarity metric is a technique for comparing an unknown view with prototypical views and measuring the similarity by evaluating the distance between the segmented shapes of 3D objects and the prototypical views. The shape similarity metric is used to rank the similarity between unknown objects in unknown views and samples that are stored in the metric to recognize real objects and their pose. An aspect graph is implemented to specify the viewing sphere area. The aspect is produced by evaluating the similarity of each 2D shape and grouping its views, then each aspect is represented by a prototypical view. The system identifies a prototype view from a database of all views of objects that stores views in five-degree increments. The prototypical views help the system decrease the search time by indexing. The testing process starts by randomly selecting an object from the database and then generating views from the selected object at random angles. The unknown views are compared to the prototypical views and the results from matching are sorted and reported to the unknown object [132].

3D model recognition

The 3D model recognition technique creates 3D metric models from images of an object that are taken by a camera. The system then recognizes new images by matching with the 3D models and accurately calculates object poses for superimposing a virtual object onto the new images. The research specifies a solution for accurate 3D pose calculation so that a virtual model is superimposed on recognized objects at the correct pose. First, the system processes SIFT feature extraction from the sample images and implements indexing by linking images to create a spanning tree. The correspondences are utilized to create a matrix model of the real objects. Moreover, the system calculates the camera calibration parameters and camera poses relevant to the image viewpoints. The next process detects features of the video frame and matches the features with the real model using a Best-Bin-First (BBF) algorithm. Then the system calculates the current pose of the real object and presents the virtual object superimposed on the objects using a RANdom SAmple Consensus (RANSAC) and Levenberg-Marquardt algorithm [133]. Figure F.8 shows 3D object recognition using the 3D model recognition technique.

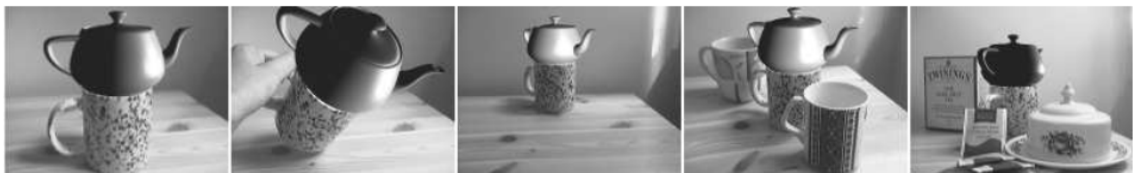


Figure F.8 3D object recognition using 3D model recognition [133]

Neural network for 3D object recognition

Another research that implements machine-learning techniques is a multiple feature extraction method of 3D objects from 2D images to increase the performance of an appearance-based 3D object recognition system. The 2D image features to be extracted include texture, color, Hu’s moment and affine moment invariants. Hu’s moment invariants represent the invariant rotation, scale and translation properties. Affine moment invariants represent the invariant transformation properties. For each 2D image of a 3D object, these characteristics are mapped to a 1D feature vector of 23 components of the 2D image. The vector is used to train a back propagation neural network and the trained neural network is used to identify 3D objects. In addition, the color moments and texture characteristics are used to recognize 3D objects of the same shape [134].

The concept for developing an object recognition system can be applied to unloading processes in order to organize and detect the pose of universal packaged goods in a container unit by using sensor data. The unloading process is challenging because the object recognition system for goods in undefined shapes is complicated and requires robustness and reliability. Moreover, the unloading process requires a proper sensor system to acquire shape and depth information of the goods in packaging scenarios. The system uses time-of-flight cameras that can work with the specific environment inside the containers. The system needs many range images from the cameras to improve the efficiency in sensor detection. Then, the system analyses the range images to extract the important features and creates a class of the packaged goods relevant to the features [135].

There are algorithms for 3D object recognition including the Clonal Selection Algorithm (CLONALG) and Particle Swarm Optimization (PSO). The modeling process evaluates the 2D image appearance of a 3D object and creates a 3D model. The first stage is segmenting the 2D image with Otsu’s threshold method. The second stage is extracting a set of invariant features. The third stage is assigning the neural network weights by utilizing CLONALG and PSO. Finally, the neural network is trained by implementing a Levenberg-Marquardt (LM) algorithm. The experiment outputs show that the CLONALG-LM algorithm has higher efficiency and performance compared to PSO-LM and other traditional algorithms [136].

3D modeling, rendering and reconstruction

In the area of 3D modeling and reconstruction, model-based reconstruction is developed by implementing 3D model-based tracking and 3D object recognition such as whole body modeling and tracking. Tracking humans, modeling their shape and appearance and recognizing their activities are all applications in development. Therefore, the idea of integrating 3D modeling and tracking by generating 3D models or shapes from the tracked objects in real time is a possible development for an AR environment. In addition, model reconstruction and rendering techniques can be incorporated. These can be used to enable new mobile interactive applications in the area of mixed reality that combines virtual objects with real objects and a real world scene. One research in this area is an online interactive primitive modeling technique that enables a user to reconstruct and label geometry relative to a marker in a scene. The research uses object-based tracking incorporating a feature tracker for tracking the movement of the camera relative to the object. Its outcome is a lightweight and fast approach to scene modeling that might be suitable for use on mobile computer devices [137].