A crowdsourcing framework for the management of mobile multimedia nature observations

Giannis Skevakis, Chrisa Tsinaraki, Ioanna Trochatou and Stavros Christodoulakis
School of Electronic & Computer Engineering, Technical University of Crete/Laboratory of Distributed Multimedia Information Systems and Applications, Chania, Greece

Abstract
Purpose – This paper aims to describe MoM-NOCS, a Framework and a System that support communities with common interests in nature to capture and share multimedia observations of nature objects or events using mobile devices.
Design/methodology/approach – The observations are automatically associated with contextual metadata that allow them to be visualized on top of 2D or 3D maps. The observations are managed by a multimedia management system, and annotated by the same and/or other users with common interests. Annotations made by the crowd support the knowledge distillation of the data and data provenance processes in the system.
Findings – MoM-NOCS is complementary and interoperable with systems that are managed by natural history museums like MMAT (Makris *et al.*, 2013) and biodiversity metadata management systems like BIOCASE (BioCASE) and GBIF (GBIF) so that they can link to interesting observations in the system, and the statistics of the observations that they manage can be visualized by the software.
Originality/value – The Framework offers rich functionality for visualizing the observations made by the crowd as function of time.

Keywords Multimedia capturing, Observation, Annotation, Crowdsourcing, Ontology, Interactive map, Visualization, Event, Path, Mobile multimedia

Paper type Research paper

1. Introduction
We believe that several scientific fields (biodiversity, biology, agriculture, etc.) would greatly benefit if it was possible to have the observations made by casual users with interest in the domain, informed outsiders and experts, captured in multimedia together with their geospatial and time information. These multimedia observations could be stored in observation repositories, and further annotated. Crowd intelligence can contribute to correctly classify and annotate the observations without the direct involvement of the experts in the domain. Moreover, advanced visualization services (including geovisualization) could be offered on top of the observation repositories. These services may assist the domain researchers to draw useful scientific conclusions related, for example, to sustainability and preservation. In application environments like biodiversity, where the capturing process is extremely slow due to the limited number of experts and funding, a software framework that manages the mobile

International Journal of Pervasive Computing and Communications
Vol. 10 No. 3, 2014
pp. 216-238
© Emerald Group Publishing Limited
1742-7371
DOI 10.1108/IJPCC-06-2014-0038

multimedia observations of biodiversity data and their correct identification and knowledge distillation by the crowd that has interest in the field is extremely valuable.

Multimedia observation capturing may be assisted from the modern media capturing devices (cameras, mobile phones, tablets, etc.) that may also store the capturing parameters, including spatiotemporal and direction information. There is, though, an urgent need for supporting users (both domain experts and informed outsiders) that make scientific observations to capture and annotate the captured material. Such users should be assisted from software application frameworks that allow them to easily capture, upload and annotate their observations. These frameworks should support the interaction with other users in ways familiar to the users (e.g. social network interaction), to discuss on the existing observations and even argue on issues relevant to them (classification, conclusions, etc.).

Research that has been carried out in this direction has so far treated specific research questions, but has not focused on integrated frameworks that support the whole process. Multimedia content and context capturing has evolved with the availability of mobile/smart capturing devices, and its applications range from the user preference-based multimedia content retrieval and filtering (Tsinaraki and Christodoulakis, 2006) to the mobile capturing of multimedia material and the context-based multimedia capturing of events (Christodoulakis *et al.*, 2010; Panteli *et al.*, 2011; Volgin *et al.*, 2005).

The multimedia content annotation is also an active research topic, and the ontology-based approaches for annotations have seen intense interest recently (Bannour and Hudelot, 2011; Kallergi *et al.*, 2009; Schreiber *et al.*, 2001). The importance of the annotations of every type of content has led to the formation of a W3C working group that aims to provide an open annotation model standard (Sanderson *et al.*, 2013). The visualization of information on top of maps and other spatial plans has also gained attention due to several important applications ranging from the geovisualization of time-series data (Becker, 2009) and historical GIS (Berman, 2009) to event geovisualization (Tarantilis *et al.*, 2011) and urban planning facilitation (Phan and Choo, 2010).

In this paper we present MoM-NOCS, a Framework and a System for the Management of Mobile Nature Observations using Crowd Sourcing. The Framework supports the capturing of observations (like objects of interest or events) in nature, by (possibly naïve) users carrying any mobile device. During the capturing process, the system extracts rich contextual data (location, direction, etc.) using the device contextual capturing capabilities. The mobile users are provided with the ability to supply in real time or later their annotations for each observation, thus helping in the correct identification and understanding of the depicted species. The multimedia annotation services offer capabilities for identifying and annotating parts of images using graphical overlays. Interface mechanisms support and encourage crowd participation in the discussion about the species and observed events, as well as their correct identification via data provenance processes. The system also supports the spatiotemporal visualization of the observations and related time-varying statistics on top of maps.

The system that we are implementing is complementary and interoperable with biodiversity metadata management systems like BIOCASE (Berendsohn *et al.*, 2002) and GBIF (GBIF), as well as with systems that are managing the collections of the multimedia objects owned by natural history museums like MMAT (Makris *et al.*, 2013). MMAT is an integrated system that we have previously developed to allow publishing, semantically describing and managing nature objects of natural history museums that are captured in multimedia. The interoperability makes it possible for them to link to interesting observations in our system, and our observation visualization software to be used for visualizing the statistics on the observations that they manage. Thus our system can be used as an independent system for managing crowd observations of nature and their data provenance, which may be run by a region or country. It can also be used as a system that systematically manages and visualizes observations by museums. The museum personnel may be part of the crowd participating in the data provenance processes (with high authority and expertise that is supported by the system workflows).

The rest of the paper is structured as follows: The MoM-NOCS functionality is described in Section 2 and the MoM-NOCS conceptual model is presented in Section 3; The MoM-NOCS system architecture is described in Section 4, and a demonstration of the MoM-NOCS framework in a real-world scenario, including the capture and annotation of a flora specie observation, is outlined in Section 5. The paper concludes in Section 6, where we also discuss our future research directions.

2. The MoM-NOCS functionality

In this section we present the functionality of the MoM-NOCS integrated framework that we are developing, which allows the multimedia capturing and further annotation of observations made by domain experts and/or informed outsiders. It also allows offering advanced visualization services on the observations on top of maps and other spatial plans.

The MoM-NOCS system workflow is outlined in Subsection 2.1, followed by the MoM-NOCS functionality description in Subsection 2.2.

2.1 Workflows

The MoM-NOCS framework workflow is outlined in Figure 1. There are two types of individuals that may interact with the MoM-NOCS framework:

(1) the *experts*, who are essentially the scientific domain experts; and

(2) the *users*, who are the informed and interested outsiders.

The following paragraphs describe in detail all the activities that the framework users can perform:

- *Observation capturing*: Any user may create/edit observations. The capturing can be real-time, using a mobile application, or asynchronous, using a web-based system and uploading the captured multimedia content. An observation may be associated with any number of multimedia objects (an example can be an observation with multiple images alongside with audio).
- *Observation annotation*: All users may annotate the observations that have been captured by them or other MoM-NOCS users. This may be a social activity that allows the different users to complete the annotation of the observations, leading to the collaborative classification and characterization of the content of the multimedia objects associated with the observation. The observation annotation

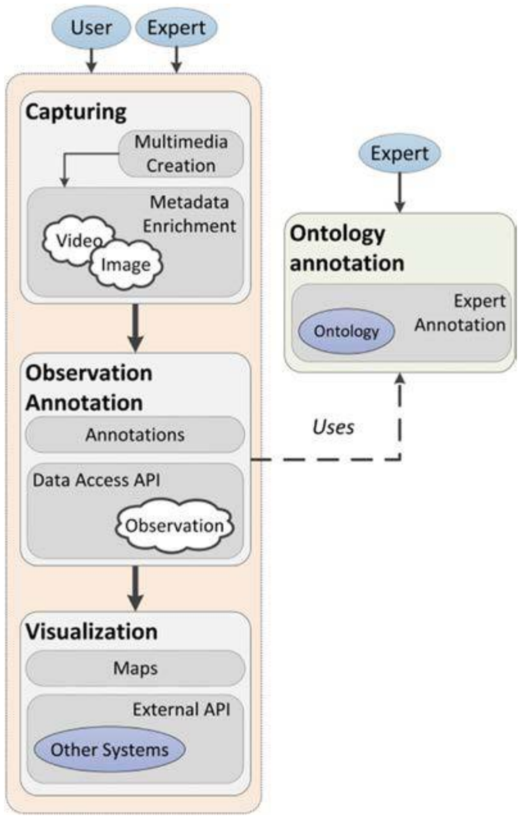


Figure 1.
The MoM-NOCS framework workflow

activity is assisted by an annotated ontology that allows the classification of the multimedia object content based on *taxonomic keys* (i.e. domain-specific criteria that allow the classification of the domain individuals). The users may annotate specific parts of the media objects using *selectors* to draw the attention to these parts (e.g. they may draw polygons or other geometric shapes around the different real-world objects contained in a picture).

- *Ontology annotation*: The MoM-NOCS experts may annotate the ontology that assists the observation annotation activity. The ontology may be annotated using multiple media to assist the users in the classification of the multimedia object content based on taxonomic keys. As an example, consider two species of the same plant that differ in the leaf shape. The ontology classes that represent these species should be annotated with images where the plant leaves can be easily recognized. These images should be also annotated: each of them should have a shape selector around the plant leaves and the selector should have a textual annotation that describes the leaf characteristics and emphasizes its distinguishing features.
- *Visualization*: All the MoM-NOCS users may visualize the observations stored in the MoM-NOCS repository on top of maps or other spatial plans.

2.2 The MoM-NOCS functionality

In this section we present the functionality of the MoM-NOCS framework. The operations presented below are grouped based on the subsystem that offers them (i.e. the *Annotation Subsystem* or the *Capturing and Visualization Subsystem*).

2.2.1 Annotation subsystem functionality. Ontology annotation: The expert users may annotate the ontology that assists the observation annotation. Using the system they can browse the ontology, select the classes they want to annotate, fill in the necessary fields (a small description, common name, etc.) and associate the class with relevant multimedia resources (that may be uploaded by the user or already existing in the system database). The users may also annotate the class with its *taxonomic keys*, which are the features of the real-world entity described by the class that distinguish the class instances from similar ones that belong to other classes (e.g. the shape and color of the leaves may be the taxonomic key that allows the users decide to distinguish "*Arum italicum*" from "*Arum creticum*"). Additionally, the annotation may include drawing the appropriate selectors around the taxonomic key if it is displayed in the visual resources (i.e. images, video).

Observation annotation: The MoM-NOCS users can annotate the existing observations using a web-based interface. This includes the annotation of the multimedia resources associated with the observation and, optionally, the identification of the observation contents. The multimedia resources can be tagged and selectors may be drawn around the resource contents. This annotation can lead to the identification of the species depicted in the observation.

Observation tagging: The MoM-NOCS users can also tag the multimedia resource contents and may refer to the taxonomic keys of the observation contents; they may also use tagging to spot the differences between species.

Observation identification/Commenting: The MoM-NOCS users can identify the contents of the multimedia resources contained in an observation. This may be achieved either by a single user, who tags the real-world objects with the names of the ontology classes that they belong, based on the taxonomic keys, or in collaboration with other users browsing the observations that exist in system. In the latter case, a user initiates the observation identification session, which is followed by the comments of the other users on his identification proposal. Furthermore, any user is allowed to leave a comment on any observation. This can be from a simple textual opinion, to an agree/disagree position about the identification proposals.

2.2.2 Capturing and visualization subsystem functionality. Media capturing: The MoM-NOCS users are allowed to create their observations either by uploading multimedia resources (text, images, audio, video) on a web-based system, or by using the MoM-NOCS mobile application for real-time observation capturing. The multimedia resources are associated (when possible) with the GPS coordinates of the observed real-world entities (or, approximately, of the media capturing location). For every observation, the user can capture/upload any number of relevant multimedia resources (an example can be an observation with multiple images from different angles, along with a video).

Observation description: After the capturing process, the MoM-NOCS users are asked to give a title and a brief description of the observation. They may also give additional descriptive tags, allowing for more efficient searching, as well as spatiotemporal information (GPS location, time, etc.). Whenever possible, the spatiotemporal data are automatically provided by the capturing device (e.g. newest mobile phones can be exploited to extract location and direction data). The description and title of the observation can optionally be verbal, allowing the users to create observations faster.

Path capturing: The MoM-NOCS users can create/edit observation paths. These can be suggested paths for other users to follow, or paths that they have walked while making observations. Both of these types will need a set of GPS points in sequence (which form a path node), and each node can be connected with observations in a specific place. For example, a user on a tour in a tropic forest will walk a long distance while making numerous observations that draw his attention. Afterwards, he can review them in the order that they were captured and view the path that he has followed. Other users can also review and possibly follow the captured path, making the same or additional observations, or change the path to a more complex one, incorporating more observations from others. Later they can follow it using the directions provided by the mobile application.

Browsing/Retrieval: The MoM-NOCS users can browse the existing observations. They may also exploit some basic social features, like browsing the observations made by friends or the most popular ones and find specific observations for species of interest, or for a given geographic area. This information can be presented as a single list or on top of interactive maps, according to their browsing and retrieval preferences. The system has also incorporated services that allow the browsing of data from other well-established sources, like Natural Europe (Natural Europe) and GBIF (GBIF). These services provide data generated from experts in professional institutes, thus giving valuable information to our end-users.

3. Conceptual model

In this section we present the conceptual model that we have developed for the representation of mobile nature observations in crowdsourcing environments and we formally describe the more important model classes (due to space limitations). An overview of the MoM-NOCS conceptual model is given in Figure 2. Our model allows the systematic integration of ontologies that capture domain-specific knowledge. These ontologies are composed of instances of the "Object" class (Figure 2). An "Object" instance *object(n, d, cn, u)* has a name *n*, is described in a textual or multimedia description *d*, has a set of common names *cn* and may be accessible from a set of URLs *u*. It may be compound (i.e. composed of other objects) and may be associated with semantic tags ("SemanticTag" instances, see below for details).

The "central" class in Figure 2 is the "Resource" class, which represents all the resources that are handled by the MoM-NOCS framework. A "Resource" instance *resource*(*resourceID*) is identified from the unique value of its *resourceID* attribute and may be associated with other class instances through the following relationships (notice that the syntax *R(n, c, src, trg*) describes a relationship having *n* as name, of cardinality *c*, which has *src* as source and *trg* as target):

- *R (hasTag, 1*, "Resource", "Tag"*), which states that a "Resource" instance may have one or more tags. The tags are represented by the "Tag" class. A specialization of the "Tag" class is the "SemanticTag" class, which allows associating an annotation with the constructs specified in the system ontology, as every "SemanticTag" instance is associated with an "Object" instance.
- *R (hasSelector, *, "Resource", "Selector"*), which states that a "Resource" instance may have any number of selectors that draw the attention on specific resource parts. The

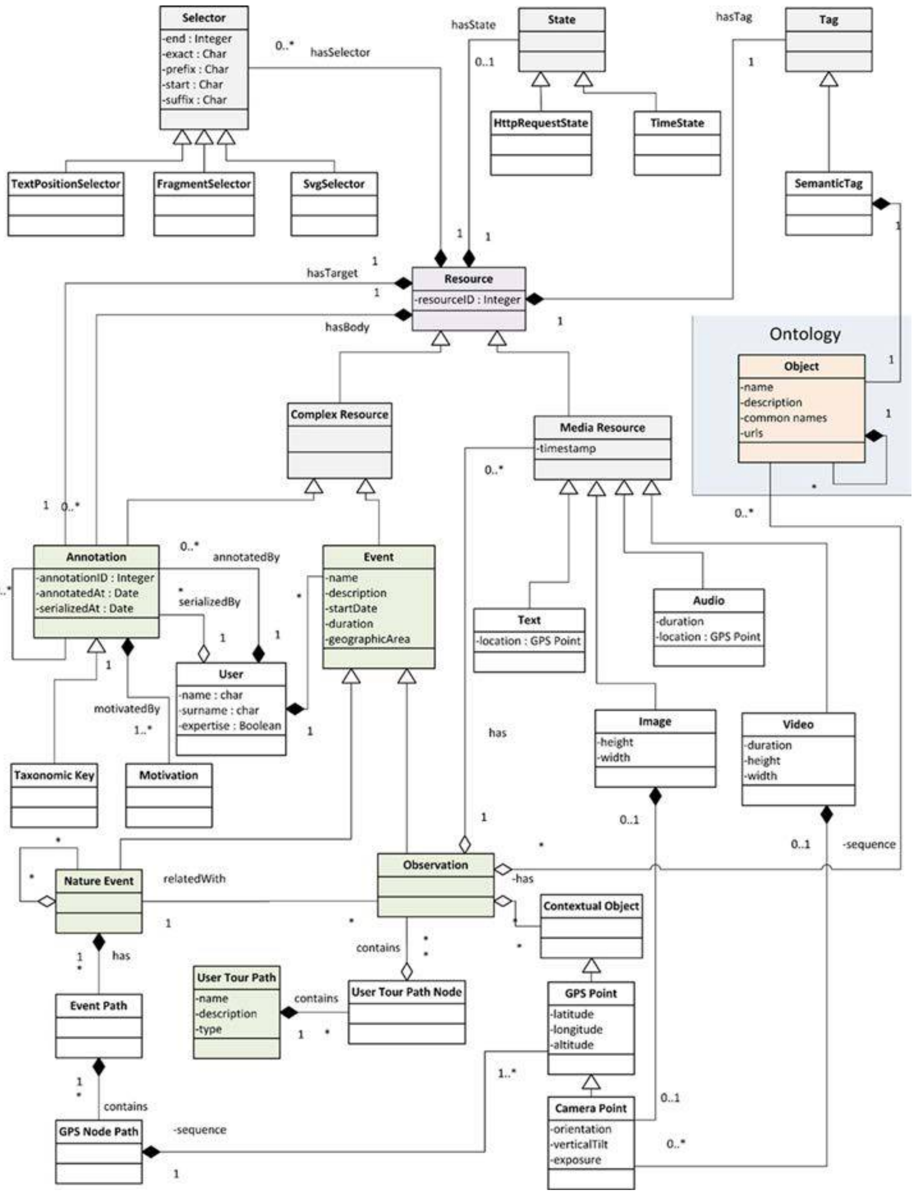


Figure 2.
UML class diagram that presents the conceptual model for the representation of mobile nature observations in crowdsourcing environments

selectors are represented by the "Selector" class and are further specialized in text selectors, media fragment selectors and SVG selectors (represented, respectively, by the *TextPositionSelector*, *FragmentSelector* and *SvgSelector* classes).

- *R (hasState, 1, "Resource", "State")*, which states that a "Resource" instance may be in a specific state. The states are represented by the "State" class.

- *R (hasBody, 1, "Annotation", "Resource")*, which states that a "Resource" instance may be the body of one or more annotations. The annotations are represented by the "Annotation" class (see below for details).
- *R (hasTarget, 1, "Annotation", "Resource")*, which states that a "Resource" instance may be the target of one or more annotations.

The different resource types are represented by the subclasses of the "Resource" class: The resources are distinguished in media resources (represented by the "Media Resource" class) and complex resources (represented by the "Complex Resource" class).

A "Media Resource" instance *mediaresource(resourceID, timestamp*) has been created in the *timestamp* time point. The "Media Resource" class is further specialized by the "Text", "Image", "Audio" and "Video" classes, which represent, respectively, the textual, image, audio and video resources.

A "Text" instance *text*(*resourceID, timestamp, location*) is a textual resource that was created in the *location* geographic point.

An "Audio" instance *audio*(*resourceID, timestamp, location, duration*) is an audio resource that was created in *location* and has a *duration* length.

An "Image" instance *image*(*resourceID, timestamp, height, width*) is an image resource that has *height* height and *width* width. It is associated with the location it was captured through the relationship *R (capturedAt, 1, "Image", "CameraPoint")*. The location where the image was captured is an instance of the "CameraPoint" class. An instance *camerapoint(longitude, latitude, altitude, orientation, verticalTilt, exposure*) of the "CameraPoint" class represents the picture-taking location that is geographically identified by *longitude, latitude* and *altitude* and also holds the picture-taking parameters *orientation, vertical tilt* and *exposure*.

A "Video" instance *video*(*resourceID, timestamp, height, width, duration*) is a video resource that comprises pictures of *height* height and *width* width and has a *duration* length. It is associated with the locations where its scenes were captured through the relationship *R (capturedAt, *, "Video", "CameraPoint")*.

The "Complex Resource" class has the subclasses "Annotation" and "Event".

The "Annotation" class represents the resource annotations that are specified and handled within the MoM-NOCS framework. An "Annotation" instance *annotation(annotationID, annotatedAt, serializedAt*) is identified from the unique value of its *annotationID* attribute, was created in the *annotatedAt* date and was serialized in the *serializedAt* date. An annotation may be nested (i.e. an annotation may include other annotations) and has, in addition to the relationships that associate it with its body and target resources, the following relationships:

- *R (motivatedBy, 1*, "Annotation", "Motivation")*, which states that an "Annotation" instance may be the target of one or more motivations. The motivations are represented by the "Motivation" class.
- *R (annotatedBy, 1, "Annotation", "User")*, which states that an "Annotation" instance has been created by a user. The users are represented by the "User" class.
- *R (serializedBy, 1, "Annotation", "User")*, which states that an "Annotation" instance has been serialized by a user.

The “Annotation” class is specialized by the “Taxonomic Key” class, which represents the taxonomic keys, a special type of ontology annotations that specify which are the differentiations between the real-world instances represented by different classes. Note that our model is fully compatible with the *Open Annotation Data Model* (Sanderson *et al.*, 2013) that is under development by the W3C Open Annotation Community Group.

All the events that are handled in the MoM-NOCS framework are represented in our model by the “Event” class. An “Event” instance *event(name, description, startDate, duration, geographicArea*) has a name *name*, is described in a textual or multimedia description *description*, has started in the time point *startDate*, has a *duration* length and has taken place in a geographic area *geographicArea*. It is also associated with a user that has created it through the relationship *R (createdBy, 1, “Event”, “User”*).

The events are distinguished in nature events and observations (represented, respectively, by the “NatureEvent” and “Observation” classes).

A “NatureEvent” instance *natureevent(name, description, startDate, duration, geographicArea*) may be compound (i.e. a nature event may include other nature events) and has the following relationships:

- *R (relatedWith, *, “NatureEvent”, “Observation”*), which states that a nature event may be related with any number of observations that may have been performed on the nature event.
- *R (has, *, “NatureEvent”, “EventPath”*), which states that a nature event may have any number of event paths, which are essentially sequences of observations related with the same event. The event paths are represented by the “EventPath” class. The “EventPath” instances are associated, through the *contains* relationship, with *GPS Node* instances that represent the path coordinates.

An “Observation” instance *observation(name, description, startDate, duration, geographicArea, location*) takes place in the *location* location and has the following relationships:

- *R (has, *, “Observation”, “MediaResource”*), which states that an observation may be associated with any number of media resources.
- *R (has, *, “Observation”, “Object”*), which states that an observation may be associated with any number of ontology classes.
- *R (has, *, “Observation”, “UserTourPathNode”*), which states that an observation may compose user tour path nodes (represented by the “UserTourPathNode” class) that form user tour paths. A user tour path is a path of observations in sequence and is represented by the “UserTourPath” class.
- *R (has, *, “Observation”, “ContextualObject”*), which states that an observation may be associated with any number of contextual objects, like GPS points and camera points.

4. Architectural overview

This section describes the overall architecture of the MoM-NOCS framework, identifies its basic components and provides an in-depth analysis of the internal functionality. Built as a web application, the system adopts the *Rich Internet Application (RIA*) principles, which promote the development of web applications as desktop applications performing business logic operations on the server side, as well as on the client side. The architecture distinguishes two subsystems, based on the functionality and the responsibilities that each of them provides. These are the *Capturing/Visualization* subsystem and the *Observation/Ontology Annotation* subsystem. The overall architecture is presented in Figure 3. The Capturing/Visualization subsystem handles the capturing of observations both from mobile devices and the web and the visualization of the observations on top of maps. The Observation/Ontology Annotation subsystem handles the collaborative annotation of the observations from multiple users, using the annotated ontology provided by the system, and provides the expert users with the means for the annotation and semantic enrichment of the MoM-NOCS ontology.

The following sections describe the architecture of the two subsystems. Subsection 4.1 describes the Capturing/Visualization subsystem, and Subsection 4.2 describes the Observation/Ontology Annotation subsystem.

4.1 Capturing/Visualization subsystem

The Capturing/Visualization subsystem comprises the *Client Side* and the *Server Side* logic. The Client Side logic operates within the web browser running on a user's local computer, while the Server Side logic operates on the web server hosting the application. The description of the architecture has been split in two parts; Subsection 4.1.1 describes the Client Side architecture, and Subsection 4.1.2 describes the Server Side architecture.

4.1.1 Client side architecture. The Client Side of the Capturing/Visualization subsystem is responsible for the interaction with the user. All the actions performed by an individual using the system are handled by the client side logic, which undertakes the presentation of the information as well as the communication with the server. To achieve a high level of decoupling between the components forming the client side logic, we adopted the Model View Controller (MVC) design pattern, as well as the Observer pattern.

The usage of the MVC pattern introduces the separation of the responsibilities for the visual display and the event handling behavior into different entities, named, respectively, the View and the Controller. Some of the advantages of this approach are:

- maximization of the code that can be tested with automation (web pages containing HTML elements are hard to test);
- code sharing between pages that require the same behavior; and

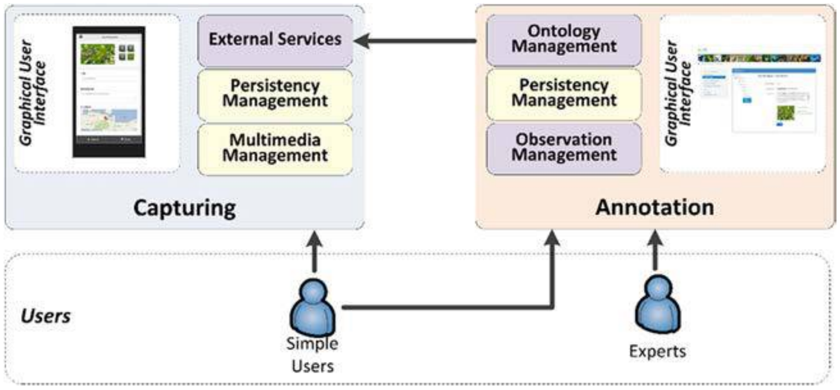


Figure 3. Overview of the MoM-NOCS system Architecture

- separation of Business logic from User Interface logic to make the code easier to understand and maintain.

As can be seen in Figure 4, the client side is composed of a number of distinct modules. These are described in more detail in the following paragraphs:

- *Model*: The Model encompasses all the business objects used by the system. When the system needs to present information about a business object, the client side requests the respective information. In turn, the server side services transfer the Model to the client side. When an update on the Model needs to be persistent, the client side sends the updated Model to the server side, initiating the corresponding procedures.
- *View*: In the MVC pattern, the view is responsible for all the information presentation. Each view controls a number of widgets on the user's web browser. It contains the Action Handlers, which listen to the user's actions, and the Templates that define the presentation of the widgets.
- *Controller*: The Controllers contain most of the client side logic of the Capturing/Visualization Subsystem. They are the modules that respond to the user input and define the Views that comprise the visual effect on the user interface. Additionally, they maintain the Model and change it appropriately. For every distinct visual

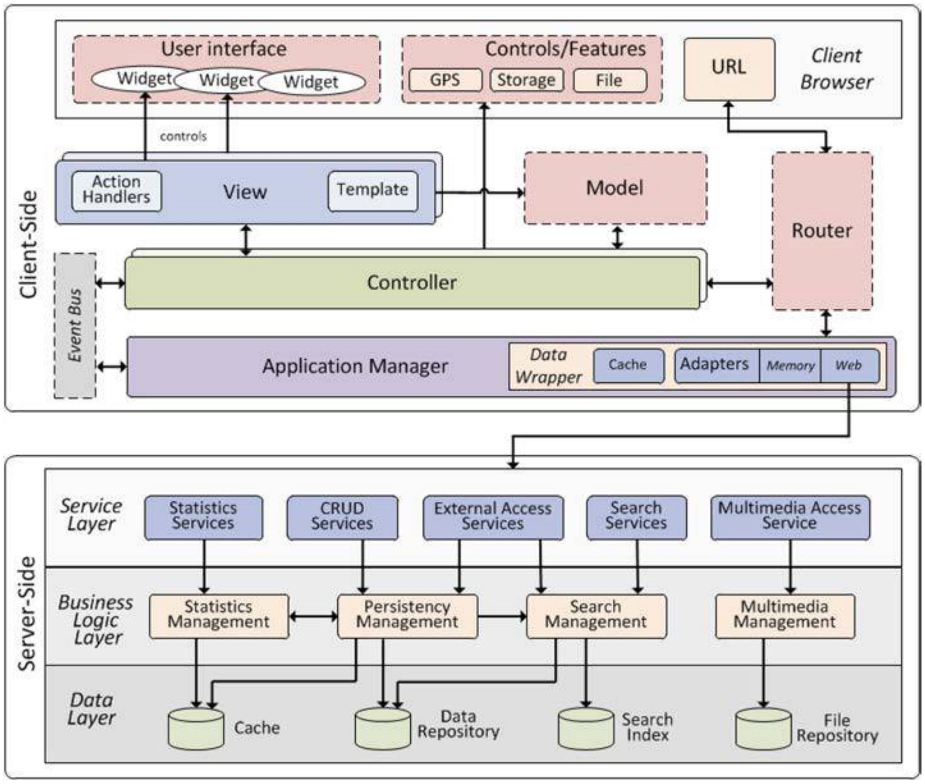


Figure 4.
Overview of the capturing/visualization subsystem architecture

interface of the system, there is a Controller, who in most cases handles a single view. Moreover, there are several cases where a "composite" Controller manages a number of other Controllers, creating complex interfaces. The Controllers also access the browser features that provide access to the Storage space, the File Uploads, the GPS location, etc.

- *Event Bus*: The Event Bus implements the Publish-Subscribe (Observer) pattern, enabling the decoupling of the user interface components. It is responsible for the message transmission between the entities that reside on the Client Side. It provides mechanisms for publishing events and subscribing to events.
- *Application Manager*: The Application Manager acts as a centralized point of control, handling the communication between the Controllers and the server side by making calls to the services exposed in the Service Layer, and notifying the Presenters for their responses. It contains the Data Wrapper, which implements the Mediator pattern, handling all the communication between the Controllers and the Server Side. Moreover, it enables the seamless caching of the data on the client side with the use of a dedicated cache. On each data request, the Data Wrapper checks if the data already exist in the cache before requesting them from the server, thus reducing the number of requests and speeding up the application execution.
- *Router*: The Router manages the URL of the client browsers. Its main purpose is to provide a different URL to each distinct interface, without raising a browser event that will force a reload on the whole page. Additionally, when the URL changes from other controls on the page (e.g. a click on a link), the Router analyzes the new location and handles the transition to the new interface. This contains a mapping between the different URLs supported in the system, as well as the Controller responsible for each mapped user interface.

4.1.2 Server side architecture. The Server Side part of our framework follows a multi-layered architectural pattern consisting of three basic layers: Service Layer, Business Logic Layer and Data Layer. This increases the reusability of the system components as well as the system maintainability, scalability, robustness and security. As shown in Figure 4, the server side comprises a number of distinct modules that will be described in the following sections:

(1) *Service Layer*: The Service Layer controls the communication between the client logic and the server logic, by exposing a set of services (operations) to the client side components. These services comprise the middleware concealing the application business logic from the client.

The basic system services are:

- *CRUD Services*: Facilitate the creation, retrieval, update and deletion of an observation, a multimedia object associated with an observation, a user etc.
- *External Access Services*: Provide the means for external systems to search and use the data of the system.
- *Search Services*: Provide the Client Side with the ability to search the data.
- *Multimedia Access Services*: Allow access to the uploaded multimedia files and their respective thumbnails.

- *Statistics Services*: Supply the usage statistics of the observations and the objects.

(2) *Business Logic Layer*: The Business Logic Layer, also known as Domain Layer, contains the business logic of the application and separates it from the Data Layer and the Service Layer. It comprises the following modules:

- The *Persistency Management Module*, which manages the access to the data on the repository.
- The *Statistics Management Module*, which manages the statistics of the system.
- The *Search Management Module*, which responds to the queries on the data with the appropriate results.
- The *Multimedia Management Module*, which manages the persistence and the delivery of the multimedia files, as well as their analysis and thumbnail generation.

(3) *Data Layer*: The Data Layer accommodates external systems that are used to store data accessed by it. Such systems are the Cache Server, which holds the cached information for faster access; the *Data Repository*, which holds all the data of the system; the Search Index allowing faster full-text queries; and the *File Repository*, persisting the multimedia files and the thumbnails.

4.2 Annotation subsystem

The Annotation subsystem comprises the *Client Side* and the *Server Side* logic. The Client Side logic operates within the web browser running on a user's local computer, while the Server Side logic operates on the web server hosting the application. The description of the architecture comprises two parts; Subsection 4.2.1 describes the Client Side architecture, and Subsection 4.2.2 describes the Server Side architecture. The implementation status as well as the technologies used are described in Subsection 4.2.3.

4.2.1 Client side architecture. The Client Side of the Annotation subsystem is responsible for the interaction with the user. All the actions performed by an individual using the system are handled by the client side logic, which undertakes the presentation of the information as well as the communication with the server when needed. To achieve the scalability of the User Interface, and the distinction between the components forming the client logic, we adopted the *Model-View-Presenter (MVP*) design pattern.

The usage of the MVP pattern allows us to decouple the client side architecture, introducing the separation of the responsibilities for the visual display and the event handling behavior into different entities, named, respectively, the view and the presenter. Some of the advantages of this approach are:

- maximization of the code that can be tested with automation (web pages containing HTML elements are hard to test);
- code sharing between pages that require the same behavior; and
- separation of the Business logic from the User Interface logic to make the code easier to understand and maintain.

The components in the system architecture that compose the MVP pattern are the *Model*, the *View* and the *Presenter*. The View is responsible for the user interface. It implements the Display Interface, whose members are technology striped UI elements of the view. The Presenter controls the behavior of the View. The user interacts with the View and all the user actions generate a User Interface event which is handled by the Presenter. The Model is used for all the logic operations of the Presenter, representing the business objects.

As can be seen in Figure 5, the client side of the Annotation subsystem comprises two distinct modules following the same architectural design. These are:

(1) *Observation annotation*: Handles the collaborative annotation of the observations from multiple users, using the annotated ontology provided by the system.

(2) *Ontology annotation*: Provides the expert users with the means for the annotation and semantic enrichment of the MoM-NOCS ontology.

Model: The Model encompasses all the business objects used by the system. When the system needs to present information about a business object, the client side requests the respective information. In turn, the server side services transfer the Model to the client side. When an update on the Model needs to be persistent, the client side sends the updated Model to the server side, initiating the corresponding procedures.

View: In the MVP pattern, the view is responsible for all the information presentation (colors, borders, UI layout) and the management of the controls on a page or part of a page. Both the HTML pages and the StyleSheets are handled by views.

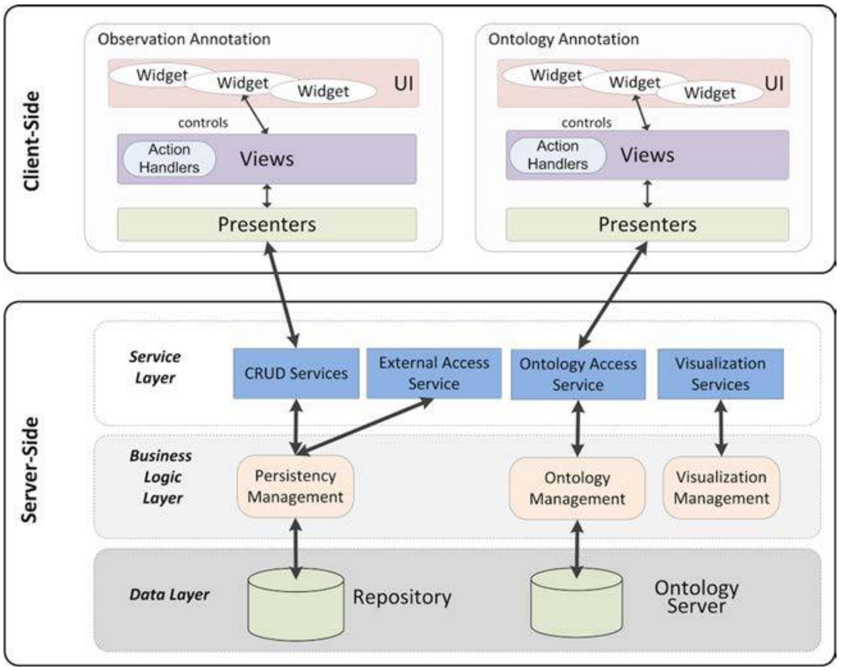


Figure 5.
Overview of the annotation subsystem architecture

Presenter: A Presenter contains the client side logic of our application. It is the key entity in the MVP design pattern, for it is a bridge between the Model and the View. As a general rule, for every View there is a presenter that instantiates and manipulates it, but there might be cases where one Presenter needs to control multiple Views. The presenter has handlers for the events that take place in his view/s or in the application in general. The life cycle of a presenter is composed of four distinct phases, in the respective order:

(1) *Initialization phase*: The presenter initializes its corresponding view/s, and instructs them where to deploy. In addition, the presenter registers handlers to the crucial UI events and application events. When the initialization and handler registration are completed, the presenter goes into the sleeping phase.

(2) *Sleeping phase*: This is the phase where the presenter does nothing but waiting for a UI event from the view, or an application event.

(3) *Operating phase*: When an already registered event is raised, the presenter wakes up and handles it; it contacts the services on the server side, updates the view or raises new events.

(4) *Termination phase*: When the user chooses to leave the application, all the presenters go into the termination phase, the final stage before they are destroyed. During this phase, the presenter might need to inform the server for the termination of any ongoing processes.

As already stated, the presenter has no knowledge of any widget-based code inside the view and all the interaction with the view is done through a number of interfaces, achieving the decoupling between the two.

4.2.2 Server side architecture. The Server Side part of the Annotation subsystem follows a multi-layered architectural pattern consisting of three basic layers: Service Layer, Business Logic Layer and Data Layer.

The advantages of this approach include the increased system maintainability, reusability of the system components, scalability, robustness and security. Moreover, the adoption of a multitier architecture and the provision of well-defined interfaces for each layer allows any of the layers to be upgraded or replaced independently (with minimum effort) as requirements or technology change.

(1) *Service Layer*: The *Service Layer* controls the communication between the client logic and the server logic, by exposing a set of services (operations) to the client side components. These services actually comprise the middleware concealing the application business logic from the client.

The basic system services are listed below:

- *CRUD Services*: Facilitate the creation, retrieval, update and deletion of annotations.
- *External Access Services*: Provide the means for other, well-described, external application to search and use the data of the system.
- *Visualization Services*: Responsible for supporting all the visualization functionality.
- *Ontology Access Service*: Provides access to the species ontology.

(2) *Business Logic Layer*: The *Business Logic Layer*, also known as *Domain Layer*, contains the business logic of the application and separates it from the Data Layer and the Service Layer, which are used by the Client Side modules. It consists of following basic modules:
   - The *Persistency Management Module*, which manages the access to the repository data.
   - The *Visualization Management Module*, which is used for the visualization of the observations along with their annotations.
   - The *Ontology Management Module*, which manages a classification of the species stored in the ontology server.

(3) *Data Layer*: The *Data Layer* accommodates external systems that are used to store data accessed by it. Such systems are the *Repository*, which holds all the data structured according to our conceptual model, and the *Ontology Server*, which holds all the information about the system ontology.

4.2.3 Implementation status and technologies. The system is currently in implementation phase, with most of the functionality already in final state. The client side is based on the latest web-application standards and the server side relies on the Java technology. The client side application has also been packaged as a native mobile application with the use of PhoneGap (PhoneGap), making it compatible with all the major mobile platforms. This allows the application to run without the need of Internet access, or the loading of its source each time that the user loads it. Additionally, PhoneGap allows us to provide more functionality by using various native platform features that are otherwise unavailable to web applications.

The client side is built with *JavaScript*, *HTML5* and *CSS3*. For the code organization of the client side, we have used a number of libraries/frameworks.

Backbone.js has been used to structure the system as an MVC-compatible application. The use of Backbones models, collections, events and views provides the application with a very basic "solid" structure. The models and the collections allow the application objects to act as object-oriented classes with key-value binding and custom listeners on each of the properties. Moreover, the events allow custom events to be broadcasted from any part of the application, reducing the coupling of the components. Finally, the views allow the creation of independent widgets and their composition into complex user interfaces.

Requirejs has been used to allow the organization of the code in smaller files and the management of the dependencies for each JavaScript file. This introduces modular programming to the client side, breaking the large JavaScript files into smaller blocks of manageable code. This eases the maintenance effort and increases reusability. However, this introduces the dependencies between the smaller blocks of code. The management of these dependencies between the modules is handled in full by RequireJS. The modules are defined in separate files along with their dependencies, and RequireJS takes on the responsibility to provide each module with the required dependencies at run time. Modular loading also allows incremental "lazy" loading of the different parts of the application at the time that they are needed, instead of loading all the JavaScript code at loading time. This can have a big impact on application loading and running, especially in recent web applications that rely heavily on JavaScript.

Leaflet has been used to provide support for maps. It is a minimal JavaScript library, optimal for usage on mobile phones. The maps that are used here are provided by OpenStreetMaps.

Finally, *Node.js*[1] and *Grunt*[2] have been used to automate the necessary tasks during the development and deployment stages. These involve the compilation and compression (minification) of the source code, the unit testing and the static source code checking. Node.js allows the execution of JavaScript just as any other scripting language on any machine. This enables Grunt to automate every part of the development process. Apart from the aforementioned purposes, Grunt is also used for the compilation of the application to the various formats supported by PhoneGap.

The server side (backend) of the system has been based on the Java programming language. In particular, we used the Spring Framework extensively. The main features of Spring that we used were the Inversion of Control container, the Spring Data framework, Spring Web and Spring Security.

For the persistence of the data, we have chosen a document database, *MongoDB*. MongoDB is a non-relational (NO-SQL) database, which allows us to represent the data of the system as JSON documents, while providing rich query expressivity and selectivity. We have split the model in a number of entities, with each entity represented by a collection in MongoDB. The most important of these entities are Observation, Object, Multimedia, User and Statistics. Apart from JSON documents, MongoDB can also persist blobs (files), which allowed us to put the multimedia files and the thumbnails along with model collections.

For indexing and searching purposes, the data are also pushed into an Elasticsearch[3] instance. *Elasticsearch* is based on Lucene, providing a distributed, multitenant-capable full-text search engine with a RESTful web interface and schema-free JSON documents. Every part of the data that is searchable is indexed in an Elasticsearch cluster.

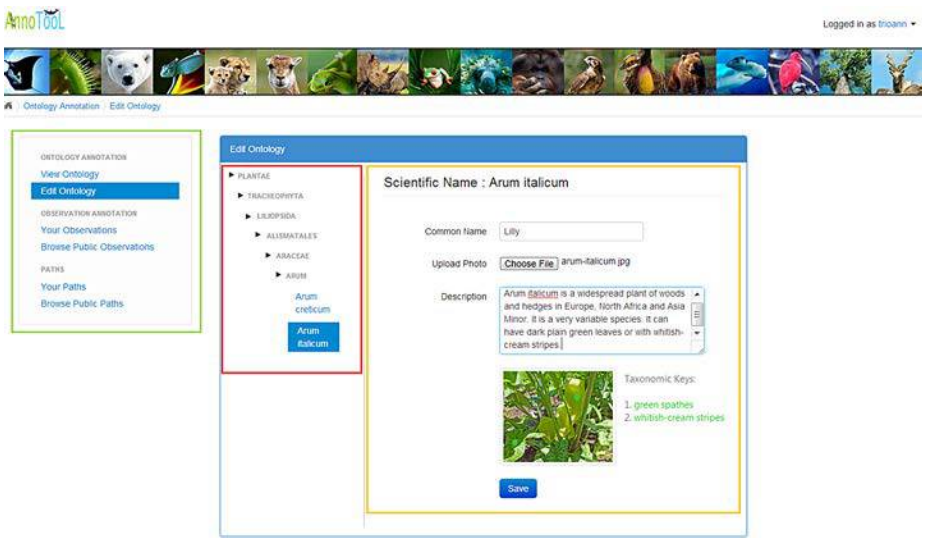
Another very important part of the server side architecture is the caching system. To this end, we have used the *Redis* server, which is an open-source, networked, in-memory, key-value data store. A very important feature that it provides is the optional durability to any data that it holds. This allows us to specify the maximum size of main memory that we want to allocate to the cache, while Redis handles the removal of the least used elements when the limits are reached. The usage of the server cache increases the response time of the application when the clients request objects that are in the cache.

5. MoM-NOCS in action

A demonstration of the MoM-NOCS framework in a real-world scenario is outlined in this section. This includes the capture and annotation of a flora specie (*Arum italicum*) observation, as well as the visualization of relevant observations.

We have already mentioned that the users may be assisted by the system ontology during observation annotation. This ontology is annotated by the expert users, using the *AnnoTool* component, which allows both ontology and observation annotation and is depicted in Figure 6.

The snapshot in Figure 6 shows the annotation of the *Arum italicum* class. The user has selected to perform the ontology annotation operation through the left-side menu, on which we draw the attention with the green rectangle. Notice that the user has browsed



Scientific Name : Arum italicum

Common Name: Lilly

Upload Photo: arum-italicum.jpg

Description: Arum italicum is a widespread plant of woods and hedges in Europe, North Africa and Asia Minor. It is a very variable species. It can have dark plain green leaves or with whitish-cream stripes

Taxonomic Keys:

1. green spathes
2. whitish-cream stripes

Figure 6.
The MoM-NOCS ontology annotation interface

the ontology, as is shown in the area designated by the red rectangle, and has selected the *Arum italicum* class. Then, in the area that is surrounded by the yellow rectangle, he has filled in the necessary fields (a small description, common name, etc.) and has associated the class with a relevant image. He has also annotated the class with its *taxonomic keys*, which are the features of the real-world entity described by the class that distinguish the class instances from similar ones that belong to other classes: green spathes, and whitish-cream stripes. This annotation includes drawing the appropriate numbered selectors around the taxonomic keys in the *Arum italicum* picture.

Consider now a naturalist who is on an excursion, during which some lilies in a garden draw his attention. He decides to take videos or photos of the plant using the MoM-NOCS capturing subsystem, and in particular the mobile application that he has already been installed on his phone. Using this application, he also gives a title and a brief description outline of what he is observing, either in textual or in speech form. The mobile user interface for observation capturing and description is presented in Figure 7. As is shown in Figure 7, the user is presented not only with the title and the description he has given, but he also views the multimedia resources associated with the observation and his location on the surrounding area map.

When the naturalist is at home, he will be able to make a review of the photos and videos that he has captured and annotate his observation using *AnnoTooL*, the MoM-NOCS web application for ontology and observation annotation. He may also want to classify the herb depicted in the captured material – essentially to locate the herb class it belongs.

Let the user superficially decide that the flower that he has observed is *Arum creticum*, as *Arum creticum* and *Arum italicum* have several similarities and he has not exploited all the functionality offered by the MoM-NOCS framework (he could have been assisted in the classification task by the annotated ontology). Once the annotation of the observation is complete, the user submits it so that other users may view it, comment on it and express their opinion on the classification.

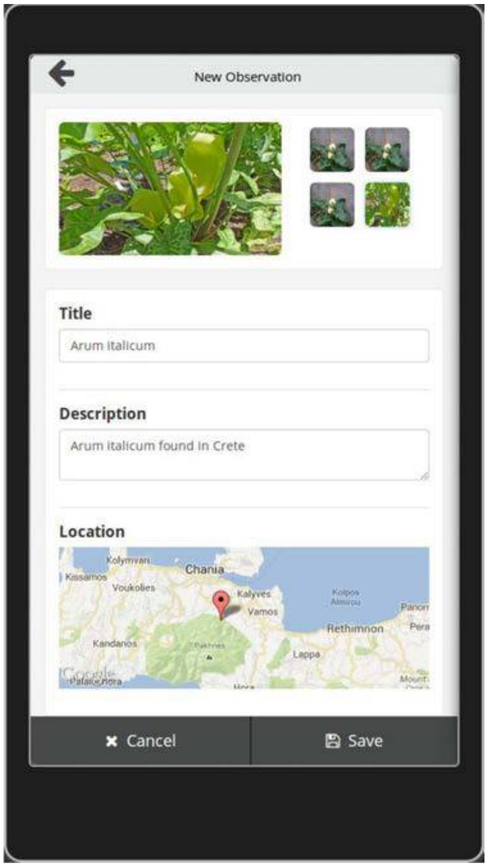
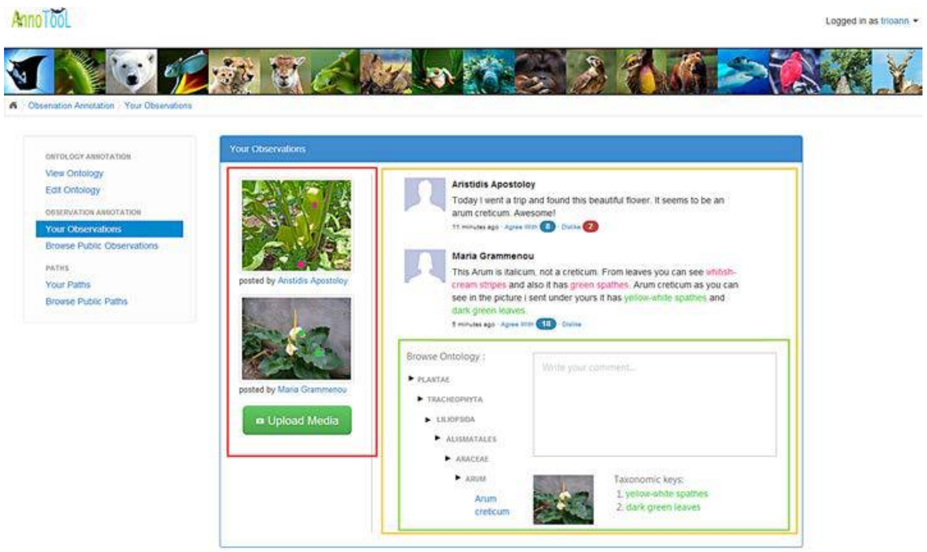


Figure 7.
The MoM-NOCS capturing interface

Consider another user, who reviews the annotation and believes that the observation is not correctly classified, as the flower depicted in the multimedia material is *Arum italicum*. To be able to argue on her opinion, she browses the class hierarchy of the ontology. As the ontology is annotated with multimedia material, she views characteristic images of both *Arum italicum* and *Arum creticum* as well as descriptions of the taxonomic keys of the two flowers. Then, she comments on the annotation, stating that she believes that it is an *Arum italicum*, based on the characteristic photo of *Arum creticum* and the taxonomic keys, as is shown in the area of Figure 8 that the green rectangle draws the attention. Notice that she refers to the taxonomic keys in her comment and she uses selectors to draw the attention to the taxonomic keys.

Other users can also express their opinion below the existing conversations, agreeing or disagreeing with each other as well as leaving their comments. When there is enough information provided, the system proposes a probable classification for the observation contents based on the number of people agreeing for a classification and their authority level. The user reputations are taken into account in the classification proposal provided



Aristidis Apostoloy
Today I went a trip and found this beautiful flower. It seems to be an arum creticum. Awesome!

Maria Grammenou
This Arum is italicum, not a creticum. From leaves you can see whitish-cream stripes and also it has green spathes. Arum creticum as you can see in the picture i sent under yours it has yellow-white spathes and dark green leaves.

Notes: The red rectangle draws the attention on the available observations. The annotation area is designated by the yellow line and the green line shows the ontology and taxonomic key support

Figure 8.
The MoM-NOCS observation annotation interface.

by the system. In particular, the annotations, comments and votes made by users with a "good" annotation/classification history are weighed higher than the ones made by users with a poor annotation/classification history.

Once the observations are captured and annotated, the users may view the (geo)visualizations of the observations of interest that are in their surrounding area using the web interface or their mobile device. An example of this can be seen in Figure 9, where we present the geovisualization of the *Arum italicum* in the area near the current user location.

The screenshot is taken from the mobile application, and the user can see the nearby observations that have been made by others (denoted by the placeholders on which we draw the attention with a light blue curve that has been drawn around them in Figure 9), as well as his current position and direction on the map (denoted by the light blue triangle on which we draw the attention with a red circle in Figure 9). This way, he can navigate to the exact position of an observation made by another user (in the case that the observation subject is not moving, like a plant), or approach an area, knowing that there is high possibility to find the desired species.

6. Conclusions – future work

In this paper, we have presented *MoM-NOCS (Capturing, Annotation and Visualization)*, an integrated framework that allows the mobile capturing of multimedia observations by crowds of people interested in a domain as well as the further annotation of the observations by the crowd members. It also allows offering advanced visualization services on the observations on top of maps and other spatial plans. The prototype application implementation is carried out in the biodiversity domain, where the capturing process is extremely slow due to the limited number of experts and

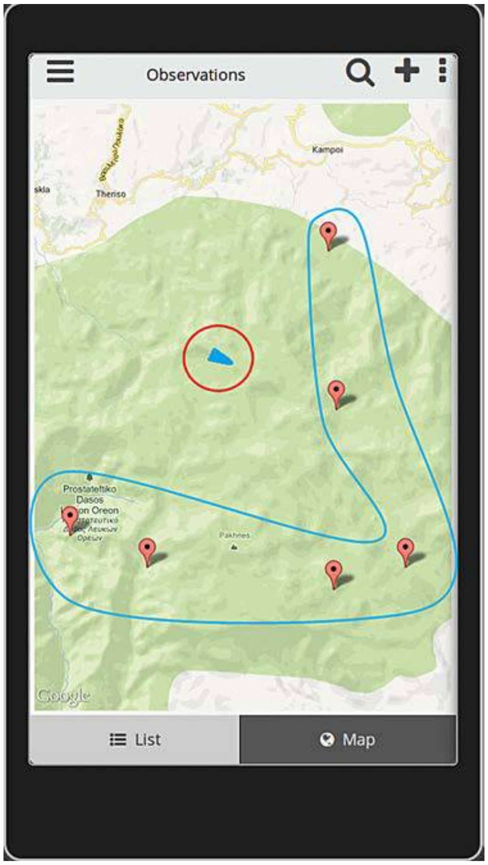


Figure 9.
Geovisualization of the *Arum italicum* observations on a mobile device

funding. The system has been designed to be interoperable with existing metadata services in the biodiversity domain and with collection management systems in natural history museums.

Several mobile multimedia applications and services that use the infrastructure described in this paper for different user communities are currently developed.

Our future work includes conducting a survey to potential users, to get some feedback both for the system functionality and the user interface design. It also includes the experimentation with different rating schemes for the determination of the user reputation regarding annotation and/or classification.

Notes

1. http://nodejs.org/
2. http://gruntjs.com/
3. http://www.elasticsearch.org/

References

Bannour, H. and Hudelot, C. (2011), "Towards ontologies for image interpretation and annotation", in *Proceedings of the 9th International Workshop on Content-Based Multimedia Indexing (CBMI)*, Madrid, pp. 211-216.

Becker, T. (2009), "Visualizing time series data using web map service time dimension and svg interactive animation", Master thesis, International Institute for Geo-Information Science and Earth Observation (ITC), Enschede.

Berendsohn, W., Döring, M., Gebhardt, M. and Güntsch, A. (2002), "BioCase – a biological collection access service for Europe", *Tech. Rep.*, available at: http://www.biocase.org/

Berman, M. (2009), "Modeling and visualizing historical GIS data", in *Proceedings of the Spatio-Temporal Workshop, Harvard University*, *Cambridge, MA*.

Christodoulakis, S., Foukarakis, M., Ragia, L., Uchiyama, H. and Imai, T. (2010), "Picture context capturing for mobile databases", *IEEE Multimedia*, Vol. 17 No. 2, pp. 34-41.

Global Biodiversity Information Facility (GBIF). available at: http://www.gbif.org/

Kallergi, A., Bei, Y. and Verbeek, F.J. (2009), "The ontology viewer: facilitating image annotation with ontology terms in the CSIDx imaging database", *Proceedings VISSW, CEUR-WS Proceedings*, *Workshop on Visual Interfaces to the Social and the Semantic Web (VISSW2009), IUI2009, Feb 8 2009, Sanibel Island, Florida*, Vol. 4430.

Makris, K., Skevakis, G., Kalokyri, V., Arapi, P. and Christodoulakis, S. (2013), "Metadata management and interoperability support for natural history museums", in *Proceedings of the 17th International Conference on Theory and Practice of Digital Libraries (TPDL'13)*, *Valletta*, *September*, pp. 120-131.

Panteli, A., Tsinaraki, C., Ragia, L., Kazasis, F. and Christodoulakis, S. (2011), "Mobile multimedia event capturing and visualization (MOME)", in *The Proceedings of FTRA Conference Multimedia and Ubiquitous Engineering (MUE 2011), FTRA*, *Loutraki*, pp. 101-106.

Phan, V.T. and Choo, S.Y. (2010), "A combination of augmented reality and google earth's facilities for urban planning in idea stage", *International Journal of Computer Applications (0975-8887)*, Vol. 4 No. 3, pp. 26-34, PhoneGap, available at: http://phonegap.com/

Sanderson, R., Ciccarese, P. and Van de Sompel, H. (Eds) (2013), *Open Annotation Data Model, W3C Community Draft*, available at: http://www.openannotation.org/spec/core/

Schreiber, A.T., Dubbeldam, B., Wielemaker, J. and Wielinga, B. (2001), "Ontology-based photo annotation", *IEEE Intelligent Systems*, Vol. 16 No. 3, pp. 66-74.

Tarantilis, N., Tsinaraki, C., Kazasis, F., Gioldasis, N. and Christodoulakis, S. (2011), "Evisuge: event visualization on google earth", in *Proceedings of the 6th International Workshop on Semantic Media Adaptation and Personalization (SMAP)*, *Pontevedra*, pp. 68-73.

Tsinaraki, C. and Christodoulakis, S. (2006), "A multimedia user preference model that supports Se-mantics and its application to MPEG 7/21", in *The Proceedings of the IEEE Multimedia Modeling 2006 Conference (IEEE MMM 2006)*, *Beijing*, pp. 35-42.

Volgin, O., Hung, W., Vakili, C., Flinn, J. and Shin, K.G. (2005), "Context-aware metadata creation in a heterogeneous mobile environment", in *The Proceedings of the International Workshop on Network and Operating System Support for Digital Audio and Video*, *Stevenson, WA*, pp. 75-80.

Further reading

The Natural Europe Project. available at: www.natural-europe.eu/

Potel, M. (1996), *MVP: Model-View-Presenter, 'The Taligent Programming Model for C++ and Java'*, Taligent, available at: http://www.wildcrest.com/Potel/Portfolio/mvp.pdf

About the authors

Giannis Skevakis is a Research Assistant at the Laboratory of Distributed Multimedia Information Systems and Applications of the Electronic and Computer Engineering Department of the Technical University of Crete. He holds a diploma and a Master of Science degree in electronic and computer engineering from the Technical University of Crete. Giannis Skevakis is the corresponding author and can be contacted at: skevakis@ced.tuc.gr

Chrisa Tsinaraki received her PhD in 2008 from the Technical University of Crete. She has worked as a Postdoc Researcher and a Visiting Lecturer in the Technical University of Crete and as a Postdoc Researcher in the University of Trento and is now working, as a Postdoc Researcher, in the Joint Research Center of the European Union.

Ioanna Trochatou is a Graduate Student at the Department of Electronic and Computer Engineering at the Technical University of Crete. Previously, she earned BSc degree in computer science from the University of Crete.

Stavros Christodoulakis is a Full Professor in the School of Electronic and Computer Engineering, Technical University of Crete, and Director of the MUSIC/TUC lab. His research interests cover the areas of information systems, databases, multimedia, web and mobile architectures, applications and services. He is the author or co-author of many scientific papers in those fields. He is in the Scientific Board of the Institute of Telecommunications of Crete, and has been in the Scientific Board of the Institute for Computer Research (ICR) of the University of Waterloo, and several other research institutes, organizations and international scientific journals, as well as member or chair of many international conferences in his areas of expertise.

Professor Christodoulakis has a PhD degree in computer science from the University of Toronto, and he has been Professor in the Departments of Computer Science of the University of Toronto and University of Waterloo, Canada.