OntoWiki Mobile – Knowledge Management in Your Pocket

Timofey Ermilov, Norman Heino, Sebastian Tramp, and Sören Auer

Universität Leipzig, Institut für Informatik, AKSW,
Postfach 100920, D-04009 Leipzig, Germany,
`lastname@informatik.uni-leipzig.de`
`http://aksw.org`

Abstract. As comparatively powerful mobile computing devices are becoming more common, mobile web applications have started gaining in popularity. In this paper we present an approach for a mobile semantic collaboration platform based on the OntoWiki framework. It allows users to collect instance data, refine the structure of knowledge bases and browse data using hierarchical or faceted navigation on-the-go even without a present data connection. A crucial part of OntoWiki Mobile is the advanced replication and conflict resolution for RDF content. The approach for conflict resolution is based on a combination of distributed revision control strategies and the EvoPat method for data evolution and ontology refactoring. OntoWiki mobile is available as an HTML5 Web application and can be used in scenarios where semantically rich information has to be collected in field-conditions such as during bio-diversity expeditions to remote areas.

1 Introduction

As comparatively powerful mobile computing devices are becoming more common, mobile web applications have started gaining in popularity. Mobile web applications such as *Google Mail* or *Calendar* are already in use everyday by millions of people. Some of these applications already use the Semantic Web technologies and information in the form of RDF (e. g. TripIt). An important feature of these applications is their ability to provide *offline functionality* with local updates for later synchronization with a web server. The key problem here is the reconciliation, i. e. the problem of potentially *conflicting updates* from *disconnected clients.*

Another problem current mobile application developers face is the plethora of mobile application development platforms as well as the incompatibilities between them. *Android* (Google), *iOS* (Apple), *Blackberry OS* (RIM), *WebOS* (HP/Palm), *Symbian* (Nokia) are popular and currently widely deployed platforms, with many more proprietary ones being available as well. As a consequence of this fragmentation, realizing a special purpose application, which works with many or all of these platforms is extremely time consuming and inefficient due to the large amount of duplicate work required.

The W3C addressed this problem, by enriching HTML in its 5th revision with access interfaces to local storage (beyond simple cookies) as well as a number of devices and sensors commonly found on mobile devices (e. g. GPS, camera, compass etc.). We argue, that in combination with semantic technologies these features can be used to realize a *general purpose*, mobile collaboration platform, which can support the long tail of mobile special interest applications, for which the development of individual tools would not be (economically) feasible.

In this paper we present the *OntoWiki Mobile* approach realizing a mobile semantic collaboration platform based on the OntoWiki framework [2]. It comprises specifically adopted user interfaces for browsing, faceted navigation as well as authoring of knowledge bases. It allows users to collect instance data and refine the structured knowledge bases on-the-go. OntoWiki Mobile is implemented as an *HTML5 web application*, thus being completely mobile device platform independent. In order to allow offline use in cases with restricted network coverage (or in order to avoid roaming charges) it uses the novel HTML5 local storage feature for replicating parts of the knowledge base on the mobile device. Hence, a crucial part of OntoWiki Mobile is the advanced conflict resolution for RDF stores. The approach is based on a combination of the EvoPat [8] method for data evolution and ontology refactoring along with a versioning system inspired by distributed version control systems like Git.

There are already a number of mobile semantic applications ranging from semantic backend services [11] for mobile devices to applications covering very specific use cases (e. g. *DBpedia Mobile* [1] or *mSpace Mobile* [14]). OntoWiki Mobile, however, is a generic, application domain agnostic tool, which can be utilized in a wide range of very different usage scenarios ranging from instance acquisition to browsing of semantic data on the go. Typical OntoWiki Mobile usage scenarios are settings where users need to author and access semantically structured information on the go or in settings where users are away from regular power supply and restricted to light-weight equipment (e. g. scientific expeditions).

The paper is structured as follows: We outline the general architecture of OntoWiki Mobile in Section 2. We describe our replication and reconciliation strategy in Section 3. The OntoWiki Mobile user interface and the implementation of browsing and authoring in restricted mobile environments is presented in Section 4. A description of a use case for OntoWiki mobile in the domain of field expeditions in bio-diversity research is presented in Section 5. We give an overview on related work in Section 6 and conclude with an outlook on future work in Section 7.

2 Architecture

OntoWiki was developed to address the need for a Web application for rapid and simple knowledge acquisition in a collaborative way. OntoWiki can be used for presenting, authoring and managing knowledge bases adhering to the RDF data model. In order to render its functionality, OntoWiki relies on several APIs

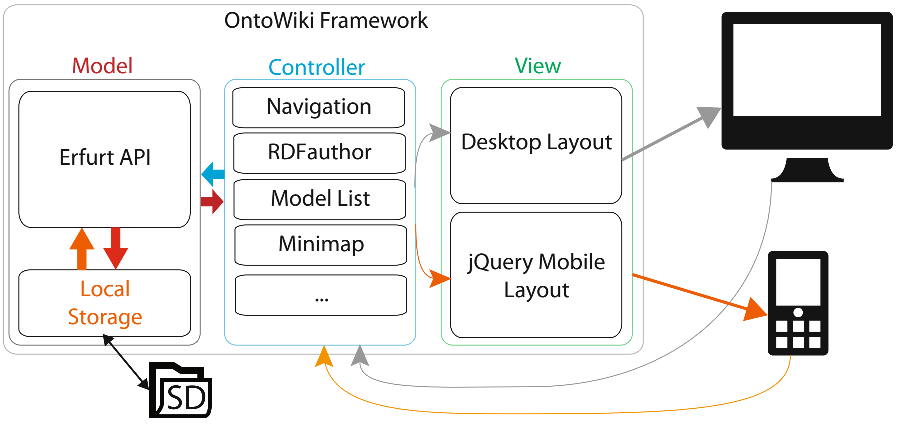


Fig. 1. OntoWiki Mobile architecture

that are also available to third-party developers. Usage of these programming interfaces enables the users to extend, customize and tailor OntoWiki in several ways. OntoWiki's architecture consists of three separate layers: persistence layer, application layer and user interface layer. Those layers represent the standard MVC[1] architecture. The persistence layer consists of the Erfurt API which provides an interface to different RDF stores (e. g. Virtuoso, MySQL). Content in OntoWiki is rendered through the templates (user interface layer). The controller action serving the request renders its output in a template. OntoWiki as a Web application is based on the Zend Framework[2] which lays out the basic architecture and is primarily responsible for request handling. Such architecture allows to easily extend the functionality of OntoWiki and change the layout based on context parameters of the user request. OntoWiki Mobile is based on the OntoWiki Framework. It utilizes all of the described OntoWiki Framework architecture and tailors it to better fit mobile usage scenarios by e. g. replacing with mobile-specific layout (see Figure 1).

The mobile user interface was built using HTML5 and the jQuery Mobile[3] framework which includes the core jQuery library in an improved version to ensure compatibility across all of the major mobile platforms. Built on a jQuery and jQuery UI foundation, it allowed us to create a unified user interface regardless of the actual platform the user's device runs on. The resulting source code presents a thin JavaScript layer, built with Progressive Enhancement principles so as to allow for a minimal footprint.

To access the device's hardware (e. g. camera, GPS sensor) OntoWiki Mobile uses the extended HTML5 API[4]. Geolocation API is used from JavaScript to

[1] Model-View-Controller.
[2] `http://framework.zend.com/`
[3] `http://jquerymobile.com/`
[4] `http://w3.org/TR/html5/offline.html#offline`

get user's current latitude and longitude. Local storage is a part of the HTML5 application caches and is a persistent data storage of key-value pair data in Web clients. It is used to store replicated parts of knowledge bases at the client-side. OntoWiki Mobile stores RDF data as JSON-encoded strings for offline usage and to increase page loading speed while online (e. g. in cases where the resource has not been changed and does not require reloading). Attached photograph are stored to local cache using HTML5 Canvas base64 encoding. Usage of local storage allows to export and import user-gathered data, for example to do back-ups (snapshots) of data to external SD card or to share data with other mobile devices via bluetooth.

Resource editing in OntoWiki is done using RDFauthor [12]. The system makes use of RDFa-annotations in web views in order to make RDF model data available on the client. Embedded statements are used to reconstruct the graph containing statements about the resource being edited. A set of editing widgets, tailored to specific editing tasks and equipped with end-user supporting functionalities (e. g. resource autocompletion from OntoWiki and Sindice) are selected based on the statements contained in the graph. In OntoWiki Mobile, RDFauthor has been adapted to better cope with mobile environments by adapting the user interface and introducing lazy script loading.

Data replication and conflict resolution is the most complex part of the OntoWiki Mobile. The process consists of three steps, handled by separate components (explained in more detail in Section 3):

– a client-side replication component utilizing HTML5 local storage,
– a server-side replication component and
– a server-side conflict resolver.

The conflict resolver uses additional mechanisms to simplify merging concurrent edits of the same resource. The first one is policy-based semi-automatic merging tool that utilizes the EvoPat engine – an OntoWiki extension for dealing with evolution of knowledge bases using patterns [8]. Evolution patterns in EvoPat consists of variables, a SPARQL query template and a SPARQL/Update query with functional extensions. Results of the SPARQL query are bound to variables which in turn are used in SPARQL/Update queries to perform knowledge base transformations. OntoWiki Mobile uses specifically created patterns that can be applied by the user. The second mechanism provides a user interface for manual conflict resolution. It allows the user to select which statements from different version to include in a merged version of a resource.

3 Replication

One critical requirement for OntoWiki Mobile was the ability to work without an Internet connection. In cases where several users edit the same resources without synchronization in between, replication issues may occur. At least one of the users is likely to be working with an outdated version of a resource.

When the user attempts to synchronize data with the main OntoWiki server, several steps are taken to minimize the need for human intervention. However, fully automatic conflict resolving is not possible in all cases.

3.1 Concepts

The unit of editing and display in OntoWiki is a *resource*. Since OntoWiki Mobile needs to identify the same resource at different points in time, we define a resource r_t as a set of triples contained in some graph that share the same subject s at a certain point in time, i. e. a description of r at timestamp t. When an editing operation is carried out, all the changed triples are saved/deleted at once for a given resource. Thus, a *diff* d_{t_1,t_2}, $t_1 < t_2$ is the change applied to a resource description from timestamp t_1 to timestamp t_2. It is defined as a quadruple

$$d_{t_1,t_2} \coloneqq (t_1, t_2, \mathsf{Add}, \mathsf{Del}) = (t_1, t_2, r_{t_1} \setminus r_{t_2}, r_{t_2} \setminus r_{t_1}).$$

That is, it contains a set of added and a set of removed statements that led from r_{t_1} to r_{t_2}. Two diffs $d_{s_1,t_1} = (s_1, t_1, \mathsf{Add}_1, \mathsf{Del}_1)$ and $d_{s_2,t_2} = (s_2, t_2, \mathsf{Add}_2, \mathsf{Del}_2)$ are said to be *in conflict* if both of the following conditions are met:

$$2 \cdot |\mathsf{Add}_1 \cup \mathsf{Add}_2| > |\mathsf{Add}_1| + |\mathsf{Add}_2| \tag{1}$$

$$\mathsf{Del}_1, \mathsf{Del}_2 \neq \emptyset \Rightarrow \mathsf{Del}_1 \cap \mathsf{Del}_2 \neq \emptyset \tag{2}$$

In other words, both diffs remove at least one identical and add at least one different triple. The empty diff $d_{s,t} = (s, t, \emptyset, \emptyset)$ does not conflict with any other diff. This definition gives necessary, but not sufficient, conditions for conflicting changesets, i. e. there are non-conflicting changesets that meet both conditions.

Let $d_{s_1,t_1} = (s_1, t_1, \mathsf{Add}_1, \mathsf{Del}_1)$ and $d_{s_2,t_2} = (s_2, t_2, \mathsf{Add}_2, \mathsf{Del}_2)$ be two diffs at timestamps t_1 and t_2 with $s_i < t_i$, $t_1 < s_2$. The *concatenation* operation $\circ : \mathsf{D} \times \mathsf{D} \longrightarrow \mathsf{D}$ (D denoting the set of all diffs) yields a new diff with the combined additions and deletions from d_{s_1,t_1} and d_{s_2,t_2}:

$$d_{s_1,t_1} \circ d_{s_2,t_2} = (s_1, t_2, \mathsf{Add}_1 \cup \mathsf{Add}_2, \mathsf{Del}_1 \cup \mathsf{Del}_2).$$

Consecutive diffs to the same resource r_t are combined to a *changeset*s, which are exchanged between mobile devices (OntoWiki Mobile) and OntoWiki on synchonization.

3.2 Synchronization

We are now in the position to specify what happens when users synchronize data with OntoWiki. Given a re-established data connection and the user's consent, OntoWiki Mobile sends all changesets back to the server's synchronization component. Let c be a changeset on resource r with diffs $(d_{s_1,t_1}, d_{s_2,t_2}, \ldots, d_{s_k,t_k})$. OntoWiki Mobile concatenates the diffs contained in c into a single diff d_{s_1,t_k}. A server diff is then calculated as $d_{s,t}$ where s and t are the largest timestamps for changes on r smaller than s_1, t_k respectively. Using conditions (1) and (2) conflicting diffs are determined. In case of a conflict, all diffs $d_{s,t}$ in c are applied sequentially (w. r. t. t) until the conflict occurs. At this point, two branches of r are created having as last common version $r_{t_{k-1}}$ where t_k is the conflicting patch.

For merging branches, two different ways exist: manually (using the OntoWiki's merging UI) or semi-automatic (using EvoPat). EvoPat allows the application of policy-based merging patterns on conflicting branches. Patterns for the following merging policies are provided with OntoWiki Mobile:

– *User privilege-based* – changesets from users with higher priority have prevalence or
– *time privilege-based*, which can also be called "first-come, first-serve" – the latest changes are considered least prioritized.

Additional policies (in the form of EvoPat evolution patterns) can be created by the user, if needed.

There are some situations that cannot be completely resolved without user intervention or creation of additional rules for EvoPat. For example, in cases where two users create a resource describing the same real world object by using different identifiers.

3.3 Example

Consider two users *Alice* and *Bob* who both work on the same resource r_{t_0} which has two statements, s_1 and s_2, as of timestamp t_0. The described scenario is depicted in Figure 2.

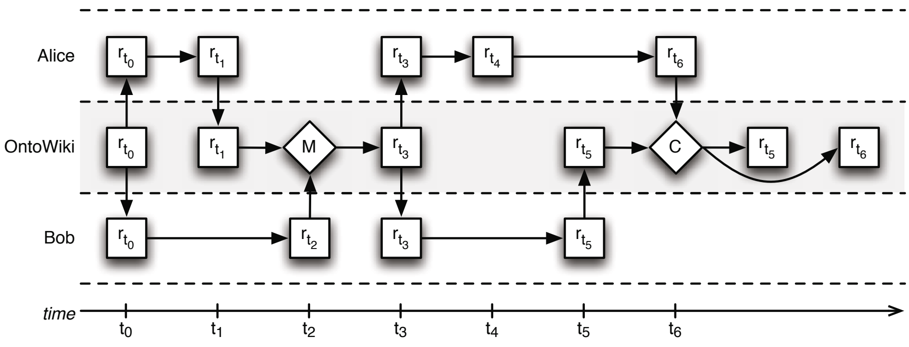


Fig. 2. Data replication example with merge (M) and conflict detection (C)

At t_1, Alice changes statement s_1 to s_3; this is actually reflected as deleting s_1 and adding s_3. In the same way, Bob removes s_2 and adds s_4 at t_2. When both synchronize with OntoWiki, their respective changesets contain only one patch each

$$c_{t_0,t_1,\text{Alice}} = (t_0, t_1, \{s_3\}, \{s_1\}) \quad \text{and} \quad c_{t_0,t_2,\text{Bob}} = (t_0, t_2, \{s_4\}, \{s_2\}).$$

Since $\{s_1\} \cap \{s_2\} = \emptyset$, condition (2) does not hold and we have non-conflicting changesets that can be merged into r_{t_3} at t_3. Alice then adds another statement s_5 at t_4 and later discovers that she entered duplicate information and decides to remove s_3 at t_5. Meanwhile, Bob also notices the error on s_3, removes it and adds s_6 at t_4. This time, when both synchronize their data with OntoWiki, we have patches

$$c_{t_3,t_6,\text{Alice}} = (t_3, t_6, \{s_5\}, \{s_3\}) \quad \text{and} \quad c_{t_3,t_5,\text{Bob}} = (t_3, t_5, \{s_6\}, \{s_3\}).$$

As can be easily verified, both conditions now hold and we deal with a conflicting changeset. OntoWiki Mobile thus creates two versions, r_{t_5} and r_{t_6}, resulting from applying $c_{t_3,t_5,\text{Bob}}$ and $c_{t_3,t_6,\text{Alice}}$ to r_{t_3}, respectively.

4 User Interface

The OntoWiki Mobile user interface supports currently three different usage patterns: standard browsing along the taxonomic structures (e. g. class hierarchies) found in the knowledge base, faceted browsing for filtering instances based on property values as well as authoring of new information on-the-go.

4.1 Standard Browsing

Figure 3 shows the OntoWiki Mobile standard navigation user interface in different browsing states. In accordance with popular touch-oriented mobile software platforms, the user interface was based on lists so as to simplify navigating through interlinked resources. The first screenshot (Figure 3a) shows the list of all knowledge bases. The login button in the top-left corner allows to log in as a

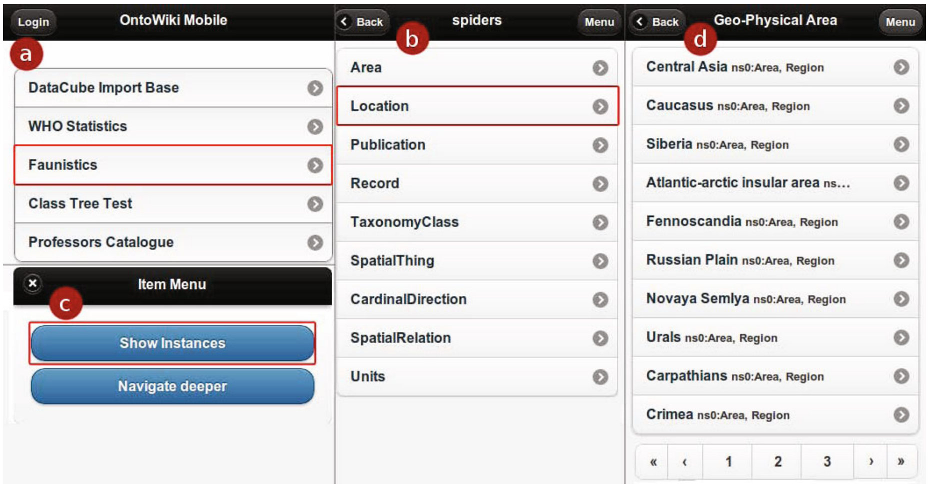


Fig. 3. OntoWiki Mobile standard browsing interface

registered user (e. g. to write to protected knowledge bases). After the user selects the knowledge base by tapping on it (tapped areas show as red squares), the top level class structure is displayed, as shown in second screenshot (Figure 3b). Once a particular class is chosen the user has to select (Figure 3c) whether he wants to see the list of instances for this class or navigate deeper in a class tree. Navigating through the class tree simply changes the classes list entries and refreshes the view. If the user chooses to view instances, a new list of instances from this class is presented (Figure 3d). After selecting a particular instance all properties are grouped by predicates and rendered in a list.

4.2 Faceted Browsing

Faceted browsing is a special way to navigate through instances in a specific knowledge base. It allows for simple and efficient filtering of the displayed instances list by applying available instance properties as filters. Faceted browsing can be used with any instance list in OntoWiki Mobile. As show in Figure 3d instance list view has a menu button in the upper-right corner. Using this button the user can access the instance list menu (Figure 4a), where he can execute a simple string search in current knowledge base or use the "Filters" button to access the faceted browsing feature. As shown in Figure 4b, the active filters list view displays all currently applied filters. By checking filters and pressing the "Delete" button at the bottom of the screen, the user can remove filters he does not like to apply to the list. To add a new filter the "Add" button in the upper-right corner of the screen can be used, which will display the list of all available filters (Figure 4c). Selecting one of the displayed filters will open the list of values for it (Figure 4d). Selecting values from the list shown will apply the new filter value restriction on the previously displayed instances list.

4.3 Authoring

Data authoring in OntoWiki Mobile is done using the RDFauthor [12] – a JavaScript-based system for RDF content authoring. As mentioned earlier,

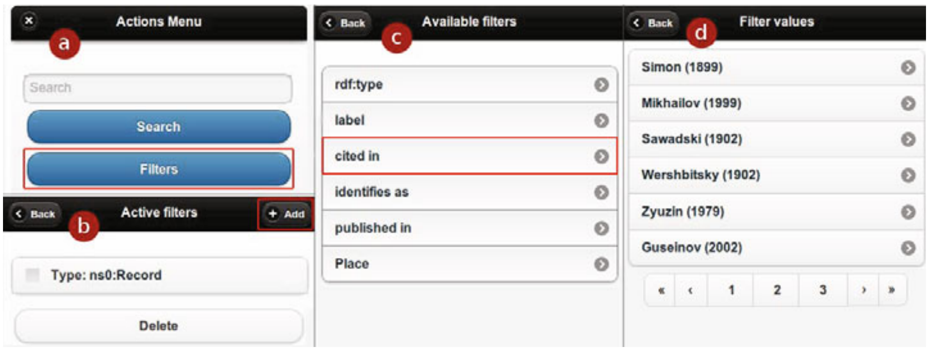
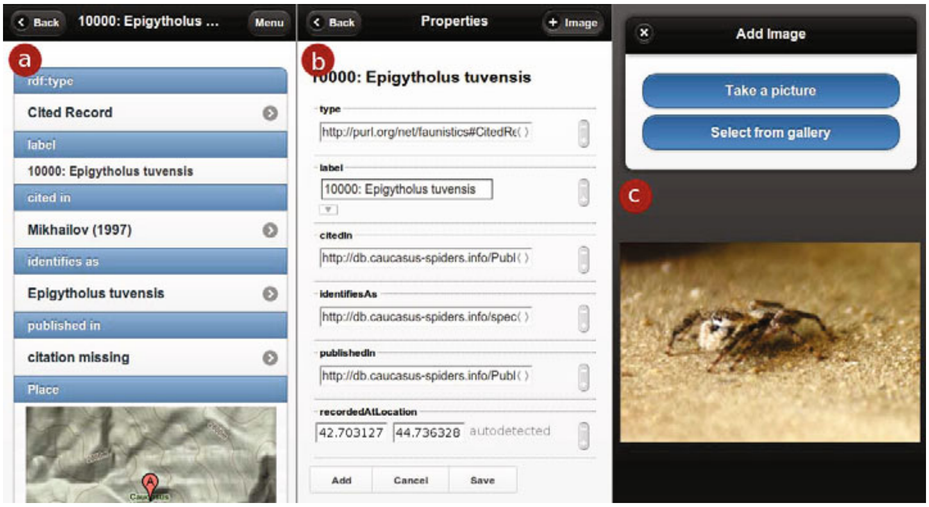


Fig. 4. OntoWiki Mobile faceted browsing interface



10000: Epigytholus ...

rdf:type
Cited Record
label
10000: Epigytholus tuvensis
cited in
Mikhailov (1997)
identifies as
Epigytholus tuvensis
published in
citation missing
Place

10000: Epigytholus tuvensis

type
http://purl.org/net/faunistics#CitedRe
label
10000: Epigytholus tuvensis
citedIn
http://db.caucasus-spiders.info/Publ
identifiesAs
http://db.caucasus-spiders.info/spec
publishedIn
http://db.caucasus-spiders.info/Publ
recordedAtLocation
42.703127 44.736328 autodetected

Fig. 5. OntoWiki Mobile authoring interface

instance properties are grouped by predicates and rendered as lists (Figure 5a). The rendering of properties and their values is based on the data type and ontology structure (see the display of a map for attached geo-coordinates). Figure 5b shows how an instance can be created or edited using forms, which are automatically created by RDFauthor based on the underlying ontology structure in the knowledge base (note auto-detected geographical coordinates for current location). Figure 5c shows an example of the interaction of OntoWiki mobile with the sensors of the mobile device in terms of accessing the integrated camera for adding a picture to an ontology instance.

5 Use Case and Evaluation

The development of OntoWiki Mobile was triggered by users aiming to gather data in field conditions. To simplify the data collection we created a mobile interface that allows users to enter data instances on mobile devices, such as mobile phones and tablets. In particular, there is a community of scientists who collect data about spiders in the Caucasus region [5] in a web portal[5]. The project consists of two major software parts:

– The portal backend, which is based on the semantic data wiki OntoWiki. Each arachnologist can login to this backend and use it for data entry, management and queries. The backend itself is a standard OntoWiki installation with some custom and some common vocabularies imported.

[5] `http://db.caucasus-spiders.info`

- The portal front-end, which itself is an extension of OntoWiki, generates a more visitor-friendly representation of the databases resources. The main focus for these visitors are species checklists, which give an overview of verified species of a given region.

To calculate these species checklists for a specific area, data from original finding spots (e. g. "I've found the species Pholcus phalangioides near Khashuri in Georgia in a cave.") as well as data from the literature (e. g. "Mkheidze (1964) published he found this species in Lentekhi as well.") is collected by the users. While the literature research is done at home using the desktop browser-based version of OntoWiki, the field data is gathered according to the following workflow:

1. A research team travels to the area and sets up traps at specific locations or specifically catches interesting individuals.
2. The finding spots are documented and the individual animals are associated with these finding spots (e. g. by signing a conservation container with the location ID).
3. The individuals are carried to the laboratory where they have to be identified. This is a challenging task and often individuals are sent to specialists for a specific genus or family of spiders.
4. Finally, the individual is identified as a certain species. This event either increases the finding spot counter for this species in a certain area or adds another species entry to the species checklist for this area. In the latter case, this (re-)discovery of the species in a certain area can be published.

In this workflow, only the second step is relevant for the evaluation of our work since step 3 and 4 are done with the standard wiki and the portal front-end. The goal of step 2 is to describe the finding spot where a specific animal was found in order to proof assumption about the habitat and living of a specific species. These finding spots are classified according to a nature-phenomenological system (e. g. a cave, field, . . .) and are allocated to a nearby populated place. Populated places are ordered and associated to a political and administrative system (e. g. counties, administrative regions, country). The database currently consists of 1060 populated places and locations as well as 191 areas. A complete finding spot documentation is then entered in the following steps (see N3 example in Figure 6):

- Instantiate a specific type of location (e. g. a cave, example line 8).
- Add geo-coordinates to this resource (taken automatically from the GPS subsystem, example line 13).
- Associate pictures with this resource (taken from the camera subsystem, example line 12).
- Associate a nearby populated place or an area by searching the local store for an existing one or by creating a new resource (example line 14).
- Add any other information either specific for the location type (e. g. height for glaciers), specific for the researcher (e. g. comments and tags) or specific for the research journey (e. g. internal location ID for the conservation containers, refer example line 9–11).

The result of a finding spot location documentation is shown in Figure 6.

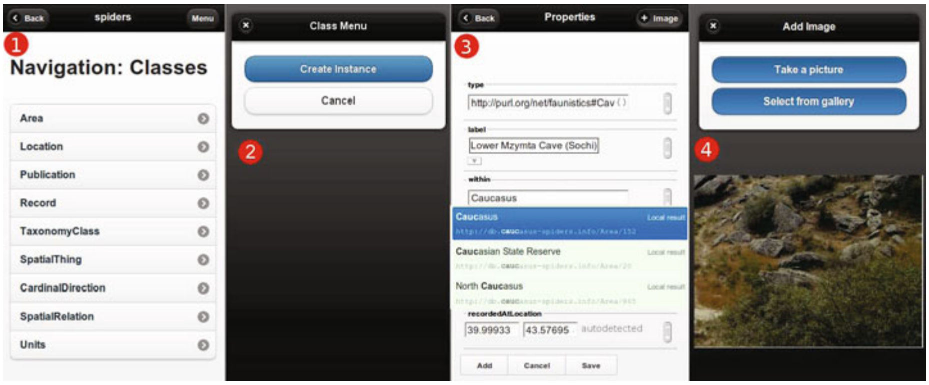
```
@prefix db: <http://db.caucasus-spiders.info/> .
@prefix faun: <http://purl.org/net/faunistics#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix geo: <http://www.w3.org/2003/01/geo/wgs84_pos#> .
@prefix dc: <http://purl.org/dc/elements/1.1/> .

<http://db.caucasus-spiders.info/Place/555> a faun:Cave ;
    rdfs:label "Lower Mzymta Cave (Sochi)";
    rdfs:comment "container 5";
    dc:creator "Stefan Otto"; dc:date "2010-07-21";
    foaf:depiction db:FotoXXX;
    geo:long "39.99933"; geo:lat "43.57695" ;
    faun:nearby db:Place19; faun:within db:Area439.
```

Fig. 6. Example finding spot documentation in N3

The data in this listing correspond to the screenshots shown in Figure 7.

After using the prototype for a few weeks on an Android developer phone, the following pro and con statements were obtained from the OntoWiki Mobile evaluation participants during interviews:

– Even without doing extensive data entry, the feature of associating images to existing resources was liked very much.
– There was a constant fear for data loss by breaking, misusing or loosing the mobile device. The added feature to export and import file backups from and to the application eased this. If there are more than one mobile device in the field, these backups can be additionally used to approve and inspect the data with a second pair of eyes.
– Obtaining GPS data from the mobile phone is nice but slightly inaccurate since the internal GPS systems of mobile phones are not as good as dedicated devices e. g. for hiking. Auto-completion of these values is a nice feature but users need to be able to correct them or, even better, receive them from another device and overwrite the existing values.
– The user experience strongly depends on the given CPU of the mobile device. Users running a mobile device with 500Mhz (HTC Hero) complained about the slowly responding user interface. In comparison to that, users with a 1000Mhz device (Samsung Galaxy S) reported a fast and reliable interface. In addition to the CPU, the version of the hosting Android operating system and esp. the used browser version strongly affects the user experience since newer Android versions also ship a new browser with a faster JavaScript execution engine.



type: http://purl.org/net/faunistics#Cav

label: Lower Mzymta Cave (Sochi)

within: Caucasus

Caucasus — Local result — http://db.caucasus-spiders.info/Area/132

Caucasian State Reserve — Local result — http://db.caucasus-spiders.info/Area/20

North Caucasus — Local result — http://db.caucasus-spiders.info/Area/945

recordedAtLocation: 39.99933 43.57695 autodetected

Fig. 7. Screenshots illustrating the workflow for creating a new finding spot according to the listing in Figure 6. From left to right: (1.) Searching or browsing for the class which needs to be instantiated. (2.) Initialization of a new resource from this class; all properties which are offered, are used in other instances of this class; GPS data is automatically requested and pre-filled by the phone. (3.) Entering literal data as well as linking to other resources. (4.) Assignment of existing images from the phone's image library.

6 Related Work

Related work can be roughly divided into the categories mobile semantic applications, strategies for replication, reconciliation in mobile usage environments.

6.1 Mobile Semantic Applications

The application of Semantic Web technologies on mobile devices is not new – one of the earlier works dates back to 2003 [4]. However, is was not pursued very actively due to the large number of mobile device limitations common in that era. In the light of increasing processing power, data connectivity and flexibility of modern mobile devices, the use of mobile Semantic Web technologies is becoming more feasible. There are a few publications which review the state of the mobile Semantic Web and the possibilities thereof to improve mobile Web (e. g. [3]). Also, there are publications on more complex topics like Semantic Web system for mobile devices named *SmartWeb* [11] which uses semantic technologies in combination with device specific input capabilities (e. g. voice input) to enhance mobile web services. On the frontend side there are semantic mobile applications like *DBpedia Mobile*[1] or *mSpace Mobile* [14] that implement or utilize the Semantic Web technologies directly on mobile devices. However, these frontend applications focus on very specific use cases – information about points of interest in the case of DBpedia Mobile and information for university students in the case of mSpace. OntoWiki Mobile on the other hand is a generic, application domain agnostic tool, which can be utilized in a wide range of very different usage scenarios ranging from instance acquisition to browsing of semantic data on the go.

6.2 Strategies for Replication

One of the most anticipated topics in distributed semantic data bases is data replication. Though, the topic of semantic data replication and conflict resolution in semantic data bases is not new, most of the existing approaches developed for desktop application rely heavily on client resources. On the other hand, topic of data replication in distributed databases on mobile devices is highlighted very thoroughly[6]. Semantic data replication for mobile devices adopts some of the approaches for generic mobile databases. There are generic approaches like change detection using a *Version Log* [7]. Also, there are specific approaches allowing to create an ontology versioning system for a particular RDF-based ontology language like *SemVersion* [13]. There are a large number of publications on data versioning and replication in the distributed database area, but most of these approaches require substantial computational power from client devices (which is not reasonably applicable in the case of mobile devices).

6.3 Reconciliation in Mobile Usage Environments

There is already a significant number of publications about data replication and versioning in the mobile Semantic Web environment. There are currently several trends in existing approaches:

– Complex client-side replication engines (like *MobiSem Replication and Versioning framework* [10]) that provide functionality to make relevant data available on the mobile device and to synchronize changes when connectivity is recovered.
– Enrichment of graphs with additional metadata (like *Triple Bitmaps* [9]) to simplify replication, merging and conflict resolution.

However, the implementation usually requires full-fledged RDF storage on the client-device with additional layers of versioning functionality. Adding such a additional layer is not always possible or difficult to implement for the large variety of existing mobile target platforms. OntoWiki Mobile on the other hand does not require any client-side technology beyond support for HTML5.

7 Conclusions

As the penetration of mobile devices able to access and interact with the Web can be expected to dramatically increase within the next years, the Semantic Web can ultimately only be successful if the use of semantic technologies on mobile devices is fully supported. With OntoWiki Mobile we tackled one particular but crucial aspect – the provisioning of a comprehensive knowledge management tool for mobile use. It employs the new HTML5 application cache functionality to support offline work and has advanced conflict resolution features built-in. OntoWiki Mobile demonstrates that a comprehensive semantic collaboration platform is possible to implement for mobile devices with minimal requirements based on recent Web standards (in particular HTML5). Although OntoWiki

Mobile was already used in a specific use-case by a number of non-IT, domain expert users more effort is required to evaluate the tool in a wider range of application scenarios. Due to its general purpose architecture OntoWiki Mobile is particularly suited to support the long tail of domain-specific mobile applications, for which the development of individual tools would not be (economically) feasible.

Future work will focus on representation of provenance and use of the mobile device's sensors for context-aware knowledge base exploration. With regard to the replication we plan to develop a rule-based approach for the selection of knowledge base parts to replicate on the mobile device. The approach will take mobile context information (such as the time, location) as well as usage patterns (e. g. browsing history) and manually supplied user preferences into account.

Acknowledgments

We would like to thank our colleagues from AKSW research group for their helpful comments and inspiring discussions during the development of this approach. This work was supported by a grant from the European Union's 7th Framework Programme provided for the project LOD2 (GA no. 257943).

References

1. Becker, C., Bizer, C.: Exploring the Geospatial Semantic Web with DBpedia Mobile. J. Web Sem. 7(4), 278–286 (2009)
2. Heino, N., Dietzold, S., Martin, M., Auer, S.: Developing Semantic Web Applications with the Ontowiki Framework. In: Pellegrini, T., Auer, S., Tochtermann, K., Schaffert, S. (eds.) Networked Knowledge - Networked Media. SCI, vol. 221, pp. 61–77. Springer, Heidelberg (2009)
3. Lassila, O.: Using the Semantic Web in Mobile and Ubiquitous Computing. In: Proceedings of the 1st IFIP WG12.5 Working Conference on Industrial Applications of Semantic Web, Jyväskylä, Finland, August 25-27 (2008)
4. Lassila, O., Adler, M.: Semantic Gadgets: Ubiquitous Computing Meets the Semantic Web. In: Fensel, D., Hendler, J.A., Lieberman, H., Wahlster, W. (eds.) Spinning the Semantic Web, pp. 363–376. MIT Press, Cambridge (2003)
5. Otto, S., Dietzold, S.: Caucasian Spiders – A faunistic Database on the spiders of the Caucasus. Newsl. Brit. Arachn. Soc. 108, 14 (2007), `http://caucasus-spiders.info`
6. Padmanabhan, P., Gruenwald, L., Vallur, A., Atiquzzaman, M.: A survey of data replication techniques for mobile ad hoc network databases. The VLDB Journal 17, 1143–1164 (2008)
7. Plessers, P., De Troyer, O.: Ontology Change Detection Using a Version Log. In: Gil, Y., Motta, E., Benjamins, V.R., Musen, M.A. (eds.) ISWC 2005. LNCS, vol. 3729, pp. 578–592. Springer, Heidelberg (2005)
8. Rieß, C., Heino, N., Tramp, S., Auer, S.: EvoPat – Pattern-Based Evolution and Refactoring of RDF Knowledge Bases. In: Patel-Schneider, P.F., Pan, Y., Hitzler, P., Mika, P., Zhang, L., Pan, J.Z., Horrocks, I., Glimm, B. (eds.) ISWC 2010, Part I. LNCS, vol. 6496, pp. 647–662. Springer, Heidelberg (2010)

9. Schandl, B.: Replication and versioning of partial RDF graphs. In: Aroyo, L., Antoniou, G., Hyvönen, E., ten Teije, A., Stuckenschmidt, H., Cabral, L., Tudorache, T. (eds.) ESWC 2010. LNCS, vol. 6088, pp. 31–45. Springer, Heidelberg (2010)
10. Schandl, B., Zander, S.: A Framework for Adaptive RDF Graph Replication for Mobile Semantic Web Applications. In: Joint Workshop on Advanced Technologies and Techniques for Enterprise Information Systems (Session on Managing Data with Mobile Devices), Milan, Italy, pp. 154–163. INSTICC Press (May 2009)
11. Sonntag, D., Engel, R., Herzog, G., Pfalzgraf, A., Pfleger, N., Romanelli, M., Reithinger, N.: SmartWeb handheld — multimodal interaction with ontological knowledge bases and semantic web services. In: Huang, T.S., Nijholt, A., Pantic, M., Pentland, A. (eds.) ICMI/IJCAI Workshops 2007. LNCS (LNAI), vol. 4451, pp. 272–295. Springer, Heidelberg (2007)
12. Tramp, S., Heino, N., Auer, S., Frischmuth, P.: RDFauthor: Employing rDFa for collaborative knowledge engineering. In: Cimiano, P., Pinto, H.S. (eds.) EKAW 2010. LNCS, vol. 6317, pp. 90–104. Springer, Heidelberg (2010)
13. Völkel, M., Groza, T.: SemVersion: RDF-based ontology versioning system. In: Proceedings of the IADIS International Conference WWW/Internet 2006, ICWI (2006)
14. Wilson, M., Russell, A., Smith, D.A., Owens, A., Schraefel, M.C.: mSpace Mobile: A Mobile Application for the Semantic Web. In: Proc. of the ISWC 2005 Workshop on End User Semantic Web Interaction. CEUR Workshop Proceedings, vol. 172, CEUR-WS.org (2005)