Linked data from your pocket: The Android RDFContentProvider

Jérôme David, Jérôme Euzenat

INRIA & LIG
Grenoble, France
{Jerome.David,Jerome.Euzenat}@inrialpes.fr

Smartphones are becoming main personal information repositories. Unfortunately this information is stored in independent silos managed by applications. We have seen that already: in the Palm operating system, application "databases" were only accessible when the application schemas were known and worked by opening other application databases.

Our goal is to provide support for applications to deliver their data in RDF. This would allow applications to exploit this information in a uniform way without knowing beforehand application schemas. This would also connect this information to the semantic web and the web of data through reference from and to device information. We present a way to do this in a uniform manner within the Android platform. Moreover, we propose to do it along the linked data principles (provide RDF, describe in ontologies, use URIs, link other sources).

We first consider how the integration of RDF could be further pushed within the context of the Android platform. We demonstrate its feasibility through a linked data browser that allows for browsing the phone information.

1 Providing RDF support in Android

Android is built around different kinds of services, one of which being `ContentProvider` which exposes some data of an application to other applications. `ContentProvider`s manage data structures (usually relational tables) and are able to answer messages of type query, insert, delete and update. A query returns a cursor on a table of tuples. So they offer a typical database interface:

```
Cursor query( Uri id, String[] proj, String select, String[] selectArgs, String order )
Uri insert( Uri id, ContentValues colValueList)
int update( Uri id, ContentValues colValueList, String select, String[] selectArgs )
int delete( Uri id, String select, String[] selectArgs )
String getType( Uri id )
```

This interface uses URIs for identifying the kind of data which is requested: content://contacts/people/ identifies all people in the contact application and content://contacts/people/22 identifies the 22nd instance of these[1].

Calling a `ContentProvider` is driven by the kind of content to be manipulated: the calling application indicates the will to retrieve some content through its type, but does not control which application will provide it. Android calls a `ContentResolver` which further looks into the query (the `id`) to find a suitable content provider on the

[1] Android URIs are not particularly portable, we leave the discussion of this out of this paper.

phone which is able to provide the required content. For that purpose, the resolver maps the query URIs to the declared providers. These providers are declared in application manifests.

In order to exchange RDF within the Android platform, we need `ContentProvider`s providing RDF. For that purpose we have developed a new abstract `RDFContentProvider` extending `ContentProvider`. Answers to queries in an `RDFContentProvider` could be:

– set of triples, which would correspond to the description of one object and the attribute values, this is restricted to queries like: tell me what you know about a particular individual;
– table of tuples, like in `ContentProvider`s or SPARQL which would correspond to values of variable in a SPARQL-like query

These interfaces can be unified since, the former is the answer to the SPARQL query:

```
SELECT ?p ?o WHERE content://contacts/people/22 ?p ?o.
```

The `RDFContentProvider` follows primarily the same kind of interface as `ContentProvider`. The minimal interface for linked data applications is:

– `RDFCursor getRdf( Uri id )`

The Cursor iterates on a table of subject-object-predicate (or object-predicate) which are the triples involving the object given as a URI. A more elaborate semantic web interface could be that of a minimal SPARQL endpoint:

– `Uri[] getTypes( Uri id )`: returns the RDF types of a local URI;
– `Uri[] getOntologies()`: ontologies used by the applications;
– `Uri[] getQueryEntities()`: classes and relation that the application can deliver;
– `Cursor query( SparqlQuery query )`: returns results tuple;
– `Cursor getQueries()`: triple patterns that the application can answer.

So far, we have only developed this interface but the three first primitives.

2 Demonstration: linked data browser

We have implemented a prototype of this architecture described in Figure 1. Precisely, we have implemented:

– `RDFContentProvider`: the provider interface;
– `RDFContentResolver`: which can decide to which class to redirect a query;
– `Pikoid`: a picture annotation application which implements `RDFContentProvider`;
– `AndroidRDFProvider`: an application encapsulating data access to the applications of the Android platform (Calendar, Contact, Map, etc.);
– `RDFBrowser`: a simple client for navigating within RDF data provided by these applications in a generic manner.

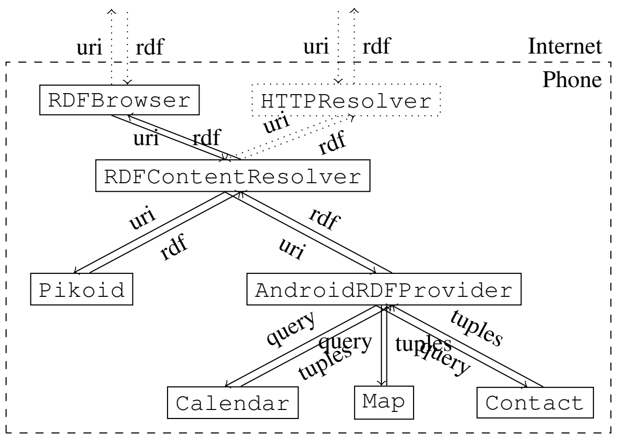


Fig. 1. The four implemented components and the communication between them. Communication with the web (dotted) is not implemented at submission time.

In this demonstration, we will show how to quickly annotate a picture with the help of the standard Android applications (Pikoid), then we will use the RDFBrowser to navigate through these annotations provided as interrelated RDF statements.

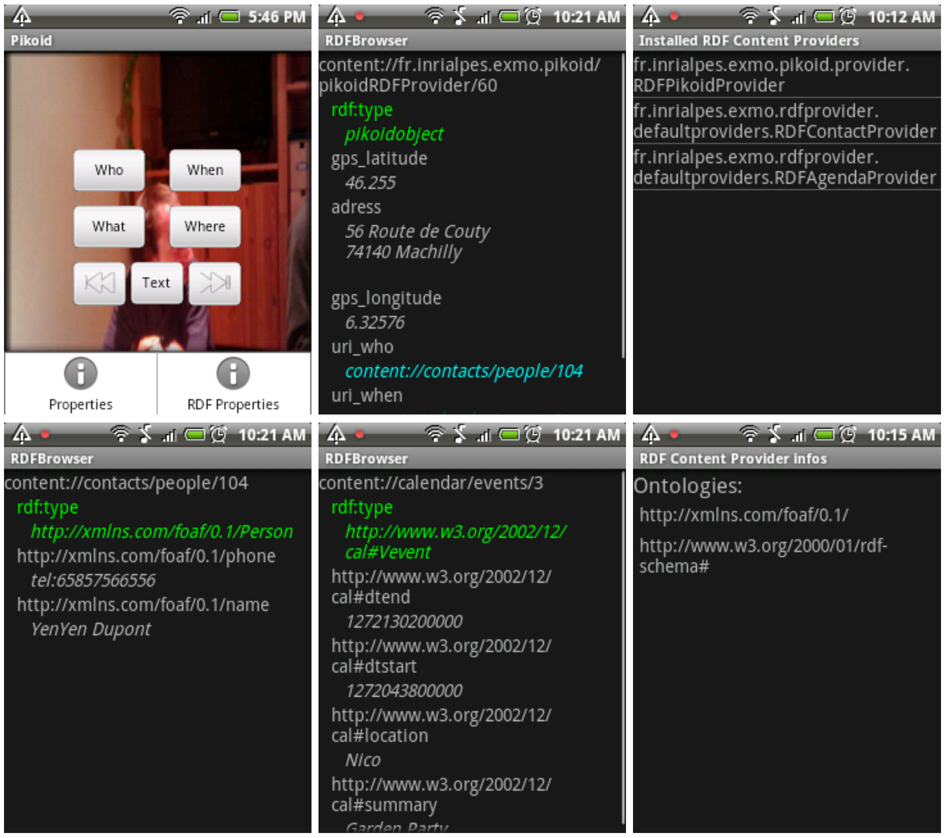
The beauty of linked data is that it is easy to understand how to navigate within this data through HTTP requests: each request (a URI) returns RDF from which URIs can be extracted for formulating further requests. The RDF browser acts as a linked data client except that it works over the Android content provider mechanism instead of HTTP: it asks triples about a particular URI, displays these and when clicked on, issue the same URI query. Figure 2 shows the current interface.

3 Conclusion

What we have to demonstrate is only a proof of concept that semantic technologies could be included uniformly in a portable platform with a minimal overhead. This would accomplish one integration step further since the seminal work of [1].

There is room for many developments on the basis of the `RDFContentProvider` interface, bringing our personal data silos closer to the semantic web. Many issues have not been considered in this first development, most importantly the connection to the web. From the Android device to the web, using REST over HTTP to reach (linked) RDF data is not a real problem. From the web to Android, implementing a HTTP server which acts as a REST proxy for Android data accessible to our `RDFBrowser` is not difficult either. The main issue is the conversion of Android local URIs to HTTP URIs.

The system is available at `http://swip.inrialpes.fr` as several applications. It has been tested on the Android emulator and HTC Android 1.6 devices. Specific developments have been made for this version which would not be necessary in newer versions of Android. It is currently being tested on other devices.



```
content://fr.inrialpes.exmo.pikoid/
pikoidRDFProvider/60
  rdf:type
    pikoidobject
  gps_latitude
    46.255
  adress
    56 Route de Couty
    74140 Machilly

  gps_longitude
    6.32576
  uri_who
    content://contacts/people/104
  uri_when
```

```
fr.inrialpes.exmo.pikoid.provider.
RDFPikoidProvider
fr.inrialpes.exmo.rdfprovider.
defaultproviders.RDFContactProvider
fr.inrialpes.exmo.rdfprovider.
defaultproviders.RDFAgendaProvider
```

```
content://contacts/people/104
  rdf:type
    http://xmlns.com/foaf/0.1/Person
  http://xmlns.com/foaf/0.1/phone
    tel:65857566556
  http://xmlns.com/foaf/0.1/name
    YenYen Dupont
```

```
content://calendar/events/3
  rdf:type
    http://www.w3.org/2002/12/
    cal#Vevent
  http://www.w3.org/2002/12/
  cal#dtend
    1272130200000
  http://www.w3.org/2002/12/
  cal#dtstart
    1272043800000
  http://www.w3.org/2002/12/
  cal#location
    Nico
  http://www.w3.org/2002/12/
  cal#summary
    Garden Party
```

```
Ontologies:
 http://xmlns.com/foaf/0.1/
 http://www.w3.org/2000/01/rdf-
 schema#
```

Fig. 2. The Pikoid application annotates images with metadata stored as RDF. RDFBrowser allows for querying this information to the Pikoid RDFContentProvider interface and displaying it. The current picture metadata is shown in the second panel (pikoidRDFprovider/60). From there, it is possible to browse the information available in the address book (people/104) and the calendar (events/3) through the AndroidRDFProvider wrapper. Finally, RDFBrowser also allows for inspecting available RDFContentProviders and the ontologies they manipulate.

References

1. Walsh, N.: Generalized metadata in your palm. In: Proc. 2nd Extreme markup languages conference, Montréal (PQ CA) (2002), `http://conferences.idealliance.org/extreme/html/2002/Walsh01/EML2002Walsh01.html`

4 Acknowledgements

We thank Yu Yang who has programmed part of this demonstration.