

A Name Resolution Mechanism for the RDF Model

Dirk-Willem van Gulik
@semantics, Leiden
foaf.aseanalytics.com/dirkx
dirkx@aseanalytics.com

Alberto Reggiori
@semantics, Milano
foaf.aseanalytics.com/alberto
alberto@aseanalytics.com

Zavisa Bjelogrić
@semantics, Rome
foaf.aseanalytics.com/zac
z@aseanalytics.com

ABSTRACT

This paper describes a name resolution mechanism suitable for loosely coupled applications without a central authority - a scenario we expect for many RDF applications where *clouds* of federated information progressively merge into a bigger cloud. The mechanism is based on Internet standards and it extends the basic name resolution mechanism with functions suitable for management of resources and resource descriptions available in data clouds. The mechanism is used as a common platform in our RDF applications, it drives the RDF gathering and supports the editing processes. The mechanism is presented using an existing production web management system where Web resources belonging to different internal and external providers are described in RDF and integrated into a common application.

Categories and Subject Descriptors

C.2.4 [Computer-Communication Networks]: Distributed Systems—*Distributed applications*; J.2 [Computer Applications]

General Terms

Design, Experimentation

Keywords

Metadata, Discovery, Data Integration

1. INTRODUCTION

The Resource Description Framework (RDF) [1] is an effective model for describing in a unified way *resources*¹. RDF and RDF Schema (RDFS) [2] are the basic layer of the Semantic Web; the Friend-Of-A-Friend network [3] and similar applications [4] provide good, simple and effective examples of usage of RDF for implementing the so called “Shallow Semantic Web”[5].

The global model introduced by RDF and RDF Schema (RDF/S) simplifies development of applications in loosely coupled environments. For example, multiple authorities (or players) may add their own descriptions - metadata to existing resources. In a similar way, initial annotations can be upgraded as the application needs grow - all this ‘*without breaking the system*’ and with minimal changes to existing workflows.

¹By definition, a resource is anything which can be identified by a URI.

However this global model needs to be extended to develop real live applications. The Image Showcase²[6] developed for the European Space Agency (ESA) is typical example for such applications. It is an umbrella over several data sets; each with a different structure, yet similar as they all cover Remote Sensing and Satellite data. A common index, search, display and annotation framework has been developed to give the appearance of a unified interface yet each data set continues to be managed with its own workflow and under its original authority. This applications, and most others build by @semantics (e.g. [7], [8]) follow this common flow (Figure 1): data is gathered from multiple sources, a URI is assigned, it is then indexed after which it can be searched and retrieved. And at that point mutations or annotations are possible.

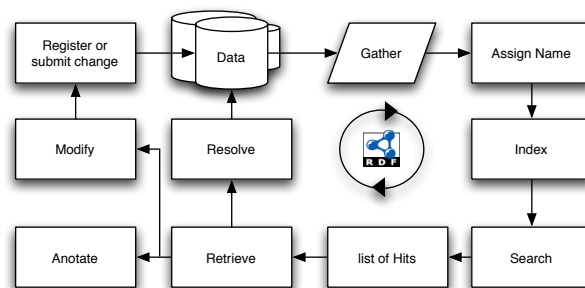


Figure 1: Simplified Application Workflow

The elements needed beyond RDF/RDFS for any such simple data - metadata management tool which can deal with multiple sources are (in no particular order):

(i) *URIs*: RDF relies heavily on URIs - for identifying resources, properties and property values. Typically, URLs are used today. This is good solution, but in the long term, URLs, especially URLs used to identify resources, can change³ thus breaking the model. Cross-linking of description from different sources can also be difficult when they come from different origins and are managed by different sources. For example, FOAF generally uses the mailbox (URL of the e-mail address) to identify a person. This URI can change over the time or multiple versions may be in use at different places making it more difficult linking of person to another person. A similar problem is found in news clipping services where

²<http://earth.esa.int/isc/>

³Even when carefully maintained; the location based nature of URLs and the lack of indirection make it naught impossible to survive, say, a corporate merger and a reassignment of the FQDN/site name part of the URL.

the URL of the same article (or image) can change because of the CMS solution adopted. Use of unique names (URNs) can be in many cases a better and *more robust* solution, especially when the naming mechanism does not require a centralized naming authority⁴.

(ii) *Where is the data and metadata?* A mechanism for discovering *all metadata* related to a resource is needed in practical applications. The RDF model allows multiple descriptions of the same resource. Anybody can say anything about a resource⁵ but there is no way to know *a-priori* which metadata was created. In an ideal system, a ‘magic’ or ‘google-ish’ mechanism will gather/refresh all the RDF descriptions available in the world to build a graph representing all ‘knowledge’ about resources on the Web. In practice, such a general approach may be *too slow* for the application requirements or it might *skip some of metadata* because of access controls or the fact that no routes to isolated descriptions was found.

Rather than a global, omni-comprehensive Web, we foresee here a situation where *clouds*, (some of which may be very private and closed) - domains of information will be progressively merged in bigger and bigger cloud. For this scenario, we need an *extended naming mechanism* which is able to define (i) which resources belong to a cloud and (ii) which annotations, e.g. RDF/XML descriptions created for these resources. All this by registering also when the resource and its metadata were modified.

An example where this problem was encountered is the Image Showcase[6] - a web management application where sample images and stories from different internal and external sources are described in RDF by different players and which is used as a basis to create new applications. In this case, we used the extended name resolution mechanism also (a) to register *which resources* - web pages and images - were included in the application domain and (b) to register which RDF descriptors were created or modified for these resources. In both cases, we registered also the date of change.

(iii) *Gathering and Editing* The extended name resolution mechanism is used for gathering - to tell the crawler which RDF/XML descriptions were created or modified by *any player* - *authority cooperating in the system*. This allowing a fast refresh of the RDF storage and making visible the changes to applications built over it. In a similar way, the naming mechanism is integrated with editing applications.

Similar problems and similar solutions has been proposed by the Life Science IDentifier (LSID)[9] initiative which defined a similar URN resolution mechanism. We extended their approach by proposing a solution which does not need a central authority and is suitable for loosely coupled or federated applications. We also extended the basic URN to URL resolution with mechanisms for metadata resolution. This allows association of data with corresponding metadata creating a basic layer for management of data and metadata.

2. NAMING MECHANISM

Our basic problem can be split in four steps: (i) Assign a name to an object, a piece of data or concept. (ii) Reference that object by the name assigned. (iv) Allow registration-association of data and metadata to the name assigned and (iv) obtain a reference to the object itself or to metadata describing the object. All this while

⁴Use of a centralized naming authority in conflict with the loosely coupled nature of these applications. This may also be a single point of failure - the responsibility for keeping consistent URLs is simply moved to the central “clearing house”

⁵A resource is explicitly identified by it’s URI in the RDF annotations.

imposing few, if any, restrictions with regard to control and central naming authority.

To solve this, a specific subset of the general Universal Resource Identifiers was used: the ‘Universal Resource Name’ or URN⁶. The actual URN Namespace definition mechanism [13] was then used to define a URN namespace and the required delegation to:

- a) The assignment of a name to an object, a piece of data or some concept (*naming*)
- b) then being able to refer to that object by name (*referencing*)
- c) still being able to either obtain the object or (meta)data about the object at any later time (*resolving*)
- d) and allowing 3rd parties to annotate the named object with their own named annotations (*annotating*)

While imposing few if any restrictions with regard to control and (central) authority for the above naming, references, resolution or annotation management.

Adherence to these 4 internet standards gives us robust naming schemes with well defined semantics. The standards are neutral with regard to the rules of the naming itself and the relations between naming authorities.

2.1 Dynamic Delegation Discovery System

The Dynamic Delegation Discovery System (DDDS) is a technology neutral resolution methodology which can be used in a very wide range of circumstances, applications and with various protocols. In our applications a Domain Name System (DNS) based resolution protocol will be used as our customers already have established DNS domains and authority policies along appropriate organisational boundaries.

RFC 3401-3405([14], [15], [16], [17] and [18]) describe in detail the process by which a URN is to be resolved. The result of this is that for a given URN one can ascertain:

- i) whether it (still) exists; i.e. is still managed as such by the name space authority or the party to which that authority has been delegated.
- ii) what techniques, such as HTTP, SOAP, LSID or even fax or email may be used to get more information about the ‘thing’ the name stands for.
- iii) if the ‘thing’ itself can be fetched and/or if (meta) data about the object can be retrieved. Or if there is an equivalent URL available at this point in time.
- iv) and for each of those, insofar as appropriate, a list of IP addresses, port numbers and protocols to be used for this.

This process is illustrated in figure 2.

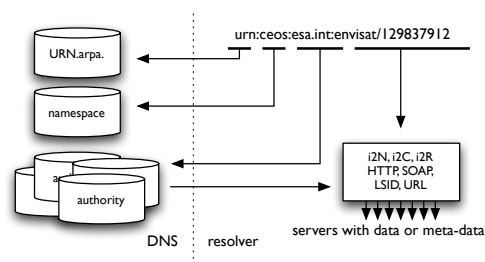


Figure 2: Simplified DDDS Resolution

⁶Functional Requirements for Uniform Resource Names were defined in[10], this document captures the essence of URNs very effectively. URN syntax was defined in [11]. The architectural concept of ‘resolving’ can be found in [12].

It should be noted that although the top level delegation (in the root domain of urn.arpa.) is central; neither DDDS nor RFC3406 [13] imposes restrictions as to how the name space is managed, the degree of centralization or its reliability. A wide range can be accommodated; from very central with well defined semantics and long-leivty promises to completely anarchy with no social contract. For practical applications however the latter far end of the range makes URN's almost akin to URL's - and the lack of an community or social contract, implicit or explicit, will hamper the effectiveness of applications.

3. METADATA MANAGEMENT

3.1 Gathering

Metadata created for an application, e.g. RDF/XML descriptions must be processed by the RDF engine to make the information available to applications. In our cases, applications were mainly user interfaces offering searching and browse features across all resources described in RDF. The information is never static in practice: new resources are added constantly and metadata was changed every day, either because of errors in initial metadata or because needs for new annotation occurred. In such situations different approaches are possible: (i) RDF descriptors are being re-processed on-the-fly - the best solution for small collections changing rapidly, (ii) RDF descriptors are gathered and stored in an RDF engine doing all pre-processing (indexing) needed - this in case of big collections of data where fast response is needed or (iii) a combination of both modes is used. In all of these cases, the RDF engine has to collect - gather all RDF/XML descriptions available in the system and to process them.

In our first RDF applications, we used ad-hoc solutions based on naming conventions, directory tree and heavy use of the Unix `make` utility. This was a pragmatic solution good for applications running on a single machine but difficult to scale to any really distributed environment. As soon as we started working with external data and with multiple authorities defining metadata (i.e. making annotations on managed resources) the `make` based solution broke due to the flawed assumption that all authoritative changes are made local.

Currently, we are using an experimental solution based on the extended naming mechanism described in the section 2.

The process can be summarized in following steps: (i) when a new resource is added to the application domain, it is registered to the name server with a certain name X; (ii) when a piece of metadata about the resource is created or modified in the domain (e.g. as an RDF/XML description), this fact is registered at the name server together with time and origin of the annotation; (iii) when queried to resolve the name X, the name service can also return a list of all annotations made for this resource together with their respective domain source/provenance information. (iv) Starting from the selected domain, the list of resources and all corresponding RDF/XML descriptions can easily be obtained and used to control gathering of RDF data and ingestion in the RDF engine.

3.2 Editing

Editing is an essential step in the information life cycle. It is used to create new annotations - metadata, to upgrade existing annotations with new metadata or to correct existing metadata (e.g. because of errors in automatic metadata extraction processes).

Despite the fact that RDF can be unconstrained; in most practical production applications both XML namespaces [19] and RDF Schema [2] are being used. These can be used to augment automated graph editors; by using `rdfs:label` properties and other augmenting information together with the structure.

A simple application; `rdfformer` was written which operates on any RDF sub-graph as an input; and creates an interactive HTML interface using the RDF schema information and optional templates or stylesheets.

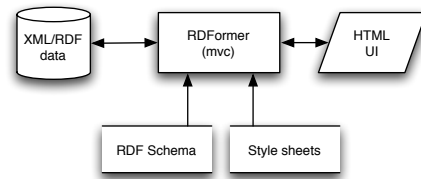


Figure 3: RDFFormer Workflow

The entire editing process is integrated with the name resolution mechanism. (i) when start working with a resource, a list of all existing RDF/XML descriptions can be retrieved from the name server. (ii) A selected description is edited and saved back and the name service is informed about this change or (iii) a new description is being created and this is associated to the original resource (See figure 3).

3.3 Security Aspects

The *registration* mechanism - notification of new events to the naming mechanism - in our applications was under full control being part of a system running in closed, protected environment. Even if external data was accessed and different players participated, all registration steps were well protected. In a more open environment, registration steps above might become security holes allowing malicious players (e.g. spammers) to add their resources to the domain or to 'advertize' malicious metadata.

An effective solution to this potential problem is the use of 'Secure Dynamic DNS Updating (SDDNS)' [20] to allow a third party to register an annotation to a resource with the naming authority of the URN of that resource. The procedure is as follows: (i) the third party creates the annotation and (ii) it assigns a URN within its own realm. (iii) it generates a public/private key pair and (iv) lodges the public key with the naming authority of the URN it is annotating. The public key, combined with the URN annotated is also used as the basis for a unique identifier of the annotation *within* the namespace of the resource its authority. From that moment on the third party is able to use SDDNS to manage the delegation of the thus assigned part of the namespace. It then (v) issues a SDDNS update, using its private key, to (vi) delegate the name to its own resolution infrastructure⁷ which is to return the URN it assigned in step 'ii'. If required this can be verified by the naming authority of the record prior to processing the update from step 'v'.

At any point in time it is then possible for the authority which assigned the URN to the resource to retrieve a full list of any annotations by retrieving the list of delegations in its own domain; and then resolving each item into the URN of the annotation. Likewise if the authority chooses to do so; a so called '(incremental) zone transfer' ([21] and [22]) may be made possible to allow third parties to also obtain a lists of these delegations. However in each

⁷It would have been possible to directly enter the URN as a CNAME, TXT or clever NAPTR record and avoid delegation. However the above method is chosen as to ensure that those creating annotation have some 'skin in the game' and to ensure a verifiable level of commitment on the annotators side. It also totally puts the onus of publishing and maintaining 'their' comment onto the commentator.

model the naming authority of the resource is to be relied on to produce a complete and honest list; and the naming authorities of the annotators are to be relied on to get the URN's of the annotations.

4. STORAGE MANAGEMENT

As URNs provide long term persistent references to both the actual data and the metadata local storage is of secondary importance. A simple caching proxy layer ensures that both types of data are always available through one interface. What needs to be accessible at all times however are the indexes about mappings between identifiers and metadata to allow for local searches. As it is not possible to reconstruct the original data (or metadata) from these indexes⁸, the search engine generally returns a URN.

This problem can be faced in two different ways: (i) the name service only provide simple identifier resolution and mapping between its domain names to common unique identifiers; and it delegates any metadata retrieval request to a third part application or name service; (ii) the name service is able to provide either identifier resolution, mapping and serve metadata requests by leveraging on some local configured metadata backend storage (e.g. an RDF storage).

In our system prototypes we have been using a hybrid solution based on both the approaches; in the Image Showcase [6] system we have configured the name service to do simple identifier resolution (URN-to-URL mapping basically) by using ad-hoc localized hashables to systematically map identifiers to URLs; and the other way round; where each metadata request was redirected to a simple HTTP GET on the corresponding RDF/XML physical file (or remote URL) containing the description of the resource being requested. In a second case, the RDF Assessment System, we have configured the name service to directly use an RDF-store [23] to manage all mappings for the identifier resolution and the metadata requests; the physical mappings between URNs and the corresponding URLs have been defined using simple RDF properties like `owl:sameAs` between the corresponding identifier. All the metadata requests are then served by the RDF-store back end using ad-hoc RDQL and RDF API queries to return *coincise bounded resource descriptions* [24][25] of the resource.

4.1 Mutations and Provenance

A key aspect of any distributed system is dealing with mutations of data; typically on record level. For example take the name, email and postal address of an 'employee' who moves houses one day. Indexes based on the RDF model are not particularly suited for dealing with these type of mutations as conceptual records are broken down into a set of 'facts'. These are expressed as triples which are topologically near each other in a graph; normally such a collection references the same 'mother domain' node which contains something akin to a (foreign) key. Unfortunately other data about that same 'mother domain' created by entities not involved in the above example record (e.g. a record from a workers-benefits insurance company) points to the same location and is not easily distinguishable. Dealing with this requires careful tracking of the provenance; and essentially a book keeping of a 4th element outside the traditional RDF triple space. [26][27] [28].

5. CONCLUSION

The name resolution mechanism described in this paper proved to be an essential element required to solve the practical reference problems found when developing applications based on RDF/S.

⁸Even if the data is converted 1:1 into RDF and stored as triples, sequential order is one of the things potential missing.

Key is not just the problem of managing unique names for resources but also of keeping track of all resources and resource descriptions handled, annotated or referenced in the system in an efficient way. The URN name mechanism was used as the base service for all gathering and editing processes in the applications. In particular this solution is useful when managing *clouds* of rapidly changing and *federated* information and merging them in larger applications. The name resolution mechanism is based on Internet standards and allows for a wide range of different naming policies and social contracts. We believe this solution will be crucial in a broad variety of distributed applications based on RDF and RDF Schema.

6. REFERENCES

- [1] e. Dave Beckett, *RDF/XML Syntax Specification (Revised)*, Feb. 2004. W3C Recommendation.
- [2] e. Dan Brickley, R.V. Guha, *RDF Vocabulary Description Language 1.0: RDF Schema*, Feb. 2004. W3C Recommendation.
- [3] "the 'friend of a friend' (foaf) project home page," <http://www.foaf-project.org/>.
- [4] Z. Bjelogrić, D. van Gulik, and A. Reggiori, "Making business sense of the semantic web," *ISWC2003*, October 2003. <http://www.aseantics.com/presos/ISWC2003/rdfbiz.pdf>.
- [5] R. Guha, "Semantic web in industry," *WWW2003 Conference*.
- [6] "Image showcase - service part of the earth observation portal," <http://earth.esa.int/showcase/>.
- [7] S. S.R.L., *Earth Watching*, European Space Agency, Dec. 2003. work in progress.
- [8] S. S.R.L., *WADI, National River and Transport Authority Netherlands*, Sept. 2003. work in progress.
- [9] "Life sciences identifier (lsid) resolution protocol home page," <http://www-124.ibm.com/developerworks/opensource/lsid/?Open&ca=daw-ws-dr>.
- [10] K. Sollins and L. Masinter, *Functional Requirements for Uniform Resource Names*, Dec. 1994. RFC 1737.
- [11] R. Moats, *URN Syntax*, May 1997. RFC 2141.
- [12] K. Sollins, *Architectural Principles of Uniform Resource Name Resolution*, Jan. 1998. RFC 2276.
- [13] L. Daigle, D. van Gulik, R. Iannella, and P. Faltstrom, *Uniform Resource Names (URN) Namespace Definition Mechanisms*, Oct. 2002. RFC 3406.
- [14] M. Mealling, *Dynamic Delegation Discovery System (DDDS) Part One: The Comprehensive DDDS*, Oct. 2002. RFC 3401.
- [15] M. Mealling, *Dynamic Delegation Discovery System (DDDS) Part Two: The Algorithm*, Oct. 2002. RFC 3402.
- [16] M. Mealling, *Dynamic Delegation Discovery System (DDDS) Part Three: The Domain Name System (DNS) Database*, Oct. 2002. RFC 3403.
- [17] M. Mealling, *Dynamic Delegation Discovery System (DDDS) Part Four: The Uniform Resource Identifiers (URI)*, Oct. 2002. RFC 3404.
- [18] M. Mealling, *Dynamic Delegation Discovery System (DDDS) Part Five: URIARPA Assignment Procedures*, Oct. 2002. RFC 3405.
- [19] A. L. e. Tim Bray, Dave Hollander, *Namespaces in XML*, Jan. 1999. W3C Recommendation 14 January 1999.
- [20] B. Wellington, *Secure Domain Name System (DNS) Dynamic Update*, Nov. 2000. RFC 3007.
- [21] P. Mockapetris, *Domain names - implementation and specification*, Nov. 1987. RFC 1035.
- [22] M. Ohta, *Incremental Zone Transfer in DNS*, Aug. 1996. RFC 1995.
- [23] D. v. G. A. Reggiori, "Rdfstore," <http://rdfstore.sourceforge.net>.
- [24] N. Patrick Stickler, *URIQA: The Nokia URI Query Agent Model*, 2003. .
- [25] e. Andy Seaborne, *Fetching RDF : Data Objects*, 2003. .
- [26] G. Klyne, *Contexts for RDF Information Modelling*, Oct. 2000. .
- [27] S. Russel, *Quads*, Aug. 2002. .
- [28] D. Beckett, *Contexts Thoughts*, 2003. .