Linking Data from RESTful Services

Rosa Alarcon
Departamento de Ciencia de la Computacion
Pontificia Universidad Catolica de Chile
ralarcon@ing.puc.cl

Erik Wilde
School of Information
UC Berkeley
dret@berkeley.edu

ABSTRACT

One of the main goals of the Semantic Web is to extend current human-readable Web resources with semantic information encoded in a machine-processable form. One of its most successful approaches is the Web of Data which by following the principles of Linked Data have made available several data sources compliant with the Semantic Web technologies, such as, RDF triple stores, and SPARQL endpoints. On the other hand, the set of the architectural principles that underlie the human-readable Web has been conceptualized as the *Representational State Transfer (REST)* architectural style. In this paper, we distill REST concepts in order to provide a mechanism for describing REST (i.e. human-readable Web) resources and transform them into semantic resources. The strategy allowed us to harvest already existing Web resources without requiring changes on the original sources, or ad-hoc interfaces. The presented strategy aims to contribute to the availability of more semantic datasets and become a further step to lower the entry barrier to semantic resources publishing.

Categories and Subject Descriptors

H.3.5 [**Information Storage and Retrieval**]: Online Information Services—*Web-based services, Data sharing*

General Terms

Design, Documentation, Languages

Keywords

REST, Web Data, Crawling

1. INTRODUCTION

There is an increasing interest in the relationship of *Representational State Transfer (REST)* [13] , and the Semantic Web, which has resulted in various approaches varying from the semantic annotation of Web resources, to middleware that mediates resource handling. Followed approaches, resemble the strategies of more traditional SOAP/WSDL semantic services and neglect basic REST properties. REST principles are somehow related to Linked Data principles in the sense that resources have a unique identifier (URI), that must be dereferenceable through HTTP; resources are interlinked, and by following those links new resources can be discovered. However, differences arise when getting deeper into the principles and rationale of both fields. For instance, on the Linked Data side, research projects aim to create large collections of RDF data by transforming structured data sources into RDF using specialized mappings, and exposing the generated RDF dataset as RDF triple stores, often with *SPARQL* endpoints. Although this strategy make available large collections of RDF data, they result also in centralistic approaches where access is typically mediated through a single "endpoint" (e.g. a dump of the whole site, an SPARQL endpoint, a Tabulator-like interface, etc.) and due to the heterogeneous nature of the data sources interfaces, they require sophisticated mechanisms to retrieve, process, and publish the information [9], which challenges the scalability and accuracy of the expose data since it can be outdated.

One of the main tenets of REST is the primacy of resources that are uniquely identified by *opaque URIs*, that is, in order to avoid coupling between clients and servers, no assumptions must be made about the structure of the URI [26]. REST requires a *uniform interface*, that is, a set of operations or methods with known semantics that changes the *state* of the resources. The interface depends on the URI scheme, for HTTP, the standard methods are `GET`, `PUT`, `POST`, `DELETE`, and `OPTIONS`. Methods are external to the resources, and are invoked by sending standard messages to the Web server indicating the URI of the requested resource, the method, the payload of the message and metadata.

A resource can have multiple *"representations"* that follow a standardized format or media type (e.g., `text/html`, `application/xml`, etc.) and can be negotiated with the Web server. *Representations* convey the *state* of the client's interaction within the application and contain hyperlinks that allow clients to discover other resources or change the state of the represented resource. Most importantly, REST services have no "endpoints", instead, they consists of a collection of resource URIs and a set of standard operations. This approach differs greatly from more traditional SOAP/WSDL, where a service publish an endpoint that exposes the set of available operations (i.e. URIs, encoding, parameters). Such operations have particular semantics that must be known in advance, in order to be properly invoked by the client (coupling).

REST yield loosely coupled design [26], where architectural concerns are separated among various standardized components such as routers, Web servers and Web browsers, resulting in a flexible, extensible and decentralized system simple to maintain and capable of massive scalability. Unlike distributed system, that hide distribution, decentralized systems make it explicit with the eventual goal of architect-

LDOW 2010, April 27, 2010, Raleigh, North Carolina.
.

ing a system of systems.

Based on these REST principles, we present the *Resource Linking Language (ReLL)*, that describes RESTful Web services and provides a natural mapping from the graph-oriented world of RESTful services (resources interlinked by links found in resource representations) to the graph-based model of RDF. By means of a ReLL description, a set of REST resources are described and exposed. Three applications were described and the resources harvested into a triple store. Section 2 briefly discuss related approaches, and section 3 describes the proposed language.

2. RELATED WORK

Semantic Web Services (SWS) for REST are mainly focused on providing a semantic description of a REST service. SA-REST [21] and hREST/MicroWSMO [20] provide a list of input and output parameters, methods, and URIs exposed by a REST service by means of property value pairs or RDFa [1] annotations. The description itself can be transformed to RDF using a GRDDL-based [12] strategy for generating a domain ontology in RDF, but no information about the REST resources themselves are retrieved.

The *Web Application Description Language (WADL)* [16] describe RESTful services and place resources, identified by predefined URI patterns, as first-class objects in a description. WADL only supports HTTP methods with *request* and *response* elements. These elements contain representations with a media type and (possibly) another URI. Representations contain typified parameters that in turn contain links to another resources' URI. Generally speaking, WADL attempts to completely describe all possible aspects of a RESTful service, down to predefined URI patterns and the ways in which query parameters have to be composed for certain types of requests, introducing a higher level of coupling for clients using such descriptions.

In the same line, Battle and Benson [6] propose semantic annotations, similar to SA-REST, and extensions to SPARQL in order to support an HTTP REST uniform interface. They also propose extensions to the payload of the HTTP REST methods (e.g., `PUT`, `DELETE` and `GET`) for maintaining consistency between a REST resource and its semantic equivalent (a triple) in some triple store.

The main problem of these approaches is that they follow the WSDL/SOAP service model; they do not align well with the principles of RESTful service design, since they disregard fundamental properties such as the hypermedia nature of REST, and the possibility of multiple representations for the resources. They also introduce coupling in their design by adhering to URI templates for describing the URIs of resources, input, and output parameters [25], or in the case of Battle and Benson, they introduce new semantics to the standard REST interface.

EXPRESS [4] is a SWS model that explicitly avoids the RPC-orientation of the approaches mentioned so far. It starts from HTTP's uniform interface, and then describes the available resources in an OWL ontology. However, the model of EXPRESS is a centralized one as well, because it is assumed that there is a complete description of a Web Service's available resources, and then this description is used to generate URIs for classes, instances, and properties.

On the Linked data side, the *Vocabulary Of Interlinked Datasets (voiD)* [3], describes datasets (sets of RDF triples) as well as the sets of Linksets, that is, triples where the subject belong to a dataset different than the object's dataset. Directionality of the links can be modeled, and other properties such as licensing (`dcterms:license`), the number of triples available in the dataset (`void:statItem`), the vocabularies used in the dataset, and a SPARQL endpoint, are also provided. voiD is accompanied of a Sitemap protocol extension that indicates the location (URI) of the voiD description so that (semantic) web crawlers can find it and use voiD's information to index the dataset. The *Silk-LSL* (Link Specification Language) [30] is an XML-based language that allows to define the rules (e.g. similarity metrics) and to find certain types of links (e.g. `owl:sameAs`) between two data sources automatically (that is, to discover Linksets in the terms of voiD).

voiD's focus is on providing access and discovery for already existing datasets by publishing metadata, but a more granular approach (i.e. information about the retrieved resources themselves) is not considered. *Silk*, allow to better index large centralized collections of RDF data, and discovering dependencies between these datasets. While these approaches are central to increasing the amount of linked data on the Web, they are rather expensive because they are based on a lot of specialized mapping and publishing work for just transforming one dataset [9].

LDDR, the Link-based Resource Descriptor Discovery [17] is a proposal submitted to IETF that focuses on the resources rather than the datasets. It allows resources to indicate their descriptor's location by using links in three modes, the `<LINK>` element available in markup representations that support typed-relations such as (X)HTML and Atom; the HTTP Link Header; and a Link-pattern contained in the resource's description document located at `{host}/.well-known/` directory. In all three cases, the descriptor itself depends on the resource's URI, in the form of `{resource uri};about`. Unlike the last approach, the former two would require to modify the resources in order to include the `<LINK>` elements either in the resource's code or in the server side in order to process the HTTP Header.

As for the descriptor itself, XRD[1], the Extensible Resource Descriptor defines a small set of elements describing the resource's URI (and URI template), an XML signature, the expiration date, and links to other resources. Links are also annotated with metadata such as the target resource URI (and its URI template), mediatype, and the `<rel>` property as defined by the HTTP Header Link Relationship Types. This approach, implies that there must exist an XDR document per resource (since the set of links is often different for each resource) which introduces high coupling and may be impractical for a Web-scale application.

If XRD focuses on individual resources, POWDER, the Protocol for Web Description Resources[2] recommended by W3C aims to facilitate the description of groups of resources identified by *Internationalized Resource Identifiers (IRIs)*. An *iriset* (a set of IRIs, not a set of resources) can be defined in terms of the properties of such IRIs, that is, the accepted schemes (e.g. http, https), hosts, paths, and ports defined via regular expressions. The *iriset* properties are described by a *descriptorset* element that groups *restriction attributes* such as `certified` (indicates if the description certifies another resource) and `sha1sum` (providing a SHA-1 sum of

[1]http://docs.oasis-open.org/xri/xrd/v1.0/xrd-1.0.html
[2]http://www.w3.org/TR/2009/REC-powder-dr/

the described resource); and *annotation properties*, such as, `displaytext` (a descriptive text), `displayicon` (an image URI) and `seealso, label, comment` that provide a related resource URI, a description and a comment respectively. Both restriction attributes and annotation properties have well-defined semantics and can be translated automatically to OWL, thought, they describe high level attributes. An additional property, `typeof` is also translated into `rdf:type` and allows to specify a class for all the elements of an *iriset*. For instance, we could define the `http:\twitter.com` *iriset* and indicate later that all the elements identified by such URI belong to the class `twitterPublicTimeLine`. Provenance information describing author, date and validity period (attribution) is also provided.

Unlike XDR, POWDER refers to group of resources identified by URI patterns (not URI templates) without requiring changes in the resources, furthermore, POWDER makes possible to assign a class to the group of resources facilitating later complex operations such as SPARQL queries. On the negative side, POWDER facilitates the description of group resources but not it does not provide support for the resources discovery or an automatic harvesting process.

In the approach described by Futrelle [14], RDF is used as the "integration layer" in a scenario of heterogeneous data sources, and the main focus is on harvesting well-known and cooperating data sources. This approach can be applied to a variety of data sources, but they have to be cooperating in the sense that they expose RDF themselves. The harvester's main role is to be notified of new and updated data, and to pull it in from these sources. While this scenario uses RDF's power to unify heterogeneous data sources on the metamodel level, it is only applicable in closed and cooperating settings. In our approach, data sources are not required to publish RDF themselves. As long as access to data is provided through RESTful services, they can be harvested and used as RDF. A weakness of the current implementation is that updating is not supported in a way that allows efficient incremental updates, but we plan to address this issue in our future work mentioned in Section 6, where we describe extensions to our language that represent update services (and thus the ability to use those for incremental updates) on the language level.

SOFIE [29] focuses on information extraction from Web resources, and ANGIE [27] on using both extracted information and Web services endpoints, for building a more interactive system that does not require an exhaustive crawl of data, but retrieves information on demand. SOFIE thus falls into the category of approaches that start from resource representations, and use information retrieval methods to extract RDF from them. The current implementation of ANGIE focus on the dynamics of query processing in the RDF data managed by the system, and uses a hardwired set of Web services as the back-end. Similar to SA-REST, it uses a set of lowering/lifting transformations to translate the results of function calls from and to RDF. ANGIE focuses on SPARQL processing (the framework is able to use Web services while processing SPARQL queries), and less on the ability to easily accommodate a large variety of RESTful services.

Deimos [5] is another system that starts with information found on Web pages or through Web forms, and then uses semantic analysis to map the syntax of these representations to semantically richer information. Instead of relying on the richness of links discovered in known resources, though, the approach taken in Deimos uses tagging services to discover new resources.

Finally, another attempt to provide a bridge between REST and the semantic Web is the W3C work in progress of an RDF vocabulary representing the HTTP protocol [3]. The approach captures properties such as the message exchanged (including the HTTP headers), the request (including the method and URI) and the response (including the HTTP status code number) with the goal of facilitating relevant tasks such as content negotiation, as well as additional HTTP headers registered by the *Internet Assigned Numbers Authority (IANA)*.

3. RESOURCE LINKING LANGUAGE

Considering the related work, we derived a set of requirements for a REST resource description language that consider REST constraints. For instance, in order to avoid coupling URIs must be opaque, they must support multiple representations, and must consider linking among resources as a fundamental property. In order to consider current installed infrastructure, it must require minimal or no intervention for existing Web resources; in order to scale it must support a partial description of the resources that can be later completed and/or modified, it must describe both single resources and groups of resources as well as the relationships among them, and finally it must be simple in order to lower the entry barrier for future developers and foster its adoption.

The main constraints for designing RESTful services are resource *identification*, *linking*, and a *uniform interface* through which linked resources can be accessed. By *linking* we refer to one of the core aspects of RESTful services, that is the use of *hypermedia as the engine of application state (HATEOAS)*, which means that service interactions that in non-REST approaches result in server state, are actually implemented as clients following links to resources representing that state. This results in services that are resource- and link-centric, and thus a description language for RESTful services should focus on these two aspects.

The other two main constraints of REST, *self-describing messages* and *stateless interactions*, are more a question of how resource representations are retrieved, and how state is handled when interacting with services. For the purpose of *designing* RESTful services, all of these design issues are relevant. For the purpose of *describing* a RESTful service interface, the most important aspects are the resources representations that can be retrieved, the ways in which these can link to other resources, and the protocol interactions that may be required to access those resources. The service semantics also require an understanding of the semantics of the representations involved in the interactions with the service, but for the mere description of a service's interface, these semantics are not required.

Figure 1 shows the schema of ReLL. Elements are shown as rectangles and attributes as dashed rectangles. Sequences are depicted as a circle with the character "S". A *service* exposes a set of one or more *resources* that have a unique identifier (*xml:id*), names and descriptions (human-readable labels) and optionally a *URI* pattern which describes the constraints for the identifiers expected to be used for spe-

[3]http://www.w3.org/TR/HTTP-in-RDF10/

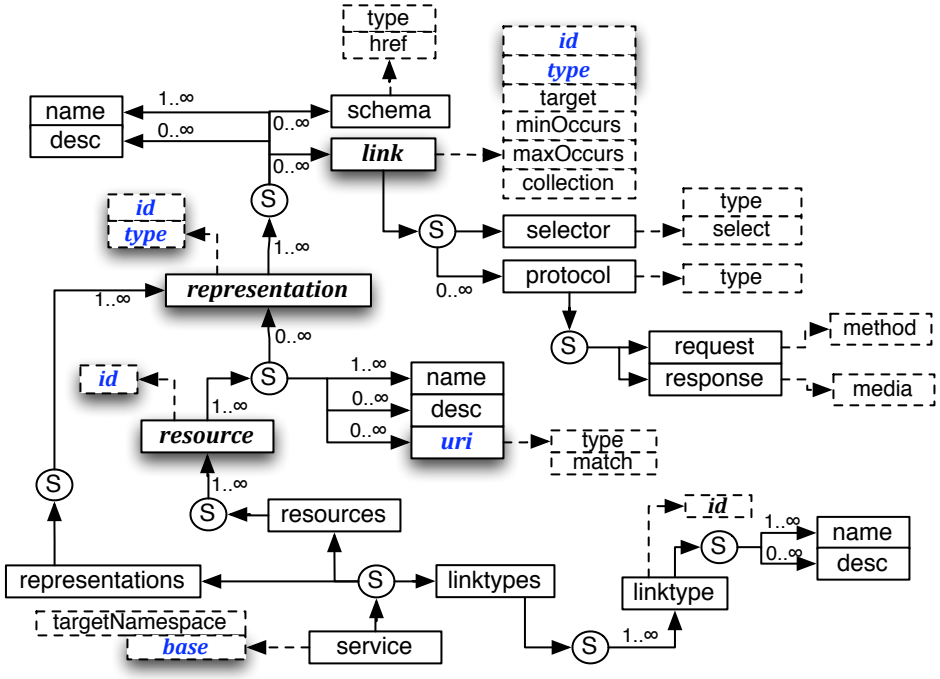

Figure 1: ReLL Description Schema

cific resources (*match*). A resource may have *representations*, which are the serialization of the resource in some syntax. This design naturally supports multiple representations for resources, but it does not support, per se, the common practice of some Web services that use different URIs for different representations of the same resource (such as two URIs with `.xml` and `.json` suffixes, if these are two supported representation formats).[4] We discuss this issue further down, when we are discussing link types.

Representations can be associated with *schemas* for possible validation (if schemas exist). Representations can also be defined as part of the service directly, in which case they are *abstract*, which means that they are not associated with any concrete resources. The most important use cases for abstract representations are conventions for media or data formats that should be described, so that they can be reused as a foundation for describing concrete resource representations. A real-world use case for this scenario is an abstract representation describing the media type `application/xml`, that serves as the basis for the abstract representation describing the `application/atom+xml` media type for feeds according to Atom [24], which in turn serves as the basis for the abstract representation describing the paged feeds media type (i.e., feeds implementing feed paging [22]). Eventually, a concrete service providing a resource may use paged feeds and thus the resource types its representation with the abstract "paged feed" representation. The rationale behind this design is that various representations in this chain of representations define different linking mechanisms (paged feeds extend Atom with new link relationships), and the effective set of link types that can appear in a concrete resource using the paged feed representation thus is the union of these different link types. Representations can be based on other representations, but only on abstract representations. The other use case of abstract representations is representations that are derived from concrete representations, such as a collection of representations that is available through a paging mechanism in representation formats.

Each representation can contain any number of *links*. A link is retrieved from the representation by using *selectors*. Selectors depend on the representation format, and thus their definition and interpretation may depend on a language (*selector type*) that is appropriated for a certain representation. For instance, for XML representations, the most popular example for a selector mechanism is the *XML Path Language (XPath)* [11, 7], which allows structured selections within XML document trees. A link defines a possible association leading from the resource's representation containing the link to another resource as determine by the *target*. Instead a resource URI, the target contains a valid resource id in order to avoid coupling with the resources' naming scheme.

A link has a *link type* which represents the semantics of the link, but ReLL does not make any attempt to formalize the semantics; link types have a name and a description and thus can be documented in a service description, but their semantics are outside of the scope of the description language. Links can also contain *protocol* descriptions which for each link specify the rules that govern the interaction with the linked resource. This is important because links in RESTful services not only have application-specific semantics, following the links also may require different ways of using the uniform interface provided by a certain protocol.

[4]Such variations in the representation's URIs could easily be covered by a URI pattern for the resource ending with `.(xml|json)`, but the variation of the suffix alone would not imply that it does not actually refer to a different resource, but only to a different representation.

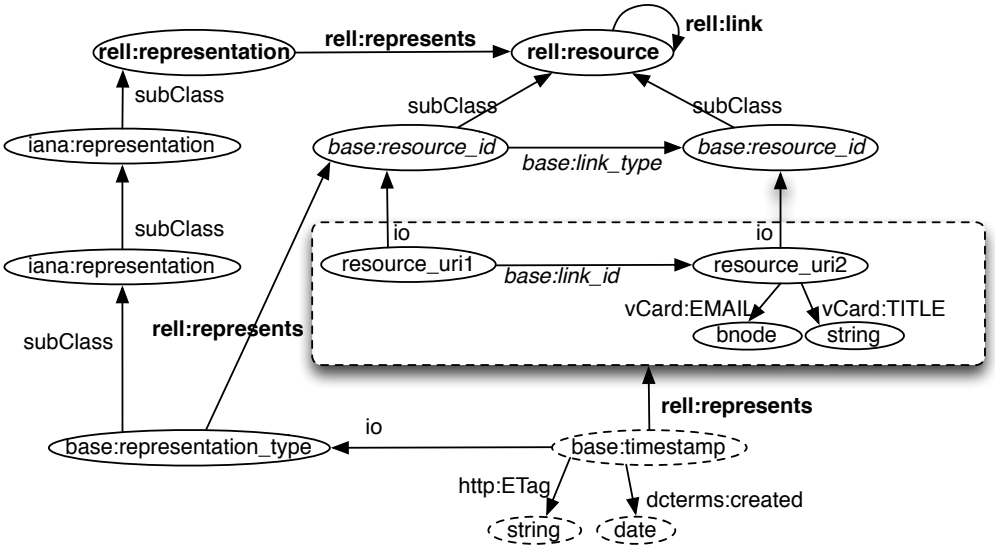

Figure 2: Generating RDF triples from ReLL descriptions

Thus, it is possible for each link to specify how this link has to be traversed using a specific protocol. Practically speaking, this means that after a link's URI has been determined (for example by extracting the URI using a selector), the protocol is determined by inspecting the URI's scheme, and then the *protocol* description might give additional hints about how to use methods or compose entities for invoking the uniform interface. Thus protocol descriptions are just one (the interface-specific) part of describing link semantics.

4. FROM RELL TO RDF

ReLL main elements such as *resource*, *representation*, and *link* serve as the core elements for a RDF/OWL minimal vocabulary shown in Figure 2 under the "rell" namespace. *Resource*, and *representation* are concepts while *link*, and *represents* are predicates. Since ReLL describes a REST application, it is used to generate a domain ontology for the application. The *resource id* annotated in ReLL is used as the resource's type and the *link type* as the predicate that relates two resources. Domain specific resources are also subclasses of the `rell:resource` entity, and currently form a domain-specific vocabulary by using the ReLL *service*'s attribute *base*.

We are maintaining the actual REST resources' URIs to identify them in the realm of the Semantic Web, however they are considered `instances of` the domain-specific classes discussed before. REST resources are linked together with a *link id* instead of a *link type*. REST resources' themselves can be transformed to RDF following a GRDDL approach. For instance, in Figure 2, a resource is annotated with properties defined in the vCard vocabulary, including simple (literals) and complex attributes (e.g. the EMAIL is generated as an internal blank node). Naturally, the proper vocabularies depend on the resources.

With this approach, it is possible to retrieve a graph of triples describing a REST resource (URI and attributes) and its relation to another REST resource, as shown by the dashed rectangle in Figure 2. The resulting graph [10] is named with an ID or timestamp (e.g., `base:r123456789`) that refers to the source or *representation* from where the graph information was collected. The representation is an instance of the *representation type* defined in the ReLL description for the retrieved REST resource.

Representations are subclasses of a concrete media type that can be derived from abstract representations or abstract *media types* as annotated in the ReLL descriptions. Abstract representations are supported as classes that serve as the basis for other abstract or concrete representations. For representations, the upper ontology contains all standardized media types from the IANA registry as classes.

The representation is then part of the provenance information obtained when retrieved the REST resources (see dashed elements in Figure 2). Other information such as the ETag property served by the Web server when retrieving the REST resource is also collected if available; the date when the information was retrieved (and hence the named graph was created) is also annotated. Other information as indicated by [18] could also be included in future developments.

5. IMPLEMENTATION

As a proof of concept, we have implemented *RESTler* [2], a crawler that follows the rules defined by ReLL descriptions in order to harvest REST resources. A complementary component (a Translator) transforms the retrieved resources into RDF. Figure 3 describes the principal components of the approach. Rectangles represent software components, UML note figures are used to represent files, straight lines represent information flow required in the configuration phase of the process (static), while dashed lines represent information flow that take place while the crawling process is being executed (dynamic).

RESTler, is a crawler that parses and uses *ReLL* descriptions as instructions for retrieving REST services' resources. The crawler takes as input an XML document which is a ReLL description, and a set of *seed URIs* (Figure 3), and

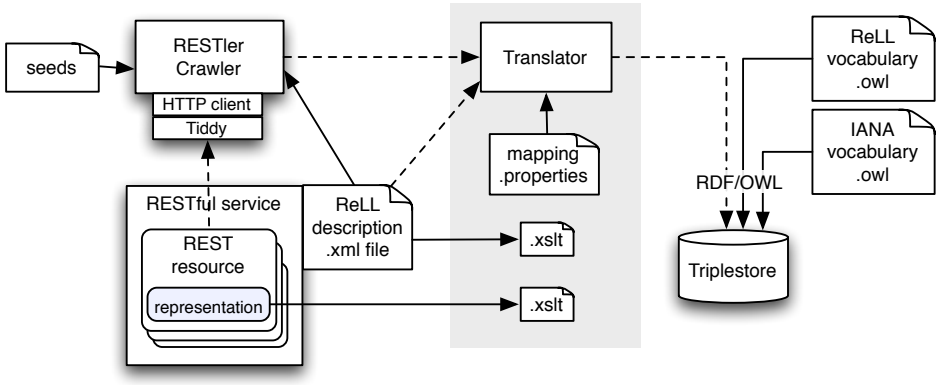

Figure 3: RESTler Architecture

produces as output a typed graph of the crawled resources and the links connecting them. The crawler also takes as input authentication information, only basic authentication is supported (username and password sent in the HTTP request) currently, but we plan to extend the crawler in order to support other authentication schemes (e.g., OAuth, AuthSub).

The crawler parses the description file, dereferences the initial URI (seeds), and retrieves the resource `representation` considering the protocol, request method, and resource media type provided. Currently we support HTTP (an HTTP client), and HTML, XHTML, Atom, JSON, RSS, and XML as media types, and only the `GET` method. But the crawler can be extended to support other media types, protocols and request methods.

The resource URI is matched against a regular expression that defines the *resource type* or `id`. From the retrieved `representation`, the crawler obtains the list of embedded `links` to other `representations` by applying an XPath expression (`selector`). The link's `target` indicates the expected resource type and requires additional information such as the `protocol`, and `request method` to follow and the expected `media` type. If the `target` is not present in the `link` element, a "nofollow" condition is implied, since it is not possible to crawl the linked resource (i.e., there is no information about the media type, protocol, request method or expected resource type).

It is possible as well to support computed links, that is, links that are calculated.[5] The crawler also evaluates whether the resource fulfills certain restrictions such as the type of the linked resources (`target` attribute), and the cardinality of the retrieved links (`minOccurs` and `maxOccurs` attributes for the `selector` element). These restrictions are optional and allow the crawler to determine whether the resource is well-formed and satisfies the preconditions given in the service description.

For each graph retrieved, a Translator is invoked for generating RDF triples based on the ReLL description, that is, the subjects (resources' URIs), properties (`rdf:type`, `base:link` `id`) and objects (linked resources' URIs or values), as well as provenance information (`base:timestamp`). Additional information is obtained trough XSLT files transforming resources into RDF sentences, as indicated for the corresponding mapping file. Each ReLL document is transformed into RDF with a generic XSLT generating an ontology specific to each application domain. Generated named graphs are stored in a triple store. We use Sesame 2.0 as triple store and the system is implemented in Java. Sesame supports named graphs as quads, and we use the fourth component for storing provenance information.

Finally, for each retrieved resource, the crawler recursively repeats the whole process.

5.1 School/Twitter/Flickr and User Matching

We applied *RESTler* to four scenarios: a subset of the Web site of the Information School at UC Berkeley, and two well known REST-based applications, Twitter and Flickr. The fourth service provide mappings among the users in each of these domains so that we can establish useful equivalences by means of an `owl:sameAs` property. *ReLL* descriptions where created for each scenario and we retrieved 11,353 resources, 22,309 links among them which generated 55,548 triples.

Figure 4 presents the ontology that was generated after transforming ReLL descriptions into RDF through a generic XSLT definition. The image was generated using OntoViz[6] and was later refined for readability. The upper left corner presents the `representation` classes and their corresponding `iana` media-types (e.g. `iana-app:xhtml+xml`, `iana-app:atom+xml`, `iana-app:xml`, `iana-txt:html` and images media types). The right-hand side presents the classes that model the UC Berkeley school domain's resources (e.g. `school:person`, `school:course`, etc) and the relationships among resources (e.g. `school:person-course`).

The left-hand side shows the classes corresponding to the Flickr domain (e.g. `flickr:photostream`, `flickr:photo`, etc) and their relationships (e.g. `flickr:photo-sizes`). At the bottom of the figure, a subgraph describes the classes that model the Twitter domain (e.g. `twitter:follower`, `twitter:user`, etc) and the hyperlinks or relationships among them (e.g. `twitter:status-reply`). At the center of the figure the minimal ontology described in Figure 2 is highlighted in bold and italics.

[5]Based on the ongoing work on the *URI Template* [15] language, it might in the future be possible to define additional ways in which a URI can be composed based on input values obtained from the current representation.

[6]A Protege plugin that generates .dot files

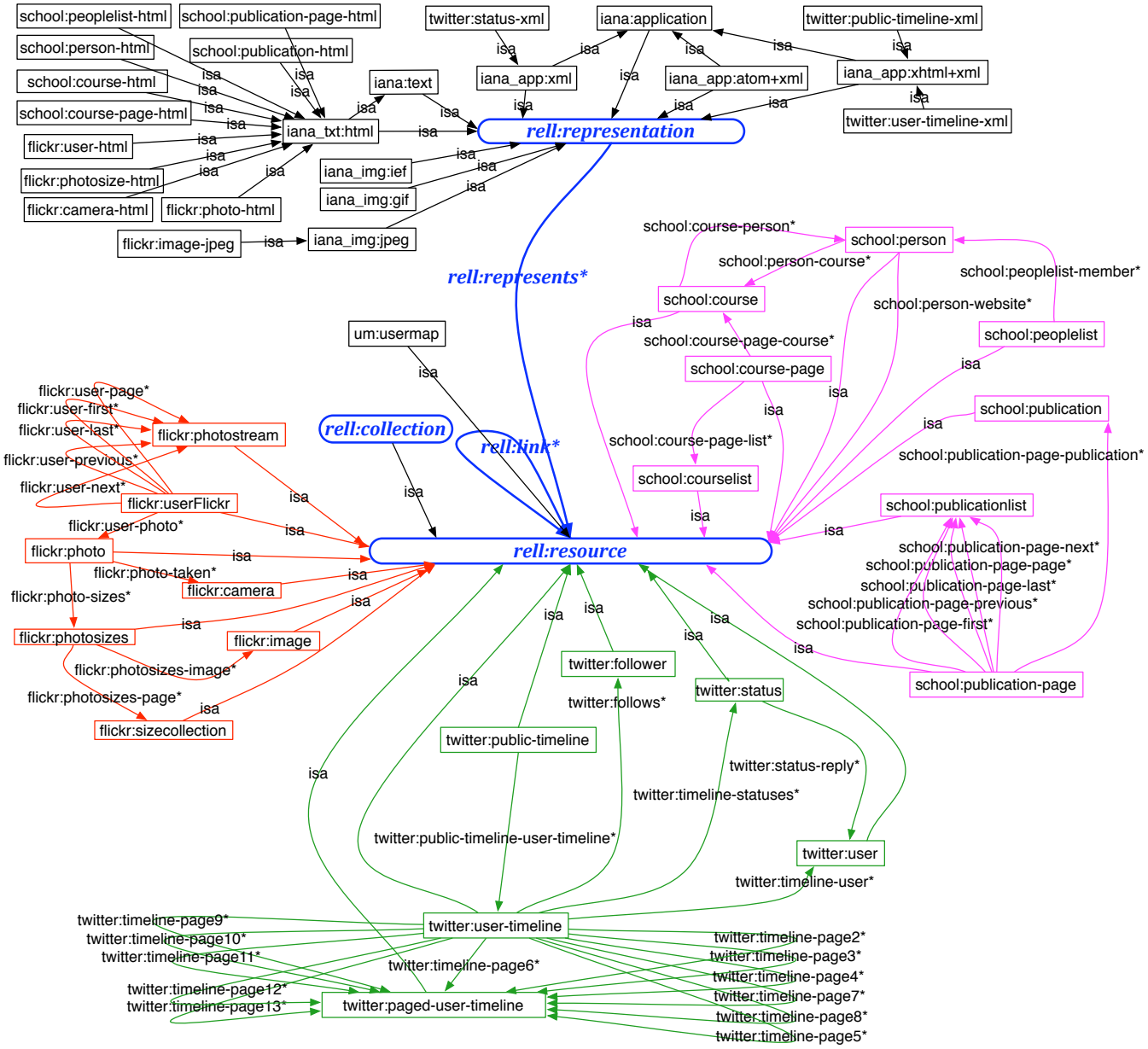

Figure 4: Resulting ontologies generated from ReLL descriptions

```
<http://www.ischool.berkeley.edu/people/faculty/erikwilde> a school:person ;
   vCard:FN "Erik Wilde" ;
   vCard:ADR _:node14m5kienpx1603 ;
   vCard:TITLE "Adjunct Professor" ;
   vCard:ORG _:node14m5kienpx1604 ;
   vCard:EMAIL _:node14m5kienpx1606 ;
   vCard:TEL _:node14m5kienpx1607 ;
   vCard:URL <http://dret.net/netdret/> ;
   vCard:PHOTO <http://www.ischool.berkeley.edu/files/imagecache/profile-pic/DSC_0176.JPG> ;
   school:person-website <http://dret.net/netdret/> ;
   school:person-course <http://www.ischool.berkeley.edu/programs/courses/242> ,
     <http://www.ischool.berkeley.edu/programs/courses/152> ,
     <http://www.ischool.berkeley.edu/programs/courses/190-waim> ,
     <http://www.ischool.berkeley.edu/programs/courses/290-wa> .
```

Figure 5: Describing an instance of a `school:person` resource using N3 notation

Collections of resources can be also identified. For instance, at the bottom of the figure, the arcs between two resources are depicted, the `twitter:user-timeline`, and the `twitter:paged-user-timeline` described a pagination relationships, that is, 13 pages of the `twitter:user-timeline` were collected and the pagination scheme is describe as links that lead to a numbered page (e.g. `twitter:timeline-page2`, `twitter:timeline-page3`, etc). For the case of Flickr and the Information School the pagination scheme considers links such as the first, last, next and previous page.

The fourth RESTful service, the Usermap is show as a single class near the center of the figure. This is because the ReLL file contains only one class of resource (the usermap), that is, an XML list mapping the users' URIs between the other three applications.

The REST resources themselves are transformed to RDF following a GRDDL approach. Figure 5 shows the attributes obtained for individuals of type `school:person`. Notice that it is possible to annotate the relationships between the REST resource (`erikwilde`) and its attributes. In the figure these relationships are annotated with *vCard*, but other information models can be used.

6. CONCLUSIONS

The REST community is still discussing whether RESTful services even should be described, and how such a description language could increase the coupling between a service provider and a service consumer, so that REST's goal of loosely coupled services could be compromised. We are taking a pragmatic position and claim that it is important to keep in mind that any kind of contract will introduce some coupling, that even loosely coupled services need a shared set of assumptions, and that a more formal way of describing those assumptions will help service providers and consumers in service documentation and consumption. A recent upswing of discoverable links between Web resources (such as an uptake of microformats [19]) has led to the idea of a central registry for link relationships in the realm of *Web linking* [23], but this activity is still under active development.

Our model is yet a static description of RESTful services that does not cover the cases in which new resources or identification and access schemes are introduced. However, such a description allows to describe the status quo and the cases which a client should expect, and therefore they also allow to reliably discover cases in which these constraints are not satisfied anymore, for example when new representations or new identification and access schemes are used.

Furthermore, this kind of RESTful service description can also include the set of preconditions that must be satisfied by a client to be able to consume a service. Should these preconditions change (because the service changes), then an analysis of the description of the preconditions used by the client allows the client to detect the change (for example, a new representation format has been introduced), and to react in an appropriate way (for example, alerting the client manager, attempting a fallback, or abort). By supporting the description of a set of preconditions, the description language can achieve loose coupling [26] and still allow clients to detect when they encounter something that they have not been designed for. As for future work, we are planning on considering more complex data models that support also methods such as PUT, DELETE and POST allowing us to model resources that can be modified, and its relation with the SPARQL proposals for supporting such operations [31].

Our minting process consist of selecting the appropriated name for the namespace (`base`), resource IDs, link IDs, link types, and representation IDs. In the example presented in Figure5, the resource instance's namespace and predicates chosen for this description correspond to the vCard, but other properties (e.g. `foaf`) could be also used. We believe that the selection of such properties must be responsibility of the ReLL designer. Furthermore, the properties used in the ReLL description itself (e.g. `school:person`) could be also described using Linked Data vocabularies. By following this approach the results of RESTler (e.g. triples datasets) could be better integrated with other Linked Data sources and the Linked Open Data cloud

By considering the URIs corresponding to REST resources, a natural content negotiation with the Web server will be possible in order to retrieve an RDF-friendly media type (e.g. `application/rdf+xml`) or the human-readable Web version of the same resource. As for limitations, we require to prepare a ReLL document for each REST service. This approach has been successfully followed by others such as Virtuoso's Sponger, that prepares Sponges or Cartridges tailored for an application interface such as REST APIs, known metadata such as MS Office, or known Web sites such as YouTube. RDB2RDF[7] is also an ad-hoc approach

[7]http://www.w3.org/2005/Incubator/rdb2rdf/

that transforms RDBMS to RDF representations.

We believe that by choosing Web technologies such as XPATH, XSLT and XML as a the basis for ReLL documents, we are lowering the entry barrier to the semantic resources publishing, since most Web developers have the knowledge and tools required to create their own ReLL description. This approach also allows developers to control the information they are collecting. Our next challenge is to further facilitate the creation of ReLL documents by supporting the dynamic and automatic generation of ReLL descriptions. One of the challenges of this goal is the fact that we need to design an specific XSLT for each resource type in order to harvest specific information. A fully automatic approach would require information retrieval, text mining and probably machine learning techniques which greatly increases the costs of the transformation an rises the entry barrier for technology adopters.

Having a document such as ReLL may serve as an intermediate layer that automatic agents can use also as a contract describing the capacities of a REST service and translating them into RDF triples, by following the semantics (types) made explicit in the document. Our approach can be seen as a complement to proposals such as voiD, since voiD describes the resulting datasets but does not support the triples harvesting process. Our approach will allow any Web content provider to publish ReLL descriptions for others to crawl their Web sites, or third-parties to develop a Web site's description that accommodates their needs. The crawler's result is a *dataset* that can be then described using voiD. Silk, can be also used for the definition of additional link patterns such as the user mapping that we created manually in this version; and LDDR's linking techniques can be also applied, since it may allow resources to link to their descriptions.

We have placed strong emphasis in a decoupled approach, where the components of the architecture maintain certain degree of independence, and require knowledge and tools already available and familiar to most Web developers, and provide a simple model that may result familiar again to Web developers. Our final goal is to contribute in making available more semantic information while keeping a lower entry barrier for developers.

7. ACKNOWLEDGMENTS

This work was partially funded by CONICYT/Bicenntenial Becas-Chile 2009.

8. REFERENCES

[1] Ben Adida, Mark Birbeck, Shane McCarron, and Steven Pemberton. RDFa in XHTML: Syntax and Processing — A Collection of Attributes and Processing Rules for Extending XHTML to Support RDF. World Wide Web Consortium, Recommendation REC-rdfa-syntax-20081014, October 2008.

[2] Rosa Alarcón and Erik Wilde. RESTler: Crawling RESTful Services. In *19th International World Wide Web Conference Posters*, Raleigh, North Carolina, April 2010. ACM Press.

[3] Keith Alexander, Richard Cyganiak, Michael Hausenblas, and Jun Zhaox. Describing Linked Datasets. In *2nd Workshop on Linked Data on the Web*, Madrid, Spain, April 2009.

[4] Areeb Alowisheq, David E. Millard, and Thanassis Tiropanis. EXPRESS: EXPressing REstful Semantic Services Using Domain Ontologies. In Bernstein et al. [8], pages 941–948.

[5] José Luis Ambite, Sirish Darbha, Aman Goel, Craig A. Knoblock, Kristina Lerman, Rahul Parundekar, and Thomas Russ. Automatically Constructing Semantic Web Services from Online Sources. In Bernstein et al. [8], pages 17–32.

[6] Robert Battle and Edward Benson. Bridging the Semantic Web and Web 2.0 with Representational State Transfer (REST). *Journal of Web Semantics*, 6(1), 2008.

[7] Anders Berglund, Scott Boag, Donald D. Chamberlin, Mary F. Fernández, Michael Kay, Jonathan Robie, and Jérôme Siméon. XML Path Language (XPath) 2.0. World Wide Web Consortium, Recommendation REC-xpath20-20070123, January 2007.

[8] Abraham Bernstein, David R. Karger, Tom Heath, Lee Feigenbaum, Diana Maynard, Enrico Motta, Krishnaprasad, and Thirunarayan, editors. *8th International Semantic Web Conference*, volume 5823 of *Lecture Notes in Computer Science*, Chantilly, Virginia, October 2009. Springer-Verlag.

[9] Uldis Bojārs, John G. Breslin, Vassilios Peristeras, Giovanni Tummarello, and Stefan Decker. Interlinking the Social Web with Semantics. *IEEE Intelligent Systems*, 23(3):29–40, May 2008.

[10] Jeremy J. Carroll, Christian Bizer, Pat Hayes, and Patrick Stickler. Named Graphs, Provenance and Trust. In Allan Ellis and Tatsuya Hagino, editors, *14th International World Wide Web Conference*, pages 613–622, Chiba, Japan, May 2005. ACM Press.

[11] James Clark and Steven J. DeRose. XML Path Language (XPath) Version 1.0. World Wide Web Consortium, Recommendation REC-xpath-19991116, November 1999.

[12] Dan Connolly. Gleaning Resource Descriptions from Dialects of Languages (GRDDL). World Wide Web Consortium, Recommendation REC-grddl-20070911, September 2007.

[13] Roy Thomas Fielding and Richard N. Taylor. Principled Design of the Modern Web Architecture. *ACM Transactions on Internet Technology*, 2(2):115–150, May 2002.

[14] Joe Futrelle. Harvesting RDF Triples. In Luc Moreau and Ian Foster, editors, *International Provenance and Annotation Workshop (IPAW 2006)*, volume 4145 of *Lecture Notes in Computer Science*, pages 64–72, Chicago, Illinois, May 2006. Springer-Verlag.

[15] Joe Gregorio. URI Template. Internet Draft draft-gregorio-uritemplate-04, March 2010.

[16] Marc Hadley. Web Application Description Language. World Wide Web Consortium, Member Submission SUBM-wadl-20090831, August 2009.

[17] Eran Hammer-Lahav. Link-based Resource Descriptor Discovery. Internet Draft draft-hammer-discovery-03, March 2009.

[18] Olaf Hartig and Jun Zhao. Using Web Data Provenance for Quality Assessment. In *First International Workshop on the Role of Semantic Web in Provenance Management*, Washington, D.C., October 2009.

[19] Rohit Khare and Tantek Çelik. Microformats: A Pragmatic Path to the Semantic Web. In *15th International World Wide Web Conference Posters*, Edinburgh, UK, May 2006. ACM Press.

[20] Jacek Kopecký, Karthik Gomadam, and Tomas Vitvar. hRESTS: An HTML Microformat for Describing RESTful Web Services. In *2008 IEEE/WIC/ACM International Conference on Web Intelligence*, pages 619–625, Sydney, Australia, December 2008.

[21] Jon Lathem, Karthik Gomadam, and Amit P. Sheth. SA-REST and (S)mashups: Adding Semantics to RESTful Services. In *First IEEE International Conference on Semantic Computing (ICSC 2007)*, pages 469–476, Irvine, California, September 2007.

[22] Mark Nottingham. Feed Paging and Archiving. Internet RFC 5005, September 2007.

[23] Mark Nottingham. Web Linking. Internet Draft draft-nottingham-http-link-header-08, March 2010.

[24] Mark Nottingham and Robert Sayre. The Atom Syndication Format. Internet RFC 4287, December 2005.

[25] Cesare Pautasso. Composing RESTful services with JOpera. In Alexandre Bergel and Johan Fabry, editors, *International Conference on Software Composition 2009*, volume 5634 of *Lecture Notes in Computer Science*, pages 142–159, Zürich, Switzerland, July 2009. Springer-Verlag.

[26] Cesare Pautasso and Erik Wilde. Why is the Web Loosely Coupled? A Multi-Faceted Metric for Service Design. In Quemada et al. [28], pages 911–920.

[27] Nicoleta Preda, Fabian M. Suchanek, Gjergji Kasneci, Thomas Neumann, Maya Ramanath, and Gerhard Weikum. ANGIE: Active Knowledge for Interactive Exploration. In *35th International Conference on Very Large Data Bases (VLDB 2009)*, pages 1570–1573, Lyon, France, August 2009. ACM Press.

[28] Juan Quemada, Gonzalo León, Yoëlle S. Maarek, and Wolfgang Nejdl, editors. *18th International World Wide Web Conference*, Madrid, Spain, April 2009. ACM Press.

[29] Fabian M. Suchanek, Mauro Sozio, and Gerhard Weikum. SOFIE: A Self-Organizing Framework for Information Extraction. In Quemada et al. [28], pages 911–920.

[30] Julius Volz, Christian Bizer, Martin Gaedke, and Georgi Kobilarov. Discovering and Maintaining Links on the Web of Data. In Bernstein et al. [8], pages 650–665.

[31] Erik Wilde and Michael Hausenblas. RESTful SPARQL? You Name It! — Aligning SPARQL with REST and Resource Orientation. In Walter Binder and Erik Wilde, editors, *4th Workshop on Emerging Web Services Technology (WEWST 2009)*, pages 39–43, Eindhoven, Netherlands, November 2009.