Data integration at scale: Creating webs of data with RDF

How resource-oriented thinking benefits data integration

Brian Sletten (brian@bosatsu.net)  
President  
Bosatsu Consulting, Inc.  
31 March 2015

In the first installment of a five-part series on data-integration standards and technologies, Brian Sletten introduces the Resource Description Framework (RDF) as the basis for a new set of standards called Open Services for Lifecycle Collaboration (OSLC). As part of the World Wide Web Consortium (W3C) Semantic Web technology stack, RDF is designed to facilitate information integration among multiple participants without the need for extensive precoordination.

View more content in this series

One of the fundamental purposes of software in the modern world is to produce and consume data. It's a simple task to create a data model or schema, map it to an object model, and serialize the data in an externalized format to a single recipient who shares your context. But doing so at scale with multiple partners and contexts remains one of the more difficult tasks that software producers face. As a result, we tend to commit only to integration projects for which the return on investment is clear. Missed opportunities and systemic inefficiencies can add to the opportunity cost of not connecting the right data sources.

Data integration at scale isn't solely a technical problem; it's also a social problem. To exchange information, we must agree on what it means — again, easy in small, point-to-point interactions but unimaginably complex at higher levels of participation. We run into limitations of culture, linguistics, psychology, perception, and politics. Changing requirements, shifting priorities, and a desire to collaborate with more partners make the situation even more untenable in dynamic domains. And in a world of fragile, snapshot-in-time technologies such as relational database schemas, XML schemas, and existing code libraries, this flux is amplified almost beyond control. We need to maintain agreement about the names and shapes of domain elements and relationships long enough to deliver software before anything changes again.
The need for consensus in the maelstrom of modern business chaos is one of the main contributors to *The Software Problem*, or the fact that it’s difficult to produce quality software in predictable timeframes. This need rarely gets the attention it deserves. With domain models expressed implicitly and explicitly in relational table schemas, object-relational mapping (ORM) files, code libraries, and service descriptions, our capacity to roll out new features or fix defects is hindered by overly dependent lifecycles. So we employ ever-larger teams with ballooning budgets that require political influence that’s hard to shift or realign. We create master data management (MDM) plans and enterprise data warehouses (EDWs) to try to unify, standardize, and control a collective understanding of a domain to facilitate a tractable exchange of information.

About this series

This series introduces, explores, and applies global standards that address large-scale data-integration challenges faced by developers, architects, and data managers daily. The cross-platform, language- and application-independent technologies that the series covers enable the integration of information in databases, documents, spreadsheets, and service APIs. The data models and tools that you’ll learn about can make your work easier and have a material impact on your organization.

The software lifecycle management domain is a grand example of how hard this problem can be. We need to manage requirements, software artifacts, defect tracking, and system changes using combinations of tools from dozens of partners along with homegrown implementations. Some aspects of the domain might cross these spaces, but each one is also a separate vertical slice of the larger problem. It will never be possible to achieve consensus among all stakeholders on how to specify all of the details of all of the domain elements.

This article series explains how large-scale data-integration problems are being addressed in useful, new ways. The series focuses incrementally on an exciting approach in the software lifecycle domain: the *Open Services for Lifecycle Collaboration* (OSLC) initiative. OSLC is based on the Representational State Transfer (REST) architectural style and Semantic Web technologies such as the *Resource Description Framework* (RDF) and the *Linked Data Platform* (LDP). The first three installments in the series lay out the technological foundation for OSLC. In the final two installments, I’ll introduce you to the OSLC project and demonstrate its application.

The web: A scalable platform of interlinked documents

The web is based on well-understood technologies that work together as a flexible, scalable platform of interlinked documents. It would have been nearly impossible to design what the web has become from scratch. But a group of standards designed to solve specific problems built upon one another to enable this amazing ecosystem to evolve and thrive.

The first technology is the Uniform Resource Locator (URL). This naming scheme — based now on the Uniform Resource Identifier (URI) specification — serves the dual role of identifying arbitrary content in a global information space and acting as the handle by which we can request this content. We share documents and content with the world by sharing these identifiers in emails, tweets, wikis, television ads, the sides of buses, or even on a napkin. Because the identifiers are grounded in domain names that we control, we strike a nice balance between central mediation and edge freedom. Because the identifiers have global scope but are generally tied to specific
domains, they can be shared without fear of collision. I can't create identifiers in domains you control and vice versa, but I can create as many as I want whenever I want in my own domain. Although good URL design is encouraged for stability, longevity, and flexibility, I can pick any name and share it with you for off-the-cuff collaboration.

“Although we're generally used to seeing URLs attached to documents or web applications, the REST architectural style allows them to be applied to arbitrary information resources also.”

Although we're generally used to seeing URLs attached to documents or web applications, the REST architectural style allows them to be applied to arbitrary information resources also. We might refer to an account via the identifier http://example.com/account/id/12345 or a product as http://example.com/product/id/sku/9823423. Effectively, we are identifying arbitrary bits of information in a nearly infinite information space.

To access the state of the resource, or interact with it in some way, we can engage the next useful web technology: HTTP. HTTP provides a uniform interface to any resource produced by anyone in any way. As the client of the information, I know nothing about the details of how the resource is implemented, nor do I care. If I issue a GET request to the resource, something returns a default representation. If I want the resource in a different form, I can try to ask the server to give it to me in that form if it can.

The final main technology that makes the web work is HTML, which is what web resources typically return to a browser client. HTML supplies the representation of the resource being requested along with all of the things I can do in the resource’s current state. I might be able to follow a link to another document, establish an authenticated session by logging in and providing suitable credentials through a form, and so on. The coupling in this system is between the browser and the returned representation, not the browser and the server. The server can relocate resources and change implementation technologies, and the browser client happily continues to work the next time it asks for a resource. The client learns where to find content, and what the user can do with it, from the representation — not because it knows something about the server.

This much of the web is widely understood, even by nontechnical people. With only slightly different thinking, the same ideas can be applied to information more generally. In the process, organizations can drive down the cost of data integration and increase chances for reuse. And they can gain the flexibility to connect data sources in ways that create new opportunities to achieve large financial benefits.

**RDF for interlinked webs of information**

“Once we make the leap to thinking about assigning resolvable identifiers to arbitrary bits of information, we can start to imagine interlinked webs of information.”
Once we make the leap to thinking about assigning resolvable identifiers to arbitrary bits of information, we can start to imagine interlinked webs of information. We might want to indicate that the subject of an article is the country of Japan and that the author is Bob. We can imagine wanting to hang arbitrary associations off of any resource to express related facts and content. People go to certain schools and work for certain organizations and are born in specific parts of the world and have family members and pets and interests and friends. We would love to be able to "intertwine" (Ted Nelson's term) all of these things. What we need is a flexible data structure and a nonconflicting way of referring to all of these things.

The URI standard already gives us the ability to assign arbitrary global-scope identifiers to anything. Now we want to apply the standard to concepts rather than documents or services. In the example from the previous paragraph, we want identifiers for the document, the country of Japan, and Bob; these are the entities of the system. What's new and different here is that we also want identifiers for the has subject and authorship relationships. The document's identifier will be the subject node. The identifiers for Japan and Bob will be the object nodes. The relationship identifiers will connect these subjects to the values as named, directed arcs. A graph is a flexible structure to use for this purpose. With a graph — unlike the table of a database or the tree of an XML document — we can add individual associations at any time without impacting the rest of the structure.

Most graph systems fail by trying to store the entire graph in a single storage system. Such attempts don't usually work if the data exceeds a certain size. But by using resolvable web identifiers (URLs), we can make the web be our graph. Now we can scale up to an arbitrarily large information space. We don't think about putting the web into containers. We put things into the web.

"Any RDF system can consume RDF from any other RDF system without any type of coordination."

RDF is a W3C standard data model with exactly these properties. The entities use URIs (preferably resolvable URLs) and are connected via directed relationships that are also identified this way. We can refer to both entities and relationships via global identifiers that resolve to explain themselves. If we serialize the graph into a standard format, we end up having completely portable data. Any RDF system can consume RDF from any other RDF system without any type of coordination. The application or users might not know what to do with the data yet, but they can at least consume it and then see what they have ingested.

I'll use a seemingly silly example to show how RDF works. If I asked you to modify your system to accept the single statement that my birthday is May 26, you wouldn't. If your tables aren't set up to accept the concepts of people and their birthdays, you have no place to put this fact. It would cost too much to change your tables and class structures. Consuming my birthday isn't a serious consideration, but this problem persists in less ridiculous scenarios. Part of the issue is that we tend to think of entities as self-contained things that we store in single locations. We don't think of conceptual entities about which we learn things from multiple sources. The difference is between a closed-world model and RDF's open-world model. In the open-world model, we accept the fact that
anyone can share facts about anything. We might never know everything about an entity, topic, or subject. This assumption might confound IT managers who want to own and control everything, but it enables a system to remain flexible and capable of capturing arbitrary facts from any source.

We need to identify me, we need to identify the birthday relationship, and we need to know how to represent my birthday's date. I chose the URL https://w3id.org/people/bsletten to refer to myself. The URL is registered with a community that has pledged to help maintain the stability of the registered identifiers — currently a GitHub repository that you can fork, modify, and issue pull requests against to add your own identifiers.

I configured this identifier to redirect to a file that describes me via an HTTP 303 See Also redirect. This is currently one of the W3C Technical Architecture Group (TAG) recommendations for noninformation resources. I don't serialize particularly well, so you can't have a representation of my current state by asking for it. To differentiate a reference to me from a document about me (which can be dereferenced and resolved), the 303 alerts the client that the request was valid but that it cannot be directly fulfilled. Instead, the response includes a header called Location that points to the document that describes me. (I'll revisit this process in the next installment.)

So, now that we know how to refer to me (the subject of the statement), I need a term to refer to birthday that is defined as the day someone was born. Nothing is stopping me from picking my own term. Something like http://bosatsu.net/ns/birthday would work, particularly if I put a description of the term at that location so anyone could resolve the property identifier to figure out what it meant. Fortunately, though, I don't need to do that. A community called Friend of a Friend (FOAF) has agreed on a series of terms related to distributed social networks and communities on the web. This specification includes a widely used term for birthday: http://xmlns.com/foaf/0.1/birthday, which means exactly what I want it to mean. If you ask for it via HTTP, you're taken to a document that describes the collection of terms. The documentation also informs you what the format should be: mm-dd. We now have a single fact involving a subject (me), a predicate (http://xmlns.com/foaf/0.1/birthday), and a value (05-26). When the three are combined, we imagine a directed graph reflecting this statement, as shown in Figure 1. The subject is the node. The predicate is the arc. The value is on the other side of the arc.
Now I have everything I need to publish my birthday as a machine-processible fact. I'll use a simple RDF serialization called N-Triples to store the statement in a file. This format has one fact per line terminated by a period:

```
<https://w3id.org/people/bsletten> <http://xmlns.com/foaf/0.1/birthday> "05-26".
```

This file can be stored on a file system or published on the web. Consequently, any RDF system should be able to consume this fact. If users of this other system don't know who I am, they can resolve the subject identifier and find out more. If they don't know what FOAF birthday means, they can do the same thing.

Perhaps I want to publish my full name also. Again, I could make up my own name for the term `name`, but it isn't necessary because FOAF already has a widely used term for this purpose:

```
<https://w3id.org/people/bsletten> <http://xmlns.com/foaf/0.1/name> "Brian Sletten".
```

The graph in Figure 2 reflects this new fact about me.
Here's where things get interesting. Regardless of whether the two facts were stored in the same file or two different files, the properties and values accumulate for the subject if they are read into the same model (as shown in Figure 3), because both statements refer to the same subject with the same identifier.

**Figure 3. Two facts expressed as a directed graph**

Here's how I use N-Triples to store two RDF facts about the same subject:

```xml
<https://w3id.org/people/bsletten> <http://xmlns.com/foaf/0.1/birthday> "05-26" .
<https://w3id.org/people/bsletten> <http://xmlns.com/foaf/0.1/name> "Brian Sletten" .
```

The more you learn about a subject, the more arcs hang off the node. There's no real limit to what you can know about something, and arbitrary extra facts do not impact the rest of the graph. Nor does it matter where the facts come from; they can all connect.

**Coming later**

Strategies exist for separating facts based on source, level of trust, classification, and so on, but you can ignore those issues for now. It's also possible to annotate graphs with...
provenance metadata using the PROV vocabulary if it's important to track how it is produced and modified. Again, I'll leave these matters for later in the series.

It's powerful that anyone can say anything about any subject via properties, but we also want the ability to organize our resources based upon what types of things they are. You can think of this in terms of class set membership. If a resource refers to a person, we could say that that individual is a member of a Person class (the convention is to use initial capital letters for class names). The same person, if he or she is an engineer, could be a member of the Engineer class. There's no limit to how many sets you can be a member of from any number of vocabularies.

It's trivial to mix schemas, but for now I'll stick with the trusty FOAF vocabulary. Because one of its primary goals is to talk about people, it includes a Person class that I can use. I need a special predicate to indicate that a resource is an instance of a class. This is a fundamentally important predicate to express, so RDF has a term for this type of relationship: http://www.w3.org/1999/02/22-rdf-syntax-ns#type. This type of statement has different semantics from the property relationships you've seen so far, but it's expressed the same way:

```ntriples
<https://w3id.org/people/bsletten> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://xmlns.com/foaf/0.1/Person> .
```

The additional arc in the graph in Figure 4 also looks similar to what you've already seen.

**Figure 4. Instance-of relationships in the directed graph**

![Graph](https://w3id.org/people/bsletten)

http://www.w3.org/1999/02/22-rdf-syntax-ns#type

http://xmlns.com/foaf/0.1/Person

Finally, as you might imagine, the combination of all three statements expressed in N-Triples continues to accumulate:
Figure 5 shows the three facts expressed as a graph.

**Figure 5. Three facts expressed as a directed graph**

```
<https://w3id.org/people/bsletten> <http://xmlns.com/foaf/0.1/birthday> "05-26" .
<https://w3id.org/people/bsletten> <http://xmlns.com/foaf/0.1/name> "Brian Sletten" .
<https://w3id.org/people/bsletten> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://xmlns.com/foaf/0.1/Person> .
```

The salient point in the story so far is that the data model supports this notion of accretion of information. At any time you can learn something new. With the use of standard global identifiers, a standard data model, and standard serialization, the notion of incompatible data doesn’t apply, and it takes virtually no effort to integrate more data.

**Converting between RDF formats**

The N-Triples format isn’t tremendously human-friendly. The format is reasonable for dumping facts in an easy-to-parse way, but all the repetition makes it difficult to see the content related to a specific subject. Now I’ll show you a nicer format called Turtle, which stands for the Terse RDF Triple Language. (Several other formats are available, including an ugly XML-based one called RDF/XML.)

Converting between formats is trivial. One way is to use a tool called **rdfcat** from the [Apache Jena](https://jena.apache.org/) project. Here, I ask **rdfcat** to convert the triples to Turtle format:

```
> rdfcat -out ttl basic.nt
<https://w3id.org/people/bsletten> a <http://xmlns.com/foaf/0.1/Person> ;
   <http://xmlns.com/foaf/0.1/birthday> "05-26" ;
   <http://xmlns.com/foaf/0.1/name> "Brian Sletten" .
```

You can see that in Turtle, each subject is mentioned only once as a subject (the same identifier might be in the object position of another relationship). Semicolons separate each fact about the subject. A period terminates the facts that we know about this subject in this file. You could, of course, learn other facts from other sources.

Assume that I've published another Turtle file on the web that contains an extra fact about me. The file uses the http://xmlns.com/foaf/0.1/depiction property to indicate that an image of me is available on the web. The property has a similar format but only one fact:

```> http get http://bosatsu.net/turtle/brian.ttl
HTTP/1.1 200 OK
Accept-Ranges: bytes
Access-Control-Allow-Origin: *
Content-Length: 124
Content-Type: text/turtle
Date: Mon, 10 Mar 2015 04:56:39 GMT
ETag: "1049e7-7c-50fd9f13a4140"
Last-Modified: Tue, 24 Feb 2015 18:46:53 GMT
Server: Apache/2.2.16 (Debian)

<https://w3id.org/people/bsletten> <http://xmlns.com/foaf/0.1/depiction>
<http://bosatsu.net/images/briansletten.jpg> .
```

As you might expect by now, the data in the local file and the data in the remote file are easily merged into a common model. Because I use the same identifier in both places, the data integrates trivially:

```> rdfcat -out ttl basic.nt http://bosatsu.net/turtle/brian.ttl
<https://w3id.org/people/bsletten> a <http://xmlns.com/foaf/0.1/Person> ;
  <http://xmlns.com/foaf/0.1/birthday> "05-26" ;
  <http://xmlns.com/foaf/0.1/depiction>
    <http://bosatsu.net/images/briansletten.jpg> ;
  <http://xmlns.com/foaf/0.1/name> "Brian Sletten" .
```

rdfcat is one of dozens of tools and libraries that know how to manipulate the standards and merge data across sources. Through the use of these standards, the cost of data integration drops to next to nothing. You can imagine, for example, the benefits of connecting the contents of a sales spreadsheet to a third-party data source that contains ratings for products sold. Combining these sources might make it obvious that the support department needs to be strengthened quickly because you are selling many poorly rated products.

**Conclusion**

RDF is an incredibly powerful data model, and subsequent installments in this series will expand on its possibilities. For now, it's enough to be aware of the simplicity and flexibility of consuming arbitrary data sources. You can understand why the OSLC initiative wants to be able to connect information from the various products and participants into an integrated, linked whole. With a little bit of careful planning, resources representing requirements can be connected to the code that satisfies them. The tests of the code can be connected to the results of running the tests. Specific changes to the source code can be connected to particular events and even the individuals who were responsible.
The use of URIs and the graph model cause some concern to developers more familiar with the simplicity of the JSON key/value pair format. The extra complexity is not unwarranted — we're talking about interlinked graphs of data, not simple serialization of one or more objects — but don't let this be a concern. Standards like JSON-LD enable you to trivially bridge JSON and RDF to gain the best of both worlds.

Stay tuned for some fun, amazing, and powerful revelations as this series continues to explore how resource-oriented thinking can benefit data integration on the web and in the OSLC project.
Resources

Learn

- **RDF**: Explore the RDF page in the W3C Semantic Web wiki.
- **OSLC**: Visit the community website for the OSLC project, which develops standards that make it easy and practical for software lifecycle tools to share data with one another. OSLC, now part of the OASIS Open Standards Network, was initiated by IBM and its partners.
- **N-Triples**: Learn more about the N-Triples line-based syntax for RDF graphs.
- **Turtle**: Check out the Turtle syntax for RDF.

Get products and technologies

- **Apache Jena**: You need Jena to use `rdfcat`.

Discuss

- **Semantic Web Interest Group**: Participate in a forum that supports developers and users of Semantic Web technologies.
About the author

Brian Sletten

Brian Sletten is a liberal-arts-educated software engineer with a focus on forward-leaning technologies. His experience spans many industries, including retail, banking, online games, defense, finance, hospitality, and healthcare. He has a B.S. in computer science from the College of William and Mary and lives in Auburn, California. He focuses on web architecture, resource-oriented computing, social networking, the Semantic Web, data science, 3D graphics, visualization, scalable systems, security consulting, and other technologies of the late twentieth and early twenty-first centuries.

© Copyright IBM Corporation 2015
Trademarks
(www.ibm.com/developerworks/ibm/trademarks/)