Data integration at scale: OSLC and the Linked Data Platform

Discover how Semantic Web technologies are facilitating tool integration

Brian Sletten
August 05, 2015

Learn how the Open Services for Lifecycle Collaboration (OSLC) community applies the concepts that are introduced in this series to collaborative software development. OSLC uses Linked Data and other Semantic Web standards to make it easier for requirements-tracking, development, and testing tools to work together throughout the software lifecycle.

View more content in this series

This series has incrementally introduced you to a collection of standard technologies that make it easy to integrate data from various sources. The first article, "Creating webs of data with RDF," discussed a resource-oriented data model for easy information sharing. Use of standard identifiers and graphs for entities and their relationships lets information flow and accumulate regardless of where it comes from. The second article, "Query RDF data with SPARQL," adds standard query mechanisms to ask questions of shared information. The third article, "Linked Data," demonstrates the use of these technologies to create large-scale integrations of potentially hundreds of sources. Now that you've seen what's possible with RDF, SPARQL, and Linked Data, this installment and the next introduce an application of these technologies to the software lifecycle management space. First, though, I'll take a quick detour into a couple of experiences that helped to excite my interest in writing about these technologies.

Tools and the software lifecycle

As a software developer, I've always been aware of the value of good tools for my day-to-day tasks: change-management (CM) systems, build tools, testing tools, and, more recently, continuous integration and delivery tools. But early in my career, I was dismissive of tools that were designed for the business side of producing software. Requirements management, test-plan management, project management, and the like seemed to be expensive toys that added no real value. They seemed divorced from reality — always aspirational, never reflective of any proximate state of a project. (What developers worked on was Truth.) And there was no way to connect the information in one tool to the information in another.
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.

Then, a former boss, the late Brian Lyons of Number Six software, blew my mind by tying the entire Rational Software suite of tools together. By using Rational Rose as the hub, he connected Requisite Pro, Rose, ClearCase, and whatever Rational's testing tools were at the time. It was a revelation to understand what his proficiency with the tool suite unlocked. I saw him connect requirements to the use cases and realizations to the code to the tests to the test results and all code changes in-between. For example, Lyons used color coding on the Rose elements to indicate test coverage and status. With this integrated suite of tools, you might intertwingle the business requirements, the people who wrote them, the code, and the developers who produced the code.

If software failures occurred, you knew where they were, who produced them, and which requirements they were associated with. It became clear that software failure was an organizational issue, not necessarily the fault of a single individual. If requirements were poorly specified, it made sense that developers got confused when they designed and built what they thought was needed. The quality of the requirements, of the code, or of both might be responsible for the failure. Being able to determine the issue’s source was compelling. Historically, this was all pre-Agile, so the idea of measurement and integration across business and technical boundaries was exciting and new.

The only problem was that you needed wizard-level knowledge about a large and complicated tool suite interlinked by custom scripting languages. Lyons had been at Rational before cofounding his own company; he was a master of all of Rational's products, but most people wouldn't be. I felt like I was shown a brief view of the future, but reality wasn't there yet.

Several years later, I cochaired a conference track with Martin Nally, then CTO of IBM Rational. I learned from him that IBM Rational was trying to make tool integration possible again — through REST interfaces and XML structures. By this time, I had become a strong advocate of Linked Data and Semantic Web technologies for integration, and I lamented the use of custom XML document formats. But I was impressed that, in theory at least, arbitrary tools might talk to one another if tool makers came to an agreement about the formats of the exchanged content.

The IBM Rational initiative was called Open Services for Lifecycle Collaboration (OSLC). When I finally looked into OSLC, I was pleasantly astonished to learn that it was based on open standards that use RDF, HTTP, REST architectural principles, and Linked Data. The group had recruited dozens of participants along the way. Among the organizations that produce and use these vertical slices of standards-based tool data, you'll find financial institutions, blue-chip technology companies, defense firms, open source projects, government organizations, and developer-friendly startups.
Bringing tool producers together for a complex, consensus-driven specification initiative wasn’t in the cards. A lightweight, flexible, extensible, and standards-based approach made sense.

The group apparently concluded that consensus among all the players — across the various aspects of the Application Lifecycle Management (ALM) and Product Lifecycle Management (PLM) tool suites — was unlikely. So was selling an entire tool suite to organizations. The average software-development team had already committed to a CM system like Git or Subversion and integrated a myriad of other commercial, open source, or custom tools. Spending the time to bring all of these different tool producers together for a complex, consensus-driven specification initiative was also not in the cards. In the end, a lightweight, flexible, extensible, and standards-based approach made sense. These are the technologies that this series has already introduced you to. They work on the web, and they work as a web. This article and the next look at their application to a specific problem.

The OSLC mission

The OSLC community has produced an iterated series of specifications for integrating tools. The initiative is in the 3.0 stage and has been picked up by the OASIS standards group to carry the work forward. OSLC aims not to overextend its reach by overspecifying how tools can communicate, instead taking the approach of “just enough” integration. The effort is built on standards such as Linked Data and HTTP and practices like the REST architectural style. Specific workgroups tackle individual subdomains such as requirements management, change management, configuration management, testing, and the like, but they are all based on a core set of concepts and styles to remain interoperable.

The goals of the project are to build capabilities and vocabularies that make the system:

- **Scenario-driven**: Like use cases, the scenarios describe the activities that the tools might integrate. Use of scenarios ensures that the standards activity focuses on meaningful tasks and provides sufficient coverage to the requirements of the constituent members.
- **Incremental**: Rather than try to get everything right up front, standards work should add value to the ecosystem incrementally, and sooner, to validate the approach.
- **Loosely coupled**: Consistent with the web, clients of compatible tools should have no special insight into the implementation details of the server. This gives the tool producers freedom to make changes without affecting the clients and should enable the clients to work with a wider collection of tools.
- **Minimalistic**: The standards should be sufficient to meet the needs of the agreed-upon scenarios and should not try to exceed those requirements.

OSLC formed individual Technical Committees (TCs) to tackle the specification of terms and types as vocabularies for specific domains. Each TC can focus on what’s unique to, say, quality-management tools, but they’re encouraged to reuse terms in cross-domain integrations where it makes sense to do so. This continues the approach of minimal but sufficient (“just enough”) integration across the tools and scenarios.
After working through some of its early vocabularies and means of representing the artifacts of ALM systems, the OSLC community realized that actual running software would be necessary to help move these ideas forward. If it overspecified an architecture for integration, the impact on the implementations might drive tool producers away. If it underspecified the architecture, the tools might vary too significantly to easily and consistently share information.

**In transition**

The OSLC initiative is in a transition period. The move from version 2.0 to 3.0 has involved changing the project’s home and scope, and adopting the LDP as the basis for the architecture. Many of the main participants have changed. It can be a little unclear from the tutorials and documentation what is current and what is earlier data. OASIS is a large, complex organization that requires membership for access to many of the interactions. Fortunately, we are starting to see the web pages begin to be updated with a clarifying direction. You can find the latest activity on the [Workgroups](https://www.ibm.com/developerworks/opensource/oslc) page.

Rather than just solving the problem for specific domain elements (such as requirements or changes) individually, OSLC decided that a reusable platform for managing the various data types and their relationships to one another would be useful. At around the same time, the World Wide Web Consortium (W3C) started receiving feedback that members wanted a read/write platform for managing linked data. OSLC decided to join these other groups and became a key contributor by providing a reference implementation, which formed the basis of discussions about what eventually became the [Linked Data Platform](https://www.w3.org/TR/ldp/) (LDP) specification.

Four main TCs are under the OASIS OSLC banner:

- **OSLC Lifecycle Integration Core (OSLC Core)**: Manages the basic definitions of the elements that span various scenarios including how to create, retrieve, link, and update them by way of the LDP.
- **OSLC Lifecycle Integration for Automation (OSLC Automation)**: Defines REST endpoints and interaction models to support automation workflows.
- **OSLC Lifecycle Integration for Change and Configuration Management (OSLC CCM)**: Defines the elements for change, configuration, and asset management across application and product lifecycles.
- **OSLC Lifecycle Integration for Project Management of Contracted Delivery (OSLC PROMCODE)**: Defines the elements associated with project management activities.

Before I delve further into a working OSLC system in the final article, it will help to understand the architecture on which the tools are expected to be built.

**Understanding the Linked Data Platform**

“The LDP builds on the principles of REST, the web, and Linked Data but provides more structure to harmonize behavior across compatible containers.”

In theory, the resources we want to refer to can be hosted by any HTTP-savvy container. One of the major benefits of the loose coupling of the web is that clients don't need to know anything about the implementation of the server. For communities such as OSLC that want to engender
a community of interoperable tools, a slightly more detailed specification of behavior around the management of Linked Data resources is required. This is where the Linked Data Platform (LDP) comes in. The LDP builds on the principles of REST, the web, and Linked Data but provides more structure to harmonize behavior across compatible containers. If tool vendors understand and support these standards consistently, the ecosystem will be more likely to work together. Actual implementation details are still largely irrelevant, provided the resource-oriented behavior is supported.

The Linked Data Platform is largely about managing the state of Linked Data Platform Resources (LDPRs). These are arbitrary resources that can be stored in Linked Data Platform Containers (LDPCs). Containers aren't the same as web servers or databases. You can host multiple containers within an instance of the LDP. You can think of them as arbitrary logical containers in which you store resources. Common examples include a user's contact list, a project and its related defects, or a team and its members.

### Security and the LDP

To keep things simple, this tutorial's example ignores security entirely. LDPjs doesn't use authentication or authorization protection against the LDPCs and LDPRs stored in it. Any LDP implementation that you use in production should use an authentication and authorization protection model such as OAuth 2.

A client can interact with an LDPC to create, retrieve, update, or delete an LDPR through HTTP and its appropriate methods. A significant part of the LDP is about resource discovery. Clients learn about what resources are available in a container and how they relate to one another. The LDP has a namespace (http://www.w3.org/ns/ldp#) associated with the vocabulary terms that are reusable across any application of the concepts. When I use the ldp: prefix in examples, that's the namespace I'm referring to.

### Installing an LDP implementation

Install an implementation of the LDP so that you can play around with it. You have several choices. I've chosen LDPjs, an easy-to-use Node-based version. (See Resources for a link to a Java-based implementation you can try.) First install git, Node.js, and Mongo if you don't already have them. Start Mongo and then fetch the LDPjs code:

```bash
> git clone https://github.com/spadgett/LDPjs.git
Cloning into 'LDPjs'...  
remote: Counting objects: 495, done.  
remote: Total 495 (delta 0), reused 0 (delta 0), pack-reused 495  
Receiving objects: 100% (495/495), 256.19 KiB | 158.00 KiB/s, done.  
Resolving deltas: 100% (293/293), done.  
Checking connectivity... done.  
```

Now, start the server:
> npm install
... npm output
> node server.js
configuration:
{ listenHost: 'localhost',
  listenPort: 3000,
  scheme: 'http',
  host: 'localhost',
  port: 3000,
  context: '/r/',
  appBase: 'http://localhost:3000',
  ldpBase: 'http://localhost:3000/r/',
  mongoURL: 'mongodb://localhost:27017/ldp' }
App started on port 3000
Connected to MongoDB at: mongodb://localhost:27017/ldp

**Base container information**

At this point, you can see that the base container (as defined by the config.json file) is `http://localhost:3000/r/`. Using an HTTP client (in this case, `httpie`), the LDPC reveals its basic information:

> http get http://localhost:3000/r/
HTTP/1.1 200 OK
Accept-Post: text/turtle,application/ld+json,application/json
Allow: GET,HEAD,DELETE,OPTIONS,POST
Connection: keep-alive
Content-Type: text/turtle
Date: Sat, 13 Jun 2015 21:50:52 GMT
ETag: W/"eb95ad12674f4327408681c918a91f51"
Link: <http://www.w3.org/ns/ldp#Resource>; rel="type",
   <http://localhost:3000/constraints.html>; rel="http://www.w3.org/ns/ldp#constrainedBy",
   <http://www.w3.org/ns/ldp#BasicContainer>; rel="type"
Transfer-Encoding: chunked
Vary: Accept
X-Powered-By: Express

<http://localhost:3000/r/> a <http://www.w3.org/ns/ldp#Resource>,
   <http://www.w3.org/ns/ldp#RDFSource>,
   <http://www.w3.org/ns/ldp#Container>,
   <http://www.w3.org/ns/ldp#BasicContainer>;
   <http://purl.org/dc/terms/title> "LDP.js root container".

You can see that the container identifier is just another resource. You can interact with it as with any other HTTP-based resource. Here are some important observations about the container information:

- You can add new resources to the LDPC as defined by the LDP specification through `POST` as
  Turtle (`text/turtle`), JSON-LD (`application/ld+json`), or JSON (`application/json`):

```bash
> http post http://localhost:3000/r/ ...
```

**JSON-LD**

Turtle is a popular serialization of RDF among Semantic Web aficionados, but it's difficult for non-Semantic Web frameworks to use. **JSON-LD** is an increasingly popular...
You are also allowed to issue other HTTP methods against the container itself. Another requirement is support for the `OPTIONS` method so that clients can determine what they are allowed to do to the resource.

- You see that by default, the LDPC returns Turtle representations of the resources. The LDP specification also mandates support for ETags to allow clients to detect when they change:

  ```
  Content-Type: text/turtle
  Date: Sat, 13 Jun 2015 21:50:52 GMT
  ETag: W/"eb95ad12674f4327486861c918a91f51"
  ```

- Even though Turtle is the default representation, the LDPC should also support content negotiation. In this case, you see a successful request for a JSON-LD representation:

  ```
  > http get http://localhost:3000/r/ Accept=application/ld+json
  HTTP/1.1 200 OK
  Accept-Post: text/turtle,application/ld+json,application/json
  Allow: GET,HEAD,DELETE,OPTIONS,POST
  Connection: keep-alive
  Content-Type: application/ld+json
  ...
  {
    "@context": {
      "dcterms": "http://purl.org/dc/terms/",
      "foaf": "http://xmlns.com/foaf/0.1/",
      "ldp": "http://www.w3.org/ns/ldp#"
    },
    "@id": "http://localhost:3000/r/",
    "@type": [
      "ldp:Resource",
      "ldp:RDFSResource",
      "ldp:Container",
      "ldp:BasicContainer"
    ],
    "dcterms:title": "LDP.js root container"
  }
  ```

- Among the response headers are several [RFC 5988](https://tools.ietf.org/html/rfc5988)-compliant links. Two links have a `rel` relationship of `type`, indicating that the container is an instance of an LDP `Resource` and an LDP `BasicContainer`. This makes sense. You can retrieve a description of the container because it’s a resource, and you can add things to it because it is a container. There are a few different container types, but the `BasicContainer` supports simple interactions and containment relationships. Whereas `type` is an important relationship to RDF itself, other domain-specific relationships can also be expressed through link responses. In this case, there’s also an LDP-specific relationship, `constrainedBy`. This link points to a document that you are free to resolve to find out what kind of constraints apply to this particular LDPC and LDP implementation:

  ```
  Link: <http://www.w3.org/ns/ldp#Resource>; rel="type",
  <http://localhost:3000/constraints.html>; rel="http://www.w3.org/ns/ldp#constrainedBy",
  <http://www.w3.org/ns/ldp#BasicContainer>; rel="type"
  ```

- A quick note about the `Vary` response header:

  ```
  Vary: Accept
  ```

Because the server allows clients to ask for the same resources with different formats by using content negotiation, it’s crucial to advertise this fact to intermediary processors to avoid
caching issues. Here, the server announces that it varies its response based on the `Accept` header. This is an easily forgotten step that, if overlooked, can lead to maddening debugging sessions.

- Finally, a representation of the container resource itself can be seen as both Turtle and JSON-LD:

  ```html
c<http://localhost:3000/r/> a
<http://www.w3.org/ns/ldp#Resource>,
<http://www.w3.org/ns/ldp#RDFSource>,
<http://www.w3.org/ns/ldp#Container>,
<http://www.w3.org/ns/ldp#BasicContainer>;
```

Some of what was advertised in the response headers is replicated here. That might seem wasteful, but part of the value of RFC 5988 headers is that they provide useful information to clients that might not know how to parse the returned representation. You see in addition to a title for the container that it can be known as a couple of other types. It is up to the implementation, and how it was set up, to describe itself. Most LDPCs use LDP terms, but they’re not limited to those terms.

**The LDPjs HTML interface**

An empty container isn’t all that interesting. You’ve learned that you can POST new resources to the container, but for fun, look at the HTML interface that the LDPjs server also supports. It was advertised to exist at [http://localhost:3000](http://localhost:3000) when the server came up and can be seen in Figure 1. The user interface includes a cute visualization of the contents of the container, currently empty. This is what the little folder labeled `/r/` represents.
If you select the **POST** link just above the container URL, the UI switches to show an example bug report expressed in Turtle and shown in Figure 2. Notice first that the thing being described is not identified. It is an unnamed node in Turtle, because you, as the client, don't know what to call it. The server will give it identity and let you know what it is called after it's added to the container.
Next, notice that the Turtle is using an arbitrary namespace and vocabulary describing the resource being created. This is one of the value propositions of the LDP. Any resource of any type using any vocabularies can be stored and managed in an LDPC. If you click **POST**, this Turtle is sent to the server to be stored in the container. Feel free to add additional metadata about the fake bug to see how flexible the approach is.

After you add the bug, you should see a link between the /r/ folder and the new resource. Beyond the visual representation, you see that the connection also is advertised in the data about the container. The Turtle now looks like what's shown in Listing 1. Notice that there's a link to the new resource using the contains relationship. Because it might be unwieldy to send back all of the contained resources for large containers, an optional pagination extension to the LDP allows a client to easily walk the entire collection as needed.

**Listing 1. LDPC reflecting the new resource**

```
<http://localhost:3000/r/> a <http://www.w3.org/ns/ldp#Resource>,
<http://www.w3.org/ns/ldp#RDFSource>,
<http://www.w3.org/ns/ldp#Container>,
<http://www.w3.org/ns/ldp#BasicContainer>;
<http://purl.org/dc/terms/title> "LDP.js root container";
```

The new resource in my LDP instance is known as `http://localhost:3000/r/res1434238382253` (your resource will most likely contain a different identifier). As you might imagine, when this resource is discovered, you need no special awareness of how to interact with it. It's like any other resource: You issue a **GET** request as shown in Listing 2. At this point, you are told about the resource, what it is (**http://www.w3.org/ns/ldp#Resource**), what you are allowed to do to it, and then the more domain-specific representation.
Listing 2. Requesting the new resource

```plaintext
> http get http://localhost:3000/r/res1434238382253
HTTP/1.1 200 OK
Allow: GET,HEAD,DELETE,OPTIONS,PUT
Connection: keep-alive
Content-Type: text/turtle
Date: Sat, 13 Jun 2015 23:54:23 GMT
ETag: W/"2d1e900102c0c8e6d6a5a558c41b1e8"
Link: <http://www.w3.org/ns/ldp#Resource>; rel="type",
   <http://localhost:3000/constraints.html>; rel="http://www.w3.org/ns/ldp#constrainedBy"
Transfer-Encoding: chunked
Vary: Accept
X-Powered-By: Express

<http://localhost:3000/r/res1434238382253> a
   <http://example.org/vocab/bugtracker#Bug>;
   <http://purl.org/dc/terms/title> "Product C crashes when shutting down."
   <http://example.org/vocab/bugtracker#isInState> "New".
```

Conclusion

There's much more to the LDP, but by now, I've shown you enough to give you a sense of what these extra requirements do for the ecosystem. You are free to create whatever kind of resources you want to using arbitrary terms. You can mix common vocabularies such as FOAF and Dublin Core with more domain-specific concepts like issues, requirements, and tests. This tutorial has focused on RDF-based resources (LDP-RSs), but LDPCs can also store non-RDF-based resources (LDP-NRs) such as images, spreadsheets, and documents.

This combination of features enables the intertwingling of an organization, its employees, the software requirements, artifacts, code, tests, and test results. These can all be connected, using standards, by arbitrary tools and clients. In the final tutorial of this series, I'll use and extend these ideas to demonstrate a simple working example of OSLC terms, workflows, and activities in action.

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