XDI

Writing about XDI in this fifth issue of our Personal Data Journal is a special pleasure for me. It was XDI (and its companion technology XRI) which sparked my involvement with the world of user-centric identity and personal data in the first place, and it is still an important focus of the work I am doing today. Therefore, I might not be the most impartial person when it comes to writing about this subject. However, this article will attempt to provide an objective overview of the origin and rationale of these technologies, and of their inner workings.

Both XRI and XDI have been subject to controversial viewpoints over the years. XRI has become infamous as the only standard vote in OASIS history to ever have failed, and it has been criticized as re-inventing the wheel, or as breaking existing web infrastructure. And when mentioning XDI at various conferences, the experience has sometimes been that people have looked at it but do not understand it, or that they do not feel comfortable using it.

Both technologies have undergone significant evolutionary changes over the years. Today, the reality is that with the exploding amount and importance of personal data online, interest about XDI and about concrete projects that use it are on the rise, and this technology might have the potential to provide some of the instruments that will be needed for solving key challenges of the emerging Personal Data Ecosystem.

What are XRI and XDI?

XDI stands for XRI Data Interchange, and XRI in turn stands for eXtensible Resource Identifier. Both technologies are being developed within OASIS Technical Committees (TCs), and they are closely related.

XRI provides a syntax and resolution protocol for abstract, structured identifiers that can be used to reliably point to resources in a way that is independent of any concrete location, and they can be used across different applications and protocols.

XDI is a way of storing data. It is also a way of accessing data, manipulating data, sharing data, linking data and synchronizing data. It is therefore both a data model and a protocol. XDI is based on the RDF graph model as defined by the W3C Semantic Web activity. Within this XDI data model, XRIs are used for identifying resources, just like URIs are used for identifying resources in RDF. One might therefore say that XRIs are to XDI what URIs are to RDF. In a similar manner as URIs, XRIs also offer discovery, which means that given an XRI such as =alice, Alice's XDI server can be looked up. This discovery functionality serves to create a global, distributed network of linked data.

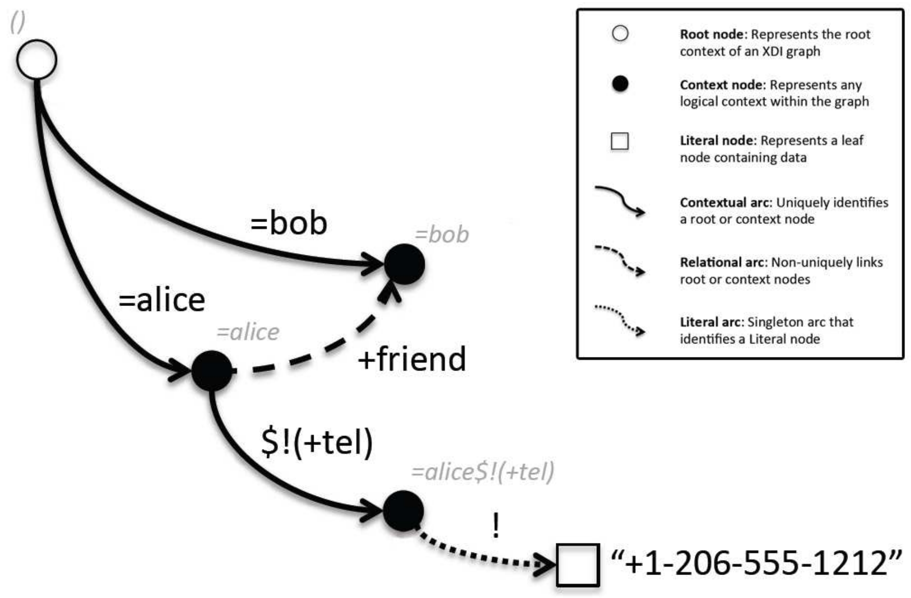
However, unlike URIs, which in RDF are considered opaque (i.e. not serving any functional purpose other than identifying a resource), XRIs have an internal semantic structure. Some of XRI's syntactical features, such as "global context symbols", "sub-segments" and "cross references", affect both the semantic interpretation and the functionality of XDI. For example, if the XRI =alice is used within XDI data, then an XDI processor can immediately tell that it refers to a person (because the = symbol is used). Another example would be the $msg XRI, which an XDI processor will treat as an XDI message that performs operations on XDI data.

A “container” of XDI data is called an XDI graph or XDI document. An XDI graph can be stored in memory, in a relational SQL database, in an LDAP directory, or in any other kind of storage mechanism. XDI storage works well with any kind of key/value based store, such as a Berkeley DB, HTML5’s localStorage, or even a Distributed Hash Table (DHT). XDI data can also be stored in serialized form in a simple text file.

The basic building blocks with which data is modeled in XDI are quite simple and consist only of “contexts”, “relations”, and “literals”. These building blocks form “triples” (also called “statements”), and all of them together constitute the XDI graph. One distinct feature of XDI is that all these building blocks have a unique address that can be used to identify their location within the graph.

Example

Introductions about Internet technologies are often better provided with examples than with long paragraphs of dry technical specifications. Therefore, the following diagram shows a basic XDI sample graph. The information represented by this graph states that Alice’s phone number is +1-206-555-1212, and that Bob is her friend.


()
=bob
=bob
=alice
=alice
+friend
$!(+tel)
=alice$!(+tel)
!
“+1-206-555-1212”
Root node: Represents the root context of an XDI graph
Context node: Represents any logical context within the graph
Literal node: Represents a leaf node containing data
Contextual arc: Uniquely identifies a root or context node
Relational arc: Non-uniquely links root or context nodes
Literal arc: Singleton arc that identifies a Literal node

The same information can also be expressed as a list of five XDI statements, which correspond to the five arcs in the diagram and are explained as follows:

```
()/()/=alice
()/()/=bob
```

Within the XDI graph’s root context, there are the two subcontexts =alice and =bob.

```
=alice/+friend/=bob
```

Within the context =alice, there is a relation of type +friend to the context =bob.
`=alice/()/$!(+tel)` Within the context =alice, there is a subcontext $!(+tel).
`=alice$!(+tel)/!/(data:,+1-206-555-1212)` Within the context =alice$!(+tel), there is a literal whose value is Alice’s phone number.

There is also a convenient JSON serialization of XDI data, for example the five above statements would correspond to the following JSON data:

```
{
"()/()": [
   "=alice",
   "=bob"
],
"=alice/()": [
   "$!(+tel)"
],
"=alice$!(+tel)/!" : [ "+1-206-555-1212" ],
"=alice/+friend" : [ "=bob" ]
}
```

How can you use XDI?

From a high-level perspective, there are several ways to “do XDI”, for example:

- An application can operate simply as an XDI client, which means that it internally stores and handles data in any non-XDI way, but it still uses XDI at various times simply for accessing data from external XDI servers. This requires knowledge of an XDI address that is being accessed, plus credentials to authenticate to the XDI server.
- An application can also adopt XDI as its native data model and store all its data in XDI. This would be a situation similar to using an RDF triple store such as Apache Jena or a graph database such as Neo4j. In fact, it would be quite accurate to state that XDI **is** itself a graph database. The advantage of this approach of working with XDI natively is that the application could immediately exploit advanced XDI features by simply adding an XDI server to its architecture, and then share, link and synchronize its data with other XDI servers.
- Finally, an application can also be architected to use a non-XDI model internally, e.g. a traditional relational model, and then set up an XDI server on top of it that dynamically “maps” its data to XDI. This requires manual implementation of the mapping, which must be maintained whenever the application’s internal data model changes. The advantage of this approach is that it can add XDI compatibility to an already existing application, without introducing a strong dependency on XDI.

XDI Messaging

In any case – whether an application only acts as an XDI client, whether XDI is used as the native data model throughout one's architecture, or whether internal non-XDI data is dynamically mapped to XDI – the way for XDI clients to interact with XDI servers is through XDI messaging. XDI messages are themselves XDI documents, and they contain basic operations such as $get, $add, $mod, $del, $copy or $move to retrieve and manipulate XDI data. XDI messages can be sent over different protocol bindings, such as HTTP, WebSockets, or SMTP.

When an XDI message is received by an XDI server, the operations in the message are executed and applied to the underlying graph, and a result or an error message (both are also XDI documents) is returned. In addition to this basic mechanism, support for more advanced functionality such as asynchronous messaging, signed messages, or long-live transactions is also being envisioned by the OASIS XDI TC. Besides the operations to be performed, an XDI message also contains an XRI identifying the message sender, a timestamp, and a reference to an XDI link contract.

XDI Link Contracts

XDI link contracts are another important and one of the most famed (and sometimes little understood) features of XDI. Link contracts are the basic access control mechanism of XDI, expressing who can perform what operations on an XDI graph. Link contracts live side-by-side with the graph which they protect, i.e. they are part of it. This means that when XDI data is moved from one place to the other, the permissions stay with the data.

When a message is received by an XDI server, link contracts are evaluated to check if the message's operations are authorized or not. In its simplest form, a link contract consists of two components: One or more sender XRIs identifying the parties who are given permissions by the link contract (the "assignees"); and one or more XDI operations and XDI addresses, specifying what operations are allowed on what parts of a graph. Link contracts also specify how the sender of a message is expected to authenticate, e.g. by signing the message, or by providing a password or OpenID Connect token. Some work is being done to pursue the vision of a tight relationship between OpenID Connect and XDI, which means that in a global, distributed ecosystem, OpenID Connect would provide identity federation, and XDI would provide data federation.

XDI within PDEC

Currently, several members of the PDEC Startup Circle are working with XDI technology: Gluu is leading the OpenXDI (OX) implementation, which offers a feature-rich, LDAP-based XDI server, as well as XDI client code in Java, Ruby and JavaScript. OpenXDI also contains components to interface with related technologies such as OpenID Connect and SCIM, and it has contributed to corresponding interoperability testing efforts.

Another XDI Java implementation called XDI² ("XDI Squared") is available as part of Project Danube, which attempts to be lightweight and modular enough to fit a wide range of use cases, such as deployment on embedded devices and in distributed peer-to-peer networks. XDI² is also being used as the technological basis for Connect.me's initial implementation steps of its Respect Network.

Work is also underway to integrate XDI with the Kynetx KRL technology ("Kinetic Rules Language"), which is a language for handling events in a network where everybody and everything is connected (the "Live Web"). The vision behind such integration is that XDI could offer a data abstraction and semantic interoperability layer which events and event handlers can build on, in order to create what could be considered an operating system for the cloud.

XDI Specifications

For the XDI 1.0 suite of specifications, the OASIS XDI TC has opted to pursue an approach of multiple small and modular documents, rather than a monolithic one that contains all aspects of XDI. It is also possible that a distinction will be made between "core specs" (addressing essential parts of XDI with a high priority) and "extension specs". The following is a list of the different documents, as currently expected by the TC:

- XDI 1.0 Graph Model: The normative graph model and addressing syntax.
- XDI 1.0 Serialization: The normative JSON serialization of XDI graphs.
- XDI 1.0 API: The XDI operations on XDI graphs using XDI messages, which can be sent over a number of bindings such as HTTP, Websockets, SMTP.
- XDI 1.0 Dictionary: The semantics to define an XDI dictionary, which defines valid structures of an XDI graph and can be considered the XDI equivalent to the schema of a relational database, or the ontology of an RDF graph.
- XDI 1.0 Link Contracts: The standard structure and vocabulary XDI link contracts.
- XDI 1.0 Discovery: The mechanism to discover an XDI endpoint given an XDI address.
- XDI 1.0 Versioning: The semantics of versioning data in an XDI graph.
- XDI 1.0 Synchronization: The semantics of XDI link contracts that establish synchronization and notification relationships between two or more XDI servers.
- XDI 1.0 Queries: A query language that builds on the XDI 1.0 API.
- XDI 1.0 Asynchronous API: An asynchronous way of sending XDI messages to an XDI server.

Further Resources

The OASIS XDI TC currently offers three introductory documents about the XDI graph model, as well as a wiki covering many advanced topics and current design issues. A public XDI demo server is also available where different XDI implementations can be tried out by XDI users and developers.

Text for a side box:

XDI in one sentence

"XDI allows for multi-participant data sharing and data portability, where every piece of data is addressable, regardless of the underlying network topology, and every participant retains control of the permissions on the data that they share."

Text for a side box:

The Three Generations

XDI has seen three major evolutionary steps since its inception:

1. The ATI model. After the founding of the OASIS XDI TC in 2004 and some early groundwork and development in 2005, this first XDI model was fully developed by 2006. In this model, all data was expressed through statements consisting of authorities, types, instances, links and refs.
2. The RDF model. In early 2007, this new model was developed based on the RDF graph model from the W3C Semantic Web activity. In this model, data consisted of subject/predicate/object triples, whereas subjects and predicates were expressed as XRIs and an object could be either an XRI, a literal, or an inner graph (a concept similar to RDF blank nodes).
3. The XDI 1.0 graph model. This latest version of the XDI graph is explained in detail in the main text of this article. Like its predecessor, it also assumes triples as its foundation, but also includes a strong notions of contextuality. It is expected to become the foundation of the XDI 1.0 specifications.

Text for a side box:

Statements about XDI by OASIS XDI TC members:
Phil Windley, Kynetx: “I think one of the great benefits is location independent data references. As the world moves to more and more APIs and people's personal data is being stored in more and more places, giving programmers the ability to reference data without knowing the details of where and how it's stored is of great importance.”
Michael Schwartz, Gluu: “The XDI standard provides the data structure, messaging and security to enable data federation, enabling autonomous parties to securely exchange information not only about people, but about any entity type.”
Drummond Reed, Connect.Me: “My single favorite XDI feature is link contracts. Few people understand how hard the problem of portable permissions is, i.e., how an authority for a set of data can create a set of permissions on that data that can be understood by anyone else, even when the data is moved from one service provider to another. The XDI globally addressable graph model for data is what makes link contracts possible, and it is the combination of global addressability and controllability that will finally bring us true data portability.”
Cameron Hunt, bitWorld: “The ability to cleanly assert and reference metadata statements within XDI is one of the most powerful features for sophisticated data representation and processing.”