The Highs and Lows of Integrating LDAP with XML

Steven Legg

eB2Bcom

steven.legg@eb2bcom.com

Introduction (1)

- There is a strong tendency for newer application protocols and their data payloads to be defined in an XML schema language for rendition as XML documents
- Such applications typically have a need for the kind of user and other object data that has traditionally been held in the entries of an LDAP directory service, as well as defining new kinds of structured data to be associated with those same objects

Introduction (2)

- There is a clear advantage to data consistency and ease of administration in having a common directory service for all of an enterprise's applications, whether or not they are based on XML
- Unfortunately, the core LDAP specifications have no inherent support for XML-formatted object data or XML-formatted protocol messages
- Four general approaches to supporting XML within a directory service are identified, compared and contrasted

Approaches (1)

- Ad-hoc solutions using the existing LDAP framework
  - XML documents embedded in directory attributes
  - With either the Octet String or Directory String syntax
  - Specific-purpose syntaxes and matching rules
- LDAP-inspired XML-based directory protocols
  - Directory Services Markup Language v2.0 (DSMLv2)
  - DSMLv2 Profile of the Service Provisioning Markup Language Version 2 (SPMLv2)

Approaches (2)

- XML-based registry or discovery services
  - Universal Description, Discovery and Integration (UDDI)
  - ebXML Registry
  - XSD Profile of SPMLv2
- The XML-Enabled Directory (XED)
  - A collection of extensions to ASN.1, LDAP and X.500
    - Robust XML Encoding Rules (RXER) for ASN.1 (RFC 4910)
  - XML Lightweight Directory Access Protocol (XLDAP)
    - LDAP with RXER substituted for BER

Single Transfer Syntax

- A single transfer syntax is preferred for a protocol message and its data payload
- All the protocols except LDAP use XML for the protocol messages, but not necessarily for the data payloads
- The ebXML Registry, UDDI and the XSD Profile of SPMLv2 use a single transfer syntax

Transfer Syntaxes - DSMLv2

- Example - a value of the objectClasses directory attribute in DSMLv2

```
<dsml:value>( 2.5.6.6 NAME 'person' SUP top
STRUCTURAL MUST ( sn $ cn ) MAY ( userPasswo
rd $ telephoneNumber $ seeAlso $ description
 ) )</dsml:value>
```

- An XML processor and a parser for LDAP-specific encodings is required
  - Likewise for the DSMLv2 Profile of SPMLv2
- A BER encoder and decoder is required
  - To process LDAP controls and extended operations

Transfer Syntaxes - XLDAP (1)

- X.500 protocols use a single transfer syntax (i.e., BER)
- Not so for LDAP, where directory attribute values are conveyed as the content of ASN.1 OCTET STRINGs
- XLDAP replaces LDAP's OCTET STRING containers with the open types used by X.500
- An XLDAP message and the contained attribute values are one continuous RXER encoding

Transfer Syntaxes - XLDAP (2)

- Example - a value of the objectClasses directory attribute in XLDAP

```
<value>
 <identifier>2.5.6.6</identifier>
 <name><item>person</item></name>
 <information>
  <kind>structural</kind>
  <mandatories>
   <item>2.5.4.4</item><item>2.5.4.3</item>
  </mandatories>
  <optionals>
   <item>2.5.4.35</item><item>2.5.4.20</item>
   <item>2.5.4.34</item><item>2.5.4.13</item>
  </optionals>
 </information>
</value>
```

Continuous Encodings

- A single pass of an XML processor is preferred for parsing a protocol operation and its element content data payloads
- The ebXML Registry, UDDI and the XSD Profile of SPMLv2 have a continuous encoding
- With regard to the other protocols, suppose we want to store an XML-formatted record in a new directory attribute:

```
<product>
 <code>XJ459</code>
 <quantity>137</quantity>
</product>
```

Element Content Payloads in DSMLv2 (1)

- Using a directory attribute with the Directory String syntax:

```
<dsml:value>
 &lt;product>
  &lt;code>XJ459&lt;/code>
  &lt;quantity>137&lt;/quantity>
 &lt;/product>
</dsml:value>
```

- Likewise for the DSMLv2 Profile of SPMLv2

Element Content Payloads in DSMLv2 (2)

- Using a directory attribute with the Octet String syntax:

```
<dsml:value xsi:type="xsd:base64Binary">PHB
yb2R1Y3Q+DQogPGNvZGU+WEo0NTk8L2NvZGU+DQogPH
F1YW50aXR5PjEzNzwvcXVhbnRpdHk+DQo8L3Byb2R1Y
3Q+DQo=</dsml:value>
```

- Likewise for the DSMLv2 Profile of SPMLv2
- Obtaining the component parts of the inventory record requires a second invocation of an XML processor

Element Content Payloads in XLDAP (1)

- XED makes it possible to define a directory attribute to hold the XML-formatted record in its natural form:

```
<value>
 <product>
  <code>XJ459</code>
  <quantity>137</quantity>
 </product>
</value>
```

- XLDAP has a continuous encoding

Element Content Payloads in XLDAP (2)

- XED defines a special ASN.1 type, called the Markup type, whose abstract values can hold the content and XML attributes of an XML element
  - Analogous to the XML Schema anyType
  - The content and XML attributes are embedded in instances of the ASN.1 UTF8String type
- An RXER encoder knows to output the content of Markup values in-line
- Directory attributes with the Markup type as their syntax use RXER as their LDAP-specific encoding

Scoping Issues

- The octets for a nested abstract value in a BER encoding are themselves a well-defined and complete BER encoding
- It is not generally true of an XML document that a nested element is a well defined and complete XML document
  - Qualified names may depend on namespace declarations in an enclosing element
  - Entity names may depend on the DTD of the enclosing XML document

Namespace Accumulation Problem (1)

- Consider this short XML document, representing an operation of a simple, fictitious protocol:

```
<p:addObjectRequest
   xmlns:p="http://protocol.example.com"
   xmlns:u="http://user.example.com">
 <p:object class="1">u:data</p:object>
</p:addObjectRequest>
```

  - If `<p:object>` is extracted, then it is necessary to retain the namespace declarations from `<p:addObjectRequest>` that define namespaces used by qualified names within `<p:object>`
  - Assume retained namespace declarations are copied to the start tag of the extracted element

Namespace Accumulation Problem (2)

- It is always possible to recognize qualified names as the names of elements and attributes
- Although the character data content of `<p:object>` looks like a qualified name, this could be a coincidence
- It is necessary to know the data type of an element to decide what really is a qualified name in the character data

Namespace Accumulation Problem (3)

- The data type definition may not be enough
- Elements and XML attributes can contain formatted character strings, such as XPath expressions, that contain qualified names
  - The data type for such formatted character strings is often defined to be the XML Schema string data type
  - The safe, quick and simple strategy is to just assume that all the in-scope namespaces are significant

Namespace Accumulation Problem (4)

- Suppose that the server holding the extracted `<p:object>` responds to a request for `<p:object>`:

```
<q:fetchObjectResponse
   xmlns:q="http://protocol.example.com">
 <p:object
    xmlns:p="http://protocol.example.com"
    xmlns:u="http://user.example.com"
    class="1">u:data</p:object>
</q:fetchObjectResponse>
```

Namespace Accumulation Problem (5)

- If the receiver extracts `<p:object>` and simply retains all the in-scope namespace declarations then the extracted object becomes:

```
<p:object
     xmlns:p="http://protocol.example.com"
     xmlns:u="http://user.example.com"
     xmlns:q="http://protocol.example.com">
     class="1">u:data</p:object>
```

- The receiver's version of `<p:object>` has an additional namespace declaration compared to the sender's version of `<p:object>`

Namespace Accumulation Problem (6)

- LDAP, DSMLv2 and the DSMLv2 Profile of SPMLv2 are immune to namespace accumulation
  - An XML document in a directory attribute value is parsed in isolation
- Namespace accumulation is a critical problem if the payload is subject to a digital signature

XML Canonicalization (1)

- Digital signatures are usually computed over a canonical representation
- Three well-known canonicalizations for XML are:
  - Canonical XML
  - Exclusive XML Canonicalization
  - Schema Centric Canonicalization
- Adding any namespace declaration change the Canonical XML representation and break any digital signature computed over it

XML Canonicalization (2)

- The Exclusive XML Canonicalization form is not changed by adding namespace declarations
  - Must augment the original payload with a list of the significant namespace prefixes
  - Support for Exclusive XML Canonicalization has to be built into the data type definition for the payload
  - Only the creator of the payload knows for certain which namespace prefixes are significant.

XML Canonicalization (3)

- Schema Centric Canonicalization uses knowledge of the data types to normalize the content and XML attributes of elements
- Namespace declarations are reconstructed so unnecessary namespace declarations will be eliminated
- Data type definitions must be annotated to indicate the presence of embedded languages such as XPath expressions
  - Schema Centric Canonicalization is essentially open-ended

Namespace Accumulation and XED (1)

- XED takes a simpler approach that is compatible with the three canonicalization schemes for XML and limits the accumulation of unnecessary namespace declarations regardless of whether canonicalization is a consideration
- XED requires elements where the data type is the ASN.1 Markup type to be be self-contained with respect to namespace declarations
- Any element with the Markup type as its ASN.1 type can be freely extracted and composed into new protocol operations without revising its namespace declarations

Namespace Accumulation and XED (2)

- Any element where the actual data type is not an ASN.1 type is represented in ASN.1 using the Markup type
  - With a reference to the non-ASN.1 data type definition, if one exists
- Character strings that contain an embedded language are represented in ASN.1 using the Markup type

Namespace Accumulation and XED (3)

- Directory attribute values that have an ASN.1 type other than the Markup type as their syntax would normally be decoded into abstract values
  - Namespace declarations are not preserved beyond the decoding step
  - Namespace declarations are added as required when these abstract values are re-encoded
  - Decoding and encoding will eliminate unnecessary namespace declarations
  - There is no need to annotate data type definitions with information about embedded languages

Namespace Accumulation and XED (4)

- A directory client or server might not know the syntax for a directory attribute value it receives
  - So won't know whether the value is supposed to be self-contained
- RXER provides a special XML attribute, called the context attribute, to address this situation
  - It contains a list of the namespace prefixes of all the namespace declarations added to an element
  - If the data type for a received element is known to be the Markup type, then the context attribute can be used to strip off the added namespace declarations
  - It does not explicitly appear in ASN.1 type definitions

Support for Canonicalization (1)

- XLDAP is compatible with payloads that depend on Canonical XML, Exclusive XML Canonicalization or Schema Centric Canonicalization
- LDAP, DSMLv2 and the DSMLv2 Profile of SPMLv2 are compatible with payloads that depend on Canonical XML, Exclusive XML Canonicalization or Schema Centric Canonicalization
  - They are immune to namespace accumulation

Support for Canonicalization (2)

- UDDI implementations are not guaranteed to support payloads that depend on Canonical XML or Exclusive XML Canonicalization
- The ebXML Registry and the XSD Profile of SPMLv2 cannot support payloads that depend on Canonical XML
  - They are susceptible to namespace accumulation

Matching Capabilities

- It is desirable to be able to search XML-formatted data for instances matching specified criteria, taking into account the underlying data type
- For example, if some element notionally contains a boolean value, then users would like to evaluate whether the element's value is true or false

Matching XML in LDAP (1)

- Here are some ways to represent the "true" value for the XML Schema boolean type:

```
<value>true<\value>
<value>1<\value>
<value> tr&#x75;e <\value>
<value>tr<!-- a comment -->ue<\value>
<value>
 <![CDATA[tr]]>ue
<\value>
```

- The existing string matching rules of LDAP are inadequate for matching of XML-formatted data
- DSMLv2 and the DSMLv2 Profile of SPMLv2 depend on LDAP matching rules

Matching XML in LDAP (2)

- Preserving compatibility with Canonical XML and Exclusive XML Canonicalization limits opportunities to normalize XML-formatted data before it is stored
  - Each of the example boolean values produces a different Canonical XML form:

```
<value>true<\value>
<value>1<\value>
<value> true <\value>
<value>tr<!-- a comment -->ue<\value>
<value>
 true
<\value>
```

Type-Aware Matching (1)

- The ideal solution is to have matching rules that are aware of the underlying data type of the XML being matched
  - Could be achieved in LDAP on a case-by-case basis by defining new syntaxes and matching rules
- The component matching rules for LDAP already provide a general capability to match arbitrary parts of structured data
  - Component matching only applies to data with an underlying ASN.1 type

Type-Aware Matching (2)

- The XED framework extends the component matching rules to apply to structured data described by an XML Schema or DTD
  - A subset of the XPath abbreviated syntax is used to identify components
  - Thus, XLDAP is able to perform data-type aware matching of element content in directory attribute values
- The XSD Profile of SPMLv2, UDDI and the ebXML Registry provide data-type aware matching of element content in objects
  - With varying degrees of expressiveness

Query Generality (1)

- LDAP search filters, in conjunction with the component matching rules, provide a general-purpose query mechanism
  - This mechanism is inherited by DSMLv2, the DSMLv2 Profile of SPMLv2 and XLDAP
- The XSD Profile of SPMLv2 is similarly general-purpose in its use of XPath expressions to filter objects
  - XPath expressions are generally more expressive than component matching
  - Some of the capabilities of component matching have no equivalent in XPath

Query Generality (2)

- The filter operations of UDDI and the filter query syntax of the ebXML Registry have rigidly defined filter structures
  - Supporting new object classes or additions to existing object classes requires the implementation of protocol extensions
- The optionally-supported ebXML Registry SQL query syntax avoids the limitations of the filter query syntax

Joins Between Objects (1)

- LDAP has no support for expressing a join between objects (entries)
  - Likewise for DSMLv2, the DSMLv2 Profile of SPMLv2 and XLDAP
- UDDI only has some predefined join criteria for relationships between business entity objects

Joins Between Objects (2)

- The ebXML Registry filter query syntax has a wider range of predefined join criteria for relationships between various classes of object
- The XSD Profile of SPMLv2 has a limited, general-purpose join capability through XPath expressions
- The ebXML Registry SQL query syntax provides an unrestricted join capability

Object Extensibility (1)

- LDAP administrators can easily extend the directory schema with new object classes and directory attributes
  - DSMLv2, the DSMLv2 Profile of SPMLv2 and XLDAP inherit this capability
  - XED additionally provides the means for administrators to create new directory attribute syntaxes

Object Extensibility (2)

- UDDI search and update protocol operations are tied to specific object classes
  - Adding a new class of object requires implementation in the server of new protocol operations to find and manipulate objects of the class
- The ebXML Registry filter query syntax has filter expressions that are specific to each object class
  - Adding a new object class requires implementation of new filter expression evaluation routines in the server

Object Extensibility (3)

- Objects in the ebXML Registry have user-defined properties called slots
  - Slots are limited to just character strings
- The XSD Profile of SPMLv2 is at least compatible with administrators providing definitions of new classes of objects
  - A mechanism to provide new definitions is not specified

Summary of Capabilities

Feature	Protocol						
	LDAP	DSMLv2	SPMLv2		UDDI	ebXML Registry	XLDAP
			DSMLv2 Profile	XSD Profile			
Single transfer syntax	no	no	no	yes	yes	yes	yes
Continuous encoding	no	no	no	yes	yes	yes	yes
Supports Canonical XML in payload	yes	yes	yes	no	no	no	yes
Supports Exclusive XML Canonicalization in payload	yes	yes	yes	yes	no	yes	yes
Supports Schema Centric Canonicalization in payload	yes	yes	yes	yes	yes	yes	yes
Supports unparsed entities	yes	yes	yes	no	no	no	no
Data-type aware XML matching	no	no	no	yes	yes	yes	yes
Flexible query expressions	yes	yes	yes	yes	no	maybe	yes
Joins between objects	no	no	no	partial	minimal	yes	no
Allows user-defined object classes	yes	yes	yes	maybe	no	no	yes
Allows user-defined properties (attributes)	yes	yes	yes	maybe	no	yes	yes
Allows user-defined property data types (syntaxes)	no	no	no	maybe	no	no	yes

Summary (1)

- LDAP, DSMLv2 and the DSMLv2 Profile of SPMLv2 are identical in terms of the characteristics described in the table
- XLDAP has the single transfer syntax and continuous encoding that are typical of protocols designed around XML
  - Without losing compatibility with Canonical XML

Summary (2)

- The XPath expressions of the XSD Profile of SPMLv2 and the SQL query syntax of the ebXML Registry are more expressive than search requests in XLDAP
  - There is no impediment to extending the capabilities of XLDAP in this area

Summary (3)

- LDAP has particular strengths in terms of supporting user-defined schema extensions
  - DSMLv2, the DSMLv2 Profile of SPMLv2 and XLDAP inherit this capability
  - The XED framework extends it further to directory attribute syntaxes
  - The XSD Profile of SPMLv2 has the *potential* to support user-defined schema extensions as readily as XLDAP
- The schema in UDDI and the ebXML Registry cannot be extended in practice without involving the server vendor

Conclusion

- XLDAP realizes the twenty year old directory model in a way that compares favourably with much newer XML-based protocols