Mapping OASIS KMIP to an XML WSDL

Matt Ball, Sun Microsystems

February 2, 2010

Overview

This document proposes a method to map OASIS KMIP (Key Management Interoperability Protocol) onto an XML WSDL (Web Services Description Language) schema. This document assumes that the reader is familiar with the OASIS KMIP specification, XML WSDL, XML XSD, and XML SOAP.

The general strategy is to use XML SOAP to map KMIP onto a WSDL with Document/Literal encoding so that it is possible to validate the WSDL against the WS-I Basic Profile 1.0.

XML benefits vs. binary encoding

Here is a list of some of the benefits to using an XML encoding over the standard binary encoding used by KMIP:

- Wide availability of parsers. There are many well-tested and well-used XML parser generators available for many programming languages. For example, the gSOAP XML parser generator can create C/C++ code. This is in contrast to the KMIP binary encoding, which (as of this writing) has no available tools to help create a parser
- Easier to read and debug. XML is human-readable. Conversely, the KMIP binary encoding is difficult to read (without the aid of an interpreter).
- Faster to develop. Because of the availability of XML parser generators and because the output is easier to read, it's possible to develop an XML application much faster.
- More easily extended. XML is self-describing and can be easily extended by layering on additional namespaces.
- Hardware acceleration currently available for speeding up XML parsing.

Binary encoding benefits vs. XML

To be fair, here is the main benefit of the KMIP binary encoding:

- Relatively compact binary format that can be parsed with a smaller parser than can generally be produced for XML parsing. This is advantageous in clients with very limited code space and processing capabilities.

Proposed WSDL

The file

Data type mapping

Primitive data types

Table 1 shows a proposed mapping of primitive OASIS KMIP data types to their corresponding XSD types and proposed C++ programming language types.

Table 1: KMIP Primitive Type Mapping to XSD

KMIP Primitive Type	XML XSD type	Proposed C++ type
Integer	xsd:int	int
Long Integer	xsd:long	long long int
Big Integer	xsd:base64Binary	struct { unsigned char *ptr; int size; }
Enumeration	xsd:string / xsd:enumeration	enum { ... }
Boolean	xsd:boolean	bool (or char)
Text String	xsd:string	wchar_t * (or char *)
Byte String	xsd:base64Binary	struct { unsigned char *ptr; int size; }
Date-Time	xsd:dateTime	time_t (assuming a 64-bit system)
Interval	xsd:duration	char * or long long int

Complex data types

In addition to primitive data types, KMIP also supports complex data types, both explicit and implicit. Table 2 shows the mapping of KMIP complex types (both explicit and implicit) onto XML and C++.

Table 2: KMIP Complex Type Mapping to XML

KMIP Complex Type	XML WSDL	Proposed C++ Format
Structure	<complexType><sequence>	struct { ... }
(implied) Array	<complexType name="ArrayOf..."> <sequence> <element ... maxOccurs="unbounded">	struct { struct {} *ptr; int size }
(implied) Union	<complexType><choice>	union { ... }

The KMIP specification implies the Array type when allowing an element to appear more than once. Encoding this in XML typically requires creating an implied ComplexType to encapsulate the array.

The KMIP specification implies a Union type when it states that the encoding "Varies". For example, the KMIP Attribute Value (see KMIP 2.1.1) has an encoding that 'Varies'. This can be handled in XML using a 'choice', or in C using a 'union' with an index that enumerates which encoding was chosen.

Naming strategy

Namespace

The proposed XML namespace is 'KMIP'.

Element naming

The general strategy for creating XML names from the KMIP specification is to directly use the names as-is with the spaces removed. Other characters not allowed in XML (for example ':') are either removed or replaced with a hyphen ('-').

For (implied) Arrays, the WSDL uses the convention of prefixing the type with "ArrayOf...", and this naming convention appends the word "List" to the end of the name. Example:

WSDL:

```
  <complexType name="ArrayOfUniqueIdentifier">
   <sequence>
    <element name="item" type="KMIP:UniqueIdentifier" minOccurs="0" maxOccurs="unbounded"
nillable="true"/>
   </sequence>
  </complexType>

  <element name="DeriveKey">
   <complexType>
    <sequence>
     <element name="ObjectType" type="KMIP:ObjectType" minOccurs="1" maxOccurs="1"/>
     <element name="UniqueIdentifierList" type="KMIP:ArrayOfUniqueIdentifier" minOccurs="1"
maxOccurs="1" nillable="true"/>
     <element name="DerivationMethod" type="KMIP:DerivationMethod" minOccurs="1" maxOccurs="1"/>
     <element name="DerivationParameters" type="KMIP:DerivationParameters" minOccurs="1"
maxOccurs="1"/>
     <element name="TemplateAttribute" type="KMIP:TemplateAttribute" minOccurs="1"
maxOccurs="1"/>
    </sequence>
   </complexType>
  </element>
```

Encoding Differences

When using XML/SOAP, there is a functional model that is different than that of KMIP. In particular, SOAP has a basic method for encoding procedures and the corresponding responses that differs from the KMIP method of wrapping each command and response with a message envelope. While it is possible to emulate this using SOAP, it forces everything to be interpreted as a single command and removes the benefits that XML parser generators provide in performing parameter type-checking.

In practice, this will serve to simplify things on both the server and client side.

Caveats

The proposed XML encoding has the following caveats as compared to the KMIP binary format:

- No standard KMIP header (this is replaced by a SOAP header that has slightly different features)
- Batching not supported
- Asynchronous commands not supported