Verification of Data Identifiers via Web Services

Alberto Accomazzi

NASA Astrophysics Data System
Harvard-Smithsonian Center for Astrophysics
aaccomazzi@cfa.harvard.edu

What are Data Identifiers?

- Strings associated with a particular data set
- Format is *instrumentID:datasetID*, where *instrumentID* and *datasetID* can have their own hierarchical structure
- Examples:
  - Sa/ASCA:X/86008020
  - Sa/ROSAT:X/701576n00
  - Sa/RXTE:50184-03

Why Dataset Verification?

- We want to be able to...
  - *identify* a dataset in a unique way, so we can create a permanent name for it
  - *refer to* a dataset in the published literature, so other researchers can access it
  - *locate* the dataset, irrespective of the number of the different data centers that archive it

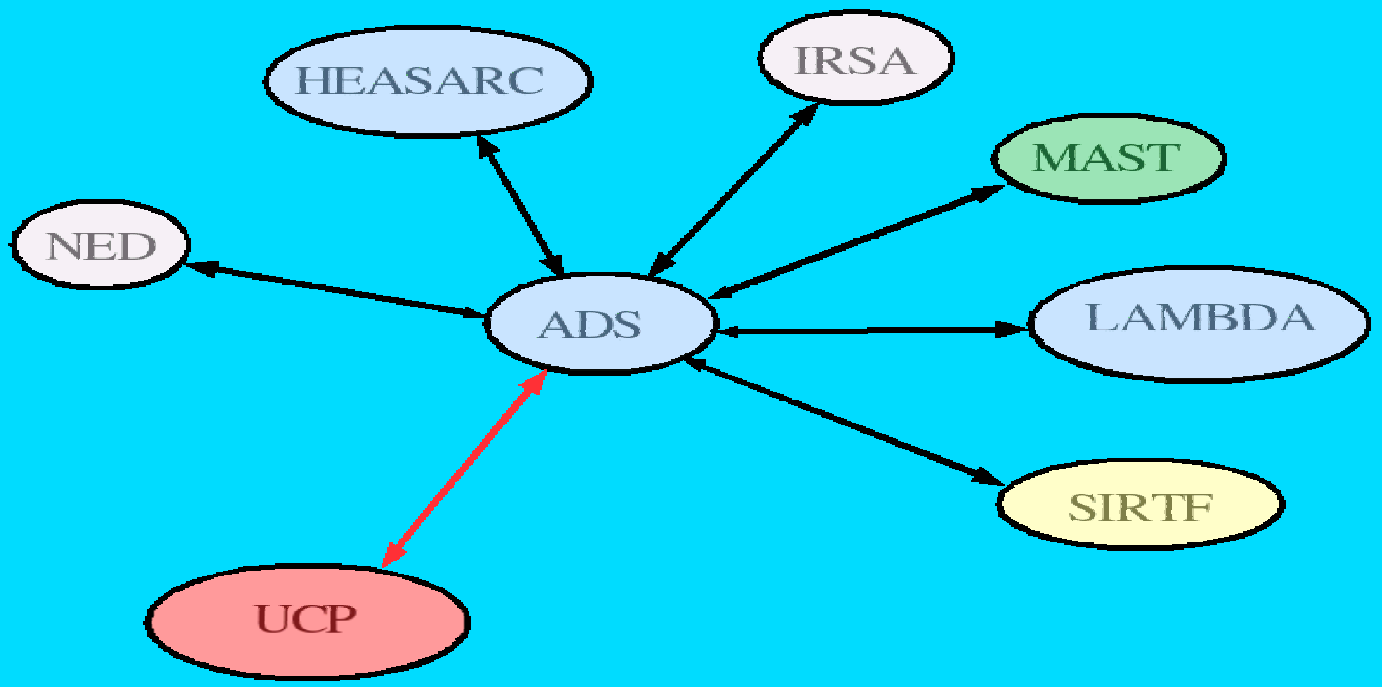
Current Prototype Requirements

- Simple to use from the User's and Publisher's point of view
- Get the job done now
- Provide an upgrade path towards “VO compliance” in the future

Why ADS?

- ADS is involved in the ADEC ITWG (Interoperability Technical Working Group)
- ADS has a long-standing relationship with publishers and data archives
- ADS has implemented several custom-designed harvesting and validation services over the years (e.g. Data linking and bibcode verification)
- ADS is testing the deployment of web services within the NASA Astrophysics Data Centers and this is a good test case

Architecture



Protocols:

SOAP

TBD

Toolkits:

SOAP::Lite (PERL)

NuSOAP (PHP)

BEA Weblogic (java)

TBD

The Perl toolkit: server

- Easy server set up via CGI deployment
- Serves its own WSDL via the ***uri?WSDL*** syntax
- Simplifies deployment by requiring data provider to code just one function that verifies the validity of the identifier

```
### sample CGI SOAP server:
 use ITWG::DataVerifier;
 my $server = ITWG::DataVerifier::Server::CGI->new->handle;

 # example of verification routine;  $res is
 # -1 for illegal syntax, 0 for unrecognized id, 1 for valid id
 sub verify_id {
    my $id = shift;
    my $res = 1;
    my $url = ($res > 0) ? 'http://foo.org/bar?' . $id : '';
    return ($res,$url);
 }
```

The PERL toolkit: Client

- Client is initialized either via WSDL or proxy endpoint
- Serialization and encoding are hidden from user
- Result of SOAP call is an array of hashes that can readily be used

```
## sample client setting up connection via WSDL description
use ITWG::DataVerifier;
my $wsdl = 'http://ads.harvard.edu/ws/ITWG DataVerifier test?WSDL';
my $client = ITWG::DataVerifier::Client >new(service => $wsdl);

my @ids = $client >verify('foo', 'bar')
   or die "could not verify data ids";
my $id  = $ids[0]->{input};  # this should be 'foo'
my $res = $ids[0]->{result}; # this should be 1
my $url = $ids[0]->{url};    # URL that can be used for linking
```

SOAP Interoperability Issues

- Serialization of data structures
  - Conventions used to encode complex datatypes are not consistent across toolkits
  - Overriding serialization rules difficult if not impossible
- SOAP vs. literal encoding
  - SOAP (“section 5”) encoding evolved as a need to serialize RPC calls and supports multi-ref accessors
  - Literal encoding used for “document-style” services, in conjuction with XML schema validation

Data Serialization

```
Use SOAP::Lite;
my $s = SOAP::Serializer->new;
print $s->serialize({ identifiers => [ 'bar', 'baz' ] });

<?xml version="1.0" encoding="UTF-8"?>
<c-gensym1 xsi:type="namesp1:SOAPStruct">
  <identifiers xsi:type="SOAP-ENC:Array"
      SOAP-ENC:arrayType="xsd:string[2]">
    <item xsi:type="xsd:string">bar</item>
    <item xsi:type="xsd:string">baz</item>
  </identifiers>
</c-gensym1>


print $s->serialize(SOAP::Data->name('identifiers')->value([
   map { SOAP::Data->name('identifier')->value($_) } ('bar','baz')]));

<?xml version="1.0" encoding="UTF-8"?>
<identifiers xsi:type="SOAP-ENC:Array"
    SOAP-ENC:arrayType="xsd:string[2]">
  <identifier xsi:type="xsd:string">bar</identifier>
  <identifier xsi:type="xsd:string">baz</identifier>
</identifiers>
```

SOAP vs. Literal Encoding

```
<SOAP-ENV:Body>
  <namesp1:verify xmlns:namesp1="http://ads.harvard.edu/DataVerifier">
    <verifyRequest>
      <header xsi:type="namesp2:SOAPStruct">
        <protocolversion xsi:type="xsd:float">0.2</protocolversion>
      </header>
      <identifiers xsi:type="SOAP-ENC:Array" SOAP-ENC:arrayType="xsd:string[2]">
        <identifier xsi:type="xsd:string">foo</identifier>
        <identifier xsi:type="xsd:string">bar</identifier>
      </identifiers>
    </verifyRequest>
  </namesp1:verify>
</SOAP-ENV:Body>

<SOAP-ENV:Body>
  <verify xmlns="http://ads.harvard.edu/DataVerifier">
    <verifyRequest>
      <header>
        <protocolversion>0.2</protocolversion>
      </header>
      <identifiers>
        <identifier>foo</identifier>
        <identifier>bar</identifier>
      </identifiers>
    </verifyRequest>
  </verify>
</SOAP-ENV:Body>
```

SOAP encoding

- Fits well in the “RPC-style” paradigm
- Is the easiest to implement for many toolkits (serialization is taken care of by software package)
- Works best with homogeneous clients and servers
- Doesn't scale up well for large messages

Literal encoding

- Fits well in the “document-style” paradigm
- Gives developers the greatest freedom in serializing/deserializing the SOAP envelope
- Integrates well with XML schema for validation
- Is becoming the “default” encoding for many SOAP toolkits, including .NET
- Requires more work by the developer

Discussion

- Encoding and serialization
  - Is SOAP encoding on the way out?
  - Can a service provide support for multiple encodings?
  - Should we use SOAP MIME attachments instead?
- WSDL
  - Often needs “tweaking” to be understood by toolkit
  - Level of support for WSDL features often not clear
- The future
  - WS-I consortium to the rescue?
  - When will WS 1.0 be available?