SWAD-Europe deliverable 4.2: Semantic Web and Web Services: RDF/XML and SOAP for Web Data Encoding

Project name:
W3C Semantic Web Advanced Development for Europe (SWAD-Europe)
Project Number:
IST-2001-34732
Workpackage name:
4. Semantic Web and Web Services
Workpackage description:
☞http://www.w3.org/2001/sw/Europe/plan/workpackages/live/esw-wp-4.html
Deliverable title:
SWAd-Europe deliverable 4.2: Semantic Web and Web Services: RDF/XML and SOAP for Web Data Encoding
URI:
☞http://www.w3.org/2001/sw/Europe/reports/sw_soap_design_report/
Authors:
☞Nikki Rogers
Abstract:
This document is in support of the work conducted for SWAD-Europe deliverable 4.2. It is a Public design document describing the architecture of a working SOAP-based "Semantic Web Service". For example we aim to show the use of W3C's XML Protocol and Web Service specifications to describe, discover and communicate with remote RDF/Semantic Web query or inference services. This design document is illustrated throughout with working examples.

Contents

Introduction

In order to explore object marshalling and unmarshalling possibilities in the context of distributed semantic web services in which various RDF-based and non-RDF SOAP solutions may proliferate, an example web services architecture is shown in this report.

The architecture aims to demonstrate how an RDF-based web service (in this case we happen to use Sesame for the RDF service 'backend') might be combined with SOAP (in this case we happen to use Apache Axis for the RDF service's SOAP 'frontend') in order to serialise object data. In the examples, the service's WSDL is used in order to generate client-side stubs so that we may demonstrate the ease with which some client on the network (whether it be "RDF-savvy" or otherwise) may consume such a service.

We also show how soap-encoded data may be transformed (via an XSL stylesheet) to RDF data and suggest scenarios where this may be useful.

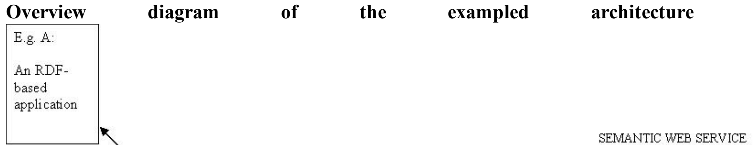
The example dataset we use is very small and chosen merely to demonstrate how object data may be serialised as SOAP encoded data, for exchange on the network. We example how SOAP encoded data may be transformed 'back' to RDF data (the "round trip") in order to show the relationship between the two data models. However, we do not attempt a full comparison of the data models: we do not provide a transformation from RDF to SOAP for example, hence we do not consider the serialisation of arbitrary RDF data via the soap data encoding syntax. Instead we aim to demonstrate some potential for using the SOAP data encoding for standardised data exchange in a network environment in which both semantic web services and more traditional web services reside.

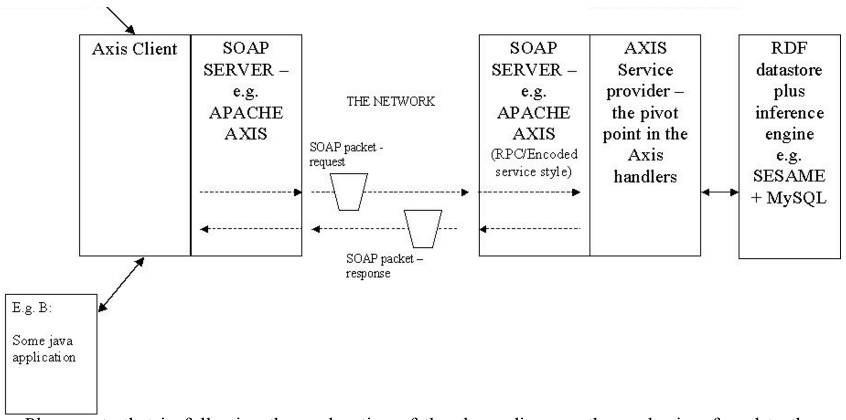
We note that at the time of writing this report, Document/Literal message styled SOAP Web Services have gained industry support over RPC/Encoded services (or any other combination). We give a brief outline of the impact of this ☞document/literal versus RPC/Encoded debate for SOAP services in Section 1.

In ☞Section 1 of this report we give an overview of our chosen hypothetical illustration of a semantic web service scenario. We present a ☞diagram of this example web service architecture. Then, in ☞section 2 we explain our example architecture for a Semantic Web Service at a more technical level. In ☞section 3 we explain how we have opted for the most generic approach with which to integrate Apache Axis and Sesame software, in order to run trials. We give our example data and configurations used and and we hope to make the given sample implementation downloadable via a link on this page in the near future. In ☞section 4 we briefly discuss the Soap data model and the RDF data model - transforming from the former to the latter.

Section 1: Overview of an example web service architecture

Overview diagram of the exampled architecture -




Axis Client
SOAP SERVER – e.g. APACHE AXIS
THE NETWORK
SOAP packet - request
SOAP packet – response
SOAP SERVER – e.g. APACHE AXIS (RPC/Encoded service style)
AXIS Service provider – the pivot point in the Axis handlers
RDF datastore plus inference engine e.g. SESAME + MySQL
E.g. B:
Some java application

Please note that in following the explanation of the above diagram, the reader is referred to the ☞Web Services Architecture, W3C Working Draft (August 2003) for certain equivalent concepts - for example the client side view we discuss here corresponds to the Requestor Agent in that report, the Server side view we discuss corresponds to the Provider Agent.

We have adopted Apache Axis as the software for both the client and server sides. Axis is Java software, designed around handler chains which are configurable and allow the developer to write any handlers needed in order to process the incoming and outgoing SOAP messages. A special sort of Axis handler is a "provider" - a pivot handler representing the handling of data at the point at which Axis actually interacts with the service itself. In the example service described below we use the Java:RPC provider that is built in to Axis. This provider aids the marshalling and unmarshalling of objects between Java and the SOAP encoded data model. So long as it is possible/required to derive object-oriented data from RDF graphs, this standard Axis Java:RPC provider may be used 'out of the box', meaning that no custom handler has to be written.

The Axis handlers on the client side (left hand side in above diagram) can be thought of as the mirror image of the ones on the server side. The client side initiates RPC transactions in order to consume some web service on the network - the service is represented on the right hand side in the above diagram. The service itself is a queryable data store - where the data is stored as RDF. We happen to have used Sesame as the RDF database. The fact that the service uses an RDF database is 'invisible' in terms of its exposure on the network - it is fronted by a SOAP server which can handle the marshalling of objects via a simple Axis<->Sesame bridging class (which we discuss further on).

Data returned by the SOAP server is encoded according to the Soap Encoding specification (see ☞http://www.w3.org/TR/soap12-part2/#soapenc). On the client side, it is very simple to configure the built-in Axis handlers to unmarshal soap-encoded objects returned from the service in response to RPC requests.

What happens to these objects next is dependent on what the client side application requires them for. For example, as Axis can unmarshal complex objects (it generates javabeans from a service's WSDL and uses these to map them to java objects), they can immediately be used within some java application for example (which is the case for 'e.g. B' in the diagram above). Where there are non-java API's on either the client or server sides, software other than Apache Axis is required, but the principles remain the same in terms of marshalling and unmarshalling objects.

In a scenario where perhaps the client side is being used as part of some aggregator (semantic) web service, it is possible to simply transform the soap encoded data to RDF (via XSLT) in order to add it to an RDF datastore, say ('e.g. A' in the diagram above). ☞Scenarios where this can be useful are discussed further below.

We note that the scenarios consider here are within the context of stateless connections only.

Relevant Scenarios - We give here an example high-level scenario to illustrate the potential of using the SOAP data encoding syntax for a web services architecture as outlined above.

Suppose I wish to purchase a house in the South West of the UK. I may be interested in various data sets: houses for sale in the area (obviously), according to price band and location, plus environmental information (such as pollution factors, tendency for landslides, flooding and so on), plus also schools information on which I can base decisions regarding my children's education (government Offsted reports, or schools' own web sites for example). It currently tends to be the case that no single online service will provide such information. So these data repositories exist in the closed worlds sense.

However, if that information is available via the network via SOAP services then I can use a SOAP client to query these sources. I should be able to aggregate the results using the unique identifiers these sets of information have in common - postcodes. I may well use RDF-based tools to assist aggregation at my end, with inferencing capabilities to support querying over the aggregated data (perhaps using OWL for example in order to link equivalent terms for the cases where various data providers have used different terms for "number of bedrooms" and so on).

In a perhaps more globablly "highly evolved" - semantically - version of the same context, a data provider on the web - an estate agent's online service for example - might in fact choose to use an "intelligent RDF backend" to aggregate objects in order to centralise access to some of these resources. By doing this, a data provider thereby enhances its business services by allowing end users (or other services - maybe an online jobs recruitment service) a central point to access distributed information, rather than having to develop their own aggregator clients.

The main advantage of using RDF to store the data in the backend database is that, whilst a range of content providers may publish data all related to a similar domain, their choice of data modelling will naturally vary from provider to provider. The RDF standard (plus OWL for example) makes it simpler to aggregate such related but heterogeneous data. It provides the basis for a simple inferencing layer, meaning that for example my aggregated data store can be simply designed to 'know' that a "semi-detached bungalow" and a "detached bungalow" are both sub-classes of a type called "bungalow", which itself is a sub-class of a type called "residential property" and so it may continue. Or that resources which one content provider may classify as "flats/apartments" are the same as what another content provider classifies as "apartments or flats". And so on, this system being fully extensible and flexible in RDF, meaning that not only is it quite 'clever' in terms of the sorts of questions that can be asked of and answered by it, but it also may be maintained with less overheads in terms of human effort than when using, say, a traditional relational database approach. It will also be the case that even if a data provider should add new attributes to the objects that they are modelling, when the objects are unmarshalled at the client end they will not break the service's model used for aggregating the data (RDF resources are allowed arbitrary, non-reserved properties).

The main advantage of using SOAP as the transportation protocol for such objects is that the SOAP specification supports the SOAP encoding data model which promotes the standardised serialisation of arbitrarily complex objects (allowing multi-reference and cyclic data structures). And this remains consistent in either an RDF-aware or RDF-non-aware context.

The document/literal versus RPC/Encoded debate for SOAP services - Industry support for SOAP messaging styles has recently - over the lifetime of the ☞SWAD-e project - swung in favour of Document/Literal. In this section we take a brief look at the place of SOAP encoding in the Soap 1.2 specification and also its absence in the WS-I Basic Profile. We consider some of the arguments against using SOAP encoding for Web Services and note the potential for using document/literal styled SOAP Web Services for the Semantic Web.

The place of SOAP Encoding in Soap 1.2

Having been in section 5 of the ☞Soap 1.1 specification, the SOAP encoding is revised and moved to an adjunct section in the SOAP 1.2 specification (see ☞http://www.w3.org/TR/soap12-part2/#soapenc). Here it states "This encoding MAY be used to transmit data in SOAP header blocks and/or SOAP bodies. Other data models, alternate encodings of the SOAP Data Model as well as unencoded data MAY also be used in SOAP messages".

The WS-I Basic Profile The ☞Web Services Interoperability Organization, formed in February 2002 (and led by Microsoft and IBM), published the ☞Basic Profile 1.0 specification in August 2003. The specifications covered by the Basic Profile include SOAP 1.1, WSDL 1.1, UDDI 2.0, XML 1.0 and XML Schema, with the aim of promoting Web services interoperability. With regards to soap encoding, the specification says "The soap:encodingStyle attribute is used to indicate the use of a particular scheme in the encoding of data into XML. However, this introduces complexity, as this function can also be served by the use of XML Namespaces. As a result, the Profile prefers the use of literal, non-encoded XML." And it goes on to specify that this attribute must not be used.

It appears that the argument against using soap encoding is "why bring another sort of encoding into the arena when we already have XML and XML namespaces?". However, semantic web initiatives frequently point to the clear gap left by XML in a context where it is important to maximise semantic coherence at machine-to-machine level across distributed web services. In contrast to XML, the SOAP data encoding allows for multi-referencing on a grand scale and, when augmented with RDF, RDFS and OWL, opens up the space occupied by web services such that the links between vast repositories of related but heterogenous data may be exploited in a largely automated and arguably efficient fashion. It is hard to see how such links can be managed sensibly using the XML tree structure. A directed graph structure such as specified by the SOAP data encoding offers plenty in terms of managing the way information may be merged on the network - with RDF-based layers making it usefully 'meaningful'.

None-the-less, RDF has its own RDF/XML syntax and parallel SWADE work is reviewing the relationship of RDF/XML to XML, for example looking at the development of a normalised RDF/XML. This means that, should a SOAP-based web service require to be document/literal based, it might then be configured to offer SOAP message responses where the SOAP message content is encoded in RDF/XML. Schemas - both RDF and XML schemas - might then be associated with this same content (via namespaces) to permit either or both of the XML processing and RDF processing alternatives on the client side. Again, this promotes a relaxed situation in which both RDF-aware and non RDF-aware services may share out data to RDF-aware and non RDF-aware consumer applications equally.

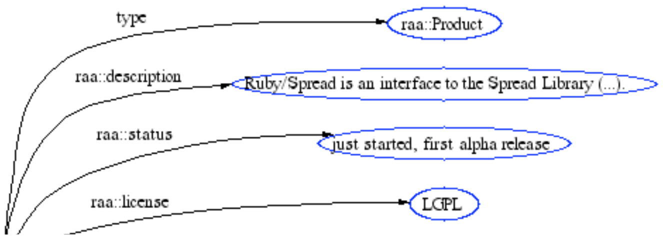
It is recommended that work that follows on from this workpackage look more closely at just this sort Document/Literal option. In the example architecture described below, scenarios are limited by the sort of data structures that can be handed over from Sesame (the RDF database) to the Axis server software (for creating SOAP data encoded SOAP body content). Only Sesame's tabulated data structures for result sets can be very easily mapped to Java beans for straight forward consumption by Axis' bean and array serialisers. RDF result sets encoded in RDF/XML would require a custom serialiser to be written for Axis - one that would handle a complete RDF to SOAP mapping. We have not conducted full investigations in this respect, but such a feature would be required to handle arbitrary RDF structures. At this point the utility of a Document/Literal options becomes apparent again.

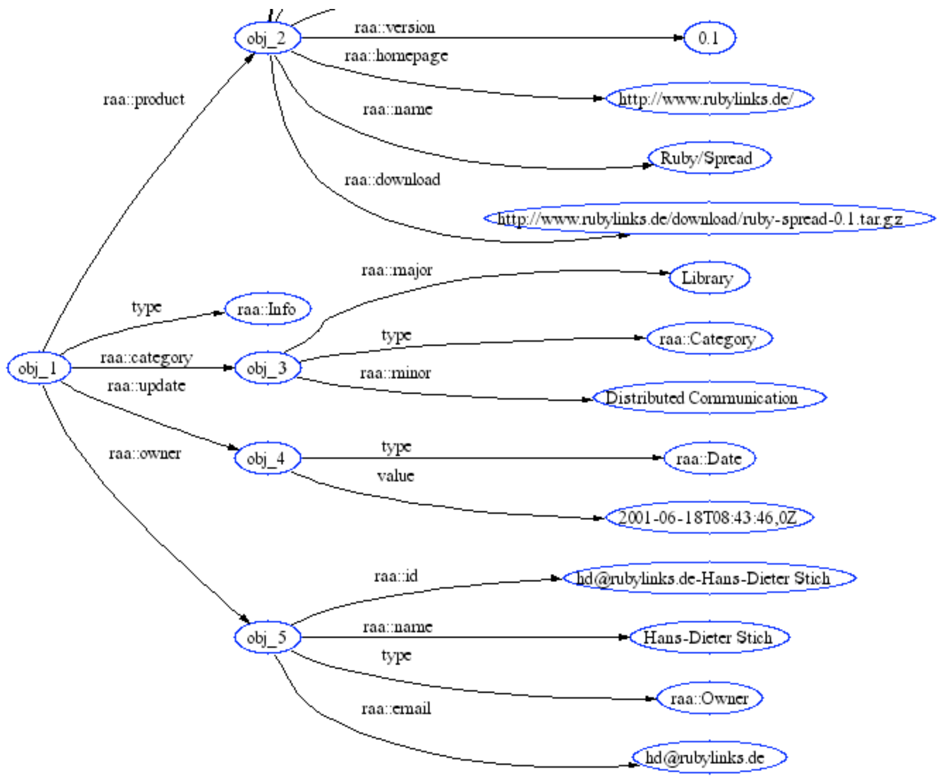
Section 2: Architecture for an example Semantic Web Service

This section takes a quick look at the data being exchanged over the network in the specific scenario we have produced for this report: how that data originates, how objects are "preserved" during transportation, and what can be done with the data on the receiving end as it were.

The exampled web service architecture is obviously not intended to be representative of the scale and capabilities of current or future web services, but simply to illustrate the extent to which SOAP-to-RDF mappings are possible, and indeed useful in the context of web services. In other words, this is merely a proof-of-concept demonstration.
We begin with another diagram, similar to the more general ☞overview diagram above, but specifically labelled with the java classes and stylesheets used in our example scenario.

In our example scenario the SOAP, RPC-Encoded style service that we have called "rubysoapservices", happens to hold a small amount of data about Ruby software products. These software products are encapsulated as what we have named "rubyinfo" objects which essentially are objects that have "product", "owner", "update" and "category" properties - themselves objects, with further properties such as "category_major", "category_minor" etc. This is best explained quickly with the diagram from this workpackage's ☞first report:


type
raa::Product
raa::description
Ruby/Spread is an interface to the Spread Library (...).
raa::status
just started, first alpha release
raa::license
LGPL



WSDL for the Web Service - Please take a look at the ☞WSDL we have for our example (localhost) Ruby Soap Service. You will see that the "request" message name is rubyQueryRequest and that it contains one part, (not named - we could have more sensibly given this a generic name such as "query_expresssion", say) of type "xsd:string".

```
<wsdl:message name="rubyQueryRequest">
 <wsdl:part name="in0" type="xsd:string" />
</wsdl:message>
```

REQUEST MESSAGE - Here is an example request message in which we see that the single string part to the RPC message body is in fact an RQL query (for information on the RQL language as interpreted for Sesame, see ☞http://sesame.aidministrator.nl/publications/rql-tutorial.html). This soap message specifies the encoding style to be used (soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"). And the RQL query effectively says:

"give me all objects that have PRODUCT_NAME, VERSION etc properties, using the namespace ruby = http://example.ruby-language.org/rda-vocab/test2.rdf#.

```
POST /swadWP4/services/RubySoapServiceTest HTTP/1.0Content-Type: text/xml; charset=utf-8Accept: application/soap+xml,
application/dime, multipart/related, text/*User-Agent: Axis/1.1RC1Host: 127.0.0.1Cache-Control: no-cachePragma:
no-cacheSOAPAction: ""Content-Length: 1271<?xml version="1.0"
encoding="UTF-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
                    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
                    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
 <soapenv:Body>
  <ns1:rubyQuery soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:ns1="urn:rubysoapservices">
   <arg1 xsi:type="xsd:string">select PRODUCT_NAME, VERSION, STATUS, HOMEPAGE, DOWNLOAD, LICENSE, DESC, EMAIL,
                                 OWNER_NAME, OWNER_ID, CATEGORY_MAJOR, CATEGORY_MINOR from
                             {X} ruby:ruby_product . ruby:product_name {PRODUCT_NAME} ,
                             {X} ruby:ruby_product . ruby:version {VERSION} ,
                             {X} ruby:ruby_product . ruby:status {STATUS},
                             {X} ruby:ruby_product . ruby:homepage {HOMEPAGE},
                             {X} ruby:ruby_product . ruby:download {DOWNLOAD},
                             {X} ruby:ruby_product . ruby:license {LICENSE},
                             {X} ruby:ruby_product . ruby:desc {DESC},
                             {X} ruby:ruby_owner . ruby:email {EMAIL} ,
                             {X} ruby:ruby_owner . ruby:owner_name {OWNER_NAME} ,
                             {X} ruby:ruby_owner . ruby:owner_id {OWNER_ID} ,
                             {X} ruby:ruby_category  . ruby:category_major {CATEGORY_MAJOR} ,
                             {X} ruby:ruby_category  . ruby:category_minor {CATEGORY_MINOR}
                             using namespace ruby = http://example.ruby-language.org/rda-vocab/test2.rdf#</arg1>
  </ns1:rubyQuery>
 </soapenv:Body>
</soapenv:Envelope>
```

We have permitted the request message to contain a single string parameter deliberately so as to allow the testing of different query expressions on the remote service. **We could of course specify several seperate queries (for e.g. asking all ruby information objects with product_name value is so-and-so) in the WSDL** . Normally we would deploy the service so as the parameter would be more informatively named than simply "arg1" (the Axis default).

```
 HTTP/1.1 200 OKSet-Cookie: JSESSIONID=722B237142FD7F9C1187FE3509DEE13A;
Path=/swadWP4Content-Type: text/xml;
```

```
 charset=utf-8Date: Thu, 16 Oct 2003 13:18:26 GMTServer: Apache
Coyote/1.0 Connection: close
<?xml version="1.0" encoding="UTF-8"?>
 <soapenv:Envelope
 xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
 <soapenv:Body>
   <ns1:rubyQueryResponse
soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:ns1="urn:rubysoapservices">
    <rubyQueryReturn xsi:type="soapenc:Array"
soapenc:arrayType="ns1:RubyInfo[3]"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/">
    <item href="#id0"/>
    <item href="#id1"/>
    <item href="#id2"/>
   </rubyQueryReturn>
  </ns1:rubyQueryResponse>
   <multiRef id="id0" soapenc:root="0"
soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
 xsi:type="ns2:RubyInfo"
 xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:ns2="urn:rubysoapservices">
   <ruby_category href="#id3"/>
   <ruby_owner href="#id4"/>
   <ruby_product href="#id5"/>
   <update xsi:type="xsd:string" xsi:nil="true"/>
  </multiRef>
  <multiRef id="id1" soapenc:root="0"
soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
xsi:type="ns3:RubyInfo" xmlns:ns3="urn:rubysoapservices"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/">
   <ruby_category href="#id6"/>
   <ruby_owner href="#id7"/>
   <ruby_product href="#id8"/>
   <update xsi:type="xsd:string" xsi:nil="true"/>
  </multiRef>
  <multiRef id="id2" soapenc:root="0"
soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
xsi:type="ns4:RubyInfo" xmlns:ns4="urn:rubysoapservices"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/">
   <ruby_category href="#id9"/>
   <ruby_owner href="#id10"/>
   <ruby_product href="#id11"/>
   <update xsi:type="xsd:string" xsi:nil="true"/>
  </multiRef>
  <multiRef id="id7" soapenc:root="0"
soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
xsi:type="ns5:Owner" xmlns:ns5="urn:rubysoapservices"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/">
   <email xsi:type="xsd:string">mailto:neoneye@diku.dk</email>
   <owner_id xsi:type="xsd:string">neoneye@diku.dk-Simon -neoneye- Strandgaard</owner_id>
   <owner_name xsi:type="xsd:string">Simon -neoneye- Strandgaard</owner_name>
  </multiRef>
   <multiRef id="id4" soapenc:root="0"
soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
 xsi:type="ns6:Owner" xmlns:ns6="urn:rubysoapservices"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/">
   <email xsi:type="xsd:string">mailto:hd@rubylinks.de</email>
   <owner_id xsi:type="xsd:string">hd@rubylinks.de-Hans-Dieter Stich</owner_id>
   <owner_name xsi:type="xsd:string">Hans-Dieter Stich</owner_name>
  </multiRef>
   <multiRef id="id3" soapenc:root="0"
soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
 xsi:type="ns7:Category" xmlns:ns7="urn:rubysoapservices"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/">
   <category_major xsi:type="xsd:string">Library</category_major>
   <category_minor xsi:type="xsd:string">Distributed Communication</category_minor>
  </multiRef>
   <multiRef id="id10" soapenc:root="0"
soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
 xsi:type="ns8:Owner" xmlns:ns8="urn:rubysoapservices"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/">
   <email xsi:type="xsd:string">mailto:cilibrar@ofb.net</email>
   <owner_id xsi:type="xsd:string">cilibrar@ofb.net-Rudi Cilibrasi</owner_id>
   <owner_name xsi:type="xsd:string">Rudi Cilibrasi</owner_name>
  </multiRef>
  <multiRef id="id9" soapenc:root="0" soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
 xsi:type="ns9:Category" xmlns:ns9="urn:rubysoapservices"
 xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/">
   <category_major xsi:type="xsd:string">Library</category_major>
   <category_minor xsi:type="xsd:string">Parallel</category_minor>
  </multiRef>
   <multiRef id="id8" soapenc:root="0"
soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
 xsi:type="ns10:Product" xmlns:ns10="urn:rubysoapservices"
 xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/">
    <desc xsi:type="xsd:string">pure ruby editor for pair-programming</desc>
    <download
xsi:type="xsd:string">http://prdownloads.sourceforge.net/
metaeditor/aeditor-ruby-0.3.tar.gz?download</download>
    <homepage
xsi:type="xsd:string">http://metaeditor.sf.net/</homepage>
   <license xsi:type="xsd:string">BSD</license>
   <product_name xsi:type="xsd:string">aeditor</product_name>
   <status xsi:type="xsd:string">alpha</status>
   <version xsi:type="xsd:string">0.3</version>
  </multiRef>
  <multiRef id="id11" soapenc:root="0"
soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
 xsi:type="ns11:Product" xmlns:ns11="urn:rubysoapservices"
```

```
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/">
   <desc xsi:type="xsd:string">MPI Ruby</desc>
   <download
xsi:type="xsd:string">http://prdownloads.sourceforge.net/mpiruby/
mpi_ruby-0.4.tar.bz2?download</download>
    <homepage
xsi:type="xsd:string">http://mpiruby.sourceforge.net/</homepage>
    <license xsi:type="xsd:string">OpenSource</license>
   <product_name xsi:type="xsd:string">mpi_ruby</product_name>
    <status xsi:type="xsd:string">Beta - seems to work with some
testing</status>
   <version xsi:type="xsd:string">0.4</version>
  </multiRef>
   <multiRef id="id5" soapenc:root="0"
soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
 xsi:type="ns12:Product" xmlns:ns12="urn:rubysoapservices"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/">
    <desc xsi:type="xsd:string">Ruby/Spread is an interface to the
Spread Library at
http://www.spread.org/. Spread is a toolkit and daemon that
provides multicast and group communications support to
applications across local and wide area networks. Spread is
designed to make it easy to write groupware, networked
multimedia, reliable server, and collaborative work
applications.</desc>
    <download
xsi:type="xsd:string">http://www.rubylinks.de/download/ruby-spread-0.1.tar.gz</download>
    <homepage
xsi:type="xsd:string">http://www.rubylinks.de/</homepage>
    <license xsi:type="xsd:string">LGPL</license>
    <product_name
xsi:type="xsd:string">ruby-spread</product_name>
    <status xsi:type="xsd:string">just started, first alpha
release</status>
    <version xsi:type="xsd:string">0.1</version>
  </multiRef>
  <multiRef id="id6" soapenc:root="0"
soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
xsi:type="ns13:Category" xmlns:ns13="urn:rubysoapservices"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/">
    <category_major
xsi:type="xsd:string">Application</category_major>
   <category_minor xsi:type="xsd:string">Editor</category_minor>
  </multiRef>
 </soapenv:Body>
</soapenv:Envelope>
```

This response message effectively shows that the soap service (i.e. where Axis "talked" to Sesame) returned an array of 3 RubyInfo objects ("rubyQueryReturn xsi:type="soapenc:Array" soapenc:arrayType="ns1:RubyInfo[3]" xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/").

Each of these objects is referenced by id's and contain further properties, ruby_category, ruby_owner, ruby_product and update. Each of these reference further objects (except in the case of update which is just a string value). You will see that these objects correspond to the types given in the service's ☞WSDL, which also indicates that the response message will contain an array of ruby info objects:

```
 <wsdl:message name="rubyQueryResponse">
  <wsdl:part name="rubyQueryReturn" type="impl:ArrayOfRubyInfo" />
 </wsdl:message>
```

Soap Encoding Specifications - transforming from SOAP 1.1 and SOAP 1.2 -

Axis uses soap encoding as specified in Section 5 in ☞the SOAP 1.1 specification (see XML schema at ☞http://schemas.xmlsoap.org/soap/encoding/). In the SOAP Version 1.2, the soap data model and soap encoding specs are given in ☞Part 2: Adjuncts (see XML schema at ☞http://www.w3.org/2003/05/soap-encoding). For information about the changes to the Soap encoding specification from 1.1 to 1.2, see information in the primer at ☞http://www.w3.org/TR/soap12-part0/#L579. Briefly listed these are:

- The syntax for the serialization of an array has been changed in SOAP 1.2 from that in SOAP 1.1.
- The support provided in SOAP 1.1 for partially transmitted and sparse arrays is not available in SOAP 1.2.
- SOAP 1.2 allows the inline (embedded) serialization of multiref values.
- The href attribute in SOAP 1.1 (of type xs:anyURI) is called enc:ref in SOAP 1.2 and is of type IDREF.
- In SOAP 1.2, omitted accessors of compound types are made equal to NILs.
- SOAP 1.2 provides several fault sub-codes for indicating encoding errors.
- SOAP 1.2 has removed generic compound values from the SOAP Data Model.
- Types on nodes are made optional in SOAP 1.2.
- SOAP 1.2 has added an optional attribute enc:nodeType to elements encoded using SOAP encoding that identifies its structure (i.e., a simple value, a struct or an array).

For more information, see also an analysis by Marc Hadley (who works for XML Technology Centre at Sun Microsystems, and is co-editor of the SOAP 1.2 specification) at ☞http://www.hadleynet.org/marc/whatsnew.html#S2

We provide a ☞Stylesheet for transforming soap encodings from the SOAP 1.1 specification to that of the 1.2 specification (Author: Max Froumentin). After transformation, the body of the above response message looks like this:

```
<?xml version="1.0" encoding="utf-8"?><Body xmlns="http://www.w3.org/2003/05/soap-envelope">
   <ns1:rubyQueryResponse xmlns:ns1="urn:rubysoapservices"
 xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:ns0="http://www.w3.org/2003/05/soap-encoding"
 ns0:encodingStyle="http://www.w3.org/2003/05/soap-encoding">
   <rubyQueryReturn xmlns=""
```

```
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 ns0:itemType="ns1:RubyInfo" ns0:arraySize="3">
    <item ns0:ref="id0"/>
    <item ns0:ref="id1"/>
    <item ns0:ref="id2"/>
   </rubyQueryReturn>
  </ns1:rubyQueryResponse>
  <multiRef xmlns=""
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:ns2="urn:rubysoapservices"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:ns1="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ns0="http://www.w3.org/2003/05/soap-encoding" ns0:id="id0"
ns0:encodingStyle="http://www.w3.org/2003/05/soap-encoding"
xsi:type="ns2:RubyInfo">
   <ruby_category ns0:ref="id3"/>
   <ruby_owner ns0:ref="id4"/>
   <ruby_product ns0:ref="id5"/>
   <update xsi:type="xsd:string" xsi:nil="true"/>
  </multiRef>
  <multiRef xmlns="" xmlns:ns3="urn:rubysoapservices"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:ns1="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ns0="http://www.w3.org/2003/05/soap-encoding"
ns0:id="id1" ns0:encodingStyle="http://www.w3.org/2003/05/soap-encoding"
xsi:type="ns3:RubyInfo">
   <ruby_category ns0:ref="id6"/>
   <ruby_owner ns0:ref="id7"/>
   <ruby_product ns0:ref="id8"/>
   <update xsi:type="xsd:string" xsi:nil="true"/>
  </multiRef>
  <multiRef xmlns="" xmlns:ns4="urn:rubysoapservices"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:ns1="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ns0="http://www.w3.org/2003/05/soap-encoding"
ns0:id="id2" ns0:encodingStyle="http://www.w3.org/2003/05/soap-encoding"
xsi:type="ns4:RubyInfo">
   <ruby_category ns0:ref="id9"/>
   <ruby_owner ns0:ref="id10"/>
   <ruby_product ns0:ref="id11"/>
   <update xsi:type="xsd:string" xsi:nil="true"/>
  </multiRef>
  <multiRef xmlns="" xmlns:ns5="urn:rubysoapservices"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:ns1="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ns0="http://www.w3.org/2003/05/soap-encoding"
ns0:id="id7" ns0:encodingStyle="http://www.w3.org/2003/05/soap-encoding"
xsi:type="ns5:Owner">
   <email xsi:type="xsd:string">mailto:neoneye@diku.dk</email>
    <owner_id xsi:type="xsd:string">neoneye@diku.dk-Simon -neoneye-
Strandgaard</owner_id>
   <owner_name xsi:type="xsd:string">Simon -neoneye-
Strandgaard</owner_name>
  </multiRef>
  <multiRef xmlns="" xmlns:ns6="urn:rubysoapservices"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:ns1="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ns0="http://www.w3.org/2003/05/soap-encoding"
ns0:id="id4" ns0:encodingStyle="http://www.w3.org/2003/05/soap-encoding"
xsi:type="ns6:Owner">
   <email xsi:type="xsd:string">mailto:hd@rubylinks.de</email>
    <owner_id xsi:type="xsd:string">hd@rubylinks.de-Hans-Dieter
Stich</owner_id>
   <owner_name xsi:type="xsd:string">Hans-Dieter Stich</owner_name>
  </multiRef>
  <multiRef xmlns="" xmlns:ns7="urn:rubysoapservices"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:ns1="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ns0="http://www.w3.org/2003/05/soap-encoding"
ns0:id="id3" ns0:encodingStyle="http://www.w3.org/2003/05/soap-encoding"
xsi:type="ns7:Category">
   <category_major xsi:type="xsd:string">Library</category_major>
    <category_minor xsi:type="xsd:string">Distributed
Communication</category_minor>
  </multiRef>
  <multiRef xmlns="" xmlns:ns8="urn:rubysoapservices"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:ns1="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ns0="http://www.w3.org/2003/05/soap-encoding" ns0:id="id10"
ns0:encodingStyle="http://www.w3.org/2003/05/soap-encoding"
xsi:type="ns8:Owner">
    <email xsi:type="xsd:string">mailto:cilibrar@ofb.net</email>
    <owner_id xsi:type="xsd:string">cilibrar@ofb.net-Rudi
Cilibrasi</owner_id>
   <owner_name xsi:type="xsd:string">Rudi Cilibrasi</owner_name>
   </multiRef>
   <multiRef xmlns="" xmlns:ns9="urn:rubysoapservices"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:ns1="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ns0="http://www.w3.org/2003/05/soap-encoding" ns0:id="id9"
ns0:encodingStyle="http://www.w3.org/2003/05/soap-encoding"
xsi:type="ns9:Category">
   <category_major xsi:type="xsd:string">Library</category_major>
   <category_minor xsi:type="xsd:string">Parallel</category_minor>
  </multiRef>
  <multiRef xmlns="" xmlns:ns10="urn:rubysoapservices"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:ns1="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ns0="http://www.w3.org/2003/05/soap-encoding" ns0:id="id8"
ns0:encodingStyle="http://www.w3.org/2003/05/soap-encoding"
xsi:type="ns10:Product">
    <desc xsi:type="xsd:string">pure ruby editor for
pair-programming</desc>
   <download xsi:type="xsd:string">http://prdownloads.sourceforge.ne
t/metaeditor/aeditor-ruby-0.3.tar.gz?download</download>
    <homepage
xsi:type="xsd:string">http://metaeditor.sf.net/</homepage>
   <license xsi:type="xsd:string">BSD</license>
   <product_name xsi:type="xsd:string">aeditor</product_name>
   <status xsi:type="xsd:string">alpha</status>
   <version xsi:type="xsd:string">0.3</version>
  </multiRef>
  <multiRef xmlns="" xmlns:ns11="urn:rubysoapservices"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:ns1="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ns0="http://www.w3.org/2003/05/soap-encoding" ns0:id="id11"
ns0:encodingStyle="http://www.w3.org/2003/05/soap-encoding"
xsi:type="ns11:Product">
   <desc xsi:type="xsd:string">MPI Ruby</desc>
    <download
xsi:type="xsd:string">http://prdownloads.sourceforge.net/mpiruby/mpi_ruby-0.4.
tar.bz2?download</download>
    <homepage
xsi:type="xsd:string">http://mpiruby.sourceforge.net/</homepage>
   <license xsi:type="xsd:string">OpenSource</license>
   <product_name xsi:type="xsd:string">mpi_ruby</product_name>
   <status xsi:type="xsd:string">Beta - seems to work with some
testing</status>
   <version xsi:type="xsd:string">0.4</version>
  </multiRef>
  <multiRef xmlns="" xmlns:ns12="urn:rubysoapservices"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:ns1="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ns0="http://www.w3.org/2003/05/soap-encoding" ns0:id="id5"
ns0:encodingStyle="http://www.w3.org/2003/05/soap-encoding"
xsi:type="ns12:Product">
   <desc xsi:type="xsd:string">Ruby/Spread is an interface
to the Spread Library at
http://www.spread.org/. Spread is a toolkit and daemon that
provides multicast and group communications support to
applications across local and wide area networks. Spread is
designed to make it easy to write groupware, networked
multimedia, reliable server, and collaborative work
applications.</desc>
    <download
xsi:type="xsd:string">http://www.rubylinks.de/download/ruby-spread-0.1.ta
r.gz</download>
   <homepage
xsi:type="xsd:string">http://www.rubylinks.de/</homepage>
   <license xsi:type="xsd:string">LGPL</license>
   <product_name xsi:type="xsd:string">ruby-spread</product_name>
   <status xsi:type="xsd:string">just started, first alpha
release</status>
   <version xsi:type="xsd:string">0.1</version>
  </multiRef>
  <multiRef xmlns="" xmlns:ns13="urn:rubysoapservices"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:ns1="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ns0="http://www.w3.org/2003/05/soap-encoding" ns0:id="id6"
ns0:encodingStyle="http://www.w3.org/2003/05/soap-encoding"
xsi:type="ns13:Category">
   <category_major
xsi:type="xsd:string">Application</category_major>
   <category_minor xsi:type="xsd:string">Editor</category_minor>
  </multiRef>
 </Body>
```

Transforming from SOAP 1.2 encoding to RDF/XML - the "round trip" - The stylesheet we have developed (Max Froumentin and Nikki Rogers) for this transformation is ☞soap_to_rdf.xsl. It is not completely generic - the first element in the soap body to be treated as an 'edge' has to be identified as noted in the stylesheet. Also the choice of name for SOAP accessors that have 'id' attributes is arbitrary and for Axis is "multiRef". Therefore these multiref accessors are matched as part of the transformation in order to be mapped to RDF typed elements.

When applied to the above response body data, it produces:

```
<?xml version="1.0" encoding="utf-8"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:env="http://www.w3.org/2003/05/soap-envelope"
xmlns:enc="http://www.w3.org/2003/05/soap-encoding"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:ns1="urn:rubysoapservices" xmlns:ruby="urn:rubysoapservices">
     <rdf:Description>
      <rubyQueryResponse xmlns="urn:rdf2soap">

          <rubyQueryReturn>

             <item rdf:nodeID="id0"/>
```

```
            <item rdf:nodeID="id1"/>

            <item rdf:nodeID="id2"/>

         </rubyQueryReturn>

      </rubyQueryResponse>
   </rdf:Description>

   <rdf:Description rdf:nodeID="id0">
      <rdf:type rdf:resource="ruby:RubyInfo"/>

      <ruby_category xmlns="urn:rdf2soap" rdf:nodeID="id3"/>

      <ruby_owner xmlns="urn:rdf2soap" rdf:nodeID="id4"/>

      <ruby_product xmlns="urn:rdf2soap" rdf:nodeID="id5"/>

      <update xmlns="urn:rdf2soap">
         <rdf:Description/>
      </update>

   </rdf:Description>

   <rdf:Description rdf:nodeID="id1">
      <rdf:type rdf:resource="ruby:RubyInfo"/>

      <ruby_category xmlns="urn:rdf2soap" rdf:nodeID="id6"/>

      <ruby_owner xmlns="urn:rdf2soap" rdf:nodeID="id7"/>

      <ruby_product xmlns="urn:rdf2soap" rdf:nodeID="id8"/>

      <update xmlns="urn:rdf2soap">
         <rdf:Description/>
      </update>

   </rdf:Description>

   <rdf:Description rdf:nodeID="id2">
      <rdf:type rdf:resource="ruby:RubyInfo"/>

      <ruby_category xmlns="urn:rdf2soap" rdf:nodeID="id9"/>

      <ruby_owner xmlns="urn:rdf2soap" rdf:nodeID="id10"/>

      <ruby_product xmlns="urn:rdf2soap" rdf:nodeID="id11"/>

      <update xmlns="urn:rdf2soap">
         <rdf:Description/>
      </update>

   </rdf:Description>

   <rdf:Description rdf:nodeID="id7">
      <rdf:type rdf:resource="ruby:Owner"/>

      <email xmlns="urn:rdf2soap"
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">mailto:neoneye@diku.dk</email>

      <owner_id xmlns="urn:rdf2soap"
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">neoneye@diku.dk-Simon -neoneye- Strandgaard</owner_id>

      <owner_name xmlns="urn:rdf2soap"
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Simon -neoneye- Strandgaard</owner_name>

   </rdf:Description>

   <rdf:Description rdf:nodeID="id4">
      <rdf:type rdf:resource="ruby:Owner"/>

      <email xmlns="urn:rdf2soap"
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">mailto:hd@rubylinks.de</email>

      <owner_id xmlns="urn:rdf2soap"
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">hd@rubylinks.de-Hans-Dieter Stich</owner_id>

      <owner_name xmlns="urn:rdf2soap"
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Hans-Dieter Stich</owner_name>

   </rdf:Description>

   <rdf:Description rdf:nodeID="id3">
      <rdf:type rdf:resource="ruby:Category"/>

      <category_major xmlns="urn:rdf2soap"
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Library</category_major>

      <category_minor xmlns="urn:rdf2soap"
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Distributed Communication</category_minor>

   </rdf:Description>

   <rdf:Description rdf:nodeID="id10">
      <rdf:type rdf:resource="ruby:Owner"/>

      <email xmlns="urn:rdf2soap"
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">mailto:cilibrar@ofb.net</email>

       <owner_id xmlns="urn:rdf2soap"
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">cilibrar@ofb.net-Rudi Cilibrasi</owner_id>
```

```
       <owner_name xmlns="urn:rdf2soap"
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Rudi Cilibrasi</owner_name>

   </rdf:Description>

   <rdf:Description rdf:nodeID="id9">
      <rdf:type rdf:resource="ruby:Category"/>

       <category_major xmlns="urn:rdf2soap"
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Library</category_major>

       <category_minor xmlns="urn:rdf2soap"
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Parallel</category_minor>

   </rdf:Description>

   <rdf:Description rdf:nodeID="id8">
      <rdf:type rdf:resource="ruby:Product"/>

       <desc xmlns="urn:rdf2soap"
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">pure ruby editor for pair-programming</desc>

       <download xmlns="urn:rdf2soap"
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">http://prdownloads.sourceforge.net/
metaeditor/aeditor-ruby-0.3.tar.gz?download</download>

       <homepage xmlns="urn:rdf2soap"
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
>http://metaeditor.sf.net/</homepage>

       <license xmlns="urn:rdf2soap"
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">BSD</license>

       <product_name xmlns="urn:rdf2soap"
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">aeditor</product_name>

       <status xmlns="urn:rdf2soap"
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">alpha</status>

       <version xmlns="urn:rdf2soap"
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">0.3</version>

   </rdf:Description>

   <rdf:Description rdf:nodeID="id11">
      <rdf:type rdf:resource="ruby:Product"/>

       <desc xmlns="urn:rdf2soap"
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">MPI Ruby</desc>

       <download xmlns="urn:rdf2soap"
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">http://prdownloads.sourceforge.net/mpiruby/mpi_ruby-0.4.
tar.bz2?download</download>

       <homepage xmlns="urn:rdf2soap"
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">http://mpiruby.sourceforge.net/</homepage>

       <license xmlns="urn:rdf2soap"
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">OpenSource</license>

       <product_name xmlns="urn:rdf2soap"
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">mpi_ruby</product_name>

       <status xmlns="urn:rdf2soap"
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Beta -
seems to work with some testing</status>

       <version xmlns="urn:rdf2soap"
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">0.4</version>

   </rdf:Description>

   <rdf:Description rdf:nodeID="id5">
      <rdf:type rdf:resource="ruby:Product"/>

       <desc xmlns="urn:rdf2soap"
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Ruby/Spread is
an interface to the Spread Library at
http://www.spread.org/. Spread is a toolkit and daemon that
provides multicast and group communications support to
applications across local and wide area networks. Spread is
designed to make it easy to write groupware, networked
multimedia, reliable server, and collaborative work
applications.</desc>

       <download xmlns="urn:rdf2soap"
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
http://www.rubylinks.de/download/ruby-spread-0.1.tar.gz</download>

       <homepage xmlns="urn:rdf2soap"
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
http://www.rubylinks.de/</homepage>

       <license xmlns="urn:rdf2soap"
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">LGPL</license>

       <product_name xmlns="urn:rdf2soap"
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
ruby-spread</product_name>

       <status xmlns="urn:rdf2soap"
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
```

```
just started, first alpha release</status>

       <version xmlns="urn:rdf2soap"
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">0.1</version>

   </rdf:Description>

   <rdf:Description rdf:nodeID="id6">
      <rdf:type rdf:resource="ruby:Category"/>

       <category_major xmlns="urn:rdf2soap"
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Application</category_major>

       <category_minor xmlns="urn:rdf2soap"
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Editor</category_minor>

   </rdf:Description>

</rdf:RDF>
```

You may run both the original data uploaded to sesame and the above RDF (obtained from a transformation of the SOAP response body's data) through the online RDF validator at ☞http://www.w3.org/RDF/Validator/ to compare triples that both graphs produce. They are simliar with a couple of differences [TODO : 1. leading edge of soap body response 2. datatyping added in stylesheet - our version of sesame could not cope with this, more recent version might be able to]

Section 3: Guide to integrating Axis and Sesame

Installation of axis and sesame is quite a lengthy process, and we hope to wrap up what we have put together for this report as a downloadable war. file with an ant build to assist installation and configuration. In the meantime, we provide a summary and some more detailed instructions to supplement those available from the Axis and Sesame websites.

Summary instructions - Obtaining Sesame
See the ☞Sesame server installation instructions which describe how the downloads for a sesame installation. Briefly they are:

Overall: you will need to download and install Sesame, and relational database software (MySQL for example) plus a suitable JDBC driver. You will also need to download Axis and also have Ant installed so as to use the Axis build targets. Here are some summary instructions (more detailed instructions follow):

For Sesame, you will need to create a new user account (e.g. called 'sesame') in MySQL, a new database (e.g. called 'rubyinfodb'), and then grant the user all privileges on the database you created. You can upload our ☞sample 'Ruby Info' data to the database as described below.

Assuming tomcat is your servlet container: In your tomcat web apps directory you will need 2 directories - one for sesame & one for axis. We have used:

```
[PATH TO TOMCAT]/webapps/ -- [SESAME_DIR]
                          -- [SWADWP4_DIR]
```

In the / -- **sesame** directory you will need to extract the **sesame.war** file and edit this file as discussed below:

```
                  [SESAME_DIR]/ WEB-INF / system.conf
```

And also add the appropriate **jars** (those found with the sesame distribution, plus the jdbc driver) in

```
                   [SESAME_DIR]/  WEB-INF / lib /
```

```
Then you will need to edit

                        [SESAME_DIR]/  WEB-INF / web.xml /

We example a test java class below, for testing out
suitable queries on the 'Ruby Info' data, from the commandline, over http.
```

For Apache **Axis**, you will need to download and install Axis. There are several jar files that need to be placed in Axis' lib/ directory. When you build Axis, the compiled classes produced need to be placed in the appropriate [SWADWP4_DIR] in the tomcat servlet container.

Add the appropriate **jars** to

```
                  [SWADWP4_DIR]/  WEB-INF / lib /
```

You will need to check you can get the axis test page and also validate Axis with "happyaxis" before you proceed.
Next you can compile the sample Axis provider class we provide (RubySoapServiceTest.java) and copy the class to a directory you create in tomcat: **[SWADWP4_DIR]/ WEB-INF / classes / rubysoapservices** . In addition you will need the deploy.wsdd & undeploy.wsdd files we provide, and the classes for each of the beans required for this demo (named ☞Category, ☞Product, ☞RubyInfo, ☞Owner).

You can now use the AdminClient tool that comes with Axis to **deploy** the Ruby service (this should update your server-config.wsdd file in the [SWADWP4_DIR]/WEB-INF directory with the information for your new service).

Once successfully deployed ..... you can test the service via the **Axis client** we provide (Client.java), which is called from the commandline and can take various query strings. This client will do two things 1) It will unmarshall returned data back into javabeans (using Axis' built in array and bean deserialisers) 2) Together with **soap_1_1_to_1_2.xsl** and **soap_to_rdf.xsl,** it will output a file named 'transformed_to_rdf_latest.xml' (the results of a soap encoded data reponse from the service, the data now transformed and serialised in the RDF/XML syntax). 1) and 2) represent prerequisit stages in how the client may want to handle data it consumes from the service. 1) is for handling data in an (java) object-oriented, non RDF-aware environment. 2) is for handling data in an RDF e nvironment. 1) and 2) correspond to e.g. B and e.g. A respectively in the ☞diagram that represents this service in ☞Section 2. For client applications that need to deal with XMl documents and associated schema s, there is the ☞Document/Literal style service option, not documented here.

- Sesame itself,
- A DBMS with a JDBC-driver (we have used MySQL version 3.23, installed on the same machine as the sesame software, with the mysql-connector-j-2.0.14-bin.jar for database connectivity) and
- a Java servlet container for running the Sesame servlets (we've used jakarta-tomcat-4.1.12 with j2sdk1.4.0_02)

Sesame and the servlet container

As described in the sesame installation instructions, you will need to place the extracted sesame.war file in a directory (called, say 'sesame') in your [TOMCAT_DIR]/webapps/ directory. And you will need to place the appropriate jar files (the jdbc jar and xerces.jar, soap.jar, icu4j.jar and jena.jar that are found in the sesame distribution) in [SESAME_DIR]/WEB-INF/lib/ (see ☞http://sesame.aidministrator.nl/publications/users/ch01s05.html.)

To configure a fresh installation of Sesame you can copy the example sesame configuration file at [SESAME_DIR]/WEB-INF/system.conf.example. to [SESAME_DIR]/WEB-INF/system.conf. Then you will need to edit this file for the correct path to the directory where Sesame is installed on your machine, and for a new SESAME_PASSWORD. Again, this is all detailed in the Sesame installation instructions.

Uploading the Ruby test data to Sesame

Similarly to as described in the Sesame installation instructions, you can:

- create a new user account (with all privileges granted) - called 'sesame' and then
- create a new MySQL database called 'rubyinfodb' (something like this:

```
C:\mysql\bin>mysql -u sesame -p
Enter password: ******
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 74 to server version: 3.23.54-max-nt

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> CREATE DATABASE rubyinfodb
    -> ;
Query OK, 1 row affected (0.01 sec)

mysql> GRANT ALL ON rubyinfodb.* TO sesame;
Query OK, 0 rows affected (0.01 sec)

mysql> QUIT
Bye

C:\mysql\bin>
```

- With tomcat stopped, now open the file [SESAME_DIR]/WEB-INF/system.conf and add a new entry for the new repository, copying the format of the test one that comes with sesame e.g.:

```
                <repository id="mysql-rubyInfo">
                        <title>MySQL Ruby Products Information DB</title>

                        <sailstack>
                                <!-- SyncRdfSchemaRepository prevents
multiple concurrent
                                transactions and reads during a
transaction -->
                                <sail
class="nl.aidministrator.rdf.sail.sync.SyncRdfSchemaRepository"/>
                                <sail
class="nl.aidministrator.rdf.sail.sql92.MySQLSail">
                                        <param name="jdbcDriver"
value="com.mysql.jdbc.Driver"/>
                                        <param name="jdbcUrl"
value="jdbc:mysql://localhost:3306/rubyinfodb"/>
                                        <param name="user"
value="sesame"/>
                                        <param name="password"
value="sesame"/>
                               </sail>
                        </sailstack>

                        <!--Access Control List can contain zero or more 'user' elements-->
                         <acl worldReadable="true"
worldWritable="true">
                                <user login="testuser"
readAccess="true" writeAccess="true"/>
                        </acl>
                </repository>
```

- startup tomcat
- Bring up http://localhost:8080/[SESAME_DIR] and you should see the new database in the drop down list
- Select this database (rubydb) and choose (from the toolbar) to Add (copy-paste) data (if not url). You can copy the data (representing "Rubyinfo" objects) that we have used for this example: from here: ☞upload data. You should now get all the data parsed and uploaded (Sesame will tell you the results

of the transaction) and you can choose to extract it - have a look at it.

A standalone test of Sesame over http

It is possible to do command-line testing of Sesame (over http) via the SesameClient.java file (SESAME_SOURCE_DIRECTORY\nl\aidministrator\rdf\client\SesameClient.java). We have adapted this client to form the "bridge" between Axis and Sesame as described below. The adapted file is ☞SesameTestStandalone.java. Call this class from the commandline

```
[PATH TO SESAME]>java rubysoapservices\SesameTestStandalone
```

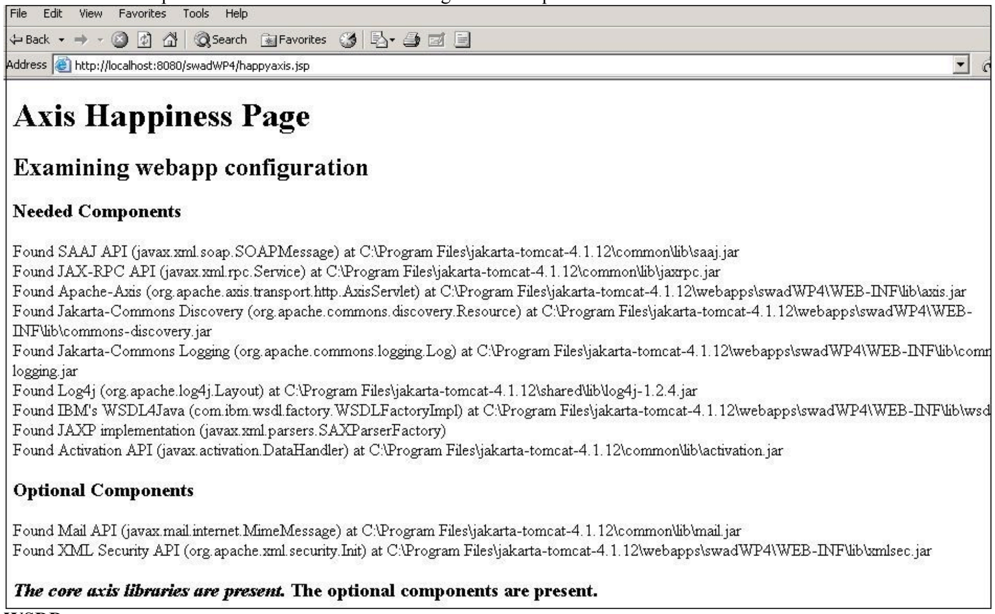
It issues a hardcoded query against the sesame database "RubyInfodb" in which we uploaded data in order to test we can soap<->rdf roundtrip. All matching RubyInfo "Objects" are returned using Sesame's tabulated form for result sets. The result set is then mapped to javabeans in a proprietary way.

The query is String query = "select X, PRODUCT_NAME, VERSION, STATUS, HOMEPAGE, DOWNLOAD, LICENSE, DESC, EMAIL,OWNER_NAME, OWNER_ID, CATEGORY_MAJOR, CATEGORY_MINOR from {X} ruby:ruby_product . ruby:product_name {PRODUCT_NAME} , {X} ruby:ruby_product . ruby:version {VERSION} ,{X} ruby:ruby_product . ruby:status {STATUS}, {X} ruby:ruby_product . ruby:homepage {HOMEPAGE}, {X} ruby:ruby_product . ruby:download {DOWNLOAD}, {X} ruby:ruby_product . ruby:license {LICENSE}, {X} ruby:ruby_product . ruby:desc {DESC}, {X} ruby:ruby_owner . ruby:email {EMAIL} , {X} ruby:ruby_owner . ruby:owner_name {OWNER_NAME} , {X} ruby:ruby_owner . ruby:owner_id {OWNER_ID} , {X} ruby:ruby_category . ruby:category_major {CATEGORY_MAJOR} , {X} ruby:ruby_category . ruby:category_minor {CATEGORY_MINOR} using namespace ruby = http://example.ruby-language.org/rda-vocab/test2.rdf#";

The query string can be adapted according to RQL conventions. The beans required are ☞RubyInfo.java, ☞Category.java, ☞Owner.java ☞Product.java.

Obtaining Axis

We installed Axis as described in Axis' ☞installation instructions and also in the README that comes with the Axis download. REQUIRE: We used Ant Version 1.5.1 in order to use the build for Axis. We placed the Axis classes from the build in TOMCAT_DIR]/webapps/ in a directory we have called 'swadWP4". We performed the "validate" and testing tasks as required for an Axis installation.


File Edit View Favorites Tools Help
Back Search Favorites
Address http://localhost:8080/swadWP4/happyaxis.jsp

Axis Happiness Page

Examining webapp configuration

Needed Components

```
Found SAAJ API (javax.xml.soap.SOAPMessage) at C:\Program Files\jakarta-tomcat-4.1.12\common\lib\saaj.jar
Found JAX-RPC API (javax.xml.rpc.Service) at C:\Program Files\jakarta-tomcat-4.1.12\common\lib\jaxrpc.jar
Found Apache-Axis (org.apache.axis.transport.http.AxisServlet) at C:\Program Files\jakarta-tomcat-4.1.12\webapps\swadWP4\WEB-INF\lib\axis.jar
Found Jakarta-Commons Discovery (org.apache.commons.discovery.Resource) at C:\Program Files\jakarta-tomcat-4.1.12\webapps\swadWP4\WEB-
INF\lib\commons-discovery.jar
Found Jakarta-Commons Logging (org.apache.commons.logging.Log) at C:\Program Files\jakarta-tomcat-4.1.12\webapps\swadWP4\WEB-INF\lib\comm
logging.jar
Found Log4j (org.apache.log4j.Layout) at C:\Program Files\jakarta-tomcat-4.1.12\shared\lib\log4j-1.2.4.jar
Found IBM's WSDL4Java (com.ibm.wsdl.factory.WSDLFactoryImpl) at C:\Program Files\jakarta-tomcat-4.1.12\webapps\swadWP4\WEB-INF\lib\wsd
Found JAXP implementation (javax.xml.parsers.SAXParserFactory)
Found Activation API (javax.activation.DataHandler) at C:\Program Files\jakarta-tomcat-4.1.12\common\lib\activation.jar
```

Optional Components

```
Found Mail API (javax.mail.internet.MimeMessage) at C:\Program Files\jakarta-tomcat-4.1.12\common\lib\mail.jar
Found XML Security API (org.apache.xml.security.Init) at C:\Program Files\jakarta-tomcat-4.1.12\webapps\swadWP4\WEB-INF\lib\xmlsec.jar
```

The core axis libraries are present. **The optional components are present.**

WSDD

The appropriate classes for the service (☞RubySoapServiceTest.java, ☞RubyInfo.java, ☞Category.java, ☞Owner.java ☞Product.java) are placed in the TOMCAT_DIR/webapps/swadWP4/WEB-INF/classes directory, and the appropriate jar files (example for sesame and the sesame client) in the lib directory for this webapp.

A WSDL file for the service can be authored "by hand", or it can be auto-generated by the Axis java2wsdl tool if run in the same webapp directory. For example:

```
java org.apache.axis.wsdl.Java2WSDL
-o semwebserv.wsdl
-l"http://localhost:8080/axis/services/SesameServiceTest"
-n "urn:semwebservices"
-p"semwebservices" "urn:semwebservices"
semwebservices.SesameServiceTest
```

Where:

- -o indicates the name of the output WSDL file
- -l indicates the location of the service
- -n is the target namespace of the WSDL file (which should more correctly be http//... not urn)
- -p indicates a mapping from the package to a namespace. There may be multiple mappings. the class specified contains the interface of the webservice.

In order to deploy a service via Axis it is necessary to create a WSDD file as mentioned earlier. This can be authored "by hand", or from a temporary folder we can now run the Axis wsdl2java tool - with the server-side switch - on the service's classes. This will generate the deploy.wsdd and undeploy.wsdd files (and also server-side skeletons which in fact we do not use in this example). The wsdd generated will look something like this

```
!-- Use this file to deploy some handlers/chains and services      -->
<!-- Two ways to do this:                                           -->
<!--   java org.apache.axis.client.AdminClient deploy.wsdd          -->
<!--      after the axis server is running                          -->
<!-- or                                                             -->
```

```
<!--   java org.apache.axis.utils.Admin client|server deploy.wsdd   -->
<!--      from the same directory that the Axis engine runs         -->

<deployment
    xmlns="http://xml.apache.org/axis/wsdd/"
    xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">

  <!-- Services from RubySoapServiceTestService WSDL service -->

   <service name="RubySoapServiceTest" provider="java:RPC" style="rpc"
use="encoded">
       <parameter name="wsdlTargetNamespace"
value="urn:rubysoapservices"/>
       <parameter name="className"
value="rubysoapservices.RubySoapServiceTest"/>
       <operation name="rubyQuery" qname="operNS:rubyQuery"
xmlns:operNS="urn:rubysoapservices" returnQName="rubyQueryReturn"
returnType="rtns:ArrayOfRubyInfo" xmlns:rtns="urn:rubysoapservices" >
         <parameter name="in0" type="tns:string"
xmlns:tns="http://www.w3.org/2001/XMLSchema"/>
       </operation>
      <parameter name="allowedMethods" value="rubyQuery"/>
      <parameter name="scope" value="Session"/>

      <typeMapping
        xmlns:ns="urn:rubysoapservices"
        qname="ns:ArrayOfRubyInfo"
        type="java:rubysoapservices.RubyInfo[]"
        serializer="org.apache.axis.encoding.ser.ArraySerializerFactory"
        deserializer="org.apache.axis.encoding.ser.ArrayDeserializerFactory"
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
      />
      <typeMapping
        xmlns:ns="urn:rubysoapservices"
        qname="ns:RubyInfo"
        type="java:rubysoapservices.RubyInfo"
        serializer="org.apache.axis.encoding.ser.BeanSerializerFactory"
        deserializer="org.apache.axis.encoding.ser.BeanDeserializerFactory"
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
      />
      <typeMapping
        xmlns:ns="urn:rubysoapservices"
        qname="ns:Owner"
        type="java:rubysoapservices.Owner"
        serializer="org.apache.axis.encoding.ser.BeanSerializerFactory"
        deserializer="org.apache.axis.encoding.ser.BeanDeserializerFactory"
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
      />
      <typeMapping
        xmlns:ns="urn:rubysoapservices"
        qname="ns:Category"
        type="java:rubysoapservices.Category"
        serializer="org.apache.axis.encoding.ser.BeanSerializerFactory"
        deserializer="org.apache.axis.encoding.ser.BeanDeserializerFactory"
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
      />
      <typeMapping
        xmlns:ns="urn:rubysoapservices"
        qname="ns:Product"
        type="java:rubysoapservices.Product"
        serializer="org.apache.axis.encoding.ser.BeanSerializerFactory"
        deserializer="org.apache.axis.encoding.ser.BeanDeserializerFactory"
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
      />
  </service>
</deployment>
```

As you can see from this file, we use the Axis built in RPC provider, with a SOAP data-encoded format for the response message body. The deploy and undeploy.wsdd files in the servlet container (i.e. tomcat in our case) with the service's java class files. In the same directory it is now possible to deploy the service using Axis' AdminClient tool from the command line, like this:

```
   java org.apache.axis.client.AdminClient -lhttp://localhost:8080/swadWP4/services/AdminService
deploy.wsdd
```

Accessing the service

Having deployed the service it should now be possible to access the WSDL for the service from the localhost machine, say, thus:

http://localhost:8080/swadWP4/services/RubySoapServiceTest?wsdl

You can access the service:

- via a custom ☞client, the required beans, and the two stylesheets: ☞soap_1_1_to_1_2.xsl, ☞soap_to_rdf (note we used ☞Saxon for the XSLT to RDF)
- watch the soap messages go back and forth via the Axis tcpmon (TCP Monitor) tool (this is a very handy tool).

Section 4: The Soap data model and the RDF data model- transforming from the former to the latter

[TODO] Rough thinks:

- - SOAP data model - has no equivalent way of defining a schema as with RDF schema, but you can do some typing in your instance data (optional typing of nodes). It's not as "rich" as RDF & certainly, now, undeveloped - RDF - has RDFS & also layers s.a. Owl & also rdf query etc, whereas there's no SOAP query. However SOAP in conjuction with RDF schema allows "meaningful" data aggregation opportunities?

- - SOAP nodes & edges roughly correspond to RDF resources & properties & In RDF 'everthing is a resource' - and in SOAP syntax ELEMENT INFORMATION ITEM elements are used for nodes/edges
- - SOAP simple values are like rdf literals - in the syntax SOAP uses CHARACTER INFORMATION ITEM elements for these
- - Where RDF has b-nodes, SOAP has ref/id pairings - using ATTRIBUTE INFORMATION ITEM attributes to reference them
- - Where RDF kinda has support for collections, SOAP supports ARRAYS

More research required .. e.g. re trying to serialise more arbitrary structures in the SOAP data encoding syntax. For e.g. stuff described in OWL - how would that map to the soap data encoding model easily via XSLT?