Semantically Enabled SOS with Topic Maps

Robert Barta, Thomas Bleier

Austrian Research Centers Seibersdorf
rho@devc.at, Thomas.Bleier@arcs.ac.at

Abstract

Sensor Web Sensor Web Enablement is a consistent standardization effort of the OGC (Open Geo Consortium) to cope with an environment of pervasive sensor networks. Its ultimate goal is to define web-based interfaces to integrate dispersed and technically disparate sensors into one network, facilitating data aggregation for consolidated processing or user consumption.

As the webservice-oriented approach involves the exchange of XML documents encapsulating sensor meta information and data itself, much of the programming effort revolves around creating XML containers, filling them with content as mandated by the underlying XML schemas. If implemented conventionally, sensor meta-data, configuration information and sensor data itself has to be integrated (at least conceptually) before being funneled into the appropriate XML containers. This implementation technique is not only error-prone, but also cumbersome given the large amount of XML schemas to honor.

Here we report about a proof-of-concept implementation to replace a conventionally programmed web service with a semantically-enabled one. For this we abandon any distinction between data and meta data and model the whole information space as one semantic network, specifically a topic map. Such maps consists of nodes (topics) carrying topical information, together with appropriate names and addressing information. But the maps also include node connections (associations) which reflect which and how various topics are connected.

Once pertinent information has been brought into this form, it is susceptible to be queried with a Topic Maps query language (TMQL). That way not only the different storage details can be ignored, TMQL can also directly generate XML content, relieving the implementor from this task.

1. Sensor Web Enablement (SWE)

During the last years the Open Geospatial Consortium (OGC) has developed a suite of standards called *Sensor Web Enablement* (SWE). These deal with providing standardized access to sensors and sensor related services using today's web technologies. The term *sensor* in this case is defined very loose; basically everything that produces some values based on a stimulus is regarded a sensor.

The standard suite includes services for discovering sensors, tasking of sensors, providing alarms and notifications derived from sensor data and, last but not least, for accessing the actual sensor data. The service dedicated to this purpose is called Sensor Observation Service (SOS, [7]). The Sensor Observation Service uses other standards like Observations & Measurements (O&M, [2]) and Sensor Model Language (SensorML, [1]) for the description of the sensor data or sensor metadata.

ISBN 978-0-620-42117-1

The Sensor Observation Service is defined as an implementation of an generic OGC web service. As such it provides a set of methods or operations to its client, that allow them to discover the contents offered by the service and to retrieve data from the service. The basic operations defined for the Sensor Observation Service are:

- GetObservation for retrieving sensor data
- DescribeSensor for retrieving information about a specific sensor
- GetCapabilities for retrieving information about the data offered by the service

Requests of the latter kind are usually quite short

```
<GetCapabilities service="SOS" updateSequence=""
     xmlns="http://www.opengis.net/sos/1.0"
     xmlns:ows="http://www.opengis.net/ows/1.1">
     <ows:AcceptVersions>
         <ows:Version>1.0.0</ows:Version>
     </ows:AcceptVersions>
</GetCapabilities>
```

but the response document is massive:

```
<sos:Capabilities version="1.0.0" updateSequence="2008-07-10T13:03:00+02"
     xsi:schemaLocation="http://www.opengis.net/sos/1.0
     ....
   <ows:ServiceIdentification xmlns:ogc="http://www.opengis.net/ogc"
       xmlns:ows="http://www.opengis.net/ows/1.1">
     <ows:Title>ARC Demo SOS</ows:Title>

  <ows:ServiceType codeSpace="http://opengeospatial.net">
     OGC:SOS
  </ows:ServiceType>
     ...
   </ows:ServiceIdentification>
   <ows:ServiceProvider xmlns:ows="http://www.opengis.net/ows/1.1">
     <ows:ProviderName>Austrian Research Centers</ows:ProviderName>
     ...
   </ows:ServiceProvider>
   <ows:OperationsMetadata xmlns:ogc="http://www.opengis.net/ogc"
     ...
   </ows:OperationsMetadata>
   <sos:Filter_Capabilities xmlns:ogc="http://www.opengis.net/ogc" ...>
     ...
   </sos:Filter_Capabilities>
   <sos:Contents>
     <sos:ObservationOfferingList>
       ...
     </sos:ObservationOfferingList>
   </sos:Contents>
</sos:Capabilities>
```

The O&M specification defines a data model for the description of observations and measurements received from a sensor. In addition, an XML application schema for encoding observations in XML is provided.

The most important properties of an observation according to the O&M specification are:

- the *feature of interest* for defining the target of the observation,
- the *observed property*, which is the phenomenon the observation is based on,
- the *procedure*, a description of the process to generate the result,
- the *sampling time*, which is the actual time where the result applies to the feature of interest, and
- the *actual result*, a value generated by the procedure.

A procedure description characterizes the process that is used to reach from the stimulus of the sensor to the observation value. The Sensor Model Language is one of the specifications in the SWE suite that can be used for that purpose. It not only provides a data model on its own, but also an XML encoding to specify the creation of XML instance documents. Since the SWE suite in general and the Sensor Observation Service in particular are meant to be applied in a very broad range of application scenarios, SensorML contains a very generic set of components and data submodels, which are packaged as *SWE Common*. Based on that, SensorML contains mechanisms to describe various types of processes: physical processes, non-physical (e.g. mathematical) processes, chained processes, etc.

Common elements for all those process definitions are:

- the *input* of the process
- the *output* of the process
- the *parameters* for the process

2. Traditional Implementation

An SOS server will have to wait for incoming HTTP requests. After parsing the XML message and analyzing the request parameters a proper XML response will be constructed before being sent to the requesting client.

The information necessary to draw up the response document ideally resides in a single, consolidated database. Real-life scenarios, however, involve several data sources which typically are only loosely related: configuration data for the service itself, listings of offerings together with some meta data and general properties, administrative and contact information and then also the sensor data itself.

It is the task of the implementor to first conceptually harmonize all sources by creating a data model spanning over these sources and by identifying unique keys for all involved objects to secure a consistent cross-reference between them. The SOS software has then to harvest the necessary information from involved backends in order to organize it into the response XML instance.

To alluviate at least this last step, binding frameworks such as XML beans for Java can be used. Still quite some complexity remains as the following typical workflow for creating a *Capabilities* document shows.

The following lines are needed to create an empty capabilities document and to set the version:

```
CapabilitiesDocument xb_capsDoc = this.loadCapabilitiesSkeleton();
CapabilitiesBaseType xb_capBaseType = xb_capsDoc.getCapabilities();
CapabilitiesDocument xb_capsd   = CapabilitiesDocument.Factory.newInstance();
Capabilities         xb_caps    = xb_capsd.addNewCapabilities();
xb_caps.setVersion(xb_capBaseType.getVersion());
```

To be able to fill the document with sensor offerings, first an empty (sub)document has to be created, that will have to be filled with empty contents, from which we fetch a handle to the list of offerings:

```
ContentsDocument xb_contentDoc    = ContentsDocument.Factory.newInstance();
Contents xb_result                = xb_contentDoc.addNewContents();
ObservationOfferingList xb_ooList =xb_result.addNewObservationOfferingList();
```

Only now we can iterate over the offerings which we get from some backend (below referred to as dataManager):

```
Collection<DataOffering> offerings = dataManager.getOfferings();
for (DataOffering offering: offerings) {
    ObservationOfferingType xb_oo = xb_ooList.addNewObservationOffering();
    xb_oo.setId(offering.getId());
    BoundingShapeType xb_boundedBy = xb_oo.addNewBoundedBy();
    xb_boundedBy.addNewEnvelope();
    xb_boundedBy.setEnvelope(offering.getBoundingBox());
    CodeType xb_name = xb_oo.addNewName();
    xb_name.setStringValue(offering.getDisplayName());
    ...
}
```

The offering list document is then attached to the capabilities document:

```
Contents xb_contents = xb_caps.addNewContents();
xb_contents.set(xb_contentDoc);
```

Similar steps are repeated for all other relevant information, such as filter capabilities or service meta data.

While binding frameworks for XML help to hide XML idiosyncracies and ensure the validity against an underlying XML schema, they impose an intricate workflow onto the developer. And because of the object-oriented nature of these frameworks, the workflow is not linear with the flow of the generated document. Consequently this programming style results in numerous variables, all with their own type, making maintenance a less sought-after activity.

3. Knowledge-Oriented Implementation

Like for the traditional case we assume here that the underlying data has been conceptually and technically consolidated. Different to above, though, all information has been uplifted into a semantic network form so that all concepts, types and all instance data are nodes in a semantic network.

The application, in our case the SOS server, would approach such a network via an API, or better with a fully-fledged query language. A naïve approach would use the query language to extract relevant content to be injected into an XML structure as before. A more sophisticated alternative is to let the query language do all the XML weight-lifting.

3.1 Topic Maps

Topic Maps (TM, [4]) is a knowledge representation technology quite comparable to the more main-stream RDF/S framework [6]. While in the latter all information is couched in the form of triples (*subject*, *predicate*, *object*), TM basic concepts are designed in a more anthropomorphic way:

Topics represent subjects, which can be *anything*, physical or not. To further knowledge aggregation, topics can be enriched by various identifiers. In the case of objects which reside at certain network locations, such identifiers will naturally be URLs. For a given SOS deployment its endpoint can be used for identification:

```
demo-sos isa SOS-deployment
 = http://env05.arcs.ac.at/SOSsrv/
 ! ARCS Demo SOS
```

In the notation above (AsTMa=, [8]) this is symbolized by prefixing a IRI with `=`. The local identifier `demo-sos` is only local within the map.

As also shown in the example above, topics can have *types*, i.e. are instances of a class. That itself is just another topics, to be defined in the map or by some environment ontology. And topics can also have a number of names attached, in our case only one (signalled with `!`). Names can also be typed to allow to use different names in different contexts, such as in

```
arcs isa research-center
 ! acronym: ARCS
 ! Austrian Research Centers
```

Relationships between topics are expressed via *associations*, whereby every involved topic is a player of a certain role. The fragment

```
provisioning (provider: arcs, service: demo-sos)
```

means that *arcs as provider provisions a service* `demo-sos`. Obviously the whole association itself is also of a certain type. The roles themselves (provider, service) are again topics to be detailed somewhere to the extent necessary.

3.2 TM Query Language (TMQL)

Instead of using an API into a consolidated topic map, we leverage TMQL [3] because it not only can detect and extract the required content; it can also organize it into XML.

Like any other query language TMQL has two objectives:

- locate and detect certain information in the queried TM database
- generate output based on the detected information

One familiar type of output is tabular and it can be requested using a SELECT-ish syntax:

```
   select $p / acronym, $s =
    where
       provisioning (provider: $p, service: $s)
```

A query processor will first try to find all associations which follow the pattern above, i.e. have the required association type and the given roles. Once such an association is found, the variables `$p` and `$s` will be bound to the players in the captured association.

On the outgoing side, `$p` and `$s` will be used in the SELECT clause to evaluate path expressions. The `$p / acronym` would evaluate to all acronyms of the thing `$p` is bound to. The expression `$s =` would return all subject addresses of the thing bound to `$s`. The overall result would be:

"ARCS"	“http://env05.arcs.ac.at/SOSsrv/”

The query language is flexible enough to also generate XML output, not as string, but in an internal representation, say as DOM. For this we have to switch into *FLWR* style (*for*, *let*, *where*, *return*):

```
<services>{
   where
      provisioning (provider: $p, service: $s)
   return
      <service href="{$s =}">{$p / name}</service>
}</services>
```

While we have used the same WHERE clause as above and also bind the same variables to the same topics, all this is now embedded in an XML template structure. The expected output would then be

```
<services>
   <service href="http://env05.arcs.ac.at/SOSsrv/">ARCS</service>
</services>
```

3.3 SOS Response Generation

Equipped with a topic map we can now reimplement the SOS responder, whereby the plan is to burden as much as possible onto TMQL's feature to generate XML content.

Once the responder has determined which kind of request (GetCapabilities, DescribeSensor, etc.) has to be satisfied, it will launch a single TMQL query against the underlying topic map. For a Capabilities request this looks like this (slightly trivialized, and using Perl):

```
use TM;
my $tm    = ....;

use TM::QL;
my $query = new TM::QL (qq{return <sos:Capabilities .../>});
my $res   = $query->eval ($tm);
```

In a first step the topic map is fetch from one (or more) backends. Then the query is compiled, so that it can be evaluated directly afterwards. The only parameter for the evaluation is the map object.

On successful termination the result is an XML structure which can be serialized into a string:

```
print $res->toString;
```

Inside the XML query string, it is trivial to embed topic map information, such as the name of the service provider:

```
    <ows:ProviderName>{$p / acronym || $p / name}</ows:ProviderName>
```

TMQL expressions can be used to deal with incomplete or highly data. Above, for instance, we looked first for provider acronyms. If there were none, the query would fall back to the full name of the provider (`||` is the *shortcircuit or* in TMQL).

Naturally TMQL supports loops over repetitive items, so it is straightforward to include, say, a list of offerings:

```
   <ows:Parameter name="offering">
      <ows:AllowedValues>{

    for $o in // offering return
        <ows:Value>{$o !}</ows:Value>

      }</ows:AllowedValues>
    </ows:Parameter>
```

The subexpression // offering will compute all instances of offering in the map, interestingly not only direct ones, but also instances along any subclass hierarchy. If we were happy to get returned short, internal identifiers for the offering topics, then the expression $o ! would just give us one.

4. Conclusions

The prototype was built merely with open source components. Mason (`HTML::Mason` on CPAN) is a conventional, but robust web component framework. With it the request is parsed and it also dispatches the appropriate component to serve the request. Perl TM (`TM` on CPAN) is used for both, the map management and the query processor. Any logging is done with Log4Perl (`Log::Log4Perl` on CPAN).

From an architectural viewpoint there are several interesting points:

- The fact that all objects are addressed the same way, namely as topics, has reduced the development effort. The TMQL query expressions are exactly the boundary to be defined how content is to be packaged into XML.
- All information, be it configuration or operational data has been brought into one paradigm, that of Topic Maps. In a simple implementation such a map can be completely human-authored. In larger deployments some data will be blended in from external databases. This is achievable with wrappers which provide a *topic-mappish* view on the content. This is the only place where such mapping and the provenance of the data becomes visible. Should the implementation strategy change over time, the topic map

abstraction itself is not affected.

- Since TMs understand type hierarchies queries along this axis are much more robust against reorganisation of the underlying database.

While maybe acceptable for a prototype, runtime performance is problematic, especially when it comes to query larger maps. This is a problem area currently addressed, both from a theoretic angle as well as implementation-wise.

On a notational frontier we consider to extend the Topic Maps notation to express time series data and geospatial phenomena more naturally (similar to OWL Time[5]).

Also the virtualisation infrastructure is quite immature, making it necessary to write wrappers of existing data sources manually, mostly from scratch. Here more work has to be invested to find a more consistent methodology.

Bibliography

[1]. M. Botts and A. Robin. Sensor Model Language (SensorML), Open Geospatial Consortium Inc., OGC 07-000, 2007.

[2]. S. Cox. Observations and Measurements, Part 1 - Observation Schema, Open Geospatial Consortium Inc., OGC 07-022r1, 2007.

[3]. L. M. Garshol and R. Barta. ISO 18048: Topic Maps Query Language (TMQL) - Committee draft, 2008.

[4]. L. M. Garshol and G. Moore. Topic Maps - Data Model, ISO 13250, iso/iec jtc1/sc34, information technology, 2006.

[5]. J. R. Hobbs and F. Pan. Time ontology in OWL, W3C working draft 27 september 2006

[6]. G. Klyne and J. J. Carroll. Resource Description Framework (RDF): Concepts and Abstract Syntax, W3C Recommendation 10 February 2004. W3C, 1993.

[7]. A. Na and M. Priest. Sensor Observation Service, Open Geospatial Consortium Inc., OGC 06-009r6, 2007.

[8]. R. Barta. AsTMa= language definition. 2004. Technical Report, http://astma.it.bond.edu.au/astma=-spec-xtm.dbk.