

A SOAP Based Metadata Service - IWICOS Broker

Haajanen, Jyrki, M.S, VTT Information Technology, Information Systems, Tekniikantie 4, P.O.Box 1201

FIN-02044 VTT, Finland

E-mail: jyrki.haajanen@vtt.fi, Tel: 358-9-456-6028, Fax: 358-9-456-6027

Berglund, Robin, M.S, VTT Information Technology, Information Systems, Tekniikantie 4, P.O.Box 1201

FIN-02044 VTT, Finland

E-mail: robin.berglund@vtt.fi, Tel: 358-9-456-6018, Fax: 358-9-456-6027

Abstract

Simple Object Access Protocol (SOAP) is an emerging technology supported by major actors in the Internet scene. When SOAP is attached with the HTTP the organizational firewalls become virtually transparent for the defined SOAP services. This leads to unforeseen possibilities in cross-organizational interoperability. This paper describes an SOAP based implementation of a cross-organizational interoperable GIS product metadata service called IWICOS Broker.

Contents

1	THE IWICOS PROJECT	1
2	ARCHITECTURE AND CONCEPTS	1
2.1	THE IWICOS SERVICE CHAIN	1
2.2	THE METADATA AND MINIMAL SEARCHABLE SET	1
2.3	THE ARCHITECTURAL COMPONENTS.....	2
2.4	HIGH-LEVEL COMMUNICATION SEQUENCES	3
3	IWICOS BROKER	4
3.1	PRODUCER SERVER - BROKER INTERFACE	5
3.2	FACADE - BROKER INTERFACE	7
3.3	INTERNAL COMPONENTS OF THE BROKER.....	7
4	SOAP	8
4.1	SOAP IN IWICOS BROKER.....	9
5	CONCLUSIONS	11
	REFERENCES	11

1 The IWICOS Project

The IWICOS project is targeted to research evaluate and demonstrate the technologies and approaches required for an interoperable weather, ice, and ocean data service. The aim for the demonstration is to plan and implement a service chain for interoperable cross-organization GIS data production and delivery for seafarers, and the use of such data. The demonstration will be given with a set of prototype implementations, in two phases. The first phase, called the Baseline System, is to be accomplished in summer 2001 and the second phase, called the Extended System will be performed in the year 2002. The project is funded by the European Commissions IST Programme.

The project consortium consists of several governmental organizations from the Nordic countries. The consortium members are: Danish Meteorological Institute, Danish Technical University, Finnish Institute of Marine Research, Icelandic Meteorological Office, Nansen Environmental Research Centre from Norway, and Technical Research Centre of Finland (VTT).

The projects target domain, meteorology, sea ice, and oceanographic information, poses a large set of various requirements that the intended service has to fulfill. Meteorological data is delivered in large files and has a high frequency of updates - many forecasts per a geographical area per day. The ice information is typically expressed by compiled maps or satellite images; the update frequency for them is up to once per day. The oceanographic information has the largest scale of attributes, it may represent anything from sea currents to the sea bed formation details.

2 Architecture and Concepts

The architecture for the IWICOS system is based on four different subsystems: Producer Server, Broker, Facade, and Client Software. The Facade and the Client Software together form the End-user System. Data providers implement the Producer Server and integrate it to their existing production process. Correspondingly each Service Provider implements an End-user System.

2.1 The IWICOS Service Chain

The IWICOS subsystems form an interoperable service chain for producers and users of the met, ice, and ocean data (Figure 1).

The first element of the IWICOS service chain is the Producer Server, that produces the GIS data products, and XML based metadata descriptions of their contents.

The next element in the service chain, the Broker, has a key role in implementing the interoperability. The Producer Servers inform the Broker when they have new data available. The Broker then collects the metadata from them to a single central storage. The End-User Systems can query for products that satisfy the customer needs using the query interface that the Broker provides for them.

As mentioned above the End-User System consist of the Facade and the Client Software. The Facades communicate with the Broker over a high-bandwidth Internet connection. The various service providers have their own Facades. These are connected to the Client Software implementations on ships. The communication line to the ships is likely to be narrow band. In this sense the Facade will act as a Proxy, that will assist in filtering out the data that is not critical to the end-user on the sea. The Facade and Client Software will co-operate to implement relaying between a ground station and a ship out on the sea.

2.2 The Metadata and Minimal Searchable Set

The IWICOS metadata was specifically defined for the purposes of the project - although there are metadata standards for GIS services, such as [OGI99], we found them to be too large to implement within the scope of a research project demonstration. Since we would only have been able to implement only a subset of such a

standard (and thus not be standard compliant) we decided to define a limited set of metadata for the IWICOS. The metadata definition is given as an XML Schema and the metadata files are XML documents instantiated from this schema.

As the anticipated queries would only be based on a small subset of the metadata attributes, we introduced the Minimal Searchable Set (MSS) of metadata elements enhanced with some Broker implementation dependent fields, such as a unique product identifier the for products in the Brokers internal storage.

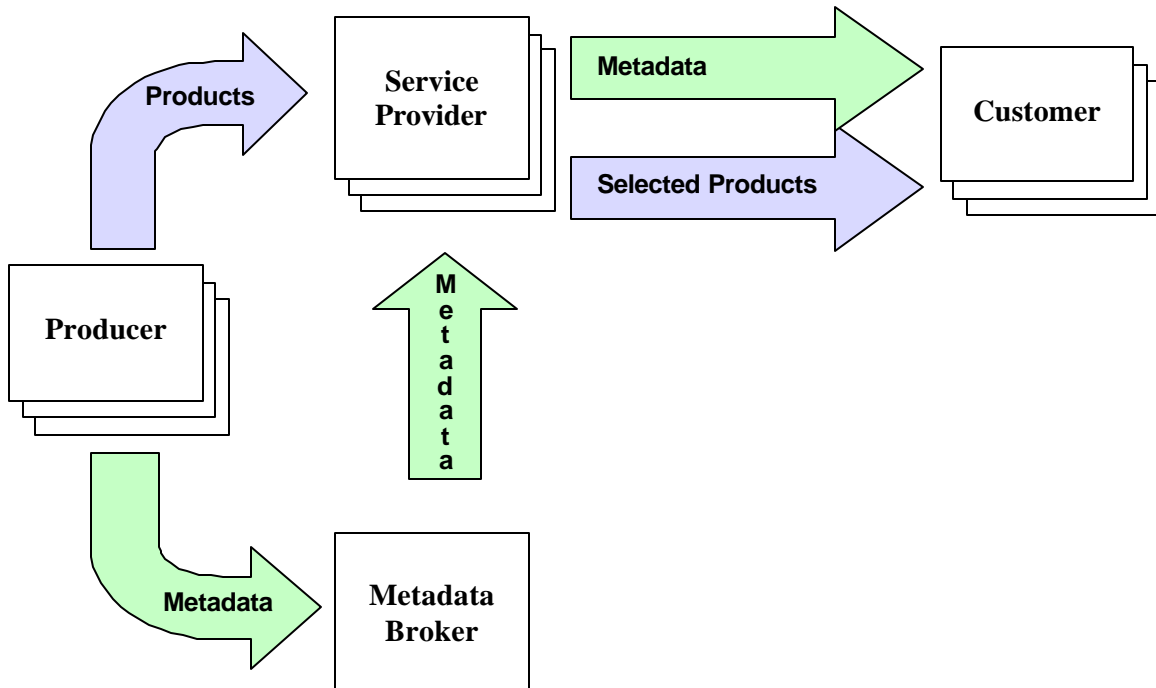


Figure 1: The IWICOS Service Chain.

2.3 The Architectural Components

The Producer Server integrates the IWICOS system to the producer's existing production process. It communicates with the Broker using specific small stub programs designed for the messaging between these components.

The Producer Server consists of two parts (Figure 2):

1. Active part, that informs the Broker of the events, such as a new product becoming available or an old product getting outdated, that occur in the production process.
2. Passive part, that is basically a web server where the active part places its output for the Broker (the metadata) and the Facades (the product data) to retrieve.

The set of data formats for the communication between the Producer Servers and the Facades was limited to Binary Sequential Files (BSQ), GRIB (GRIdded Binary), Shapefile, and XML for reducing the complexity of the implementation. Within the End-User Systems (Facade and Client Software pair) there is no such limitations. The Facades can use the base types listed above to generate products in any format. Basically the End-User System is a black box - the subsystems outside should and shall not be interested in what happens there.

The Broker provides two interfaces for metadata operations. One is intended for the Producer Servers for managing the Broker's metadata content. The other one is the query interface for the End-User Systems through which they can access the Broker's metadata content.

The Facade implementations can vary a lot. For a simple web-browser based client it will be closely integrated to the Client Software and it is hard to say where one begins and the other one ends. On the other hand it may be very complicated element containing reasoning of user needs based on a profile, product generation based on the products provided by the Producer Servers, and filtering of unnecessary components of products (e.g. layers with unnecessary data content).

The Client Software is intended for presenting the acquired data and the implementations have a range from thin to thick clients. All types of clients (thin, balanced and thick) will probably be tested in the demonstrations during the project.

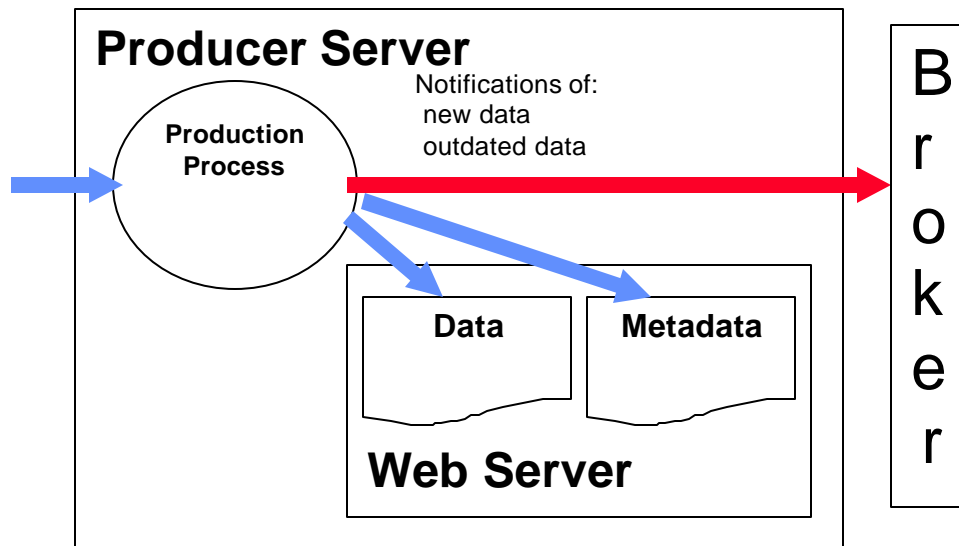


Figure 2: The IWICOS Producer Server Components and Internal Data Flow.

2.4 High-level Communication Sequences

The IWICOS Architecture is illustrated by the four following communication scenarios between the various subsystems: Producer Server - Broker, Facade - Broker, Facade - Producer Server, and Facade - Client Software. These are explained in more detail below. Figure 3 shows the communication sequences in these scenarios.

Producer Server - Broker

In this scenario the Producer Server manages the Broker metadata content with the following operations: *newProduct*, *outdatedProduct*, and *productList*. The operations are used for informing the Broker of new and outdated products, and acquiring of a list of the producer's products that have metadata stored to Broker.

Facade - Broker

In this scenario the Facades execute queries on metadata at the Broker to gain a list of products that might be applicable for user needs. This interface contains a single operation called *query*. However, it is more complicated than the previous scenario hence the parameter and reply contents is more dynamic. We will get back on this issue in the chapter describing the Broker implementation details.

Facade - Producer Server

In this scenario the Facades acquire the actual product data based on the metadata retrieved in the previous scenario. The product data at the passive (web-server) part of the Producer Server is accessed using the HTTP-protocol and thus is not very complicated to implement.

Facade - Client Software

This scenario is actually a black-box one - the decision of what protocols and other means of communications to use here is left for the designer of the Client Software - Facade pair. In the scope of the IWICOS System it is enough for us to know that such implementation is present.

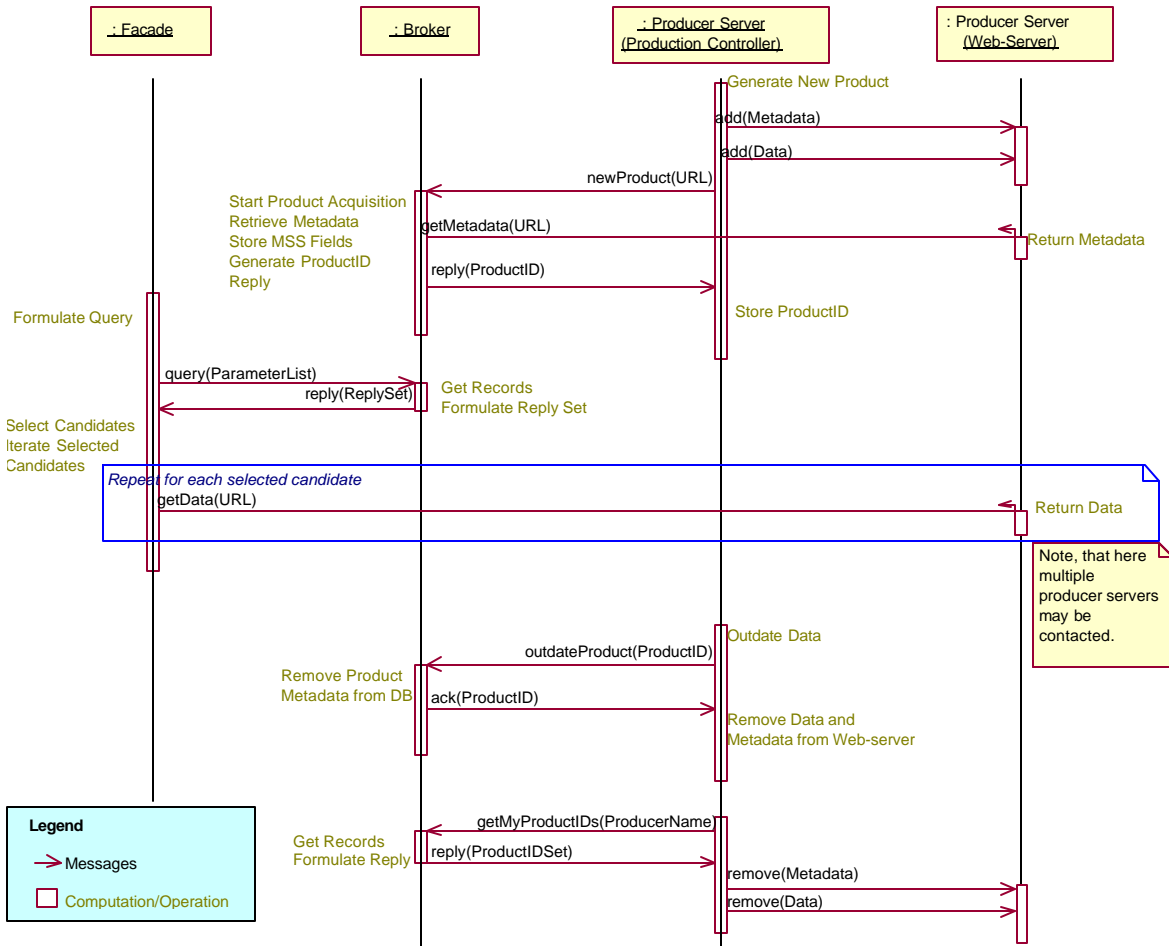


Figure 3: IWICOS High-Level Communications Scenarios.

3 IWICOS Broker

The Broker is based on freely available elements, Apache Tomcat web-server [Apa01c], Apache SOAP 2.0 implementation [Apa01b], MySQL database [MyS01], and Linux OS (RedHat). The service programs are written in Java.

The communication is implemented with the Remote Procedure Calls (RPC) implemented over the Simple Object Access Protocol (SOAP) [Apa01b, Box01, Gla01a, W3C00]. The RPC uses a small program called a *stub* in both ends of the communication, that will marshal the required parameters and return values of the call. The arguments marshaling in the Broker is mainly based on the automatic Java to XML type conversion provided in the SOAP. The only exception to this is in the query operation where an XML document is passed as an argument and an explicit marshaling routine is applied on the parameter to avoid confusion in the SOAP XML message structure and the message content. Example of stubs is shown in Figure 4.

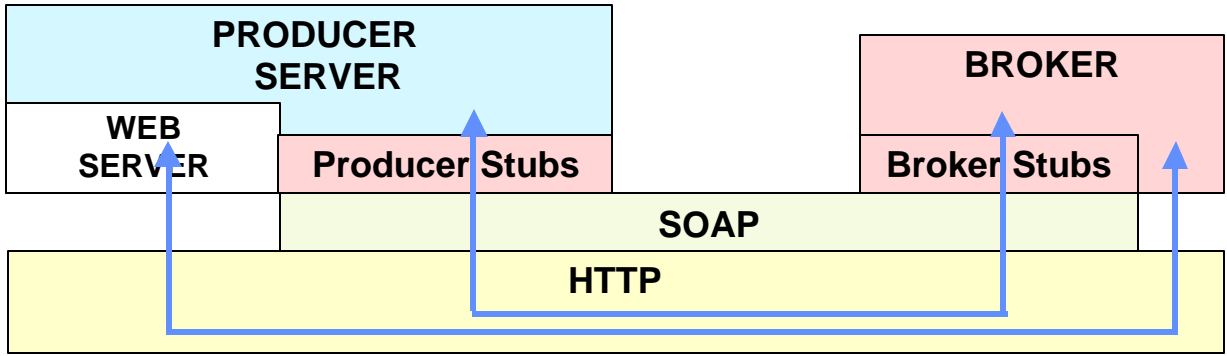


Figure 4: Stub Example - The IWICOS Producer Server and Broker Stubs.

The Broker has two external interfaces, corresponding to the Producer Server - Broker, and Facade - Broker communication scenarios presented above. The communication over these interfaces occurs as shown in Figure 5.

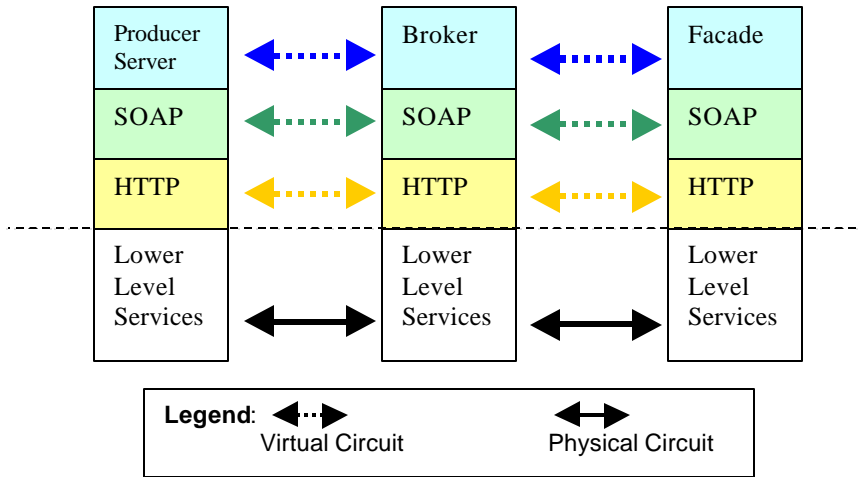


Figure 5: The SOAP Based Communication in the Broker Interfaces.

3.1 Producer Server - Broker Interface

The operations and their parameters and return values for the Producer Server - Broker Interface are explained in Table 1.

In the *newProduct* operation the Broker validates the *metaURI* parameter, retrieves the IWICOS metadata instance document located there and parses its contents to the internal database. If the operation was successful it replies with the new *productID*. The parser is implemented based on the Xerces [Apa01e] tool.

The *outdatedProduct* operation is performed with first checking that the *productName* and the *productID* given are valid and matching. Then the corresponding product is located in the Broker internal structures and references to it are removed.

Operation	Parameters	Return Value
NewProduct	<ul style="list-style-type: none"> ProducerName, a String holding the name of the producer MetaURI, a String holding the Universal Resource Identifier for the metadata. 	productID , a positive integer indicating the ID of the new product, if the operation was successful. Negative values indicate errors.
outdatedProduct	<ul style="list-style-type: none"> ProducerName, a String holding the name of the producer ProductID, an integer specifying the ID of the outdated product. 	productID , a positive integer indicating the ID of the removed product, if the operation was successful. Negative values indicate errors.
productList	ProducerName , a String holding the name of the producer	A String containing a comma separated list of the product ID's for the specified producer and their respective metadata locations.

Table 1: IWICOS Producer Server - Broker Interface Operations.

The productList operation starts with the validation of the given producerName. Then the corresponding elements are retrieved from the database and the reply is generated.

The IWICOS metadata sample is shown in Figure 6. It describes the properties of the product. It first lists the spatio-temporal properties of the product, and gives outlined processing information. This is followed by the element *data* defining the format of the product (BSQ, GRIB, Shapefile or XML), and finally the projection is defined.

```

<?xml version="1.0" encoding="UTF-8"?>
<Product xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance" Category="Sealce"
  ProductType="Analysis" HeaderVersion="1.0" AccessConstraints="Restricted Use">
  <Extent>
    <UL SpatialID="1" Lon="-49 30 02.35" Lat="61 56 45.95"/>
    <UR SpatialID="2" Lon="-39 20 38.61" Lat="63 06 07.36"/>
    <LR SpatialID="3" Lon="-38 03 07.28" Lat="58 16 51.88"/>
    <LL SpatialID="4" Lon="-47 00 55.43" Lat="57 29 35.57"/>
  </Extent>
  <ReferenceTime>
    <Instant Time="2001-04-27T09:20:00.000"/>
  </ReferenceTime>
  <ValidTime>
    <Instant Time="2001-04-27T09:20:00.000"/>
  </ValidTime>
  <ScheduledTime NextProduct="2001-04-28T09:12:00.000"
    Comment="Planned: 2001-04-28T09:17:00.000 ; 2001-04-29T09:12:00.000"/>
  <ProcessInfo Generator="DMI Ice Chart Analyst"
    ProducerLocator="Some URI/URL"/>
  <Data Availability="true" FileLocator="Some URI/URL/20010427092000IA.zip"
    FileSizeInBytes="144000" Compression="zip">
    <Shapefile>
      <ParameterEntry Parameter="IceConcentration" Units="tenths" Shape="polygon"/>
      <ParameterEntry Parameter="IceType" Units="class" Shape="polygon"/>
    </Shapefile>
  </Data>
  <Projection Datum="WGS-84" Spheroid="WGS-84">
    <ProjectionLambertConformalConic UnitsPlanar="meter">
      <ParametersLCC LatStdP1="66" LatStdP2="76" LonCtrMer="-39" LatOrg="55"
        FalseE="1500000" FalseN="0"/>
    </ProjectionLambertConformalConic>
  </Projection>
</Product>

```

Figure 6: The IWICOS Metadata Sample (Shapefile data).

The Broker automatically parses the contents of the metadata and extracts all elements applicable to the MSS format to its database for later queries. The MSS format is presented in the Figure 7, (note that the element *ProductID* cannot be determined when parsing the metadata since it is internal to the Broker implementation, it will, however, be updated automatically when the DB insert is done.)

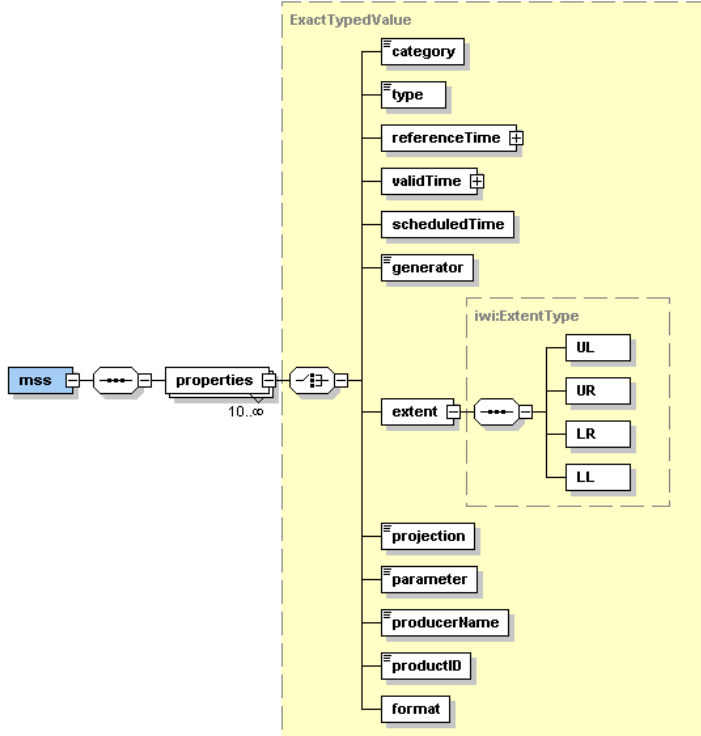


Figure 7: The Minimal Searchable Set at IWICOS Broker.

3.2 Facade - Broker Interface

The Facade - Broker Interface uses XML based presentations of the queries and the replies. It consists of a single Remote Procedure Call, the query routine, which takes an instance of the query XML document as a parameter (see Figure 8). It replies to the call with an instance of the reply XML document (see Figure 9). Here we added an explicit marshaling and unmarshaling routines that are applied on the ends of communication, to avoid the confusion of the content of the SOAP message and the XML content to be delivered. A more elegant way of solving this with the SOAP primitives will be studied for the Extended IWICOS System.

The query implementation is quite similar to handling of expressions in a programming language compiler design. Examples can be found in literature (e.g. [ASU86]). The query can be posed in two modes: 'brief' and 'full', in the latter the main body of the response contains the actual product metadata, and in the previous only the stored MSS fields are returned.

3.3 Internal Components of the Broker

The Broker consists of the following components (see Figure 10):

- The Database, a relational MySQL based DB that holds the MSS elements for each inserted product.
- The Metadata Parser, an Xerces based parsing solution for identifying the MSS elements from the metadata and inserting the to the DB. The parser actively retrieves the metadata for parsing from the Producer Servers web-server part based on the *metaURI* given as parameter.

- The Query Engine, a component that parses the XML based query definition and performs a corresponding SQL query on the DB, and formulates an XML based reply from the results of the SQL query. The query parsing is implemented with Extendable Stylesheet Language Transformation (XSLT) -language [W3C01c] which is processed with the Xalan XSL processor [Apa01d]. This solution is applicable hence the processing is serial.
- The Service Stubs, these are Java programs that activate the services defined in the interfaces.

```

<?xml version="1.0" encoding="UTF-8"?>
<query xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance" xmlns:mss="mss" xmlns:iwi="
IWICOSSchema" xsi:noNamespaceSchemaLocation="query.xsd" mode="full">
  <expressionOperatorExpression operator="AND">
    <expression1>
      <rangeExpression>
        <lo limit="greater">
          <mss:validTime>
            <iwi:Instant Time="2001-01-01T00:00:00.000"/>
          </mss:validTime>
        </lo>
        <hi limit="lessOrEqual">
          <mss:validTime>
            <iwi:Instant Time="2001-12-00T00:00:00.000"/>
          </mss:validTime>
        </hi>
      </rangeExpression>
    </expression1>
    <expression2>
      <rangeExpression>
        <lo limit="greater">
          <mss:location Lat="00" Lon="00"/>
        </lo>
        <hi limit="less">
          <mss:location Lat="90" Lon="90"/>
        </hi>
      </rangeExpression>
    </expression2>
  </expressionOperatorExpression >
</query>

```

Figure 8: A Sample Query for the Facade - Broker Interface.

4 SOAP

The Simple Object Access Protocol (SOAP) is an open standard that supports the interoperability of autonomous systems. Since it can be run over the Hypertext Transfer Protocol (HTTP), it helps to resolve the communication problems that occur when the applications have to operate through the firewalls of different organizations. Furthermore, SOAP supports RPC for invoking services on the service. SOAP messages can contain data packed in XML format, so they provide an ideal platform for implementing a tailored messaging service. Due to the XML approach the SOAP messages are easy to understand when compared to their binary correspondents such as CORBA, DCOM and RMI. The character-based nature of the messages introduces the requirement for encoding and decoding at the ends of communication, however this is relatively light task when compared to the cost of the transmission itself.

The SOAP documentation has been lagging behind the rapid paced development of the implementations of the protocol, luckily, the situation is getting better all the time.

The SOAP popularity seems to be growing - at least based on a rudimentary monthly statistics collected with the Northern Light search engine (Figure 11). The results were acquired with performing a search for phrase "Simple Object Access Protocol" with the time range set to one month over the period from May 2000 to May 2001. The results are used just to get a trend of what is happening on the scene - a detailed analysis of resulting references is not performed.

```

<?xml version="1.0" encoding="UTF-8"?>
<reply xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance" xmlns:mss="\mss" xmlns:iwi="IWICOSchema"
xsi:noNamespaceSchemaLocation="reply.xsd" query="1" products="3">
  <brief>
    <metadata >
      <mss:category>SatImg</mss:category>
    </metadata >
    <metadata >
      <mss:type>Forecast</mss:type>
    </metadata >
    <metadata >
      <mss:referenceTime>
        <iwi:Instant TemporalID="" Time="2001-05-31T12:00:00"/>
      </mss:referenceTime>
    </metadata >
    <metadata >
      <mss:validTime>
        <iwi:Instant TemporalID="" Time="2001-05-31T12:00:00"/>
      </mss:validTime>
    </metadata >
    <metadata >
      <mss:scheduledTime>
        <iwi:NextProduct>2001-05-31T12:00:00</iwi:NextProduct>
      </mss:scheduledTime>
    </metadata >
    <metadata >
      <mss:generator>Some Nice Generator</mss:generator>
    </metadata >
    <metadata >
      <mss:extent>
        <iwi:UL SpatialID="" Lon="20" Lat="20"/>
        <iwi:UR SpatialID="" Lon="30" Lat="20"/>
        <iwi:LR SpatialID="" Lon="20" Lat="10"/>
        <iwi:LL SpatialID="" Lon="30" Lat="10"/>
      </mss:extent>
    </metadata >
    <metadata >
      <mss:projection>Mercator</mss:projection>
    </metadata >
    <metadata >
      <mss:parameter>IceType</mss:parameter>
    </metadata >
    <metadata >
      <mss:producerName>IMO</mss:producerName>
    </metadata >
    <metadata >
      <mss:productID>1</mss:productID>
    </metadata >
  </brief>
</reply>

```

Figure 9: A Reply Sample for Broker - Facade Interface.

4.1 SOAP in IWICOS Broker

Our approach to implementing a interoperable metadata services introduces the use of SOAP, SOAP RPC and extensive use of other XML related technologies where applicable.

The SOAP was selected to be the basis for our implementation after we had already designed the system architecture - virtually the architecture would be quite the same although we had known this in advance.

We found that at the time the implementation began in February 2001, there was not much material available for a SOAP developer, which seems to be backed up by the results on the new SOAP web references.

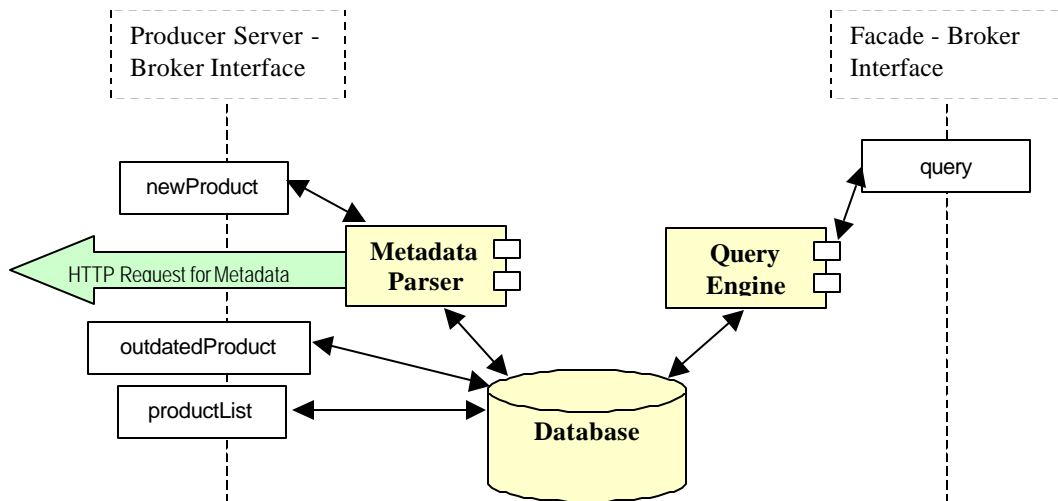


Figure 10: The Broker Internal Structure.

The hardest part in implementing the system was to get the initial configuration (Apache Tomcat Server, SOAP 2.0, MySQL, and Xerces Components) to work, although the article series by Graham Glass [Gla00a, Gla00b, Gla01a, Gla01b] supports setting up the infrastructure. After we solved the details of configuration the actual development work appeared to be relatively easy. The productivity was high when compared to our earlier experiences with CORBA. This was mainly due to the self-explanatory nature of the SOAP messages that made the debugging easy.

Altogether the Broker implementation required approximately four man months with the study of the SOAP and the configuration details included, excluding the System architecture design that was done earlier and took approximately same effort. The metadata definitions were produced in a task force formed by Morten Lind from DMI, Leif Toudal Petersen from DTU, and Jyrki Haajanen from VTT.

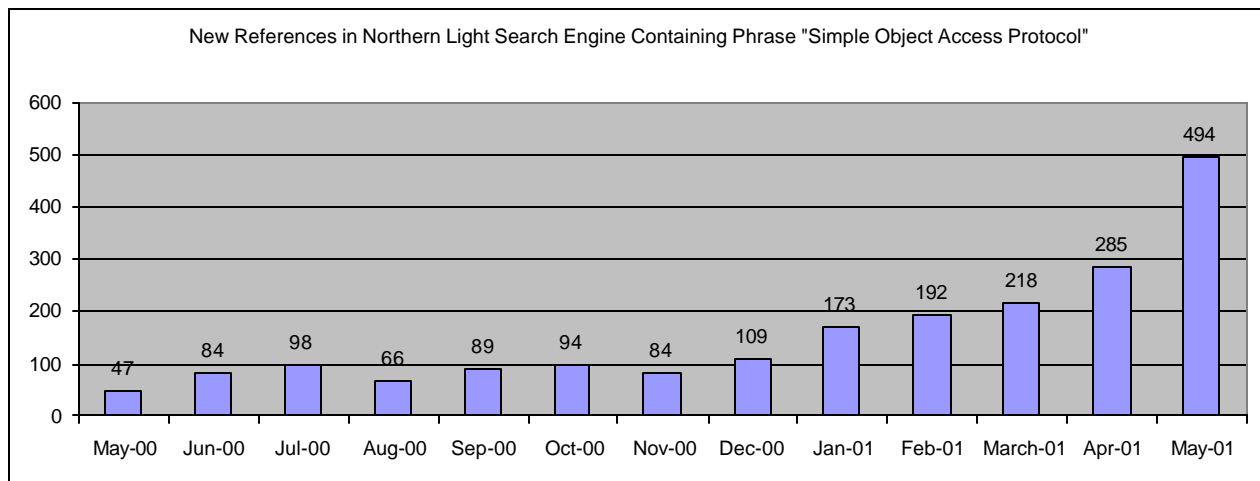


Figure 11: Number of Monthly New Northern Light References to "Simple Object Access Protocol".

5 Conclusions

SOAP is an emerging technology with an impressive growth rate of the user group. Our experience shows that it can be used to implement a interoperable metadata service implementation within a reasonable time and effort.

Best features in SOAP are the clarity of the messages, the firewall passing capability, and easy RPC support. Although we experienced some problems with the proxy configuration in one organization, generally SOAP worked fine.

SOAP provides a very powerful solution especially in addition to other XML related tools, such as XML, XML Schema, and XSLT. The full potential of the SOAP related technologies such as the Universal Description, Discovery, and Integration (UDDI) [UDD01], and Web Services Description Language (WSDL) [CCM01] is a thing that we cannot predict yet, but if they prove to be nearly as successful as the SOAP we will see an imminent change in the way we think about the Internet and cross-organizational interoperability.

References

- [Apa01a] The Apache Software Foundation, "Apache HTTP Server Project", <http://httpd.apache.org/>, 2001.
- [Apa01b] The Apache Software Foundation, "Apache SOAP", <http://xml.apache.org/soap/>, 2001.
- [Apa01c] The Apache Software Foundation, "Jakarta Tomcat", <http://jakarta.apache.org/tomcat/>, 2001.
- [Apa01d] The Apache Software Foundation, "Xalan-Java version 2.1.0", <http://xml.apache.org/xalan-j/>, 2001.
- [Apa01e] The Apache Software Foundation, "Xerces Java Parser Readme", <http://xml.apache.org/xerces-j/>, 2001.
- [ASU86] Aho, Sethi, Ullman: "Compilers, principles, techniques, and tools", Addison-Wesley, Reading, 1986
- [Box01] Don Box, "A Young Person's Guide to The Simple Object Access Protocol: SOAP Increases Interoperability Across Platforms and Languages", <http://msdn.microsoft.com/library/periodic/period00/soap0300.htm>, 2001.
- [CCM01] Erik Christensen, Francisco Curbera, Greg Meredith, Sanjiva Weerawarana, "Web Services Description Language (WSDL) 1.1", <http://msdn.microsoft.com/xml/general/wSDL.asp>, 2001.
- [Gla00a] Graham Glass, "The Web services (r)evolution, Part 1 - Applying Web services to applications", <http://www-106.ibm.com/developerworks/library/ws-peer1/?dwzone=ws>, 2000.
- [Gla00b] Graham Glass, "The Web services (r)evolution, Part 2 - Hello world, Web service-style", <http://www-106.ibm.com/developerworks/library/ws-peer2/?dwzone=ws>, 2000.
- [Gla01a] Graham Glass, "The Web services (r)evolution, Part 3 - How SOAP works", <http://www-106.ibm.com/developerworks/library/ws-peer3/?dwzone=ws>, 2001.

- [Gla01b] Graham Glass, "The Web services (r)evolution, Part 4 - Web Services Description Language (WSDL)", <http://www-106.ibm.com/developerworks/library/ws-peer4/?dwzone=ws>, 2001.
- [MyS01] MySQL Company, "MySQL Website", <http://www.mysql.com/>, 2001.
- [OGI99] Open GIS Consortium, "OpenGIS - Catalog Interface Implementation Specification (Version 1.0)", 1999.
- [UDD01] UDDI Organization, "UDDI Organization Website", <http://www.uddi.org/>, 2001.
- [W3C00] World Wide Web Consortium, "Simple Object Access Protocol (SOAP) 1.1 - W3C Note 08 May 2000", <http://www.w3.org/TR/2000/NOTE-SOAP-20000508/>, 2001.
- [W3C01a] World Wide Web Consortium, "Extensible Markup Language (XML)", <http://www.w3.org/XML/>, 2001.
- [W3C01b] World Wide Web Consortium, "XML Schema", <http://www.w3.org/XML/Schema>, 2001.
- [W3C01c] World Wide Web Consortium, "Extensible Stylesheet Language (XSL)", <http://www.w3.org/Style/XSL/>, 2001.