Intercloud Directory and Exchange Protocol Detail using XMPP and RDF

David Bernstein
Huawei Technologies, USA
dbernstein@huawei.com

Deepak Vij
Cloud Strategy Partners, LLC
deepak@cloudstrategypartners.com

As submitted to

CBSA: Cloud-based Services and Applications Workshop of:

Co-Located With:

Pre-acceptance draft, not for distribution

Intercloud Directory and Exchange Protocol Detail using XMPP and RDF

David Bernstein
Huawei Technologies, USA
dbernstein@huawei.com

Deepak Vij
Cloud Strategy Partners, LLC
deepak@cloudstrategypartners.com

Abstract

Working groups have proposed building a layered set of protocols to solve the Cloud Computing interoperability challenge called "Intercloud Protocols". Instead of each cloud provider establishing connectivity with another cloud provider in a Point-to-Point manner resulting in the n^2 complexity problem, Intercloud Directories and Exchanges will act as mediators for enabling connectivity and collaboration among disparate cloud providers. Point to Point protocols such as HTTP are not suitable beyond 1-to-1 models, therefore the discussions around many-to-many mechanisms have been proposed, including XMPP. This paper details the use of an XMPP mechanism for such mediation. On top of that, for the federation of the resources themselves, we define a resources catalog approach, using the Semantic Web Resource Definition Framework (RDF) along with a common Ontology of Cloud Computing Resources to work across a variety of heterogeneous cloud providers.

1. Introduction

Cloud Computing has a well accepted terminology [1], and Use Cases and Scenarios for Cloud IaaS and PaaS interoperability [2][3] have been detailed in the literature along with the challenges around actually implementing standards-based Intercloud federation and hybrid clouds. Work detailing high level architectures for Intercloud interoperability were proposed next [4][5]. More recently, specific implementation approaches for Intercloud protocols [6][7] have been proposed, including specifically Extensible Messaging and Presence Protocol (XMPP) [8][9] for transport, and using Semantic Web [10] techniques such as Resource Description Framework (RDF) [11] to specify resources.

Following that work, we will go about outlining detailed approaches for these Intercloud protocols; first a detailed analysis on the feasibility of XMPP as a control plane operations for Intercloud, and second how Cloud Computing resources can be described, cataloged, and mediated using Semantic Web Ontologies, implemented using RDF techniques.

2. Intercloud Topology

Cloud instances must be able to dialog with each other. One cloud must be able to find one or more other clouds, which for a particular interoperability scenario is ready, willing, and able to accept an interoperability transaction with and furthermore, exchanging whatever subscription or usage related information which might have been needed as a pre-cursor to the transaction. Thus, an Intercloud Protocol for presence and messaging needs to exist which can support the 1-to-1, 1-to-many, and many-to-many Cloud to Cloud use cases.

The vision and topology for the Intercloud we will refer to [2][3] is as follows. At the highest level, the analogy is with the Internet itself: in a world of TCP/IP and the WWW, data is ubiquitous and interoperable in a network of networks known as the "Internet"; in a world of Cloud Computing, content, storage and computing is ubiquitous and interoperable in a network of Clouds known as the "Intercloud"; this is illustrated in Figure 1.

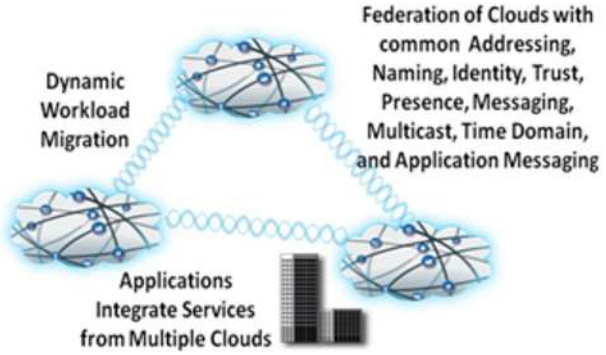

Figure 1. The Intercloud Vision

The reference topology for realizing this vision is modeled after the public Internet infrastructure. Again, using the generally accepted terminology [1][2][3][4][5][6][7], there are Public Clouds, which are analogous to ISP's and Service Providers offering routed IP in the Internet world. There are Private Clouds which is simply a Cloud which an organization builds to serve itself.

There are Intercloud Exchanges (analogous to Internet Exchanges and Peering Points) where clouds can interoperate, and there is an Intercloud Root, containing services such as Naming Authority, Trust Authority, Directory Services, and other "root" capabilities. It is envisioned that the Intercloud root is of course physically not a single entity, a global replicating and hierarchical system similar to DNS [12] would be utilized. All elements in the Intercloud topology contain some gateway capability analogous to an Internet Router, implementing Intercloud protocols in order to participate in Intercloud interoperability. We call these Intercloud Gateways. The entire topology is detailed in Figure 2.

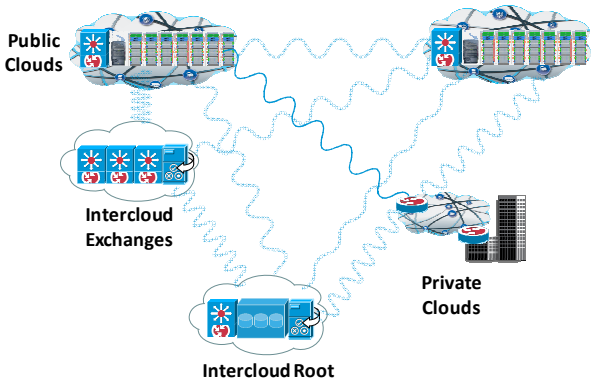

Figure 2. Reference Intercloud Topology

The Intercloud Gateways would provide mechanism for supporting the entire profile of Intercloud protocols and standards. The Intercloud Root and Intercloud Exchanges would facilitate and mediate the initial Intercloud negotiating process among Clouds. Once the initial negotiating process is completed, each of these Cloud instance would collaborate directly with each other via a protocol and transport appropriate for the interoperability action at hand; for example, a reliable protocol might be needed for transaction integrity, or a high speed streaming protocol might be needed optimized for data movement over a particular link.

3. Intercloud Root, Exchanges, and Catalog

Various providers will emerge in the enablement of the Intercloud. We first envision a community governed set of Intercloud Root providers who will act as brokers and host the Cloud Computing Resource Catalogs for the Intercloud computing resources. They would be governed in a similar way in which DNS, Top Level Domains [13] or Certificate Authorities [14] are, by an organization such as ISOC [15] or ICANN [16]. They would also be responsible for mediating the trust based federated security among disparate clouds by acting as Security Trust Service providers using standards such as SASL [17] and SAML [18].

The Intercloud Root instances will work with Intercloud Exchanges to solve the n^2 problem by facilitating as mediators for enabling connectivity among disparate cloud environments. This is a much preferred alternative to each cloud vendor establishing connectivity and collaboration among themselves (point-to-point), which would not scale physically or in a business sense.

4. XMPP Architectural Considerations

First we investigate how Intercloud Exchange providers will facilitate the negotiation dialog and collaboration among disparate heterogeneous cloud environments, working in concert with Intercloud Root instances. XMPP is a set of open XML technologies for presence and real-time communication developed by the Jabber open-source community in 1999, formalized by the IETF in 2002-2004, continuously extended through the standards process of the XMPP Standards Foundation. XMPP supports presence, structured conversation, lightweight middleware, content syndication, and generalized routing of XML data.

Intercloud Root instances will host the root XMPP servers containing all presence information for Intercloud Root instances, Intercloud Exchange Instances, and Internet visible Intercloud capable Cloud instances. Intercloud Exchanges will host second-tier XMPP servers. Individual Intercloud capable Clouds will communicate with each other, as XMPP clients, via XMPP server environment hosted by Intercloud Roots and Intercloud Exchanges. We will be using a Cloud extension to XMPP.

5. XMPP Services Framework

First, we must consider how to construct a Services Framework layer on top of XMPP, analogous to the HTTP-based Web service technologies, like the Simple Object Access Protocol (SOAP) and REpresentational State Transfer (REST) services. Today these are the most common technologies for interfaces on a services framework. However, the intrinsically synchronous HTTP protocol is unsuitable for time-consuming operations, like computationally demanding database lookups or calculations, and server timeouts are common obstacles. A very common workaround is to implement a ticketing mechanism in the service, where the client receives a ticket that can be used to repetitively poll for results and is highly inefficient.

XMPP based services, on the other hand, are capable of asynchronous communication. This implies that clients do not have to poll repetitively for status, but the service sends the results back to the client upon completion. As an alternative to RESTful or SOAP service interfaces, XMPP based services are ideal for lightweight service scenarios. To address this issue, we leverage a series of XMPP extensions (XEP series) defined by XMPP standards foundation. One of these extensions is XEP-0244 [19]. Extension XEP-0244 provides a "services" framework on top of base XMPP, named IO Data, which was designed for sending messages from one computer to another, providing a transport for remote service invocation and attempting to overcome the problems with SOAP and REST. A reference implementation for the IO Data XEP, XMPP Web Services for Java (xws4j), is already in place and available [20], which we are using.

6. XMPP Encryption & Authentication

XMPP includes a method for securing the XML stream from tampering and eavesdropping. This channel encryption method makes use of the Transport Layer Security (TLS) protocol [21], along with a "STARTTLS" extension that is modeled after similar extensions for the IMAP [22], and POP3 [23] protocols. Clouds use TLS to secure the streams prior to attempting the completion of SASL based authentication negotiation. SASL has a method for authenticating a stream by means of an XMPP-specific profile of the protocol. SASL provides a generalized method for adding authentication support to connection-based protocols. Currently, the following authentications methods are supported by XMPP-specific profile of SASL protocol: "DIGEST-MD5", "CRAM-MD5", "PLAIN", and "ANONYMOUS". SAML provides authentication in a federated environment. Currently, there is no support for SAML in XMPP-specific profile of SASL protocol. However, there is a draft proposal published that specifies a SASL mechanism for SAML 2.0 that allows the integration of existing SAML Identity Providers with applications using SASL. The following sample shows the data flow for a Cloud securing a stream to an Intercloud Root, using STARTTLS. It also shows SAML2.0 based authentication steps.

Step 1: Cloud starts stream to Intercloud Root:

```
<stream:stream
    xmlns='jabber:client'
    xmlns:stream='http://etherx.jabber.org/streams'
    to='intercloudexchg.com'
    version='1.0'>
```

Step 2: Intercloud Root responds by sending a stream tag to client:

```
<stream:stream
    xmlns='jabber:client'
    xmlns:stream='http://etherx.jabber.org/streams'
    id='cloud1_id1'
    from='intercloudexchg.com'
    version='1.0'>
```

Step 3: Intercloud Root sends the STARTTLS extension to Cloud:

```
<stream:features>
  <starttls xmlns='urn:ietf:params:xml:ns:xmpp-tls'>
    <required/>
  </starttls>
</stream:features>
```

Step 4: Cloud sends Root the STARTTLS command:

```
<starttls xmlns='urn:ietf:params:xml:ns:xmpp-tls'/>
```

Step 5: Intercloud Root informs Cloud to proceed:

```
<proceed xmlns='urn:ietf:params:xml:ns:xmpp-tls'/>
```

Step 5 (alt): Root informs Cloud that TLS negotiation has failed and closes both stream and TCP connection:

```
<failure xmlns='urn:ietf:params:xml:ns:xmpp-tls'/>
</stream:stream>
```

Step 6: Cloud and Intercloud Root attempt to complete TLS negotiation over the existing TCP connection.

Step 7: If TLS negotiation is successful, Cloud initiates a new stream to Intercloud Root:

```
<stream:stream
    xmlns='jabber:client'
    xmlns:stream='http://etherx.jabber.org/streams'
    to='intercloudexchg.com'
    version='1.0'>
```

Step 7 (alt): If TLS negotiation is unsuccessful, Intercloud Root closes TCP connection.

Step 8: Intercloud Root responds by sending a stream header to Cloud along with any available stream features:

```
<stream:stream
    xmlns='jabber:client'
    xmlns:stream='http://etherx.jabber.org/streams'
    from='intercloudexchg.com'
    id=' cloud1_id2'
    version='1.0'>
<stream:features>
  <mechanisms xmlns='urn:ietf:params:xml:ns:xmpp-
sasl'>
    <mechanism>DIGEST-MD5</mechanism>
    <mechanism> CRAM-MD5</mechanism>
    <mechanism>PLAIN</mechanism>
    <mechanism>ANONYMOUS</mechanism>
    <mechanism>EXTERNAL</mechanism>
    <mechanism>SAML20</mechanism>
  </mechanisms>
</stream:features>
```

Step 9: Cloud continues with SASL based authentication negotiation.

Step 10: Cloud selects an authentication mechanism:

```
<auth xmlns='urn:ietf:params:xml:ns:xmpp-sasl'
mechanism='SAML20'/>
```

Step 11: Intercloud Root sends a BASE64 [24] encoded challenge to Cloud as an HTTP Redirect to the SAML assertion consumer service with the SAML Authentication Request in the redirection URL.

Step 12: Cloud sends a BASE64 encoded empty response to the challenge:

```
<response xmlns='urn:ietf:params:xml:ns:xmpp-sasl'> =
</response>
```

Step 13: The Cloud now sends the URL to the local Intercloud Gateway for processing. The Intercloud Gateway engages, just like a browser would, in a normal SAML authentication flow (external to SASL), like redirection to the Identity Provider. Once authenticated, the Intercloud Gateways is passed back to the Cloud who sends the AuthN XMPP response to the Intercloud Root, containing the subject-identifier and the "jid" as an attribute.

Step 14: Intercloud Gateway informs Cloud of successful authentication:

```
<success xmlns='urn:ietf:params:xml:ns:xmpp-sasl'/>
```

Step 14 (alt): Intercloud Gateway informs Cloud of failed authentication:

```
<failure xmlns='urn:ietf:params:xml:ns:xmpp-sasl'>
<temporary-auth-failure/>
</failure>
</stream:stream>
```

7. XMPP based Service Invocation

It was envisioned that the way a Cloud would find the appropriate services is by leveraging a catalog of available resources published in a directory residing in the Intercloud Root. The Cloud's resource needs would be specified similarly, and a query would match the availability to the need. The technologies to use for this are based in the Semantic Web [25] which provides for a way to add "meaning and relatedness" to objects on the Web, by way of specifying Ontologies. For the Intercloud, we use this technique to specify resources such as storage, computing, and all the other possible services which Cloud both expose and consume. RDF is a way to specify such resources, and SPARQL [26] is a query/matching system for RDF. Later we will expand specifically on the RDF and SPARQL areas of the Intercloud problem, but for now let us detail within XMPP, how one would go about invocation of a SPARQL query with an Intercloud Root.

The following request invokes a SPARQL query over an XMPP connection to the Intercloud Root, to apply preferences and constraints to the resources in the computing semantics catalog for determining if the service description on another Cloud meets the constraints of the first Cloud's interest. We use IO Data XEP, XMPP Web Services for Java (xws4j):

```
<iq type='set'
    from='user@cloud1.org'
    to='service.intercloudexchg.com'
    id='cloud1_id1'>
    <command xmlns=
      'http://jabber.org/protocol/commands'
      node='constraint_catalog_resources'
      action='execute'>
      <iodata xmlns=
      'urn:xmpp:tmp:io-data' type='input'>
      <in>
      <constraints xmlns='http://www.csp/resOntology'>
        <constraint>
         <attribute>availabilityQuanity </attribute>
         <value>99.999</value>
        </constraint>
        <constraint>
         <attribute>replicationFactor</attribute>
         <value>5</value>
        </constraint>
        <constraint>
         <attribute>tierCountries</attribute>
         <value>JAPAN</value>
        </constraint>
        <constraint>
         <attribute>StorageReplicationMethod
         </attribute>
         <value>AMQP</value>
        </constraint>
        <constraint>
         <attribute>InterCloudStorageAccess
         </attribute>
         <value>NFS</value>
        </constraint>
      </constraints>
      </in>
    </iodata>
  </command>
</iq>
```

The above service invocation request results into the following result set:

```
<iq type='result'
    from='service.intercloudexchg.com'
    to='user@cloud1.org'
    id='cloud1_id1'>
  <command xmlns=
    'http://jabber.org/protocol/commands'
    sessionid='RPC-SESSION-0000001'
    node='constraint_catalog_resources'
    status='completed'>
    <iodata xmlns=
    'urn:xmpp:tmp:io-data' type='output'>
      <out>
        <matchingClouds
         xmlns=' http://www.csp/resOntology'>
           <cloudName>cloud2</cloudName>
           <cloudName>cloud5</cloudName>
        </matchingClouds>
      </out>
    </iodata>
  </command>
</iq>
```

The example shows how the service invocation works inside of an XMPP conversation.

8. XMPP based Presence & Dialog

Next, assume that the requesting cloud has found a target cloud with which to interwork. It must now turn directly to the target cloud and dialog with it. This last section describes such a cloud-to-cloud presence and dialog scenario. The code sample is based on Google AppEngine XMPP JAVA API set [27].

The following code sample tests for a service availability then sends a message as part of the collaboration dialog:

```
// ...
        JID jid = new JID("user@cloud2.com");
        String msgBody = "Cloud 2, I would like to use
your resources for storage replication using AMQP over
UDT protocol.";
        Message msg = new MessageBuilder()
            .withRecipientJids(jid)
            .withBody(msgBody)
            .build();

        boolean messageSent = false;
        XMPPService xmpp =
XMPPServiceFactory.getXMPPService();
        if (xmpp.getPresence(jid).isAvailable()) {
            SendResponse status =
xmpp.sendMessage(msg);
            messageSent =
(status.getStatusMap().get(jid) ==
SendResponse.Status.SUCCESS);
        }

        if (!messageSent) {
            // Send an email message instead...
        }
```

Step 2: The following code sample shows how the recipient Cloud responds back to the chat message as part of the collaboration dialog.

```
/* Handler class for all XMPP activity. */
public class XmppReceiverServlet extends HttpServlet
{
  private static final XMPPService xmppService =
      XMPPServiceFactory.getXMPPService();

  public void doPost(HttpServletRequest request,
HttpServletResponse response)
      throws IOException {
    Message message =
xmppService.parseMessage(request);

    Message reply = new MessageBuilder()
        .withRecipientJids(message.getFromJid())
        .withMessageType(MessageType.NORMAL)
        .withBody("Cloud 1, please go ahead and use my
resources for storage replication using AMQP/UDT
protocol.")
        .build();

    xmppService.sendMessage(reply);
  }
```

9. Ontology based Cloud Computing Resources Catalog

In order for the Intercloud capable Cloud instances to federate or otherwise interoperate resources, a Cloud Computing Resources Catalog system is necessary infrastructure. This catalog is the holistic and abstracted view of the computing resources across disparate cloud environments. Individual clouds will, in turn, will utilize this catalog in order to identify matching cloud resources by applying certain Preferences and Constraints to the resources in the computing resources catalog. The technologies to use for this are based on the Semantic Web which provides for a way to add "meaning and relatedness" to objects on the Web. To accomplish this, one defines a system for normalizing meaning across terminology, or Properties. This normalization is called an Ontology.

Comprehensive semantic descriptions of services are essential to exploit them in their full potential. That is discovering them dynamically, and enabling automated service negotiation, composition and monitoring. The semantic mechanisms currently available in service registries such as UDDI [28] are based on taxonomies called "tModel" [29]. tModel fails to provide the means to achieve this, as they do not support semantic discovery of services [30][31].

We are proposing a new and improved service directory on the lines of UDDI but based on RDF/OWL [32] ontology framework instead of current tModel based taxonomy framework. This catalog captures the computing resources across all clouds in terms of "Capabilities", "Structural Relationships" and Policies (Preferences and Constraints). This Catalog is illustrated in Figure 3.

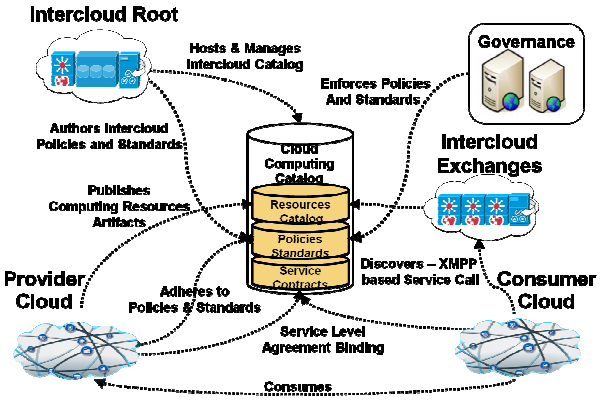

Figure 3. Cloud Computing Catalog

10. Cloud Computing Resources Ontology

In order to ensure that the requirements of an intercloud enabled cloud provider are correctly matched to the infrastructure capabilities in an automated fashion, there is a need for declarative semantic model that can capture both the requirements and constraints of computing resources.

Over the last several years, there has been ongoing effort around automation of datacenter/s by companies such as Elastra [33]. Elastra has defined a modeling language called EDML [34] for specifying the datacenter computing resources semantics in terms of XML based markup language.

We are proposing a similar ontology based semantic model that captures the features and capabilities available from a cloud provider's infrastructure. These capabilities are logically grouped together and exposed as standardized units of provisioning and configuration to be consumed by another cloud provider/s. These capabilities are then associated with policies and constraints for ensuring compliance and access to the computing resources.

The proposed ontology based model not only consists of physical attributes but quantitative and qualitative attributes such as "Service Level Agreements (SLAs)", "Disaster Recovery" policies, "Pricing" policies, "Security and Compliance" policies, and so on. The following is a high level schematic of such ontology based semantic model.

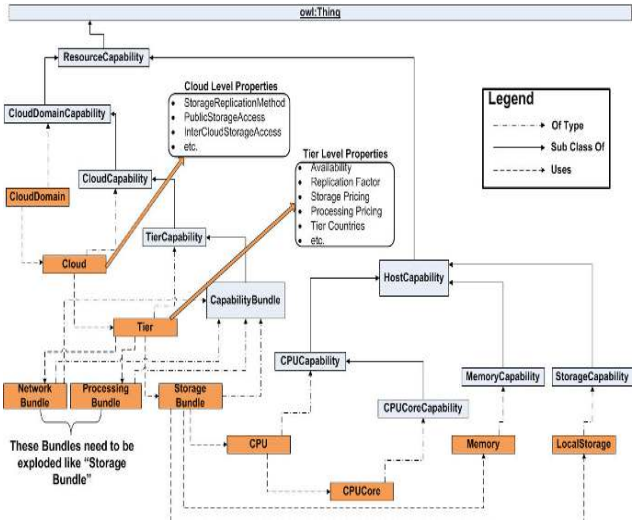

Figure 4. Cloud Computing Resources Ontology

At a very basic level, the RDF model is called a "triple" as it consists of three parts, Subject/Property/Object. It essentially contains one or more "descriptions" of resources. A "description" is a set of statements about a resource. It is structurally similar to entity/attribute/value. Essentially, a statement in RDF pulls resources, properties, and property values together. Statements are typically called triples because they include a subject (the resource), a predicate/verb (the property), and an object (the property value or another resource itself). RDF allows you to define a group of things with common characteristics called "Classes". "Classes" are allowed to inherit characteristics and behaviors from a parent class. Each user-defined class is implicitly a subclass of super class called "owl:Thing".

The hierarchy of user-defined classes in our proposed ontology scheme are "ResourceCapability" → "CloudDomainCapability" → "CloudCapability" → "TierCapabil;ity" → "CapabilityBundle".

In order to demonstrate a working example, the following is a code snippet of N-Triples [35] based ontology semantic model instead. N-Triples and Turtle [36] are a human-friendlier alternative to RDF/XML. N-Triples or Turtle code, in turn, can be easily converted to RDF/XML format using a converter tool. The following sample shows the flow for semantic model for cloud computing resources. Due to the large size of the proposed semantic model for cloud computing resources, we are unable to capture the sample RDF code snippet in this document. In order to demonstrate our working example, we are showing N-Triples [35] code snippet instead.

Step 1: In our ontology example, "CloudDomain" is an instance of class "CloudDomainCapability". It consists of three resources "Cloud.1", "Cloud.2" and "Cloud.3":

```
<http://cloud/domain>
<http://www.csp/resOntology#hasCapability>
<http://cloud/domain/#cloud.1>.

<http://cloud/domain>
<http://www.csp/resOntology#hasCapability>
<http://cloud/domain/#cloud.2>.

<http://cloud/domain>
<http://www.csp/resOntology#hasCapability>
<http://cloud/domain/#cloud.3>.

<http://cloud/domain> <http://www.w3.org/1999/02/22-
rdf-syntax-ns#type>
<http://www.csp/resOntology#ClouddomainCapability>.

<http://cloud/domain> <http://www.w3.org/2000/01/rdf-
schema#label> "Cloud Computing
domain"^^<http://www.w3.org/2001/XMLSchema#string>.
```

Step 2: “Cloud.1”, in turn, consists of tier instances “tier.1”, “tier.2” and “tier.3”:

```
<http://cloud/domain/#cloud.1>
<http://www.csp/resOntology#hasCapability>
<http://cloud/domain/cloud.1#tier1>.

<http://cloud/domain/#cloud.1>
<http://www.csp/resOntology#hasCapability>
<http://cloud/domain/cloud.1#tier2>.

<http://cloud/domain/#cloud.1>
<http://www.csp/resOntology#hasCapability>
<http://cloud/domain/cloud.1#tier3>.
```

Step 3: Each of these cloud instances has associated properties such as “StorageReplicationMethod”, “InterCloudStorageAccess” etc. etc. These properties are, in turn, used for determining if the computing resources of a cloud provider meet the preferences and constraints of the requesting cloud’s interest and requirements:

```
<http://cloud/domain/#cloud.1>
<http://www.csp/resOntology#hasCapability>
<http://cloud/domain/cloud.1#Storage-Replication-
Method>.

<http://cloud/domain/#cloud.1>
<http://www.csp/resOntology#hasCapability>
<http://cloud/domain/cloud.1#Inter-Cloud-Storage-
Access>.

<http://cloud/domain/#cloud.1>
<http://www.csp/resOntology#hasCapability>
<http://cloud/domain/cloud.1#Public-Storage-Access>.

etc
```

Step 4: Computing resources are logically grouped together as bundles and exposed as standardized units of provisioning and configuration to be consumed by another cloud provider/s. These bundles are “StorageBundle”, “ProcessingBundle” and “NetworkBundle”. Each “Tier”, in turn, consists of instances of resource bundles such as “StorageBundle” etc. Each “Tier” also has its own associated properties depicting preferences and constraints:

```
<http://cloud/domain/cloud.1#tier1>
<http://www.csp/resOntology#hasCapability>
<http://cloud/domain/cloud.1/bundle/#storage1>.

<http://cloud/domain/cloud.1#tier1>
<http://www.csp/resOntology#hasCapability>
<http://cloud/domain/cloud.1/bundle/#processing1>.

<http://cloud/domain/cloud.1#tier1>
<http://www.csp/resOntology#hasCapability>
<http://cloud/domain/cloud.1/bundle/#network1>.

etc
```

Step 5: “StorageBundle”, in turn, consists of resources such as “CPU”, “CPU Cores”, “Memory” and “LocalStorage”:

```
<http://cloud/domain/cloud.1/bundle/#storage1>
<http://www.csp/resOntology#hasCapability>
<http://cloud/domain/cloud.1/bundle/storage1#CPU>.
```

```
<http://cloud/domain/cloud.1/bundle/#storage1>
<http://www.csp/resOntology#hasCapability>
<http://cloud/domain/cloud.1/bundle/storage1#LocalStor
age0>.

<http://cloud/domain/cloud.1/bundle/#storage1>
<http://www.csp/resOntology#hasCapability>
<http://cloud/domain/cloud.1/bundle/storage1#Memory>.

etc
```

11. SPARQL Query Language

SPARQL is a very powerful SQL-like language for querying and making semantic information machine process-able. The structure and example of a SPARQL Query is illustrated in Figure 5.

Structure:

```
PREFIX: Prefix definition (optional)
SELECT: Result form
FROM: Data sources (optional)
WHERE: Graph pattern (=path expression)
    •FILTER
    •OPTIONAL
```

Example:

```
PREFIX geo: <http://www.geography.org/schema.rdf#>
SELECT ?X ?Y
FROM <http://www.geography.org>
WHERE {  ?X geo:hasCapital ?Y.
           ?Y geo:areacode ?Z }
ORDER BY ?X
```

Figure 5. Structure & Example of SPARQL Query

SPARQL provides a very powerful language for executing very complex queries into the RDF data which are often necessary. In our case, the following example query applies certain Preferences and Constraints to the resources in the computing semantics catalog for determining if the service description on another cloud meets the constraints of the first cloud’s interest:

```
PREFIX  xsd: <http://www.w3.org/2001/XMLSchema#>
SELECT  ?cld1 ?cld2 ?cld3 ?cld4 ?cld5
WHERE   { ?cld1
<http://www.csp/resOntology#availabilityQuanity> ?avai
labilityQuanity .
  ?cld2
<http://www.csp/resOntology#replicationFactor> ?replic
ationFactor .
 ?cld3
<http://www.csp/resOntology#tierCountries> ?tierCountr
ies .
  ?cld4
<http://www.csp/resOntology#StorageReplicationMethod>
?StorageReplicationMethod .
  ?cld5 <http://www.csp/resOntology#
InterCloudStorageAccess > ?InterCloudStorageAccess .
FILTER ( ?availabilityQuanity = 99.999 )
FILTER ( ?replicationFactor = 5)
FILTER ( ?tierCountries = "Japan")
FILTER ( ?StorageReplicationMethod = "AMQP")
FILTER ( ?InterCloudStorageAccess = "NFS")
   }
```

12. Conclusions

We have gone into some detail to test the proposal that XMPP is a suitable control plane protocol for Intercloud. We successfully addressed topology, security, authentication, service invocation, and transported RDF and SPARQL within XMPP, We also used an XMPP Java API to a Cloud Service. Next we tested the proposal that Intercloud Exchanges with Ontology based Computing Resources Catalog can be the enablement of a "Federated Cloud" environment. The conclusion is that we have found XMPP and RDF along with the Intercloud Topology concepts, and an Intercloud Catalog using Ontology, to be a flexible and usable approach to the Intercloud problem.

13. References

[1] Youseff, L. and Butrico, M. and Da Silva, D., *Toward a unified ontology of cloud computing*, GCE'08 Grid Computing Environments Workshop, 2008.

[2] Lijun Mei, W.K. Chan, T.H. Tse, *A Tale of Clouds: Paradigm Comparisons and Some Thoughts on Research Issues*, APSCC pp.464-469, 2008

[3] *Cloud Computing Use Cases Google Group (Public)*, at http://groups.google.com/group/cloud-computing-use-cases, http://www.scribd.com/doc/18172802/Cloud-Computing-Use-Cases-Whitepaper ,

[4] Buyya, R. and Pandey, S. and Vecchiola, C., *Cloudbus toolkit for market-oriented cloud computing,* 1st International CloudCom , 2009

[5] Yildiz M, Abawajy J, Ercan T., Bernoth A., *A Layered Security Approach for Cloud Computing Infrastructure*, ISPAN, pp.763-767, 2009

[6] Bernstein, D., Ludvigson, E., Sankar, K., Diamond, S., and Morrow, M., *Blueprint for the Intercloud - Protocols and Formats for Cloud Computing Interoperability*, ICIW '09. Fourth International Conference on Internet and Web Applications and Services, pp. 328-336, 2009

[7] Bernstein, D., *Keynote 2: The Intercloud: Cloud Interoperability at Internet Scale*, NPC, pp.xiii, 2009 Sixth IFIP International Conference on Network and Parallel Computing, 2009

[8] *Extensible Messaging and Presence Protocol (XMPP): Core,* and related other RFCs at http://xmpp.org/rfcs/rfc3920.html

[9] *XMPP Standards Foundation* at http://xmpp.org/

[10] *W3C Semantic Web Activity*, at http://www.w3.org/2001/sw/

[11] *Resource Description Framework (RDF*), at http://www.w3.org/RDF/

[12] *Domain Names – Concepts and Facilities,* and related other RFCs, at http://www.ietf.org/rfc/rfc1034.txt

[13] *Domain Name System Structure and Delegation*, at http://www.ietf.org/rfc/rfc1591.txt

[14] *Internet X.509 Public Key Infrastructure, Certificate Policy and Certification Practices Framework,* at http://tools.ietf.org/html/rfc3647

[15] *The Internet Society,* at http://www.isoc.org/

[16] *The Internet Corporation for Assigned Names and Numbers*, at http://www.icann.org/

[17] *Simple Authentication and Security Layer (SASL*), at http://tools.ietf.org/html/rfc4422

[18] *Security Assertion Markup Language (SAML*), at http://saml.xml.org/saml-specifications

[19] *XEP-0244: IO Data*, at http://xmpp.org/extensions/xep-0244.html,

[20] *XMPP Web Services for Java (XWS4J*), at http://sourceforge.net/projects/xws4j/

[21] *The Transport Layer Security (TLS) Protocol*, at http://tools.ietf.org/html/rfc5246

[22] *Internet Message Access Protocol (IMAP*), at http://tools.ietf.org/search/rfc3501

[23] *Post Office Protocol (POP3*), at http://tools.ietf.org/html/rfc1939

[24] *The Base16, Base32, and Base64 Data Encodings*, at http://www.ietf.org/rfc/rfc4648.txt

[25] *W3C Semantic Web Activity*, at http://www.w3.org/2001/sw/

[26] *SPARQL Query Language for RDF*, at http://www.w3.org/TR/rdf-sparql-query/

[27] *Google App Engine, The XMPP Java API*, at http://code.google.com/appengine/docs/java/xmpp/

[28] *OASIS UDDI Specification TC,* at http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=uddi-spec

[29] *UDDI Registry tModels,* at http://www.uddi.org/taxonomies/UDDI_Registry_tModels.htm

[30] Paolucci, M., Kawamura T., Payne T., and Sycara K., *Importing the Semantic Web in UDDI*, Web Services, E-Business and Semantic Web Workshop, 2002.

[31] Moreau, L. and Miles, S. and Papay, J. and Decker, K. and Payne, T., *Publishing semantic descriptions of services*, First GGF Semantic Grid Workshop, held at the Ninth Global Grid Forum, Chicago IL, USA, 2003

[32] Web Ontology Language, at http://www.w3.org/TR/owl-features/

[33] Elastra, at http://www.elastra.com

[34] EDML, at http://www.elastra.com/technology/languages/edml

[35] N-Triples, at http://www.w3.org/2001/sw/RDFCore/ntriples/

[36] Turtle – Terse RDF Triple Language, at http://www.w3.org/TeamSubmission/turtle/#sec-diff-n3