Using XMPP as a transport in Intercloud Protocols

David Bernstein
Huawei Technologies, USA
dbernstein@huawei.com

Deepak Vij
Huawei Technologies, USA
dvij@huawei.com

Abstract

Cloud Computing is a term applied to large, hosted datacenters, usually geographically distributed, which offer various computational services on a "utility" basis. Most typically the configuration and provisioning of these datacenters, as far as the services for the subscribers go, is highly automated, to the point of the service being delivered within seconds of the subscriber request. Additionally, the datacenters typically use hypervisor based virtualization as a technique to deliver these services. The concept of a cloud operated by one service provider or enterprise interoperating with a clouds operated by another is a powerful idea. So far that is limited to use cases where code running on one cloud explicitly references a service on another cloud. There is no implicit and transparent interoperability. This interoperability should be more than cloud to cloud, it should embody 1-to-many and many-to-many models. Working groups have proposed building a layered set of protocols to solve this interoperability challenge called "Intercloud Protocols". 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 investigates and details the use of XMPP in Intercloud protocols and concludes that logically it is a perfectly suited choice.

1. Introduction

Cloud Computing has emerged recently as a label for a particular type of datacenter. For the purposes of this paper, we define Cloud Computing as a datacenter/s which:

1. May be hosted by anyone; an enterprise, a service provider, or a government.
2. Implement a pool of computing resources and services which are shared amongst subscribers.
3. Charge for resources and services using an "as used" metered and/or capacity based model.
4. Are usually geographically distributed, in a manner which is transparent to the subscriber (unless they explicitly ask for visibility of that).
5. Are automated in that the provisioning and configuration (and de-configuration and un-provisioning) of resources and services occur on a "self service" basis, usually programmatic request of the subscriber, occur in an automated way with no human operator assistance, and are delivered in one or two orders of seconds.
6. Resources and services are delivered virtually, that is, although they may appear to be physical (servers, disks, network segments, etc) they are actually virtual implementations of those on an underlying physical infrastructure which the subscriber never sees.
7. The physical infrastructure changes rarely. The virtually delivered resources and services are changing constantly.
8. Resources and services may be of a physical metaphor (servers, disks, network segments, etc) or they may be of an abstract metaphor (blob storage functions, message queue functions, email functions, multicast functions, all of which are accessed by running of code or script to a set of API's for these abstract services). These may be intermixed.

Cloud Computing services as defined above are best exemplified by the Amazon Web Services (AWS) [1][2] or Google AppEngine [3][4]. Both of these systems exhibit all eight characteristics as detailed above. Various companies are beginning to offer similar services, such as the Microsoft Azure Service [5], and software companies such as VMware [6] and open source projects such as UCSB Eucalyptus [7][8] are creating software for building a cloud service.

In case 8, where the resources and services are of a physical metaphor, the cloud is said to be exposing "Infrastructure as a Service", or IaaS. In the last case described above (number 8), where the resources and services are of an abstract metaphor, the cloud is said to be exposing "Platform as a Service", or PaaS. A PaaS cloud looks like a remote, virtual, distributed implementation of a managed code container, or

"Application Server", similar to J2EE [9] or .NET [10]. The terms are well accepted now [11].

Use Cases and Scenarios for Cloud IaaS and PaaS interoperability [12][13] 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 [14][15]. More recently, specific implementation approaches for Intercloud protocols [16][17] have been proposed, including specifically proposing XMPP [18][19] as a transport protocol within the Intercloud protocol suite.

This paper continues that work where we specifically create XMPP code of Intercloud functions, to validate the suitability of XMPP at a logical level and to explore the suitability at an implementation level.

2. Intercloud Topology

The vision and topology for the Intercloud we will refer to [12][13] 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.

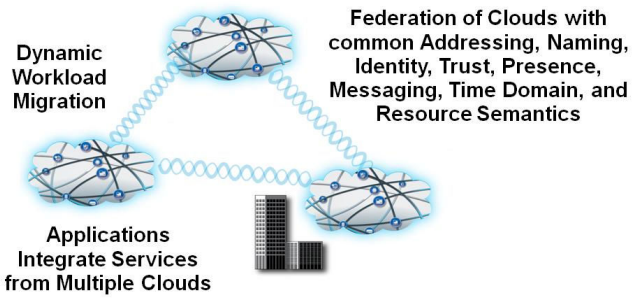

Figure 1. The Intercloud Vision

The reference topology for realizing this vision is modeled after the public Internet infrastructure. Again, using the generally accepted terminology [11][12][13][14][15][18][19], 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 [20] 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.

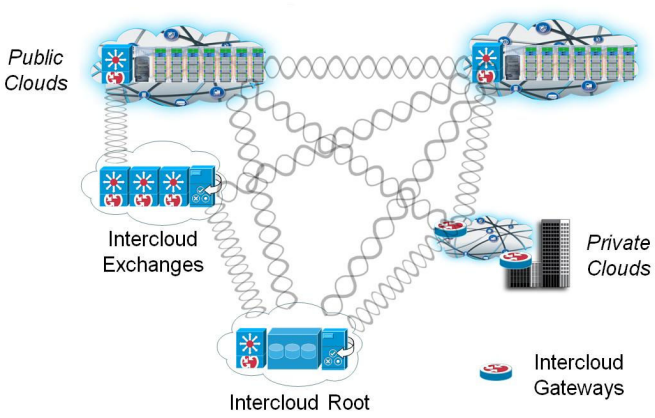

Figure 2. Reference Intercloud Topology and elements

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. It is this Presence and Messaging capability we are considering in this paper. 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. However, if the Cloud instances needed a signaling to control that collaboration process, we believe that the requirements for that are the same as the initial negotiating phase. Thus, we consider a protocol for both cases of the "control plane" of presence and messaging.

3. XMPP Architectural Considerations

As detailed in earlier work [12][13], 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.

Extensible Messaging and Presence Protocol (XMPP) [18][19] is exactly such a protocol. 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.

For Intercloud protocols, XMPP is a viable control plane presence and dialog protocol. XMPP root services would be located in the Intercloud Root in the topology explained above.

XMPP defines protocols for communicating between groups of entities which register with an XMPP server. Registration is dynamic and provides the basis for Presence. In a large implementation, such as the global Intercloud envisioned herein, XMPP servers are connected together. This is identical to the way service providers connect XMPP servers together already supporting cross-domain Instant Messaging. In this way, XMPP facilitates both presence and many-to-many messaging across service provider domains. XMPP messages are extensible, and can be used to carry messages of different types. For example, an XMPP Message can carry Instant Messaging (IM) type traffic. We will be using a Cloud extension to XMPP.

XMPP servers support encrypted communication (SASL (Simple Authentication and Security Layer) and TLS (Transport Layer Security)) with the option to restrict XMPP servers to accept only encrypted client-to-server and server-to-server connections.

4. 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 [21]. 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 [22], which we are using.

5. XMPP Encryption and 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 [23], along with a “STARTTLS” extension that is modeled after similar extensions for the IMAP [24], and POP3 [25] protocols. Clouds use TLS to secure the streams prior to attempting the completion of SASL based authentication negotiation.

SASL is a method for authenticating a stream by means of an XMPP-specific profile of the protocol [26]. 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 [27] 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 the STARTTLS command to Intercloud Root:

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

Step 5: Intercloud Root informs Cloud that it is allowed to proceed:

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

Step 5 (alt): Intercloud 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 [28] encoded challenge to Cloud in the form of an HTTP Redirect to the SAML assertion consumer service with the SAML Authentication Request as specified 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>
```

6. XMPP based Service Invocation

Once the Cloud has now secured a connection to the Intercloud root, it can look for a suitable other Cloud with which to interoperate. It will either interoperate through an Intercloud Exchange, or directly Cloud to Cloud, as the case may be. It was envisioned [17] 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 [29] 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 [30] is a way to specify such resources, and SPARQL [31] is a query/matching system for RDF. As illustrated in Figure 3 below, the following diagram shows the overall vision of how ontology based available cloud computing resources information will reside in a directory as part of the overall Intercloud topology.

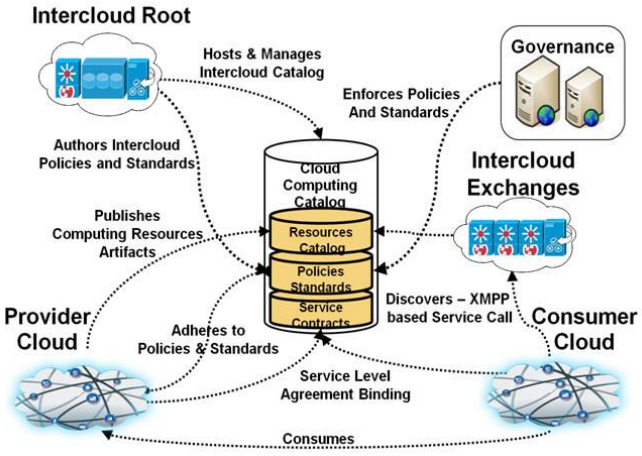

Figure 3. Cloud Computing Catalog

Later work of ours will expand specifically on the RDF and SPARQL areas of the Intercloud problem, but for now let us detail within XMPP, how one would invoke a SPARQL query with an Intercloud Root.

The following service request invokes a SPARQL query over the XMPP connection to the Intercloud Root, in order to apply 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. Again, this uses 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>
```

7. XMPP based Presence and 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 [32]. 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);
  }
```

8. XMPP versus SIP

Currently, real-time collaboration capabilities are built on top of two prominent standards for Messaging and Presence applications: XMPP and SIP/SIMPLE [33][34].

Based on our research we have concluded that XMPP is a much better and a suitable fit for Intercloud computing environment. XMPP software stack is much more lightweight than SIP/SIMPLE stack; additional features can very easily be added via extensions to the XMPP protocol which makes it flexible. Popular internet applications such as Google GTalk use XMPP as the underlying protocol. The XML foundation of XMPP greatly simplifies integration with existing environments and eases the movement of data to and from the XMPP network.

XMPP.org [35] has published a detail comparison of these two standards for all the relevant features of Messaging and Presence functionality. As described, SIP/SIMPLE standard lacks several key features typically desired in a many-to-many collaborative environment. Another salient feature of XMPP is the fact that all messages go through a server, which allows the server to mediate, log and audit messages. As explained earlier as part of the Intercloud topology section that Intercloud Exchanges play a very important role of catalysts or market makers for bringing cloud buyers and sellers together. A centralized controlled environment such as XMPP is a key requirement for enabling Intercloud Exchanges as market makers and be able to practice arbitrage. SIP/SIMPLE, on the other hand, is a peer-to-peer based standard, hence not suitable for Intercloud environment.

9. Conclusions and Future Work

We have gone into some detail to test the proposal that XMPP is a suitable control plane protocol for Intercloud. We tried a variety of different techniques along the way:

- Fitting XMPP into an Intercloud Topology
- Securing the XMPP conversation using TLS
- Authentication over XMPP using SAML
- Service Invocation over XMPP using IO Data XEP, XMPP Web Services for Java (xws4j)
- RDF and SPARQL within XMPP
- XMPP Java API to a Cloud Service

The conclusion is that for each of these techniques we found XMPP to be flexible and usable. There are also widespread industry techniques for using XMPP available. We believe we have confirmed XMPP as a core Intercloud transport protocol.

As to continuing work, we are continuing to develop the suite of Intercloud protocols. Our next work will be to elaborate on the RDF Ontology definitions and SPARQL query work for Cloud Computing resources, with an eye towards IaaS (Storage) interoperability first. With the XMPP approach and the RDF Ontology we should be able to demonstrate a "Simple Storage Replication Protocol" for Intercloud next.

10. References

[1] Amazon Web Services at http://aws.amazon.com/
[2] James Murty, *Programming Amazon Web Services; S3, EC2, SQS, FPS, and SimpleDB,* O'Reilly Press, 2008.
[3] Google AppEngine at http://code.google.com/appengine/
[4] Eugene Ciurana, *Developing with Google App Engine,* Firstpress, 2009.
[5] Microsoft Azure, at http://www.microsoft.com/azure/default.mspx
[6] VMware VCloud Initiative at http://www.vmware.com/technology/cloud-computing.html
[7] Nurmi D., Wolski R., Grzegorczyk C., Obertelli G., Soman S., Youseff L., Zagorodnov D., *The Eucalyptus Open-source Cloud-computing System,* Proceedings of Cloud Computing and Its Applications, Chicago, Illinois (October 2008)
[8] Nurmi D., Wolski R., Grzegorczyk C., Obertelli G., Soman S., Youseff L., Zagorodnov D., *Eucalyptus: A Technical Report on an Elastic Utility Computing Architecture Linking Your Programs to Useful Systems,* UCSB Computer Science Technical Report Number 2008-10 (August 2008)
[9] JSR 88: Java Enterprise Edition Application Deployment at http://jcp.org/en/jsr/detail?id=88
[10] Microsoft .NET at http://www.microsoft.com/net/
[11] Youseff, L. and Butrico, M. and Da Silva, D., *Toward a unified ontology of cloud computing*, GCE'08 Grid Computing Environments Workshop, 2008.
[12] 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 IEEE Asia-Pacific Services Computing Conference, 2008
[13] *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 , accessed March 2010
[14] Buyya, R. and Pandey, S. and Vecchiola, C., *Cloudbus toolkit for market-oriented cloud computing,* Proceeding of the 1st International Conference on Cloud Computing (CloudCom), 2009
[15] Yildiz M, Abawajy J, Ercan T., Bernoth A., *A Layered Security Approach for Cloud Computing Infrastructure*, ISPAN, pp.763-767, 10th International Symposium on Pervasive Systems, Algorithms, and Networks, 2009
[16] 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
[17] 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
[18] *Extensible Messaging and Presence Protocol (XMPP): Core,* and related other RFCs at http://xmpp.org/rfcs/rfc3920.html
[19] *XMPP Standards Foundation* at http://xmpp.org/
[20] *Domain Names – Concepts and Facilities,* and related other RFCs, at http://www.ietf.org/rfc/rfc1034.txt
[21] *XEP-0244: IO Data*, at http://xmpp.org/extensions/xep-0244.html, accessed March 2010
[22] *XMPP Web Services for Java (XWS4J*), at http://sourceforge.net/projects/xws4j/ , accessed March 2010
[23] *The Transport Layer Security (TLS) Protocol*, at http://tools.ietf.org/html/rfc5246
[24] *Internet Message Access Protocol (IMAP*), at http://tools.ietf.org/search/rfc3501
[25] *Post Office Protocol (POP3*), at http://tools.ietf.org/html/rfc1939
[26] *Simple Authentication and Security Layer (SASL*), at http://tools.ietf.org/html/rfc4422
[27] *Security Assertion Markup Language (SAML*), at http://saml.xml.org/saml-specifications
[28] *The Base16, Base32, and Base64 Data Encodings*, at http://www.ietf.org/rfc/rfc4648.txt
[29] *W3C Semantic Web Activity*, at http://www.w3.org/2001/sw/
[30] *Resource Description Framework (RDF*), at http://www.w3.org/RDF/
[31] *SPARQL Query Language for RDF*, at http://www.w3.org/TR/rdf-sparql-query/
[32] *Google App Engine, The XMPP Java API*, at http://code.google.com/appengine/docs/java/xmpp/
[33] *SIP: Session Initiation Protocol*, at http://tools.ietf.org/html/rfc3261
[34] *SIMPLE: Session Initiation Protocol for Instant Messaging and Presence Leveraging Extensions*, at http://datatracker.ietf.org/wg/simple/charter/
[35] *XMPP-SIMPLE Feature Comparison*, at http://xmpp.org/about/xmpp-simple.shtml