Generic Web Services: A Step Towards Green Computing

Mydhili K Nair
Dept.of Information Science & Engg.
M S Ramaiah Institute of Technology
Bangalore, India
mydhili.nair@gmail.com

Dr. V. Gopalakrishna
Integra Micro Systems
Bangalore, India
vgopi@integramicro.com

***Abstract*—Software as a Service (SaaS) is a buzzword especially in the realm of Cloud Computing. We can reduce the number of applications deployed on the data-centers by using similar applications hosted by the SaaS providers. Thus, *"reusability"* of a SaaS is a key-point of focus promoting the same code-base of an application serving multiple clients. This reduces the need for more storage space and computational power in the data-centers thus aligning the SaaS architecture to a *"greener"* computing paradigm. In this paper, we provide a prototype implementation framework, which uses the *same web service* to *Register* and *Report* the results of a backend Network Monitoring (NM) as well as Weather Monitoring application. We have designed and developed this *"Generic Web Services Framework"* as well as both the backend applications. The NM application is done using both Mobile Agents and SNMP and the WM application is implemented using Temperature Sensors. This paper aims to give a prototype implementation as proof of SaaS maturity levels of a generic / reusable web service thus orienting towards green computing.**

Keywords-Cloud Computing; Network Monitoring ; Web Services; Green Computing;

I. INTRODUCTION

Software as a Service (SaaS) is a model in which a service provider offers an Internet hosted version of their application with the hosting done either in house through a dedicated server or at hosting space managed by a third party. The customers of the service vendor access this application over Internet and pays for its utilization on a per-use, per-product or subscription-basis [7]. An example is a small handicraft business in a remote village in India, which immediately has a global presence and immediate access to the global market by just listing its products on a well-known portal such as e-Bay!

Thus, ***SaaS*** enabled by the amalgam of corporate intranets and the Internet proves to have a great potential to impact our everyday lives. The *most common implementation* of a ***SaaS*** offered over the Internet is as ***Web Services*** where *'Services'* are an implementation of the functionality of a business enterprise, which can be utilized by users of different applications. The logic of the service is *hidden* from the consumers of the service. In simple terms, services are the end-points of a connection [8]. *'Web Services(WS)'* are the set of protocols by which these end connections are made by allowing the services to be *published*, *discovered* and *used* in a technology neutral, standard form.

The ***five*** key benefits of ***Web Services Based SaaS*** are [2]:

- Save money
- Save time
- Focus on business needs of the software rather than on deployment and maintenance infrastructure.
- Gain immediate access to the latest innovations to the software offered as a service.
- Join a global community using the software as a service thus gaining instant benchmarking data.

Therefore, we leverage on the concept that SaaS is a business model, which is proven and will continue to have a global impact It is certainly not one of those transient disruptive technologies that comes and goes is here to stay through the coming years.

Hence, in this paper, we focus on ***optimizing*** *SaaS* design, deployment and maintenance by using *generic / reusable Web Services(WS),* thus aligning business architectures towards ***energy efficient green computing*** as well as adding a ***sixth key benefit*** to ***SaaS*** described above, namely,

- **Save Energy** by reducing storage, computational and processing power at the servers where these "***green" services*** are deployed.

In this paper, we depict here the use of ***'Generic' Web Services*** at the business enterprise, which are not tightly coupled / tied down to any specific application. This enables multiple applications to ***'use' / 'consume'*** these Web Services. We also present implementation results of usage of ***two*** such *Generic WS* by ***two completely different*** backend monitoring legacy applications, namely *Network Monitoring and Weather (specifically Temperature) Monitoring*.

The structure of the paper is as follows. Section 2 describes the overall architecture of our Green Computing Framework. as well as highlights the design of the Generic Web Services. Section 3 presents the implementation results while Section 4 focuses on the conclusion and scope of future enhancements to our *Generic Web Services Framework.*

ISSN : 0975-3397

II. OVERALL ARCHITECTURE .

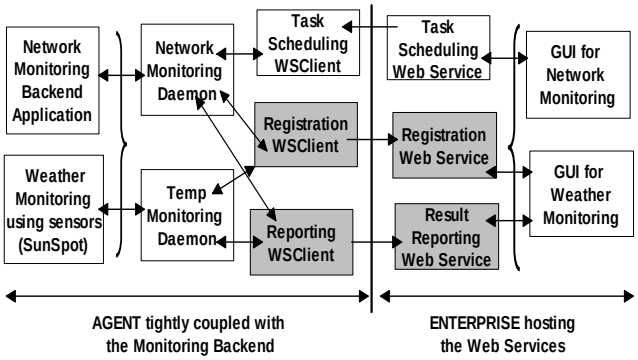

Figure 1. Overall Deployment Diagram of Generic SaaS Framework

The *generic / reusable web services green computing framework*, we describe in this paper, consists of three parts:

a) The Backend Monitoring Application
b) Agent Daemon (tightly coupled with backend app)
c) The Enterprise with the Web Services

The *shaded blocks* shown in Figure 1 are the reusable web services for both the backend applications, namely:

- Computer Network Monitoring(NM)
- Weather Monitoring(WM)

A. Backend Monitoring Applications

'Network Monitoring' (NM) application was developed by us in-house[9]. It uses ***SNMP*** (***S**imple **N**etwork **M**anagement **P**rotocol)* as well as ***Mobile Agents*** to communicate between the manager and its managed nodes. It adheres to the principles of Network Management, namely, ***FCAPS*** (***F**ault, **C**onfiguration, **A**ccounting, **P**erformance, and **S**ecurity*) and is a fully functional implementation. *Mobile Agents* are small pieces of software that migrate from host to host in a network. In our case, they carry instructions to from the manager node to interact with the *SNMP MIB* (***M**anagement **I**nformation **B**ase*) at the managed nodes.

'Weather Monitoring'(WM) application is an on-going work being developed by us for this research work. At this stage, we have implemented the ***Temperature Monitoring*** aspect of monitoring the weather, using *temperature sensors* called ***Sun-Spots*** given for academic research from Sun Microsystems. In this paper we show the implementation results of testing it to communicate with the Reusable Enterprise Web Services. A *Sun-Spot* kit consists of *two free-range devices* and *one base-station*. The free-range devices are those in which we can embed code to sense the temperature. We have used ***J2ME***(***J**ava **2** **M**obile **E**dition*) *API* to embed temperature *Midlets* into the free range devices. It communicates using ***Radiogram Protocol*** with the Base-Station device connected to a computer linked to the Internet.

B. Software Agent Daemon

Each of the aforementioned backend monitoring applications has a ***Software Agent*** tightly coupled with its implementation. They act as a *glue[10]* communicating with the Enterprise as well as the backend application. In our work we have two agents, both implemented as a *daemon processes*.

- The Network Monitoring Agent[10]
- The Temperature Monitoring Agent.

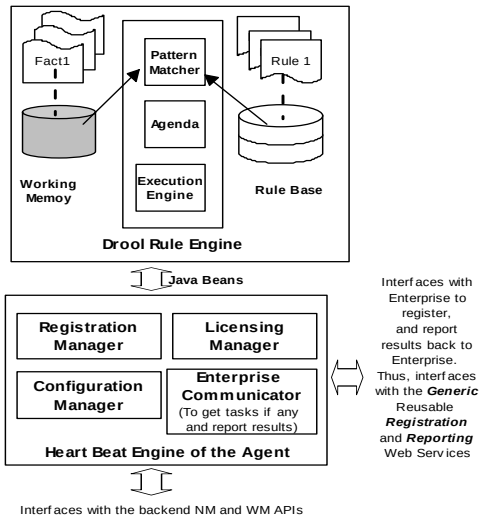

Figure 2. Architecture of the Agent of Generic SaaS Framework

As shown in Figure 2, the general design of our Software Agents has *four* parts, namely,

- Registration Manager
- Licensing Manager
- Configuration Manager
- Enterprise Communicator

The ***Registration Manager*** and ***Licensing Manager*** invokes the *Enterprise Generic Registration and Licensing WS*, to register itself. with a Device ID which is *unique* to the entire enterprise database.

The Algorithm described in Figure 3 below, clearly indicates the steps taken by the Agent's *Registration* and *Licensing Manager* to ensure that only a valid Agent is allowed to access and utilize the Services offered by the Enterprise. Each Agent comes with a contract expiry date within which it can access the Enterprise *SaaS*. The validity of the Agent is determined by this date which also has a *grace period*, within which the contract needs to be renewed of it has expired. These details are checked and maintained by the ***Configuration Manager.***

The *Agent* interacts with the *Drools Rule Engine* if its Contract Expiry Date is expired, to check the Rule Base and fire the appropriate action. A ***Rule Engine*** is made up of rules

ISSN : 0975-3397

```
Var DevID: Agent's Device ID given by the
        Configuration Manager.
        ContractDate: License Contract Expiry Date

Begin
 # Checks if the Agent is valid, registered with Enterprise.
 If DevID of Agent = DevID recorded at Enterprise
    # Check if the Contract has not expired

    If not ContractDate
     --- Implementation ----
     # Allow the agent to use the Enterprise
     # Update Enterprise with the new Asset details if any
     # Generates new activation code and records the same
      at the Enterprise to prevent fraud

    else
    ---Implementation of Rule Engine. Refer Figure 4.---

    # Control transfers to the Drools Rule Engine. This engine
      checks if the license is within the grace period of the
      contract. If so, it allows the Agent to interact with the
      Enterprise. If not, the Agent is brought down.
# Checks the validity of the Agent

else if DevID of Agent != DevID recorded at Enterprise
# Bring the Agent down, not allowing it to access  Enterprise.

End
```

Figure 3. Algorithm for the Agent Registration and Licensing for SaaS

.

which are discrete , as each of them applies to some subset of a potential problem, which needs to be solved.

```
package com.agent.license;
import com.agent.license.LicenseChecker;
import com.agent.license.LicenseAction;

rule "Contract Expired"
   when
      lc:LicenseChecker(status==
                          LicenseChecker.EXPIRED)
      la : LicenseAction( )
    then
      lc.setMessage( "Contract Period Expired, no
      grace period, set new license" );
      // Call the relevant action for the rule
      la.licenseExpiry();
      lc.setStatus(LicenseChecker.GRACEPERIOD);

 /* Notifying the Rule Engine modified Facts, so that
   it can be re-process */
      update( lc );
end
rule "License Grace Period"
   when
      lc:LicenseChecker(status==
                        LicenseChecker. GRACEPERIOD)
      la : LicenseAction( )
   then
      lc.setMessage( "Contract Period Expired, but
                        within grace period" );
     // Call the relevant action for the rule
      la.licenseGracePeriod();
end
```

Figure 4. 'License.drl', Sample Rule Base of Java Based Drools[10]

As depicted in Figure 2, the *Drools Rule Engine* has a *Working Memory* of *Facts* and a *Rule Knowledge Base* of Rules. *Facts* are objects (Java Beans) from your application that you insert into the working memory, which the rule engine can access. A sample ***'License.drl'*** Rule Base is depicted in Figure 4. The interaction between the Agent and the Rule Engine is through Java Beans. Therefore, in our case, the *LicenseBean* is a *Fact*. The *Facts* needs to be inserted into the working memory of the Rule Engine, *before* any of the rules are fired [10]. We use the *JBoss Java based* ***Drools*** *Rule Engine Version 4.0*. We store the rules in a *.drl file*, a sample of which is given in Figure 4.

C. The Enterprise Web Serivces

The *Enterprise* offers ***three Web Services*** all of which is a *SaaS*. Out of this *two* of them are the *Generic Web Services*, which we focus in this paper. The Web Services are:

- Registration and Licensing Web Service
- Result Reporting Web Service
- Task Scheduling Web Service

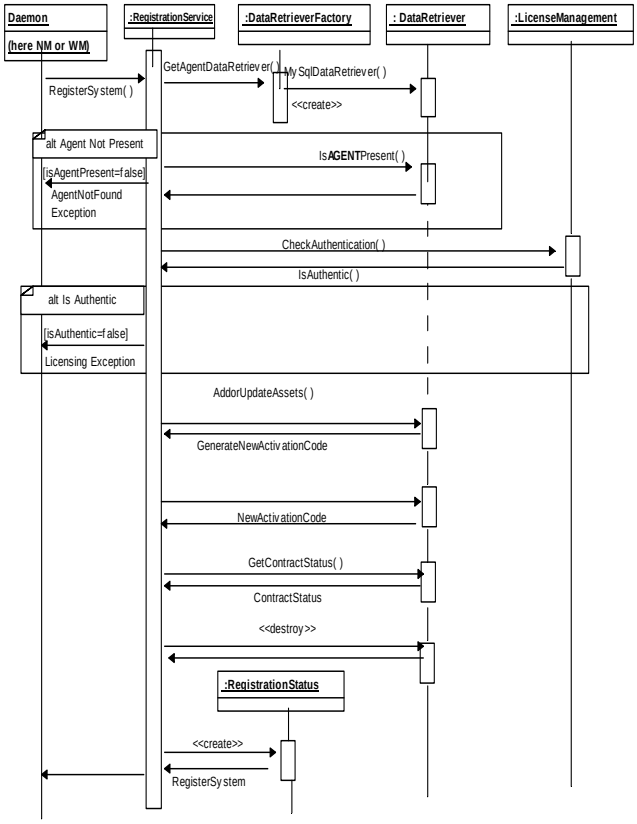

Figure 5. Sequence Diagram of the Generic Registration and Licensing Web Serivce

The ***Enterprise Communicator*** component of the *Software Agents*, depicted in Figure 2 above, have the client

ISSN : 0975-3397

components of these ***SOAP (Simple Object Access Protocol)*** based Web Services deployed in it. This is the one, which enables the Software Agent to communicate with the Enterprise Web Services, through SOAP over HTTP.

As shown in Figure 1, the *Result Reporting* and *Registration Web Service* are reusable because of their generic design. In our work, we have used these ***same WSs*** for both the *Network* and *Weather Monitoring* backend applications. The *NM* and *WM Agent* registers themselves with the Enterprise using their respective Registration Managers, depicted in Figure 2. The Agent's Registration Manager in turn invokes the *Registration and Licensing Web Service*, whose sequence diagram of execution is given below in Fig 5.

In Figure 5, we have shown the sequence of interaction that takes place for the *Registration and Licensing Web Service*. We use a combination of *factory method* and *abstract factory design pattern* to decide which backend application needs to be invoked. Due to the use of these generic design patterns, the Web Service is reusable. Any backend monitoring application that needs an online monitoring component can interact with these generic Web Services.

The detailed algorithm of the Agent interacting with this service is given in Figure 3. We have tested this framework with two backend applications *Network Monitoring* as well as *Weather Monitoring*, specifically *Temperature Monitoring Applications*. Both these applications are Software Agent-Based and it is the Agent, which acts as a bridge between the legacy / backend application and the Generic Web Services. The execution results are shown in Section 3.

III. EXECUTION RESULTS

The following figures depict the execution results of using the *Registration and Licensing SOAP Based Web Services* for ***both*** the *Online Network* and *Weather Monitoring Applications.* Figure 6 shows the interface to register a Network Monitoring Software Agent, while Figure 7 depicts the registration and licensing GUI for a Weather Monitoring Agent. It is evident from these figures that the *Unique Device ID*, distinguishing the Software Agent of the backend application are different implying that it is two different Agents we are trying to register. Also note from these figures that the Services offered by both the Agents are different.

The NM Agent offers

- Account Monitoring Services
- Performance Monitoring Services
- Fault Monitoring Services

while the WM Agent offers

- Pressure Monitoring Services
- Temperature Monitoring Services
- Humidity Monitoring Services

Figure 6. Online Network Monitoring Licensing GUI

Figure 7. Online Weather Monitoring Licensing GUI

Thus, when the *Software Agent*, which is tightly coupled with the backend application, registers itself with the Enterprise, it makes available to the user, the services offered by the backend application. With these results, we prove that the *Registration and Licensing Process* is ***Generic***, independent of the backend application. Any other such legacy applications, which needs to be scaled up for online monitoring over the internet, could ***re-use these Web Services***. The Services offered by the backend application is thus available to the end user of the online monitoring application depending on the licensing agreement and payment terms of using these ***Softwares as a Service(SaaS).***

ISSN : 0975-3397

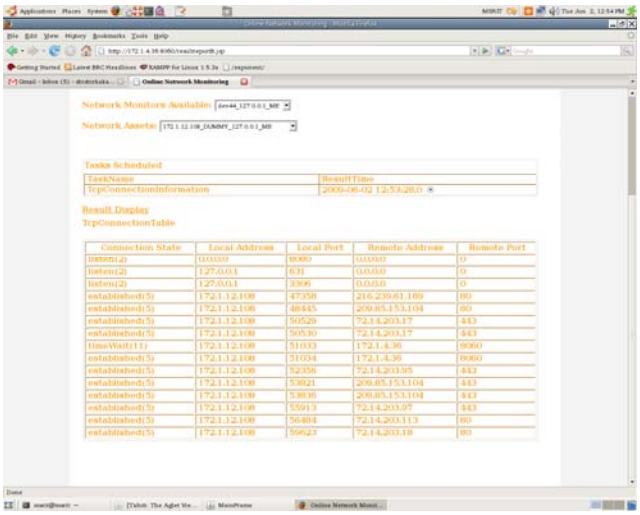

Figure 8. TCP Connection Table Using Generic Reporting Web Serivce

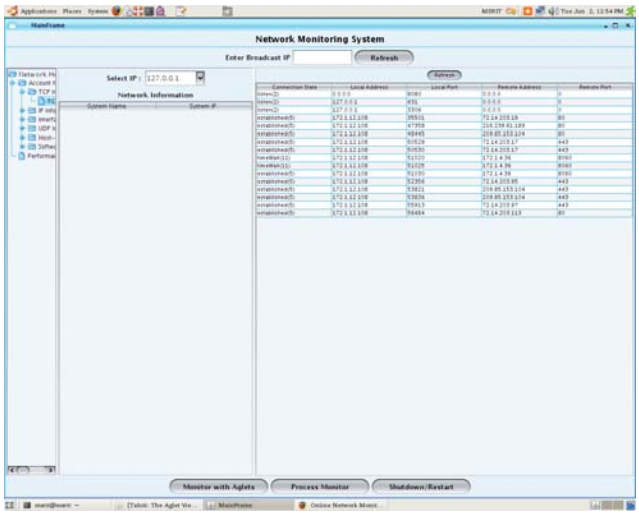

Figure 9. TCP Connection Table of the Backend Application

```
<definitions
targetNamespace="http://ResultReporter.WebService/"
name="ResultReportingServiceProviderService"
xmlns:tns="http://ResultReporter.WebService/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns="http://schemas.xmlsoap.org/wsdl/">
<types>
<xsd:schema>
<xsd:importnamespace="http://ResultReporter.WebService/"schemaLocatio
n="ResultReportingServiceProviderService_schema.xsd"/>
</xsd:schema>
</types>
<message name="ReportResult">
<part name="parameters" element="tns:ReportResult"/>
</message>
<message name="ReportResultResponse">
<part name="parameters"element="tns:ReportResultResponse"/>
</message>
<portType name="ResultReportingServiceProvider">
<operation name="ReportResult">
<input message="tns:ReportResult"/>
<output message="tns:ReportResultResponse"/>
</operation>
</portType>
<binding name="ResultReportingServiceProviderPortBinding"
type="tns:ResultReportingServiceProvider">
<soap:binding transport="http://schemas.xmlsoap.org/soap/http"
style="document"/>
<operation name="ReportResult">
<soap:operation soapAction=""/>
<input><soap:body use="literal"/></input>
<output><soap:body use="literal"/></output>
</operation>
</binding>
<service name="ResultReportingServiceProviderService">
<port name="ResultReportingServiceProviderPort"
binding="tns:ResultReportingServiceProviderPortBinding">
<soap:address
location="http://localhost:8060/ReportingResultNM/ReportingResu
ltNMServicePortTypeBndPort"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"/>
</port>
</service>
</definitions>
```

Figure 10. WSDL Created for the Generic Reporting Web Service

```
<xs:schema version="1.0"
targetNamespace="http://ResultReporter.WebService/"
xmlns:tns="http://ResultReporter.WebService/"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:element name="ReportResult" type="tns:ReportResult"/>
<xs:element name="ReportResultResponse"
type="tns:ReportResultResponse"/>
<xs:complexType name="ReportResult">
<xs:sequence>
<xs:element name="results" type="tns:resultDetails" minOccurs="0"/>
</xs:sequence>
</xs:complexType>
<xs:complexType name="resultDetails">
 <xs:sequence>
 <xs:element name="AssetId" type="xs:string" minOccurs="0"/>
 <xs:element name="ResultValue" type="xs:string" minOccurs="0"/>
 <xs:element name="ScheduledTaskId" type="xs:int"/>
 <xs:element name="DeviceInfo" type="tns:deviceIdentification"
minOccurs="0"/>
</xs:sequence>
</xs:complexType>
<xs:complexType name="deviceIdentification">
<xs:sequence>
<xs:element name="AuthenticationToken" type="xs:string"
minOccurs="0"/>
<xs:element name="DeviceIdentifier" type="xs:string"
minOccurs="0"/>
<xs:element name="CreateLocalTimeGMT" type="xs:dateTime"
minOccurs="0"/>
</xs:sequence>
</xs:complexType>
<xs:complexType name="ReportResultResponse">
<xs:sequence>
<xs:element name="return" type="xs:boolean" minOccurs="0"/>
</xs:sequence>
</xs:complexType>
</xs:schema>
```

Figure 11. XSD Created for the Generic Reporting Web Service

Figure 8 and Figure 9 execution results shows that the backend network monitoring desktop application in indeed scaled up for online monitoring over the Internet. The results retrieved for a TCP Connection Table of a node is exactly the same when retrieved by the desktop application (Figure 9) and

ISSN : 0975-3397

when retrieved by the Enterprise Reporting Web Service (Figure 8). Figure 10 shows the ***WSDL (Web Services Definition Language)*** created for the Reporting Web Service, while Figure 11 shows its ***XSD*** (***XML Schema Definition***). The detailed attributes and elements of the messages 'ReportResult' and 'ReportResultResponse' in the *WSDL* are defined in the *XSD* as shown. Thus, the Java Objects created and filled by the Software Agents are converted into XML form, put into SOAP envelopes, and exchanged between the Enterprise and the Agent through SOAP over HTTP. This creates the overhead of having to marshal and unmarshall the SOAP Messages, but provides a standard meta-data in the form of XML to exchange information. This makes the framework very generic, meaning any kind of data pertaining to diverse backend application can be exchanged as finally it is sent in the form of XML.

IV. CONCLUSION AND FUTURE SCOPE

The *initial phase* of this work was demonstrated in ***SUN Tech Days*** held at ***Hyderabad*** in ***February 2009***. There we demonstrated the use of the Enterprise-Agent SOAP Based Web Services Framework to scale-up a legacy Network Monitoring Application *("in press"[9])*,which we had developed in-house. We have *extended this work* to demonstrate with implementation results the proof that our Framework indeed provides *Software as a Service* with multi-tenancy and therefore ***SaaS Level 3 Maturity***, where multiple applications(tenants) use the *same code-base*(in our case the two generic web services, namely *Registration* and *Reporting*). The two applications reusing the Web Services were *Network* and *Temperature Monitoring*.

Currently, though the second backend application is designed to be a *Weather Monitoring Application*, does only *Temperature Sensing*. Work is in progress to extend it to other monitoring domain such as *Pressure* and *Humidity Sensing*, all of which will use our Generic / Reusable Web Services thus providing ample proof of multi-tenancy.

We also hope to employ Cluster Computing Techniques to our Framework thus extending the solution to be a complete Platform as a Service along with Generic Software as a Service thus aligning our work towards a Greener Computing Paradigm and Framework oriented towards Cloud Computing.

ACKNOWLEDGMENT

We would like to thank ***SUN Microsystems*** for providing us *Sun Spot Devices* for Academic Research. We are indebted to the Technical Architects and the University Relations Team of SUN Microsystems, Bangalore for making us adhere to strict milestone reviews conducted by them. This ensured that the quality of our research work is maintained, helping it to be selected for demonstration for the ***University Track of Sun Tech Days***. We would also like to thank Keerthi Ramnarayan, Akhil Ravindran, Shishir M Kakaraddi and Suhas D for their valuable technical inputs and the numerous design and implementation related assistance provided. We are also indebted to Dr.Aswatha Kumar for providing the facilities of High Performance Computing, Sun Technologies and other Labs, which we could use as test-beds for our work.

REFERENCES

[1] Richard Dim, "Why Software as a Service: Helping Customers Reduce Costs and Increase Revenue", *White Paper by OpSource:The Business of Web Operations.* http://www.opsource.net

[2] Trumba Corporations, "Five Benefits of Software as a Service", *Trumba White Paper, Feb 2007*

[3] Oracle Corporation, "SaaS Data Architecture", *An Oracle White Paper, October 2008*

[4] GianPaolo Carraro, Fred Chong, Eugenio Pace, "Efficient Software Delivery Through Service Delivery Platoforms", *The Architectural Journal, pp 7-13, Journal 12*. http://www.architecturejournal.net

[5] Erich Gamma, Ralph Johnson, Richard Helm, John Vissicles, "Design Patterns: Elements of Reusable Object Oriented Software", *E-Book*, http://www.abook2read.com/

[6] Tameer Nasser, Murali Vrindachalam, "Software as a Service: Build a Web Delivered SaaS Framework for forms and work-flow driven applications", *IBM White Paper, Dec 2008.* http://www.ibm.com/developerworks/architecture/library/ar-saasframe/

[7] Accelarance and OpSource White Paper, "Transforming your Software Product into a Service", *by OpSource:The Business of Web Operations.* http://www.opsource.net

[8] O. P. Rishi1, Ashok Sharma, Archana Bhatnagar and Ashutosh Gupta, "Service Oriented Architecture for business dynamics: An Agent Based Approach", pp.19-28

[9] Mydhili K.Nair, Chandan Bhosle, V. Gopalakrisha, "Net Mobile-Cop: A Hybrid 'Intelli-Agent' Framework to Manage Networks", *presented at the IEEE International Conference on Intelligent and Multi-Agents(IAMA2009). In Press.*

[10] Mydhili K Nair, Shishir M Kakaraddi, Keerthi M Ramnarayan, V Gopalakrishna, "Agent with Rule Engine: The 'Glue' for Web Service Oriented Computing applied to Network Management System", *accepted for publication in the proceedings of IEEE International Conference on Service Computing(SCC2009)*

[11] Hui Xu, Student Member, IEEE, and Debao Xiao, "Applying Semantic Web Services to Automate Network Management", *Proceedings of the 2nd IEEE International Conference on Industrial Electronics and Applications, pp.461-466, May 2007.*

[12] Aimilios Chourmouziadis, George Pavlou, "Web Services Monitoring: An Initial Case Study on the Tools Perspective", *Proceedings of IEEE Network Operations and Management Symposium, NOMS 2008, pp.827-830, April 2008.*

[13] Ricardo Lemos Vianna, Maria Janilce Bosquiroli Almeida, Liane Margarida Rockenbach Tarouco, Lisandro Zambenedetti Granville," Investigating Web Services Composition Applied to Network Management",*Proceedings of IEEE ICWS 2006, Chicago, USA, pp.531-540, September 2006.*

[14] Torsten Klie, Adrian Belger, Lars Wolf," A Peer-to-Peer Registry for Network Management Web Services", *Springer LNCS, Volume 4531/2007, pp. 133-138, June 2007.*

[15] Herwig Mannaert , Paul Adriaenssens ,"Web Services Based Systems for Network Management and Provisioning:",*Proceedings of Advanced Industrial Conference on Telecommunications Service Assurance, IEEE Computer Society, pp. 442-445, July 2005.*

ISSN : 0975-3397