© 2014 IJIRT | Volume 1 Issue 8 | ISSN: 2349-6002

Troubleshooting in Software as a Service (SaaS) Environments using Multi-Agent Technology

PPG Dinesh Asanka[1], Asoka S. Karunananda[2]
Department of Computational Mathematics
University of Moratuwa, Moratuwa
Sri Lanka

Abstract*— Troubleshooting in Software as a Service (SaaS) environment is inherently complex due to involvement of multiple sub-systems, which operate under lots of uncertainty. A research has been conducted to develop a Multi Agent System (MAS) for troubleshooting in SaaS environments by implementing an agent for each subsystem. Each agent has its personal ontology with the sub-system specific knowledge, whereas the domain ontology comprises of commonly accessible knowledge. In presence of an event requiring troubleshooting, the agents start communicating with each other and arrive at a globally acceptable solution. The communications among agents ensure the proper coordination and negotiation so that resource utilization can be optimized within the environment. The MAS solution has been evaluated with real world SaaS operation in a medium scale software development company. The experimental results show that Multi Agent solution for troubleshooting in SaaS can generate more accurate solutions in a lesser time.

***Index Terms* — Multi Agent Systems, SaaS, Message Space Agent, Ontologies**

I. INTRODUCTION

Software as a Service (SaaS) is a software delivery method that provides access to software and its functions remotely as a Web-based service. Software as a Service allows organizations to access business functionality at a cost typically less than paying for licensed applications since SaaS pricing is based on a monthly fee or some sort of subscription for the service that are utilized by end users. Also, because the software is hosted remotely, users do not need to invest in additional hardware. Software as a Service removes the need for organizations to handle the installation, set-up, disaster recovery, high availability and often daily upkeep and maintenance [1], [22].

SaaS is a software distribution model, designed mainly for web delivery, user can deploy and access through the Internet hosting. SaaS providers need to build information for all network infrastructures, software, hardware, operating platform, and is responsible for the implementation of all post-maintenance and other services including disaster recovery. When Software and hardware and people resources are correctly associated, deploying a SaaS application becomes the more cost effective option in many cases [23]. Compared with the traditional method of service, SaaS not only reduces the cost of traditional software licensing, and vendors deploy application software on a unified server, eliminating the end-user's server hardware, network security devices and software upgrade and maintenance expenses, the customer does not need other IT investment in addition personal computers and Internet connections to obtain the required software and services [2]. Typical SaaS includes many sub-systems such as network, databases, storage, servers, and applications are interconnected, distributed and undergo changes. When there is an issue with the one of the sub-system or small change in the one environment, it will be significantly visible at some other sub-system which is called butterfly [3].

In today's complex IT environments, it doesn't take much time to cause a high impact incident. Any minute misconfiguration or omission of a single configuration parameter can quickly lead to an incident with high impact: reputation damage, dissatisfied customers, financial losses, legal liabilities, and full re-organization. Productivity drops drastically as IT incident management teams are transformed into a group of 'firefighters', running against time to stabilize high-priority crises [4].

In today's fiercely competitive environment, there are few processes introduced by different organizations such as ITIL. Most of the recommend process is to discuss between sub-system experts to have a discussion to resolve the issue.

So, in case of an issue, these sub-systems need to be communicated between them to identify the cause of the issue. This kind of solution is difficult to produce with traditional software technologies, because of the limits of these technologies in coping with dynamically changing and unmanaged environments. It would appear that agent-base technologies represent a promising tool for deployment of such applications because they offer the high level software abstractions needed to manage complex applications and because they were invented to cope with distribution and inter-operability [5].

Though there are different practices proposed by organizations like ITIL, there is no standard method to troubleshoot issues in SaaS system. Different teams use their own customized ways to troubleshoot. However, by using properties like emergent, butterfly effect etc of Multi Agent Systems [3], system is proposed to troubleshoot issues in SaaS.

II. SAAS PRACTICES IN TROUBLESHOOTING

In case of SaaS system, there can be multiple sub-systems and these sub-systems will have heterogeneous supporting platforms. These sub-systems have lot of interconnected systems. For example, database sub-system might contain different databases with different versions and editions like SQL Server, Oracle, MySQL (Relational Database

Management Systems) and MongoDB, Cassendra, Neo4j (NoSQL). Similarly, multiple operating systems like Windows and Linux can be operated in the system. Storage can be in different types. In case of SaaS there should expert knowledge on different supporting platforms as each platform needs subject matter expert (SME) knowledge.

In SaaS, there can be three types of issues, Incidents, Problems and Errors [6].

Incident is any event that is not part of the standard operation of a service and causes or may cause an interruption to or reduction in the quality of that service. A problem is an unknown, underlying cause of one or more incidents. A single problem may generate several incidents. An error is a problem for which the root cause has been identified and a workaround or permanent solution has been developed. Errors can be identified through analysis of user complaints or by vendors and development staff prior to production implementation [6].

The key concepts and language of incident and problem management are shown in Figure 1 as indicated by ITIL. There is a lifecycle relationship among incidents, problems and errors: incidents are often the indicators of problems; problems lead to the identification of the root cause of the underlying error; errors are then systematically eliminated [6].

Incident Management (IM) refers to activities undertaken to restore normal service operation as quickly as possible while minimizing adverse impact on business operations. IM is a reactive, short-term focus on restoring service as shown in Figure 1 [6].

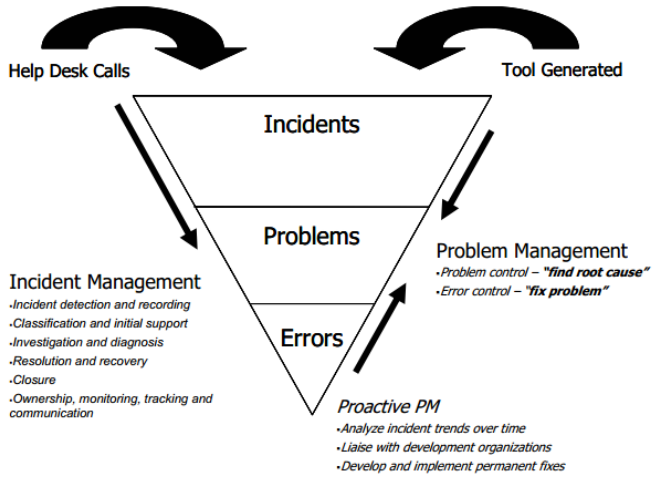

Figure 1: Incident Management Process [6]

Since incident managers need to communicate with other relevant resources, there are few mechanism used in the industry. War room is one of the main mechanisms used. A war room is a central location where members of the resource team are available. These members man the phones, answer questions, take down issues, and work issues. It is good to have a war room staffed with whiteboards, markers, phones, and food. Resources will be available until IM declared that the issue is resolved [7].

Different organizations use different mechanisms for War room depending on the scale and the geographical location of the resources. If resources are available with the same premises, teams tend to have war rooms as verbal discussions. If the teams are geographically separated, it will be either conference calls or message boards broadcasting. In message board broadcasting, team will broadcasts the message where other members will acknowledge and responds [7].

Whatever the technology it uses, most of these processes can be modeled with Agent technologies. During the war room, after the discussion among the team members new knowledge will evolve. Also, most of the resources are autonomous as those resources have expert skills and knowledge on their relevant subject area and need little information to work.

Pro-active and reactive are another two important features of agents. During the war room, resources will react to the questions from other resources, where as if these resources found any issues with relevant to their own sub-system, they will share the issue with other resources. Then, all resources will troubleshoot issues with respect to the concern raised by the other teams.

III. Environment of SaaS

Number of dimensions was identified along which task environments can be categorized. These dimensions will determine the appropriate agent design and the applicability of each of the principal families of techniques for agent implementation [8].

In SaaS environment, it is not practical to collect information all the time. Most of the time, data collection is time base. For example, in the database systems, we need to capture CPU percentage of the database instances. However, it is not possible to retrieve the CPU continuously. Most of the systems capture this information periodically, typically in five minutes intervals. So between these intervals, vital information can be lost or missed. On the other hand, some of the parameters may not be able to collect during the all the time due to the nature of the parameter. This means that there can be situations where data capturing on different sub-systems are slower and system will receive data after some time. By considering both factors SaaS, environment is partially observable. In order to behave truly effectively in a partially observable environment, it is necessary to use memory of previous actions and observations to aid in the disambiguation of the states of the world [9].

If the next state of the environment is completely determined by the current state and the action is executed by agent, the environment is said to be deterministic [8]. In the SaaS environment, next state events cannot be determined by the current events as there are lots of changes occurring in different sub-systems. Also, change of one sub-system will affect one or many sub-systems. This means that in the SaaS environment, current events cannot be determined by the previous events. Therefore, SaaS environment is stochastic.

In an episodic task environment, the agent's experience is divided into atomic episodes. Each episode consists of the agent perceiving and then performing a single action [8]. In case of SaaS environment, there will be a sequence of events occur from different sub-systems. Therefore, SaaS is sequential.

If the environment can change while an agent is deliberating, then the environment is said to be dynamic [8]. In case of SaaS, issues can escalate to different level during the deliberation. For example, when database sub-system is having high connections and while multi-agent database sub- system troubleshooting, CPU level might increase rapidly and when CPU is increased there can be other implications, hence SaaS is dynamic.

The discrete/continuous distinction can be applied to the state of the environment, to the way time is handled, and to the percepts and actions of agent [8]. With regards to SaaS, most of the situation they act as continuous.

Since there are lots of sub-systems and each sub-system works in several clusters and these sub-systems has to communicate between them [8]. Therefore, SaaS environment is obviously a multi-agent environment.

All the environment's parameters indicate that SaaS operates in complex environment as one might expect, the hardest case is partially observable, stochastic, sequential, dynamic, continuous and multi-agent [8].

IV. Features of Agents

There are mainly three types of agents involve in the proposed Multi Agent System.

- **Unit Agents running on each sub-system**

For each unit of the SaaS, agent will be running. For example, each database instance is allocated to a dedicated agent. This agent will capture relevant values for assigned parameters for given frequency. This means that unit agents are time based agents. Since for every unit has agent (unit agent) hence these agents are thin. Since these are thin agents, by executing these agents, it does not consume many resources from the system.

- **Agent Coordinator**

For each sub-system, there is a coordinator which will capture data from unit agents which is refereed earlier. Agent Coordinators can be clustered depending on any feature set. For example, databases, it can be configured to have one agent coordinator for all the databases. Similarly, different databases will have one agent coordinator. For example, SQL Server databases will have one agent coordinator, Oracle databases will have one agent coordinator and MySQL will have another agent coordinator etc.

Also, you can cluster agents with respect to system features as well. For example, transaction systems can have one agent coordinator while data ware houses will have another agent coordinator. Defining, agent coordinator will be up to the users depending on their SaaS environment.

- **Message Space Agent**

Message Space Agent is the main agent which will coordinate with sub-system agent. Until Message Space Agent decides that the issue or the incident is over, all sub-systems communicate to solve the incident.

Out of these agents, unit agents are time based agent whereas agent coordinators and Message Space agent are event base agents. Time based involves periodic sampling which leads to significant over-provisioning of network resources since the predetermined task period is determined by a worst situation time interval in order to assure the system performance [10].

In event-triggering agents the system state is sampled and transmitted when a certain internal measurement function exceeds a threshold [11]. Advantageously, the event-driven control improves the overall control system performance while maintaining the utilization rate of the communication resources. See example [12].

Following are the agent management services introduced to better management of the proposed multi agent system [13].

- **Configuring**

Different sub-systems have different parameters to check depending on the sub-system. For example, database sub-system checks database system for CPU [14], memory [15], number of connections and number of blocking queries etc. whereas storage sub-system checks SAN for port traffic, number of seconds per write and number of seconds read.

Also, these ranges are abnormal depending on the time of the day, day of the week or month of the year [16]. For example, during the peak load season high CPU for a server is normal whereas during the off-peak hours even medium CPU would be a concern and it needs to be addressed.

Also, these configurations should be different from server to another server since different servers are providing different features of SaaS. For example feature1 will be a highly used feature whereas feature2 is less used feature. Therefore, configurations for feature1 and feature2 are different from each other.

- **Initiate Agent**

In Multi Agent System, the level of dynamism allowed for adding new agents has a significant effect on the properties of the system. Multi Agent System allows agents to leave or enter the system dynamically, during run time, without any explicit message to all of the other agents in the system. The advantage of such openness is in the ability of the system to dynamically adjust itself to changes in the environment, tasks, and availability of capabilities and resources. This type of dynamism is important for Multi Agent System that is deployed in environments with high levels of uncertainty [17].

In SaaS system, infrastructure is not static as with the time new servers are added. Whenever new server is added to the SaaS, there should be option to add new unit agent to so that new agent should start communicate with the new server and provide information to the agent coordinator.

- **Execute the Agent**

Agents are extracting information from the sub-systems which the agent is allocated to. After the information is gathered it will send the information to coordinate agent. Then the coordinate agent will validate it with the configurations. If it is needed to be escalated to the message space agent (co-ordinate agent will decide after analyzing with the anthology), it will post the message to the message space agent. In case, message space agent or any other sub-system request for information, sub-system will post the current status irrespective of the current status. For example, during an incident, message space agent will ask for status update from all sub-systems which are involved in the incident. Then, database sub-system will send a message to the message space agent saying that there are no issues with the databases.

Also, there are coordination agents who will co-ordinate with the sub-system agent and message space agent.

- **Supporting the Agent Security**

Knowledge base can be divided into public and private with respect to accessibility of the knowledge base. Public knowledgebase can be accessible by any agent. However, only relevant sub-system agent should have the access to its private knowledge base and other sub-system agent should not have the access to other private knowledge base.

- **Suspending an Agent**

Though agents are theoretically needs to run 24x7, there can be instance where we need to suspend agent. During maintenance events like service pack updates, infrastructure upgrades, code releases and other events, relevant sub-system agents need to be suspended [13]. If these agents are not suspended during the maintenance event, unnecessary events will be escalated to the Message Space Agent. If other Subsystem agents are unaware about the maintenance event, those agents will act to trouble shoot which is will raise false alarms.

- **Restarting an Agent**

When the maintenance event or any other code releases are finished, suspended agent needs to be restated. This option would be helpful in case of internal issues in the agent.

- **Terminating an Agent**

As agents can initialize when there is new sub-system components are added, whenever sub-system components are removed from the system relevant agent should be terminated [13]. Also, coordinating agents should remove sub-system components from its coordination and communication.

V. Multi Agent System Architecture

Proposed Multi Agent System uses Blackboard model of problem solving proposed by H. Penny [18] in Message Space Agent will act as the blackboard while the agents will act as knowledge sources.

Figure 2 shows the system architecture of the proposed multi-agent system.

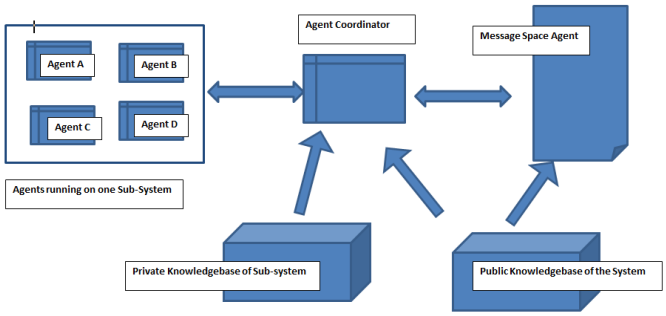

Figure 2: System Architecture for proposed Multi Agent System

Each sub system will be configured for swarm agent where each agent is responsible for acquiring the different parameters from each node at given intervals. For example, agent running on one database node is configured to acquire parameters like CPU, Memory and number of user connection etc. These agents are shown in “agents running on one sub-system” in the figure 2. These agents are sending data to its agent coordinator at a configured time interval which is by default five minutes.

Agent coordinator will raise an alert or event to the message space agent if the data received from the sub-agents has fallen to the critical state. Since different subsystems have different critical levels depending on the time, agent coordinator has to retrieve data from knowledge base.

Upon receiving a message, Message Space Agent will read public knowledge base and it will identify which sub-systems needed to be informed. Message Space Agent posts a message requesting relevant subs-systems to check for their sub-systems. Each relevant sub-system will acknowledge and respond to the Message Space Agent and send their information.

Message Space Agent needs to get the current status time to time depending on the impact of the issue. If it is high impact issue, it can request statuses in high frequency. To measure the customer impact, another agent can be deployed get the number of customer incidents.

These agents can recommend actions need to be taken to resolve the issue by reading public and private knowledge bases.

Until Message Space Agent declares that the issue is closed, all agents should be continuously engaged in troubleshooting.

VI. Sub-system Parameters

Different sub-systems measure or monitor different parameters to measure the health of each sub-systems.

TABLE I. Sub-System Parameters

Sub System	Measure
Application	Failing Scenarios
	Slowness
Application Servers	Memory
	CPU
	Server Availability
Databases	CPU
	Memory
	User Connections
	Blockings
	IO Request Time
	Server Availability
Storage	Seconds per Write
	Seconds per Red
	Port Loads
Network	Network Utilization
	Data Packet Loss
Client Services	Number of issues reported

If there are new parameters which are relevant to the sub-system should be able to add. Every sub-system unit agent may not collect all the parameters. For example, database systems might be configured to capture above parameters specified in the above table. However, in data ware house databases CPU and Memory will reach high level during the time of user access. Therefore, from date ware house database subsystems, CPU and memory parameters can be ignored [19].

When the parameters are introduced, it needs to be configured for the ranges and for different times as shown in figure 3.

VII. Ontologies

As indicated earlier, for each sub-system has its own private ontology which is accessible only to that sub-system. Figure 3 shows sample of threshold values for one database sub-system.

DB Inst	Para.	Range Value	Day Type	From Date Range	To Date Range	From Time	To Time	Low Range	High Range
DB01	CPU	Low	Weekday	1/1/2014	6/30/2014	00:00.0	06:00.0	0	30
DB01	CPU	Medium	Weekday	1/1/2014	6/30/2014	00:00.0	06:00.0	31	75
DB01	CPU	High	Weekday	1/1/2014	6/30/2014	00:00.0	06:00.0	75	100
DB01	CPU	Low	Weekday	1/1/2014	6/30/2014	00:01.0	00:00.0	0	40
DB01	CPU	Medium	Weekday	1/1/2014	6/30/2014	00:01.0	00:00.0	41	80
DB01	CPU	High	Weekday	1/1/2014	6/30/2014	00:01.0	00:00.0	81	100
DB01	CPU	Low	Weekday	1/1/2014	6/30/2014	00:01.0	59:59.0	0	30
DB01	CPU	Medium	Weekday	1/1/2014	6/30/2014	00:01.0	59:59.0	31	75
DB01	CPU	High	Weekday	1/1/2014	6/30/2014	00:01.0	59:59.0	75	100
DB01	CPU	Low	Weekend	1/1/2014	6/30/2014	00:00.0	59:59.0	0	20
DB01	CPU	Medium	Weekend	1/1/2014	6/30/2014	00:00.0	59:59.0	21	65
DB01	CPU	High	Weekend	1/1/2014	6/30/2014	00:00.0	59:59.0	66	100

Figure 3: Sample of Multi-Agent Configuration

Figure 3 is configured for one database instance (DB01) and for one parameter (CPU). In case of CPU, point of attention is varied depending on the date, time and week day or week end day. Since one Database server and one parameter has nine configurations for half a year and if there are ten database servers with three configuration there will be 360 configurations just for the database sub-system. When other sub-systems are considered, there can be large number of configurations.

The Unit Agent running on DB01, should have a mechanism to capture CPU of the DB01. To obtain CPU value for DB01, agent should have a mechanism. Similarly, for each parameter there should be a way to obtain current value for the sub-system. To obtain those values, Multi Agent System needs to keep connection information. Since these connections are security sensitive, connection information needs to be encrypted.

Unit Agent running on DB01, periodically captures data and sends data to Database Agent which is the agent coordinator for the database sub systems. Database agent coordinator receives all the information about database agents and if it is in the range of high, database coordinator will identify this as an issue and report to the message space agent. Message Space Agent will call for other relevant sub-systems to verify whether there are any issues with their own sub-systems.

Public ontology has the connection between different sub-systems. For example, when application sub-system reveals that there are failures in one of the scenario, then Message Space Agent needs to recognize what are servers involves with this application features. Then Message Space Agent will request for the health of the application servers. At the same time, Message Space Agent will request for health of the databases which is relevant to the failing sub-system. Also, Message Space Agent will request for network health of the system, which are connected to databases and application servers. If there is an issue with any of the sub-system, it will be sending the continuous updates to Message Space Agent until the issue is resolved. To troubleshoot issues between sub-systems, relations about different sub-systems should be maintained.

VIII. Evaluation

Some real world scenarios were identified to verify the implemented Multi Agent System.

- A database server restarted automatically.
  This incident is automatically recovered to its previous stage and no troubleshooting needed. However, since the database server was not available for 1-2 minutes, server availability parameter was triggered. With the unavailability of database triggered some applications to fail and application failure scenarios were also trigged. So during the incident database sub-system and application sub-system were initially communicated and later application server sub-system was called in to verify whether there are issues with the there are any issues with the servers.

Since this was recovered automatically, database server agent indicated to Message Space Agent that there are no issues with the database end. However, application scenarios were failing for little while and then recovered. Until they were recovered, Message Space Agent was receiving issues from the Application sub-system. Once the application sub-system verified that the all the issues were reverted, Message Space Agent indicated that the issue was resolved.

- High CPU in a Database Server.
  One of the databases CPU went from 80 – 96 % within 5 minutes. Database Coordinate Agent informed Message Space Agent that the server CPU is 90% and after around 1 minute Database Servers again indicated that blockings are high on one of the stored procedures released recently. With using the Multi Agent System, SaaS was able to indicate the issue very quickly.

  Apart from above two issues, two other issues also were able to rectified using Multi Agent System.

  It was identified that database systems taking more time for IO and during this incident storage and database sub-systems work together to trouble shoot.

  In another incident, one client scenarios are failing, application servers indicated the issue to the Message Space Agent and Windows server had some memory issues and was able to resolve the incident.

It was found that whenever the evaluation done, it needs more parameters to collect.

Whenever, there are other third party tools available as part of SaaS, they need to support proposed Multi Agent System. If not, issues related to third party tools will not be able to troubleshoot.

IX. Future Works

- Current research is only doing the troubleshooting of an incident. Root cause analysis is not part of this research. However, in typical route cause analysis, effected sub-system owners will get to gather and analysis the root cause so that this issue can be fixed permanently. Root Cause Analysis helps organizations avoid the tendency to single out one factor to arrive at the most expedient resolution [20]. For example, in the first evaluation which is database restarted automatically probably due to many reasons like, network drive issue, database bug, driver issue etc. By using Multi Agent technologies, it can be designed for root cause analysis as well.

- Currently configurations are static and users have the option of changing them manually which is not automatic. With the growth of the system it is obvious that these values should change time to time. These values can be predicted by using data mining algorithms such as time series [21].

X. Conclusion

This research is to implement Multi Agent System to troubleshoot issues and incidents in SaaS system. SaaS system environment is partially observable, stochastic, sequential, dynamic, continuous and multi-agent which is the most difficult combination of properties. Proposed Multi Agent System uses blackboard model where sub-system will post messages to the Message Space Agent. By using the private and public ontologies, Multi Agent System will identify the location of the troublesome sub-system.

Proposed Multi Agent System has the capabilities of initiate agents, suspend agent, terminate agent etc for better management of agent services. By evaluating the proposed system with the real world scenarios, it can be conclude that proposed system will be very much helpful to troubleshoot issues and incidents.

By using, Agent Technologies to troubleshoot issues in SaaS, it will be effective and efficient as less human intervention is needed for standard tasks.

References

[1] "SaaS - Software as a Service, Storage as a Service", webopedia, [Online]. Available: http://www.webopedia.com/TERM/S/SaaS.html [Accessed 14 02 2014].

[2] M.V. Luis, R. M. Luis, C. Juan, L. Maik. “A Break in the Clouds: Towards a Cloud Definition” . Computer Communication Review, vol.39, pp.50-55, 2009.

[3] George Rzevski, Petr Skobelev, "Emergent Intelligence in Large Scale Multi-Agent Systems", International Journal of Computers, Issue 4, Vlume 1, 2007.

[4] "ITIL Incident Management and Investigation“, EVOLVEN, http://www.evolven.com/solutions/itil-incident-management.html, [Accessed 21 03 2014].

[5]e Fabio Bellifemine, Agostino Pogg, Giovanni Rimassa, "Developing multi-agent Systems with a FIPA-Complaint Agent Framework“,Software-Practice and experience, 2001, pp: 103-128.

[6] "A Framework for Incident and Problem Management “, International Network Services, Victor Kapella, http://kwesthuba.co.za/downloads/02_ins_incident_management_0403.pdf, [Accessed 21 03 2014].

[7] "The War Room ", International Network Services, Elyse, http://www.anticlue.net/archives/000595.htm, [Accessed 21 03 2014].

[8] S. Russel, P. Norvig, "Agents" in Artificial Intelligence, A modern Approach, Second Edition, Pearson Prentice Hall, 2011, p. 39-42.

[9] Leslie Pack Kaelbling , Michael L. Littman, Anthony R. Cassandra , ""Planning and acting in partially observable stochastic domains" ", Artificial Intelligence 101 (1998) 99-134.

[10] Dong Xue , Sandra Hirche,, "Event-triggered Consensus of Heterogeneous Multi-agent Systems with Double-Integrator Dynamics".

[11] P. Tabuada, "Event-triggered real-time scheduling of stabilizing control tasks," IEEE Transactions on Automatic Control, vol. 54, pp. 452–467, 2007.

[12] K. A˚ Strom, B. Bernhardsson, "Comparison of riemann and lebesgue sampling for first order stochastic systems," (USA),
pp. 2011–2016, in Proc. 41st IEEE Conf. Decision and Control, 2002.

[13] D. Cowen, M. Griss, "Making Software Agent Technology available to Enterprise Applications", Software Technology Laboratory HP Laboratories Palo Alto, 2002.

[14] S. Bathige, PPG Dinesh Asanka, "Middleware Layer for Replication between Relational and Document Databases," International Journal of Engineering Research & Technology, Vol. 3 - Issue 5 (May - 2014).

[15] Peter Boncz, Stefan Manegold, Martin Kerstanm, "Database Architecture Optimized for the new Bottleneck: Memory Access", 25th VLDB Conference, Edinburgh, Scotland, 1999.

[16] "Peak Season Prep Guide: Preparing your Ecommerce Site for the Next Big Rush," rackspace, [Online]. Available: http://www.rackspace.com/knowledge_center/whitepaper/peak-season-prep-guide-preparing-your-ecommerce-site-for-the-next-big-rush [Accessed 20 03 2014].

[17] O. Shehory, "Architectural Properties of Multi Agent Systems", The Robotics Institute, Carnegie Mellon University , 1998.

[18] H. Penny Nii, "The Blackboard Model of Problem Solving and Evolution of Blackboard Architectures", AI Magazine Volume 7 Number 2,1986.

[19] Steve Rees, Naveen K Singh, Thomas Rech, Olaf Depper, Gang Shen, Roman B. Melnyk, "Best practices Tuning and monitoring database system performance", IBM, July 2013, pp.25.

[20] "Root Cause Analysis ", Washington State Department of Enterprise Services, [Online]. Available: http://www.des.wa.gov/services/Risk/AboutRM/enterpriseRiskManagement/Pages/rootCauseAnalysis.aspx [Accessed 02 05 2014].

[21] "Time Series Analysis: The Basics ", Site for the Next Big Rush," Australian Bureau of Statistics, [Online]. Available:
http://www.abs.gov.au/websitedbs/D3310114.nsf/home/Time+Series+Analysis:+The+Basics [Accessed 20 03 2014].

[22] S. Banerjee, S. Jain, "A survey on Software as a service (SaaS) using quality model in cloud computing", International Journal Of Engineering And Computer Science ISSN:2319-7242, Volume 3 Issue 1, January 2014 Page No. 3598-3602

[23] Software and Information Industry Association, "Software-as-a-Service; A Comprehensive Look at the Total Cost of Ownership of Software Applications", IBM, September 2006, pp.2.