International Journal of Computer and Communication Technology

Volume 2 | Issue 2 Article 7

April 2011

Distributed Computing for Ubiquitous Systems

Mona Mani
Ghaziabad, Uttar Pradesh, India, monachristian@gmail.com

Distributed Computing for Ubiquitous Systems

Mona Mani
Ghaziabad, Uttar Pradesh, India
monachristian@gmail.com

Abstract—**Distributed computing provides a strong foundation on top of which a powerful ubiquitous system can be realized. However, distributed computing is not solely sufficient for the ubiquitous systems –a completely new era of computer world which is not based on the conventional mainframes or PCs but on the virtually intelligent silent objects/devices used for day-to-day human activities. This article de scribes basics of ubiquitous systems at introductory level and discusses developing a ubiquitous system as an extension to DIICS (Distributed Intelligent Instrument Control System) – a temperature monitor and control system developed by t he author. Moreover, it also discusses significant role of Tini InterNet Interface - a technology used in DIICS, for embedded device support in ubiquitous systems. It also discusses how distributed system fundamentals best fits for ubiquitous systems. Furthermore, it discusses various distributed system goals and specifies how these goals can contribute to ubiquitous systems solely or partially. It also specifies respective changes, and demand for new or adapted architecture/platform, to accommodate it for an efficient ubiquitous system. Towards the end it describes Remote Process Call (RPC) and Event Notification model, two powerful distributed system techniques.**

Keywords-Distributed computing, Ubiquitous systems, Calm technology, Distributed system goals, RPC, Event Notification model

I. INTRODUCTION

Ubiquitous technology is a third important era in the world of computer technology [1]. In the realm of man and machine utilization, Mainframe, the first era involves so many people interacting with a single computing unit, followed by the second paradigm of personal computing where a human interacts with machines which works in unnatural way to the human being. Human Computer Interaction comes with an aid to accommodate the system as much as possible to the natural human tendencies. However, as a revolutionary approach, a ubiquitous system comes up with an idea of an omnipotent system virtually invisible through its integration with day-to-day objects. It provides an environment where everyday objects are made smart enough to sense and response human presence and activities in the most humanistic way.

As specified in the **Wikipedia**, **Ubiquitous computing** (**ubicomp**) is a post-desktop model of human-computer interaction in which information processing has been thoroughly integrated into everyday objects and activities. A person does ordinary activities using ubiquitous system made up of various computational devices in the form of 'smart' daily objects. [2]. The idea of ubiquitous computing as invisible computation was first articulated by Mark Weiser in 1988 at the Computer Science Lab at Xerox PARC [3].

Ubiquitous Computing as being a pervasive and omnipotent system may include anything and everything like home environment monitoring and controlling for variables like temperature, humidity, water capacity and level, pressure etc, kitchen auto inventory-purchase system management, food menu generator based on items availability, human identification and home security system, smart car, smart watch etc. Mark Weiser, the father of the Ubiquitous Computing well describes it with a virtual scenario of a smart home environment in "The Computer for the 21st Century" [4]. He also highlights the revolutionize power of ubiquitous computing in changing the fundamentals of the digital world and its influence on human lives by saying "Neither an explication of the principles of ubiquitous computing nor a list of the technologies involved really gives a sense of what it would be like to live in a world full of invisible widgets".

II. UBIQUITOUS SYSTEM AS AN EXTENSION TO DIICS

Ubiquitous space at a broad and general scale can think of as a core user's home, office and other remote locations dispersed globally. The ubiquitous system can think of as three integration levels as specified below,

1.1 Design and deployment of Smart components corresponding to daily human factors.
1.2 Integration and communication among this smart components distributed in small proximity of home/office.
1.3 Home/office components interacting with outside world.

The author has developed DIICS - an N-tire temperature monitoring and control system stepping parallel to the kind of applications that comprehend with the social and industrial demands for security systems, equipment automation, smart and entertaining home appliances, and environmental control systems. Integrated technologies, Distributed computing and embedded device support through Tini Internet Interface (TINI) [5], used in DIICS provides a powerful platform covering 1.1, 1.2 and 1.3 efficiently. TINI - a chip set developed by Dallas Semiconductors is a powerful tool with hardware as well as software support to design and deploy a wide verity of 'smart' devices. Considering DIICS as one of the units/sub-modules and extending it by adding various task specific modules corresponding to various human operations provide means to materialize a ubiquitous system.

Fig. 1 shows the DIICS n-tire client-server architecture where RMI (Remote Method Invocation) server serves central role for distributed computing and TINI is an embedded device technology to incorporate non-network type devices/sensors as network entities into the system network. Fig. 2 shows a ubiquitous architecture which is an extended approach to the DIICS architecture. A ubiquitous system can have client-server architecture with a hidden master system/server or else it can have a P2P architecture where all network entities are in equal level relationships without any master system over them.

A ubiquitous system needs a number of interaction screens generally for user interaction to serve input and/or information presentation in the most humanistic way. Location or position of these screens may vary depending upon the context. For example, a digital calendar can be mounted on wall or desktop; a door can have a security screen; a considerably big screen can take its place on room wall for weather information, stock exchange information or some entertainment touch screen tool; the master bed room may have a screen specifying position and activities or video streaming of children or other family members; the kitchen may have single or multiple different size screens to take cooking or cleaning instruction; some device/equipment mounted screens etc. A ubiquitous system may have different strategies for activation/deactivation of the screens like all-time visible, pop up screens based on current context with respect to environmental or external factors. More specifically, ubiquitous system entities based on its work-load distribution with respect to functional, presentation and data management for context awareness can be classified as,

1) Exclusive for presentation
2) Fully task performing
3) Task performing with presentation/interaction.

This ubiquitous characteristic best fits with that of the client-server architecture, well defined as "Client-Server is a special type of co-operative distributed computing architecture as the client(s) and server(s) co-operate to perform the total application by distributing its presentation, functional and data management tasks" [6].

Fig. 1 shows basic client-server communication model. Clients are generally requestors mainly dealing with the data presentation and servers are generally dedicated to be ready to accept the client requests, to perform the actions for the request and to send the reply back. Clients and servers can be located on the same processor, different multi-processor nodes, or on separate processors at remote locations. The client server architecture must have at least one client and one server. There can be more than one client and/or more than one server. In the multi server system each server generally stands for some task specific services.

Ubiquitous entities solely dedicated for presentation can serve as 'Thin client' client-server architectural approach. As specified on Wekipedia, "A thin client (sometimes also called a lean or slim client) is a computer or a computer program which depends heavily on some other computer (its *server*) to fulfill its traditional computational roles" [7]. Some ubiquitous presentation screens may require being intelligent enough to perform entry level validations or being extra intelligent functional entities leaving only data management on server known as database server.

A ubiquitous entity dedicated for application business logic can serve as an application server providing 3-tire architecture of presentation units, an application server and a database server as a whole. Ubiquitous system utilizes embedded system, where one or more entities can be fully dedicated to serve the central role to back the embedded system functional requirements. For example, in a TINI oriented embedded system, a 'TINI server' can play the central role in providing embedded device management system. A ubiquitous system can have an embedded device server in addition to presentation units/clients, one or more application servers, and one or more database servers.

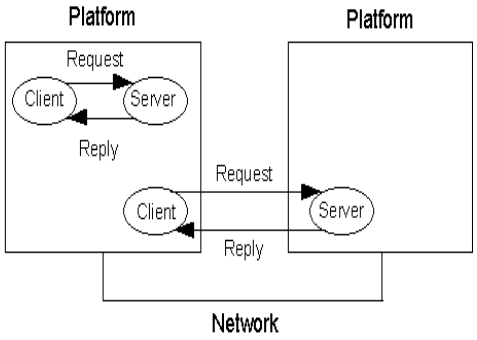

Figure 1: Basic Client/Server Communication Architecture [6]

Though TINI and similar technologies provide support to design and deployment of a vast range of equipments, a huge range of daily use objects/functioning is awaiting to incorporate them 'smartly'. Also, a ubiquitous oriented master system/platform at operating system, middleware, and application level for integration and communication can serve as a powerful development tool.

III. Distributed System Fundamentals and Ubiquitous System

A distributed system is a coherent collaborative system of integrated computer entities that performs distributed computing in which computer entities work together collectively to achieve an on-demand task. Wikipedia specifies, "A distributed system consists of multiple autonomous computers that communicate through a computer network. The computers interact with each other in order to achieve a common goal" [8]. Also, "Distributed computing is the process of aggregating the power of several computing entities to collaboratively run a single computational task in a transparent and coherent way, so that they appear as a single, centralized system"[9], and "Distributed computing is a programming model in which processing occurs in many different places (or nodes) around a network. Processing can occur wherever it makes the most sense, whether that is on a server, personal computer, handheld device, or other smart device" [10].

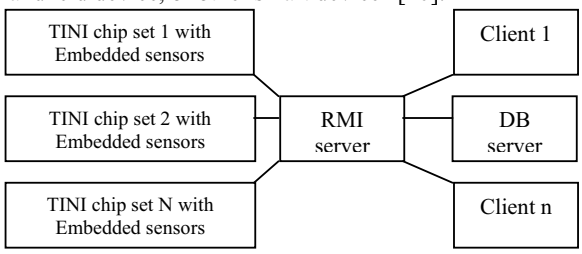

Figure 1: DIICS Architecture

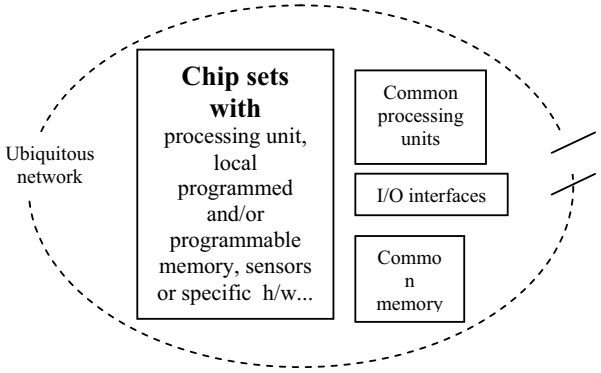

Figure 2: Ubiquitous Network

IV. Prepare Your Paper Before Styling

An important characteristic of a distributed system is to provide a single coherent system utilizing total power of all the available resources at a time. Ubiquitous network includes a large number of smart devices, dedicated for specific tasks oriented design and functionality which is different than that of other devices. Due to this unique characteristic devices may not be compatible to perform a collaborative task. The most basic characteristic of collaborative task performance does not sound suitable for ubiquitous network of high diversity over task-specific unique devices, then why to park ubiquitous systems on top of distributed systems?

Mark Weiser, the father of ubiquitous system emphasizes on the need of working towards the unique capability of physical diversity, saying, “Trends in "distributed computing" are to make networks appear like disks, memory, or other non-networked devices, rather than to exploit the unique capabilities of physical dispersion. The challenges show up in the design of operating systems and window systems” [4].

The most significant requirement of ubiquitous systems is the realization of such heterogeneous system. Distributed system’s powerful feature of efficient performing over the heterogeneity can serve as the basics of ubiquitous systems. Piece of software performing operating system (and/or middleware) can be realized to manage the network diversity and to provide a repository of devices available to perform on-demand collaborative task at a time.

V. Distributed System Goals and Ubiquitous Technology

Features like transparency, scalability, fault tolerance etc serves the basics of ubiquitous systems. This section includes discussion for their intact or adaptive usefulness to the ubiquitous systems.

A. User Connectivity and Resource Sharing

Distributed systems aims to connect various users and provide them an easy and controlled way to access and share various resources transparently. When it comes to the user connectivity, Ubiquitous system is fundamentally different with the basic aim of emphasizing on the day-to-day functions of a single or a group of persons’ life style. It mainly deals with the connectivity of the smart and unique devices which are different than mainframe/PC. All components of a Mainframe/PC enters in on/off state through a single switch event, but ubiquitous subunits/devices requires activation/deactivation as and when required based on some real time parameters or events than the whole system at a time. Ubiquitous system features a highly dynamic special kind of ubiquitous space, and thus resource management based on connectivity and sharing of smart heterogeneous task oriented devices, can be one of the major goals of a ubiquitous system.

A TINI with its basic characteristic of providing a task-oriented network entity with unique identification through Ethernet address with IEEE registered MAC ID can serve vital role by providing mechanisms for an efficient ubiquitous network/space management system. TINI enables easy addition/removal and activation/deactivation at subunit/equipment level. Distributed systems which allow pooling of the resources including CPU cycles, data storage, I/O devices can be utilized as an efficient platform to realize resource management.

Fig. 3 shows user connectivity in DIICS. In a client-server based ubiquitous system request generated by an entity, based on user interaction or some internal/external change factors can be served by a server(s) through calls or callbacks. Callbacks are a high level depiction of event-notification model explained in section 6.

B. Scalability

Scalability is a term used to measure system expansion-extension capacity. Tanenbaum (1985) citing Neumann [11] defines three different scales to find the scalability of a system. First measure to find system scalability is size scalability that how easily more users and resources can be added to the system. Second, a geographically scalable **system** is one in which the users and resources may lie far apart. The third measure is the administrative scalability to find how easily the system still can be managed even if it spans many independent organizations. However unfortunately, system often exhibits loss of performance as the system scales up in one or more of these three dimensions.

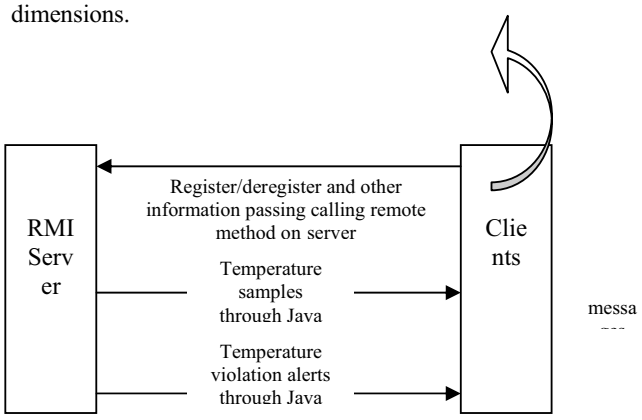

Figure 3: DIICS connectivity and sharing

A distributed system design for Local Area Network (LAN) that uses synchronous communication is difficult to scale geographically into a Wide Area Network (WAN) distributed system which requires asynchronous communication. Also, for the application demand where further processing depends on the result/outcome of the current request synchronous communication is inefficient.

A ubiquitous system where daily human operations/functioning are the foremost priority, home/office centric LAN communications are likely to be at considerably high frequency. At the same time, it would be a short vision to think a ubiquitous system limited to a small proximity like a house, in today's world of globalization where a life style is greatly involved in the long distant social, business, entertainment, tours and travel etc. affairs. Considering a general scenario where most of the operations/events are local home/office centric and the core user utilizes various service utilities for global activities, a ubiquitous system generally requires high frequency short distance, as well as global communications. However, the frequencies of local communication and global communication are highly human life style centric depending on various personal, social, regional, cultural etc aspects. These aspects require ubiquitous system to emphasis more on networking and communication architectures that provides efficient local as well as global both communications than on the geographical scaling.

Analysing target users' communication frequencies categorised in in-house, remote-LAN and global-WAN helps realizing efficient communication architecture and/or to accommodate it to the application specific requirements. For the systems where such analysis can not be done/predicted in prior, a real-time efficient and intelligent communication system, which can sense/analyze on-demand communication type and deploys synchronous or asynchronous communication with batch processing and parallelism, is required to serve the purpose. Replica and cashing are such related communication issues that come into the considerations for various applications.

A local ubiquitous system is less difficult to scale at administrative level, but administrative standardization and scaling for a system based on various communities diverse over various cultures and regions is difficult over global perspectives. It may conflict with different issues like policies regarding resource usage, management and security.

Size scaling, in a ubiquitous system where addition of new nodes is all to gather a new approach, is an important goal as its components are apart from the conventional frame of nodes/computers. Architecture must provide easy ways to change the physical level without need of much change in the logical view or business logic. As discussed in the section III.A, A TINI with its basic characteristic of providing a task-oriented network entity provides scalability via easy addition/removal of any unit/equipment.

C. *Transparency*

Distributed system aims transparency by providing its users a view of a single coherent system hiding the underneath complexity. At programming level, object oriented programming languages like Java simplifies the development of distributed ubiquitous system with powerful features like transparency and openness. A ubiquitous system can utilizes advantage of transparency. In addition to that, a ubiquitous system also known as Calm technology, aims to provide a virtually natural system by hiding the awareness of the fact of the machines being used can also be thought of as a transparency at different level. This can be achieved through making day-to-day equipments/objects smart. Applicability of distributed system and TINI embedded device which supports distributed computing has been discussed in various sections of this article. A number of basic transparencies for distributed systems, defined by Tanenbaum and Renesse [11], can further be explored with respect to ubiquitous systems.

D. *Openness*

Tanenbaum and Renesse defines open distributed systems as, "An Open Distributed System is a system that offers services according to standard rules that describe the syntax and semantics of those services" [11]. Openness can be a powerful tool in development of an efficient ubiquitous system. Moreover, standardization and compatibility supporting design, development and deployment of daily human operation based components can serve vital role providing interoperable, portable, flexible, and scalable (shrinking and expansion both) ubiquitous system. Standard task specific components, based on various lifestyle factors like personal, social, regional, cultural etc, can allow users to choose their own specifications to construct or to scale a ubiquitous system.

E. *Fault Tolerance*

As defined by Gartner, "The term fault is usually used to name a defect at the lowest level of abstraction, e.g., a memory cell that always returns the value 0" [12]. As ubiquitous systems aims usability of daily human life operations, some components like fire alarm, security systems etc that deals with the life and death of human beings must be fault tolerant. Fault tolerance over specific components rather than entire system will serve as an efficient approach. Fault tolerance of a ubiquitous system should be flexible enough to accommodate it according to application needs without much changes in respective other views of physical, logical or business logic. As discussed in the section III.A, A TINI with its basic characteristic of providing task-oriented network entities facilitates to provide support to fault tolerance at unit/equipment level.

VI. Remote Procedure Call – a Distributed Computing Communication Technique

Providing high degree of transparency to it programmers RPC looks like an IPC (Inter Process Communication) made on the same/local computer hiding the complexity of remote communication. Network communication in a distributed system occurs through RPC. Wikipedia defines RPC as, "Remote Procedure Call (RPC) is a protocol that allows a computer program running on one computer to cause a subroutine on another computer to be executed without the programmer explicitly coding the details for this interaction. When the software in question is written using object-oriented principles, RPC may be referred to as remote invocation or remote method invocation" [13].

Traditionally RPC is a synchronous operation in which the client procedure will be suspended until the remote procedure returns result/status data. In addition to transparency, it also provides another powerful feature of concurrency through asynchronous lightweight processes known as threads.

Fig. 4 shows RPC functioning, where RPC stub comprising of client stub module and server stub module initiates a RPC requests, builds the messages through coding and encoding of arguments/return values of message calls/replies. Registry known as binding server/agents binds client and server and serves the need for locating the server for a client's RPC request.

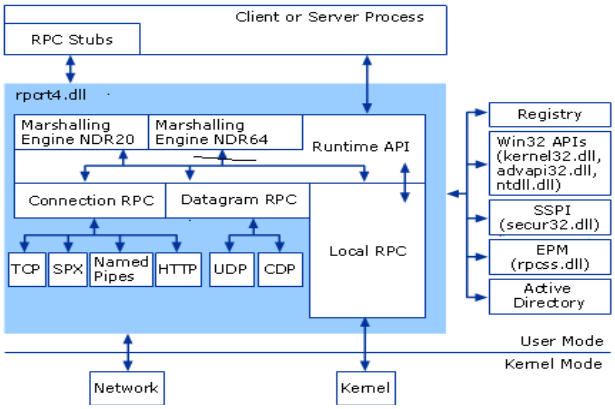

Figure 4: RPC Architecture [14]

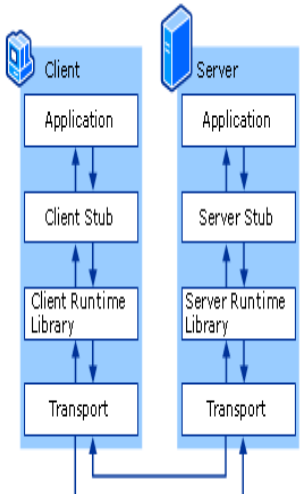

Figure 5: RPC Process [14]

Fig. 5 shows RPC process. Following are the steps in a typical processing cycle.

1. Client procedure calls client stub in normal way
2. Client stub builds message by marshalling the call parameters in to the partially filled header packet buffer obtained by client procedure.
3. The RPC run time obtains the remote object reference and calls the local Operating System (OS).
4. The client's OS sends the message to the remote OS using the remote object reference.
5. The server OS sends the message to the server stub.
6. The server stub unpacks the parameters and calls the server.
7. The server executes the procedure and sends the result back to the server stub.
8. Server stubs packs the result/status data and sends the built message to the local OS.
9. The server OS sends the result message to the client OS.
10. The client OS gives the message to the client stub.
11. The client stub unpacks the result message and sends the result to the client.

VII. Event and Notification Model in RPC

Traditional RPC communication mechanism, also known as Polling is inefficient for the applications where client (service requestor) is required to contact the server (service provider) contiguously to get information updates. Output units like LCDs/PDAs in a ubiquitous system may frequently require to be updated based on some internal/external information source. With the traditional approach, in such type of applications, clients can be implemented with an infinite loop prompting the server after a specified quantum of time. A small quantum may end up with a client requesting and/or updating the unchanged information unnecessary, and a large quantum may insert a delay in keeping the client up to date for the information change, possibly leading to a serious inconsistency. An efficient approach is to implement a mechanism through which the server sends information updates to the clients whenever information change occurs. The basic idea is to reflect a set of remote objects based on object changes occurring through some external/internal source. Publishers-Subscribers concept based Events and Notification model implements this functioning.

VIII. Conclusion

N-tire Client-server based distributed architecture best fits for the ubiquitous requirement of work load distribution at functional, presentation and data management level among various network entities. Network diversity and, resource management based on connectivity and sharing of smart heterogeneous task-oriented devices, are crucial aspects of ubiquitous systems. Distributed systems, with its powerful features like efficient functioning over heterogeneous network with transparency, builds the basics of ubiquitous systems. TINI, with its basic characteristics of embedding non-network type devices as unique network entities, provides an efficient resource management systems supporting easy addition/removal or activation/deactivation of various task-oriented units/equipments. Openness with standardization and compatibility supporting design, development and deployment of daily human operation based components can serve vital role providing interoperable, portable, flexible, and scalable (shrinking and expansion both) ubiquitous system. Fault tolerance over

specific components rather than entire system is a ubiquitous requirement and can be efficiently achieved though TINI or similar embedded device technologies. An architecture supporting size scaling as well as flexibility over choosing/changing task specific components without need of much change in the logical view or business logic is also one of the ubiquitous requirements and can be served efficiently using TINI or similar embedded technologies. A real-time intelligent communication system, which can sense/analyze on-demand communication type (based on geographical distance) and deploys synchronous or asynchronous communication with batch processing and parallelism, can provide an efficient network communication.

REFERENCES

[1] Wieser, M (March 1996) *Ubiquitous Computing PARC* available at, http://sandbox.xerox.com/ubicomp/ (Accessed 15 May 2010)

[2] Wikipedia, (2010) *Ubiquitous Computing* Available at: http://en.wikipedia.org/wiki/Ubiquitous_computing (Accessed 15 April 2010)

[3] Vdict (2010) *Ubiquitous Computing* available at: http://vdict.com/ubiquitous%20computing,6,0,0.html (Accessed 20 May 2010)

[4] Wieser, M (September 1991) *The Computer for the 21st Century* Available at: http://www.ubiq.com/hypertext/weiser/SciAmDraft3.html

[5] Dallas (2007) *Getting Started With TINI*, Available at: http://www.maximic.com/products/tini/pdfs/TINI_GUIDE.pdf (Accessed: 25 June 2010).

[6] The Open Group (2006) *Client/Server Model* [online] UK:TOGAF information web site Available at: http://www.opengroup.org/architecture/togaf8-doc/arch/p4/views/vus_comp.htm [Accessed 2 July 2010]

[7] Wikipedia (2010) Thin Client Available at: http://en.wikipedia.org/wiki/Thin_client (Accessed 02 July 2010)

[8] Wikipedia (2010) *Distributed Computing* Available at: http://en.wikipedia.org/wiki/Distributed_computing (Accessed 15 February 2010)

[9] The Chemistry Encyclopedia (2010) *Distributed Computing* Available at: http://chemistrydaily.com/chemistry/Distributed_system [Accessed 17 March 2010]

[10] Microsoft Corporation (2001) *Distributed Computing* Available at: http://docs.msdnaa.net/ark/Webfiles/glossary.htm [Accessed 10 Dec 2009]

[11] Tanenbaum, A. S. and Renesse, R. (1985) *Distributed Operating Systems* 1st Ed. U.S.: Prentice Hall. [Accessed 25 JUNE 2010]

[12] Gartner, F. C. (1999) *Defining Faults and Fault Models* [online] USA: ACM Computing Surveys Available at: http://www.cs.nyu.edu/courses/fall05/G22.2631-001/survey.pdf [Accessed 27 June 2010]

[13] Wikipedia (2010) *Remote Procedure Call* Available at: http://en.wikipedia.org/wiki/Remote_procedure_call (Accessed 15 February 2010)

[14] Microsoft (2003) *RPC Architecture* [Online] USA: Microsoft TechNet Available at: http://technet2.microsoft.com/WindowsServer/en/Library/4dbc4c95-935b-4617-b4f8-20fc947c72881033.mspx?mfr=true [Accessed 28 June 2010]