International Journal of Advanced Computer Research (ISSN (print): 2249-7277 ISSN (online): 2277-7970)
Volume-3 Number-1 Issue-9 March-2013

A Survey of Middleware in different languages for Ubiquitous Services

Neha Sharma[1], Usha Batra[2], Saurabh Mukherjee[3]

Abstract

The advent of abrupt advancement in technology and availability of numerous applications has made it imperative to adopt a technology that is felicitous for every domain. There exists a huge count of technologies working in all domains. The transient nature of technology itself is a reason for it being obsolete. Enterprise application development (EAI) solutions support many on-going projects ranging from proprietary, open or academic projects to scientific experiments. After analysing various middleware, we agreed on the fact that a language works best in a specific domain but may not be amicable with other domains. The rapid augmentation in technology and unstable requirements are responsible for having dearth in languages. Therefore, succinctly the environment is unstable and will be in future as well. This research discusses middleware technologies available in various languages. In this paper, on the basis of literature survey we present a comparative analysis of their features, flexibility, scalability and trade-offs and making it impeccable at some extent is what we aspire.

Keywords

Middleware, Java, Cloud Computing, CORBA, Web Services.

1. Introduction

Enterprise application integration (EAI) is used to integrate a set of enterprise computer applications with support of software and computer systems architectural principles. The term EAI is related to Middleware and provides a workspace for two or more applications to collaborate. However, Middleware is considered as a continually evolving term [1]. It is difficult to consider a single concrete solution to implement in every domain.

Neha Sharma, Department of Computer Science, ITM University, Gurgaon, Haryana, India.
Usha Batra, Department of Computer Science, ITM University, Gurgaon, Haryana, India.
Saurabh Mukherjee, Department of Computer Science, Banasthali, Rajasthan, India.

Since much of the software business is driven through the perceptions of the "hottest" current technologies [2], middleware should offer flexible ways to execute different applications running on diverse range of technologies.

In addition to flexibility, EAI also minimizes the number of connections among applications and hence results in reduced network overhead and better performance. For instance, consider a model without EAI approach. The number of total connections required for a full point-to-point network with n nodes within or across the organizations will be (n(n-1))/2 . For analogy to achieve communication among ten different nodes, 45 connections lines will be needed. For 50 nodes it will need 100 connections lines and so on resulting in a very expensive and complex communication mesh. However, using EAI, the number of connections can be reduced to minimal 'N' connections, where n represents number of nodes in the communication network.

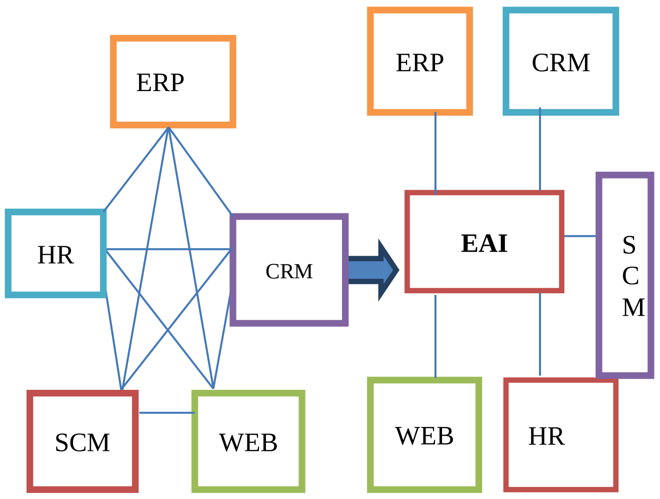

Figure 1: Enterprise Application Integration

Middleware not only helps in reducing the number of connections required to achieve interoperability but it also offers flexibility in highly dynamic environment. The best example is the rapid increase in the demand of smart phone as well mobile users resulting in demand of ubiquitous services. In 2006, Julien, C., Roman et al.[3] proposed that ubiquitous computing normally refers to as constant change; users must interact with heterogeneous systems using different devices at different locations over different networks with different quality-of-service requirements [3].The above requirements also motivated Middleware to develop new approaches like reflective middleware, message oriented middleware, Database middleware etc. In this paper, we describe how we can use various languages to implement a middleware in ubiquitous environment. The remainder of this paper is organized as follows: the background and related work is described in section 2. In section 3, different technologies are detailed. Section 4 gives a brief discussion table on all technologies followed by Section 5 containing our conclusion and future work.

2. Related work

Several middleware are proposed in different application regions such as Finance, Banking, E-healthcare, companies, distributed systems, sensor management, network management and commercial use, particular with respect to performance and resources trade-offs. Traditionally this work has been driven either from a Quality of Service (QoS) perspective or from an aim to simplify implementation. In 2011, Mangal Sain et al.[4] compared some middleware approaches on basis of their design, tasks, features, execution environment and security and classified these middleware as mentioned below.

Table 1: Adapted from Mangal Sain et al. [4]

Sno	Middleware	Description
1	Message Oriented Middleware	It enables the communication between distributed systems through messages. It is accomplished using a message queuing middleware (MQM) to store and forward messages whenever required and a message broker to maintain the queue and allow the recipient to retrieve messages at its convenience.
2	Database-inspired Middleware	It makes use of SQL like queries to provide a bridge between end-user application and database. This interface allows user to make a database request, then process the data and returns the result back to the end-user. Few examples are COUGAR, TinyDB[5], SINA etc.
3	Reflective Middleware	It offers dynamic implementation of ubiquitous services by allowing the programmer to modify the existing infrastructure. Various language-specific as well as language-neutral middleware are developed in this domain. For Example DynamicTao, FlexiNet, VisiBroker etc.
4	Tuple space Middleware	This middleware allow data sharing and communication among different distributed processes using a shared memory called tuple space. Communication between processes is achieved by maintaining these tuples that stores elementary data structures. For example, EgoSpaces[6], TinyLIME, TOTA etc.
5	Service Discovery Middleware	It uses either a distributed or a centralized approach to provide link between devices that needs to share data and services dynamically. For example, MiLAN, M2MI[7] etc.

Some middleware offers abstraction, application knowledge like features but each tend to lack in other issues like information quality, service quality, adoptability, interface and most important security. As per the table above, we can see how different middleware are designed and implemented.

3. Different Technologies for Middleware

Generally, just one or two form of interaction like RPC or transactions is provided to application programmers in middleware platforms. As industry is expanding, the middleware platforms are expected to be a “plug-n-play” kit providing privacy, transaction, authorization, legacy, flexibility, transparency and many other features. At Present, systems offer features but none of them imposes a whole package. After a literature survey on these middleware platforms, we categorize them (on basis of the languages used) into following subsection:

- Java Middleware.
- CORBA based Middleware

- Web Services Middleware
- Cloud Middleware

3.1 Java Middleware

Java Middleware envelops many application server namely WebLogic, SpiritWave and some primary servlets like ORB-based and OODB based. All of these are built upon DMS legacy and add server-based java execution feature. Java is an easy to use and mature language with open services and protocols. It offers interconnection through internet, ability to share data and processes using a centralized application management for load balancing, recovery, maintenance and storage. Due to its ability to reuse, it has been widely accepted in many organizations as a middleware technology. Also, features like language level introspection, and dynamic code creation make it particularly suited to the design of middleware systems.

3.1.1 FlexiNet

In 1998, Richard Hayton et al.[8] proposed a Java middleware platform called FlexiNet that allows programmers to adjust the underlying platform and communication infrastructure. FlexiNet is regarded as a reflective technique and an example of language level introspection [8]. It is language specific, ergo all applications must be written in java language. However, this gives a great deal of support when it comes to introspection and reflection providing FlexiNet strong internal type codes unlike CORBA. In addition to the above fact, the tailoring of platform benefits us to keep track of relationship among components. FlexiNet presents a clear computational model composed of objects, interfaces, methods and invocations. The transparent access makes it easier to move applications among different environments. It uses references to interfaces for parameter passing, and multiple binders for binding protocols support. FlexiNet is a component based framework which is primarily designed to obtain modularity and reusability, rather than performance. Although, to increase performance, many layers of RMI stack could be implemented as one module [8]. The researchers at APM also did a comparative analysis of FlexiNet with earlier versions of Java, and concluded that FlexiNet performs as efficient as any other java middleware.

3.1.2 Enterprise Java Beans (EJB)

Enterprise Java Beans (EJB) technology is considered as the server-side component architecture for Java Platform, Enterprise Edition (Java EE) [9]. The main objective of EJB is to abstract the application business logic from underlying middleware. It enables rapid and simplified development of applications based on Java technology [9]. Two main concepts dominate the EJB paradigm: **Beans**; that fits together in a standard way containing pure application logic and hooks and **Containers**; middleware that performs communication between Beans and Clients and manage bean's creation and deletion.

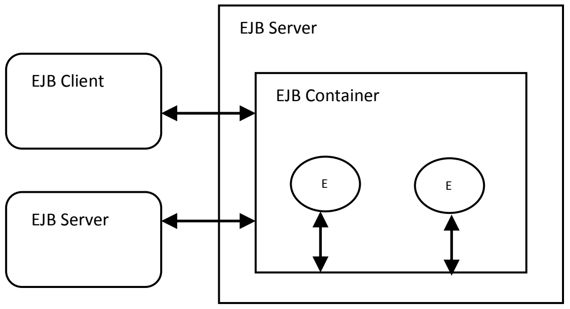

Figure 2: Adapted from Masslight [10]

An EJB server provides a number of services such as database access (JDBC), transactions, messaging, naming and management support (JMX).The server manages one or more EJB containers and the container is responsible for providing database connection and component pooling, transaction, lifecycle and client session management, persistence, authentication, and access control [11].

EJB has two different implementations: Entity beans to map data stored in the database and session beans to perform either temporary operations (stateless session beans) or represent temporary objects (stateful session beans).EJB application is easier to write and deploy [12] however its framework is complex. It works well with JSP pages or Servlets, believed as a major reason for many E-Commerce companies' products to shift towards EJB and Java-based www technologies. Among all vendors' EJB products, Sun's specification is considered ahead.

3.2 CORBA based Middleware

CORBA is a technique that allows you to make calls between java objects and objects written in other languages making it unique from RMI or java. To achieve so, it needs an Object Request Broker (ORBs) on both sides. In 2003, Arvind et al. [13] defined CORBA ORB as , "CORBA Object Request Broker (ORB) allows client to invoke operations on distributed objects" regardless of the type of Programming language, architecture or the operating system it is using for execution. The interface description language (IDL) for CORBA enables the independency to perform the above. In fact, CORBA was the middleware product used and deployed for Windows-based PCs by 1998. A java-compatible version of CORBA named VisiBroker was distributed to millions of PCs to provide communication to remote objects via internet. In addition to this de facto CORBA specification defines more than a dozen "services" that ORB can use for various tasks. CORBA was extended to create Real-Time CORBA adopting a priority based scheduling although it is provided as an option.

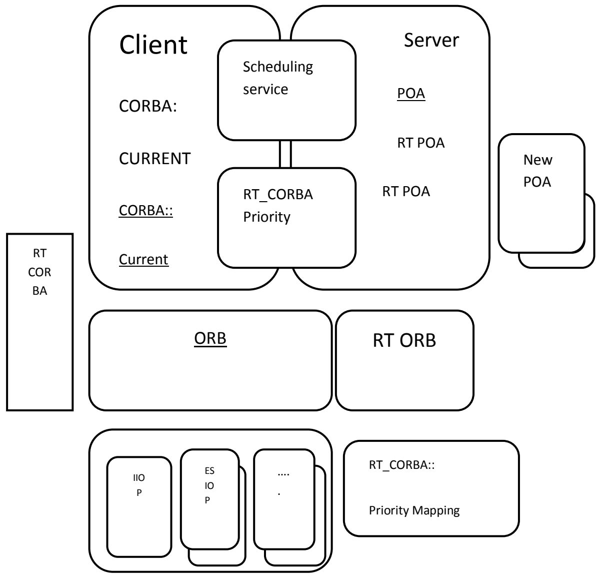

Figure 3: Enhancement of Real-Time CORBA, Adapted from Arvind et al.[13]

CORBA is considered unique because it is integrated with both Internet and Java. Integration with Java makes it easier for developers to use simpler and network oriented features and Integration with Internet allows developers to use internet infrastructure to build efficient distributed application relatively faster and more flexible.

3.2.1 VisiBroker

VisiBroker is a CORBA 2.3 complaint ORB with a suite of tools embedded inside and offers a complete implementation of the CORBA specification. VisiBroker provides a C++ programming language support and include libraries and headers that allow client and server to write programs for communication purpose without having to deal with the complicated code underlying in the network.

The first step of creating an application with VisiBroker is to specify all of your objects interfaces using CORBA's IDL i.e. Interface Definition language. First, an interface specification is created and it is used by VisiBroker IDL compiler to generate stub routines. The client application will use these routines for all method invocations. Then, the code you write and the skeleton code integrates to create a server that will implement the objects. Once the code for the client and object is completed, it is used as input to your **C++** compiler and linker to produce the executable client application and object server.

It offers features like Object naming, persistent objects support, creating dynamic object and interoperability [14] with other ORB implementation. Visibroker like any other CORBA product contains every function that is needed to get two programs to talk to each other. An executable code called idl2cpp.exe is used to compile CORBA IDL to C++ code and the compiled code can be used for both client and server applications. As a feature, it offers "Out-of -box" security [15], connectivity to internet, persistent storage and seamless integration to J2EE Platform.

3.2.2 DynamicTao

TAO contains a highly optimized protocol engine (Gokhale and Schmidt 1998) that implements the CORBA 3.0 General/Internet Inter-ORB Protocol (GIOP/IIOP), version 1.0, 1.1 [13],and 1.2. TAO can therefore interoperate seamlessly with other ORBs that use the standard IIOP protocol. Initially, some conventional static approaches for middleware were made in TAO for application such as Avionic Mission Computing systems [17]. But with increase in complex requirements, dynamic approach was considered. DynamicTao [16] is an extension of TAO and a reflective CORBA ORB that take efforts in modularizing and organizing a middleware. It achieves the property of reconfiguring and low inspecting of its internal engine by exporting an interface. It contains a component configuration file specifying all the strategies like concurrency, security, and monitoring and connection management. Component implementations are designed to link with ORB process at runtime and organize them in categories representing different aspect of ORB internal engine. It is primarily designed to deal with diversity and it achieves the goal by yielding a middleware which works effectively in ubiquitous environment. DynamicTao offers flexibility, scalability and maximum utilization but as a CORBA implemented middleware it had weak vendor to vendor interoperability of products and a less integrated tool support than Distributed component object model (DCOM).

3.3 Web Services based Middleware

A web service is a key to facilitate application integration by presenting the functionality of information system and making it available through standard web technologies. Web service is an ambiguous term reflecting many different meanings in diverse situations. UDDI consortium [18] gave a detailed and specific definition of Web service as "self-contained, modular business applications that have open, Internet-oriented, standards-based interfaces". However, it doesn't really specifying the core working of the concept. The definition is further refined and redefined by W3C organization as [19], "A Web service is a software system identified by a URI, whose public interfaces and bindings are defined and described using XML. Its definition can be discovered by other software systems. These systems may then interact with the Web service in a manner prescribed by its definition, using XML based messages conveyed by Internet protocols". The W3C's definition is more accurate and indicates the working of the application.

The advantages of web technologies, such as universal client side and easy maintenance, bring out the traditional application into a browser-Server structure application. But, it also exposes some issues accompanying its development. For an instance, the web pages are designed for human being not for applications. A web service uses XML language which contains a web service description language (WSDL). A WSDL keeps a file with all the functions needed to be implemented. As a result, whenever required user doesn't have to know about whole web service but the WSDL file describing what it can do. Various open technologies and standards are defined for web services. The **Extensible Markup language (XML)** allows application to exchange information regardless of the way they are internally structured, thereby providing easily understandable and usable programming language. The **Universal Discription, Discovery and Integration** (UDDI) are designed to create B2B applications. It brings flexibility, and dynamic use of web services. **WSDL** intends to define web services regardless of their implementation or even the technology that is used for implementation; however it relies upon XML to describe a web service.

3.3.1 CoServices

In 2013, Wangyi Xie et al. [20], proposed a system named CoServices. After Studying numerous systems, the researchers came up with a conclusion that the systems contain some common modules with very similar functions, including message notification module, management module for user and shared data, multimedia communication tools, concurrency control module etc. which consequently lead to additional cost when developers have to develop new application models from scratch. So, researchers advanced a universal middleware framework known as CoServices which provides common modules as well as unified API for cooperative applications. Thus developers have the common modules at hand while developing a new application. For example, following figure consists of some common modules as well as some domain-specific modules.

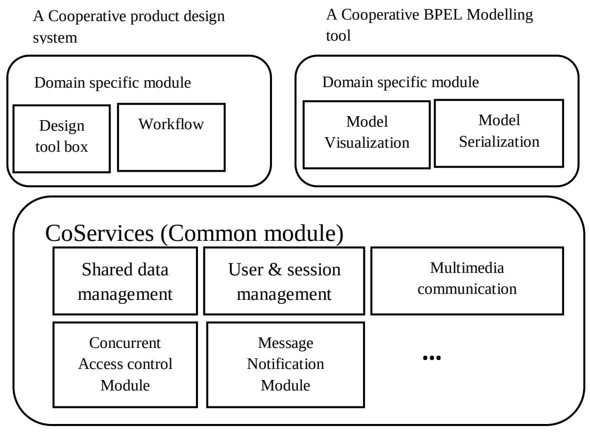
A Cooperative product design system
A Cooperative BPEL Modelling tool
Domain specific module
Design tool box
Workflow
Domain specific module
Model Visualization
Model Serialization
CoServices (Common module)
Shared data management
User & session management
Multimedia communication
Concurrent Access control Module
Message Notification Module
...

Figure 4: Cooperative applications based on CoServices, Adapted by Wangyi Xie [20]

The data sharing model and synchronization mechanism of CoServices is based on two concepts: Web Services Resource Framework (WSRF) and Web Services Notification (WSN). **WSRF** defines a generic and open framework and includes mechanism to describe views on the resources and there extensibility to groups of services, and **WSN** defines a mechanism similar to "Publish/Subscribe" for web services. However considering the design of web services, its implementation was done using Java Program Language.
To evaluate the effectiveness of the system, two cooperative applications named cooperative

workflow modelling tool and cooperative medical image browser [20] were developed based on CoServices. The data management module and message notification module were duplicated in both models while other modules were distinct. Both models justified the fact that CoServices can prove as a very effective model when it comes to reusability and effectiveness.

3.4 Cloud based Middleware

Cloud computing is a recent IT technology used increasingly in industries, government and private organisations. There exists several cloud based middleware such as Nimbus, Eucalyptus, and openNebula[21], [22] . Cloud computing has gone through remarkable changes over some recent years. It enabled Ubiquitous, on-demand network access and a shared pool of computing resources in hand. It is emerged as an alternative computation model that demonstrated applicability to a wide range of problems in scientific and other domain specific areas. It is cheap, on-demand and ready to use with a wide pool of resources. User can access a large amount of data through resource managing middleware and they have to pay based on the usage of resources. Clouds can be classified in three parts: **Private cloud** used only for the dedicated organizations, **public cloud** shared between multiple organization and a **hybrid cloud** that is a combination of both providing in-house and external connections.

3.4.1 SciCumulus

With the emergence of cloud computing as an alternative to make a middleware model, scientists are starting to adopt this model in scientific domains and moving their workflows form local to cloud environment. However to accomplish this, they have to manage new aspects of cloud. A lightweight Cloud middleware "SciCumulus" is designed by researchers [23] to enable parallel execution by placing it amidst SWfMS and the cloud. The SWfMS provides a way to define, execute and monitor scientific workflows either locally or in remote environment. The best advantage of using SciCumulus is its distributed behaviour. SciCumulus is composed of three layers: **Desktop layer** to dispatch workflow activities to be executed in the cloud environment, **Distribution layer** to manage the execution of activities in cloud environments, and **Execution layer** to execute programs. To diminish efforts of scientists, SciCumulus offers two type of parallelism: parameter sweep and data parallelism [23]. Numerous experiments were held to evaluate and analyse the overhead imposed by clouds on the simulation environment and degree of parallelism gained. SciCumulus achieved transparent parallelism using SWfMS and high performance while keeping track of the workflow provenance.

4. Discussion

In last section, we presented a brief literature survey about languages applied in different area of research and few examples of middleware that have been developed by researchers with the change in time and requirements. On the basis of this survey and conceptual analysis, we summarize this paper in table 2 illustrating a comparative analysis of these middleware technologies.

Table 2: The comparison of middleware languages

Basis	**Java Middle-ware**	**CORBA based Middle-ware**	**Web services based Middle-ware**	**Cloud Middle-ware**
Flexibility	High	High	High	High
Scalability	High	High	Low	High
Security	Low	Low	Low	Relatively High
Benefits	Modular, Reusable, Transparent, easy to write and deploy.	Reflexive, Portable, Extensible, Compatible with different client platform	Compati ble with various client platform	Parallel execution, Scheduling support
Drawbacks	Complex frame-work and performance is not considered as a primary goal.	Data sharing issues	Data sharing issues	Overhead of transferring data
Frame-work	Component based and pure application logic	Component based and three layered architecture .	3 layered architect-ure	3 layered architecture

5. Conclusion and Future Work

We considered a set of middleware functionalities and on their basis; we got a comparative analysis of different middleware languages. Four popular languages platform named Java, CORBA, Web services and Cloud computing were presented. All middleware provides a good virtualization support and basic functionalities, but most of them lacked in security standards. Cloud based middleware seems to be favourable when security is considered. In 2007, Lenin et al. [24] also offered ISO-WSP, a secure middleware for web services. However, all paradigms are developed for different requirements so we cannot declare one as “best” solution. To develop a robust middleware, the flaws must be eliminated.

In future, we intend to have an experimental analysis on use of Aspect oriented programming (AOP) approach along with FlexiNet (using Weaver Compiler) to gain higher level abstraction and scalability. Aspect oriented programming gives the programmer the ability to identify and manipulate only those parts of the software that are relevant. Using weaver compiler we intend to create some modules for functionalities that are lacking in existing middleware and combine them with the final executable code. For instance, we can develop a single aspect (e.g. security) and instead of adding it in every class, we can combine it in final executable code.

References

[1] Mala Chandrasekaran, “Middleware Technology”, Online at https://www.classle.net/projects/project_ideas/ middleware-technology (as of 19 December 2010).

[2] Middleware Technology, Rogers State University, Emerging Technologies, TECH 3023, 31 Jan 2001.

[3] Julien, C., Roman, G.C, “EgoSpaces: Facilitating Rapid Development of Context-Aware Mobile Applications”, IEEE Trans. Software Eng., 32(5), 2006.

[4] MangalSain, Pardeep Kumar and Hoon Jae Lee “A Survey of Middleware and Security approaches for Wireless Sensor Networks” Dongseo University, Busan, Republic of Korea, 617716, 2011.

[5] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong, “TinyDB: An acquisition query processing system for sensor networks”, Journal ACM Transactions on Database Systems”, 30(1):122–173, 2005.

[6] M. Roman, F. Kon, and R.H. Campbell, "Reflective Middleware: From the Desk to your Hand," IEEE Distributed Systems Online, 2001, vol. 2, no. 5, 2001.

[7] Hans-Peter Bischof, Joel Varela Donado “M2MI Service Discovery Middleware Framework”. Rochester Institute of Technology, 102 Lomb Memorial Dr., Rochester, NY 14623, July 2005.

[8] Richard Hayton, Andrew Herbert, Douglas Donaldson, “FlexiNet - A flexible component oriented middleware system”, 8th ACM SIGOPS European workshop on Support for composing distributed applications, pp17-24, 1998.

[9] Oracle Technology Network, “Enterprise JavaBeans Technology”, Online at http://www.oracle.com/technetwork/java/javaee/ejb-141389.html.

[10] Masslight, “Chapter 6. Enterprise JavaBeans”, Online at http://j2ee.masslight.com/Chapter6.html.

[11] Emmanuel Cecchet, Julie Marguerite and Willy Zwaenepoel, “Performance and Scalability of EJB Applications” Rice University/INRIA, 17th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications, pp246-261, Nov. 2002.

[12] A. Popovici, “Middleware Technology”, Information and Communication Systems Research Group, ETH Zürich. Part VIII – EJ.

[13] Arvind S. Krishna Raymond Klefstad Douglas C. Schmidt, “Real-time CORBA Middleware”, Nov 2003.

[14] Visibroker for C++ Programmer’s Guide, “Chapter 1: VisiBroker Basics”, Online at http:// www.ime.usp.br/~reverbel/SOD97/Manuais/vbrokerc++/prog_gd/noframes/chap01.htm.

[15] Borland Software Corporation “Borland VisiBroker™ 8.0” Online at http://techpubs.borland.com/am/ visibroker/v80/en/vb_overview.pdf (as of April 2007).

[16] F. Kon et al., "Monitoring, Security, and Dynamic Configuration with the dynamicTAO Reflective ORB," Proc. of the IFIP/ACM Int’l Conf. on Distributed Systems Platforms and Open Distributed Processing (Middleware’2000), ACM Press, New York, pp. 121-143, Apr. 2000.

[17] David L. Levine, Christopher D. Gill, and Douglas C. Schmidt, ”Dynamic Scheduling Strategies For Avionics Mission Computing”, Digital Avionics Systems Conference, 17th DASC, pp1 – 8, Nov. 1998.

[18] UDDI Executive White Paper November 14, 2001.

[19] Michael Champion, Chris Ferris, Eric Newcomer, Iona and David Orchard, “Web Services Architecture”, Online at

http://dev.w3.org/2002/ws/arch/wsa/wd-wsa-arch.html#whatisws (as of 25 November 2003).

[20] Wangyi Xie, Zhuqing Li, Yongwang Zhao "CoServices: A Web Service Based Middleware Framework for Interactive Cooperative Applications", Third International Conference on Intelligent System Design and Engineering Applications, 2013.

[21] F. Feldhaus, S. Freitag and C. El Amrani, "State-of-the-Art Technologies for Large-Scale Computing", Werner Dubitzky, Krzysztof Kurowski and Bernhard Schott, LargeScale Computing Techniques for Complex System Simulations, Wiley-IEEE Computer Society Pr, Ch. 1, pp. 1-17, 2011.

[22] L. Wang, M. Kunze, J. Tao, G. von Laszewski, "Towards building a cloud for scientific applications", Advances in Engineering Software, Vol 42, pp. 714–722, 2011.

[23] Daniel de Oliveira, Marta Mattoso "SciCumulus: A Lightweight Cloud Middleware to Explore many Task Computing Paradigm in Scientific Workflows", IEEE 3rd International Conference on Cloud Computing, 2010.

[24] Lenin Singaravelu, Jinpeng Wei, Calton Pu "A Secure Middleware Architecture for Web Services", College of Computing, Georgia Institute of Technology, May 2007.

Neha Sharma obtained B.Tech Engineering in Computer Science and Engineering from Maharshi Dayanand University, Haryana in 2012. She is currently pursuing her M. Tech Engineering in Computer Science Department from ITM University Gurgaon, Haryana.

Usha Batra is currently Assistant Professor (Senior Scale) in CSE Department at ITM University. She has more than 9 years of Teaching Experience. She is currently pursuing Ph.D in Computer Science. She has done M.Tech (CSE) in 2006 and B.Tech (CSE) in 2003. She has Published 3 papers in International Journals, 10 Papers in international conference proceedings, 3 in national Journals and 2 in National Conference proceedings and one.

Dr. Saurabh Mukherjee is currently working as an Associate Professor in Computer Science Department at Banasthali University, Rajasthan. He has done Ph.D., MCA and M.Sc. in Computer Science Department. His area of interest includes Medical Image Processing, Biomedical Engineering, Cognitive Science and Soft Computing.