CHAPTER 6

DESIGN OF ARCHITECTURAL MODELS FOR REAL
TIME MONITORING OF ELECTRICAL DRIVE SYSTEM

6.1 INTRODUCTION

The remote access, maintenance operations and monitoring of electric drives can be done through the network and web browser. Based on the commonly web design technology, the embedded web server is designed by integrating embedded CGI (Common Gateway Interface), ActiveX and Java Applet technology and the problem of communication security is considered. The system can complete the remote access, monitoring and maintenance operations of electric drives through the network and web browser. The remote monitoring and control architecture consists of a Web-Services-based Monitoring and Control Gateway (WSMCG), distributed ethernet-real I/O modules, safety detection modules, web cameras, and networks. It incorporates the newest network-related technologies with the concept of ensuring appliance safety for developing remote monitoring and control systems with novelty and high practicability.

6.2 NEED FOR WEB BASED MODEL FOR DRIVES MONITORING

The web-based client-server computing system is categorized into four types: the proxy computing, code shipping, remote computing and agent-based computing models. The Proxy Computing (PC) model is typically used
in web-based scientific computing. According to this model a client sends
data and programs to a server over the web and requests the server to perform
certain computing. The server receives the request performs the computing
using the programs and data supplied by the clients and returns the results
back to the clients. Typically, the server is a powerful high-performance
computer or it has some special system programs that are necessary for the
computation. The client is mainly used for interfacing with users.

The Code Shipping (CS) model is a popular Web-based client-
server computing model. A typical example is the downloading and executing
of java applets on web browsers. According to this model, a client make a
request to a server, the server then ships the programs over the web to the
client and the client executes the program using the local data. The server acts
as the repository of programs and the client performs the computation and
interfaces with users

The Remote Computing (RC) model is typically used in web –based
scientific computing and database application, the client sends data over the
web to the server and the server performs the computing using programs
residing in the server. After the completion of the computation, the server
sends the result back to the client. Typically the server is a high performance
computing server equipped with the necessary computing programs and
databases. The client is responsible for interfacing with users. The Net solve
system uses this model.

The Agent-based Computing (AC) model is a three-tier model, in
this the client sends either data or data and programs over the Web to the
agent. The agent then processes the data using its own programs or using the
received programs. After the completion of the processing, the agent will
either send the result back to the client if the result is complete, or send the
data/program/medium result of the server for further processing. In latter case,
the server will perform the job and return the result back to the client directly (or via the agent). Nowadays more and more web-based applications have shifted to the AC model.

The rapid developments of the internet and distributed computing have opened the door for feasible and cost-effective solutions in engineering application. Today’s large industry has very long range interconnections with many legacy systems and different operators which make the maintenance of various electrical machines is a challenging task. The computer applications used in monitoring the electrical drives have undergone profound changes since last few decades. The interconnected bulk electrical drives are becoming integrated with vast networked information systems.

The present conventional client-server architecture for monitoring electrical drives system is complicated, memory management is difficult, source code is bulky and exception-handling mechanism is not so easy. In the conventional system, it is assumed that the information required for the monitoring and controlling of electrical drives is centrally available and all computations are to be done sequentially at a single location.

With respect to sequential computation, the server has to be loaded every time for each client’s request and the time taken to deliver the solution is also relatively high. A good deal of literature and implementation is available pertaining to real-time monitoring the electrical drives through conventional client-server architecture. In spite of this, it is needed to develop an effective architecture to make this in a completely platform, language and location independent environment. So it is proposed to develop the new architectural models for monitoring electrical drives.
6.3 DESIGN CRITERIA FOR DEVELOPMENT OF ARCHITECTURE MODELS

The proposed architectural model provides an open, loosely coupled, seamlessly integrated computing paradigm for solving engineering problems. The proposed new architectural models for monitoring electrical drives should have the following characteristics (Chen et al 2006):

**Scalability:** Based on the same services repository, systems or applications with different sizes can be easily implemented by composing different number or types of services. This infrastructure provides the flexibility to facilitate frequent changes of developed service. When new challenges arise, new applications for them can be implemented by recomposing existing services or by adding new services into the system. The architectural models should be designed in such a way that any number of electrical machines can be served without limit.

**Security:** Security is an important issue for web based applications. Firewalls are needed in this integrated infrastructure in order to isolate or minimize the possibility that unauthorized client access modifies or destroys any critical information. Web Service Security (WSS) service provides a framework within which authentication and authorization take place. WSS offers a trusted means for applying security to Web services by providing the necessary technical foundation for higher-level services.

**Portability:** In the context of control centres, portability refers to the possibility of running the same software on different hardware/software platforms. Some existing models for drive system are based on versions of the Microsoft Windows operating system. Actually, they are not portable, but their dependence on specific vendors is limited to the operating system. Another important aspect of the portability is related to the programming
languages used. Web technologies are changing this picture dramatically, because common Web browsers may easily provide cross-platform access to graphical interfaces.

**Interoperability:** Interoperability is the ability to run software modules (identical or not) on different platforms, in the same network, at the same time, all communicating and interacting with one another. A good feature of systems with interoperability is the possibility of running exactly the same software on different hardware platforms, in the same network. Interoperability is not easy to achieve. The true benefits of portability become evident only if interoperability is also achieved. Web service is a possible solution for such integration.

**Expandability:** The control centre software must be able to efficiently support the expansion of the system. Both the growth of the machines available in industry and the inclusion of new software functionality must easily be accommodated while keeping the performance at acceptable levels. In short, control centre software must have expandability. This obviously has a very negative impact on the initial costs of the control centre. Portability and interoperability together make expandability a problem that is relatively easy to solve in open architecture control centres. Investment in idle hardware is not necessary because processing power can be added or upgraded later as needed. Distributed processing in networks of heterogeneous hardware allows for the safe addition of new computers or replacement of obsolete ones. Improvements in portability and interoperability will make expandability a goal that will become gradually easier to be achieved. But portability and interoperability are not sufficient to achieve expandability. The software must also be designed to accommodate growth in the number of clients accessing developed service, the number of
computers attached to the real-time network and the expansion of the system being controlled.

**Modularity:** Modularity is related to the ability of modifying the software with a negligible impact on software components that are not directly involved. Software modules can be added, modified, replaced, and, in many cases, even removed without affecting other modules.

**Availability:** Most of the utility real-time information systems require high availability. Traditionally, this is achieved by dedicating a redundant machine running the same applications for backup. When the primary system fails, the backup system will be automatically brought online replacing the failed primary system. Special software and hardware may be needed to synchronize the status, detect the failure and automate the transition. For a web service-based real-time information system, the high availability can be achieved through a redundant web service running on the backup machine. During the normal conditions, the redundant web service registers as a hibernating service. Its only task is to imitate the primary service by replicating its data and status. This can be achieved through the data exchange between the primary service and backup service. Specifically, the backup service subscribes all the data in the primary service. Any data changes in the primary service will be propagated to the backup service. When the primary service fails, the backup service will be discovered and awakened, taking over the duty of the failed primary service. Since the integration infrastructure provides full support, no special software or hardware is required to automate the above process.

**Adaptability:** One of the major features of the design integrated utility information system is its adaptability to the changing environment. When new services are added or the old services are replaced with the new
upgraded services, the integrated system can smartly adapt to the new environment.

**Flexibility:** Services in this infrastructure can be easily reconfigured or replaced. Service deployment can be conducted incrementally and scaling can take place over time.

**Pervasivity:** All the clients should be able to access the deployed service independently.

**Openness:** Different clients should be able to interact between them even in the presence of different technological gaps and different internal system types and organizations.

**Service Provider Independence:** The clients should not be forced or compelled to use a unique or privileged service provider to access network services and applications.

**Implementation Neutrality:** The implementation of each application service and its own complexity is opaque through encapsulation of language or platform differences behind a common interface. Services can be modified without the service’s users being affected. Authorized clients can request an information service transparently without knowing what servers, where or how they are attached to the network.

**Cooperation Capability:** No matter how the heterogeneous hardware or software environments are, the standardized data models, communication protocols and service interface descriptions easily support the cooperation among different operators, different applications such as process-oriented applications or business-oriented applications and different domains. Depending on agent-oriented cooperation and coordination services, the
services that are potentially related to a domain or a special process can be automatically located and discovered.

**Integration Capability:** Based on common ontology, information services can easily be integrated with other services either statically or dynamically. Lower-level services can be composed into higher-level services and legacy applications can easily be encapsulated or wrapped into a new environment.

### 6.4 DESIGN OF VARIOUS ARCHITECTURAL MODELS FOR MONITORING ELECTRICAL DRIVE SYSTEM

DCS (Distributed Control System)(Lilantha Samaranayake 2003) is a computerized control system used to control the production line in the industry. The entire system of controllers is connected by networks for communication and monitoring. DCS is a very broad term used in a variety of industries, to monitor and control distributed equipment. The basic advantage of DCS is its capability of controlling many such systems from a single location at a lower installation and maintenance cost. A distributed application is a system comprised of programs running on multiple host computers. The architecture of this distributed application is a sketch of the different programs, describing which programs are running on which hosts, what their responsibilities are, and what protocols determine the ways in which different parts of the system talk to one another.

In the beginning, individual electrical drives are connected to individual computers and the programs access all the parameters from the drives as input, saves and process the data output through computer-connected devices. With the invention of networks, drives connected to individual computers can be accessed by faraway computers by distributed programming called distributed models.
6.4.1 Conventional Model for Monitoring Drive System

Basically, if your application is running on a single computer, it has one-tier architecture as shown in Figure 6.1. A one-tier application is simply a program that doesn't need to access the network while running. The advent of the web complicates, since a web browser is part of a two-tier application since a web server being the other part. If a web browser downloads a java applet and runs it, for the present purposes, we will say that the self-contained applet is a one-tier application, since it is contained entirely on the client computer. Hence, a program written in JavaScript or VBScript deployed inside an HTML page would also qualify as a one-tier application.

![Figure 6.1 Conventional model of electric drive system](image)

One-tier architecture has a huge advantage of its simplicity, since it doesn’t need to handle any network protocols, so their code is simpler. Such code also benefits from being part of an independent operation. It doesn't need to guarantee synchronization with faraway data, nor does it need exception-handling routines to deal with network failure, bogus data from a server, or a server running different versions of a protocol or program.
6.4.2 Two Tier / Three Tier: Client/Server Model

A two-tier architecture is client server application and three-tier architecture is web based application. The two-tier is based on Client Server architecture. The two-tier architecture is like client server application. The direct communication takes place between client and server. There is no intermediate between client and server. Because of tight coupling a two-tiered application will run faster. Here the direct communication between client and server, there is no intermediate between client and server.

The evolution of distributed architecture resulted in the birth of client-server architecture. In distributed computing, the client-server paradigm refers to a model of network applications. Even though the client-server model provides a more cost-effective solution of computing, developing the client-server programs is more expensive than traditional approaches with respect to engineering applications. Distributed computing overcomes the drawbacks with the monolithic and client-server applications and performs computations by a more flexible and cost effective approach.

Real-time monitoring of electrical drive system can be carried out in distributed environment using remote procedure calls. Each electrical machine present in the drive system is considered as a client and hence multiple clients / single server architecture can be implemented, as shown in Figure 6.2, to carry out the real-time monitoring of electrical drive system in distributed environment.
Client Machines

Figure 6.2 Single-server / multiple-client architecture for real-time monitoring electrical drive system in distributed environment

A control centre basically monitors the various electrical machines through an applet for every specific period of time and frequently exchanges data with the client computer which is interfaced with machine. The server presents in the control centre does the required analysis and then distributes the results. Chronologically, the server process should be started first so that it can take the initiative to set up a connection link. It then starts waiting till it receives a connection request from the client. A client can register itself with the remote object just by invoking the registration procedure on the server object, when it needs a service from it. The remote object obtains the necessary information from the registered clients (electrical machine) and responds them correspondingly with the results at specific intervals of time. The total processes can be automated by making the server to get the measured values of various electrical parameters from each registered client at every specific period of time.
6.4.3 Multi-Tier Architectural Model for Monitoring Electrical Drive System

The multi-tier architectural model developed for monitoring electrical drive system is mainly three-tier architecture which contains a service tier, a data storage tier and a Man Machine Interface (MMI) tier, as shown in Figure 6.3.

![Three-tier client/server architecture for monitoring electrical drive system](image)

**Figure 6.3 Three-tier client/server architecture for monitoring electrical drive system**

To overcome the limitations of two-tier architecture, another layer called data storage tier has been added in addition to the man machine interface tier and the service tier. This is where the application logic of the system now resides in the middle tier and performs a number of different operations. The middle tier can be physically located on a separate machine but this is not always necessary. In application where the number of clients is limited, the application server can be deployed on the client machines or on
the database server, reducing the number of interface connections required making the system faster. As the numbers of clients in industry grow, it can be moved on to a separate machine which addresses the desired scalability benefits.

6.4.4 CORBA Model for Monitoring Electrical Drive System

CORBA model provides language transparency that facilitates the implementation of logic in any programming language. It uses Java platform by providing a distributed object framework, services to support that framework and interoperability with other languages. Java Interface Definition Language (IDL) enables distributed Java applications to transparently invoke operations on remote network. In CORBA model, the client can access the remote server through CORBA server using Internet Inter ORB Protocol (IIOP), as shown in Figure 6.4.

![Figure 6.4 CORBA model for real-time monitoring of electrical drive system](image)

In CORBA model, the client is represented as Java applet and can be downloaded in the client machine. The client applet is designed in such a way that it maintains the previous state until it receives the control signal from the main server for a given values of measured parameters. Updating of
the client can be easily done through network. The client communicates with the ORB in order to convey a request for an operation invocation to the server which receives the data and then sends the control signal via the ORB back to the client.

6.4.5 RMI-based Client-Server Model for Monitoring Electrical Drive System

Real-time monitoring of electrical drive system requires real-time data obtained from the different clients. Each sub-system of interconnected system is considered as a client and these clients are interconnected with the control center. The client applications are running in a heterogeneous environment. The present conventional client-server architecture is complicated, memory management is difficult and source code is bulkier and exception-handling mechanism is not so easy.

A client computer basically does the monitoring in distributed environment through an applet for every specific period of time and frequently exchanges data with the server. The server generates the control signal and then distributes it. Chronologically, the server process should be started first so that it can take the initiative to set up a connection link. It then starts waiting till it receives a connection request from the client. A client can register itself with the remote object just by invoking the registration procedure on the server object, when it needs a service from it. The remote object obtains the necessary data from the registered client objects and responds back correspondingly with the results. This total process can be automated by making the server get the input data for every specific period of time. Transaction of data between clients and server takes place several times and so the possibilities of the occurrence of errors may be high. Hence it must be handled properly. Following are the limitations of using RMI in Applets on the Internet:
The most popular web browsers--Netscape Navigator and Microsoft Internet Explorer do not adequately support RMI. Navigator supports RMI on most platforms but does not support HTTP tunneling. The clients connected to a web through the proxy server cannot use RMI applet.

- RMI registry can experience problem when handling connections via the Internet.
- The RMI registry is not as flexible as most server applications and RMI server application cannot support multiple domains on a single Web server which in turn limits the deployment options.
- An application requires the data exchanged between the client and server to be encrypted or compressed. RMI requires only implementing socket classes and RMI socket factory that creates specific special sockets. However, overriding the default socket factory disables RMI’s ability to do HTTP tunneling thereby preventing access by proxy server users.
- Deploying RMI’s application requires additional network firewall configuration by default RMI object. RMI models are bound to random “anonymous” server port numbers, which make it impossible to predict the ports that must be available to internet clients. The proposed web-based model allows specifying a port number when creating a remote object.
6.4.6 RMI_IIOOP Model for Drive Monitoring

RMI-IIOOP denotes the Java Remote Method Invocation (RMI) interface over the Internet Inter-Orb Protocol (IIOOP), which delivers Common Object Request Broker Architecture (CORBA), distributed computing capabilities to the Java platform. It was initially based on two specifications: the Java Language Mapping to OMG IDL, and CORBA/IIOOP. With features inherited from CORBA, software components that work together can be written in multiple computer languages and run on multiple computers. In other words, it supports multiple platforms and can make remote procedure calls to execute, subroutines on another computer as defined by RMI.

The Java RMI-IIOOP specification was created to simplify the development of CORBA applications, while preserving all major benefits. It was developed by Sun Microsystems and IBM, combining features of Java RMI technology with features of CORBA technology. RMI-IIOOP uses generated code for remote objects and does not require supplementary classes for non-trivial data, unlike CORBA. This results in less complexity and a smaller footprint. Both CORBA and RMI-IIOOP utilize the general Inter-ORB protocol communication standard. RMI-IIOOP is largely based on the object by value concept that serves as a container or direct replacement for CORBA structures, unions, sequences, arrays and strings. No separate IDL is necessary. Instead, the data structure definitions are discovered automatically via reflection mechanisms. However, it is possible to generate the IDL definitions for the involved RMI-IIOOP data structures and use these definitions to exercise finer control between RMI-IIOOP and CORBA communicating partners. Recent versions of RMI-IIOOP derive their servants from the standard servant class. Hence, it is possible to connect them to a CORBA ORB manually, involving one or more of portable object adapters,
6.4.7 Socket Model for Drive Monitoring

A network socket is an endpoint of an inter-process communication flow across a computer network. Today, most communication between computers is based on the Internet Protocol (IP); therefore most network sockets are internet sockets. A socket API is an application programming interface, usually provided by the operating system, which allows application programs to control and use network sockets. Internet socket APIs are usually based on the Berkeley sockets standard. A socket address is the combination of an IP address and a port number, much like one end of a telephone connection is the combination of a phone number and a particular extension. Based on this address, internet sockets deliver incoming data packets to the appropriate application process or thread. An Internet socket is characterized by a unique combination of the following:

- Local socket address: Local IP address and port number
- Remote socket address: Only for established TCP sockets. As discussed in the client-server section below, this is necessary since a TCP server may serve several clients concurrently. The server creates one socket for each client, and these sockets share the same local socket address from the point of view of the TCP server.
- Protocol: A transport protocol (e.g., TCP, UDP, raw IP, or others). TCP port 53 and UDP port 53 are consequently different, distinct sockets.
Within the operating system and the application that created a socket, a socket is referred to by a unique integer value called a socket descriptor. The operating system forwards the payload of incoming IP packets to the corresponding application by extracting the socket address information from the IP and transport protocol headers and stripping the headers from the application data.

In IETF Request for Comments, Internet Standards, in many textbooks, as well as in this article, the term socket refers to an entity that is uniquely identified by the socket number. In other textbooks, the socket term refers to a local socket address, i.e. a "combination of an IP address and a port number".

There are several Internet socket types available:

- Datagram sockets, also known as connectionless sockets, which use User Datagram Protocol (UDP)

- Stream sockets, also known as connection-oriented sockets, which use Transmission Control Protocol (TCP) or Stream Control Transmission Protocol (SCTP).

- Raw sockets (or Raw IP sockets), typically available in routers and other network equipment. Here the transport layer is bypassed, and the packet headers are made accessible to the application.

Computer processes that provide application services are referred to as servers, and create sockets on start-up that are in listening state. These sockets are waiting for initiatives from client programs.
A TCP server may serve several clients concurrently, by creating a child process for each client and establishing a TCP connection between the child process and the client. Unique dedicated sockets are created for each connection. These are in established state, when a socket-to-socket virtual connection or Virtual Circuit (VC), also known as a TCP session, is established with the remote socket, providing a duplex byte stream. A server may create several concurrently established TCP sockets with the same local port number and local IP address, each mapped to its own server-child process, serving its own client process. They are treated as different sockets by the operating system, since the remote socket address (the client IP address and/or port number) are different; i.e. since they have different socket pair tuples.

A UDP socket cannot be in an established state, since UDP is connectionless. Therefore, netstat does not show the state of a UDP socket. A UDP server does not create new child processes for every concurrently served client, but the same process handles incoming data packets from all remote clients sequentially through the same socket. It implies that UDP sockets are not identified by the remote address, but only by the local address, although each message has an associated remote address.

The socket is primarily a concept used in the transport layer of the internet model. Networking equipment such as routers and switches do not require implementations of the transport layer, as they operate on the link layer level (switches) or at the internet layer (routers). However, stateful network firewalls, network address translators, and proxy servers keep track of active socket pairs. Also in fair queuing, layer 3 switching and Quality of Service (QoS) support in routers, packet flows may be identified by extracting information about the socket pairs. Raw sockets are typically available in
network equipment and are used for routing protocols such as IGRP and OSPF, and in Internet Control Message Protocol (ICMP).

6.4.8 XML-RPC Technology for Drives Monitoring

XML-RPC is a Remote Procedure Call (RPC) protocol, use XML to encode its calls and HTTP as transport mechanism, by sending a HTTP request to a server to implement the protocol. The client is software to call a single method of a remote system. Multiple input parameters can be passed to the remote method, one return value is returned. The parameter types allow nesting of parameters into maps and lists, thus larger structures can be transported. Therefore XML-RPC can be used to transport objects or structures both as input and as output parameters. Calls among architecture processes are performed using XML-RPC. Data transport is TCP-IP based; therefore the system is accessible from a conventional Internet link.

Identification of clients for authorization purposes can be achieved using popular HTTP security methods. Basic access authentication is used for identification; HTTPS is used when identification (via certificates) and encrypted messages are needed. Both methods can be combined.

XML-RPC is simpler to use and understand than Service Oriented Architecture Protocol because it

- allows only one method of method serialization, whereas SOAP defines multiple different encodings
- has a simpler security model
- does not require (nor support) the creation of WSDL service descriptions, although XRDL provides a simple subset of the functionality provided by WSDL.
6.5 WEB-BASED MODEL FOR MONITORING ELECTRICAL DRIVE SYSTEM

Web technologies enable communication between dissimilar computers over a large geographical region via Intranet or Internet. These provide a general distributed computing environment so that distributed applications can be implemented in it to exploit cheap but powerful parallel virtual machines. Web-based computing permits data sharing and computing over a large system range on heterogeneous hardware and software platforms, permitting the execution of number of operations simultaneously. This technology is basically based on client-server paradigms and concurrent programming is shown in Figure 6.5.

The advantages that a Web-based system offers over traditional legacy systems are:

- Reduced costs (adding new resources, training and maintenance).
- Improved access to the system through re-deployment and re-orientation of existing hardware and software resources.
- Anytime, anywhere secured access to users and customers.
- Easy access to users over the Internet since no extra hardware or software is required to access the application.
- Ease of maintenance from a Programming / Maintenance group perspective.
- User-friendly interface that requires minimal training / re-training.
- Ease in deployment and enhancement of functionality.
Figure 6.5 Web-based share computing mode

6.6 PERVASIVE COMPUTING FOR MONITORING ELECTRICAL DRIVE SYSTEM

Pervasive computing can be defined as “The overall infrastructural support needed to provide proactively a rich set of computing capabilities and services to a user every time everywhere in a transparent, integrated and convenient way”. Figure 6.6 shows the components of pervasive computing. Pervasive computing also called ubiquitous computing the growing idea towards embedding microprocessors in everyday objects so they imbedded with chips to connect the device to an infinite network of other devices. The idea that technology is moving beyond the personal computer to everyday devices with embedded technology and connectivity as computing devices become progressively smaller and more powerful. The words pervasive and ubiquitous mean "Existing Everywhere."
Pervasive computing is about four things; users, applications, middlewares and networks. In between Personal and pervasive computing there are three more evolutionary steps namely distributed, web and mobile computing. For successful implementation of pervasive computing, protocol layering, packet switching, information caching, distributed file and database systems and encryption systems are also needed in pervasive computing. Pervasive computing defines a major paradigm shift from “Anytime Anywhere” computing to “All-time Everywhere” computing. The support for pervasiveness will come from interoperability, scalability, smartness and invisibility on the top of mobility.
The technological advances that are needed to build a pervasive computing environment can be framed into four broad areas: devices, networking, middleware and applications as shown in Figure 6.7. New intelligent devices called smart devices are embedded with microprocessors that allow users to plug into intelligent networks and gain direct, simple and secure access to both relevant information and services. In case of mobiComp, distributed Computing, a shell of middleware is essential to interface between the pervasive network and end-user applications running on pervasive devices. Some of the functionalities of pervasive middleware are

**Figure 6.7 Framework for monitoring electrical drive system with pervasive devices**
smartness, context-awareness, proactivity, transparency, mobility management, invisible interface and adaptability.

6.7 PERFORMANCE COMPARISONS AND DISCUSSIONS

The performance analyses are carried out in distributed environment. The distributed architectural models for real-time monitoring electrical drive system has been constructed using different technologies like RMI, RMI-IIOP, CORBA, XML-RPC and Web service. The consolidated views of all these approaches and their performance analysis have been presented. The major factor that influences the performance of the proposed models is the Round Trip Time (RTT) that includes the convergence time. The RTT measures the time needed from the point when the client sends the request and receives the response. The RTT is measured for the different system networks that invoke the services.

Java socket is an endpoint for communication between two processes i.e., server process and client process running in the same network. RMI provides sophisticated distributed and platform-independent environment to solve the engineering problems. The location transparency is the key frame of CORBA/RMI-IIOP. Location transparency of the proposed model is the ability to access and invoke operations on the CORBA server object without needing to know where the object resides. The Web service provides platform independence, language independence and location independence (Gray 2004).

Two identically configured computers have been used, one acting as client and the other server, connected in 100Mbps switched network. Both systems are running Microsoft Windows XP Operating Systems. Both systems have the same settings containing Pentium core 2 duo processor with
3 GHz speed, 2GB RAM, Java 2 platform standard edition version 1.4, Java web Services developer pack version 1.5, Apache 2.2 server.

RTT has been measured for each request and response for various client applications by using above mentioned distributed technologies. The primitive data types like int, float, char, double, long, boolean, byte and string have been considered for analyzing the proposed models. To measure the performance of different models, different technologies have been used for implementation and a simple test client was prepared to obtain the desired performance results. The test has been carried out in the following different cases.

6.7.1 Server and Client on the same Machine (Local)

The performance analysis was carried where all the services and the clients are running in the same machine. RTT has been measured. The graph is plotted between the round trip time calculated and for the different systems, as shown in Figure 6.8.

![Figure 6.8 Performance result - on local invocation](image-url)
6.7.2 Server and Client on Different Machines (Remote Invocation)

The developed service and the client are running in different machines. RTT is measured for different cases and the graph is plotted and shown in Figure 6.9.

![Figure 6.9 Performance result - on remote invocation](image)

6.7.3 Server and Client on different Machines (Remote through Proxy Server Invocation)

The server and the client are running in different machines, where the remote machine is configured with firewalls. RTT is measured for different cases and the graph is plotted and shown in Figure 6.10.
Figure 6.10 Performance result - on remote invocation through proxy server

6.7.4 Different Web Services Invocation - Performance

The performance analysis was carried using the Web service, where the server and the clients are running in the same machine. RTT has been measured and the graph is plotted between the round trip time calculated and for the different system, as shown in Figure 6.11.

Similarly, the Web service model on remote invocation and proxy server invocation has been developed to carry out the real time monitoring of drive system. The performance analysis was carried out and RTT has been measured. It is found that the time taken to invoke the service and to run the results increase as number of client increases.
CONCLUSION

Since both SOAP and WSDL are XML-based, XML messages have to be parsed on both the sides. The client side proxies have to be generated on the client side before any communication can take place. XML parsing at runtime requires additional processing time which may result in longer response time of the server in case of a web service server. The overhead of the web service stems mainly from the usage of XML producing human readable text and is employed when interoperability with other web services and applications is essential. Even though web services suffer from poor performance compared to other distributed computing approaches such as RMI and CORBA, the Web enabled model provides a cross-platform, cross-language data model that facilitates developing heterogeneous distributed applications.

XML-RPC is found a useful way to tie together the systems written in different languages on different operating systems and enabling them to cooperate. The real advantage is that the structure of XML-RPC is flexible enough to be put to different engineering applications.