On the development of base specification for implementing Distributed Applications on Embedded Platforms

K.Chaitanya[1]
Department ofC.S.E
[1]Research Scholar in JNTU Hyderabad
Hyderabad, India

K. Rajasekhara Rao[2]
*Department of*C.S.E
[2]Director *in Sri Prakash college of Engineering, Tanuku,E.G.Dist.,India*

Abstract

Many loaded systems exist as on date which work on the principle of distributed processing and distributed data. Networking of the computing units and heterogeneity of the platforms that are used at respective computing locations are the main issues that are needed to be addressed in regard to implement distributed embedded systems. Middleware is generally used to address the heterogeneous issues. Communication is one of the important issues that must be addressed for implementing a distributed application. Many loaded applications have been developed and implemented in the past based on the concept of distributed computing.

Embedded systems are being used for implementing most of the real time applications. These days, the concept of distributed computing is also being extended for the implementation of applications that require usage of many embedded systems, each embedded system implementing a particular aspect of the real time processing; keeping in view of response time and need for extending the scope of real time processing

Not much work has been reported to be done in developing a network of embedded systems. Each embedded system is different as it may use different micro controllers, real time operating systems and programming languages used for the development of embedded software.

Thus it becomes necessary to bring out all the issues involved in developing a distributed application using embedded systems as a general requirement. This paper is related to the development of a specification document that brings out all the issues related to development of distributed embedded systems based applications.

Keywords

Distributed Embedded systems,Middleware,Networking issues,Performance issues

IINTRODUCTION

A. Theoretical foundation:

An embedded system is a special-purpose information processing systemthat is closely integrated into its environment. It is usually dedicated to acertain application domain.Theknowledge about the system behavior at the time of design can be used to minimize resources while maximizing predictability. Embedding into a technical environment and the constraints imposed by a particular application domain very often lead to heterogeneous and distributed implementations. In this case, systems are composed ofhardware components that communicate via some interconnection network.

Many embeddedsystem applications exist in day to day life. Some of the applications have been developed and implemented based on the concept of Distributed embedded systems. Automobile systems use the distributed embedded system in a more advanced way these days.

Distributed process is a valuable design alternative for the system to achieve the real time constraints. The distributed systems are more complex because they are integrated with heterogeneous network. These conditions lead to the concept of Middleware. The term middleware suggests that it is software positioned between the operating system and the application and can be spread itself over a heterogeneous network, concealing the complexity of underlying technology from the application being run on it.

Distributed embedded system can be defined in terms of, multiple computers interconnected by a network that share some common state and cooperate to achieve some common goal.Distribution generally leads to sharing of resources and sharingimproves availability, reliability, fault tolerance, maintainability, performance and scalability.

Many issues have to be considered while attempting to develop a distributed system which generally includeMiddleware, Communication, Performance,Networking, Heterogeneity and Security. Application segmentation and allocation of the application segments to various embedded systems is another important issue that needs consideration. A thorough understating of these issues is necessary so as to embark on the development of distributed embedded systems. Literature has not brought out general requirement specifications for the development of distributed embedded systems.

B. Problem Definition:

Keeping the correlations as aforesaid in view, the problem can be said as: To bring out a general requirement specification that must be considered each time, one proceeds to develop distributed embedded system. The general requirement specification must include all these issues at macro level for developing a distributed embedded system.

C. Scope:

The scope of this work includes the following,

1. Literature survey for determining the requirements which should be considered for development of distributed embedded system
2. Making comparative analysis of the recommendations.
3. Verification of various requirements projected by the authors to check whether all the development related issues have been projected by various other authors.
4. To comprehensively shortlist all the requirements that are to be considered for undertaking development of a distributed embedded system.

D.Limitations:

This work brings out general requirement specification which addresses various issues that must be considered while attempting to develop a distributed embedded system. Howeverthese issues must be fine-tuned and the applicability of the same must be verified each time a user attempts to develop specific distributed embedded system. The present scope of the work is limited to wired distributed embedded systems.

II LITERATURE SURVEY

Various issues have been referred from the literaturewhich are needed to be considered for undertaking development of distributed embedded systems. The issues that have been traced from the literature include Middleware issues, Communication issues, Performance issues, Networking issues, Heterogeneity issues, and Security issues. The survey conducted related to each of the issue is detailed below:

A.Survey related to Middleware Issues:

Middleware requirements

Middleware [8] is a class of software technology designed to help manage the complexity and heterogeneity inherent in distributed systems. It is defined as a layer of software, which is above the operating system but below the application program that provides a common programming abstraction across a distributed system.

The distributed systems create new problems that do not exist in centralized systems. It is obvious that new concepts and mechanisms are necessary, but there is no specific need to know at which level they should be embedded. From support at the hardware levels, all the ways for extension of programming languages to enable support of distributed applications. Software solutions typically provide the greatest flexibility because of their suitability for integrating existing technologies. These conditions lead to the concept of Middleware. Middleware offers general services that support distributed execution of applications. The term middleware suggests that it is software positioned between the operating system and the application. Viewed abstractly, middleware can be spread itself over a heterogeneous network, concealing the complexity of the underlying technology from the application being run on it.

Using middleware for distributed systems has several advantages viz.a.viz.it eases the distributed system development, increases the portability of the software and improves the system maintenance and reliability. The economic benefits of middleware are significant with up to 50% decrease reported in software development time and costs. Despite these benefits, general-purpose middleware poses numerous challenges when developing real-time systems.

Middleware services requirements

The main issue that must be addressed in regard to the middleware is related to the services that must be included into the middleware that essentially addresses the heterogeneity. Services that must be included into Middleware [10] fall into different categories. The categories include communication services (Procedure calls across networks, Remote-object method invocation, Message-queuing systems, advanced communication streams, Event notification service etc.), Information system services **(**Services that help manage data in a distributed system (Large-scale, system wide naming services, Advanced directory services (search engines, Location services for tracking mobile objects, Persistent storage facilities, Data caching and replication), Control services**(**Services giving applications control over when, where, and how they access data, Distributed transaction processing, Code migration), Security services(Services for secure processing and communication:, Authentication and authorization services, Simple encryption services, Auditing service)

Middleware object requirements

Various tasks/ objects that must also be included into the Middleware [11] have been presented into the literature.

An object encapsulates state and behavior and can only be accessed via a well-defined interface. The interface hides details that are specific to the implementation, thereby helping to encapsulate different technologies. An object therefore becomes a unit of distribution. Recall that objects communicate with each other by exchanging messages.

Various kinds of objects that must be used to form the middleware software include Model **support object (**Middleware should offer mechanisms to support the concepts incorporated in the object model), Operational interaction object **(**Middleware should allow the operational interaction between two objects. The model used is the method invocation of an object-oriented programming language), Remote interaction object **(**Middleware should allow the interaction between two objects located in different address spaces), Distribution transparency object (from the standpoint of the program, interaction between objects is identical for both local and remote interactions), Technological independenceobject**(**The middleware supports the integration of different technologies).

Middleware structure requirements

The structure of Middleware platform [12] plays a major role on which the middleware must be implemented.

Middleware is conceptually located between the application and the operating system (as in below figure2.1) the middleware hides the heterogeneity that occurs in a distributed system. This heterogeneity exists at different places:

Programming languages:

Different objects can be developed in different programming languages.

Operating system:

Operating systems have different characteristics and capabilities.

Computer architectures:

Computers differ in their technical details. (e.g.: data representation).

Networks:

Different computers are linked together through different network technologies.

Middleware overcomes this heterogeneity by offering equal functionality at all access points. Applications have access to this functionality through an Application Programming Interface (API).Because the API depends on the programming language in which the application is written, the API has to be adapted to the conditions of each programming language that is supported by the middleware.

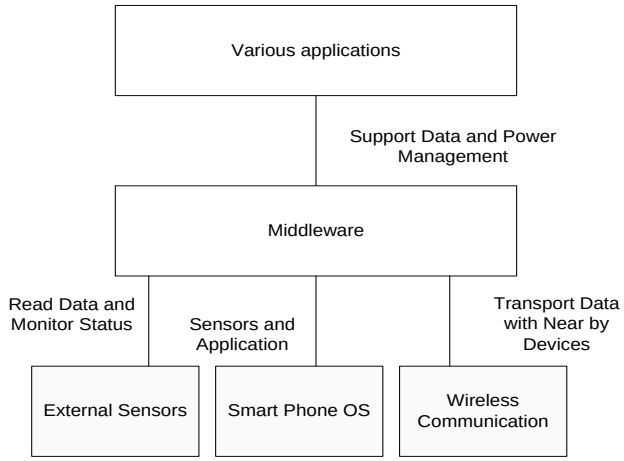

Figure 2.1: Middleware Architecture

An applications programmer typically sees middleware as a program library and asset of tools. The form these takes naturally depends on the development environment that the programme is using. Along with the programming language selected, this is also affected by the actual compiler/interpreter used to develop a distributed application.

If we were to project a middleware to a global, worldwide network, we would find special characteristics that differ from those of a geographically restricted distributed system. At the global level, middleware spans several technological and political domains, and it can no longer be assumed that a homogenous technology exists within a distributed system.

Due to heterogeneity and the complexity associated with it, we cannot assume that one vendor is able to supply middleware in the form of products for all environments. From the standpoint of market policy, it is generally desirable to avoid having the monopoly on a product and to support innovation through competition. However the implementation of middleware through several competing products should not result in partial solutions that are compatible.

Middleware – Portability and inter-operability issues

In the context of middleware, a standard has to establish the interfaces between different components to enable their interaction with one another. We want to distinguish between two types of interface: horizontal and vertical. Horizontal interface exists between an application and the middleware and defines how the application can access the functionality of the middleware. This is also referred to as Application Programming Interface (API).The standardization of the interface between middleware and application results in the portability[9] of an application to different middleware because the same API exists at each access point as shown in fig 2.2

In addition to the horizontal interface there is a vertical interface that defines the interface between

two instances of a middleware platform. This vertical interface is typically defined through a protocol on the basis of messages, referred to as protocol data units (PDUs).APDU is a message sent over the network. Both client and server exchange PDUs to implement the protocol.

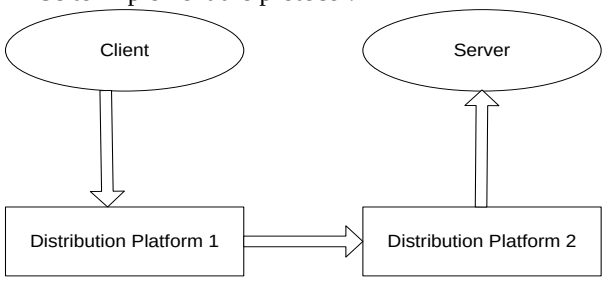

Figure 2.2 portability and interoperability

The vertical interface separates technological domains and ensures that applications can extend beyond the influence area of the product of middleware. The standardization of this interface allows interoperability between applications.

Applications programmers are typically only interested in horizontal interface because it defines the point of contact to their applications. From the view of the applications programmer, the vertical interface is of minor importance for the development of an application. Yet an implicit dependency exists between horizontal and vertical interfaces.

B.Survey related to Communication Issues:

Communication [3] is essential to achieving a dependable distributed embedded system. Designers of these systems are faced with several challenges in specifying the communication network. Complex systems usually require some sort of shared media network. In this environment, the designer must recognize the fundamental trade-off that exists between the efficiency and the predictability of the network. Given this trade-off, the designer must evaluate and select the communication network. Particular attention must be given to the protocols, which determine much of the network behaviour. Finally, many error detection methods are available which are necessary to build a reliable communication system.

The majority of embedded communication systems can be classified as either point-to-point networks (data links) or shared media networks (data highways). It is important to understand the trade-off between these two types of systems. In point-to-point networks, each node of the system is connected to every other node. These systems are simple and reliable. Reliability is high since correct transmission between two nodes only depends on a single transmitter and receiver. Since each link is dedicated to communication between two nodes, it is easy to meet real-time deadlines without any sophisticated scheduling mechanism. In shared media systems all nodes are connected together using a ring or bus topology. The primary motivation for shared media is the reduction in wiring (and thus cost). These networks are easily extendable without adding any new data ports to individual nodes. Limited new cabling is required.

Event versus State Based Communication

In practice communication systems may not be purely event or state based[1]. A communication protocol may contain some properties of each. However, it is instructive to examine the fundamental differences between an event based system and a state based system. One of the fundamental trade-offs between these two types of systems is the efficient use of resources found in event based systems versus the predictability of the network found in state based systems. The primary resources of concern in the network are bandwidth and the buffer space required at nodes to process incoming or outgoing messages.

In an event based communication system, messages are generated and transmitted in response to "events" detected at a local node in the network. Examples of "events" include changes in the value of process variables, new alarm conditions that have been detected, conditions that represent alarms clearing or requests by other nodes for data.
Messages are generated by users whenever they send data to printers, access data on shared network drives, run applications that exist on other machines or send email to others in the network.

One goal of event based communication is the efficient use of network bandwidth. By transmitting only necessary data, an efficient use of network bandwidth is assured. However, since data is transmitted only when there is a change at the source node, every message becomes important. This places additional requirements on the communication system to assure that each message is delivered successfully. One mechanism to do this is for destination nodes to acknowledge each successful transmission and request a retry for each corrupted message. If an acknowledgement is not generated within a specified timeout, the source node may also repeat the message of its own accord. Note that this acknowledges and retry mechanism consumes some additional network bandwidth.

In a state based communication system, messages represent the entire state of a node. For instance, all of the alarms for a node are transmitted as either on or off in its message. A node sends its fixed size message at pre-defined, regular intervals. Access to the media is easily scheduled, since the message requirements of each node never change. Network load is fixed and can be easily calculated during system design. An example of a state based system is a distributed process control system. Each node has a fixed number of inputs, calculated values, and alarm conditions that it sends in its message to other nodes in the network.

The state based system is a less efficient in terms of network bandwidth than in the event based system. Network bandwidth is sacrificed for the

predictability of regular message size and regular access to the communication channel. Note that some reduction in the overall data is possible. Each piece of data occupies a fixed location in the message. Therefore the data can be restricted to value. Information about what each data point represents is not required to be transmitted with the message.

State based systems can be designed to tolerate the occasional missed message. Re-transmission may not be necessary, since the entire state will be transmitted again in the next time interval. If messages are transmitted at twice the required frequency, the system can meet its deadlines even if every second message is corrupted. In order to tolerate two corrupted messages in a row, the each node could be designed to transmit its messages at three times the required frequency.

The single most important difference between a distributed system and a uniprocessor system is the inter process communication. In a uni-processor system, most inter process communication implicitly assumes the existence of shared memory.Communication is mainly divided into two types. They are,

Transient communication

A message is discarded by a communication server as soon as it cannot be delivered at the next server, or at the receiver.

Persistent communication

A message is stored at a communication server as long as it takes to deliver it at the receiver.

Communications are often handled by layered protocols. Protocols are formal set of rules that govern the format, contents, and meaning of the messages send and received. Connection oriented and connectionless protocols. Connection oriented does some initial work to set up a "virtual circuit". Connectionless does not.

Communication: Middleware Protocols:

Middleware logically at application layer, but provides commonservices and protocols that can be used by manydifferent applications.

1. A rich set communication protocols to allow different applications to communicate
2. Naming protocols, so that different applications can easily share resources.
3. Security protocols, to allow different applications to communicate in a secure way.
4. Scaling mechanisms, such as support for replication and caching.

Transmission modes:

Different timing guarantees with respect to data transferAsynchronous (No restrictions with respect to when data is to be delivered), Synchronous (Define a maximum end-to-end delay for individual data packets), Isochronous (Define a maximum and minimum end-to-end delay (**jitter** is bounded)).

1. Pack parameters and other information into a message (marshalling).
2. Send message to process on remote machine.
3. Unpack message on remote machine (un-marshalling).

C. Survey related to Performance Issues:

[*Richard, Zurawski*] have presented several performance related issues that must be considered for the development of distributed embedded system especially the design related issues [2] [7].

Correctness:

The results of the analysis should be correct, i.e. there exist no reachable system states and feasible reactions of the system environment such that the calculated bounds are violated.

Accuracy:

The lower and upper bounds determined by the performance analysis should be close to the actual worst case and best case timing properties.

Embedding into the design process:

The underlying performance model should be sufficiently general to allow the representation of the application (which possibly uses different specification mechanisms), of the environment (periodic, aperiodic, busty, different event types), of the mapping including the resource sharing strategies (pre-emption, priorities, time-triggered) and of the hardware platform. The method should seamlessly integrate into the functional specification and design methodology.

Short analysis time:

Especially, if the performance analysis is part of a design space exploration, a short analysis time is important. Inaddition, the underlying model should allow for configurability in terms of application, mapping and hardware platform.

As distributed systems are heterogeneous in terms of the underlying execution platform, the diverse concurrently running applications, and the different scheduling and arbitration policies used, modularity is a key requirement for any performance analysis method. We can distinguish between several composition properties.

Process Composition:

Often, events need to be processed by severalconsecutive application tasks. In this case, the performance analysismethod should be modular in terms of this functional composition.

Scheduling Composition:

Within one implementation, differentscheduling methods can be combined, even within one computingresource (hierarchal scheduling); the same property holds for thescheduling and arbitration of communication resources.

Resource Composition:
A system implementation can consist of different heterogeneous computing and communication resources. It shouldbe possible to compose them in a similar way as processes and scheduling methods.

Building Components:
Combinations of processes associated scheduling methods and architecture elements should be combined into components. This way, one could associate a performance component to acombinedhardware/OS/software module of the implementation that exposes the performance requirements but hides internal implementation details.

1. Performance loss due to communication delays:
   - Fine-grain parallelism: high degree of interaction
   - Coarse-grain parallelism
2. Performance loss due to making the system fault tolerant.

D. Survey related to Networking Issues:

[*.cs.wpi.edu/~cs535/s08/week3-comm*] have presented several performance related issues that must be considered for the development of distributed embedded system especially the design related issues[4] are Performance(latency and data transfer rate (throughput))**,**Scalability**,** Reliability(high for physical networks, with most errors due to software or buffer overflow),Security(common use of a firewall to filter messages, encryption, Virtual Private Network (VPN) does encryption),Mobility(big and growing issue),Quality of service(applications have different demands, no longer just "best effort")Multicasting(sends to a group).

2.4.1 Performance issues:

Latency:
The delay that occurs after a send operation is executed before data starts to arrive at the destination computer. It can be measured as the time to transfer an empty message. It forms a part of process-to-process latency.

Data transfer rate:
The speed at which data can be transferred between two computers in the network once transmission has begun, bits/s.
Message transmission time = latency + length / data transfer rate.

Data transfer rate is determined primarily by network physical characteristics, whereas the latency is determined primarily by software overheads, routing delays and delay of accessing to transmission channels. In distributed systems, messages are always small in size, so the latency is more significant than data transfer rate.

Total system bandwidth:
The total volume of traffic that can be transferred across the network in a given time.

- **Ethernet:** system bandwidth is as same as data transfer rate.
- **WAN:** multiple channels, deteriorates when there are too many messages.

Comparison of different communication channel,

- **Local network** – a null message transmission time is under a millisecond.
- **Local memory** - 1000 or more times faster than local network.
- **Local hard disk** - 500 times slower than fast local network.
- **Internet** - round-trip latencies are in 300-600ms due to switching and contention.

2.4.2 Survey related to Scalability:

- No traffic figures are available for the internet, but the impact of traffic on performance can be gauged from communication latencies.
- future Internet: several billion nodes and hundreds of millions of active hosts, new addressing and routing mechanisms
- the ability of the internet infrastructure to cope with this growth will depend upon the economics of use, in particular charges to users and the patterns of communication that actually occur. ex.: degree of flexibility.

2.4.3 Survey related to Reliability:

Networks are highly reliable, whereas client and server computers and their software often aren't, so error detection and recovery is best performed end-to-end at the highest feasible level.(Errors: e.g. failures in software of sender or receiver, or buffer overflow).

2.4.4 Survey related to Quality of Service:

It is the ability to meet deadlines when transmitting and processing streams of real-time multimedia data.Require guaranteed bandwidth and bounded latencies.

Multicasting:
Multicasting can be simulated by sends to several destinations, this is more costly than necessary, and may not exhibit the fault-tolerance characteristics required by applications. It can be need for one-to-many communication.

E. Survey related to Security Issues:

Firewall: to protect the resources in all of the computers inside the organization from access by external users or processes and to control the use of resources outside the firewall by users inside the organization, always runs on a gateway.Secure

network environment (e.g. VPN (virtual private network).

Security[5]is a generic term used to indicate several different requirements in computing systems. Depending on the system and its use, several security properties may be satisfied in each system and in each operational environment. Overall, secure systems need to meet all or a subset of the following requirements:

Confidentiality:
Data stored in the system or transmitted from the system have to be protected from disclosure; this is usually achieved through data encryption.

Integrity:
A mechanism to ensure that data received in a data communication was indeed the data transmitted.

Non repudiation:
A mechanism to ensure that all entities (systems or applications) participating in a transaction cannot deny their actions in the transaction.

Availability:
The system's ability to perform its primary functions and serve its legitimate users without any disruption, under all conditions, including possible malicious attacks that target to disrupt service, such as the well known Denial of Service (DoS) attacks.

Authentication:
The ability of the receiver of a message to identify the message sender.

Access control:
The ability to ensure that only legal users may take part in a transaction and have access to system resources. To be effective, access control is typically used in conjunction with authentication.These requirements are placed by different parties involved in the development and use of computing systems, for example, vendors, application providers, and users.

IIICOMPARATIVE ANALYSIS

The various issues and the requirements are initiated along with its associated properties as suggested by different authors for developing a distributed embedded system are tabulated and placed in the table 3.1.

Table 3.1 Comparative Analysis – Requirements projection for distributed Embedded

S.No	Issues	Richard Zurawski	Andrew S. Tanenbaum	Koopman	Shibu	Rajkamal
1	**Middleware**	X	√	√	X	X
	Horizontal interface					
	Vertical interface					
	RTOS					
	Micro Controller					
	Languages					
2	**Communication**	√	√	√	√	√
	Protocols					
	RS232C					
	I^2C					
	USB					
	Fire wire					
	Ethernet					
	ESA					
	PCI					
	Direct Connecting					
	IrDA					
	Wi-Fi					
	Hi-Fi					
	Bluetooth					
	Any other					
	Protocol conversion					
3	**Performance**	√	√	X	X	X
4	**Networking**	X	X	X	√	√
	Topologies					
	Interfacing					
	RS232C					
	I^2C					
	USB					
	Ethernet					
	ESA					
	PCI					
	Direct Connecting					
	IrDA					
	Wi-Fi					
	Hi-Fi					
	Bluetooth					
	Any other					
5	**Heterogeneity (In the absence of Middleware)**	X	√	X	X	X
	RTOS					
	Micro Controller					
	Languages					
6	**Security**	√	√	X		
7	**EDLC at each of the Embedded Systems**				√	
8	**Design Models to be used**				√	√
	State Machines				√	√
	Sequential Flow				√	√
	Data Flow				√	√
	HW and SW Co-design				√	√
	Clean room					

	software engineering models					
9	**Design models for undertaking Testing**	X	X	X	X	X

IV INVESTIGATIONS AND FINDINGS

Thus the requirement specifications for developing a distributed embedded application are investigated.The issues that have been traced from the literature include Middleware issues, Communication issues, Performance issues, networking issues, and Security issues. While developing any distributed application, Middleware plays an important role for heterogeneity services.

Distributed systems are mainly used for sharing of resources and messages through the network connections. Thus the communication between the systems is done through the messages.

Recommended Requirement Specification for development of distributed Embedded Systems
Middleware, Communication, Performance
Networking, Security

VCONCLUSION

The survey of literature revels that those different types of issues being used for developing an effective distributed embedded systems. Distributed systems are mainlyused for sharing of resources between the systems/devices and they are all combined together to form a single application. These sharing of resources are done through the messages with the help of network connections. Thus it becomes necessary to bring out all the issues involved in developing a distributed application using embedded systems as a general requirement specification.

REFERENCES

[1]. [Kopetz97] Kopetz, H., Real-Time Systems, Design Principles for Distributed Embedded Applications, Klower Academic Publishers, 1997, Chpt.7-8.

[2]. Ti-Yen Yen and Wayne Wolf, "Performance Estimation for Distributed Embedded System", IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, VOL. 9, NO. 11, NOVEMBER 1998

[3]. [Koopman94] Koopman, P.J., and Upender, B.P, "Communication Protocols for Embedded Systems", Embedded Systems Programming, 7(11), November 1994, pp. 46-48,http://www.cs.cmu.edu/People/koopman/protsrvy/protsrvy.html, Accessed: May 8, 1999.

[4]. http://web.cs.wpi.edu/~cs535/s08/week3-comm.pdf

[5]. Richard,Zurawski , Embedded Systems Hand Book,2006,chapter17

[6]. distributed operating system by Andrew S.Tanenbaum

[7]. Richard,Zurawski , Embedded Systems Hand Book,Performance Analysis of Distributed Embedded Systems,Lothar Thiele and Ernesto Wandeler,2006,chapter15

[8]. Wayne Wolf,"Middleware Architectures For Distributed Embedded System",11th IEEE Symposium on Object Oriented Real-Time Distributed Computing,2008,pp.377-380

[9]. Arno Puder,KeyRomer& Frank Pilhofer, Distributed systems:A Middleware Approach,Elsevier,2006,24-25

[10]. Liu JingYong, Zhang LiChen, Zhong Yong and Chen Yong, Middleware-based Distributed Systems Software Process, International Journal of Advanced Science and Technology Volume 13, December, 2009

[11]. Arno Puder,KeyRomer& Frank Pilhofer, Distributed systems:A Middleware Approach,Elsevier,2006,21-22

[12]. Arno Puder,KeyRomer& Frank Pilhofer, Distributed systems:A Middleware Approach,Elsevier,2006,22-24