Proceedings of the 17th World Congress
The International Federation of Automatic Control
Seoul, Korea, July 6-11, 2008

Robot System providing Fault-Tolerant Services: Research of Middleware supporting Fault-Tolerance

Bum Hyeon Baek, Hyeong Seob Choi, Hong Seong Park*, Sung Hoon Kim and Jung Bae Kim**

** Electronics and Telecommunication Engineering Department ,Kangwon National University Chuncheon, Korea (e-mail:{bhbaek,21thbomb, gazam}@control.kangwon.ac.kr, hspark@kangwon.ac.kr).*
*** Robot Software Architecture Research Department, Intelligent Robot Research Group, Electronics and Telecommunication Research Institute, Daejeon, Korea (e-mail:{saint, jjkim}@etri.re.kr)*

Abstract: Recently, robot technology is actively going on progress to the field of various services such as medical care, entertainment. These service robots are in use for home management nearby person, and so need to operate safely. Fault tolerance is a performable capacity without influence of fault although fault is occurred hardware or software and guarantees safe operation of systems. This paper proposes a robot system providing fault-tolerant services in distributed environment. The systems are developed to apply to robot middleware for supporting fault-tolerance. The robot middleware is divided into three layers of a Service Layer (SL), Network Adaptation Layer (NAL), Network Interface Layer (NIL) and includes Operating System Abstraction Layer (OSAL), Fault-Tolerant Manager (FTM). Especially, Service-Adaptive Engine (SAE) in SL and Fault-Tolerant Manager provides fault tolerance for this middleware and are easy to dynamic expansion. Also, these systems are component-based structure, and so provide reusability, lightweight to load various robot systems.

1. INTRODUCTION

Recently, robot technology is actively going on progress to the field of various services such as medical care, entertainment. The applicable service robot in the field of service is robot to provide a person with convenient service of every kind and is in use in various places, especially home management. The service robot for home management is important that it performs its operation safely because it operates nearby person. Because service robot often stays near person, abnormal operation of service robot is possible to gives wound to person. For example, if guidance robot has hardware or software fault when guidance robot show around the way, a person may be wounded cause by collision between the robot and the person. Therefore, research is necessary that it prevent abnormal operations and it make performs normal operation if fault is occurred in service robot.

Fault tolerance is a performable capacity without influence of fault although fault is occurred hardware or software. In other words, although it couldn't cope with every fault or error, it has a capacity which copes with prior defined fault or error at design time. When service robot includes fault tolerance functionality, there are some points to be considered in relation to characteristic of service robot. First of all, service robot need structure supporting network and communication functionality in heterogeneous environment so that it forms network and operates in network and is applied to various network environments. Also, it should enable various data to process according to accommodate various network interfaces. It is necessary to provide flexibility using available network interface when network session is closed by fault occurrence suddenly. Secondly, service robot internally needs an apt structure for providing various services in robot and should guarantee safe operation of service robot by means of granting fault tolerance in service robot.

Many researches have been studied for supporting fault tolerance in distributed systems. Till now there are systems supporting fault tolerance which is used in like as FT-CORBA (Fault-Tolerant CORBA: Common Object Request Broker Architecture), ROAFTS (Real-time Object-oriented Adaptive Fault Tolerance Support), Rainbow framework (Architecture-Based Self-Adaptation with Reusable Infrastructure). Most of these systems use redundancy or replication mechanism of their own for supporting fault tolerance and are based on TCP/IP in order to recover faults caused in network. But these systems are used in general-purpose distributed systems, these are difficult to apply to service robot since robot system is used and is operated in limited environment. There are various CPU from 8bit series to 32bit series for using in robot system and most of memory to load robot software is restricted within size of maximum 32Mbyte or 64Mbyte. Therefore, execution code is large and memory consumption is so much at run time like as these systems are difficult to use in robot.

Therefore, fault tolerant system should be lightweight to apply to robot. This paper defines lightweight in robot system. The lightweight is said that execution code size is less than 500Kbyte and execution memory the mount used is less than 4Mbyte. Because, robot software is composed of various application and software, fault tolerant system with these sizes in part of robot software is suitable. Also, fault tolerant system make reduce overhead of message which is transmitted to network and should manage network session to cope with abnormal close of network. In this paper, we propose and implement fault tolerant systems which may be capable to use robot middleware in distributed robot environment. These systems provide fault tolerance with robot middleware and are easy to dynamic expansion. These systems are component-based structure, and so provide reusability, lightweight to load various robot systems.

This paper structured as follows: Section 2 introduces middleware for robot and describes middleware's functionality, Section 3 describes architecture providing various services, Section 4 describes fault-tolerant system for robot, Section 5 describes implementation of these systems. Conclusions are drawn in outlined in Section 6.

2. MIDDLEWARE STRUCTURE FOR MODULE-BASED ROBOT

This paper mentioned middleware is to load module-based robot architecture. Each module present a part of robot structure that takes charge main function of robot addition, delete, exchange though do purpose put. Physical, electrical, logical interface between module independent use surrounding and each module is important. Each module include network interfaces like as Ethernet, IEEE1394, CAN, RS232, and so is capable to connect other interfaces at any time. Figure 1 shows example of construction among modules.

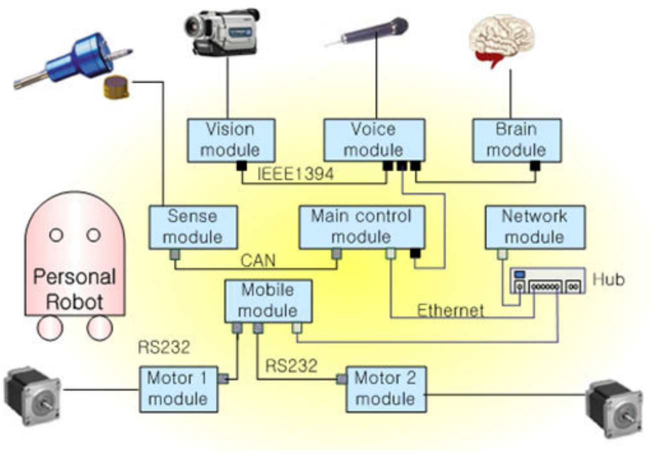

Fig. 1. Example of internal structure of Module-Based Robot

This paper mentioned middleware structure for robot is depicted in Figure 2. Robot middleware is consisted of Service Layer (SL) which provide service with robot application, Network Adaptation Layer (NAL) which accommodate various network, Network Interface Layer (NIL) which takes charge of dependent functionality of network, Operating System Abstraction Layer (OSAL) which abstracts functionality and wraps APIs of OS, Fault-Tolerant Manager (FTM) which provide fault tolerance with robot middleware and application.

The Service Layer is consisted of essential elements about application which uses middleware service. Major functionality of the Service Layer provides mechanism to access application of remote module irrespective of network type. In other words, the Service Layer provides services like as variable reading, method invocation in remote module and performs application management, transaction management. The Service-Adaptive Engine from among proposed systems in this paper exist Service Layer, and so performs these functions.

The Network Adaptation Layer integrates various network components which is added to Network Interface Layer and provides various services like as message routing, module addressing among heterogeneous networks, naming, etc. And this layer provides interfaces between Service Layer and Network Interface Layer which are consisted of data entity for data transmission/receipt, management entity for command transmission/receipt.

The Network Interface Layer includes various network components which are dependent on hardware and software. The Network Interface Layer enables middleware to add or delete new network without modification of middleware.

The Operating System Abstraction Layer provides OS-independent and abstracts system-dependent function like as thread, timer, event provided by OS, and so enable user or programmer to use OS APIs without distinction.

The Fault-Tolerant Manager manages components in robot middleware and application and observes network and communication between modules. Also, the Fault-Tolerant Manager manages component's status like as initialization/loading/configuration/finalization and enable component to communicate between components using internally defined event, and so enable it to observe component's status in real-time. Also, the Fault-Tolerant Manager provides exception handling, and so is capable to cope with exception. And it observes network session, and so prevents transmission error which is possible during data transmission/receipt. In other words, it enables middleware to connect between modules continuously.

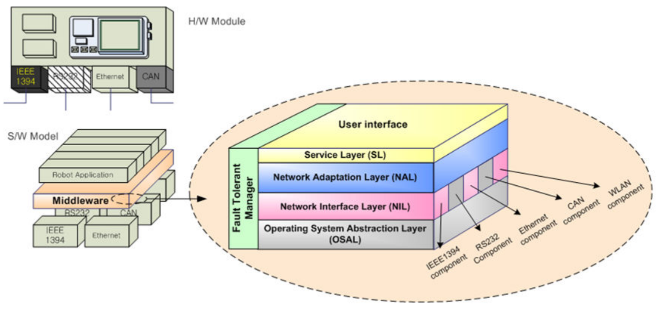

Fig. 2. A Structure of Robot Middleware

As depicted in Figure 2, structure of robot middleware is easy to add network interface if user wants. When application developers access variable or object in remote or local network, they are able to design and implement application using middleware irrelevant to network type.

Figure 3 shows detailed middleware structure, middleware structure for robot is suitable supporting open interface.

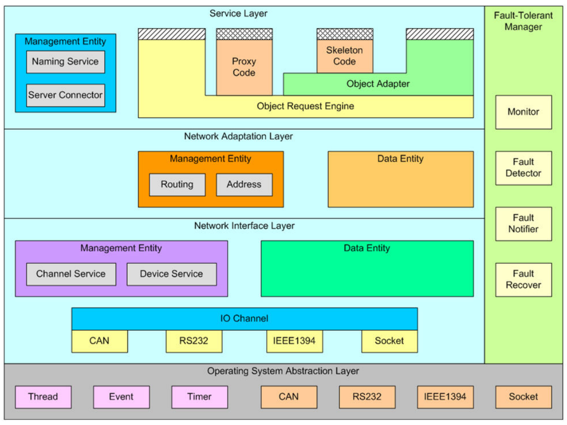

Fig. 3. A detailed Structure of Robot Middleware

3. A STRUCTURE OF SERVICE-ADAPTIVE ENGINE (SAE)

Figure 4 shows a detailed structure of Service Layer (SL). The Service Layer is consisted of essential elements about application which uses middleware service and Service-Adaptive Engine (SAE) is a core of SL. Major functionality of the SAE provides mechanism to access application of remote module irrespective of network type. The SAE manages APIs which enables robot application to use middleware, and so robot application is easy to register in SL.

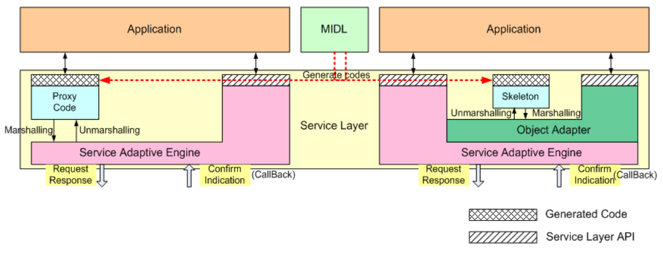

Fig. 4. A Structure of Service Layer and Service-Adaptive Engine

Robot applications are capable to be offered the middleware service after robot applications is registered in SL. For example, object invocation service requests the SL in remote or local module when request side SL want to access receipt side SL, and waits response. If requests of the object invocation are transferred at once, responses corresponds to the requests should be provided the request much. But, response corresponds to the request is hard to identify when response is received, if transactions for response are many. To prevent these faults or errors, the SL manages transaction per each request and handles request with high priority, and so guarantees QoS.

The request of application should be capable to transfer to network so that local application access to remote application. But, to transfer the message using application level to network lead to ambiguous situation when receipt side handles data. For example, when brain module invokes method which is service ‘b’ in remote module like as actuator module, message like as “actuator.b” is transferred to actuator module. But, actuator module does not know that this message handles either general string or service invocation. Also, when application transfers the message of application level, special character like as blank character, delimiter may incur overhead. Therefore, to transfer message to network, it is necessary that data requested by application need to represent by the moderate form.

A process of message representation could be solved by using proxy. Proxy is a agent for object invocation and is generated from IDL (Interface Definition Language) definitions and, therefore specific to the types of objects and data you have defined in IDL. The proxy code has tow major functions:

- It provides a down-call interface for the client. Calling a function in the generated proxy API ultimately end up sending an RPC message to the server that invokes a corresponding function on the target object.
- It provides marshaling and unmarshaling (encoding and decoding data for transmission) code.

Marshaling is the process of serializing a complex data structure, such as a sequence or a dictionary, for transmission on the wire. The marshalling code converts data into a form that is standardized for transmission and independent of the endian-ness and padding rules of the local machine. Unmarshaling is the reverse of marshalling, that is, deserializing data that arrives over the network and reconstructing a local representation of the data in types that are appropriate for the programming language in use.

The proxy enables developer to support for interfacing with application in system core and to support through IDL. Mostly, the latter is many used, and so this paper provides IDL for convenience of application development and provides IDL pre-compiler for code generation from IDL. Interfaces, operations, and the types of data that are exchanged between modules are defined using the MIDL (Module IDL) language. MIDL allows you to define the client-server contract in a way and the MIDL definitions are compiled by a MIDL compiler into an API for a specific programming language, that is, the part of the API that is specific to the interfaces and types you have defined consists of generated code. Finally, user obtains proxy codes such as skeleton which is used by server application, stub which is used by client application. Figure 5 shows the situation when both client and server are developed in C++. The MIDL compiler generates two files from a MIDL definition in a source file “hello.midl”: header files (hello_proxy.h, hello_skeletion.h).

- The hello_proxy.h header file contains definitions that correspond to the types used in the MIDL definitaion. It is included in the source code of client to ensure that client agrees about the types and interfaces used by the application.
- The hello_skeleton.h header file also contains definitions that correspond to the types used in the MIDL definition. It is included in the source code of

server to ensure that server agrees about the types and interfaces used by the application.

Finally, client and server application request and offer service with these file.

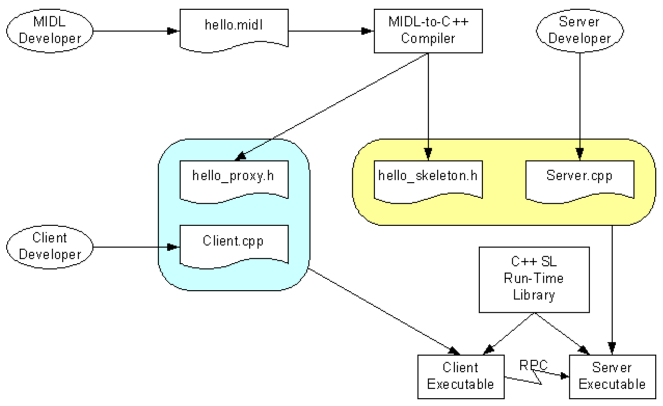

Fig. 5. Development process if client and server share the same development environment

The Service-Adaptive Engine is a logical bus that substantial performs service request and offer. Figure 6 illustrates how the server application and client application are operated for object invocation.

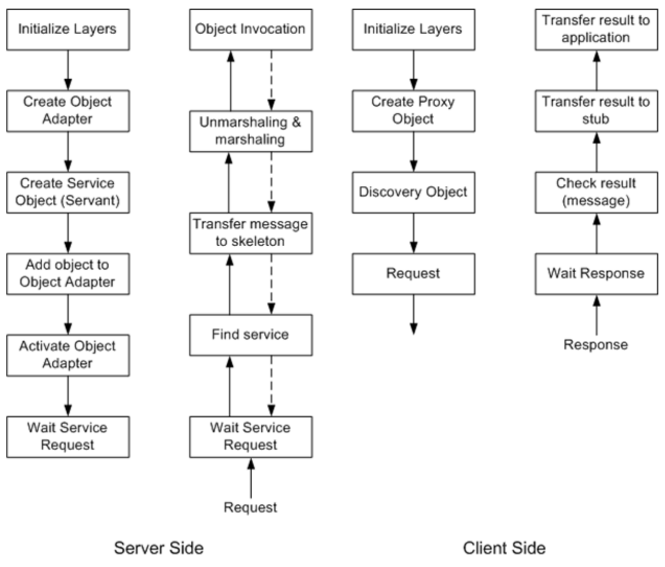

Fig. 6. An operation flow of server and client application

Server application initializes each layer of middleware by calling initialization function. And it creates an object adapter and create object called by servant. Finally, it activates object adapter and waits request from client application.

When the request message incomes through the client application, the SL receives message and analyzes. After message analysis, the SL sends message to skeleton. When skeleton receive message from the SL, skeleton send message to server application. Finally, server application invokes service and responds to request. And the application send response message to the SL and the SL send message to network.

Client application also initializes each layer of middleware by calling initialization function. Next, to actually talk to servant, client application creates a proxy object for remote object invocation and it discovers servant. If so, the call returns a proxy to an object; otherwise, if the proxy denote an interface of some other type, the call returns exception message such as “object not exist”, “operation not exist”. And the SL may re-discover the service in robot network.

If object discovery is succeeded, client application send request message to server application and waits response. After request is normally performed from server application, client application receives response such as “success”, exceptions.

4. A STRUCTURE OF FAULT-TOLERANT MANAGER (FTM)

The Fault-Tolerant Manager (FTM) is a component which provides fault tolerance with middleware and robot application. In other words, the FTM manages objects in middleware and application from fault which could be occurred in middleware or application. At this point, fault tolerance is said that is mainly used when is designed computer system or its elements. Fault tolerance enables system to avoid service suspension using by altering preliminary element or procedure. Fault tolerant service is provided by software, but it may be provided embedded form in hardware or a coupled form. This paper only considers the side of software.

The software implementation for fault tolerance enables programmer to check important data at a point of determined time in advance. Like this, fault-tolerant manager observes entity which performs transaction and independently recovers fault or error from system when fault or error is occurred.

The entity performing transaction in middleware and application is mainly task or object. These provide service in substance and performs operation related it. Therefore, it is basic matter that task and object are checked their status and life cycle. Also, network connection management is important because these interoperate though network and transfer or receive data. And system should be capable to re-connect though network management when network connection is suddenly closed. Finally, dependency between modules should be considered module-based robot, and modules should discover and use without a hitch. The FTM associating with these requirements guarantees safety from fault of system.

The FTM is designed component-based feature which is reusable, because any software requiring fault tolerance may use it. The FTM is a composite component, and is composed of Monitor, Fault Detector, Fault Notifier, Fault Recover. Monitor observes every object in middleware and application, and Fault Detector detects occurred fault and analyze fault type. And Fault Notifier send fault information from Fault Detector to Fault Recover and performs scheduling according to fault type. Finally, Fault Recover independently recovers fault from system. Figure 8 illustrates a structure of FTM.

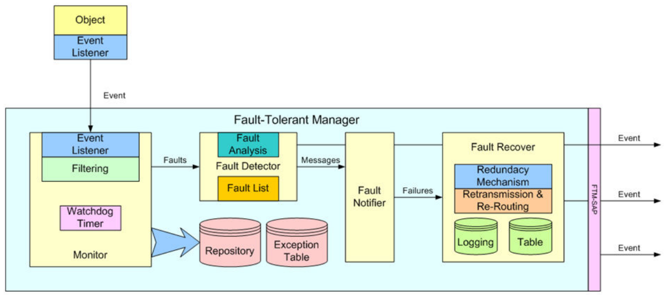

Fig. 7. A Structure of Fault-Tolerant Manager

From among these components, Monitor take charge of a preliminary operation related fault detection. Monitor periodically observes service object in middleware and application, taking charge of network session, and so grasps life cycle and status of objects. Life cycle and status of object is grasped using two models: push model and pull model.

In the push model, the direction of control flow matches the direction of information flow. With this model, monitorable objects are active. They periodically send heartbeat messages to inform other objects that they are still alive. If Monitor does not receive the heartbeat from a monitorable object within specific time bounds, it starts suspecting the object.

In pull model, information flows in the opposite direction of control flow, i.e., only when requested by consumers. With this model, monitored objects are passive. The monitors periodically send liveness requests to monitored objects. If a monitored object replies, it means that it is alive. Figure 8 illustrates how the push model and pull model are used for object.

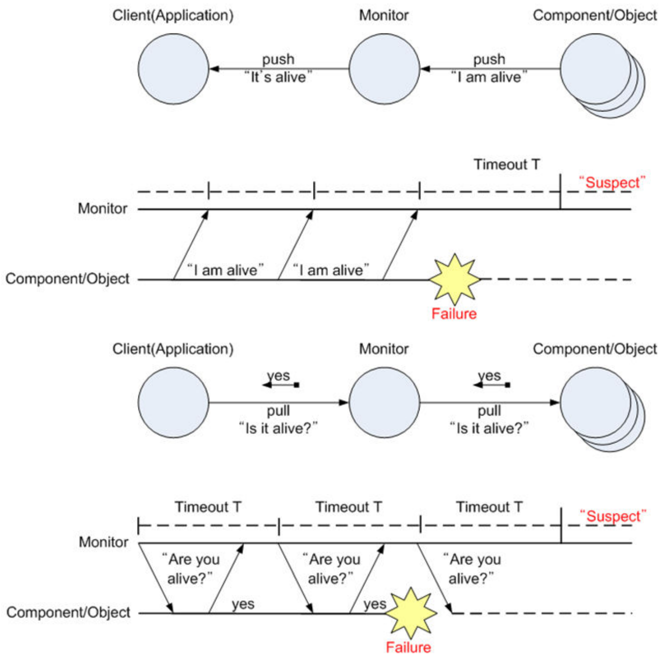

Fig. 8. A push/pull model

To transfer status between objects, the FTM provides event type and event transmission is achieved using publish/subscribe mechanism. Normal event transmission between objects is achieved in 0.85us and filtered event transmission is achieved in 1.02us. Event type is defined by each publisher and is transferred to subscriber. Subscriber registers published event using Event Listener which is included in subscriber and intercepts events. Every object in middleware and application is publisher, and so send event to Monitor. Monitor is a subscriber and is a publisher because of transmission of event to Fault Detector the same time. And Monitor uses watchdog timer, and so commits fault handling to Fault Detector when status event such as heartbeat is not received within specific time bounds.

The Fault Detector detect potential fault and report using indicated fault list at design time. Figure 9 illustrates a detailed structure of Fault Detector.

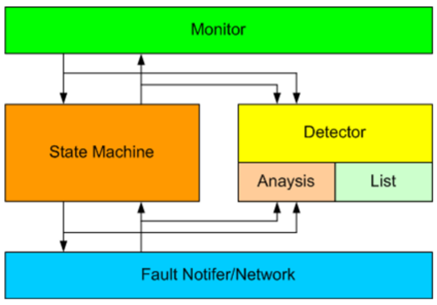

Fig. 9. A Structure of Fault Detector

The Fault Detector is divided into Finite State Machine (FSM) and Detector. FSM includes status macro and methods of each object. Detector decides existence and non-existence of fault occurrence comparing between indicated fault list and occurred fault and analyzes occurred fault. This progress is called by fault isolation, which analyzes fault reports and decide what or where has failed. After fault analysis, Detector transfer information about fault to Fault Notifier, and so make Fault Recover recovers the fault.

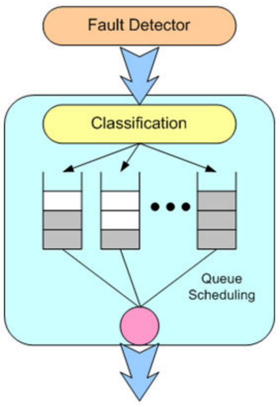

Fig. 10. A Structure of Fault Notifier

5. CONCLUSIONS

A conclusion section is not required. Although a conclusion may review the main points of the paper, do not replicate the abstract as the conclusion. A conclusion might elaborate on the importance of the work or suggest applications and extensions.

REFERENCES

A. Avizienis, J. C. Laprie, B. Randell, C. Landwehr, “Basic Concepts and Taxonomy of Dependable and Secure Computing,” *Journal of Dependable and Secure Computing IEEE Transaction*, Vol. 1, Issue 1, pp. 11-33, Jan, 2004.

FT-CORBA specification. http://www.omg.org

K.H (Kane) Kim, “ROAFTS: A Middleware Architecture for Real-time Object-oriented Adaptive Fault Tolerance Support,” Proc. *IEEE CS 1998 High-Assurance Systems Engineering (HASE) Symp.*, WashinTon, D.C., pp. 50-57, Nov, 1998.

David Garlan, Shang-Wen Cheng, An-Cheng Hyang, Badley Schmerl, Peter Steenkiste, “Rainbow: Architecture-Based Self Adaptation with Reusable Infrastructure,” *Computer*, Vol. 37, no. 10, pp. 46-54, October, 2004.

Geon Yun, Hyeong-Yuk Kim, Hong Seong Park, “Middleweare Structure for Module-based Personal Robot,” *Journal of Control, Automation and Systems*, Vol. 10, No. 5, pp. 464-474, May, 2004.

IEEE standard for a High Performance Serial Bus “IEEE std 1394-1995, IEEE 1394 std 1394a-2000”.

Universal Serial Bus Specification revision 1.1: September 23, 1998.

CAN specification Part A and Part B.

Felber, P. Defago, X. Guerraoui, R, Oser, P, “Failure Detector as First Class Objects,” *Distributed Objects and Applications, 1999. Proceedings of the International Symposium*, pp. 132-141, 5-6 Sept, 1999.

Lightweight Fault Tolerance for Distributed Real-Time Systems Request for Proposals. http://www.omg.org

Michi Henning, Mark Spruiell. 2006. Ice Specification: Distributed Programming with Ice, http://www.zeroc.com

E.Gamma, R. Helm, R. Johnson, and K. Vlissides, *Design Patterns: elements of Reusable Object-Oriented Software*, Reading, MA: Addison-Wesley, 1995.

F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal, *Patten-Oriented Software Architecture – A System of Patterns*, J. Wiley and Sons Ltd., 1996.

D. C. Schmidt, M. Stal, H. Rohnert, and F. Buschmann. *Pattern-Oriented Software Architecture – patterns for Concurrent and Distributed Objects*. J. Wiley and Sons Ltd., 2000.