Towards a Middleware for Distributed CAN Applications

Hasina M. Abdu and David H. Yoon
University of Michigan-Dearborn

ABSTRACT

With the increasing popularity of distributed and open frameworks, the trend of connectivity has extended to real-time control systems such as CAN. A generic middleware for CAN will facilitate the design, implementation and management of distributed applications for CAN systems, instead of programmers having to build custom made applications. This paper describes current approaches to tailoring existing middleware frameworks to the specific features of embedded systems in general, and CAN in particular.

INTRODUCTION

In response to advances in technology and the increasing complexity of distributed systems and applications, the concept of middleware was created to hide the underlying networked environment's complexity by protecting users/applications from explicit protocol handling, disjoint memories, data replication, network faults and parallelism [26]. In this context, middleware systems consist of the software layer between the operating system and the distributed applications that interact via the underlying network. In addition, middleware masks the heterogeneity of computer architectures, operating systems, programming languages, and networking technologies to facilitate application programming and management, thus resulting in truly distributed platforms. Examples of middleware include: OMG's CORBA, JINI, Java RMI and Microsoft's DCOM.

Originally designed for enterprise systems, the middleware approach and the trend of connectivity and internet-based applications is now shifting to the world of real-time control systems such as CAN. As such, various attempts have been made to establish the foundations for a middleware that can satisfy the stringent features of embedded systems, and, at the same time, maintain connectivity with existing and future enterprise systems. For example, the need for open frameworks to connect household wireless devices is motivated in [9]. This paper motivates the need for a generic middleware approach for real-time control systems in general, and CAN in particular. Existing approaches to such a middleware are described, followed by a description of the approach to be taken by the authors.

MIDDLEWARE FOR CAN: motivation

Figures 1 and 2 illustrate possible applications involving CAN controllers in cars, other real-time control systems and enterprise systems. In Figure 1, the route of the aircraft can be remotely controlled from a management station, in case of emergency. In Figure 2, internal communication between the controllers in the car can aid the driver in parking or reversing. Also in Figure 2, remotely located dealers can be consulted via the Internet to locate possible faults in the car, or for remote licensing.

The challenge does not lie on the applications themselves (in fact, some car manufacturers have implemented similar applications), but in creating a framework that supports the modeling, implementation, deployment and management of a variety of applications. This framework will be the middle tier between CAN network and internal/external applications, thus referred to as middleware (Figure 3). The main advantage of middleware approach over custom designed applications is that the former can free software designers from developing custom communication layers between processes. In addition, a middleware should provide a generic and structured approach to:

- Intranet/Internet connectivity
- Fault detection, isolation and recovery
- Satisfy Quality of Service (QoS) requirements, such as fault-tolerance and real-time constraints.
- Enable access to services under different framework/architecture/implementation

The described features of a middleware cannot be implemented without software support. However, due to the stringent conditions of real-time embedded systems, what is known as the embedded software crisis, open distributed frameworks cannot be employed as easily as in enterprise systems. In other words, middleware solutions such as CORBA [18] and Java RMI, originally designed for enterprise systems, cannot be directly integrated into embedded systems due to the following reasons:

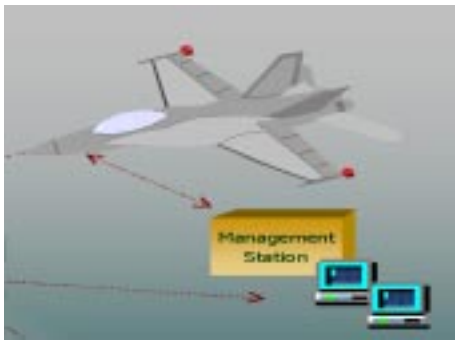

Figure 1 A sample application

- Most of the current middleware approaches assume a point-to-point, connection-oriented, transport protocol, whereas embedded systems are based on group communication.
- As opposed to enterprise systems, embedded systems have reduced resource footprint.
- Real-time embedded systems require QoS specification and enforcement, which are not part of existing middleware solutions. Predictability (real-time constraints) and fault tolerance are examples of QoS requirements.

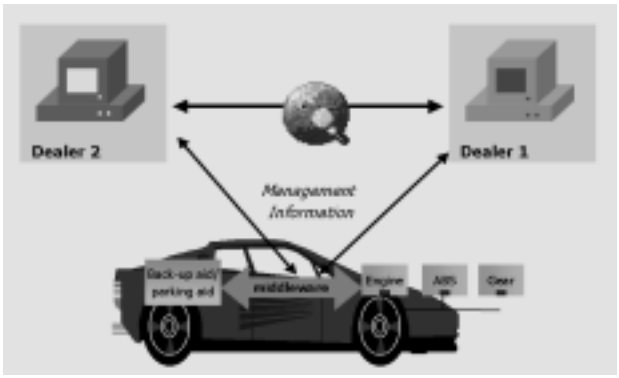

Figure 2 – Example of deploying a generic middleware for CAN

Another drawback of utilizing existing middleware solutions is that they are not interoperable, i.e., an application under a CORBA framework cannot communicate with an application under Java RMI or DCOM framework, unless specific bridges/gateways are deployed.

Existing middleware solutions must therefore be tailored to satisfy the constrained world of embedded systems. In addition, interoperability with existing and future enterprise and embedded systems must also be achieved. In the next section, we survey existing approaches to tailor OMG's CORBA to the requirements of CAN and other real-time control systems.

Tailoring CORBA to CAN Requirements

Distributed object computing has been a promising approach to support the implementation of complex distributed applications. At the heart of contemporary distributed object computing models are *Object Request Brokers* (ORBs), which facilitate communication between local and remote objects. ORBs eliminate many tedious, error-prone, and non-portable aspects of creating and managing distributed applications. One of the widely used ORB models is the Common Object Request Broker Architecture (CORBA) [18], which is standardized by the Object Management Group (OMG).

Since 1991, CORBA has been the *de-facto* middleware used in building distributed enterprise applications. Language, platform and location independence enables the communication between objects implemented in different languages, running on different platforms and on different hosts in a transparent manner. To access a service offered by a remote server, a client only has to make a local method invocation and the ORB will take care of locating the server, transferring parameters and returning results. If client and server are located in different platforms, the ORB will also take care of any required conversions. The interoperability between ORBs from different vendors is possible through the *General Inter-ORB Protocol* (GIOP). A specific example of GIOP is the *Internet Inter-ORB Protocol* (IIOP), that includes a mapping to TCP/IP.

CORBA's language independence is achieved through the *Interface Definition Language* (IDL). An IDL file contains the specification of attributes and operations that can be invoked by a client on a given server. The specification also includes exception values, type definitions, constants and operation signatures. The IDL compiler generates *client stubs* and *server skeletons* based on the specified information. Stubs and skeletons serve as the "glue" connecting remote application processes, and can be generated in various languages, including C, C++, Ada, Smalltalk and Java. Figure 4 illustrates the main components of a CORBA framework.

CORBA was developed for enterprise systems that do not have the resource-constrained environment of embedded systems. In addition, group communication and real-time requirements need to be addressed. Some of the work that addresses these points are described below.

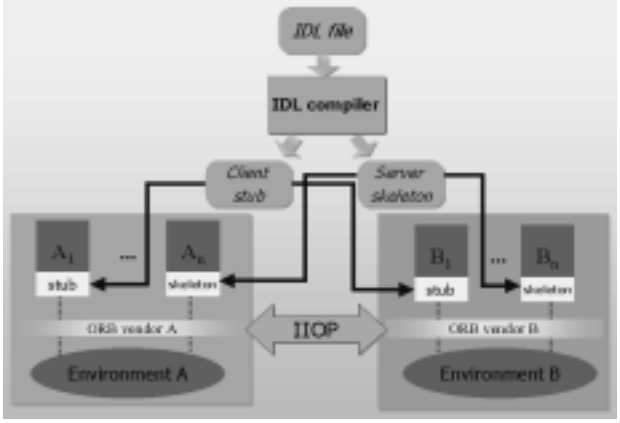

Figure 3 - Stubs, Skeletons, ORB, IDL and IIOP

Group Communication - CORBA is based on the connection-oriented transport model and an object reference denotes only a single CORBA object, thus resulting in a point-to-point communication. Embedded systems such as CAN offer a consistent broadcast mechanism in a straightforward manner via a serial broadcast medium and non-destructive priority-based bus arbitration. It also supports the producer/consumer model of data transmission, which is often referred to as the publisher/subscriber communication model [16]. In this context, a producer of a message is totally unaware of its consumers and simply broadcasts messages over the bus without specifying their destinations. Consider, for example, stopping a set of motors. When issuing a stop command, it is not of interest to address a specific motor, rather it must be ensured that all relevant motors receive the command. Similarly, when reacting to a stop command, it is not of interest which controller has issued that command. On a more abstract level, a sensor object triggered by the progression of time or the occurrence of an event spontaneously generates the respective information and distributes it to the system. It can thus be considered as a producer. There have been different approaches towards including group communication and publisher/subscriber support to CORBA. In [15], a CAN-based transport protocol is designed to support group communication. This protocol makes use of the CAN identifier structure to implement a subject-based addressing scheme, which supports the anonymous publisher/subscriber communication model. [15] also proposed an abstraction scheme called *invocation channel*, which denotes a virtual communication channel connecting a group of communication ports and a group of receivers. A *conjoiner* object, responsible for group management, dynamic channel binding and address translation, is also proposed. An invocation channel is uniquely identified as a channel tag in an IDL program. One of the main drawbacks of this approach is that interoperability between different ORB implementations is lost due to the elimination of the connection-oriented point-to-point communication services.

Another approach, combining broadcast and filtering approaches, is proposed in [13]. The underlying broadcast medium of CAN, along with its non-destructive collision handling scheme that allows for the support of real-time properties, is used as an alternative to multiple point-to-point messages. To make sure that a subscriber does not receive more messages than it has subscribed for, a filtering mechanism is also proposed. This mechanism consists of adding the subject of a message in the address of the message rather than in the message contents. The receiving CAN controllers are configured to selectively receive messages depending on the contents-related address. A masking mechanism is also available to realize "wildcarding" and recognize messages with partially identical addresses as members of a subject group. To solve the binding message problem, i.e., how to find out which address the system has to use when sending a particular message, [13] proposes a dynamic binding

mechanism that binds subjects to addresses at run-time. This mechanism denoted as *Event Channel Broker* [13] supports late binding and local address resolution.

Finally, in [1] an *Object Group Service* (OGS) aimed at facilitating the parallel processing of CORBA operation calls is proposed. This approach consists of defining a *Master* and *Worker* IDL interfaces. The *Master* interface contains one single operation *receive()*, which gets information issued by a server (the worker) registered to the OGS. The *Worker* interface includes the operation *send()* which gets information about the operation to be executed as an argument. An interaction begins when the master invokes operation *send()* on a particular group of workers, thus transmitting the message to be executed. The message is then propagated from the group object to its members, the workers. Finally, the workers will inform the master of the result of their computation by calling its operation *receive()*. In this context, the dispatching of a call to all members of a group is the responsibility of the IDL interface *Group.*

Real-time support. The goal of real-time is achieving predictability in the behavior of some external attributes of a system, such as response time to inputs [4]. Obtaining this predictability (or ``end-to-end predictability'' of the system) requires that all the components in the system behave predictably. Thus, if CORBA is to be used in real-time systems, its behavior must be predictable. Making the CORBA component of a real-time system predictable means making the time of operations invoked on CORBA objects predictable.

There have been different approaches towards achieving predictability in CORBA and other frameworks. In [12], a framework for invoking real-time objects on a CAN bus is proposed. The first byte of the arbitration field in a CAN message is used to define three levels of priority that can be assigned to any message: *hard*, *soft* and *no* real-time constraint. Messages with *hard* real-time constraints have deadline guaranteed by reserving a time slot on the bus. The transmitter enforces the reservation by dynamically increasing the priority of the message according to its laxity relative to the reserved time slot. *Soft* real-time messages are scheduled by the EDF (Early Deadline First) strategy, and non real-time messages are scheduled by assigning fixed priorities. By employing such a scheduling mechanism, timely transmission of hard real-time messages is guaranteed since they always have higher priorities than other messages, and secondly, a hard real-time message gains the highest possible priority at the beginning of its reserved time slot. The optimal scheduling of soft real-time messages is also guaranteed since their priorities are higher than that of non real-time messages, depending directly on the time remaining until its deadline, thus realizing EDF scheduling.

In [13] the use of asynchronous operations, as opposed to the synchronous request-response nature of CORBA and JAVA RMI is proposed. The motivation being the fact that synchronous operations result in blocking the caller until a response is obtained from the receiver, thus not resulting in a ``predictable'' behavior. An asynchronous or event-driven paradigm would not block the sender and would contribute in maintaining an end-to-end predictability.

In [14] an API for real-time Distributed Object Programming is proposed, which enables deadline imposition for arrival of results from an invoked object method, time-triggered actions and non-blocking invocation of object methods.

An extended IDL enabling the specification of timing constraints in a CORBA program, as well as ``fast-track messages'' used for time critical real-time traffic, are proposed in [10]. The ``fast-track messages'' can bypass layers of software and be sent to guarantee predictability.

Finally, we have OMG's *Real-time* CORBA [21], with the goal of synchronizing the ORB operations with those of the underlying Real-time Operating system's environment in order to make operations predictable. Three main aspects of achieving predictability are presented:

- *Processor predictability* - Processors are used in a predictable manner by assigning priorities that are mapped to the priorities of RTOS tasks and threads, thus integrating with non-CORBA parts of the system.

- *Memory predictability* - A ``thread pool'' abstraction is used to control the number of thread and their allocation to different parts of the CORBA system. The amount of memory that

-

▪ each thread takes up can also be controlled and the amount of memory reserved for the queuing of CORBA requests can also be configured.

▪ *Network resource predictability* - Applications can select between and configure the available network protocols, and make choices about the amount of sharing of connections that occurs.

The above features were added to CORBA as extensions through new APIs and semantics. For example, different RTOSs have different priority models. Some have *0* as the highest priority, and others have it as the lowest priority. *Real-Time* CORBA defines a priority-mapping scheme that allow priorities to be passed between parts of a system that are running on different RTOSs. This is possible through a platform-independent priority model, *called Real-Time CORBA Priority*, defined in IDL. The priority is represented as a short integer, ranging from 0 to 32767.

Other features of Real-Time CORBA include ``priority binding'': multiple connections between client and server to handle requests at different priorities.

Quality of Service - Existing work in QoS in embedded system applications has been closely linked to real-time issues. This is not surprising since controlling the real-time behavior of a system is one important dimension of the delivered QoS. In this section, we discuss two of the existing work in the area.

[23] proposes extensions to Distributed Object Computing (DOC) model that supports the control and measurement of, and adaptation to, changing QoS requirements and conditions. The assumption is that as network and end system performance continues to increase, so too does the demand for more control and manageability of their resources through the middleware interface. In addition, next-generation systems present real-time QoS requirements for shared resources and workloads that can vary significantly at run-time, which, in turn, increases the demands on end-to-end system resource management and control. The proposed framework, Quality Objects (QuO), is a distributed object computing framework designed to develop distributed applications that can specify (1) their QoS requirements, (2) the system elements that must be monitored and controlled to measure and provide QoS, and (3) the behavior for adapting to QoS variations that occur at run-time [23]. To achieve these goals, QuO provides middleware-centric policies and mechanisms for developing DOC applications that can perform the following operations *in addition to* their functional behavior:

- Specify level of desired performance or resources, operating modes, and operating regions (all can change dynamically)
- Measure environmental and system conditions using probes in their distributed environment to measure resources, characteristics, and behavior.
- Adapt to changing conditions at all levels.
- Use of *contracts* to encode service requirements, describing the possible states the system might be in, as well as which actions to perform when the state changes.
- Use of *delegates* that project the same interfaces as the stub and the skeleton, but support adaptive behavior upon method call and return.
- Instrumentation probes that can be inserted throughout the remote method invocation path. These probes can be used by the QuO infrastructure to gather performance statistics and validation information.
- Quality Description Language (QDLs) and Code Generators describe and automatically output, respectively, the components of QuO applications.
- QuO Runtime Kernel and GUI Monitor provides a *runtime kernel* that coordinates contract evaluation and provides other runtime QuO services.
- QuO Gateway allows low-level communication mechanisms and special-purpose to be plugged into an application.

In [7] a real-time adaptive QoS management is proposed for the *Kokyu* distributed framework. The proposed approach consists of encapsulating QoS management mechanisms within the Kokyu middleware framework and allowing flexible configuration of policies to shield application developers from error-prone QoS management details, and providing flexibility in meeting diverse end-to-end QoS requirements. The following QoS management mechanisms are included in the proposed framework:

- QoS service configuration
- admission control

- Qos exception propagation
- QoS exception handling
- pacing
- shaping
- classification

Fault Tolerance- Like *Real-Time* CORBA [21] and *Minimum* CORBA [20], the *Fault Tolerant* CORBA (FT-CORBA) [19] has been one of OMG's recent extensions to the CORBA standard. It is a specification that defines a standard set of interfaces, policies, and services that provide robust support for applications requiring high reliability.

Fault tolerance for CORBA objects is achieved via replication, fault detection, and recovery. Replicas of a CORBA object are created and managed as a "logical singleton" composite object. This strategy allows greater flexibility in configuration management of the replicas. The main components of FT-CORBA are:

-**Interoperable Object Group Reference (IOGR)** - FT-CORBA standardizes the format of interoperable object references(IOR) used for the individual replicas. An IOR is a flexible addressing mechanism that identifies a CORBA object uniquely. IOGR is thus a composite of IOR objects.

-**Replication Manager** - This component is responsible for managing replicas.

-**Fault detector and notifier**- *Fault detectors* are CORBA objects responsible for detecting faults via either a *pull-based* or a *push-based* mechanism. *A pull-based* monitoring mechanism periodically polls applications to determine if their objects are "alive." FT-CORBA requires application objects to implement a *PullMonitorable* interface that exports an *is alive o*peration. A *push-based* monitoring mechanism can also be implemented. In this scheme, which is also known as a "heartbeat monitor," applications implement a *PushMonitorable* interface and send periodic heartbeats to the *FaultDetector*.

-Logging and recovery - For the application-controlled consistency style, applications are responsible for their own failure recovery. For the infrastructure-controlled consistency style, however, FT-CORBA defines a logging and recovery mechanism. This mechanism is responsible for intercepting and logging CORBA GIOP messages from client objects to servers.

-Fault Tolerance Domains - Allow applications to scale to arbitrary sizes. A single fault tolerance domain consists of one or more hosts and one or more object groups.

There have been various attempts to integrate fault-tolerance into existing implementations of CORBA in enterprise systems. Some are based on the above specification. There have not been, however, equal attempts in the embedded level. One related approach is found in [11], a fault-tolerant extension to [15]. As mandated by the OMG fault tolerant CORBA draft standard, object replication is the basis
of the proposed method. Two replication strategies are used: passive replication and active replication. Since the straightforward application of these strategies leads to excessive resource demands in embedded real-time systems, a passive replication policy that does not require message logging and object state transfers is proposed.

Reduced Resource Footprint - With the goal of reducing the resource footprint of CORBA, OMG has launched *Minimum* CORBA [20], primarily intended for embedded systems. The approach was to exclude dynamic aspects of standard CORBA, that are not required for embedded systems, e.g., the Dynamic Invocation Interface and the Interface Repository that supports it.

In [15], a resource-conscious customization of CORBA is proposed for CAN-based embedded systems, by reducing the size of data representation and by customizing GIOP with simplified message types and reduced headers.

wireless applications

Previous sections motivated the need for a CORBA-based open framework for communications within embedded systems as well as between embedded and enterprise systems. We now describe two approaches that explore the potential of wireless communication on the level of technical tasks within a vehicle and between a vehicle and its surrounding area.

[24] explores the potential of Bluetooth in automotive applications. The use of wireless transmission is motivated in [24] by its flexible

and reliable features. Flexibility in design and re-design, ease of modification, extension capability and support for decentralization are some of the *flexible* features of wireless communication. Reliability results from the lack of wiring in problematic areas, no need for connectors, resistance to corrosion, robustness against mechanical vibrations and high availability.

The focus in [24] is on control and monitoring tasks inside a vehicle and in defined surroundings of a vehicle. Three levels of communications are proposed:

-Local in-vehicle communication for control and monitoring tasks - Involves communication between control units, sensor units, actuator units, lights and switches. Flexible installation of customer specific devices (plug and play) is also a possible application. Using radio link for wireless communication would increase the flexibility with respect to packaging of electronic units and mechanical design [24].

-***Communication during production*** - Involves the use of wireless communication between the vehicle and the automation system of thw production line [24]. This can enable the exchange of data for testing or diagnosis from the vehicle to the production line.

- ***Communication during service*** - Involves the use of wireless communication between the vehicle and a computer of the service station to exchange status information and service specific information. During service all single units and subsystems of the vehicle are checked, functions are adjusted, and new sets of parameters or new versions of software are downloaded to the vehicle if necessary [24].

The use of Bluetooth is proposed as the wireless connection to the CAN bus. The motivation for using Bluetooth comes from its success in the enterprise-level wireless applications. In addition, there is a good match between the bit rates between CAN and Bluetooth. 10k to 1M bits/s on CAN and 1M bits/s on Bluetooth are the raw bit rates, whereas the useful bit rates for CAN vary from 2K to 581K (at 1Mbits/s), and from 64K to 723K for Bluetooth. There is, however, a transmission and addressing mismatch: CAN is a broadcast network using message identifiers for addressing, whereas Bluetooth is based on point-to-point communication, using source and destination addresses. A gateway that translates between both addressing and transmission schemes will be required. In addition, Bluetooth's support for predictability, fault-tolerance and QoS still need to be evaluated.

In [3] remote connectivity to automotive communication networks is proposed based on BellSouth's Intelligent Wireless Network. A prototype end-to-end Telematic solution was developed for test and measurement and fleet management applications, resulting in applications such as vehicle polling (to locate a vehicle's geographic location), unauthorized movement detector, automatic mileage monitor, route navigation, text messaging (capability for sending in-vehicle hardware configuration commands, remote diagnostics, remote vehicle functions (door unlock, horn activation and light flash) and vehicle performance/monitoring. As in [24], issues such as real-time, QoS and fault-tolerance were not considered.

In summary, there is a huge potential for wireless communication to CAN applications. However, as with CORBA, requirements such as predictability, QoS and fault-tolerance need to be further investigated. In addition, the cost of wireless communications and the need for a standardized *open* solution also need to be explored.

SUMMARY AND RESEARCH DIRECTIONS

This paper motivated the need for an open framework for CAN applications. We investigated ongoing research in integrating CAN and CORBA as well as the use of wireless technologies to build CAN applications.
Due to the stringent nature of CAN, predictability, fault-tolerance, QoS and reduced resource footprint are some of the requirements to be satisfied by any proposed solution. Many of these have been addressed by existing work. Some focus on CAN systems in specific, whereas others address embedded systems in general. The research on wireless communications, on the other hand, has not addressed these requirements as of yet.

An important point that has not been addressed is the need for interoperability between CAN and embedded systems in general and existing

applications running under varying frameworks, such as Java RMI, DCOM [22], etc.. Despite the popularity of CORBA, it still lacks interoperability with other distributed frameworks.

An alternative approach that is becoming increasingly popular is to explore the use of XML [5, 8, 17] to achieve interoperability between applications in heterogeneous systems. The use of XML, however, introduces the same questions as CORBA and other technologies: how to achieve predictability, fault tolerance and other requirements of real-time control systems?

We thus propose the following approach:

- ✓ Examine the *Real-Time, Minimum* and *Fault Tolerant* CORBA - This will allow us to determine the feasibility of using these standards in our study. It will also aid in determining what additional extensions should be implemented to address the specific case of CAN .
- ✓ Support for distributed network management - There has not been much work in supporting remote and distributed management of CAN systems. The following are possible directions:
  - ✓ Investigate the use of wireless middleware for remote monitoring and diagnosis.
  - ✓ Specify the operations and attributes required to manage generic CAN controllers/sensors/actuators.
  - ✓ Specify the types of analysis required to efficiently avoid failures.
  - ✓ Verify the need and possible overhead (if any) of a distributed management framework as opposed to the current centralized version [2,6].
  - ✓ How and where can monitored information be stored or retrieved?
- ✓ Investigate the use of embedded browsers along with WML and HTTP in remote access to the Internet or remote network management.
- ✓ Compare CORBA and XML based approaches in terms of performance, resource consumption, specification and enforcement of QoS requirements (including fault-tolerance and predictability)

REFERENCES

[1] Axel M. A., "*Design and Implementation of a CORBA-based Object Group Service Supporting Different Data Dispatching Strategies*", http://citeseer.nj.nec.com/335099.html

[2] CAN in Automation (CiA), *"CAN Application Layer for Industrial Applications",* DS 201-201 Version 1.1, 1996

[3] Courtright G., Zachos M., Tsui L.,*"A Case Study in Remote Connectivity to Automotive Communication Networks",* Society of Automotive Engineers, 2001, pp 57 - 61

[4] Currey J., *"Real-time CORBA Theory and Practice*: *A Standards-based Approach of Distributed Real-time Systems"*, Embedded Systems Conference – San Jose, 2000

[5] DevelopMentor, IBM, Lotus Development Corporation, Microsoft, UserLand Software, "*SOAP: Simple Object Access Protocol*", http://www.w3.org/TR/SOAP

[6] Etschberger I. K., *"CAN-based Higher Layer protocols and profiles"*, http://www.ixxat.de/english/knowhow/artikel/hlp.shtml

[7] Gill C. D., Levine D., Schmidt D. C., *"Towards Real-time Adaptive QoS Management in Middleware for Embedded Computing Systems"*, Fourth Annual Workshop on High Performance Embedded Computing, MIT Lincoln Laboratory, 2000, http://www.ll.mit.edu/HPEC/

[8] Gu X., Nahrtedt K., Yuan W. Wichadakul D. Xu D., "*An XML-based Quality of Service Enabling Language for the Web*", Journal of Visual Language and Computing, special issue on multimedia languages for the Web. Dec. 2001

[9] Hong S., *"Coping with Embedded Software Crisis using Real-time Operating Systems and Embedded Middleware",* Invited for presentation at IEEE Asian Pacific ASIC (AP-ASIC) Conference, Korea, 2000

[10] Jeon G., Kim T. H., Hong S., *"Seamless Integration of Real-time Communications into CAN-CORBA with Extended IDL and Fast-Track Messages",* Proceedings of IFAC Workshop on Distributed Computer Control Systems (DCCS), Australia , 2000

[11] Jeon G., Kim T. H., Hong S., Kim S., *"A Fault Tolerance Extension to the Embedded CORBA for the CAN Bus Systems"*, Lecture Notes in Computer Sciences, 2000

[12] Kaiser J., Livani M., *"Invocation of real-time objects in a CAN bus-system",* IEEE International Symposium on Object-oriented Real-time Distributed computing, 1998, http://citeseer.nj.nec.com/kaiser98invocation.html

[13] Kaiser J., Mock M., *"Implementing the Real-time Publisher/Subscriber Model on the Controller Area Network (CAN)"*, Proceedings of the 2nd Int. Symp. On Object-Oriented Real-time distributed Computing (ISORC99), France, 1999, http://www.infomatik.uni-ulm.de/rs/core/isorc99.ps

[14] Kim K. H., *"APIs for Real-time Distributed Object Programming"*, IEEE Computer, pp 72-80, 2000

[15] Kim K., Jeon G., Hong S., Kim S., Kim T., *"Resource-conscious Customization of CORBA for CAN-based Distributed Embedded Systems*", In the Proceedings of 2000 IEEE International Symposium on Object-Oriented Real-time Distributed Computing , Newport Beach, pp 34-41, 2000

[16] Kim H., Jeon G., Hong S., Kim T. H., Kim S., *"Integrating Subscription-based and Connection-oriented Communications into the Embedded CORBA for CAN Bus"*, Proceedings of 2000 IEEE Real-time Technology and Applications Symposium Washington DC, 2000

[17] Nielsen M. K., Jorgensen A. B., *"XOIP – XML Object Interface Protocol",* Center for Object Technology, COT/3-34, 2000

[18] Object Management Group (OMG), *"The Common Object Request Broker: Architecture and Specification Revision 2.4"*, OMG Technical Document formal/00-11-07, 2000

[19] Object Management Group (OMG), *"Fault Tolerant CORBA Specification"*, OMG Document orbos/99-12-08 edition, 1999

[20] Object Management Group (OMG), *"Minimum CORBA"*, OMG Document formal/00-10-59, 2000

[21] Object Management Group (OMG), *"Real-time CORBA",* OMG Document formal/00-10-60 edition, 2000

[22] Platt D.S., *"Understanding COM+"*, Microsoft Press, Redmond, Wash., 1999

[23] Vanegas R. *et al.*, *"QuO's Runtime Support for Quality of Service in Distributed Objects"*, Proceedings of IFIP International Conference in Distributed System Platforms and Open Distributed Processing, Springer-Verlag, New York, pp. 207-223, 1998

[24] Wunderlich H., Schwab M., "*The Potential of Bluetooth in Automotive Applications",* Embedded Systems Conference – San Jose, 2000