# Unified Communication in Heterogeneous Automotive Control Systems

Michael Dinkel, Daniel Fengler
BMW Group Forschung und Technik
München, Germany
+49-89-382-21623, Michael.Dinkel@bmw.de

***Abstract*— Current in-vehicle networks are featured by multiple different communication systems. The different automotive network technologies are highly optimized for their special purpose. Since their application domains have very differing requirements, communication across network boundaries is not straight forward and especially upgrading new functionality into already existing vehicle systems is very difficult. This paper presents the architecture and implementation of a communication system that enables the use of very different network technologies over a simple and common API. Furthermore, the implemented publish/subscribe middleware ensures loose component coupling and upgradeability of applications to future networking technologies.**

***Index Terms*— Middleware, Messaging, Publish/subscribe, Network abstraction, Heterogeneous systems.**

## I. Introduction

CURRENT automotive computing systems have reached a nearly overwhelming complexity due to several reasons. Of course the number of applications in all domains of the vehicle has grown, but also the nature of the applications is beginning to change. While the applications of the last decade have been isolated for each domain like comfort or entertainment, the new generation of applications starts to make use of the great number of sensors and actuators located throughout the vehicle. Due to cost pressure the network technologies employed in specific domains are highly optimized for the specific needs of the respective application domain. This means a MOST[1] network is very well suited for transmitting audio signals but much to complex for very simple devices like a door module where commonly CAN[2] and LIN[3] connections are used. Historically this was sufficient since all requirements of the applications were met, the communication partners where located at the same network segment. Networked and local communication has been kept as simple as possible to prevent high costs per piece. With the introduction of central diagnosis applications the first software required inter domain communication met the vehicles. The result of these new demands paired with the well known cost pressure in automobile development has been the introduction of gateways that connect the different network types. A central gateway provides access to the vehicle infrastructure and translates the commands of the diagnosis protocol to the different networks, which results in an application specific solution that leaves as much as possible of the existing system untouched. In the course of this the gateways have been extended by additional application specific data that was needed for different functionality. Figure 1 depicts an overview of the current network infrastructure of an upper class sedan.

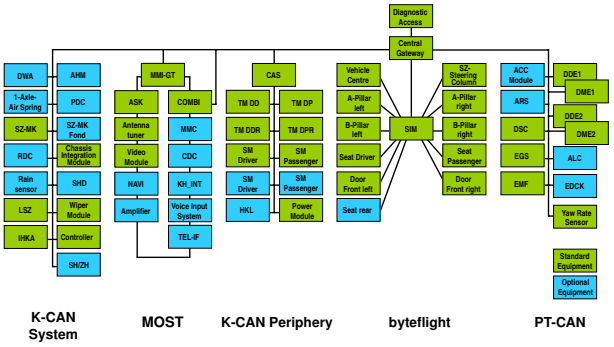


Fig. 1. Onboard network of a BMW 7-series (E65)

Here again the different networks, topologies and the gateways become visible. Even though the resulting complexity has been increased drastically, application specific gateways possess a positive feature for the vehicle safety: they keep domains of different criticality to some extend separated from each other.

[1] Media Oriented Systems Transport
[2] Controller Area Network
[3] Local Interconnect Network

Nevertheless, the area of upcoming automotive applications will even more require inter domain communication and upgrades to existing systems alone is no longer sufficient.

In this paper we describe our approach towards a unified communication infrastructure for vehicle systems. In section two we point out the different types of heterogeneity and argue that the publish/subscribe paradigm represents an adequate solution for our current problems. The architecture of a messaging middleware that enables the transparent use of different network technologies under a common API is presented in section three where we point out the main advantages of our communication infrastructure. Section four provides benchmark results in comparison to other messaging middlewares with similar capabilities. Related work is summarized in section five and section six concludes and points out future work.

## II. Mastering heterogeneity

The typical communication systems used in the automotive domain provide quite simple broadcasting communication. On the CAN bus for example, every node can read the messages on the network and addressing is done via filtering of message identifiers. This type of communication seems to be appropriate for the real-time requirements of control applications but is usually limited to one single network segment. The FlexRay technology uses a similar addressing scheme but is based on time-triggered communication as necessary for highly safety critical applications. In contrast to byteflight, FlexRay, and CAN, networks based on MOST technology are featured by a ring topology.

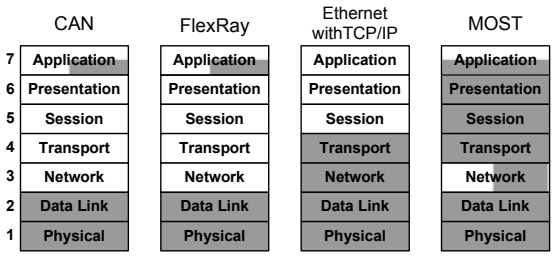


Fig. 2. Comparison of network layers

Figure 2 depicts a comparison of the network layers of different communication systems. In contrast to TCP/IP the automotive bus systems are much more than a protocol or protocol family. All of them include specifications of the physical layer as well. In figure 2 one can see that the CAN and FlexRay protocols define the ISO OSI layers one and two, which makes it obvious that these bus systems are designed for a single network segment. They provide a communication service that enables sending messages to the network but not more. In contrast to that, the MOST specification also defines the upper layers of the ISO OSI model. Even the Application layer has been defined and provides for easy integration by the definition of standard interfaces for typical infotainment components like tuners, DVD-players, navigation or amplifiers. New standards and technologies especially in the area of wireless communication are going to meet the automotive market and will have to be integrated somehow.

### A. A glimpse to the future

While today functionality in vehicles is always sold as a combination of hardware and software in the future it will no longer be possible to realize applications on a single ECU. Nor will there be one ECU for a single functionality. The automotive industry is on the way towards platforms that may execute different types of functionality and that support life cycle management. The introduction of component-based software will lead us to a new definition of the term application. *Applications* are defined as a set of communicating components that provide coherent, user-perceivable functionality [1]. Software components can be installed, removed or updated separately or as a part of an application. This notion of applications stresses one important thing, applications will be composed of different parts that may be distributed over the complete vehicle. Hence an easy to use and unified communication abstraction is needed to enable straight forward application development.

While there are research projects that try to solve this problem by just applying a single bus system to all ECUs in the vehicle, we are convinced that the current cost pressure will remain or even increase in the future. So there will always be different network technologies in a vehicle since otherwise one would have to use the most powerful and often most costly technology even for simple and cheap controllers. Along with the need for different network technologies we foresee the need for vehicle systems to support different architectures and different programming languages. So from this section we conclude that heterogeneity will persist in the future and a unified communication infrastructure has to be applicable to and has to abstract from the heterogeneity of vehicle systems to ease application development.

The life-cycle problematic which means that parts of the system have very different periods of life will

not vanish in the future. The expected increase in new applications in vehicle systems makes it even more important to provide a solution that allows updating, installing and removing software components by loose coupling between them.

### *B. A communication abstraction*

The search for an appropriate abstraction that makes it possible to use different network technologies and ECU architectures as well as different programming languages resulted in the publish/subscribe paradigm. Publish/subscribe is a messaging paradigm that enables many-to-many communication in a completely asynchronous way. All communication partners take one of the two roles `publisher`, sender of messages, or `subscriber` receiver of messages. Sending and receiving in a publish/subscribe system is both featured by asynchronous behavior, which contrasts other messaging technologies like message queuing or message passing. The addressing in publish/subscribe systems can be done in different ways like content-, type- or topic based [2]. Common to all these addressing schemes is that addressing communication partners is done indirectly which avoids the need for publishers to know the address and location of their subscribers. Subscribers declare their interest in messages assigned to a certain area of the address space and the communication system takes care that messages are delivered to the registered subscribers. Publishers are able to send messages at any time.

Publish/subscribe has been identified as an appropriate communication paradigm that can provide a lot of features which fit the requirements of the automotive domain quite well [3]. A messaging middleware provides for the needed abstraction of the technology at hand. For the automotive domain a static addressing scheme has been chosen that divides the address space into a hierarchy of channels that may be created at runtime. The following gives an overview of the reachable features:

- **Loose coupling**
  In order to be able to update and exchange parts of the system as independently from the others as possible a communication middleware has to provide loose component coupling. Publish/subscribe provides this loose coupling since publishers and subscribers do not need to know each other's address or reference[2].
- **Language and platform independence**
  In principle messaging mechanisms are independent from a specific programming language. The multi language capability is realized by a platform independent message transfer format that be interpreted on every involved platform. The message format and its concrete semantics is specific to each application domain and can be considered an application layer protocol (OSI layer 7).
- **Network-independence**
  A publish-subscribe messaging layer can be realized on top of nearly any existing communication protocol and is therefore very well suited for bridging different technology and protocol domains.
- **Communication multiplicity**
  The publish/subscribe paradigm allows having multiple publishers and subscribers for the same channel. Also joining and leaving a channel at runtime is possible. With the use of publish/subscribe messaging applications become extensible without changes at the existing components. The data is accessible via the channel indirection and the M-to-N semantics of publish/subscribe allows adding new publishers or subscribers for a channel very easily.
- **Location transparency**
  With publish/subscribe messaging the senders (publishers) and receivers (subscribers) of messages do not need to know each other in terms of addresses or object references. The indirection of channels or topics offers location transparency.
- **Unique service interface**
  The publish/subscribe paradigm offers great potential for a unified abstraction for communication between different technology domains. By the means of QoS it becomes possible to really use the same way of communicating in very different situations like real-time, continuous data streaming or just sending high priority or reliable messages. This functionality can be used in the same way in all different domains.

## III. Prototype architecture

The following have been the main aspects of heterogeneity that should be addressed by our messaging middleware: 1) bridging *different platforms*, processor architectures and programming languages, 2) abstracting from *different communication networks*, topology, underlying protocols and schemes of addressing, 3) being suitable for *different application requirements*, and 4) providing a *uniform communi-*

*cation interface* that makes no difference in remote or local use.

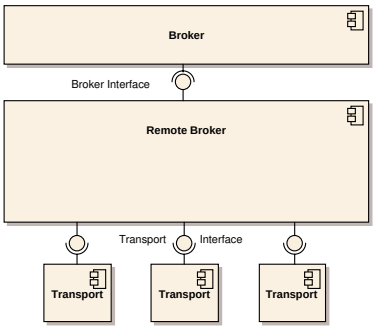


Fig. 3. BrokerSystem: layered architecture

Figure 3 gives an overview of the layered architecture of our publish/subscribe middleware. The interface for applications is exactly the same regardless of remote or local communication. We aimed at achieving good performance also for local communication on the same machine. Hence, the broker is separated into two layers so that we have the possibility to use it locally for loosely coupled and asynchronous communication on the same node without the help of the RemoteBroker and transport layers. The functionality of transcoding messages into the common transport format is located in the remote broker layer and can be considered a plug-in. Local communication can hence be performed by very efficient local method calls whereas more complex mechanisms can be used for remote communication.

### A. Broker Layer

The broker layer provides the application interface and manages the address space of the local host. The address space is hierarchically structured and is synchronized with remote hosts on every change like adding or removing a channel. Subscribers can subscribe to leaves and branches of the tree formed by hierarchical channel names. For a subscription to a channel that contains sub-channels all messages are delivered including the ones published to the subchannels. A priority driven queue is used as common resource for both sending and delivering messages. The priority queue is separated into two sections, one for real-time messages and the other one for non-real-time messages. Real-time messages are always delivered before all other messages.A thread pool provides worker threads for delivering messages from the queue. If there are remote subscribers for a message the message is delivered to the RemoteBroker layer.

### B. RemoteBroker Layer

The RemoteBroker layer has two main tasks: the remote data storage and message encoding.

*a) Remote Data Storage:* The remote data storage keeps per channel information about the remote publishers and subscribers. For the sake of system robustness all remote broker instances in the distributed systems keep a global view on the systems information. For each channel, we store the address and transport adapter of subscribers.

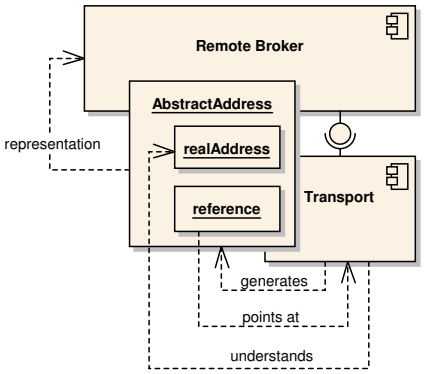


Fig. 4. Structure of abstract addresses

Since there may be multiple different transport adapters used for communication we have to store not only the address of a remote subscriber or publisher but also the transport adapter that is used to communicate with it. Addresses are stored in the special type AbstractAddress, see Figure 4, that encapsulates the network-specific address information and mediates between the remote broker and the network specific transport adapters. The remote broker can handle any form of encapsulated concrete address by the use of abstract addresses. Thus our architecture is extensible by new types of transport adapters without changing the remote broker layer.

*b) Message Encoding:* The abstraction from the systems heterogeneity has been realized by defining a generic message format in a standardized way. Due to the use of ASN.1[4] and its basic encoding rules (BER) application messages are transmitted in a *neutral transport representation* and thus are understandable on different types of network nodes. In this way there may be optimized message representations that are highly suitable for the specific environment on each node but communication is still possible due to the common understanding and representation of messages on the wire.

[4] Abstract Syntax Notation One

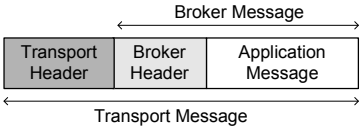


Fig. 5. Message structure

Figure 5 depicts the structure of application messages, the header added to each message at the broker layer, and the transport specific message header. Our prototype is flexible enough to cope with any kind of encoding for the application message. The broker header adds header fields like the channel name, unique message identifier, priority, time-stamp, real-time flag, and version numbers for the message and the version of the broker system.

### C. Transport Layer

The utilization of different network technologies becomes possible by abstracting the message broker from the underlying transport service, see figure 3. While the remote broker component manages the generic information about the address space with its publishers and subscribers, each transport adapter has specific knowledge of the underlying protocols and networks. Discovery of other nodes on the network also belongs to the tasks of the transport adapters since the procedure may vary for each different network. For our prototype we developed transport adapters for TCP/IP and for Bluetooth using the L2CAP[5] protocol to show the protocol independence. The TCP/IP adapter uses a discovery based on IP-multicasts and is able to make use of the UDP protocol for messages with the real-time flag.

## IV. Benchmarks

As a proof-of-concept the broker architecture has been implemented in a modular way as bundles for a Java/OSGi platform. We compared our implementation called the BrokerSystem with several open source JMS-messaging[6] implementations like OpenJMS, ActiveMQ, MantaRay, and UberMQ for the remote use case over a TCP-IP transport adapter. For the benchmarks we chose a setup of two nodes to exchange messages and measured the round-trip time of a message from one node to the other. The two nodes have been standard PC hardware (Pentium4 2,6GHz, 1GB RAM) connected via a 100MBit Ethernet running TCP/IP. Since our prototype is implemented in Java we chose to run the benchmarks on Windows 2000 and on SuSE Linux 9.1. The exchanged messages should give a broad view of possible applications. So we decided to use two different message sizes: 1) a payload of 8 Bytes ASCII data and 2) a payload of 1 Megabyte binary data. Our assessment included both remote comparison with JMS messaging solutions and local comparison with typical Java communication means.

[5] Logical Link Control and Adaptation Protocol
[6] Java Message Service

### A. Benchmark results

Figure 6 presents the results of the remote comparison with other open source JMS implementations. Persistence features have been deactivated on all competitors where possible. One can see that in the remote case our modular architecture does not prevent good results. Sending messages via UDP can perform even better for small messages while large messages can not be send via UDP. We can see that only the MantaRay implementation performs better in both use cases than the TCP version of the BrokerSystem. For local communication the comparison with the JMS implementations has also been done but our prototype outperformed all other implementations since it avoids marshalling for local communications.

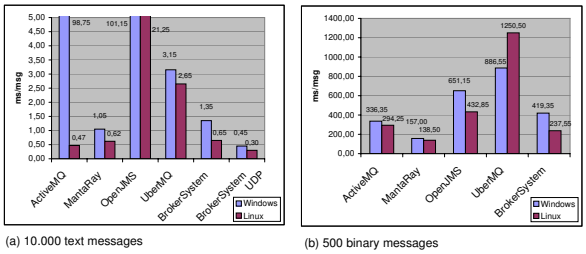


Fig. 6. Benchmark results for remote communication

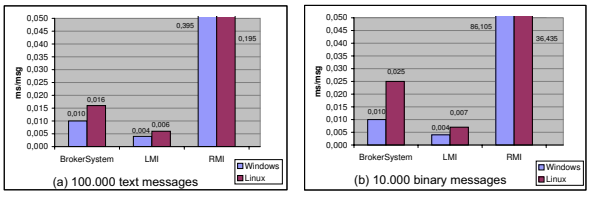


Fig. 7. Benchmark results for local communication

Since the JMS-implementations are mostly meant for remote usage we also compared the BrokerSystem with Java/RMI and direct local method invocations (LMI). In contrast to RMI our prototype does not need to encode the messages into transfer representation for local communication. The BrokerSystem performs only 3.5 times slower than local method

calls in Java which sums up to 0.018 milliseconds in the test environment. As depicted in figure 7 RMI is a lot slower for both text and binary messages. Remotely RMI performs a little better than our prototype.

## V. Related Work

Concerning the in-vehicle domain other researchers have also identified that current low-level protocols are inadequate for upcoming applications and their communication needs. A few studies have focused on the CAN bus as a very popular automotive bus system. For Example the work described in [4] tries to port the Jini technology to the CAN bus, and discovered that the very design of Jini itself made assumptions about the use of TCP and UDP. This did not prevent the porting of Jini to CAN, but imposed a significant source of inefficiency and messiness in substituting the wire protocols. In [5] the IP-protocol has been ported to the CAN bus and enables IP-based communication between CAN nodes and the Internet. The work in [3] realized an efficient implementation for the publish/subscribe model on the CAN. The remaining issue with these works is that they focused solely on the CAN bus. The interoperation between different automotive bus systems has been out of focus. One of the first ideas when thinking about a unified information exchange would be to use the Internet Protocol (IP) for all domains. IP has been the basis for the work in [6]. But since applications of the in-vehicle domain have to cope with very low-cost parts IP is currently not an option for control applications inside the vehicle. Today it is unrealistic to exchange the proven protocols in all different technology domains. So what can be done is to add a further abstraction on top of the existing protocols as done for example in [5] with IP as additional layer on top of CAN. In such an approach we have to consider bandwidth and packet sizes especially in the real-time domain. An IP packet of 1500 Bytes for example has to be fragmented into 215 CAN packets of 8 bytes size. As stated in [5] one byte has been used for sequence numbering. So from an automotive point of view a communication abstraction has to be lightweight and extremely scalable to fit the wide range of different embedded communication systems combined in a single vehicle.

## VI. Conclusion and Future Work

Future applications will increasingly have the requirement to access sensors or actuators that are not located on their home network. Hence a communication mechanism is needed that is able to bridge between different network technologies transparently. The publish/subscribe paradigm provides an abstraction from different network technologies. Message based communication is well suited for abstracting typical automotive networks since it supports their usual form of communication.

With our prototype implementation we have shown that we can overcome heterogeneity of different types. The layered architecture enables the transparent use of different networks while the ASN.1 based transport encoding ensures that messages can be interpreted on different nodes running different operating systems and clients written in different programming languages. Bridging different network technologies has been shown by connecting a TCP network with a Bluetooth domain. The benchmark results are satisfactory since this was a proof of concept implementation and there has been no optimization yet.

For the future we are going to implement transport adapters for automotive networks like CAN and MOST. We are going to include our messaging middleware into embedded devices and address the subject of network redundancy and better QoS modeling and real-time capabilities.

## References


[1] Michael Dinkel and Uwe Baumgarten, "Modeling nonfunctional requirements: a basis for dynamic systems management," in *SEAS '05: Proceedings of the Second International Workshop on Software Engineering for Automotive Systems*, New York, NY, USA, 2005, pp. 1–8, ACM Press.

[2] Patrick Th. Eugster, Pascal A. Felber, Rachid Guerraoui, and Anne-Marie Kermarrec, "The many faces of publish/subscribe," in *ACM Computing Surveys (CSUR)*, vol. 35, pp. 114 – 131. ACM Press, New York, NY, USA, 2003.

[3] Joerg Kaiser and Michael Mock, "Implementing the real-time publisher/subscriber model on the controller area network (CAN)," in *Proceedings of the 2nd Int. Symp. on Object-oriented Real-time distributed Computing (ISORC99)*, Saint-Malo, France, May 1999, pp. 172–181.

[4] Meredith Beveridge, "Jini on the control area network (can): a case study in portability failure," M.S. thesis, Carnegie Mellon University, 3 2001.

[5] Michael Ditze, Reinhard Bernhardi, Guido Kmper, and Peter Altenbernd, "Porting the internet protocol to the controller area network," in *2nd International Workshop on Real-Time LANs in the Internet Age (RTLIA2003)*, 2003.

[6] Gerardo Pardo-Castellote, Stefaan Sonck Thiebaut, Mark Hamilton, and Henry Choi, "Real-time publish-subscribe protocol for ip-based real-time communication," Tech. Rep., Real-Time Innovations, Inc., September 2001.