# A Vehicular Software Architecture Enabling Dynamic Alterability of Services Sets

Volker Feil[1], Ulrich Gemkow[1], and Matthias Stümpfle[2]

[1] University of Stuttgart, Institute of Communication Networks and Computer Engineering, Pfaffenwaldring 47, 70569 Stuttgart, Germany
{feil,gemkow}@ind.uni-stuttgart.de

[2] DaimlerChrysler AG, Research and Technology 3, Vehicle IT-Architecture, HPC T728, 70546 Stuttgart, Germany
matthias.stuempfle@daimlerchrysler.com

**Abstract.** In this paper we present the software architecture DANA based on components for distributed systems in vehicles. Components use and provide services. Since they are binary codes that are deployable and configurable at system's runtime, it is possible to alter the set of services during the life cycle of a vehicle. Due to the flexibility of this architecture emerging new application services for drivers and passengers can be added at any time. In order to enable the offering of such application services there is a need for services collaboration inside the distributed system which meets QoS requirements. Because the existence of many small control units in future vehicles is expected, an architecture must regard also the services provided by them. We show that the presented architecture is suitable for such an automotive environment.

## 1 Introduction

Recent evolution of information and communications technology increasingly change our everyday life. Also, inside vehicles new services for drivers and passengers arise. For instance, typical driver supporting services (e.g. providing traffic information or navigation) become more and more familiar. Furthermore, services that are well established in the home and office area (e.g. video, or Internet based services like WWW [4]) become available in vehicles.

Until now, there are mainly proprietary software solutions in the telematics area. However, because of the spreading of software solutions the interaction of service functions becomes more and more interesting even with respect to economical constraints.

Such tendencies are visible in the success of universal technologies for distributed computing with its main representatives Java RMI [3] and CORBA [17]. New developments like UPnP [16] or Jini [20] make distributed computing even more powerful. It is forseeable that a similar evolution will take place in the automotive environment, especially as the vehicle itself will become a node of a global network.

The approach presented in this paper regards software architectures in future

vehicles. In the next section special automotive requirements on a vehicular software architecture are listed. In section 3 the basic elements of the presented architecture are described. The management tasks of this architecture and how they are achieved are shown in section 4. Finally, a classification of the architecture is presented in section 5. This classification implies a comparison to universal technologies like UPnP and Jini, and the discussion of implementation aspects.

## 2 Requirements on a Vehicular Software Architecture

The first step of designing the architecture is to recognize the special automotive requirements.

### 2.1 Requirements – User's View

**Manifold Set of Services.** We define a set of services as a combination of user oriented application services. Within vehicles a manifold set of services will be offered. Table 1 shows a selection of application services that will be available for drivers and passengers in future.

**Table 1.** Application services in future vehicles

| Mobile Office Services | Entertainment Services | Information Services | Transaction Services | Vehicular Services |
|---|---|---|---|---|
| e.g. telephone, fax, notebook access, Internet services (WWW [4], E-Mail) | e.g. digital broadcast and TV [8], audio / video sources (DVD), games | e.g. traffic information, countryside/ city information, parking information | e.g. bookings (e.g. hotel reservation), banking | e.g. navigation, car diagnosis, emergency call |

The services require data exchange with different qualities: Speech data with delays of more than 200 ms or jitter effects are troubling [13]. Audio and video transmissions require large data rates. Data of Internet applications like WWW and E-Mail must be exchanged without losses.

**Alterability of the Services Set.** It is obvious that nobody can give accurate predictions regarding future needed services. Because a vehicle's life cycle is much larger than an usual life cycle of an application service, it is necessary to be able to update the set of services during a vehicle's life time.

**Easy Handling and Easy Maintenance.** It can not be expected that vehicle holders have software knowledge. Furthermore, it is unlikely that they would accept additional stops in a garage just for software updates. Therefore, easy handling and maintenance is required to achieve a dynamic changeable services set in a vehicle.

**Integration in a Global Network Infrastructure.** Most services shown in Table 1 are communication oriented and this communication is not limited to within the vehicle. The vehicle itself is no longer isolated, but becomes a part of a globally distributed system. Access to services provided anywhere must be possible.

### 2.2 Requirements – Manufacturer's View

**Consideration of a Heterogeneous Environment.** A distributed system with control units (devices) comprising of a processor and local memory is expected to be part of future vehicles. Even today a vehicle of the comfort class contains between 40 and 80 control units [11]. These control units are loosely coupled by communication systems. Currently, there are still rather seperate systems in a vehicle that can be roughly divided in an engine, a comfort, and a telematics area. A typical bus for the engine area is CAN (Controller Area Network) [18], and a typical communication system for the telematics area is the optical D2B [2]. D2B allows to exchange audio streams. There will be even more powerful communication systems like IEEE 1394 [14] or MOST (Media Oriented Systems Transport) [5] in the telematics area that will also provide handling of video streams. It is expected that there will be a total distributed system with very heterogenous control units and communication systems that differ in their capabilities. A vehicular software architecture has to consider this heterogeneity.

**Support of Small Control Units.** One result of the expected heterogeneity is the existence of a lot of small control units per vehicle. Today, usual vehicular control units do not contain sophisticated networking protocol entities. It is conceivable that lots of them will still not contain an IP stack even in next future.

**Support of Standard Technologies.** It is expected that besides the small control units some units will have the power to support popular, and more universal IP-based technologies, e.g. UPnP and Jini.

**Integration in a Global Network Infrastructure.** It is the manufacturer's interest to integrate the vehicle as a node in a global network. With respect to economical constraints it can be worthwhile to provide vehicle's functionality outside instead of inside the vehicle.

**Consideration of Manufacturer/Supplier Relationships.** One aim is to preserve familiar, proven, and automotive typical structures of manufacturers/supplier relationships also by introducing a new software development process.

## 3 Basic Elements of the Architecture

### 3.1 Service and Component Concept

In this paper software *components* are defined as *binary units of independent production, acquisition, and deployment that interact to form a functioning system* [10]. Interacting means that components use and provide services mutually (see Fig. 1).

Therefore, a component can be regarded as a container for services.

The usage of component software technology is meaningful to automotive manufacturers. First, components are produceable on the producer's own authority and acquirable on the vehicle manufacturer's own authority. Therefore, automotive industry suppliers can be integrated in the software development process in a familiar way. Second, components are deployable under the own control of the deploying entity. Consequently, the dynamic alterability of a services set is achieved by enabling the facility to deploy components in an existing vehicle.

There are already efforts to bring the software component idea into vehicles. For instance, in [6] a Java Beans [19] based approach is presented. In this paper we describe the component concept of DANA *(D*istributed Systems *A*rchitecture for *N*etworks in *A*utomotive Environments). DANA components are binary codes, that are deployable and configurable at system's runtime (see also [9]). In Fig. 1 the component layering model of DANA is shown.

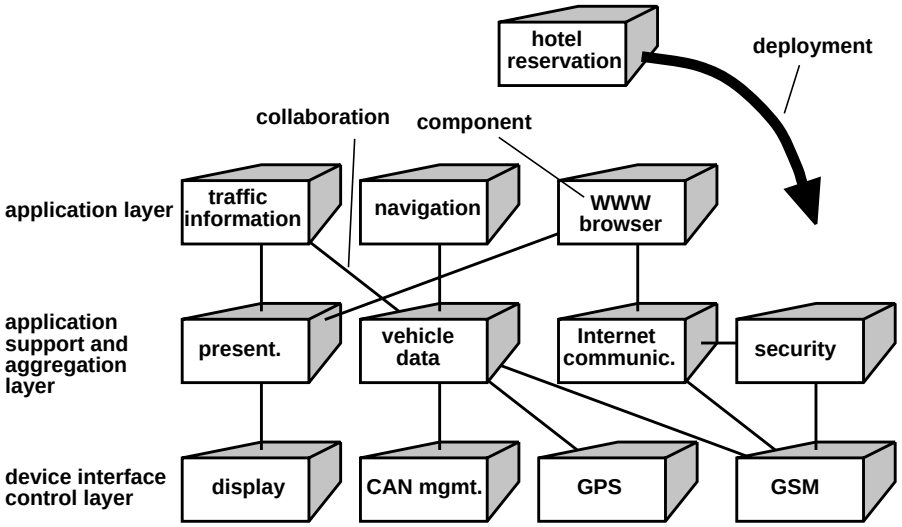


**Fig. 1.** The collaboration of vehicle specific components is shown exemplary. The components are put in a layered structure. Application components use application support and aggregation components, which itself use device interface control components. The latter allow the access to the attached hardware. Furthermore, the example depicts the deployment of a new hotel reservation component.

In order to enable the proper collaboration of the components, there is a need for constraints and rules that are imposed by the architecture. These rules consist of well defined interfaces, and in a management for component deployment, service discovery and enabling service utilization. As the system complexity should be transparent to the user, manual interventions and interaction with the user cannot be allowed for configuration purposes.

There are two ways considered in order to get a management for service discovery and for enabling service utilization without manual intervention. The first way is to apply standard building blocks (e. g. Java RMI, and Jini) in order to create a rapid prototype. The second way is to design a new architecture that respects the special automotive constraints (mainly the existence of small control units). The emphasis of this paper is to present that second way.

### 3.2 DANA Elements

In this section, the structure of a DANA component is described. In Fig. 2 the ingredients of such a component, a worker and a management, are depicted.

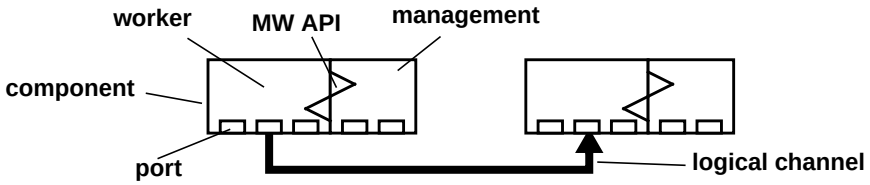


**Fig. 2.** Both parts are connected by the management/worker API (MW API). To allow data exchange among components, a component contains so called ports, at which logical channels can terminate. DANA components exchange data via logical channels.

Components can collaborate, if they are connected via logical channels. Fig. 3 shows a component distribution by means of an Internet scenario. This scenario shows the need for fulfilling a guaranteed Quality of Service (QoS). For instance, the logical channel between the WWW browser component and the Internet communications component (that provides the transport system) has to guarantee loss free data communication.

### 3.3 Service Ontology

A service can only be discovered and used if there exists a common comprehension of its capabilities.

The service that is provided by a DANA component is denoted non-ambiguously by a name (string). Several components representing the same service may exist. These components do not differ by their service name, but by their DANA standardized address. A service comprises also of a bit mask in order to denote its capabilities. A trading service is intended additionally in order to enable more complex queries.

For instance, one component in Fig. 3 provides a GSM service. It has capabilities to provide a signalling interface (E.164 numbering) for establishing and releasing connections, access to the traffic channel of an established connection, and access to a PPP entity. The traffic channel access supports a data rate up to 9,6 kbps, and a speech rate up to 13,2 kbps.

If a component looks for a GSM service explicitly by name, it knows all about the GSM service capabilities. However, if the component just looks for a signalling

interface to establish a telephone connection to an outside device, it can discover the service by analyzing the bitmasks of all available services for a suitable pattern. Therefore, an UMTS [15] service which may be available in future can also be discovered, even if the component does not know that service's name. Furthermore, a component can make use of a trading service. The result of a query to a trading service may depend on the combination of the input data and vehicle specific data (e.g. position data). Thus, the result of a query on the road may be still the GSM service, whereas the result of a query at home may be a cheaper and more powerful DECT [12] service.

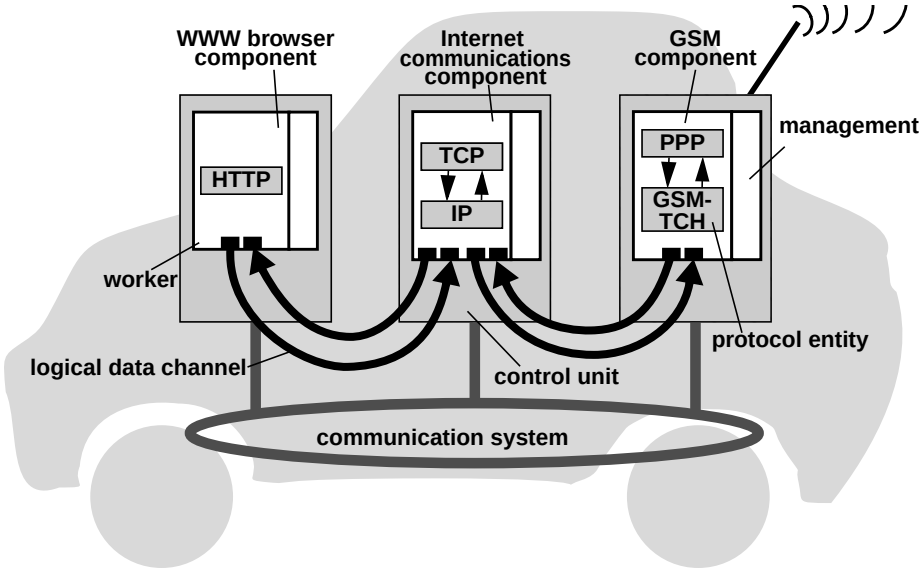


**Fig. 3.** An example for the component deployment is shown by a simplified Internet scenario. In this scenario only one component of the vehicle offers the service of a TCP/IP layer. This Internet communications component can even represent the car with one IP-address in the global Internet by having routing and masquerading functionality. The location of the TCP/IP-layer service users (e.g. the WWW browser component) does not matter as the existence of logical channels provides location transparency. Furthermore, Internet access for notebooks or PDAs is available by this constellation. Access to the external Internet is available through the GSM component that owns the PPP entity (OSI layer 2) and the GSM traffic channel (GSM-TCH, OSI layer 1). This component is assigned to the device interface control layer and enables access to the installed GSM hardware.

## 4 Management Tasks

The collaboration of the components needs to be managed. The tasks of this management are to realize discovery of components and their services, utilization of a discovered service, and deployment of components to a control unit or to the external infrastructure.

Each component contains a management (see also Fig. 2) that is connected via so called logical management channels to so called managers. Managers, which are components themselves, are also interconnected by logical management channels in order to collaborate. Only standardized messages of the DANA management can be exchanged through logical management channels.

### 4.1 Service Discovery

Three types of managers, Local Service Manager (LSM), System Service Manager (SSM), and External Service Manager (XSM), work together to realize service discovery.

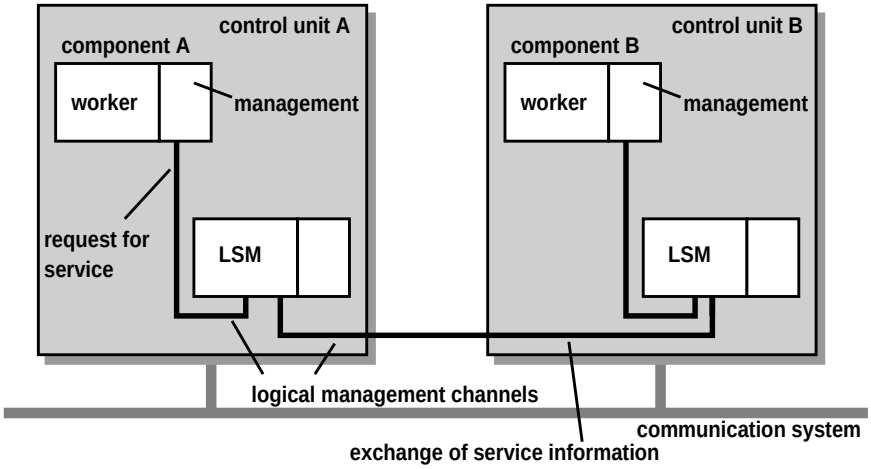


**Fig. 4.** LSMs exchange service information via logical management channels. They are responsible for all components of their unit. Therefore, the management of the components must communicate with them via local management channels.

**Service Announce Mode.** The basic mode of service discovery is based on the collaboration of Local Service Managers (LSMs) that are located in each control unit (see also Fig. 4). A LSM represents the components of its control unit and stores information about these in a local service table. After unit initialization, the LSM works in the announcing state. In this state the LSM broadcasts *service announce* messages. The content of the message consists of the representation of all services inside the unit. By evaluation of the message content each LSM builds its local system service table with all available services that are relevant for this unit. An entry of this table consists of a service name, the service capabilities, and the DANA address.

**Service Polling Mode.** If a System Service Manager (SSM) becomes active, it takes the registration task on. A SSM discovers all existing LSMs by receiving their service announce messages. It informs the LSMs about its existence and asks for terminating

the broadcasting of the service announce messages. A SSM builds its system service table by a polling process. It cyclically asks all LSMs for service information. Therefore, each LSM that is ready for working together with the SSM needs no local system service table, and changes to a polling state.

**Request for Services.** By collaboration of the managers service discovery becomes available. A component can discover the location of another component by calling the service name or some service capabilities. Thus, the service management part of a component must contact its responsible LSM (see also Fig. 4).

**Integration of External Services.** There is a distinction between the vehicular internal management and the overall operating management. The vehicular internal discovery management is based on a broadcasting concept (service announce mode) whereas the vehicular external discovery management additionally rests on a portal concept. This is due to the inefficiency of the broadcasting concept for large networks. If a LSM even by interacting with the SSM does not posses service information for a requested component the manager can interwork with the External Service Manager (XSM) additionally. The XSM clarifies whether a service is available outside the vehicle or not. The external counterpart of the XSM is a global service center (portal) that can return a global address of an externally deployed component.

### 4.2 Enabling Service Utilization

In the simplest case all components can exchange data by permanent data channels that are established during the initialization phase of a service. Such a system is static because no new services can be integrated. To achieve a dynamic system with a modifiable components set, the management must enable the establishment of logical data channels to the new deployed components.

The channel management is controlled by another set of managers: The Local Channel Manager (LCM), Range Channel Manager (RCM), System Channel Manager (SCM), and External Channel Manager (XCM) are responsible for the channel management. On each unit a LCM is deployed. It appears in two roles – the requesting role and the requested role. The requesting LCM accepts an instruction to modify (e.g. establish or release) a channel from a management of a component placed on its unit. By usage of a three-way-handshake protocol the requesting LCM negotiates the channel modification with the requested LCM (see Fig. 5). Within this interaction the requested LCM is responsible for the component of its unit that is affected by the channel modification.

Furthermore, a RCM is involved if the modified channel spans over different control units. The RCM represents the communication system at the QoS negotiation. If a channel modification affects more than one communication system, the System Channel Manager (SCM) representing the total vehicle system must be involved. If a channel modification affects the communication infrastructure outside the vehicle, the External Channel Manager (XCM) must be involved.

After establishing a new data channel, components can use it for data exchange regarding the required QoS. After releasing an existing data channel, the needed

resources are deallocated.

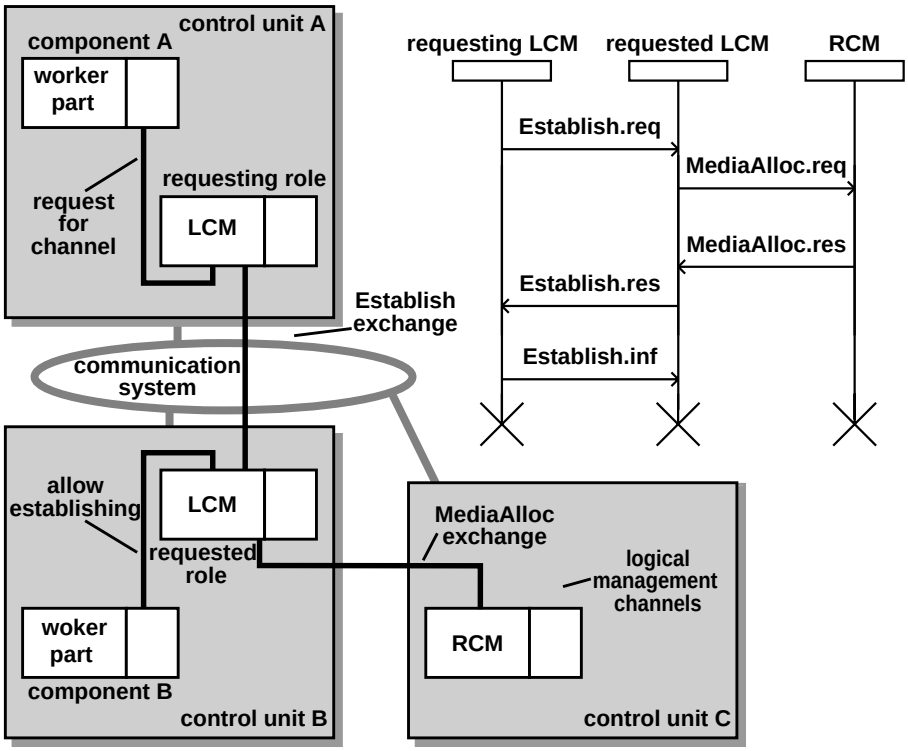


**Fig. 5.** Channel establishing example: After determining the DANA address of the suitable component B by requesting the service discovering LSM, the management of component A wants to establish a channel to that component B. Therefore, it requests the LCM of its control unit for establishing a channel with the suitable QoS features. This manager takes the requesting role and initiates a three way handshake exchange of *Establish* messages with the requested LCM that is responsible for component B. During this messages exchange, the management of component B is asked by the requested LCM for granting the channel establishing. Furthermore, the RCM is involved by exchange of *MediaAlloc* messages to clarify whether the communication system (bus) can guarantee the QoS requirements. For this reason a RCM may contain a table to manage the bandwidth resources of the bus. After the agreement of all manager parties, the channel can be established, and data between components A and B can be exchanged.

### 4.3 Component Deployment

One of the main features of the envisioned software architecture is the possibility to dynamically reconfigure the system, allowing new services to enter it. Due to the possible various configurations for the system a configuration management is needed. Configuration management knows all available components by accessing the System Service Manager (SSM) and External Service Manager (XSM) and has to decide

where to run the component. This allows to dynamically adapt the system's behavior to different external requirements like available processing power, memory constraints, and cost issues related to the wireless connection between vehicle and the external infrastructure. Technically speaking this results in an optimization problem that has to be solved continuously during system life time.

Due to the remote accessibility of component interfaces, they may also be placed on infrastructure machines if the wireless connection into the infrastructure permits, and thus using the potentially higher computing power of such a device.

## 5 Classification of the Architecture

### 5.1 Consideration of Hardware Requirements

The described architecture is specified on different levels. It is possible to develop units that follow the architectural specification for service discovery and utilization with their external behavior. As a result, the internal implementation remains proprietary. Therefore, software of any unit can be developed regarding the given constraints. For instance, if developing software for embedded units enforces the usage of little RAM, it is impossible to implement the sophisticated concepts of component deployment. However, by implementing the LSM and LCM functionality, the software in a simple control unit can participate on the communication system and interact with the software of other units.

A more detailed component level specification exists besides that unit level specification. At this level, the architecture of components inside a unit is described. Software that follows this specification level allows full use of DANA's management.

Due to this approach the application of the management functionality is scalable.

### 5.2 Comparison to and Interaction with Other Technologies

A comparison of the presented architecture to more universal solutions like UPnP and Jini is necessary to classify the presented architecture.

A manifold and alterable set of services can be handled by universal technologies due to the existence of service discovery protocols. By using these protocols the complexity of the technologies becomes transparent to the user. Furthermore, the heterogeneity problem is solved, because all interacting parties agree in common to the IP standard to solve the addressing problems, and concerning Jini also to the Java standard to achieve platform independency.

However, in vehicles support of small control units is also required. Today, typical control units in vehicles are equipped with 60 kbyte ROM and 2 kbyte RAM [7]. Even a HTTP/TCP/IP stack in a UPnP supporting smart device according to [1] needs 60 kbyte of code. Furthermore, Jini needs additional RAM due to the capability to exchange Java objects and the realization of the Java class loading concept [3]. Today, Jini works only in a universal Java 2 environment.

DANA considers the existence of small control units in vehicles, because the application of the management functionality is scalable (see also section 5). Furthermore, in opposite to the compared technologies it explicitly regards resource management (e.g. bandwidth assignment, see also section 4.2). It is designed especially for the

vehicular environment and more convenient for that specific application area than universal technologies.

Because a coexistence of DANA and universal technologies in a vehicle is conceivable, the existence of an interlinking concept is necessary. An interlinking entity is both a component in the DANA managed part of the network and e.g. a Java server object in the Jini part of the network. This entity can represent one part of the network as a proxy on the opposite side. Thereby it is possible to bring the small control units to the universal technologies.

### 5.3 Implementation Aspects

The presented architecture allows a large freedom in its realization. Programming paradigms, programming languages, platforms and communication protocols are not enforced by the architecture but are exchangeable technologies.

We use Java as implementation language of our prototype, because with Java a uniform component platform is given. It is important to mention that Java is not inevitably greedy of memory and performance. For instance, there exist some promising approaches that apply the Personal Java Standard. JBed [11] with its minimal memory need of 10 kbyte is one representative. A compiler that compiles Java byte code to object code and a linker produce a binary that is stored in a boot ROM. Furthermore, the possible existence of a compiler and linker inside the control unit enables classloading and thereby component deployment at run time.

Currently, our prototype hardware consists of Linux PCs. We regard to use a minimal set of libraries (e.g. no IP networking) and a minimal set of operating system services. Channels are realized by applying communication services of the IEEE 1394 bus directly. We consider several QoS aspects of the communication between components. Synchronous A/V streams as well as asynchronous data can be transferred via the IEEE 1394 bus. A shift from applying PCs to applying small control units is currently underway.

## 6 Conclusions

A variety of application services in future vehicles is expected. However, it is obviously not possible to predict the kind of services that will emerge. Because of the long life cycle of a vehicle we need a software architecture that allows the alterability of the offered services sets. The presented architecture DANA enables to add new functionality by the deployment of new components.

In order to achieve updateable sets of services, management functionality is needed. This management consists of service discovery, enabling service utilization with respecting QoS requirements and the deployment of components.

The architecture satisfies the specific automotive requirements. In contrast to universal service discovering technologies, also control units with small CPU power and memory can be supported due to the scalability of the architecture. Therefore, software in small control units can interact with software in high capacity control units. The functioning of the system is achieved by an uniform architecture.

## References


1. B. Christensson, O. Larson: Universal Plug and Play Connects Smart Devices. WinHec 99 White Paper (2000)
2. C. Ciocan: The Domestic Digital Bus System (D2B). IEEE Transactions on Consumer Electronics, vol. 36, no. 3, (1990) 619-622
3. P. Clip: Java's Object-Oriented Communications. Byte, McGraw-Hill, Febr. (1998) 53-54
4. A. Jameel, M. Stümpfle, D. Jiang, A. Fuchs: Web on Wheels: Toward Internet-Enabled Cars. IEEE Computer, vol. 31 (1998) 69 -71
5. R. König, C. Thiel: Media Oriented Systems Transport (MOST) – Standard für Multimedia Networking im Fahrzeug. it + ti – Informationstechnik und Technische Informatik, vol. 5 (1999) 36-42
6. M. Bathelt: Komponententechnologie für verteilte Embedded Systems. In: B. Hindel (ed.): Technologie Forum Embedded Software, 3Soft GmbH, Erlangen (1999) 153-160
7. J. Schoof: Der OSEK/VDX-Standard. In: B. Hindel (ed.): Technologie Forum Embedded Software, 3Soft GmbH, Erlangen (1999) 7-19
8. G. Sigle: Digital Multimedia Broadcasting. Electronic Systems For Vehicles, VDI Berichte 1287 (1996) 373-380
9. M. Stümpfle, A. Jameel: Eine Internet basierte Komponentenarchitektur für Fahrzeug Client- und Serversysteme. In: R. Steinmetz (ed.): Kommunikation in Verteilten Systemen (KiVS), 11. ITG/GI-Fachtagung (1999) 6 - 19
10. C. Szyperski: Component-Software. Addison-Wesley, New York (1997)
11. J. Tryggvesson, T. Mattsson, H. Heeb: JBED: Java for Real-Time Systems. Dr. Dobb's Journal (1999)
12. W.H.W. Tuttlebee: Cordless Telecommunications Worldwide. Springer-Verlag, Berlin Heidelberg New York (1997) 372-412
13. D. Wright: Broadband: Business Services, Technologies and Stratetic Impact. Artech House, Norwood(1993)
14. IEEE: Standard for a High Performance Serial Bus. IEEE Std. 1394-1995 (1996)
15. IEEE: Personal Communications. The Magazine of Nomadic Communications and Computing, vol. 4 no. 4 (1997)
16. Microsoft Corporation: Universal Plug and Play Device Architecture Reference Specification. Version 0.90 (1999)
17. OMG: CORBA 2.3.1 Specification (1999)
18. Robert Bosch GmbH: CAN Specification 2.0 (1991)
19. Sun Microsystems. Inc.: Java Beans Specification, V. 1.01 (1997)
20. Sun Microsystems. Inc.: Jini Architecture Specification, V 1.0.1 (1999)