

A GENERIC SOFTWARE ARCHITECTURE FOR A DRIVER INFORMATION SYSTEM TO ORGANIZE AND OPERATE TRUCK PLATOONS



Andreas FRIEDRICHS Philipp MEISEN Klaus HENNING
RWTH Aachen University
Aachen, Germany

Abstract

One possibility to manage the continuous increase of freight transportation and to optimize utilisation of motorway capacities is the concept of truck platoons which form “trains on road” with the aid of Advanced Driver Assistance Systems. In this case the trucks are electronically coupled keeping very short gaps (approx. 10 meters) to form truck platoons on motorways. This contributes to optimisation of traffic flow and reduction of fuel consumption advantaged by slipstream driving. In this paper a brief introduction into these truck platoons is given as well as the description of the Driver Information System which helps the truck drivers to organize and operate these platoons. Furthermore a generic software architecture for the Driver Information System of the platoon system is presented. This architecture guarantees the development of a modern, flexible, extensible and easily configurable system, especially for Human Machine Interfaces of modern and future Advanced Driver Assistance Systems.

Keywords: Truck platoons, Driver information system, Advanced driver assistance systems, Software architecture.

Résumé

L'idée des « trains routiers » composés de convois de camions équipés de systèmes avancés d'aide à la conduite, permet de gérer l'accroissement du trafic de marchandises et d'optimiser l'exploitation des capacités routières. Dans ces convois sur autoroute, les camions sont couplés électroniquement à des distances courtes d'environ 10 mètres. Ceci contribue à la fluidité du trafic et à la réduction de la consommation de carburant par effet de traînée. Cet article donne une brève introduction du concept de convoi de camions et une description du système d'informations des conducteurs leur permettant d'organiser et utiliser ces convois. En outre, une architecture logicielle générique constituant la base de ce système est présentée. Elle garantit le développement d'un système moderne, flexible, extensible et facile à configurer, en particulier pour les interfaces homme-machine de systèmes actuels et futurs avancés d'aide à la conduite.

Mot-clés: Convois de camions, systèmes d'information au conducteur, systèmes d'assistance à la conduite, architecture logicielle.

1. Introduction into Electronically Coupled Truck Platoons

1.1 Problem Definition

The integration of the new European member countries is a challenging component for national traffic planning in the near future. Due to its central geographic position, Germany has to shoulder the majority of future traffic emergence. Additionally, this traffic will predominantly encumber the road. A modern national economy needs an efficient traffic system to successfully face such a challenge. Otherwise, today's even worse traffic situation will be preassigned to collapse. The importance of this fact was emphasized by studies from the European Commission in 2006. Between 1995 and 2005 a growth of 28 % in road freight transport was detected. Furthermore, until the year 2020 an increase of 41 % in road transportation is expected (European Commission, 2006).

1.2 The Approach

One possibility to meet the rising traffic volume on the roads is the modal shift to other types of transportation (e.g. rail, shipping). A further potential lies in the optimisation of the road-side traffic flow by evaluating the application type for driving assistance systems. In the future, such systems shall perform tasks which currently have to be executed manually by the truck driver.

Since the 1990's, Advanced Driver Assistance Systems (ADAS) for trucks have been on offer, including pre-adjustment of speed and distance to the front vehicle. This is exerted automatically via computerised engine- and brakes-management in connection with an automated transmission. The combination of an Adaptive Cruise Control (ACC) together with an Automatic Guidance (AG) leads to Autonomous Driving, similar to an autopilot in an airplane.

The difference here is the necessity of a leading vehicle. Following trucks can go far distances without any manual engagement by the driver as long as another ahead-driving vehicle exists. Nevertheless, each truck must be assigned with a truck driver at all times. Due to the platoons, smaller distances between the vehicles (up to 10 meters) can be realized. These truck platoons contribute to an optimization of traffic flow of up to 9 % and a reduction of fuel consumption of up to 10 % due to slipstream driving (Savelsberg, 2005).

1.3 Objective of the Project KONVOI

The objective of the project KONVOI is the development and evaluation of the practical use of truck platoons with ADAS. With the assistance of virtual and practical driving attempts, by using experimental vehicles and a truck driving simulator, the consequences and effects on the traffic will be analysed. Within the project KONVOI, four experimental vehicles will be equipped with the required automation-, information- and automotive-technology to build truck platoons. After sufficient testing of the ADAS for proof of system security as well as clarification insurance-legal aspects, the experimental vehicles can be subsequently tested on motorways with traffic.

1.4 The Platoon System

The main system components of the platoon system are the Advanced Driver Assistance System (ADAS) and the Driver Information System (DIS) (Figure 1). The longitudinal

guidance of the ADAS is based on a LIDAR distance sensor, which is used to determine the distance in longitudinal direction and the transversal offset to the leading vehicle. There is also a need for a vehicle-vehicle-communication (WLAN), to transfer necessary vehicle data from all platoon members, which is required for the ACC-control to realise the target distance of 10 meters. In all trucks a target acceleration interface is implemented, which automatically calculates the drive-train and the management of the different brakes in the vehicles. The transversal guidance of the ADAS is based on the transversal offset to the leading vehicle and the recording of its own track position with a CMOS image processing system as well as the analysis of the data flow from the vehicle-vehicle-communication. The necessary steering moment for the automated guidance of the trucks is realized via a steering actuator on the base of an electric motor in the vehicle, which is built as a dual circuit with detached energy supply.

With the help of the Driver Information System (DIS), the truck driver plans his route, selects economic platoon participants as well as initialises and respectively confirms the platoon manoeuvres in order to build and to dissolve the platoon. According to that, the DIS is the human machine interface (HMI) of the platoon system (see section 2). The platoon organisation is realised on a central server with a data-mining-algorithm under consideration of economic aspects (Meisen et al., 2008). For this task, the DIS has to send the time schedule, route plan and GPS position of the truck with a vehicle-infrastructure-communication via G3 to the central server.

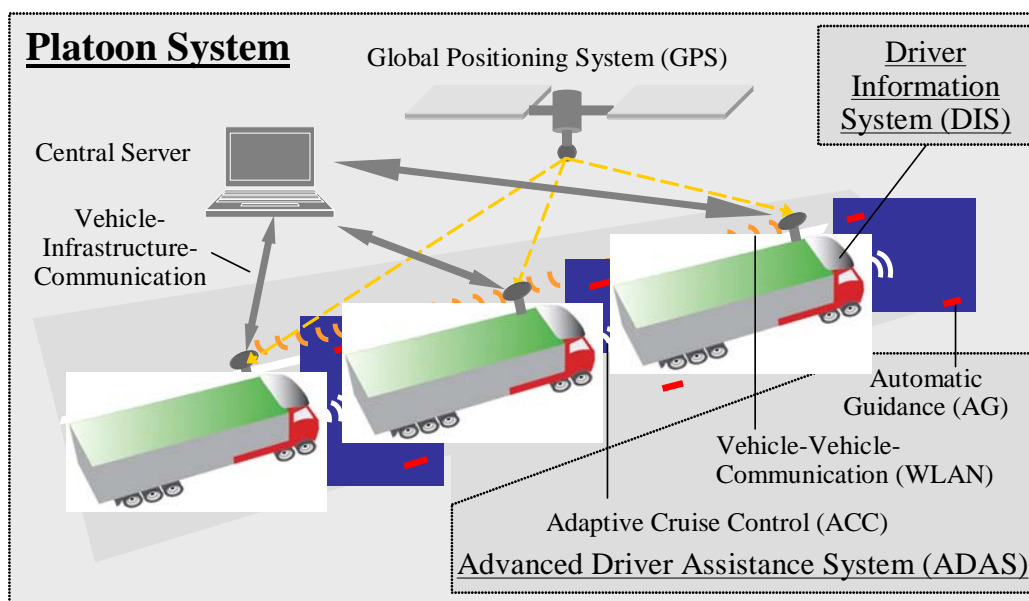


Figure 1 – The Platoon System (Friedrichs, in prep.)

1.5 Emphasis of this Paper

Within the following sections, the focus of this paper will be set on the DIS of the platoon systems and how a generic software architecture of a DIS as a HMI for ADAS should be designed. Consequently, the system architecture of the ADAS will not be the subject of this paper. A detailed overview of the system architecture can be found in Henning et al. (2007).

2. Driver Information System (DIS)

2.1 Overview – The DIS as the Information Manager

In this section the data processing concept of the DIS will be explained. Figure 2 makes clear, that the DIS is not only the HMI of the platoon system. In fact, the DIS is the information manager between the truck driver and the ADAS as well as the central server. Via touch screen the user input of the truck driver is processed by the DIS, and in dependency of the user input, the data is given to the technical systems. Vice versa the DIS processes the data of the technical systems and informs the driver in all platoon phases about the current platoon situation.

During the platoon organization, the driver has to enter the DIS settings, has to plan his route and time schedule and has to choose a suitable platoon from an offer list. This data is preprocessed by the DIS and sent via 3G to the central server. The most important data for the central server is the route information of each truck as well as the platoons chosen by the driver. On the other hand, the DIS gets a list with suitable platoons sorted by economic criteria from the central server. Furthermore, the central server immediately informs the DIS about any alterations within the planned platoons.

Two further tasks of the DIS are the navigation to the planned meeting points/destinations and the warning of danger areas such as road constructions, bridges, motorway junctions and tunnels. The platoons have to be dissolved manually by the truck driver ahead of these areas. During the platoon drive, the drivers have to initiate and respectively confirm all platoon maneuvers (connecting, dissolving and lane change). The control signals from the driver to the ADAS are sent through CAN-Bus. The DIS permanently informs the driver about the actual state of the platoon.

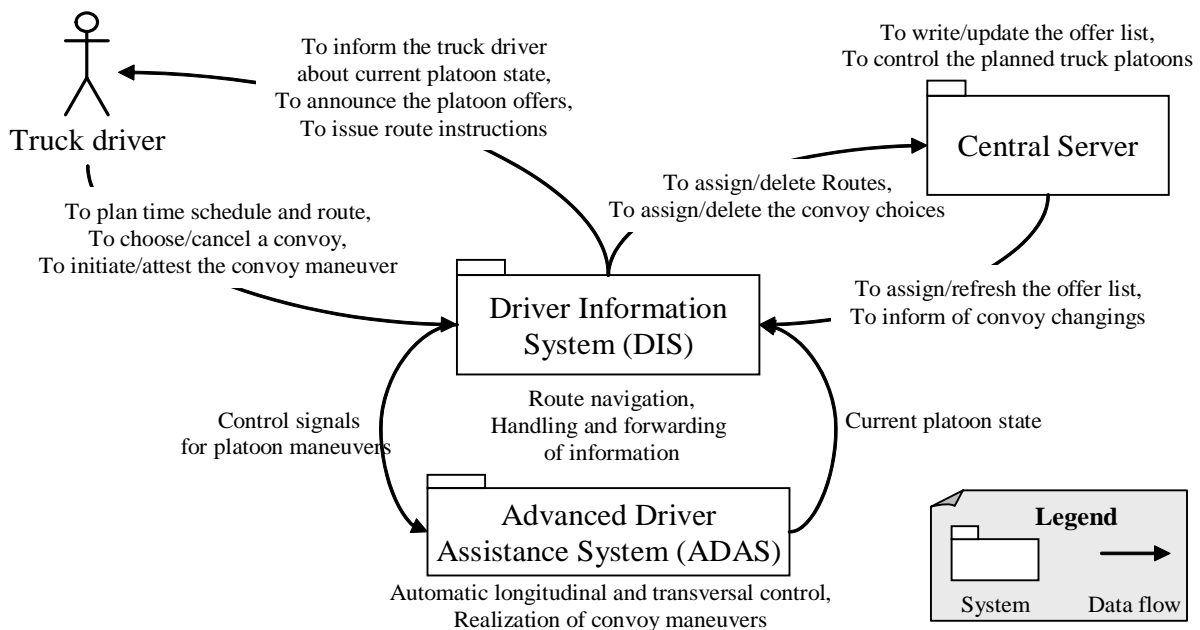


Figure 2 – The DIS as the Information Manager (Friedrichs, 2008)

2.2 Requirements Specification for the Software Architecture

The former section clarified, that the DIS is on the hand the HMI and on one hand the information manager of the platoon system. This must be explicitly considered in the specification of the requirements for the DIS software architecture. In the following, the requirements of the DIS software architecture are indicated with the letter “R” (for requirement) and a consecutive number, summarized in table 1.

Modularity, Extensibility, Flexibility and Configurability (R1)

The usual demands made to software architectures for an HMI are modularity, extensibility, flexibility and configurability. In this context modularity means that certain functionalities are combined in well-defined software components. These components have to be self-explanatory and exchangeable. For this purpose it is important that the interfaces between the modules are explicitly specified. The extensibility of the software architecture should be flexible so that additional functions can be added subsequently and easily. The whole software system – particularly the graphical design of the HMI – should be fast to configure and change with a configuration file so that no alterations of the source code are necessary.

Robustness and Reliability (R2)

The automobile sector has especially high demands on technical systems in vehicles concerning robustness and reliability (Wietzke/Tran, 2005). Therefore, the software architecture has to ensure the robustness and reliability through adequate safety mechanisms and functions.

Information and Data Management (R3)

The DIS as an information manager has to handle and process a large quantity of data as well as the transmitting of processed data to the corresponding technical systems. Therefore, the system architecture has to support an internal communication, in order to support the different software components with the required data. Furthermore, the software architecture has to support different functions to manage the data in the internal memory as well as in data bases.

External Communication with System Environment (R4)

The DIS must be able to communicate with their system environment. Accordingly, an external communication with appropriate communication interfaces (e.g. CAN-Bus, 3G) has to be implemented. Furthermore, the ease extensibility (also for other technologies e.g. WLAN, Flexray) has to be guaranteed by the software architecture (cf. requirement R1).

Interaction with the User (R5)

The DIS is the HMI of the platoon system. The relevant input is made by the driver on the user interface (e.g. touch screens). The system architecture must handle the user input through the user interface and must give the effects of the users’ manipulation back to the user interface so that the user can assess the system state.

Table 1 - Requirements for the Software Architecture

R1	Modularity, Extensibility, Flexibility and Configurability
R2	Robustness and Reliability
R3	Information and Data Management
R4	External Communication with System Environment
R5	Interaction with the User

2.3 Design of the Software Architecture

The architecture of a software system is the structure of the system, which comprises software components, the externally visible properties of those components and the relationships between them (Balzert, 2005). The choice of the architecture pattern, as the base for the software architecture, is a fundamental one and also one of the first decisions to make. These early decisions are the most important and can only be revised with much effort and time. Therefore the software developer basically refers to reliable and proven architecture patterns, to avoid the risk of wrong decisions. In the literature one can find a lot of different architecture patterns. Buschmann et al. (1998) categorizes these patterns into four groups (table 2): Mud-to-Structure, Distributed Systems, Interactive Systems and Adaptive Systems.

Table 2 - Examples of architecture patterns (Buschmann et al., 1998)

Architecture group	Examples for architecture pattern
Mud-to-Structure	Layers, Pipes and Filters, Multi-Tier, Blackboard
Distributed Systems	Mediator, Client-Server
Interactive Systems	Model-View-Controller (MVC), Presentation-Abstraction-Control (PAC)
Adaptive Systems	Microkernel, Reflection

The architecture pattern of the group Mud-to-Structure should organize the components and objects of software systems (Buschmann et al., 1998). The functionality of the software systems is divided into cooperative layers. The most common examples are Layers, Pipes and Filters, Multi-Tier and Blackboard architectures. The Distributed Systems are serving to support the distributed resources and services in networks (Buschmann et al., 1998). Two examples are Mediator and Client-Server architecture. In the area of Interactive Systems it turned out that software for a graphical user interface should have another structure as applications which display their results on terminals (Budzuhn, 2002). These software systems should separate the control of the program from their view. The most important patterns for this group are the Model-View-Controller (MVC) and the Presentation-Abstraction-Control (PAC) architectures. The last group is represented by the Adaptive Systems. In particular, the extensibility and adaptability from software systems should be supported (Buschmann et al., 1998). The most common patterns for this group are Microkernel and Reflection.

Choice of an architecture pattern

The decision for an architecture pattern must be made on the basis of the defined requirements (section 2.2). Hence, the DIS as the HMI and information manager for the platoon system must be allocated one of the previously presented architecture groups. The claimed interaction with the user (R5) classifies the DIS as an Interactive System. The DIS has to process all the incoming data (R3) and has to exchange this data with the system environment, thereby relying on the external communication interfaces (R4). Compared to the required aspects of modularity, extensibility, flexibility and configurability (R1), the DIS can be considered also as an Adaptive System, which must run robustly and reliably in the car (R2). Consequently, the defined requirements can be summarized as the claim for a modular, extensible, flexible and configurable HMI, whereby especially the aspect of extensibility accounts for the modularity and flexibility of the software architecture. The denotation of the DIS as a HMI makes the importance of a structured human-machine interaction obvious, so finally the Model-View-Controller (MVC) architecture was chosen for the development of the

DIS as an Interactive System. The PAC architecture can be disregarded, because this pattern is especially used for the development of agent systems (Buschmann et al., 1998).

Model-View-Controller (MVC) architecture

The MVC architecture pattern was first introduced by Reenskaug (1979) as a common design solution to administrate great and complex data. In the following years it turned out that this architecture pattern covers more application areas. So today MVC architecture is used in various software projects (e.g. the class library Qt for platform independent programming of graphical user interfaces). In general, the MVC architecture pattern allows illustrating the data model (model) in different ways on the graphical user interface (views). Through program logics (controllers) the input of the data is administrated. The data models as well as their manipulation schemes are available for different graphical elements and controls. The extensibility of software systems created with the MVC pattern with new designs and controls is easily possible due to the explicit interface description (Budszuhn, 2002).

Basically, the component concept of the DIS is built on the basis of the MVC pattern introduced before. In some cases the functionality of the software components are slightly different from the ones in the MVC literature. The architecture – following the MVC architecture pattern – is based on three core components, a Controller-, a View- and a Model-Component. The Model-component serves as a collection of abstract data structures, which only administers the data. Unlike the MVC pattern, where the Model-component is responsible for manipulating the dates, the Model-component in the adapted MVC pattern has solely administrative tasks. These tasks are primarily the access control and thread backup. The other components (Controller and View) are unknown to the Model-component. Along the lines of the general MVC pattern, the View-component displays the information provided by the Model-component. Data manipulation or other alterations by user interaction are communicated to the View-component by the Controller-component. The View-component knows the Model- as well as the Controller-component in order to readout data about user interaction. Simultaneously, it administers the various Gui-components. These comprise graphical elements, which visualize selected data from the Model-component for the user. The logic and functionality of the software is located in the Controller-component. This component evaluates user interactions and manipulates data. In addition the Controller informs the View-component about data manipulation or special requests of the user, which lead to a new or modified notation of a Gui-component (e.g. change of notation, deactivation of functionality). The Controller-component also administers Logic- and Comm-components. The Controller on the one hand extends the program logic and on the other hand enables the possibility of external communication (e.g. CAN, 3G).

2.4 Implementation of the Software-Architecture

Figure 3 shows a simplified UML class diagram of the DIS software architecture. The components Model, View, Controller, Logic, Gui, Comm and ModelData are realized through several classes. To avoid complexity, a well arranged overview is given in the figure, by excluding the attributes, methods and multiplicities of the classes. Basically the software architecture is divided into core components, abstract extension-components, concrete extension-components and auxiliary components. This concept enables, next to the chosen software architecture, a high flexibility concerning the extensibility with program logic, data models and design elements. In addition, dynamic libraries are used to uncouple the core source code of the concrete extension-components from the main program of the DIS. Also a static library for the DIS is used, where all necessary definitions (e.g. classes, auxiliary

classes) for the implementation of the software system are included. In the following, the realization of the different software components will be explained.

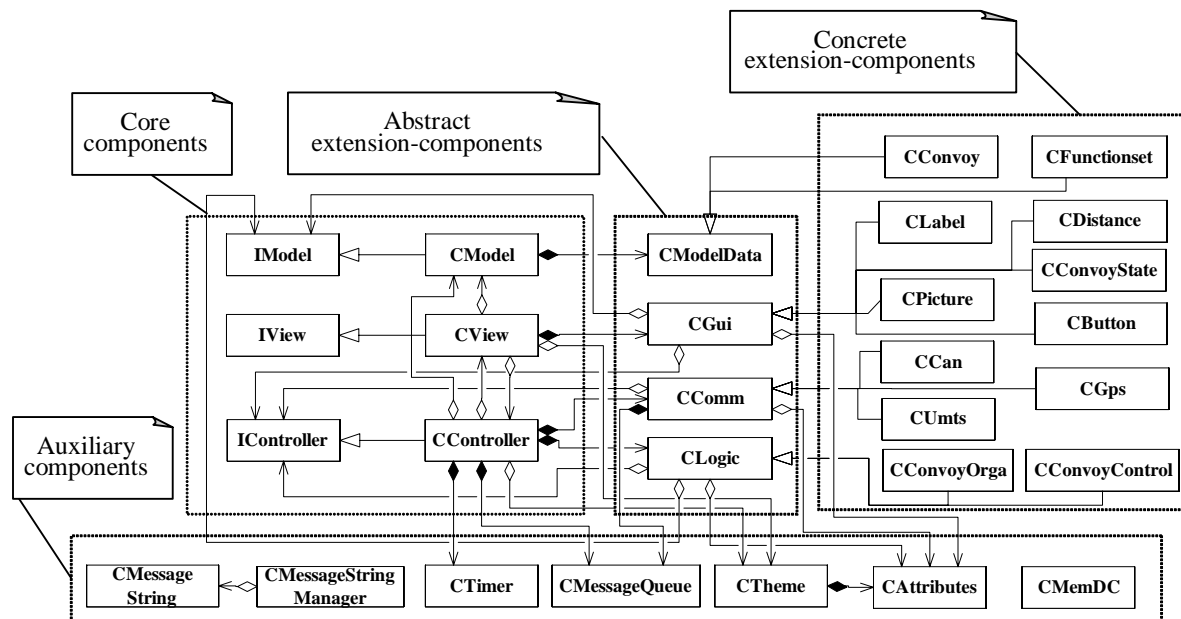


Figure 3 - UML class diagram of the DIS software architecture (Friedrichs, 2008)

Core Components

The core components inherit from interface definitions (*IModel*, *IView* and *IController*), so that some functionalities are concealed to the extension- and auxiliary components and access is only allowed to designated functionalities. The overall functionality (according to the visibility of the methods of the classes) is only known to the core components among themselves and can only be used by them. The internal communication between the Controller- and the View-components are done with an interprocess communication. Asynchrony messages are saved into message queues until the recipient retrieves them. Both core components have their own message queue. The DIS has to handle different threads during the runtime. In such a multithread-application thread safety is very important. A piece of software code is thread-safe if it functions correctly during simultaneous execution by multiple threads (Budszuhn, 2002). In the software architecture the message queues as well as the model-components are thread-safe implemented. For this purpose the synchronization mechanism *CSingleLock* and *CMultiLock* from the Microsoft Foundation Classes (MFC) of Microsoft was used. A lock is used to ensure that only one resource respectively one critical section in a software component can be used by a thread. The other threads have to wait – due to the closed locks – until the critical sections are opened for the next thread.

Abstract Extension-Components

An abstract extension-component is an abstract class, i.e. a class not completely implemented concerning the method definitions. In an abstract class some methods are defined, other methods – so called “pure virtual functions” – are not declared. More precisely this means that from an abstract class no object can be derived. In the DIS software architecture the Model-component is extended with ModelData-components, to add, remove or read out data. The View is extended with the abstract class *CGui*. The Controller is extended with two abstract extension-components: The class *CComm* for the external communication (e.g. UMTS, CAN) and the class *CLogic* for the core functionality of the software system. The

class CComm also has a message queue, which was described in the previous section. All abstract classes have to be extended with concrete classes.

Concrete Extension-Components

Concrete extension-components are concrete classes which extend the software system through logic functionalities (Logic-components: e.g. the functionality to organize and operate truck platoons), data (ModelData-components: e.g. to manage the convoy state) and views (Gui-components: the different elements for the graphical user interface, for instance pictures, buttons, labels etc.). To do this the concrete extension-components inherit from the abstract extensions components and complete the non-defined methods of the abstract classes. The abstract extension-components provide a quantity of code for the internal communication and processing within the software system, so that during the development of a concrete extension component, the relevant part of the component can be focused. Moreover, this procedure secures that every concrete extension component makes the required interfaces available. Most of the concrete extension-components are implemented in Dynamic Linked Libraries.

Auxiliary Components

The DIS software architecture provides a set of auxiliary components in a static library. Static libraries, unlike dynamic ones, are not linked with the program during runtime of the application, but already during compilation. As shown in figure 3, the static library includes the interfaces of the core components (IModel, IView, IController), the abstract extension components (CLogic, CComm, CGui, CModelData) and some auxiliary classes, for instance to support Double-Buffering (CMemDC, CDC), thread safety (CMessageQueue), manipulating text strings (CString) and timer functions (CTimer).

Configurability of the Software System with XML-Files

The design of the software architecture intends the configuration of the software system with a XML configuration file. In this XML-file the configuration of the extension-components, the behavior of the software systems by user interaction, the functionality of the core components and the design of the graphical user interface is specified.

3. Conclusion

In this paper a brief introduction into electronically coupled truck platoons with Advanced Driver Assistance Systems was given. Furthermore the Driver Information System to organize and operate truck platoons was introduced. Following, the requirements for the software architecture of the Driver Information System as the HMI and information manager of the platoon system were derived and transferred into a software design. Finally, the implementation of the architecture was described in detail.

The presented software architecture fulfills all the fundamental demands for the development of interactive software systems in the automotive sector. Hereby the architecture ensures the user interaction between the driver and the technical systems as well as the data processing between the different system components in the vehicle. This architecture guarantees a modern, flexible, extensible and easily configurable system, especially for HMI of driver information and assistance systems. Due to its interactive and adaptive characteristics, the presented architecture could be moreover seen as a generic software architecture framework. The development of the Driver Information System for the organization and operation of truck platoons and its software architecture took place in the project KONVOI. In the first

instance, the Driver Information System was implemented into a Truck Driving Simulator (Figure 4) which was used as a test environment for the module, integration and system tests. The acceptance test with finally 27 truck drivers from the freight forwarding companies of the project consortium showed a quite positive assessment and proved the functionality as well as the reliability of the software system (Friedrichs, in prep.).

In the next step, the trial implementation of the Driver Information System in four truck experimental vehicles is planned. The first test drives with two electronically coupled trucks on a closed motorway in the Netherlands were done in November 2007.

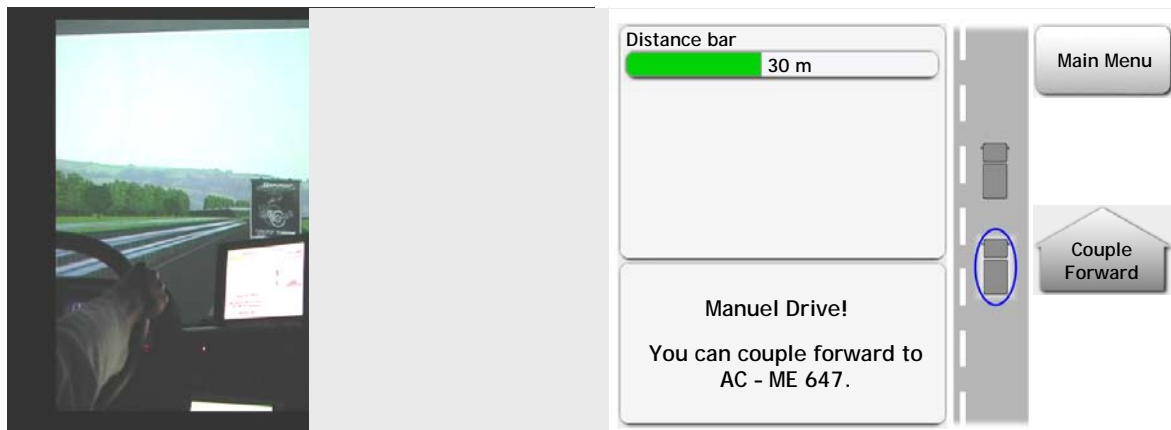


Figure 4 - Truck platoons in a truck driving simulator and the graphical user interface of the DIS to organize and operate truck platoons (Friedrichs, 2008)

4. References

- Balzert, H. (2005), "Grundlagen der Informatik", Spektrum Verlag.
- Budzuhn, F. (2002), "Visual C++-Programmierung mit den MFC", Addison-Wesley.
- Buschmann, F., Meunier, R., Rohnert, H. (1998), "Pattern-Oriented Software Architecture", A System of Patterns, John Wiley.
- Commission of the European Communities, (2006), "Keep Europe moving - Sustainable mobility for our continent", Mid-term review of the European Commission's 2001 Transport White Paper, Brussels.
- Friedrichs, A. (2008), "A Driver Information System for Truck Platoons", doctoral thesis at ZLW/IMA, RWTH Aachen University, VDI Verlag, Duesseldorf, in press.
- Henning, K., Preuschoff, E. (2003), "Einsatzszenarien für Fahrerassistenzsysteme im Strassengueterverkehr und deren Bewertung", VDI Verlag, Reihe 12, Nr. 531, Düsseldorf.
- Savelsberg, E. (2005), "Lastenheft für Elektronisch Gekoppelte Lkw-Konvois", VDI Verlag, Reihe 22, Nr. 21, Düsseldorf.
- Wietzke, J., Tran, M. (2005), "Automotive Embedded Systems, Springer Verlag, Berlin.
- Henning, K., Wallentowitz, H., Abel, D. (2007), "Das Lkw-Konvoisystem aus den Perspektiven Informations-, Fahrzeug- und Automatisierungstechnik", in Proceedings of the conference Mechatronik 2007, VDI Verlag, Düsseldorf, 133-147.
- Meisen, P., Henning, K., Seidl, T. (2008), "A Data-Mining Technique for the Planning and Organization of Truck Platoons", in Proceedings of the International Conference on Heavy Vehicles, Paris.
- Reenskaug, T. (1979), "Thing-Model-View-Editor, An Example From a Planning System", In Xerox PARC technical note, May 1979.