SOAcom: Designing Service communication in adaptive automotive networks

Alberto Ballesteros*, Marco Wagner†, Dieter Zöbel‡
*Universitat de les Illes Balears
{a.ballesteros}@uib.es
†Heilbronn University
{marco.wagner}@hs-heilbronn.de
‡University of Koblenz-Landau
{zoebel}@uni-koblenz.de

***Abstract*—This paper describes SOAcom, a communication model as well as a development process to characterize and adapt automotive network protocols to Service-based Driver Assistance Systems (DAS). Future DAS for articulated vehicles like a car with a caravan are characterized by a dynamic software and hardware architecture. In these systems Service-orientation is a promising approach. One major demand to be able to deploy such systems in an automotive environment is the ability to allow Service communication through the specialized automotive networks. This paper describes a development process to define a communication model that is built up by mechanisms which are necessary to allow this Service communication. The development process, consisting of four phases, is tested by defining a communication model for a Service-based DAS on the Controller Area Network (CAN). The result is then evaluated by means of a set of experiments.**

I. Introduction

In recent years the usage of Middleware technologies within the domain of Distributed Embedded Systems (DES) has risen. Middlewares have been identified to be of great use when these systems get very complex or heterogeneous, the time-to-market has to be reduced or the systems have to be runtime adaptive. One of the most popular Middleware approaches is the paradigm of Service-oriented Architecture (SOA). This paradigm has for example been used in the eSOA project [1] or in the SIRENA project [2]. Whereas these projects mainly target on embedded devices that are connected using IP-based protocols, the automotive domain, which also relies on embedded systems, is using specialised networks that do not allow to run Service communication directly. These network systems such as the Controller Area Network (CAN) or the Local Interconnect Network (LIN) are developed to be used in static scenarios where changes of the system are not taken into account. Furthermore they are used to directly hand over raw data instead of being a base for high-level protocols which is another difference from, for instance, Ethernet. As an example we focus on Driver Assistance Systems (DAS) for vehicles with trailers. The task of driving a vehicle with a trailer attached has at least two difficult situations. One of it is to stabilize the trailer when driving forward. The other one is to back up the combination as controlling the movements of the trailer is not intuitive to many drivers but needs a lot of training instead. These systems caught our attention because the components of these DAS are distributed over both parts of the combination. Since a trailer is typically connected and disconnected to different trucks several times a day there is a lot of change in the system as every combination might have a different set of hardware and software components available for the assistance. These changes demand the system to orchestrate itself in a functional way to offer the driver the best assistance possible. We call this class of systems Distributed Driver Assistance Systems (DDAS). In order to handle the challenges these DDAS bring up, the SOA paradigm is used. Figure 1 shows an extract of the Architecture which is used for every single Service. It is divided into four different layers:

1) **Application** Holds the implementation of the actual Service Logic.
2) **SOA Middleware** Implements the SOA specific paradigms and functionalities.
3) **Communication Model** Contains software components to adjust the network used to the SOA Middleware.
4) **Hardware Abstraction Layer** Implements the low level network drivers.

This paper focuses on SOAcom, a combination of a communication model as well as a development process which guides the software engineer through the development of this component. The SOAcom communication model has to fulfil the following requirements:

1) **Runtime Adaption** The communication model developed must allow connecting or disconnecting Services at runtime and has to handle the integration of new Services into the existing network.
2) **Ensuring advantageous attributes** All of the different network systems are designed for a special class of applications and therefore have unique attributes to ensure the efficiency, safety or timeliness demanded in these applications. These advantageous attributes need to be ensured and preserved wherever possible.
3) **Interoperability** Th communication model should be able to connect Services using a network which is used by other components, running traditional communication at the same time.

4) **Stability after initial configuration** In the event of adding a new Service to the communication channel, existing nodes should not have to re-configure their communication stack.

The rest of the paper is organized as follows. Section II will give an overview of the SOAcom communication model developed. Section III explains the associated SOAcom Development Process in detail. This Development Process is used to construct the implementation of the communication model of an example application in section IV. Section V summarizes the related work in this domain and section VI concludes the paper.

II. A COMMUNICATION MODEL FOR SOA-BASED DDAS

As stated in section I, the SOAcom communication model builds the bridge between the high-level communication in the SOA Middleware and the low-level automotive network drivers. Still, it is more than a simple wrapper since it has to compensate the shortcomings of the automotive network system related to the application requirements. It maps the functionalities like for example discovery calls to find another Service in the network to actual messages. Finally, it also organizes the integration of the Service into the underlying network and controls the fragmentation process. As shown in figure 1 it contains four major components:

1) **Addressing Scheme**
   This component maps the Service calls onto message addresses and vice versa. It contains a table which holds all the relevant information to do so. As some of the automotive network systems link priorities directly to message addresses this component has to be aware of this, too.
2) **Adaption**
   The adaption component organizes the integration of the Service within the network. It is called when the Service is added to a network, in order to announce the existence of itself and reserve addresses or time slots to be able to take part in the communication.
3) **Fragmentation**
   As some Service calls might be larger in size as the maximum load of a single message in the underlying network, a fragmentation process must be available. This component brakes down the calls into message sized junks in order to send it over the network. As well, it assembles incoming messages to reconstruct the Service calls received.
4) **Transmission Arbitration**
   Since all automotive network systems use shared segments, in order to send messages, a channel access control mechanism has to be established. This mechanism can be controlled by a central device like in some time division approaches or can be distributed over the nodes in the network. The Transmission Arbitration component makes sure that the network's channel access method keeps the priorities and scheduling defined in the application.

Those components as listed above have to be implemented for every assistance project depending on the requirements set up by the application and the actual network used. As any particular implementation of the communication model is quite complex, we defined a well structured development process that guides the software engineer through several steps. Following the development process described in the next chapter will result in an executable implementation of the SOAcom Communication Model.

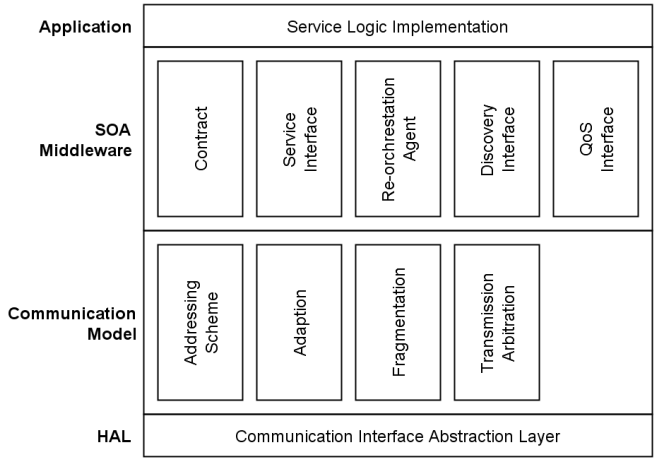


Fig. 1: Overview of the SOAcom architecture.

III. A DEVELOPMENT PROCESS FOR COMMUNICATION MODELS IN AUTOMOTIVE SOA-BASED SYSTEMS

This section will give an overview over the SOAcom process model. Furthermore it will describe exemplary some of the steps in high detail.

A. Overview of the SOAcom process model

The SOAcom process model consists of four major steps. These four steps guide the developer in creating a communication model for a specific application using a specific automotive network. Figure 2 gives an overview over the whole process.

The process starts with phase 1, which uses an extended version of a process model published in [3] which, in turn, models the functional elements of a SOA-based DDAS. As soon as a model of the application is acquired, we are able to extract the requirements of the application which influence the communication. In phase 2, the network protocol used is analysed. It can be done completely independent from the actual application as it only concerns about the network itself. We developed a questionnaire that guides the developer through this phase and summarizes those characteristics of the network which are important for designing a communication model. Phase 3 brings together the results of the phases 1 and 2 to identify which steps have to be taken to allow the usage of the desired network protocol in the specific

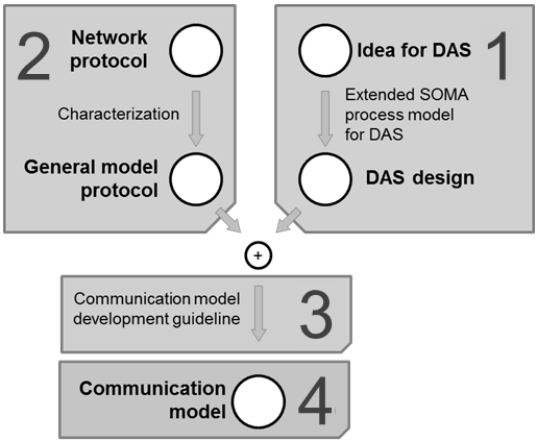


Fig. 2: Overview of the SOAcom development process.

application. Finally, phase 4 consists of the execution of the tasks identified in phase 3 and hereby in the implementation of the communication model. The rest of this section will describe the four phases in detail.

B. Phase 1: Determining the requirements of the communication model set up by the application.

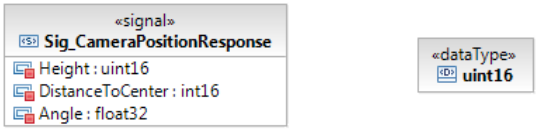
In [3] we describe a model-driven development process for SOA-based driver assistance systems. This process allows to generate a detailed description of such a system in SoaML starting from the plain idea of a DAS. By working through the phases of the development process described there, the used Services are derived from the functional requirements. These Services are enriched with descriptions about the provided and requested functionalities, as well as the contract. A contract in a SoaML model is an entity that describes how providers and consumers exchange data to one another. Therefore the UML metaclass collaboration is extended to define the roles of the interacting partners as well as their behaviour ([4]).

This SoaML model, as presented in the publication, offers a detailed description of the communication exchanged between the Service and its requester. This description serves as a formalism to derive the requirements set up by the application that have to be fulfilled by the communication model. However, there is still a gap within the model. Inside the description of the communication in each contract, which is modelled using UML Sequence Charts, messages sent by both partners are not defined in detail. Instead, only a placeholder lacking of any further details is used. In order to be able to define the communication model there have to be more details about the messages used.

To solve this issue we extended the specification phase. The main idea is to use UML Signals as a detailed description of a message. Now, a UML Signal is able to hold parameters whose Type can be defined accurately.

Figure 3a presents an example of such a Signal. In this case the name of the Signal is `Sig_CameraPositionResponse`. It has three attributes defined alongside with their type. For example, the attribute `Height` is a unsigned integer with 16 bits of length.

The data types used for the attributes of the Signals have to be defined as well. We decided to set up twelve different types which include, for example, the standard data types of the common programming languages like int8, float32 or ascii. Figure 3b presents one of these data types.



(a) Signal. (b) Data type.

Fig. 3: Example elements.

All Signals and data types are defined in a separate package to retain clarity. In order to be able to use the Signals, the type of the messages had to be changed as well. Instead of using Synchronous Messages, we changed all the contracts in the model to use Asynchronous Signal Messages. These messages can directly be mapped to a Signal which carries the detailed information. Figure 4 presents an example of a sequence chart within a contract which uses this kind of messages.

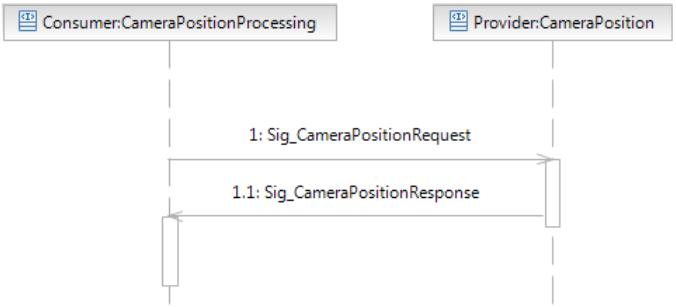


Fig. 4: Example of a communication sequence using Asynchronous Signal Messages.

With this extension to the process model we now have enough details to derive the requirements of the application regarding the communication aspects. These requirements are:

1) The number of Services in the application.
2) A list of Services alongside with their Service Class Address.
3) The size in Bytes of the biggest message in the application.
4) The presence of periodic messages in the application.

The number of Services can be easily discovered by counting the number of participants in the overall architecture, which is part of the SoaML model. For big size applications this job could also be done using a simple program. The second requirement, the list of Services, can be made by inspecting the Interfaces Package within the model. This package holds all the descriptions of the used Services and is hereby predestined to be used to gather the needed information. The corresponding Service Class Address on the other

hand, is part of our overall model for the usage of SOA in the automotive domain and can be looked up in the corresponding document.

The most convenient way to determine the biggest message in the model is to analyse the Signals and Datatypes package which holds all of the Signals used. Again, an automated script program which analyses the UML model could be used to reduce the effort of the developer. The last step, unfortunately, takes some effort to be done. Since there is no central package or file which offers an overview of all the communication sequences, the developer has to go through the sequence charts of all the contracts in the model. If there is only one periodic message in any of the contracts, this has to be reported. This last analysis of the SoaML model of the application completes the step of deriving the requirements set up by the application.

C. Phase 2: Characterization of the network protocol

Simultaneously with phase 1 we execute phase 2, which helps to provide a high-level description of the automotive network protocol. More specifically, this description allows to hide all the complexity of the network protocol's implementation and focus on its capabilities.

Nowadays, most network protocols are described using formal descriptions, like formal languages or state diagrams. The former ones describe the data structures involved and how they are inter-exchanged among the communication participants, whereas the second ones specify the possible states of the subcomponents and the events causing a change of the state. These formal descriptions provide a very low-level specification of the network protocol's behaviour, so they can contain all functional details. However, these descriptions are targeting on an easy implementation of the network components, rather than providing a high-level description. This leads to the fact that they cannot be used directly to derive network characteristics in an efficient way. In order to characterize network protocols, we developed a characterization method based on a questionnaire.

Next, we describe the list of attributes specified to describe a network protocol, as well as the questionnaire, which makes possible to collect its values.

1) Characteristic attributes of a network protocol: The newly designed network protocol description paradigm summarizes the most important attributes of the network. This list of attributes is divided in five different groups, the so-called network characteristics. The first group, called *transmission*, describes how the protocol transmits data to the channel. The second group, called *physical capabilities*, details the topology and physical medium. The third group, called *network capabilities*, gathers the characteristics concerning the whole network. The fourth group, called *dynamic capabilities*, manifests how the protocol adapts to environment changes. Finally, the fifth group, called *dependability*, describes the safety and security mechanisms provided. Figure 5 gives an overview over the characteristics used in our approach.

2) A questionnaire to characterize a network protocol: In order to facilitate the collection of the attributes' values we have set up a questionnaire, that is, a list of questions regarding the network protocol. The answer to these questions can be of any of the following three types:

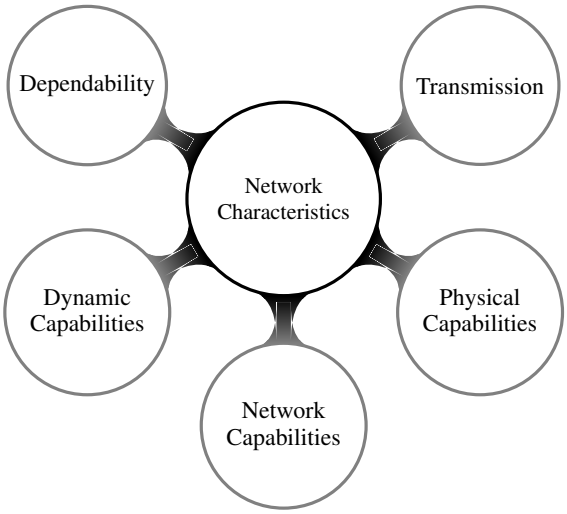


Fig. 5: Groups of characteristics of a network system.

- **Checkboxes** for allowing multiple answers to a question.
- **Radio Buttons** for restricting the possible number of answers to one.
- **Free text answers** to allow for example the input of numeric values.

An example of one of these questions is the test for the prioritization mechanism of messages in a network. This question is of the Radio Buttons type and offers the options "Node-based", "Message-based", "Scheduler-based" or "No Prioritization". Table I shows the corresponding extract of the questionnaire.

Frame prioritization
o Node-based
o Message-based
o Scheduler-based
o No prioritization

TABLE I: Extract of the questionnaire to characterize a network.

The outcome of this phase is the so-called general model of the protocol. That is, the list of the values of the attributes, from which we could extract the network protocol's capabilities. This information, together with the application requirements from phase 1, is exploited in next phase.

D. Phase 3: Mapping requirements to attributes

In this third phase we identify the automotive network protocol shortcomings, in the scope of the application, and set up the list of tasks which must be carried out to create the communication model. For this, the information generated in the two previous phases is collected. On the one hand, phase 1

provides the list of application's communication requirements which, as explained, is composed of four different values. On the other hand, phase 2 supplies the list of the network protocol attributes' values, which highlights the automotive network protocol capabilities.

We handle these two information streams by means of a set of flowchart diagrams. This kind of diagrams allows a software designer to represent algorithms or processes. In our case, they help the developer in the process of identifying the network protocol shortcomings and, in a second step, they present the tasks to be carried out to overcome them and, thus, construct the communication mode.

Next, we describe these diagrams and how they should be used. After that, a flowchart diagram example, jointly with the extended description of all of its contained tasks, is provided.

1) Flowchart diagrams to guide the developer through the process: Each of the flowchart diagrams is constructed from a structured set of questions and a list of possible tasks. On the one hand, questions relate the application's requirements from phase 1 with the attributes' values from phase 2 and, thus, guide the developer in the process. On the other hand, each time a diagram highlights a shortcoming, the task allowing to overcome it is presented. Each shortcoming alongside with the task to overcome it is recorded. The outcome of this phase is the list holding all the tasks to be executed by the developer. Flowchart diagrams are complemented by a *text document* which contains an explanation of each question, the repercussions of each answer and an extended description of each task.

We have constructed six different flowchart diagrams, each of which is related to at least one attribute of the network protocol. Moreover, a given diagram can embrace various attributes of various groups. This is because diagrams are directly related to communication model components and not to questionnaire groups. Next, flowchart diagrams created for this phase are introduced.

First, the *Addressing scheme* diagram helps to set up the addressing, so messages can be delivered to the corresponding receivers. For this, it is necessary to take into account the characteristics of the network protocol addressing scheme, the network protocol transmission method and the set of Services defined in the SOA model.

Second, the *Ability to assign addresses dynamically* diagram helps to extend the dynamic capabilities of the network protocol, so it can assign addresses at runtime to the Services and, in some cases, to the nodes. For this, it is necessary to distinguish between node-oriented (direct) and message-oriented (indirect) network addressing modes.

Third, the *Mode of communication* diagram assists the developer in adapting Service communication to the network's channel arbitration method. For this, it is necessary to know whether the network protocol uses a master/slave or a multi-master mode of communication.

Fourth, similarly to the mode of communication diagram, the *mode of prioritization* diagram assists the developer in checking whether the network is able to manage the prioritization of new Services dynamically or not.

Fifth, the *Fragmentation* diagram helps to ensure that the network protocol is able to transmit messages when their content overcomes the maximum payload. For this, it is necessary to know the size of the messages, as well as the message payload of the network protocol, and whether it already provides a fragmentation protocol or not.

Finally, the *Trigger condition* diagram helps to schedule the messages. For this, it is necessary to take into account the possibility of periodic messages and the types of scheduling schemes supported by the network protocol.

2) Flowchart diagram example: Here we present the "ability to assign addresses dynamically" diagram, shown in figure 6. Note that diamonds represent conditional statements containing the questions and, thus, are the elements that may change the execution flow. As concerns the rectangles, white-filled ones present important information, regarding the flow, whereas gray-filled ones hold a short description of a task to be carried out. Finally, the labels along together with the diamonds refer to the specific section in the *text document* which corresponds to them.

Once the network protocol shortcomings have been identified, we can define the set of tasks to be carried out. As shown in figure 6, shortcomings are followed by a short task description. However, the complete definition of the task, as well as the possible options to be taken, are compiled in the *text document*. Next, a summarized version of the document's extract corresponding to this diagram is shown.

Since we are constructing an adaptive architecture, the communication model must allow connecting new Services at runtime. Moreover, in order to be able to deliver messages to these new Services, it is necessary to ensure that they can acquire a unique address dynamically.

The first conditional statement, labelled 2.1, checks whether the network protocol is node-oriented. In this case, messages are directly addressed to the receiving nodes and, thus, each node must own a unique address. In this sense, the communication model must ensure that new nodes and/or Services installed at runtime have unique addresses.

This requirement is expressed in the second conditional statement, labelled 2.1.1. In case the network protocol does not provide such a mechanism, the designer must provide a protocol which does it. This protocol should preserve the mode of network. More specifically, the protocol should not force to add a master node when dealing with a multi-master scheme. One possible option is to check the existence of high-level implementations of the network protocol. These implementations improve the basic operation of the network protocol and, thus, they are a good point of start when looking for extensions. In case any of them already implement a suitable algorithm to assign addresses at runtime, the designer can extract it and then add it into the communication model.

Once new physical components of the network are able to retrieve a unique address, new logical components -the Services- must be provided with unique addresses, too. In this

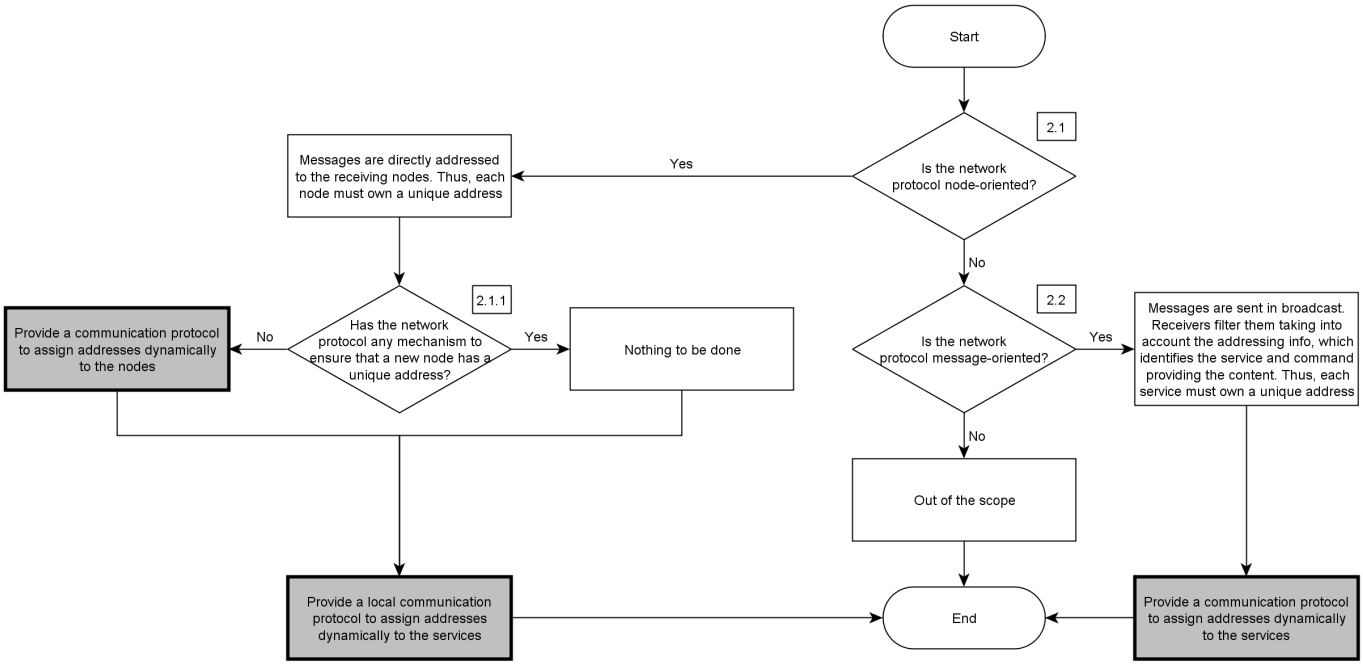


Fig. 6: Example of flowchart diagram.

sense, the designer of the communication model must provide an algorithm to assign unique addresses to the Services dynamically. Note that, since part of the node-oriented addressing scheme specifies the address of the node this algorithm can be executed locally, inside the physical device.

A completely different flow is chosen when the network protocol is message-oriented, see label 2.2 in figure 6. Messages in such systems are sent in broadcast and receivers are the ones responsible for filtering them. This can be done thanks to the addressing information, which identifies the content of each message. In our Service communication, this address identifies a Service call. As the addressing is not node-based, the designer does not have to assign addresses to the nodes but directly to the Services. Consequently, an algorithm to assign unique addresses to the Services dynamically must be provided. One possible option is to use the algorithm presented in [5]. This algorithm is based on an algorithm to elect a leader in a Distributed Computing System. The main advantage of using this algorithm is that all the address assignment process can be carried out without a central logical component.

E. Phase 4: Implementing the components of the communication model

Finally, the communication model is constructed by defining the content of each of its components. This is done by executing the tasks collected in the previous phase and hereby extending the capabilities of the network.

One important characteristic of SOAcom is its traceability. In this sense, we are able to backtrack each implemented component to a task and, beyond that, to the flowchart which brought this task onto our to-do list. Furthermore, the flowcharts can be run through backwards and hereby the specific values of the characteristics which lead to this implementation can be determined. The specific relation between components and flowchart diagrams and, thus between components and tasks, is shown in Table II.

Component	**Flowchart diagram**
Addressing scheme	Addressing scheme
Adaption	Ability to assign addresses dynamically Mode of communication Mode of prioritization
Fragmentation	Fragmentation
Transmission arbitration	Trigger condition

TABLE II: Relation between communication model components and flowchart diagrams.

IV. Running example

In this section we assess the operability of SOAcom approach by means of an example development. Our running example is a visual assistance application using the Controller Area Network. More specifically, our SOAcom process model is used to analyse the application and the network, to determine its shortcomings and specify the components of the communication model which overcomes them.

A. Example application

The example application used is a visual assistance for a car and trailer combination as shown in figure 7. This system helps the driver in the process of backing up a vehicle with a one-axle trailer connected. The main idea is to calculate the trajectories of the trailer and the overall vehicle, and to

overlay them on the picture of a rear view camera mounted on the trailer [6]. For this purpose, it is necessary to provide the steering angle and the angle between truck and trailer, the so-called bending angle. This assistance system is build up by seven different Services. Some of them are sensing Services, like the ones used to determine the current steering angle or the bending angle. Some others are Services which add value to these sensor signals, like the ones used to calculate the future path of the vehicle and its trailer, the so called trajectories. Finally the last class of Services outputs some information to the driver. In our example, this is done by a Service which displays the picture of the rear view camera overlayed by the trajectories. Figure 8 shows the interface of the system to the driver.

Fig. 7: Overview of the prototype.

Fig. 8: Human Machine Interface of the running example.

B. The Controller Area Network

The Controller Area Network protocol, which is used in this example, is a field bus developed by Robert Bosch GmbH in the 1980s for automotive applications. Its main benefits are: low cost, electric robustness, prioritised medium access arbitration mechanism, good real-time performance, as well as error- detection and containment mechanisms. Consequently, although it was primarily designed for automotive systems, it soon became widely popular in other types of distributed embedded control systems, like factory automation, robotics or medical equipment. CAN defines the lower two layers of the ISO/OSI communication reference model, that is, the physical layer and the data link layer. On the one hand, the physical layer defines the physical medium, as well as the bit synchronisation mechanisms. On the other hand, the data link layer defines how nodes can exchange data even in the presence of errors.

C. Applying the SOAcom process

In phase 1, we determine the requirements of the communication set up by the application. For this, the application previously introduced is analysed. This example application is composed of seven different Services, listed in Table III, alongside with their Service Class Address.

Name of Service	Service Class Address
SteeringAngleService	290
BendingAngleService	291
TrajTractiveUnitService	300
TrajTrailCombService	301
RearViewService	295
OutputVideoService	310
OverlayService	311

TABLE III: List of Services in the example application.

In a next step, the message with the biggest size within the application is identified, in this case, eight bytes. Besides that, from the sequence charts within the contracts of the seven Services, we distinguished event-triggered messages, as well as periodic messages which, as explained earlier, has also to be taken into account in the ongoing development process.

In phase 2, we use the questionnaire to characterise the network protocol. First, regarding the values collected from the transmission group, we observed that one of its main characteristics is its message orientation. Moreover, it has no central master, but is doing the bus arbitration one a multi-master base instead. As concerns the structure of the message, it has a relatively large identifier field, which offers to address more 536 million messages. Also, the identifier of each message is directly defining the priority of the message and therefore is a critical point in its communication model. Second, from the network capabilities group, we could identify the maximum payload offered by CAN, that is, eight bytes. A third point is the lack of mechanisms to add or remove nodes to the network at runtime. Finally, the attributes related to the physical capabilities and dependability groups are not interesting in the scope of this DAS and, thus, they are not taken into account.

Continuing with phase 3, we are using the information got through the two previous phases and the flowchart diagrams. In doing so, we are now able to identify the shortcomings of the CAN protocol in this specific scenario, and to determine the tasks to be undertaken to overcome them. The outcome of the flowchart diagrams is a list of four different tasks presented next.

1) **Set up the SOA-based addressing scheme**
   The communication model must allow to deliver SOA-based messages in CAN which, as explained, is a message-based network protocol. For this, each CAN message must identify the Service providing the content, as well as the command executed by this Service. As the addressing scheme has a relatively high number of addresses, the CAN identifier can be directly used for addressing the Services.
2) **Make the address scheme encoding consistent with the priority scheme used**
   In CAN, the message identifier directly defines the priority of the message. Thus, it has to be set up with care. The priority rules of CAN specify that the higher the number of the identifier, the lower the priority of the message. When setting up the addressing encoding this priority scheme, as well as the priority of the Service must be taken into account.
3) **Provide a communication protocol to assign addresses dynamically to the Services.**
   The communication model must allow to add or remove nodes at runtime which, in SOA-based driver assistance system, also involves the dynamic assignment of addresses to the Services. In CAN there is no such mechanism and, thus, it is necessary to provide one.
4) **Schedule periodic messages using event-driven scheduling**
   The communication model must ensure that each SOA-based message is scheduled using the most suitable scheduling policy, so time constraints are fulfilled. The CAN protocol is designed to be an event-based bus, which means that there is no in-build mechanism that allows to trigger messages on a periodic base. Unfortunately, as explained previously, the example application contains periodic messages. In this sense, it is necessary to ensure that, although these messages are scheduled using an event-driven policy, the periodicy is kept and no deadline is violated. Finally, note that is has been proven by Davis et al. in [7] that it is possible to constitute periodic messages using CAN.

Finally, in phase 4, we implement the components of the CAN communication model by adding the modules generated from the execution of the four tasks determined in phase 3. In the example application these modules are:

1) **Addressing Scheme** The two first tasks, ”Set up the SOA-based addressing scheme” and ”Make the addressing encoding consistent with the priority scheme used” lead to the modules which form the *Addressing Scheme* component. It basically manages the addressing of the Services and hereby controls the priority of each message.
2) **Ability to assign addresses dynamically** By executing the task ”Provide a communication protocol to assign addresses dynamically to the Services” the *Adaption* component is created. The main goal of this component is to provide mechanisms to manage dynamically the addresses of Services when they are added or removed from the system.
3) **Trigger condition** The task ”Schedule periodic messages using event-driven scheduling” leads to the *Transmission Arbitration* component. This component ensures that any Service is able to sent out periodic messages in an event-triggered network protocol such CAN.

The fourth component of the communication model is not touched in this running example as the biggest message within the application fits into a single CAN message.

D. Assessment of the running example

The SOAcom approach has proven to be a structured development process for constructing a communication model for CAN. Compared to the unguided development which led to the communication model described in [5], the process has now been much clearer and led to traceable requirements and well structured components.

In fact, SOAcom identified all the issues which have been addressed by the CAN Communication Model published earlier. Besides that, one feature, the possibility to send out messages periodically, is missing in that original model.

In order to confirm the correct operation of the implemented communication model we set up a prototype. This prototype is a CAN network running at a transmission rate of 500 Kbit/s using the 29 Bit Identifier. The seven nodes that run the Services are implemented on two different platforms. Four of the nodes were embedded boards with an Atmel ATmega88 microcontroller running at 18.432 MHz, as well as a Microchip MCP2515 CAN controller and a Philips PCA82C251 transceiver. Additionally, the software was implemented using the C programming language. The three remaining nodes were implemented as simulated nodes on a Laptop running the automotive network testing and simulation environment CANoe 7.6.84 (SP4) from Vector. The Laptop is connected to the CAN network using a CANcase XL bus interface also from Vector. We did some experiments using the described set up and the CAN communication model implemented using the SOAcom process model. As the implementation of the communication model showed that the adaption component is the most critical one, we focused on different scenarios to evaluate it’s capabilities. The used scenarios were reproducing real world events such as powering up all devices at the same time, which simulates turning on the ignition. The system took an average of 430ms until all seven nodes were integrated into the network. This relatively long period is mainly due to the simulated nodes. Running the same scenario with only the four embedded boards, it took only 50ms. The second scenario included the activation of the first four nodes at the beginning, and adding the remaining three ones some time later. This is an anticipation of the scenario when an already running car is connected to a trailer. As all the nodes activated later were embedded boards the system took a relatively short time of about 60ms to reconfigure itself. A third scenario extends scenario two. After starting up four nodes at the beginning and

adding three mores later, two of the nodes were disconnected and replaced by two other nodes offering the same Services as the original ones. This scenario determines whether the system is reusing the Identifiers of nodes that already left the network, as it is important to avoid having a number of dead Identifiers. The re-configuration of the system was again quit short being only 45ms long. All the experiments succeeded as in each scenario the system was able to integrate all nodes and get into a stable state were all Services could be reached through the Identifiers assigned by the algorithm.

V. Related Work

As stated before, there have been several other Middleware approaches in the domain of Distributed Embedded Systems in recent years. The eSOA project for example used the SOA paradigm to build systems that control smart buildings ([1]). However, the approach makes use of several traditional technologies of the Web Services domain such as the Web Service Description Language (WSDL). This fact, for example, requires to send XML files through the network to exchange information between Services which is a big drawback when a small network system like, for example, CAN or LIN is used. Another approach called the SIRENA project, which is explained in [2], has the same drawbacks of not being usable in low-performance networks. This is because it uses the Device Profile for Web Services standard, which is based upon IP-based communication. Lopez et al. present a Middleware concept in [8] which is targeting on the avionics domain. However, it also uses IP-based communication, which makes it not directly usable in today's cars. Two other approaches are directly aiming on automotive networks. Jahnich et al. present an approach in [9] which uses a Middleware to carry out load balancing in the events of ECU failures or overloads. As being designed for automotive infotainment systems low-performance ECUs are not taken into account. One of the most interesting projects on Middleware technologies in the automotive domain in recent years is the DySCAS project. It targeted at self-configuring automotive systems and designed an extensive architectural model. However our specific use case, namely the connection of devices distributed over a car and trailer combination, is explicitly excluded from the approach [10]. This is because of communication issues between car and trailer. We are committed that our project will be able to close this gap. There has been some work on high-level protocols for automotive networks, too. Many of this projects eventually became an industrial standard. Most of these approaches have been targeting on the CAN network. For example, the ISO TP standard defines a transport layer that allows to send frames of a maximum of 4095 bytes on CAN ([11]). The two competing standards CANopen ([12]) and DeviceNet ([13]) also aim on adding higher layers to CAN to offer some extended features. Unfortunately, all of these standards have been developed for being used in only one of the several different automotive networks. In our approach of implementing Service-based technologies, we want to use the whole spectrum of automotive network systems. Other high-level protocols like for example the XCP protocol ([14]) that enables engineers to calibrate ECUs within a car to offer interoperability throughout a higher number of network systems. Unfortunately they are very restricted in their purpose. Summing up one can say that none of the Middleware approaches or high-level protocols developed recently has all the features of SOAcom.

VI. Conclusions and future work

The Service-oriented Architecture paradigm provides high flexibility when designing distributed driver assistance systems, since it allows to hide all the complexity involving the connection and communication among logical components. However, these systems require a high grade of physical and logical adaptability, which is not provided by standard automotive network protocols. In this sense, we have constructed SOAcom, a development process to design communication models in the scope of automotive network protocols. We hereby extended the development process presented in [3] which originally did not include the development of the network components. We also took the step from a communication model exclusively designed for CAN, as explained in [5], towards a more flexible model theoretically capable of designing communication models for all automotive network systems. Specifically, this process allows us to extend the capabilities of an automotive network protocol, in order to adapt it to SOA-based DDASs.

SOAcom has proven to be suitable for identifying the shortcomings of automotive network protocols, in the scope of SOA-based DDASs and for specifying the components which should be added in order to overcome them. We have tested SOAcom using an example SOA-based DDAS application jointly with the Controller Area Network protocol. The experiments conducted showed that the implemented communication model was running stable.

In a next step, we plan to implement the SOAcom process as a software. It will automate some of the steps of SOAcom like for example the determination of the requirements set up by the application in phase 1. The usage of such a software will also help to keep all the information consistent. It is also planned to implement a communication model developed for LIN. Furthermore, we plan to add the generation of a FIBEX file that describes the network directly from SoaML model. The overall SOA approach for DDAS will be integrated and validated in a full scale prototype.

VII. Acknowledgement

Marco Wagner has been supported by a grant of the "Thomas Gessmann-Stiftung", Essen, Germany.

References

[1] A. Scholz, I. Gaponova, S. Sommer, A. Kemper, A. Knoll, C. Buckl, J. Heuer, and A. Schmitt, "SOA - Service Oriented Architectures adapted for embedded networks," in *2009 7th IEEE International Conference on Industrial Informatics*, no. 1. Ieee, Jun. 2009, pp. 599–605. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5195871

[2] H. Bohn, A. Bobek, and F. Golatowski, "SIRENA-Service Infrastructure for Real-time Embedded Networked Devices: A service oriented framework for different domains," in *International Conference on Mobile Communications and Learning Technologies*, 2006. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1628289

[3] M. Wagner, A. Meroth, and D. Zöbel, "Model-driven development of SOA-based Driver Assistance Systems," in *Proc. of the 4th Workshop on Adaptive and Reconfigurable Embedded Systems (APRES '12) in conjunction with the IEEE / ACM CPSWeek '12*. Beijing, China: IEEE, 2012, pp. 27–32. [Online]. Available: http://cps.kaist.ac.kr/apres2012/apres12.pdf

[4] OMG, "Service oriented architecture Modeling Language (SoaML) - Specification for the UML Profile and Metamodel for Services," Needham, MA, 2009.

[5] M. Wagner, A. Meroth, and D. Zöbel, "A CAN-based Communication Model for Service-Oriented Driver Assistance Systems," in *Proc. of the IEEE Vehicular Networking Conference (VNC) 2012*. Seoul, Korea: IEEE, 2012.

[6] U. Berg and D. Zöbel, "Visual Steering Assistance for Backing-Up Vehicles with One-axle Trailer," in *Vision in Vehicles 11*, Dublin, 2006.

[7] R. Davis, A. Burns, R. Bril, and J. Lukkien, "Controller Area Network (CAN) schedulability analysis: Refuted, revisited and revised," *Real-Time Systems*, vol. 35, no. 0, pp. 239–272, 2007. [Online]. Available: http://www.springerlink.com/index/8N32720737877071.pdf

[8] J. López, P. Royo, E. Pastor, C. Barrado, and E. Santamaria, "A Middleware Architecture for Unmanned Aircraft Avionics," in *Proceedings of the 2007 ACM/IFIP/USENIX international conference on Middleware companion*, 2007, p. 24.

[9] I. Jahnich, I. Podolski, and A. Rettberg, "Towards a Middleware Approach for a Self-configurable Automotive Embedded System," *Software Technologies for Embedded and Ubiquitous Systems*, pp. 55–65, 2008.

[10] V. Friesen, "Dynamically Self-Configuring Automotive Systems - Scenario and System Requirements," DySCAS PMC, Tech. Rep., 2007.

[11] International Organization for Standardization, "ISO 15765-2:2011 Road vehicles - Diagnostic communication over Controller Area Network (DoCAN) - Part 2: Transport protocol and network layer services," 2011. [Online]. Available: http://www.iso.org/iso/catalogue_detail.htm?csnumber=46045

[12] CiA, "CANopen," CiA, Tech. Rep. [Online]. Available: http://www.can-cia.org/index.php?id=canopen

[13] ——, "DeviceNet." [Online]. Available: http://www.can-cia.org/index.php?id=176

[14] K. LEMON, "Introduction to the Universal Measurement and Calibration Protocol XCP," *SAE transactions*, vol. 112, no. 7, pp. 482–488, 2003. [Online]. Available: http://cat.inist.fr/?aModele=afficheN&cpsidt=16125195