Re-configuration in SOA-based adaptive Driver Assistance Systems

Marco Wagner
Automotive Systems Engineering
Heilbronn University
Email: marco.wagner@hs-heilbronn.de

Dieter Zöbel
Faculty of Computer Science
University of Koblenz-Landau
Email: zoebel@uni-koblenz.de

Ansgar Meroth
Automotive Systems Engineering
Heilbronn University
Email: meroth@hs-heilbronn.de

***Abstract*—Unlike state-of-the-art Driver Assistance Systems (DAS) upcoming technologies may not base on a static software and system architecture. This is especially true for DAS for truck and trailer combinations were the components needed are distributed over both parts of the articulated vehicle. As a truck might be connected to different trailers changes are very common. In order to handle these changes at runtime the usage of Service-oriented Architecture (SOA) is a promising approach. Hereby the functionalities are encapsulated into Services which are re-composed whenever the system architecture changes. As the Services in such a system run on small embedded units traditional approaches to process the re-composition as known for example from the Web Services domain cannot be adopted directly. This paper discusses the different composition approaches published in recent years and suggests an algorithm suited for this class of systems.**

I. Introduction

State-of-the-art Driver Assistance Systems (DAS) like for example lane-keeping assistance are characterized by a static software and hardware architecture. This is because the number and topology of the Electronic Control Units (ECU) as well as the software modules forming the assistance do not change at runtime. In some upcoming DAS, however, the components of the system are distributed over more than one vehicle. This fact leads to a high degree of distribution, changing hardware and software configurations and the integration of heterogeneous components at runtime. We call this class of systems Distributed Driver Assistance Systems (DDAS). The focus of our work lies on systems that support the driver while using a truck and trailer combination since the functionalities needed are distributed on both parts of the articulated vehicle. In these DDAS the driver may disconnect a trailer to connect another one carrying different sensors or software functionalities. Whenever the combination changes the system is challenged to achieve the best assistance possible using the functionalities provided by the truck and the trailer.
In [1] we presented a novel approach able to handle the demands of DDAS basing on Service-oriented Architecture (SOA). In recent years this approach has been developed into a whole framework called "Service-Oriented Driver Assistance" (SODA). In SODA, the functionalities are encapsulated into Services. This allows distributing the components anywhere within the net of the networks of the vehicle. Furthermore, Services offer an abstract, well-defined interface to the outside world. Herewith, the heterogeneity of the different software functionalities is hidden. It also ensures compatibility between software components developed by different suppliers. In order to be able to react on changes at runtime SODA is re-configuring the system by automatically discovering Services currently available and re-composing them to generate the overall functionality. Furthermore we established SODAdev, a process model to develop such DDAS deploying SODA using the modeling language SoaML in [2]. This development process results in a detailed description of the Services and their interfaces. The description of each Service does not only include the functionalities offered but also the so called Requested Interfaces. A Requested Interface defines a Service that is needed to be able to fulfill its task. Hereby each Service has a local view of the types of Services surrounding itself. The Services within the network that are capable of offering the functionality requested by the interface are called available Services. These available Services can be detected by the SOA principle of Service Discovery. The Service Discovery process collects information about these available Services and ensures that they are offering the right functionality. As there might be more than one Service available offering the same functionality a optimization problem to find the best solution possible in the current scenario arises. This optimization problem has to be solved by a composition algorithm that decides for each Requested Interface which of the available Services is selected. This paper discusses the requirements of a composition algorithm in the domain of DDAS. These requirements are then used to analyze existing approaches form other SOA-based systems. Finally, the composition approach used in the SODA framework is presented.

II. Events of re-configuration in DDAS for truck and trailer combinations

The analysis of the events of system changes that can occur in a truck and trailer is needed to derive the requirements regarding the re-configuration procedure. We identified six different events:

1) Ignition on
2) Connecting a trailer at runtime
3) Disconnecting a trailer at runtime
4) Change of the type of assistance at runtime
5) Change of the quality parameters at runtime
6) Failure of a Service Instance at runtime

The first event is called Ignition on. Re-configuration is necessary in this event since the components available may have change during the vehicle was switched off.
In the second event a trailer is connected at runtime. Thereby

the system may be changed through adding additional Services. This new situation calls for a re-composition of the Services used by the application.
The counterpart of connecting a trailer is disconnecting one at runtime. This event is important for the re-configuration algorithm since SODA does not have an explicit sign-off mechanism. In this case SODA has to detect that no assistance is possible.
In the next event the type of assistance used is changed at runtime. As Uwe Berg and Dieter Zöbel have shown in [3] there is a multitude of possible human machine interfaces that could be used to assist the driver while backing up. These interfaces might for example use the visual, auditory or tactile modality. From a technical point of view each of these variants is an independent application defined by its own workflow. To guarantee the performance of the newly selected application a re-composition takes place before starting it.
The fifth event to be looked at is the change of one or more quality parameters at runtime. This could happen for example through external influences like the amount of light available when using a camera-based sensor. Such changes of the performance of a Service might lead to a bad performance of the overall application. This could be solved by re-configuring the system in such an event. However, these events might occur frequently. To avoid a permanent re-composition SODA is not tracking the performance of the Service Instances continuously but periodically. This compromise ensures long-term performance and stability.
The last event that might occur in a truck and trailer system is the failure of a Service Instance. In this case one of the following situations arises:

- Failure of a Service currently not used
- Failure of a Service currently used with alternatives available
- Failure of a Service currently used with no alternatives available

The first situation does not affect the performance of the application currently executed. Therefore it is not examined more closely. If a Service fails that is currently used, the execution of the present composition is no longer possible. Here, we have to distinguish two different situations. In the first one there are other instances of the needed functionality available. Here, the problem can be solved by carrying out a re-configuration. In the second case no other Service offering the needed functionality is available and the application has to be shut down.
These six events, their characteristics and impacts on the execution of the application have been used to develop the re-composition approach used within the SODA framework.

III. State-of-the-art in Service composition algorithms

Many composition algorithms for Services have been discussed in recent years (see e.g. [4]). We have analyzed 22 approaches divided in eight groups focussing on the special requirements of DDAS. The four requirements to the composition algorithm in the SODA framework are:

1) **Re-configuration at runtime** is needed to react to dynamic system changes.
2) **Determination of the optimal solution** is favored in order offer the best assistance possible.
3) **Executing a choreography-based approach** rather than an orchestration done by a central instance helps to avoid single points of failure.
4) **Ensure low resource consumption** in terms of computing power, memory and network load is needed as the algorithm is executed on rather small embedded devices.

Publication	Runtime composition	Optimal solution	Distributed manner	Low resource consumption	Approach
Amsden [5]	×	✓	✓	-	Design-time composition
Mayer et al. [6]	×	✓	×	-	
Mohabey et al. [7]	✓	×	×	×	Heuristics
Li et al. [8]	✓	×	×	-	
Yu et al. [9]	✓	×	×	×	
Yuan & Liu [10]	✓	×	✓	-	
Cardellini et al. [11]	✓	×	×	-	
Garcia Valls & Basanta Val [12]	✓	×	×	✓	
Liu et al. [13]	✓	×	×	×	Genetic Algorithm
Chang & Wu [14]	✓	×	×	×	
Wu et al. [15]	✓	×	×	×	
Qiqing et al. [16]	✓	×	×	×	
Tao et al. [17]	✓	×	×	×	
Grossmann et al. [18]	✓	×	×	✓	Local Optimization
Zeng & Benatallah (local opt.) [19]	✓	×	×	✓	
Zeng & Benatallah (global opt.) [19]	✓	✓	×	×	Integer Programming
Ardagna & Pernici [20]	✓	✓	×	×	
Wan et al. [21]	✓	×	×	×	Divide & Conquer
Aiello et al. [24]	✓	✓	×	×	Graph-based
Li et al. [25]	✓	✓	×	×	
Huang et al. [22]	✓	✓	×	✓	Dynamic Programming
Gao et al. [23]	✓	✓	×	✓	
SODA algorithm	✓	✓	✓	✓	

TABLE I: Comparison of different selection algorithms

The algorithms to be investigated are all listed in Table I. The first group within this list are the design-time composition approaches. Both of them are able to find an optimal solution. Nevertheless, the fact that they do not support runtime re-composition excludes these techniques from being using within DDAS.
A large group of composition algorithms makes use of heuristics to simplifiy the selcetion procedure. The general idea of these approaches is to find a configuration that matches a certain level of quality. All of these techniques are able to carry out runtime re-configuration. But only the one presented by Garcia Valls and Basanta Val is able to handle the resource constraints of embedded systems. Furthermore, only the approach of Yuan and Lin is executed in a distributed manner. Finally none of the the approaches can guarantee to find the optimal solution.
The next group of approaches uses genetic algorithms to select Services among each other. Unfortunately none of them fulfills any other requirement besides being executable at runtime. For this reason they are not considered to be used within the SODA framework.
Another group of composition approaches bases on local optimization. Instead of seeking for an optimal end-to-end

quality value these algorithms are basing on local selection. This technique does not guarantee an overall optimum (see e.g. [26]). It is the nature of local selection approaches that they use relatively simple algorithms that are characterized by low resource requirements. Nevertheless, all these approaches do not fulfill the requirement of composing the configuration with the best end-to-end performance.
Other approaches use the principles of integer programming to solve the selection problem. The basic idea of integer programming is that a set of variables, an objective function and a set of constraints is given and an algorithm tries to optimize the objective function by varying the values of the variables while enforcing the constraints. Both approaches analyzed are runtime executable and able to find the best solution possible. However, both of them are quite exhaustive and controlled by a central instance. In doing so both do not fit our profile of requirements.
Wan et al. present another technique for handling Service selection in [21]. The basic idea is to split up the overall application into smaller parts and optimize them separately. This divide and conquer approach is somehow similar to local optimization as it only guarantees the best selection locally instead of globally.
Another group of algorithms picks up classic graph-based techniques like for example the Dijkstra algorithm or a breadth first search. All these approaches manage to find the best solutions currently available. Nevertheless, they are quite exhausting. Furthermore they both need a supervising unit that overlooks the whole application in order to carry out the adaption procedure.
The last group of approaches to be discussed uses dynamic programming. This method generally solves complex problems by dividing it into smaller subproblems. These subproblems are then solved independently from one another and the subsolutions are combined in a way that the overall optimal solution is reached. Although being very close to the technique of divide and conquer, dynamic programming features two main benefits. Since the combination of the subsolutions is also subject of an optimization procedure end-to-end optimization can be carried out. Furthermore dynamic programming stores and re-uses the subsolutions to calculate them only once. Both approaches analyzed suffer from the fact that they use a central instance that overlooks and controls the whole system. Nevertheless, the principle of dynamic programming has high potential for being used within the SODA framework. It allows runtime re-configuration of the system. It is also able to find the composition offering the best end-to-end quality value. All this is done using a rather modest amount of computing power, memory and network load. Furthermore, as dynamic programming splits up the problem into smaller subproblems, the algorithms to solve them are numerous but rather simple. This fact meets the nature of DDAS as an aggregation of small, distributed, embedded devices. If the problem of being centrally controlled, that the example approaches are faced with, could be solved this technique would fulfill all demands set up by the application domain.

IV. The Service composition algorithm in SODA

As the discussion of state-of-the-art in Service composition algorithms has shown, the technique of dynamic programming is a promising approach. This is because it allows to build algorithms that find the best composition currently available. Due to its solution strategy that involves dividing the overall problem into smaller subproblems and the reuse of the subsolutions calculated it makes a contribution to create a resource-friendly algorithm. This is important since SODA has to carry out re-composition at runtime using embedded devices. The only open issue in the approaches of Huang et al. and Gao et al. id the usage of a central instance which creates a single point of failure.
For these reasons the composition algorithm designed for SODA picks up the solution strategy of dynamic programming and combines it with a distributed control mechanism. Looking at the structure of algorithms using dynamic programming one can identify three phases:

1) Partitioning of the problem into subproblems
2) Calculating the subsoltion for each subproblem
3) Combining the subsolutions to achieve the overall solution

During the first phase the overall graph containing all available Services is divided into smaller subgraphs which represent smaller parts of the problem. Hereby, the goal is to find a level of fragmentation that generates subproblems easy to solve. In SODA this is done by the explicit specification of cutting lines at design time. In order to achieve a complexity level close to the excellent values of the local composition algorithms the subproblems are limited to one Service and its direct neighbors in form of the candidates for its Requested Interfaces.
This small size of the subproblems simplifies the second step, which carries out the calculation of the subsolutions. It is reduced to selecting the best candidate subsolution offered at the moment. This can be done very easily by going through the possibilities given for a Requested Interface and choosing the one with the best QoS among these. Hereby another characteristic of dynamic programming is used. As soon as a Service calculated the solution for its subproblem it stores it for later use. In the case of a second request for the subsolution the saved result can be used without repeating the calculation.
The third step within a dynamic programming algorithm is to combine the subsolutions in such way that the best overall solution is found. In the algorithm used in the SODA framework this is automatically done due to the fact that the subproblems are solved consecutively. Hereby, each subsolution is calculated on the base of the minor subsolutions calculated before.

The re-configuration of an application can be illustrated using an example assistance system called Bending Angle Warning System (BAWS). This simple system monitors the two bending angles of a two-axle trailer and outputs a warning signal whenever one of them reaches a critical level. Therefore it needs sensor Services, one for each Bending angle, a Service that reads those values and compares them to some limit and a Service that is able to output a sound whenever such a limit is exceeded. One possible combination of available Services is presented in Figure 1. In this case for both of the needed functionalities *Sound Sink* and *Limit Check* two Service candidates are available. The procedure of re-composition is illustrated in the sequence chart given in Figure 2. It starts

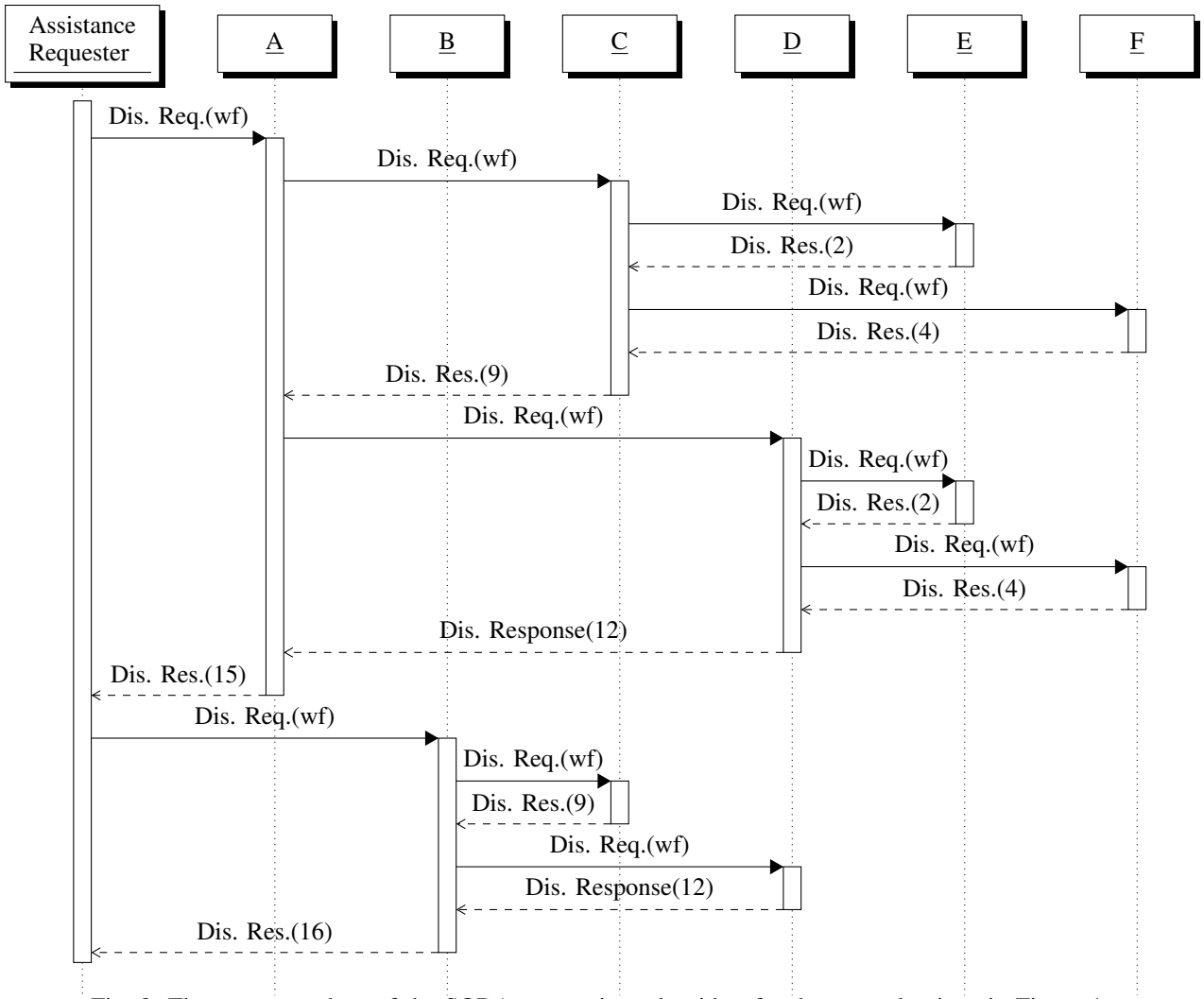


Fig. 2: The sequence chart of the SODA compositon algorithm for the example given in Figure 1

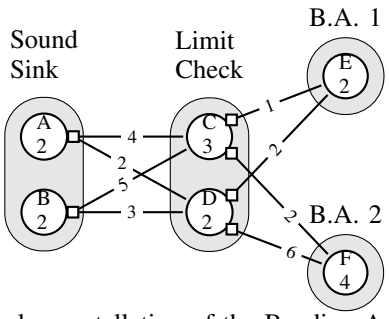


Fig. 1: A simple constellation of the Bending Angle Warning System

with a Discovery Request message including the weighting factors (wf) send from some unit that wants to execute the driving assistance (Assistance Requester). It is first received by Service A which is an implementation of *Sound Sink*. In order to determine which of the two candidates C and D, which both offer the *Limit Check* functionality, should be selected A first sends out a Discovery Request including the received weighting factors to instance C. This instance owns two Requested Interfaces, both matched by one instance each (E and F). To calculate the solution of its subproblem a Discovery Request is sent out by instance C to both instances. By sending the request the weighting factors are forwarded once again. Since E and F are implementations of the functionality *Bending Angle 1* and *Bending Angle 2* respectively both do not own any Requested Interfaces. Both do only use the weighting factors received to calculate their own QoS value and directly respond afterwards with the values 2 and 4 respectively. Instance C brings these responses together with the quality measures of the connections and its own QoS. This procedure solves the subproblem assigned to C with the result of 9 which is responded as subsolution to the requesting instance A. Since instance A identified a second candidate D it also requests D's subsolution. Just as instance C did a moment ago, instance D calls E and F to get their QoS. The results of this call are used to calculate the solution of the subproblem of D which equals 12 in this example. Since instance A has now all values available it is able to calculate and return its own subsolution. Taking into account A's own QoS, the received subsolutions and connection qualities the result of this computation is 15. In

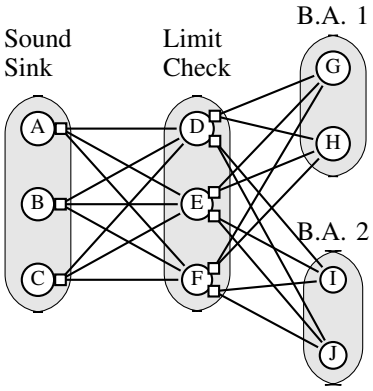
a next step, the second candidate for the Sound Sink, instance B receives the Discovery Request. As it has discovered two candidates, C and D to its Requested Interface it requests both to send their subsolutions. Both instances make now use of the re-use principle of dynamic programming. As they already solved their subproblems they directly respond to the request using these values. As a result instance B computes an overall solution of 16 and sends that to the initial requester. The Assistance Requester would now choose to use the application offered by Service Instance A with an overall QoS value of 15 over the one offered by B providing a quality measure of 16.

In order to rank the compexity of the re-configuration the overall number of Discovery Requests and thereby of executions of the Service composition algorithm is used. The reasons therefore are that it is easy to measure and it is a good indicator for the processor load caused by the adaption process. Furthermore it is independent from the actual implementation and hardware used and thereby suitable to compare the complexity of the algorithm on an abstract level.

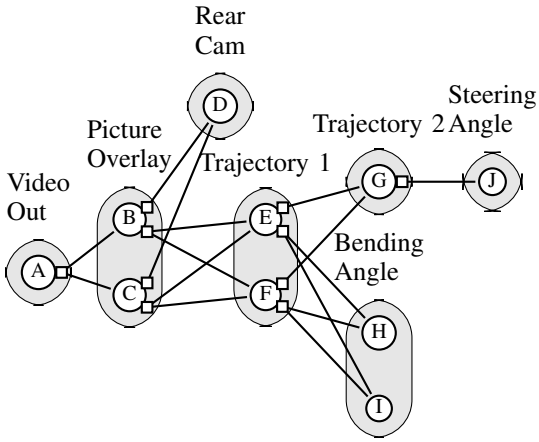
Analyzing the example given in Figure 1 the SODA algorithm sends out 10 Discovery Requests. Compared to an exhaustive depth-first-search which is also capable of finding the optimal composition (14 calls) this equals to a cutback of about 30%. Other examples confirm this benefit. Figure 3 (a) illustrates another possible constellation of the BAWS example. This time the functionalities *Sound Sink* and *Limit Check* are matched by three instances each. *Bending Angle 1* and *Bending Angle 2* are represented by two candidates in each case. While the exhaustive depth-first-search algorithm triggers 48 executions of the Service composition method, the one based on dynamic programming cuts them in half by executing the procedure only 24 times. The graph given in Figure 3 (b) represents a potential situation of a visual backing up assistance for a truck and a one axle trailer. In the given scenario some of the Services are matched by only one, some of them by two instances. Composing the application with the exhaustive depth-first algorithm would trigger 25 executions compared to only 16 with the SODA approach. The last example presented in Figure 3 (c) represents a potential constellation of instances of a visual backing up assistance for a combination of a vehicle and a two axle trailer. The given graph contains twelve functionalities and a total number of 15 instances. The reduction of Discovery Requests in this case is about 35% (46 with the exhaustive, 30 with the dynamic programming algorithm).

This last assistance system explained targeting on vehicles connected to a two-axle trailer has been implemented on a full-scale demonstrator. The basis this demonstrator is a Mercedes B-Class car connected to a two-axle trailer as pictured in Figure 4. The functionalities needed to build the assistance were implemented using the SODA framework. Some of these Services were executed on a 1 GHZ Intel Atom board running Lubuntu 12.04, others on a Raspberry PI platform running Raspbian OS. The smaller Services like for example the sensor units were implemented on embedded boards using either the ATMEL AT90CAN128 (8-bit architecture, 128KB memory) or the ATMEL ATmega88 (8-bit architecture, 8KB memory). In the constellation of instances given in Figure 3 (c) the execution of the Service composition algorithm took about 200ms.

(a) Another example of a potential constellation for BAWS



(b) An example for a constellation of Services available for DAS for one axle trailers



(c) An example for a DAS for two axle trailers

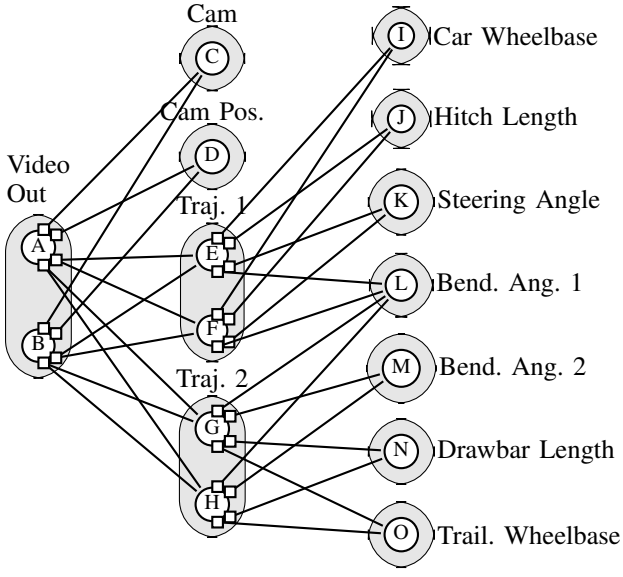


Fig. 3: Some examples for potential constellations of available Services

Fig. 4: Picture of the demonstrator vehicle.

V. Summary

In this paper an approach to react to runtime changes in driver assistance systems has been introduced. It uses the principles or Service-orientation and fulfills the requirements set up by the domain of embedded automotive systems at the same time. As the analysis has shown, the dynamic programming approach used allows to re-compose DDAS at runtime in a distributed manner while keeping the resource requirements for each participating unit low and finding the configuration with the best end-to-end performance. Furthermore, the approach has proved that it is executable on small embedded systems including 8-Bit micro controllers. The results of the experiments using the full-scale demonstrator have shown good performance by finding the best composition available in very short time.

References

[1] M. Wagner, D. Zöbel, and A. Meroth, "Towards an adaptive Software and System Architecture for Driver Assistance Systems Introducing a new approach to address dynamically changing automotive software systems," in *Proc. of the 4th IEEE International Conference on Computer Science and Information Technology (ICCSIT '2011)*, no. figure 2. Chengdu, China: IEEE, 2011, pp. 174–178.

[2] M. Wagner, A. Meroth, and D. Zöbel, "Developing self-adaptive automotive systems," *Design Automation for Embedded Systems*, Dec. 2013. [Online]. Available: http://link.springer.com/10.1007/s10617-013-9124-3

[3] U. Berg and D. Zöbel, "Gestaltung der Mensch-Maschine-Interaktion von Lenkas- sistenzsystemen zur Unterstützung der Rückwärtsfahrt von Fahrzeugen mit Anhänger," *Mechatronik 2007 - Innovative Produktentwicklung*, no. 1971, pp. 575–588, 2007.

[4] A. Strunk, "QoS-Aware Service Composition: A Survey," in *2010 Eighth IEEE European Conference on Web Services*, no. 1. Ieee, Dec. 2010, pp. 67–74.

[5] J. Amsden, "Modeling SOA : Part 4 . Service composition," pp. 1–16, 2007.

[6] P. Mayer, A. Schroeder, and N. Koch, "A Model-Driven Approach to Service Orchestration," in *2008 IEEE International Conference on Services Computing*. Ieee, Jul. 2008, pp. 533–536. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4578572

[7] M. Mohabey, Y. Narahari, S. Mallick, P. Suresh, and S. Subrahmanya, "A Combinatorial Procurement Auction for QoS-Aware Web Services Composition," in *2007 IEEE International Conference on Automation Science and Engineering*. Ieee, Sep. 2007, pp. 716–721.

[8] Y. Li, X. Zhang, Y. Yin, and J. Wu, "QoS-Driven Dynamic Reconfiguration of the SOA Based Software," in *2010 International Conference on Service Sciences*. Ieee, 2010, pp. 99–104.

[9] T. Yu, Y. Zhang, and K.-J. Lin, "Efficient algorithms for Web services selection with end-to-end QoS constraints," *ACM Transactions on the Web*, vol. 1, no. 1, pp. 6–es, May 2007.

[10] X. Yuan and X. Liu, "Heuristic algorithms for multi-constrained quality of service routing," *Proceedings IEEE INFOCOM 2001. Conference on Computer Communications. Twentieth Annual Joint Conference of the IEEE Computer and Communications Society (Cat. No.01CH37213)*, vol. 2, pp. 844–853, 2001.

[11] V. Cardellini, E. Casalicchio, V. Grassi, and F. Lo Presti, "Flow-Based Service Selection forWeb Service Composition Supporting Multiple QoS Classes," in *IEEE International Conference on Web Services (ICWS 2007)*, no. Icws. Ieee, Jul. 2007, pp. 743–750.

[12] M. García Valls and P. Basanta Val, "A real-time perspective of service composition: Key concepts and some contributions," *Journal of Systems Architecture*, vol. 59, no. 10, pp. 1414–1423, Nov. 2013.

[13] S. Liu, Y. Liu, N. Jing, G. Tang, and Y. Tang, "A dynamic web service selection strategy with QoS global optimization based on multi-objective genetic algorithm," *Lectue Notes in Computer Science*, no. 3795, pp. 84–89, 2005.

[14] W. Chang and C. Wu, "Optimizing the Dynamic Composition of Web Service Components," *Journal of Information Technology and Applications*, vol. 2, no. 4, pp. 227–234, 2008.

[15] M. Wu, X. Xiong, J. Ying, C. Jin, and C. Yu, "QoS-driven Global Optimization Approach for Large-scale Web Services Composition," *Journal of Computers*, vol. 6, no. 7, pp. 1452–1460, Jul. 2011.

[16] F. Qiqing, P. Xiaoming, L. Qinghua, and H. Yahui, "A Global QoS Optimizing Web Services Selection Algorithm Based on MOACO for Dynamic Web Service Composition," in *2009 International Forum on Information Technology and Applications*. Ieee, May 2009, pp. 37–42.

[17] F. Tao, Y. LaiLi, L. Xu, and L. Zhang, "FC-PACO-RM: a parallel method for service composition optimal-selection in cloud manufacturing system," *IEEE Transactions on Industrial Informatics*, vol. 9, no. 4, pp. 2023 – 2033, 2013.

[18] G. Grossmann, M. Schrefl, and M. Stumptner, "Model-driven framework for runtime adaptation of web service compositions," *Proceeding of the 6th international symposium on Software engineering for adaptive and self-managing systems - SEAMS '11*, p. 184, 2011.

[19] L. Zeng and B. Benatallah, "QoS-aware middleware for web services composition," *IEEE Transactions on Software Engineering*, vol. 30, no. 5, pp. 311–327, 2004.

[20] D. Ardagna and B. Pernici, "Adaptive Service Composition in Flexible Processes," *IEEE Transactions on Software Engineering*, vol. 33, no. 6, pp. 369–384, Jun. 2007.

[21] C. Wan, C. Ullrich, L. Chen, R. Huang, J. Luo, and Z. Shi, "On Solving QoS-Aware Service Selection Problem with Service Composition," in *2008 Seventh International Conference on Grid and Cooperative Computing*. Ieee, Oct. 2008, pp. 467–474.

[22] Z. Huang, W. Jiang, S. Hu, and Z. Liu, "Effective Pruning Algorithm for QoS-Aware Service Composition," in *2009 IEEE Conference on Commerce and Enterprise Computing*, no. i. Ieee, Jul. 2009, pp. 519–522.

[23] Y. Gao, J. Na, B. Zhang, L. Yang, and Q. Gong, "Optimal Web Services Selection Using Dynamic Programming," in *11th IEEE Symposium on Computers and Communications (ISCC'06)*. Ieee, 2006, pp. 365–370.

[24] M. Aiello, E. E. Khoury, A. Lazovik, and P. Ratelband, "Optimal QoS-Aware Web Service Composition," in *2009 IEEE Conference on Commerce and Enterprise Computing*. Ieee, Jul. 2009, pp. 491–494.

[25] Y. Li, J. Huai, T. Deng, H. Sun, H. Guo, and Z. Du, "QoS-aware Service Composition in Service Overlay Networks," in *IEEE International Conference on Web Services (ICWS 2007)*, no. Icws. Ieee, Jul. 2007, pp. 703–710.

[26] D. Ardagna and B. Pernici, "Global and local qos guarantee in web service selection," in *Business Process Management Workshops*, 2006, pp. 32–46.