Modeling and Analysis of Partitions on Functional Architectures using EAST-ADL*

Christoph Etzel and Bernhard Bauer

Institute of Computer Science, University of Augsburg, Germany
{christoph.etzel,bauer}@informatik.uni-augsburg.de

Abstract. The complexity in automotive systems engineering is increasing over the last decade. Autonomous driving and new comfort functions are some reasons for this growing complexity. With the introduction of multi-core processors in automotive system architectures, the shift from sequential to parallel thinking is more and more important in the different development phases. Based on the EAST-ADL, we present an approach to support the design process for distributed systems by using partitioning as an additional viewpoint on the architecture level. Therefore, we developed an extension to the EAST-ADL for partitioning and show automatic partitioning analysis on different architecture abstractions. These derived partitions can support system designers during the design process of functional architectures, by having a first insight how independent the functional components are structured from a data dependency viewpoint. This gives hints for the allocation of functions to hardware in later stages of the development process.

Keywords: System Architecture · Model-driven Systems Engineering · Automotive Systems Engineering.

1 Introduction

The trend of model-driven development to manage complexity during system development is still ongoing. Automotive systems are containing more and more hard- and software parts and forming huge distributed systems. In 2007 a BMW 7 contained 67 embedded devices providing 270 functions interacting with the user [19] and this further increased to 100 Electronic Control Units (ECUs) in premium vehicles around 2013 [14]. With the development of autonomous vehicles and its increased demand for additional sensors and data, the required computing power and the system complexity will further rise to assure a safe and comfort driving experience. Replacing single-core ECUs with multi-core systems is a possible way to lower the system complexity in vehicles [1]. This is an intensive discussed topic and first vehicles using multi-core architectures are on the road [15]. To utilize these newly created systems including the embedded

* This work was partially funded within the project ARAMiS II by the German Federal Ministry for Education and Research with the funding ID 01IS16025. The responsibility for the content remains with the authors.

multi-core technique, a "parallel thinking" is required already from the start of the system development. Having a well-designed abstract system model during early design steps, helps deriving it further down towards the concrete system. Starting the design process at the system level includes many different models and stakeholders and there exists no golden standard of methods and frameworks [7]. In the automotive domain many projects use a bottom-up approach [15]. Such an approach has the high probability to not fully understand the big picture of the system and therefore detailed analysis of the whole system are hard to achieve.

Model-based approaches can provide customized views for the current development situation to the stakeholders. This supports achieving their engineering and optimization tasks, by focusing on the level of intention. Different abstraction levels, starting with a high level view in the early stage to a more detailed technical view in a later stage, is a common way to manage development complexity. The architecture description language EAST-ADL supports such an approach by providing on the one side a step-wise refinement and on the other side dealing with cross-cutting issues. Allover, EAST-ADL allows modelling of requirements, features, functional components, timing constraints, safety constraints and other engineering related information.

The focus of our work is to support system designers during their architectural design decisions. We support the propagated "parallel thinking" through analyzing and parallelization of the logical and component-based architectures to achieve partitions, based on data dependencies. The partitioning shall give the system designer a starting point how and how well parts of the architecture can be distributed, e.g., without stressing the bus system (communication overhead). Sets of components in a partition may be executed independently from components in other partitions. To get a better understanding of the partitions, key figures for partitions are calculated. The partitioning analysis support the system designer to choose a suitable hardware architecture for system functions. The current version of EAST-ADL supports functional composition modeling, but has no modeling notations to express partitions. We introduce an extension of the EAST-ADL to store information about partitions and present partitioning algorithms to compute such partitions from an architectural model.

This paper is an extended version of our previous published work at MODELSWARD 2019 [6]. In this version, we added an additional partitioning algorithm (KaFFPa) and used it to evaluate the Single Entry Region (SER) analysis. First, introduce EAST-ADL and the used partitioning algorithms to perform our analysis. In Section 3 we present our approach and tooling. This is followed (Section 4) by the extension of the EAST-ADL meta-model with elements to handle partitioning information on in the model. Section 5 shows how the algorithms are applied to different target abstraction levels to find partitions and which key figures can be calculated to get a better understanding of a partition. The approach is evaluated in a case study showing a brake-by-wire system example (Section 6). The paper completes with the conclusion and giving an outlook for further research.

2 Preliminaries

This chapter introduces existing languages, methods and algorithms used in this paper.

2.1 EAST-ADL

EAST-ADL stands for 'Electronics Architecture and Software Technology - Architecture Description Language' and is maintained by the EAST-ADL Association [5]. Its focus is on capturing engineering information for automotive electronic system development. It offers elements to capture requirements, features, functions, software & hardware components and communication in a standardized form. The system's implementation is not part of the EAST-ADL, but the established AUTomotive Open System ARchitecture (AUTOSAR) standard [2] is used. While AUTOSAR's most abstract concept is the software architecture, the EAST-ADL provides means to model the system architecture and capture essential engineering information on this stage. [3]

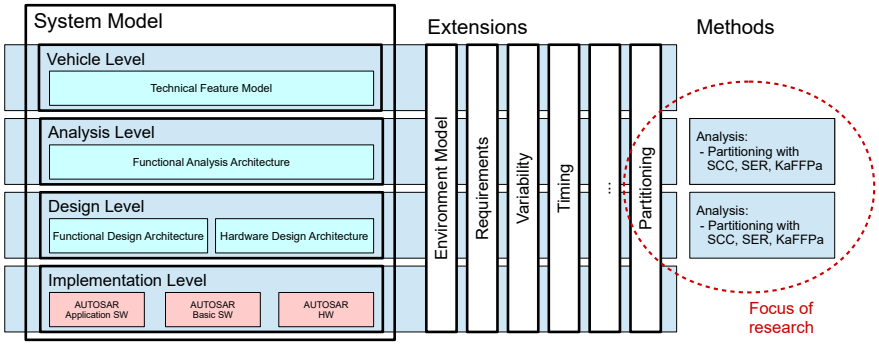

Fig. 1. EAST-ADL abstraction levels with the containing models and cross cutting extensions. On the right hand side the partitioning extension and methods provided in this research. [6]

The current release of the EAST-ADL2 [4] describes four abstraction levels to model the vehicle in different levels of detail (see left hand side of Figure 1). The **Vehicle Level** includes a Technical Feature Model of the electric and electronic system. It can be used as a software product line by using decomposition and variability to allow different feature configurations. The **Analysis Level** includes the Functional Analysis Architecture (FAA). On this level, the features of the Vehicle Level are realized by abstract functions. These functions are connected through devices (e.g., sensors or actuators) to the vehicle environment,

defining the systems boundary. The **Design Level** includes the Functional Design Architecture (FDA) and the Hardware Design Architecture (HDA). The FDA realizes the abstract function of the FAA with an implementation-oriented aspect. This includes software, middleware and hardware abstraction. The HDA captures physical resources and their connections and is used to allocate functions from the FDA to hardware entities. The **Implementation Level** is the connection to the AUTOSAR system model. The EAST-ADL is aligned with AUTOSAR and elements of the Design Level can be mapped to AUTOSAR entities [20]. These alignment and mapping capabilities enable traceability through the models during the whole development process.

Beside these abstraction levels, EAST-ADL is extended by several cross-cutting concern extensions, spanning over the layers of abstraction. Figure 1 shows the abstraction levels with their models horizontal and the extensions (cross-cutting concerns) are vertically aligned over all levels. Examples for these extensions in are Environment, Requirements, Variability and Timing. In this paper, we present an additional extension called *Partitioning* and methods applicable using this extension. Our contributions are marked with a red circle in Figure 1. Since the focus of our research is on the Functional Analysis Architecture (FAA) and the Functional Design Architecture (FDA) we provide a more detailed description of these two abstraction levels. Both architectures contain a component-based architecture model to capture the system information. The elements of the architectures are build up using a type/prototype concept similar to AUTOSAR. The basic elements are *FunctionType*s and *FunctionPrototype*. A *FunctionType* is an abstract function component description and gets instantiated by one or more *FunctionPrototype*s. A *FunctionType* contains *FunctionPort*s, which can be connected together using *FunctionConnector*s. Hierarchical architectures are realized by the specializations of *FunctionType* on the Analysis and Design Level (*FunctionAnalysisType* and *FunctionDesignType*), which can own parts in form of *FunctionAnalysisPrototype*s respectively *FunctionDesignPrototype*s. *FunctionConnector*s linking owned prototypes are called assembly connection, while a connection between a port of the type itself and a prototype is called delegation connection.

Besides the FDA, the Design Level includes the Hardware Design Architecture (HDA) to model the hardware system. The HDA defines the connectivity, capabilities and basic safety characteristics of technical architectures, e.g., the execution units (ECUs) including their cores and the communication paths. This information can be expressed using the following elements from the EAST-ADL modeling language (again using type/prototype concept): *HardwareComponentType* is the basic element for the specializations *Node*, *Sensor*, *Actuator* and *ElectricalCompmonent*. It defines *HardwarePort*s and *HardwarePin*s of the component and the embedded connections. An ECU is a *Node* element and its cores are contained *HardwareComponentPrototype*s of type *Node*. *HardwarePortConnector*s forming the bus system by connection ports and are capable to store information about the bus type and speed. *HardwarePort*s containing *Hard-*

*warePin*s and these pins are connected using *HardwareConnector*s, forming the bus system.

Since every abstraction level defined by the EAST-ADL includes a complete model of the whole system, a full tracebility between the levels can be realized. This is supported by linking elements of different abstraction levels together using the *Realization* relationship. E.g., a function on the FAA is realized by a group of functions on the FDA. Using realization links, this can be documented for later development stages or analyzes.

The EAST-ADL abstraction levels can be seen as equivalents to the phases of a system developing life cycle. For example, the abstraction levels can be used/mapped to phases of the V-Modell XT [30] as follows: the Vehicle Level, including its feature models, is part of the systems requirements analysis; the Analysis and Design Level are used in the system analysis, system architecture and system design phase; the Implementation Level belongs to the software architecture phase. Another example is the ATESST2 project, which released a methodology guideline for the development with EAST-ADL2 [27]. It defines a top-down development process and we embed our partitioning analysis into it, by making suggestions at which point of the process the partitioning step should be performed.

2.2 Partitioning Algorithms

In this section, partitioning algorithms used in this paper are explained. The strongly connected components algorithm is a classical algorithm from graph theory, while the extended single entry region algorithm is a more recent publication dedicated to AUTOSAR systems. Karlsruhe Fast Flow Partitioner (KaFPPa) algorithm is a state of the art multilevel graph partitioning algorithm.

Strongly Connected Components Strongly connected components (SCCs) are highly interconnected nodes in a directed graph. Tarjan [26] presents definitions for strong connectivity in graphs and an algorithm for computing the strongly connected components. A (sub-)graph is called strongly connected if there exists a path between each pair of nodes. A partition is formed by the set of strongly connected components.

Feedback loops, which are a very common pattern in automotive control systems, would form such a strongly connected component. Therefore, we see potential in applying the SCC on EAST-ADL architectures of the Analysis and Design Level.

Potts et al. [18] applied the SCC algorithm on system of systems (SoS) to support architectural decision making.

Single Entry Region The Single Entry Region (SER) analysis is a data dependency graph analysis for AUTOSAR system description models introduced by Kienberger et al. in [12] and further refined in [13], [11]. It is based on work

of [17], [9], [28] and [8]. The analysis tries to identify regions having a loose coupling to other parts of the system and therefore be somewhat isolated. A SER is described by the following three properties:

- The number of nodes is greater or equal to two.
- All input dependencies from nodes outside the SER are routed over a single "entry node".
- There is a path from the "entry node" to any other node inside the SER.

The SER analysis is performed on an AUTOSAR system description model, namely on the component-based architecture formed by *Runnable Entities* (AUTOSAR's atomic executable and schedulable units) and their data dependencies. From our point of view, the analysis can be used for every appropriate type of dependency in a graph. Since the components of an AUTOSAR system description model are derived from the FDA on EAST-ADL's Design Level and the FDA and FAA are component-based architectures with data dependencies, we adopt the SER analysis to EAST-ADL.

KaFFPa KaFFPa (Karlsruhe Fast Flow Partitioner) is a multilevel graph partitioning approach [22]. In a first step, it contracts the initial graph to create smaller graphs and does a first partitioning of this contracted graph. Then the contraction is reverted at each level and a local improvement is done to optimize the partitions on the coarser levels. The algorithm partitions the graph into a predetermined number of partitions, often denoted as k. The graph may contain weighted nodes and/or edges to describe the workload of a node or the communication of an edge, for example.

The KaFFPa algorithm is embedded in the KaHIP (Karlsruhe High Quality Partitioning) framework, which is public available [24]. Since the algorithm is reported to have very promising results in partitioning [21] and is easy available by using the public framework, we chose it for our approach.

3 Our Approach

Our approach has the goal to support the system designer during his architectural design decisions in order to have an architectural model that is well suited for further fine-grained development. Partitioning, in our context, is the process of grouping the system under development (SUD) into different parts without changing its functional component-based architecture. It is not intended to provide concrete mappings of functional components to hardware elements. The approach provides additional views on the SUD, depending on which criteria the partitioning is performed. We identified the FAA on the Analysis Level and the FDA on the Design Level as targets for our partitioning.

By using the extension mechanism of the EAST-ADL, the EAST-ADL system model remains unchanged and is only extended with new elements. The new extension makes use of already available EAST-ADL modeling concepts to

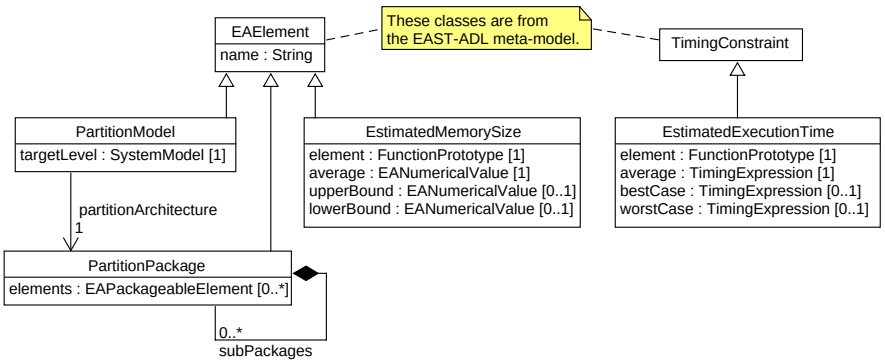

Fig. 2. The "Partitioning" meta-model extension. [6]

define its elements and use them for structuring system model elements. Figure 2 shows the extension called "Partitioning" and its meta-model elements to express partitions. Also elements to store additional information not yet available in the EAST-ADL meta model, but helpful for analyzing a SUD, are specified. Since there are only references to elements in the system model, the FAA on the Analysis Level or the FDA on the Design Level remain unchanged.

A short example shows how this works: For example, on the Analysis Level, the SUD is described by the FAA using functional devices and analysis functions. The functional devices are the connection to the environment; using sensors to get data from the environment and actuators to interact with it. Typically, chains of *AnalysisFunction*s link sensors to actuators, by performing calculations on the sensors data and react accordingly through the actuators. The connections between the devices and functions are modeled by ports to provide and receive data, which are linked together with function connectors. This component based description of the architecture together with additional data defined in the extensions is used to determine partitions, which can be persisted with the proposed *Partitioning* extension.

Tooling

AutoAnalyze The extension of EAST-ADL and the analyses are implemented in our tool AutoAnalyze. It is based on the Eclipse Modeling Framework[1], the Model Analysis Framework[2], EATOP[3] and Artop[4]. This allows us to load, edit

[1] Eclipse Modeling Framework (EMF) `https://www.eclipse.org/modeling/emf/`
[2] Model Analysis Framework - Data-flow based model analysis (MAF) `https://www.informatik.uni-augsburg.de/en/chairs/swt/ds/projects/mde/maf/`
[3] Eclipse EATOP Project `https://www.eclipse.org/eatop/`
[4] AUTOSAR Tool Platform (Artop) `https://www.artop.org/`

and save models defined with the EAST-ADL meta-model by using the EAXML format.

KaHIP KaHIP (Karlsruhe High Quality Partitioning) is a framework for doing graph partitioning with different algorithms [24]. It includes KaFFPa (Karlsruhe Fast Flow Partitioner), the multilevel graph partitioning algorithm we use in this paper and several other algorithms. KaHIP uses the Metis file format as explained in the Metis 4.0 user guide [23] [10]. AutoAnalyze is extended to export a graph in the Metis format, which then can be loaded into the KaHIP framework.

4 EAST-ADL Partitioning Extension

The focus of our approach is not limited to partition architectures on the different abstraction levels provided by EAST-ADL, but also to have a standardized way to retain and exchange the partition information. EAST-ADL structures the system model into different abstraction levels and it shall be possible to have multiple partition models per abstraction level. This is motivated by the idea that partitioning can be done with different goals to achieve different views on the model. These goals influence the selection and weight of properties going into the calculation, resulting in many possible partition views on the system. While the content of the architectures is diverse for every abstraction level, the meta-model elements shall be shared to support a common handling of partitioning in every use case. The newly introduced elements are derived from already specified elements in the EAST-ADL to be compatible with it. In the following definitions, most elements from the EAST-ADL meta-model can be identified by the prefix "*EA*", for example, *EAElement* and *EANumericalValue* are both from the EAST-ADL infrastructure package. The EAST-ADL meta-model contains some none prefixed elements, we will indicate if such an element is used. Besides having a good compatibility and extensibility using basic elements of the EAST-ADL, the partitioning extension fully benefits of already available concepts, e.g., connecting elements using EAST-ADL realization links to achieve a full traceability over the model.

The complete "Partitioning Extension" can be seen in Figure 2. On the left hand side are the meta-model elements to capture partitions and on the right hand side are elements to support the analysis of partitions. The root of the new partitioning elements is *PartitionModel*, pointing to the architectural model which is partitioned. It is derived from *EAElement*, an abstract metaclass of the EAST-ADL meta-model, defining an identifiable and named element. The *EAElement* has some attributes omitted in the figure, for example, the UUID attribute, as a global unique identifier, an expressive name and a comment attribute for additional descriptions. The partition model contains two associations the *targetLevel* and *partitionArchitecture*. The *targetLevel* is used to link the partition model to the level it partitions the architecture; i.e., the *AnalysisLevel* or *DesignLevel* object which are of the EAST-ADL meta-model super type *SystemModel*. The association *partitionArchitecture* points to the root package of

the partition architecture. Since the partitioning is done independently on every abstraction level, only elements that are part of the target level are allowed to be linked in the partition architecture and its nested packages.

Name PartitionModel
Description The *PartitionModel* is used to organize the partition architecture of an abstraction level.
Generalizations EAElement
Attributes No additional attributes.
Associations
targetLevel : SystemModel [1]
partitionArchitecture : PartitionPackage [1]
Constraints All (nested) referenced elements in the *partitionArchitecture* shall be part of the referenced *targetLevel*.
Semantics *PartitionModel* is the representation of a nested set of partitions for a specific system abstraction level.

*PartitionPackage*s are used to collect elements belonging to a partition, by using the *elements* association. The reason to define a new class *PartitionPackage* instead of using the already existing EAST-ADL meta-model element *EAPackage* is that an *EAPackage* uses a composition to aggregate the containing elements, while a *PartitionPackage* shall only provide an association to the elements in the architecture. Using the association a duplication of elements is avoided and changes to properties of elements in the architecture have not to be mirrored to the partition model. The *subPackages* association contains sub partitions and is realized using a composition. A *PartitionPackage* can contain multiple elements and packages to enable hierarchical partition architectures. To achieve a sound hierarchy, the association to elements in the target architecture shall be only once and as deep as possible in the *subPackages* structure.

Name PartitionPackage
Description The PartitionPackage is used to form partitions of elements.
Generalizations EAElement
Attributes No additional attributes.
Associations
elements : EAPackageableElement [0..*]
subPackages : PartitionPackage [0..*] {comp.}
Constraints No additional constraints
Semantics *PartitionPackage*s can be used to organize *EAPackageableElement*s that form a partition. The packages can be structured hierarchically, where each level may contain variable number of *EAPackageableElement*s and sub packages forming sub partitions.

The two elements *PartitionModel* and *PartitionPackage* enable a structural description of partitions. They link to elements in the architecture using associations and by this mechanism, no change of the architectures themselves is necessary.

The EAST-ADL includes already multiple extensions for different purposes. The timing extension, for example, defines modeling elements to specify timing constraints and other timing related information to enable timing analysis. Despite the existing extensions, there are still some elements missing from our point of view that would be helpful for analyzing partitions. To allow a more accurate partitioning of an architecture, two additional elements are defined, to store estimated values of memory footprints and execution time.

The element *EstimatedMemorySize* is used to capture the estimated memory footprint of a component. For example, this element can be used to balance partitions based on the memory size or to get an idea of the memory requirements of a partition. It has an association to an element in the system model and three values describing its estimated average memory size in bytes and optional upper/lower bound values to define a spectrum the memory size varies. The *element* and *average* associations are mandatory, otherwise no meaningful statement could be made.

Name EstimatedMemorySize
Description The estimated size of memory used by the function in bytes.
Generalizations EAElement
Attributes No additional attributes.
Associations
element : FunctionPrototype [1]
average : EANumericalValue [1]
upperBound : EANumericalValue [0..1]
lowerBound : EANumericalValue [0..1]
Constraints If set, the values shall comply to $lowerBound \leq average \leq upperBound$.
Semantics The *EstimatedMemorySize* stores the estimated or measured average memory size in bytes and optional an upper/lower bound.

The EAST-ADL timing extension describes an execution time constraint specifying the upper and lower bound run-time of an event. We introduce an *EstimatedExecutionTime* element, storing estimated or measured average execution time of a function and optionally a best and worst case value. It makes use of the already defined elements *TimingConstraint* and *TimingExpression* in the EAST-ADL timing package. *TimingExpression* allows the specification of a time including a unit and a time base. The *EstimatedExecutionTime* element is derived from the element *TimingConstraint*. The average, best and worst case elements are derived from *TimingExpression*. The *element* and *average* associations are mandatory, otherwise no meaningful statement could be made. In a SUD all defined values have be in line with already defined execution time constraints.

Name EstimatedExecutionTime
Description The estimated execution time of the function.
Generalizations TimingConstraint
Attributes No additional attributes.

Associations
element : FunctionPrototype [1]
average : TimingExpression [1]
bestCase : TimingExpression [0..1]
worstCase : TimingExpression [0..1]

Constraints If set, the values shall comply to $bestCase \leq average \leq worstCase$.

Semantics The *EstimatedExecutionTime* stores the estimated or measured values of the average execution time and optional a best/worst case value.

5 Partitioning Analysis

In this section, we describe how the SCC, SER and KaFFPa algorithms are used to automatic search for partitions on the architectures of the Analysis and Design Level. Partitions are formed by sets of functional components and analysis is done independently on the Analysis and Design Level. Besides using an algorithm to compute sets of partitions, an engineer can manually model partitions or modify the generated partitions afterwards.

5.1 Parameters for the Analysis

The main focus on our analysis are on supporting the engineer in understanding the architecture from the data dependency viewpoint. In our use cases we identified additional kinds of relevant clustering parameters: communication between functions and resource usage of functions. The communication is closely related to the data dependencies, since the data has to be transferred between the functions. Therefore, the amount of data exchanged between functions and the coupling of those can be taken into account. On the resource side execution time, execution frequency and memory consumption are values of interest. Using the newly introduced meta-model elements and already available elements in the EAST-ADL three parameter to consider these viewpoints: Data Flow Weight, Function Computational Time Weight and Function Memory Weight.

Data Flow Weight For the communication perspective we introduce a parameter to describe a weight for the data exchanged on a connection between two functions. The size of the transferred data can be calculated using the *EADatatype* specified for the connection and the repetition of the transfer, which can be derived of the function triggering (*FunctionTrigger*).

Function Computational Time Weight This parameter combines our introduced *EstimatedExecutionTime* element to estimate the computing time in conjunction with function triggering to get an idea how a processor is utilized by a function.

Function Memory Weight Using the newly specified *EstimatedMemorySize*, the memory footprint either of the binary or the resource usage during runtime including temporary memory can be calculated.

These parameters can either be used in partition search algorithms or to calculate key figures of a partition. E.g., partitions can be rated by their memory footprint summing up the Function Memory Weight of every component, or by their Function Computational Time Weight, if it is assumed that the set of functions in one partition is executed sequentially. These key figures are indicators for the system designer to judge about the architecture and possibly perform a refactoring.

5.2 EAST-ADL Analysis Level

The Analysis Level includes an abstract functional representation of the architecture captured in the FAA. This architecture is designed very early in the development process during the system analysis phase [27]. From a methodology point of view, the partitioning shall be placed in the development process after the task to specify the analysis function details. The result of partitioning analysis can then be used to further refine the architecture in an iterative way.

Before starting the analysis on the FAA, we have implemented multiple model pre-checks in our tool, such as if all directions of the ports and the binding to the function connectors are reasonable. For example, if two functions are connected via "IN" ports a warning is raised. The same applies to "OUT" ports. Additionally, it should be noted that a client-server connection in the model is interpreted as a bi-directional connection between the components.

SCC Analysis The first analysis implements the strongly connected component search. The directed graph consists of the analysis function prototypes as the vertices and the function connectors as the directed edges between the vertices. Since the SCC algorithm analyzes paths between the vertices, only the communication between the functions is taken into account to form partitions.

The results of the strongly connected component search is transferred into a partitioning model, where a set of strongly connected functions forms a partition. For every detected set with more than one component a *PartitionPackage* is created referring to the containing functions. An example with three graphs can be seen in Figure 3. The sets of strongly connected components enclosing more than one element are visualized with the same color. In the graph on the bottom of the figure is a single element "Prototype3" not colored (white background), since it forms a strongly connected set containing only itself and sets with just one element do not need a distinct color.

SER Analysis Another implemented algorithm is the Single Entry Region (SER) analysis, which was developed for AUTOSAR system description models [13]. A brief general description can be found in Section 2.2. We adapted the algorithm to fit to the EAST-ADL Analysis Level. For this purpose, every *AnalysisFunctionPrototype* contained in the FAA represents a node. The dependencies between the nodes are formed by the function connectors between the prototypes. The dependency weights are calculated by using the introduced Data

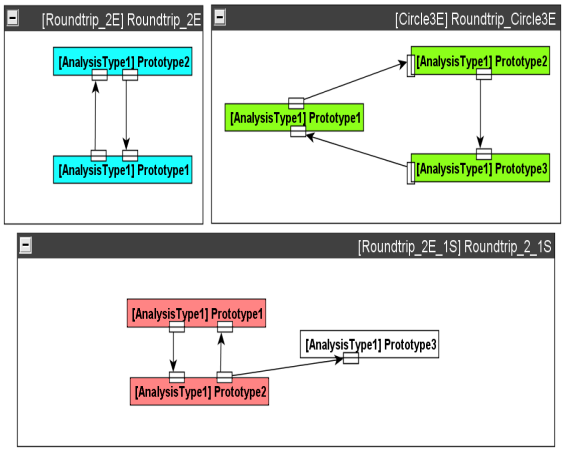

Fig. 3. Three examples of graphs with strongly connected components. [6]

Flow Weight parameter and summing it up for every connection between a pair of nodes. The output of the algorithm are regions containing sets of *AnalysisFunctionPrototype*s. This gets transferred into the partitioning model such that every calculated region forms one partition.

KaFFPa The KaHIP framework offers graph partitions algorithms with variable strategies. For our problem domain, we choose the KaFFPa algorithm and transfer the architectures to the METIS format, which serves as the input format. The *AnalysisFunctionPrototype*s of the FAA form the nodes of the graph. Optionally, the nodes can be weighted using the introduced parameters Function Computational Time Weight and Function Memory Weight introduced in Section 5.1. In contrast to the SCC and SER analyses, which use directed graphs, KaFFPa expects undirected graphs with only one edge between a pair of nodes. As a result, the direction information of the function connectors is ignored and every set of connections between two components becomes an edge. The weight of the edge is calculated by summing up the weights of all connections in this set. The weight itself is defined by the introduced Data Flow Weight (see Section 5.1), and therefore depends on the exchanged data type (to calculate the size of the data) and how often it is exchanged.

These information form a graph, which is complete to be partitioned using KaFFPa. To run KaFFPa, it needs a parameter k, defining the number of partitions the graph should be divided into. At present, this has to be provided by the engineer conducting the analysis.

The output of KaFFPa is a text file containing as many lines as nodes in the graph. Each of these lines represents a node and the value in the line represents the partition block ID. With the information how input graph was generated (knowing which *AnalysisFunctionPrototype* is which node), the output file is

transferred into the partitioning model such that all nodes with the same block ID form one partition.

5.3 EAST-ADL Design Level

The Design Level includes an implementation-oriented functional model of the architecture captured in the FDA. Looking into the design process, the FDA is specified during the design phase in parallel with the HDA [27]. This newly introduced partitioning step shall be placed in the development process after the task to specify the design details, but before the allocation the functions to the HDA. The result of partitioning analysis can then be used to further refine the architecture in an iterative way and as an input artifact to the HDA allocation task.

Since the elements of the FDA are very similar to the ones used for the analysis of the FAA on the Analysis Level, the SCC, SER and KaFFPa analysis are analogous to the analyses explained into detail in Section 5.2. The graphs are formed by function prototypes and function connectors. Even the pre-checks and the handling of client-server connections are identical.

By using a partition model of our analysis an engineer can allocate functions to elements of the HDA. Elements grouped into one partition by these two algorithms are candidates to be allocated on one node, because they communicate with each other. Placing them on one node or closely connected nodes can reduce the communication overhead. The HDA can also serve as a starting point to determine the parameter k for KaFFPa. k should be at least as high as the number of cores which are available to run components of the architecture on. KaFFPa includes an option to use a mapping algorithm, which performing a mapping which is communication and topology aware [25]. In Section 7 we discuss shortly, why this is not reasonably applicable for our approach in the automotive domain.

6 Case Study - Brake-by-Wire System Example

To evaluate the proposed approach a case study on an example architecture is carried out, showing the results of the SER and KaFFPa in detail. Since our approach tries to help an engineer understanding his/her model, we compare the different partitioning results between algorithms not by minimum cut values or other parameters, for example, but doing an expert review. It should be noted, that the SCC analysis would not find partitions with more than one component in this particular example and is therefore not discussed further. Nevertheless, we picked this model, because it illustrates the SER analysis, the differences to KaFFPa and the partition transition during the development process very clearly.

The "Brake-by-Wire for four-wheel vehicles" model is originally from the EAST-ADL Association and published on their website[5].

[5] Brake-by-Wire System II (`http://www.east-adl.info/Resources.html`) (accessed July 12, 2019)

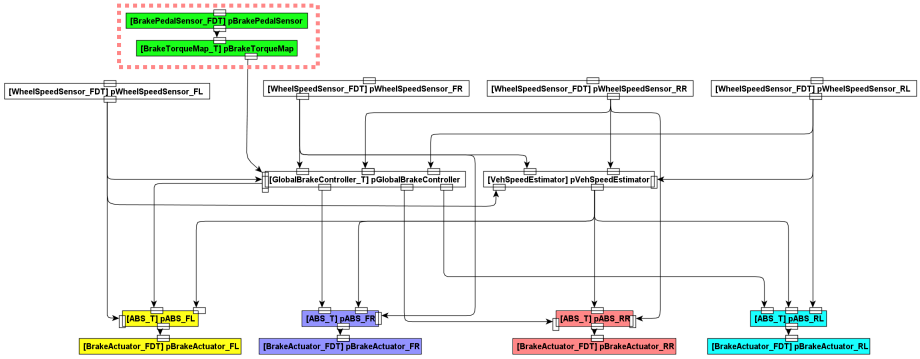

Fig. 4. Functional Analysis Architecture (FAA) of Brake-by-Wire Example. The colored elements are SER partitions. [6]

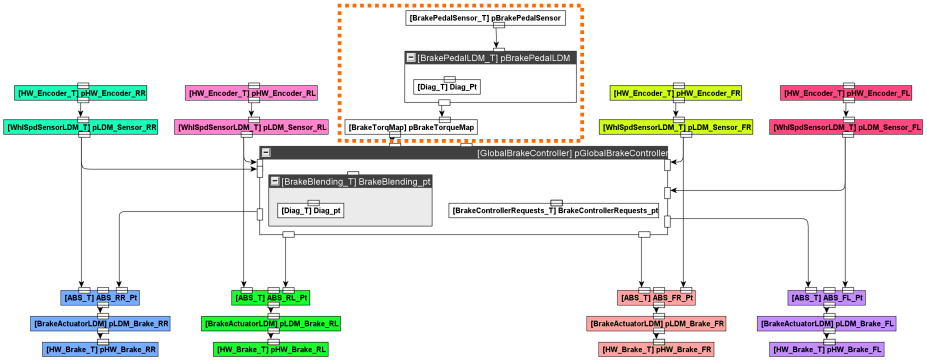

Fig. 5. Functional Design Architecture (FDA) of Brake-by-Wire Example. The colored elements are SER partitions. [6]

The FAA on the Analysis Level consists of 16 components and 26 connections between these (see Figure 4). The main function is a *pGlobalBrakeController*, which gets data from four wheel speed sensors, the vehicle speed and the requested brake force. The vehicle speed is calculated by the *pVehSpeedEstimator* getting data from the wheel speed sensors. The vehicle speed is provided to the *pGlobalBrakeController* and the four ABS controllers. The brake force is calculated by the *pBrakeTorqueMap* with data from the *pBrakePedalSensor*. The four ABS controllers are sending data to each brake actuator. The colored components in Figure 4 are partitions computed by the SER analysis. The upper green colored partition consists of two components (*pBrakePedalSensor* and *pBrakeTorqueMap*), the lower four partitions are each formed by the ABS and the brake actuator of one wheel. All three properties that a partition created by SER analysis must fulfill are very well recognizable. The partitions have more

than one element, all dependencies from outside into the partition pass through an entrance node and there is a path between every pair of nodes.

The SER analysis found five partitions and six single elements, so we set $k = 11$ for KaFFPa, since the six single elements are partitions of $size = 1$. Using this setting, we can compare the results of both algorithms. To understand the output of the analysis with KaFFPa, we give the numbering of the components as generated for the input graph: 1: *pBrakePedalSensor*, 2: *pBrakeTorqueMap*, 3: *pWheelSpeedSensor_FL*, 4: *pWheelSpeedSensor_FR*, 5: *pWheelSpeedSensor_RR*, 6: *pWheelSpeedSensor_RL*, 7: *pGlobalBrakeController*, 8: *pVehSpeedEstimator*, 9: *pABS_FL*, 10: *pBrakeActuator_FL*, 11: *pABS_FR*, 12: *pBrakeActuator_FR*, 13: *pABS_RR*, 14: *pBrakeActuator_RR*, 15: *pABS_RL*, 16: *pBrakeActuator_RL*. In Listing 1.1 the output of KaFFPa for the FAA can be seen. Each line represents a node from the input graph and contains the block ID of the node. Line 1 is the first node *pBrakePedalSensor*, associated with block/partition number 2. Line 2 *pBrakeTorqueMap*, block/partition number 3. ... It can be seen that the partitions on the bottom of Figure 4, the components 9-16, are identical generated by KaFFPa. A difference comes up for *pWheelSpeedSensor_RL* and *pVehSpeedEstimator* (lines 6 and 8), which are packed together in one partitions (block ID 7). The SER analysis puts *pBrakePedalSensor* and *pBrakeTorqueMap* (lines 1 and 2) together, which is from viewpoint of an expert review the more natural choice. Other values for k did not lead to a better evaluation result in the expert review. In our evaluation the best results to help the engineer to get a better understanding of the FAA is the SER analysis.

The design architecture (see Figure 5) is derived from the FAA. It contains 28 components and 27 connections (some components are for diagnoses, their connections to components outside the scope of this braking example have been omitted). For example, a wheel speed sensor from the functional analysis architecture is now more detailed by using two components. One is a hardware encoder providing the digital hardware signal and the other is a local device manager (LDM) encapsulating the hardware device specific parts. On the actuator side, a similar detailing is performed by using a LDM and a hardware function component for the realization. Two components for diagnose tasks are also embedded in the example. One is a diagnose component in the *pBrakePedalLDM* and the other one in the *pGlobalBrakeController*.

The partitions found using the SER algorithm are very similar to the ones on the analysis architecture. On the bottom, every ABS component together with a LDM and the actuator form a partition. Four new partitions are originated from the decomposition of the wheel speed sensors into hardware encoders and LDMs. A difference can be seen looking at the former partition of the *pBrakePedalSensor* and *pBrakeTorqueMap*, which is for a better recognition marked with a square of orange dots in both figures. Because a diagnose component (*Diag_Pt*), which provides data to other components not visible in this figure, is embedded in the *pBrakePedalLDM*, it is not marked as a potential partition on this level. An option to in- or excluding diagnose components in the analysis is part of our framework. Turning it off, the components *pBrakePedalSensor*, *pBrakePedal-*

Listing 1.1. Output partitioning FAA using KaFFPa with $k = 11$

```
1  2
2  3
3  4
4  0
5  6
6  7
7  4
8  7
9  5
10 5
11 1
12 1
13 10
14 10
15 8
16 8
```

Listing 1.2. Output partitioning FDA using KaFFPa with $k = 10$

```
1  7
2  7
3  1
4  0
5  0
6  9
7  9
8  6
9  6
10 2
11 2
12 1
13 0
14 4
15 4
16 9
17 8
18 8
19 6
20 5
21 5
22 2
23 3
24 3
```

Listing 1.3. Output partitioning FDA using KaFFPa with $k = 12$

```
1  1
2  1
3  1
4  8
5  8
6  0
7  2
8  5
9  5
10 4
11 4
12 3
13 7
14 7
15 6
16 2
17 11
18 11
19 5
20 9
21 9
22 3
23 10
24 10
```

LDM and *pBrakeTorqueMap* get together in one partition. Since there is no flag in the EAST-ADL meta-model to identify diagnose components, we are using a naming schema (the prefix "*Diag_*") to recognize these components.

For KaFFPa we discuss two outputs with $k = 10$ (number of partitions and solo components in SER analysis excluding the diagnose components) and $k = 12$ (including diagnose components). The model was simplified for the paper and the expert review by just using *pBrakePedalLDM* and *pGlobalBrakeController*, while not explicitly modeling the diagnose components and the *BrakeConrollerRequests_pt* for the KaFFPa input file. The lines to component mapping is as follows: 1: *pBrakePedalSensor*, 2: *pBrakePedalLDM*, 3: *pBrakeTorqueMap*, 4: *pHW_Encoder_RR*, 5: *pLDM_Sensor_RR*, 6: *pHW_Encoder_RL*, 7: *pLDM_Sensor_RL*, 8: *pHW_Encoder_FR*, 9: *pLDM_Sensor_FR*, 10: *pHW_Encoder_FL*, 11: *pLDM_Sensor_FL*, 12: *GlobalBrakeController*, 13: *ABS_RR_Pt*, 14: *pLDM_Brake_RR*, 15: *pHW_Brake_RR*, 16: *ABS_RL_Pt*, 17: *pLDM_Brake_RL*, 18: *pHW_Brake_RL*, 19: *ABS_FR_Pt*, 20: *pLDM_Brake_FR*, 21: *pHW_Brake_FR*, 22: *ABS_FL_Pt*, 23: *pLDM_Brake_FL*, 24: *pHW_Brake_FL*.

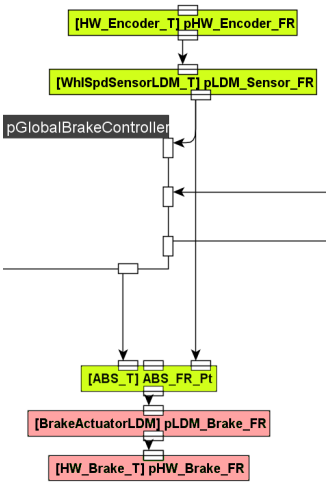

Fig. 6. Zoom into KaFFPa partitioning results ($k = 10$ and $k = 12$) of the FDA Brake-by-Wire Example.

In comparison to the SER analysis results, a noticeable difference in the KaFFPa results for $k = 10$ and $k = 12$ is that all encoder and sensor elements are in a partition together with the ABS component. Figure 6 shows an example of the partitions for the set of elements to control the front right brake. For Listing 1.2 these are the block IDs 6 and 5 and for Listing 1.3 the block IDs 5 and 9. This shows the difference to the SER characteristic, there all input edges have to be routed over a single entry node, while KaFFPa does partitioning on undirected graphs. From the expert review point of view, the SER results are more

useful to get an understanding which groups of components may be executed independently whereas KaFFPa tries minimizing the cutting. An additional difference in the partitioning for $k = 10$ is that the components *pBrakePedalSensor*, *pBrakePedalLDM* and *pBrakeTorqueMap* do not form one partition. While the first two form a partition (block ID 7), the *pBrakeTorqueMap* is placed together with the *GlobalBrakeController* (block ID 1). While this may be an optimal choice from the algorithms perspective, it would not be the natural one of an engineer. We evaluated the KaFFPa output for values of k from $[8, 16]$, but did not find more useful sets for our approach.

In summary, the results of the SER are preferable for identifying independent parts from the data flow perspective, while the KaFFPa partitions optimize the data throughput. An open point is how to determine the value of k to get results, which help the engineer understanding the architecture. Since the technical architecture, which the systems is deployed on, is in most cases heterogeneous, k equals number of ECUs or cores may not be a useful selection.

Using the analysis results, an engineer can check if the transition from the analysis architecture to the design architecture is sound (e.g., having a closer look, why one partition is now missing) and link the partitioned elements to elements of the HDA. This allocation is supported by the key figures, which can be calculated for the partitions.

7 Related Work

Using the KaHIP framework, the KaFFPa offers an option to perform a process mapping communication and topology aware process mapping developed by [25]. It was designed to address the mapping problem on modern supercomputer systems and several assumptions have been made. The hardware topology is hierarchically organized and every hierarchy level is identical. For example, every node in the topology has the same number of processors and every processor the same number of cores. This also applies to the distance value of the communication links inside each hierarchy level, which is assumed to be identical. On the other hand, automotive technical system architectures are very heterogeneous, containing different bus systems (high/low data transfer rates, non-/deterministic, ...) and ECUs (high/low performance, different architectures, ...). In addition, there are timing and safety requirements that require certain properties of individual hardware elements and thus constrain the mapping. Considering the differences, the KaFFPa process mapping is not a useful option for our problem domain, specifically for the HDA allocation task.

Marinescu et al. [16] propose a modeling extension for EAST-ADL and model analysis with the focus on resource-usage. The analysis is applied on the FDA using a priced timed automata to predict resource usage and optimizing resource utilization. In contrast to our approach, theirs is focusing on resource usage and allocation, while ours is proposing a general extension to describe partitions and algorithms focusing on the analysis of data dependencies. A mapping of parts of our extension to theirs is possible, e.g., *EstimatedMemorySize* (ours)

to *MemoryConstraint* (theirs). In the development process, their resource-usage analysis is placed after ours during the development of the Design Level elements.

Walker et al. [29] have developed a multi-objective optimization approach for EAST-ADL system architectures. Such an automation to rapidly explore architecture variants enables system designers to focus on the challenging parts. Their framework allows the connection of various analyses using an *Analysis Wrapper*. The analyses are performed independently and just provide their results to the optimization engine. This extension mechanism would make it possible to use our partitioning analysis in their framework. However, there has to be done further research how to derive and rate quantitative criteria for the optimizer from the partitioning models.

8 Conclusion and further Research

In this paper, we presented an approach to support system designers during the development process by doing partitioning on functional architectures. Therefore, we proposed an extension to the EAST-ADL meta-model to capture partitions without the need to alter the architecture. Additionally two elements are added to the extension to extend the analysis with additional information concerning the memory consumption and executing time of functions. These elements can be used to calculate key figure values of the partitions to get a better understanding of them. We presented three algorithms (SCC, SER and KaFFPa) to perform an automated analysis for partitions on the architectures of the Analysis and Design Level (FAA and FDA). These analyses are independent of the partitioning extension, if no persistence of the partitions is needed to perform further analysis. Moreover, we applied the new approach to a small case study from the EAST-ADL consortium and specifically done an expert review to compare the SER and KaFFPa results with regard to our goals.

The results concerning our approach are very promising and in the next steps we will evaluate it with additional scenarios. The best working approach for getting a better understanding of the architecture and its potential for parallelization, seems to the SER analysis. We will further refine the introduced analysis for partitioning of functional models on these levels of abstraction. We think the proposed approach is not limited to the EAST-ADL modeling language and can be transferred to similar concepts even outside the automotive domain. Examples for other languages are SysML[6] and AADL[7], both strongly influenced the EAST-ADL specification [3].

References

1. Arbeitskreis Multicore, BICCnet Innovationszirkel Embedded Systems: Relevanz eines Multicore-Ökosystems für künftige Embedded Systems: Positionspapier zur Bedeutung, Bestandsaufnahme und Potentialermittlung der

[6] Issued by the OMG, `http://www.omgsysml.org/`
[7] Issued by the SAE International, `http://www.aadl.info/`

Multicore-Technologie für den Industrie- und Forschungsstandort Deutschland. https://www.bicc-net.de/workspace/uploads/subfeatures/downloads/positionspapier_multicore_oekosys-1323952449.pdf (accessed July 15, 2019) (2011)
2. AUTOSAR: AUTOSAR website. https://www.autosar.org/ (accessed July 15, 2019) (2019)
3. Blom, H., De-Jiu, C., Kaijser, H., LÃ¶nn, H., Papadopoulos, Y., Reiser, M.O., Kolagari, R.T., Tucci, S.: EAST-ADL: An architecture description language for automotive software-intensive systems in the light of recent use and research. International Journal of System Dynamics Applications (IJSDA) **5**(3), 1–20 (2016)
4. EAST-ADL Association: EAST-ADL Domain Model Specification. Version V2.1.12 (2013)
5. EAST-ADL Association: EAST-ADL website. http://www.east-adl.info/ (accessed July 15, 2019) (2098)
6. Etzel, C., Bauer., B.: Extending EAST-ADL for modeling and analysis of partitions on functional architectures. In: Proceedings of the 7th International Conference on Model-Driven Engineering and Software Development - Volume 1: MODELSWARD,. pp. 169–178. INSTICC, SciTePress (2019). https://doi.org/10.5220/0007688301690178
7. Gajski, D.D., Abdi, S., Gerstlauer, A., Schirner, G.: Embedded system design: Modeling, synthesis and verification. Springer, Dordrecht and New York (2009)
8. Gotz, M., Roser, S., Lautenbacher, F., Bauer, B.: Token analysis of graph-oriented process models. In: 13th Enterprise Distributed Object Computing Conference Workshops. pp. 15–24 (Sept 2009). https://doi.org/10.1109/EDOCW.2009.5332020
9. Johnson, R., Pearson, D., Pingali, K.: Program structure tree: Computing control regions in linear time. In: Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI). pp. 171–185. ACM (1 1994)
10. Karypis, G., Kumar, V.: A fast and high quality multilevel scheme for partitioning irregular graphs. SIAM J. Sci. Comput. **20**(1), 359–392 (Dec 1998). https://doi.org/10.1137/S1064827595287997, http://dx.doi.org/10.1137/S1064827595287997
11. Kienberger, J.: Systematic and Methodical Analysis, Validation and Parallelization of Embedded Automotive Software for Multiple-IEU Platforms. PhD dissertation, University of Augsburg (2019)
12. Kienberger, J., Minnerup, P., Kuntz, S., Bauer, B.: Analysis and validation of AUTOSAR models. In: Proceedings of the 2nd International Conference on Model-Driven Engineering and Software Development. pp. 274–281. MODELSWARD 2014, SCITEPRESS - Science and Technology Publications, Lda, Portugal (2014). https://doi.org/10.5220/0004701002740281, http://dx.doi.org/10.5220/0004701002740281
13. Kienberger, J., Saad, C., Kuntz, S., Bauer, B.: Efficient parallelization of complex automotive systems. In: Balaji, P., Leung, K.C. (eds.) Proceedings of the 7th International Workshop on Programming Models and Applications for Multicores and Manycores. pp. 40–49. ACM (2016). https://doi.org/10.1145/2883404.2883421, http://doi.acm.org/10.1145/2883404.2883421
14. Lukasiewycz, M., Steinhorst, S., Andalam, S., Sagstetter, F., Waszecki, P., Wanli Chang, Kauer, M., Mundhenk, P., Shanker, S., Fahmy, S., Chakraborty, S.: System architecture and software design for electric vehicles. In: IEEE (ed.) Design Automation Conference (DAC), 2013 50th ACM / EDAC / IEEE. pp. 1–6 (2013)

15. Macher, G., Höller, A., Armengaud, E., Kreiner, C.: Automotive embedded software: Migration challenges to multi-core computing platforms. In: IEEE 13th International Conference on Industrial Informatics (INDIN). pp. 1386–1393 (July 2015). https://doi.org/10.1109/INDIN.2015.7281937
16. Marinescu, R., Enoiu, E.P.: Extending EAST-ADL for modeling and analysis of system's resource-usage. In: IEEE 36th Annual Computer Software and Applications Conference Workshops. pp. 532–537 (July 2012). https://doi.org/10.1109/COMPSACW.2012.99
17. Ottenstein, K.J., Ottenstein, L.M.: The program dependence graph in a software development environment. SIGPLAN Not. **19**(5), 177–184 (Apr 1984). https://doi.org/10.1145/390011.808263, http://doi.acm.org/10.1145/390011.808263
18. Potts, M., Sartor, P., Johnson, A., Bullock, S.: Hidden structures: using graph theory to explore complex system of systems architectures. International Conference on Complex Systems Design & Management. CSD & M (December 2017)
19. Pretschner, A., Broy, M., Kruger, I.H., Stauner, T.: Software engineering for automotive systems: A roadmap. In: Future of Software Engineering. pp. 55–71 (2007). https://doi.org/10.1109/FOSE.2007.22
20. Qureshi, T.N., Chen, D., Lönn, H., Törngren, M.: From EAST-ADL to AUTOSAR software architecture: A mapping scheme. In: Crnkovic, I., Gruhn, V., Book, M. (eds.) Software Architecture. pp. 328–335. Springer Berlin Heidelberg, Berlin, Heidelberg (2011)
21. Sanders, P., Schulz, C.: Engineering multilevel graph partitioning algorithms. CoRR **abs/1012.0006** (2010), http://arxiv.org/abs/1012.0006
22. Sanders, P., Schulz, C.: High quality graph partitioning. In: Bader, D.A., Meyerhenke, H., Sanders, P., Wagner, D. (eds.) Graph Partitioning and Graph Clustering, 10th DIMACS Implementation Challenge Workshop, Georgia Institute of Technology, Atlanta, GA, USA, February 13-14, 2012. Proceedings. Contemporary Mathematics, vol. 588, pp. 1–18. American Mathematical Society (2012). https://doi.org/10.1090/conm/588, http://www.ams.org/books/conm/588/11700
23. Sanders, P., Schulz, C.: Kahip v2.10 - karlsruhe high quality partitioning - user guide. CoRR **abs/1311.1714** (2019), http://arxiv.org/abs/1311.1714
24. Schulz, C.: KaHIP website. http://algo2.iti.kit.edu/kahip/ (accessed July 15, 2019) (2018)
25. Schulz, C., Träff, J.L.: Better process mapping and sparse quadratic assignment. In: Iliopoulos, C.S., Pissis, S.P., Puglisi, S.J., Raman, R. (eds.) 16th International Symposium on Experimental Algorithms (SEA 2017). Leibniz International Proceedings in Informatics (LIPIcs), vol. 75, pp. 4:1–4:15. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany (2017). https://doi.org/10.4230/LIPIcs.SEA.2017.4, http://drops.dagstuhl.de/opus/volltexte/2017/7603
26. Tarjan, R.: Depth-first search and linear graph algorithms. SIAM Journal on Computing **1**(2), 146–160 (1972). https://doi.org/10.1137/0201010
27. The ATESST2 Consortium: Methodology guideline when using EAST-ADL2. Deliverable D5.1.1 V1.1 (2010)
28. Tip, F.: A survey of program slicing techniques. Journal of Programming Languages **3**, 121–189 (1995)
29. Walker, M., Reiser, M.O., Tucci-Piergiovanni, S., Papadopoulos, Y., Lönn, H., Mraidha, C., Parker, D., Chen, D., Servat, D.: Automatic optimisation of system architectures using EAST-ADL. Journal of Systems and Software **86**(10), 2467–2487 (2013). https://doi.org/10.1016/j.jss.2013.04.001

30. Weit e.V.: V-Modell XT: Das deutsche Referenzmodell für Systementwicklungsprojekte Version 2.2 (2018)