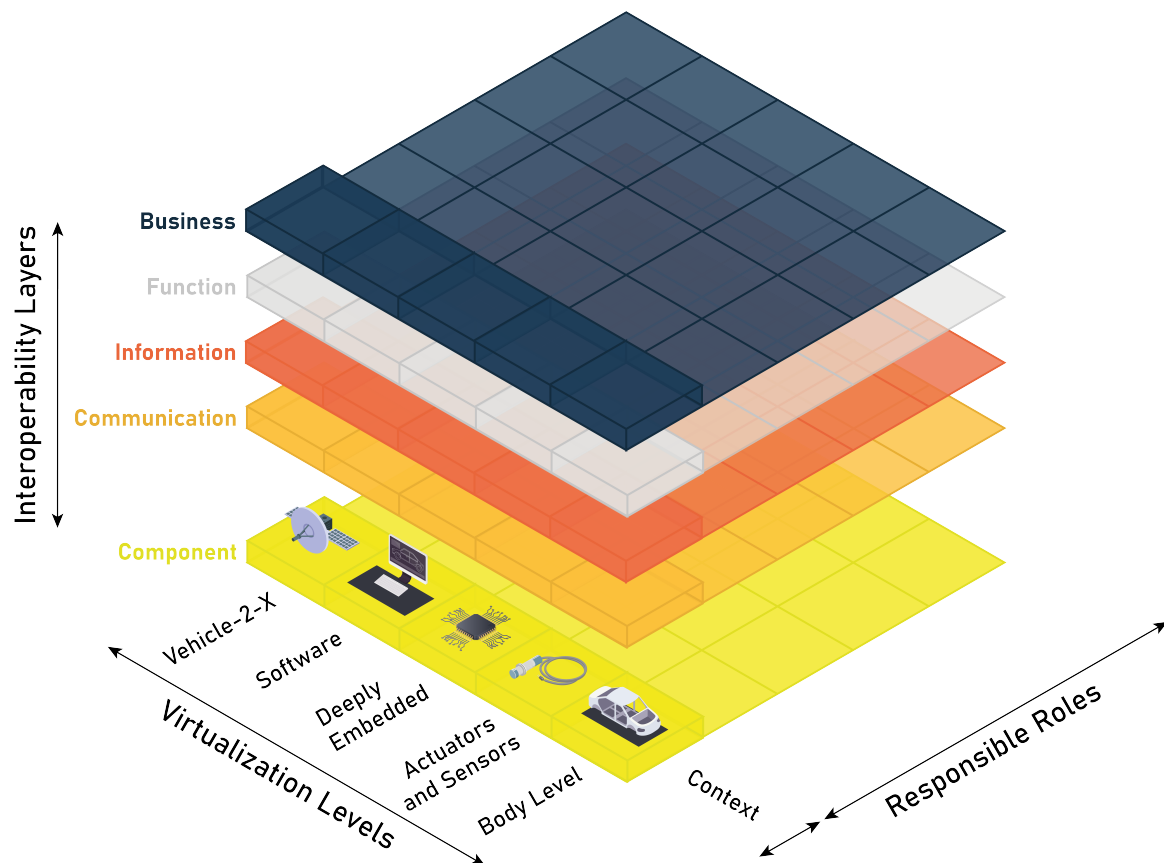


Application of the Automotive Reference Architecture Model (ARAM)



Author
DI Katharina Polanec

JOSEF RESSEL CENTRE FOR DEPENDABLE SYSTEM-OF-SYSTEMS
ENGINEERING

SALZBURG UNIVERSITY OF APPLIED SCIENCES

May 2024

Contents

1	Introduction	2
2	The Automotive Reference Architecture Model	3
2.1	The ARAM Plane	4
2.1.1	ARAM Virtualization Levels	4
2.1.2	ARAM Responsible Roles	5
2.2	ARAM Interoperability Layers	6
2.2.1	ARAM Business Layer	6
2.2.2	ARAM Function Layer	6
2.2.3	ARAM Information Layer	7
2.2.4	ARAM Communication Layer	7
2.2.5	ARAM Component Layer	7
3	ARAM Domain-Specific Language	8
4	Proposed Development Process	16
5	Installation of the ARAM Toolbox	18

1 Introduction

This document describes the concept behind as well as the usage of the *Automotive Reference Architecture Model (ARAM) Toolbox* for developing systems in the context of the automotive domain. The main purpose of this document is an introduction to the utilization of the ARAM toolbox. If you are interested in further information about the underlying methods and technologies, like systems engineering, modeling, or the handling of the involved modeling tool *Enterprise Architect* by *Sparx Systems* please consider the corresponding literature or feel free to contact us.

2 The Automotive Reference Architecture Model

The development of complex automotive systems requires a structured approach so that developers can keep track of everything that is involved. Moreover, it is important to achieve a clear separation of concerns and responsibilities. Developers, departments and companies must be aware of their own responsibilities as well as the responsibilities of entities they have to work with during the development of an automotive system. Not only the development approach is important but also the communication between all involved parties. In modern automotive systems, the number of involved participants is rather high than low. Hence, communication does not become easier, especially when different specialties meet each other and do not necessarily have the same jargon. This is where the ARAM framework (Figure 1) comes into play.

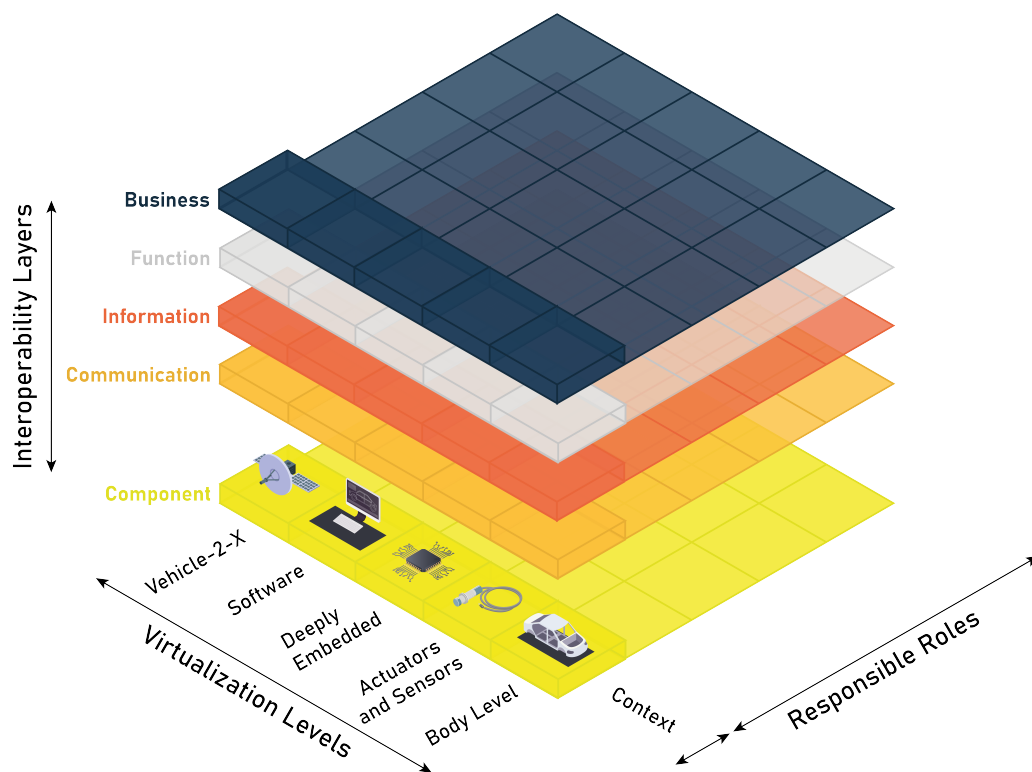


Figure 1: ARAM framework

The ARAM framework is a framework to support the initial conceptualization phase of a complex system in the automotive domain in the context of a domain-specific systems engineering (DSSE) approach. Consequently, the purpose of ARAM is mainly to ease the communication between stakeholders from different disciplines and to provide a guideline for a structured first draft of the system under development that can serve as a centralized basis for the subsequent development. The ARAM framework can be utilized and adapted to the needs of the respective system under development. It is suitable for developing rough concepts of an entire vehicle—being a very large and complex system—or for drafting a small subsystem within the vehicle. Regarding the possible involved stakeholders, ARAM can be used within one enterprise or, however, for more extensive projects involving multiple companies. To ensure this level of flexibility in a framework, ARAM consists of three axes, spanning a three-dimensional space.

The foundation of the framework is provided by the base plane, spanned between the two axes *virtualization levels* and *responsible roles*.

2.1 The ARAM Plane

The ARAM plane, as depicted in Figure 2, provides a two-dimensional space that serves the purpose of assigning responsibilities to involved entities as well as providing an overview of the level of virtualization of the system under development. This plane is available on each of the five interoperability layers (Section 2.2).

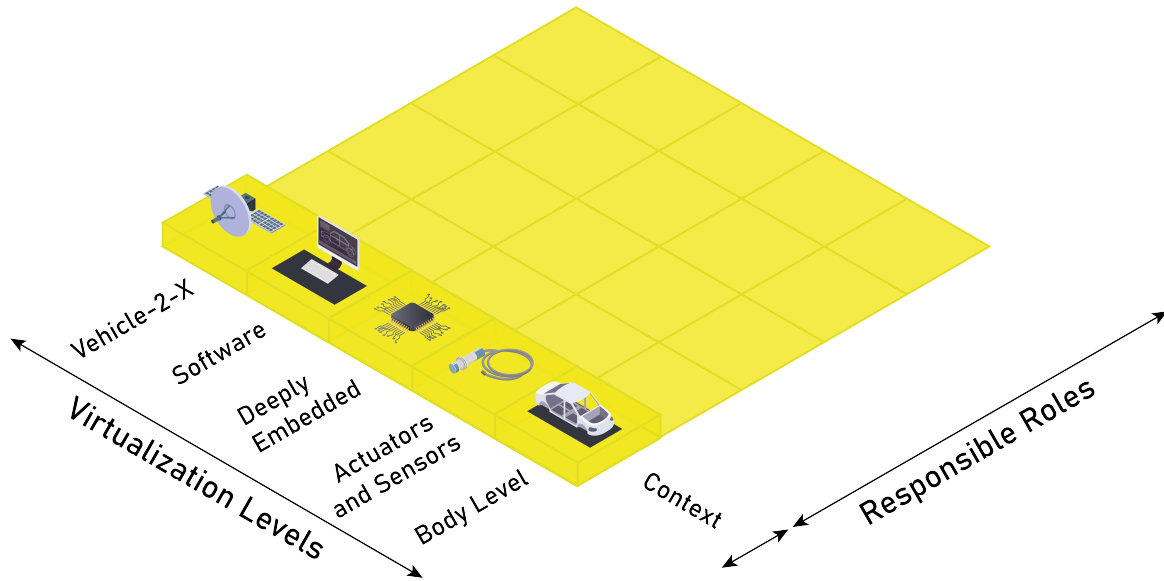


Figure 2: ARAM plane

2.1.1 ARAM Virtualization Levels

An automotive system can be comprised of mainly mechanical, physical components, rather software-intensive components, components that enlarge the system border by communicating with the environment, or even a combination of all these variations. The *Virtualization Levels* axis of the ARAM framework provides a classification for components according to their level of virtualization. By arranging the identified components in these sections, it is easily visible whether a system is a rather hardware-intense, software-intense, or even highly complex, intelligent system. Table 1 provides an overview of the possible classifications.

Body Level	The body-level section provides the lowest level of virtualization. Components in this section do not provide any type of digital intelligence on their own. These components are essential parts of the vehicle body. Examples of such components would be the wheels, the rear mirrors, or the chassis.
Actuators and Sensors	Components located on this level of virtualization are still physical components, but also provide some logical functionality. Actuators are components that are relevant for the execution of car functions. Often, actuators react to conditions that are detected by sensors. Examples of components on this level are parking sensors, rain sensors, or actuators executing orders from microprocessors in the vehicular system.
Deeply Embedded	Deeply embedded components provide a bridge between physical and virtual components. Essentially, they represent the ECUs responsible for the main functionality of the vehicle, hence an essential part of the vehicle's <i>brain</i> . Examples of such components are an airbag ECU, a brake-modulation ECU or a door-lock ECU.
Software	Components assigned to the software section are rather virtual than physical components that are mainly realized through software.
Vehicle-2-X	Vehicles cannot be considered isolated systems anymore. Most vehicles are already communicating with their surroundings or even other vehicles. This trend will even expand in the near to-distant future. Components responsible for such external communication are assigned to the vehicle-2-X section. Examples of these components are 5G connection modules, WiFi modules, or even emergency-call modules.

Table 1: ARAM virtualization levels

2.1.2 ARAM Responsible Roles

Since ARAM can be utilized in a very volatile context, it is important to identify all involved parties which can be almost everything: If the framework is used in the context of one department within a company, the involved parties could be individual people. If it is used in a greater context, throughout the whole company, those parties could be different involved departments. Utilizing ARAM in the most extensive context, a context including several enterprises, the involved parties could be those companies. In whichever context the framework is used, it is important to identify and document who is responsible for which parts of the system. This is the purpose of the *Responsible Roles* axis.

As depicted in Figure 2, the responsible-roles axis only has one label, the *context* label. This section is a dedicated area for components that are not the responsibility of any involved participant. Often, the systems under development interact with components that are not part of the system boundary or that already exist, hence are not part of the development process. Nevertheless, these components must be included in the model to provide a holistic representation of the system under development. The context section of the responsible-roles axis provides a dedicated space for such external components. Hence, a clear separation can be

made between components and subsystems that are the responsibility of an involved party and other components the system is just interacting with.

Apart from the context label, the axis labeling is empty. This enables the usage of ARAM in different contexts, including different types of roles. The purpose of this axis is that the users themselves can define which labels they need on this axis. If for instance, three departments are working on the system, the axis could have three labels, one for each department. After the first draft of the system has been finished and the components are assigned to the responsible roles, each department knows at first sight which components it is responsible for. The departments can then take this *slice* of the ARAM cuboid and develop the respective components whilst also being aware of interfaces towards components from other responsibilities.

Overall, the responsible roles axis provides a clear overview of components and their responsible roles as well as a distinct outline of interfaces towards other responsibilities which eases cross-discipline communication.

2.2 ARAM Interoperability Layers

The third axis of the ARAM framework is the *Interoperability Layers* axis. The framework contains five different interoperability layers providing different views of the respective system under development. Figure 3 gives an overview of these layers whilst providing a first insight into the ARAM domain-specific language (DSL) (Section 3). The concept of these interoperability layers is based on the *GridWise Architecture Council Interoperability Stack*. This stack also served as a basis for the development of the Smart Grid Architecture Model (SGAM) framework, a reference architecture model for the smart-grid domain. Since the automotive and smart-grid domains are closely related in current developments (e.g.: electric-vehicle charging), the ARAM framework is developed in a way that it also provides interoperability with the SGAM framework. This cross-domain interoperability will be outlined in a subsequent section of this document. In the following, the five interoperability layers of ARAM are described in detail.

2.2.1 ARAM Business Layer

The topmost layer of the framework is the *Business Layer*. This layer provides a business view of the automotive system by describing the involved business actors as well as their interest in the system under development. Hence, this layer serves as a starting point to identify entities that are involved in the development process of the system or who impose requirements that must be fulfilled by the finished system. The business layer is the place to collect user stories, business goals, and resulting high-level requirements that must be taken under consideration while further development.

2.2.2 ARAM Function Layer

Underneath the business layer lies the *Function Layer*, which provides a functional view of the system. Based on the high-level use cases identified on the business layer, on this layer functional groups and solution-neutral system components are identified. Thereby, the Functional Architecture for Systems (FAS) methodology is followed. Hence, on this layer functions and services that must be covered by the system to fulfill the envisioned business goals are identified.

Moreover, the information exchanged between functions, services, and components is specified on this layer.

2.2.3 ARAM Information Layer

The third layer is the *Information Layer* describing data that is exchanged between functions, services, or components. *Data* on this layer is a specialized type of information that is further described by its underlying data models. In general, this layer provides a view of the interoperable data exchange between entities of the system.

2.2.4 ARAM Communication Layer

The penultimate interoperability layer is the *Communication Layer* that gives an overview of the used protocols and mechanisms between different components of the automotive system. The protocols that can be modeled on this layer are based on industry standards for intra-vehicular communication.

2.2.5 ARAM Component Layer

Lastly, the *Component Layer* provides an overview of a high-level physical architecture alongside the disposition of the automotive components regarding the *virtualization levels* and *responsible roles*. The results from this layer can afterward be distributed to the respective responsible roles so that the automotive system can be further developed in detail.

3 ARAM Domain-Specific Language

The main feature of the ARAM toolbox is the DSL that is delivered with it. This DSL comprises modeling elements and relationships that aid the developers of an automotive system in expressing it using a model-based approach. Each interoperability layer of the framework uses different elements and relationships to express different views on the system. In the following, each element of the ARAM DSL is listed and explained.

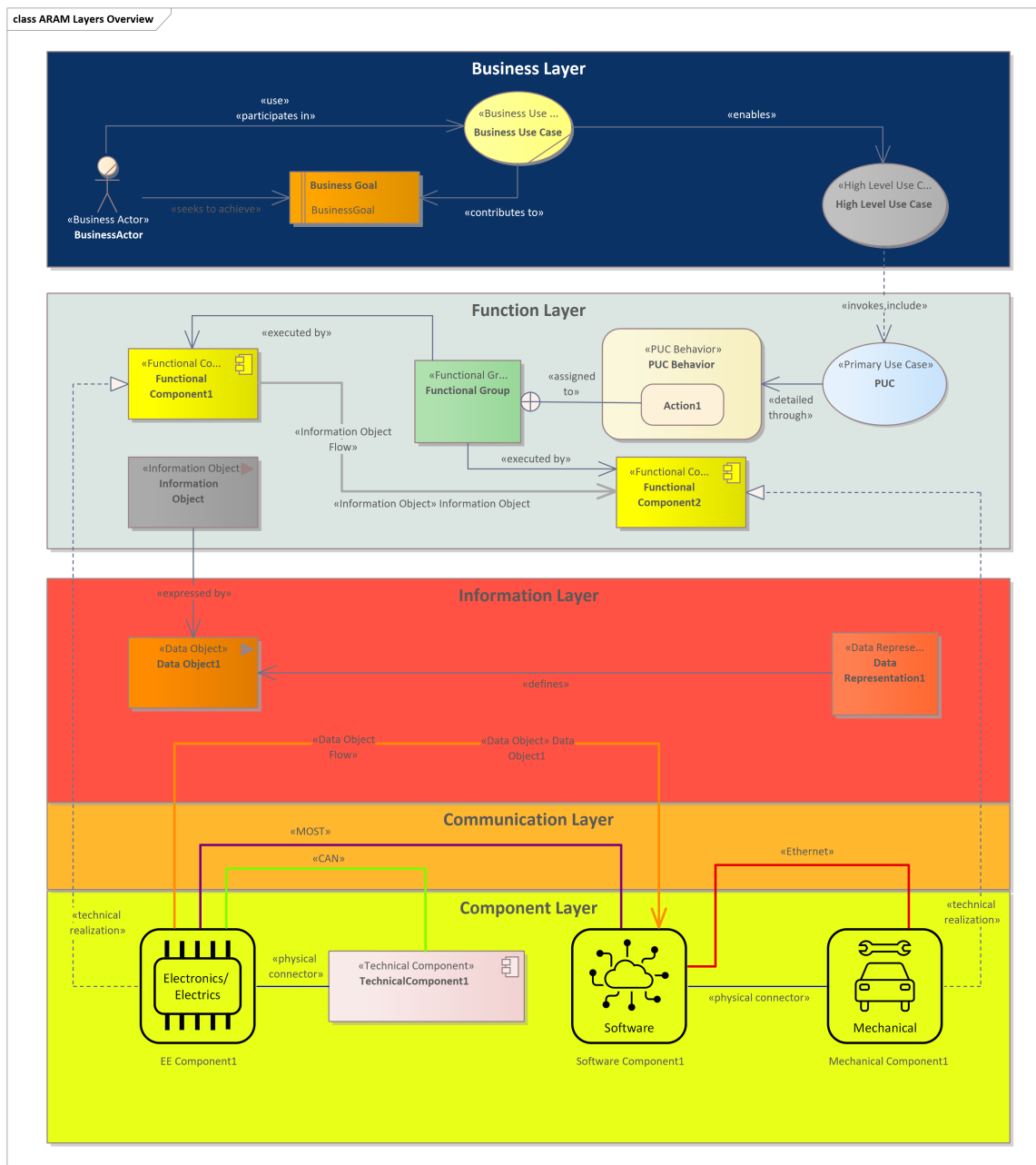


Figure 3: Overview of the ARAM DSL

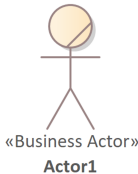
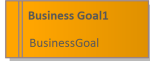
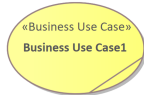
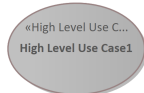
Business Layer Elements		
Business Actor	A business actor represents an entity fulfilling one role and being intrinsically linked to that role. It seeks to achieve its business goals by participating in one or more business use cases. A stakeholder can hold one or more roles relevant to the business use cases of interest and is therefore represented by one or more business actors.	
Business Goal	A business goal is a desired outcome that a business actor seeks to achieve, expressed in a way that is measurable and achievable, and that contributes to the overall success of the organization.	
Business Use Case	A business use case is a description of a sequence of actions that a business actor takes to achieve a business goal, expressed in a way that is independent of any specific system or technology.	
High-Level Use Case	A HLUC is a use case that describes a general capability, idea or concept independently from a specific technical realization like an architectural solution.	

Table 2: ARAM business layer elements





Business Layer Relationships		
Seeks to achieve	A <i>seeks to achieve</i> relationship is a relationship between a business actor and a business goal. It expresses, that a business actor has an intention and actively attempts to accomplish the respective business goal.	
Contributes to	A <i>contributes to</i> relationship is modeled between a business use case and a business goal. It expresses that the business use case is in accordance with, and purposefully facilitates the fulfillment of a business goal.	
Enables	An <i>enables</i> relationship is modeled between a business use case and a high-level use case. This relationship expresses that a high-level use case represents the system-level realization of the business utility prescribed by a business use case.	
Participates in	A <i>participates in</i> relationship indicates that a business actor is actively involved in carrying out a business use case to fulfill their business goal. Hence, it is a relationship between business actors and business use cases.	

Table 3: ARAM business layer relationships



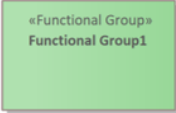
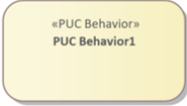
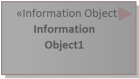
Function Layer Elements		
Functional Component	Functional components are system actors that can be either a subsystem, a person, a component or any other entity. On this level of the framework, the components are referred as <i>functional</i> because no technical solution is known yet. Functional components serve as a basis for the logical, solution-neutral architecture of the system.	
Primary Use Case	A primary use case expresses workflows that need to be realized by the system of interest to fulfill the envisioned functionality.	
Functional Group	A functional group is an element that describes a specific function that is executed by the system under development. It is used to cluster similar actions that must be executed to fulfill the system's primary use cases.	
PUC Behavior	A PUC behavior is an element that leads to a detailed description of each primary use case. This detailed description is a child diagram of the respective function and can be displayed using classical behavioral diagrams.	
Information Object	To fulfill the workflows of each primary use case, the functional components interchange information. The information object describes this information that is exchanged between functional components.	

Table 4: ARAM function layer elements


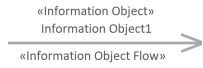


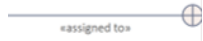
Function Layer Relationships		
Invokes	The <i>invokes</i> relationship is modeled from HLUC to PUC. It indicates that the HLUC invokes one or more PUCs to further define and realize its purpose from the system's perspective.	
Information Object Flow	The information object flow is a relationship on the function layer that indicates some sort of information exchange between two functional components. To further specify the information that is exchanged between the two components, the information object flow can transport information objects.	
Detailed Through	The <i>detailed-through</i> relationship is used between a PUC behavior and a primary use case. It expresses, that a behavior (activity diagram) refines the primary use case.	
Executed By	The <i>executed by</i> relationship connects a functional group to its executing functional component.	
Assigned To	The <i>assigned-to</i> relationship is used between an action element and a functional group. It indicates, that an action element needed for the activity diagram of a primary use case is assigned to a functional group.	

Table 5: ARAM function layer relationships



Information Layer Elements		
Data Object	A data object is a piece of data that is exchanged between components (from the component layer). More specifically, it specifies the kind of information object that is exchanged between the logical components from the function layer.	
Data Representation	The data representation element defines how a data object is represented. This can either be a standard or any other data model (could also be for instance something like a JSON representation).	

Table 6: ARAM information layer elements



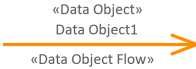
Information Layer Relationships		
Expressed By	The <i>expressed-by</i> relationship is modeled from information object to data object and from information object flow to data object flow. It means, that the information object (flow) is expressed in detail by the data object (flow).	
Definition	The <i>defines</i> relationship expresses that logical components are defined through primary use cases. Whilst defining the primary use cases of a system also the logical components which take part in the use-case process are revealed.	
Data Object Flow	The <i>data object flow</i> is a relationship on the information layer that indicates some sort of data exchange between two components from the component layer. To further specify the data that is exchanged between the two components, the data object flow can transport data objects.	

Table 7: ARAM information layer relationships

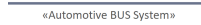
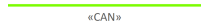
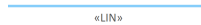
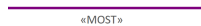


Communication Layer Relationships		
Automotive BUS System	This relationship can be used to model other connection types apart from the ones provided by the DSL on the communication layer.	
CAN	A "Controller Area Network" relationship is typically used for the connection of large numbers of ECUs.	
LIN	A "Local Interconnected Network" relationship is used for the integration of actuators and sensors into vehicle networks.	
MOST	A "Media Oriented System Transport" relationship is used for the integration of infotainment ECUs due to its special communication mechanisms and high data rates.	
Ethernet	The major advantage of ethernet relationships lies within inexpensive components and extremely high bandwidth.	
FlexRay	A FlexRay relationship enables deterministic time responses and redundancy. Therefore, it is used for security-critical applications.	

Table 8: ARAM communication layer relationships

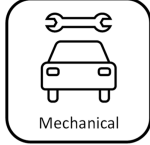
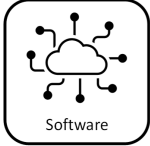
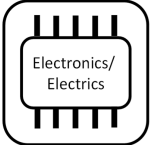
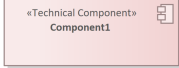
Component Layer Elements		
Mechanical Component	Mechanical components are elements used to model basic components of a vehicle like chassis, engine, and similar.	 Mechanical Component1
Software Component	Software components are elements used to model the main software units of the vehicular system.	 Software Component1
E/E Component	Electric/electronic components are elements that mainly control different systems within the vehicle, like ABS, ESP and similar.	 EE Component1
Technical Component	A technical component can be used to model a type of component apart from the ones provided by the DSL on the component layer.	

Table 9: ARAM component layer elements



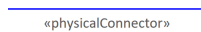
Component Layer Relationships		
Technical Realization	The <i>technical realization</i> is a relationship between functional component and technical component, expressing that the technical component is the technical realization of a functional component.	
Part Of	The <i>part-of</i> relationship can be used to express that multiple technical components are part of another (superordinate) technical component.	
Physical Connector	This relationship expresses that certain vehicular components are connected on a physical level.	

Table 10: ARAM component layer relationships

4 Proposed Development Process

Since this framework is designed to be adaptable depending on the automotive system, adherence to a specific modeling approach is not mandatory. However, general approaches as well as a best practice approach are suggested in the following.

Two main types of approaches can be differentiated: the greenfield and the brownfield approach. A completely new system—meaning it is not built on top or as an extension of an existing system—would be developed starting from the business layer all the way through to the component layer. This approach is referred to as the greenfield approach. The more common, however, is a brownfield approach: Some components, or even a whole subsystem, do already exist. In this case, it is recommended to model the component layer first, meaning to model the existing high-level architecture. Thereafter, the to-be-implemented components must be defined and outlined. This process would again start on the business layer. Hence, a brownfield approach combines existing and new parts. Figure 4 displays a best-practice approach for the development of new components and subsystems using the ARAM Toolbox.

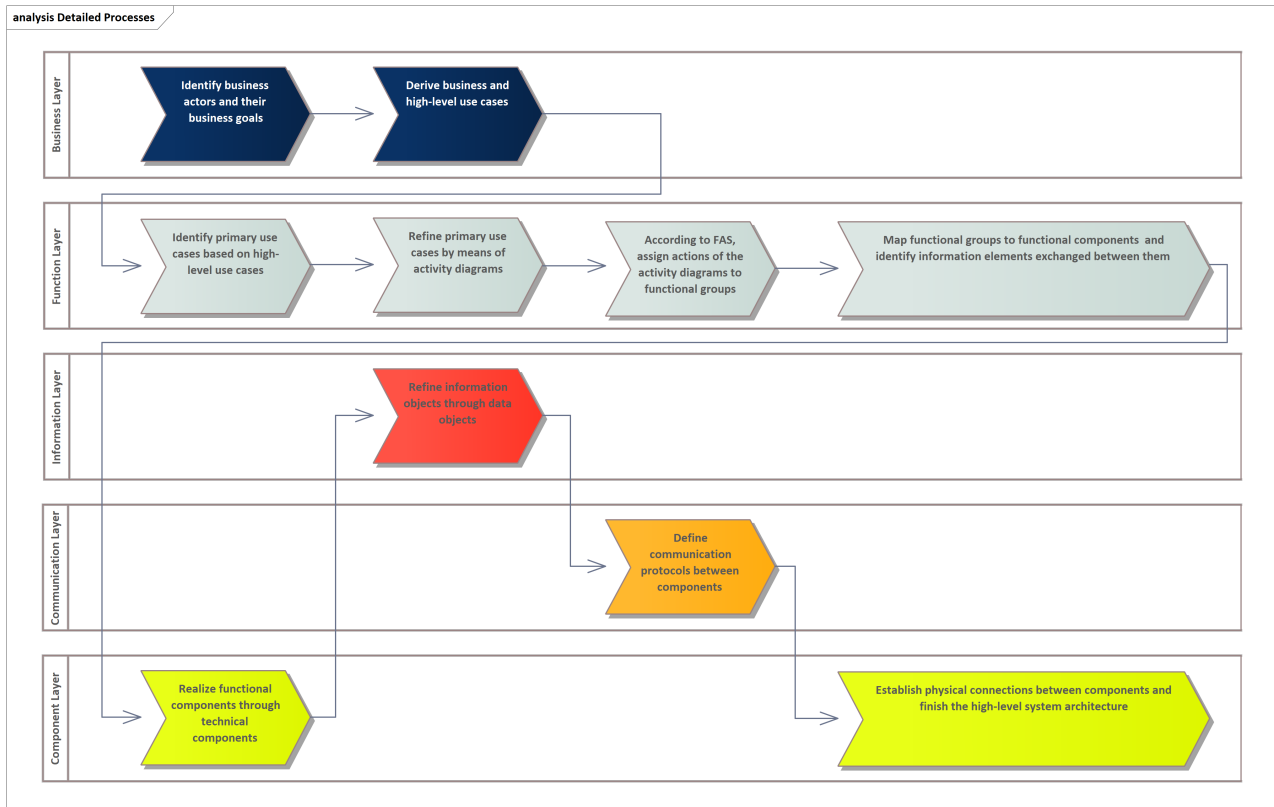


Figure 4: Best-practice development process

The process displayed in Figure 4 shows ten steps:

1. **Identify business actors and their business goals:** Before defining the actual system, it must be clearly defined who has an interest in the system under development. This is done in a preliminary context analysis and definition of interested parties. After the business actors have been identified, they are modeled on the business layer. Moreover,

each business actor has a business goal regarding the system under development. These goals are also modeled on the business layer and connected to the business actors.

2. **Derive business and high-level use cases:** The sequence of actions that a business actor takes to achieve its business goals is then specified through business use cases. These business use cases *invoke* high-level use cases that further describe a general capability, idea or concept independently of a specific technical realization.
3. **Identify primary use cases based on high-level use cases:** The connection between the business and function layer is established through an *includes* connection between the high-level use case and primary use cases. The latter describes workflows that must be executed from the system's perspective to fulfill the envisioned functionality.
4. **Refine primary use cases by means of activity diagrams:** In order to express primary use cases in detail, they are refined using activity diagrams. These diagrams express a process that is done to fulfill the respective primary use cases, by defining actions in their chronological order.
5. **According to FAS, assign actions of the activity diagrams to functional groups:** For this step, the FAS method is utilized. Based on the activity diagrams defined in the previous step, all actions are gathered and clustered into similar functionalities. These functionalities are then expressed through functional groups. A functional group thereby is a specific functionality that needs to be executed by system components to contribute to the overall functionality of the entire system.
6. **Map functional groups to functional components and identify information exchanged between them:** After the functional groups have been identified, the respective functional components, that are executing these functions, have to be defined. In order to fulfill their functionality, the functional elements also have to interact with each other by exchanging information. The type of exchanged information is documented as information objects in this step.
7. **Realize functional components through technical components:** After the upper two layers have been specified, the functional components are realized through technical (solution-specific) components. This mapping is done on the component layer.
8. **Refine information objects through data objects:** After specifying the technical components, their data exchange can be defined. This is done on the information layer based on the already identified information objects. Hence, if logical components A and B exchange information object I on the function layer, their technical realizations (technical components) a and b exchange data d which is a specialized information object I .
9. **Define communication protocols between components:** On the communication layer, communication protocols are specified between the technical components. Hence, a view of protocol exchanges is modeled on this layer.
10. **Establish physical connections between components and finish the high-level system architecture:** After all other layers have been developed, it is known which technical components are interacting with each other. These connections are finally modeled on the component layer resulting in a high-level system architecture.

This suggested best-practice approach is an iterative approach. It is always possible to refine any of the layers and work through the framework again.

5 Installation of the ARAM Toolbox

The *ARAMToolbox.Setup.0.0.msi* file will be downloaded to the specified location. After the download has finished, open the downloaded file. This will open the *ARAM Toolbox Setup Wizard*, as depicted in Figure 5.

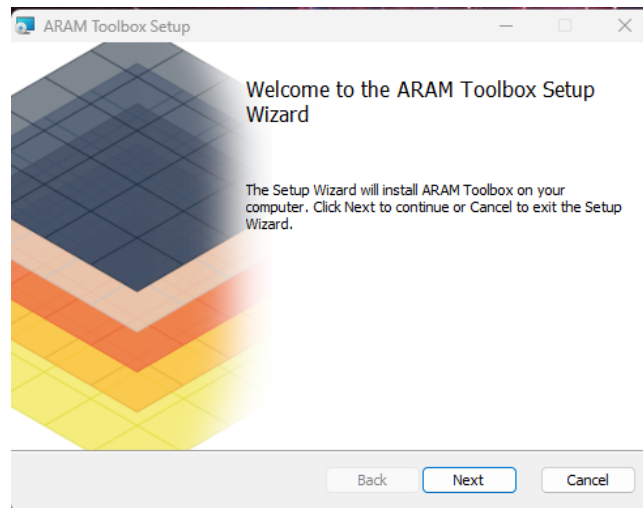


Figure 5: Installation wizard

Click *Next* and accept the license agreement, then click *Next* again. Specify the installation location for the ARAM Toolbox or leave the default location ("C:\ProgramFiles(x86)\ARAM Toolbox\"). Click *Next* and *Install*. Finally, after the installation is completed, click *Finish* (Figure 6).

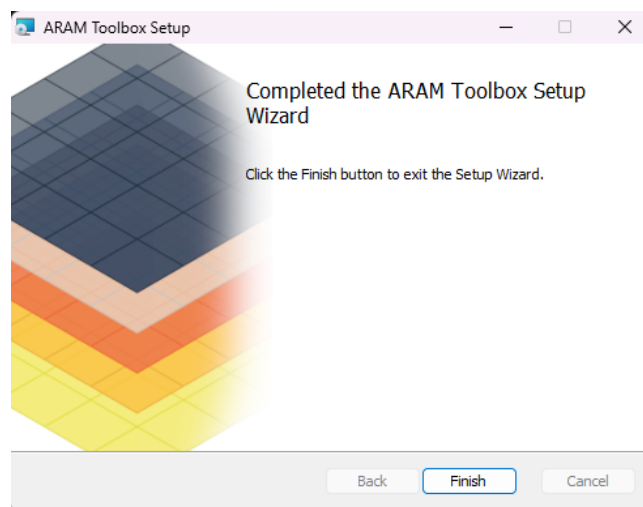


Figure 6: Finished installation wizard

Now, you can open Enterprise Architect. If you go to *Specialize* you will find the ARAM Toolbox under *Add-Ins* (Figure 7).

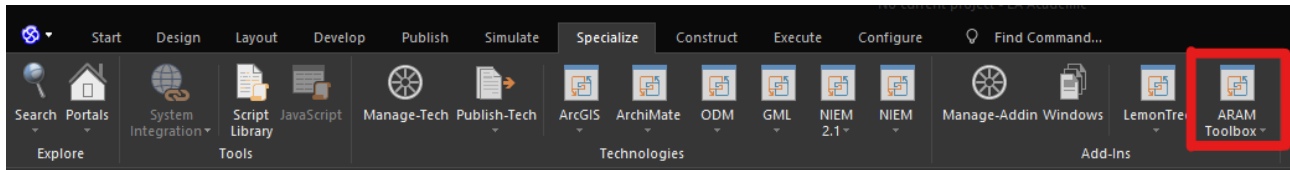


Figure 7: Where to find the ARAM Toolbox in Enterprise Architect