TITLE : SOA[Service Oriented Architecture]

AUTHOR : 정재우 / Jaewoo Joung / 郑在祐/ (A.K.A.)海绵宝宝

DATE : 2023. 01. 17

TARGET AUDIENCE : Who wants to know SOA from begining

Why we use SOA?

Compared to architectures that preceded it, SOA offers significant advantages to the enterprise.

1. *Increased business agility, faster time to market*: The efficiency of assembling applications from reusable service interfaces instead of rewriting and re-integrating every new development project allows developers to respond to new business opportunities and build applications much faster. You can.
2. *Ability to leverage legacy capabilities in new markets*: A well-crafted SOA allows developers to easily use 'locked-in' capabilities on one computing platform or environment and extend them to new environments and markets. For example, many companies are using SOA to expose functionality from their mainframe-based financial systems to the Web to service processes and information that customers previously could only access through direct interaction with the company's employees or business partners. make it possible.
3. *Better collaboration between business and IT*: In SOA, a service can be defined in business terms (eg 'generate insurance quote' or 'calculate capital equipment ROI'). This allows business analysts to work more efficiently with developers on critical insights that can lead to better outcomes, such as the scope of a business process defined by a service or the business implications of process changes.

1. Overview General

1.1 Purpose

This documentation defines in detail how to design services in a service-based architecture and guides service designers
Standardized and unified design services to establish a unified vehicle service platform.

1.2 Scope of application

This specification applies to service design, system design, architecture design, software development and other related personnel based on SOA architecture
refer to.

Canonical Architecture

Service concept and basic rules	Service concept and basic rules	Service concept and basic rules	Service concept and basic rules
SOA Concept	SOA Hierarchy	SOA Design Target	Service Design Method
FLOW	**PRINCIPLES of SERVICE DESIGN**	**PRINCIPLES of SERVICE DESIGN**	**TOOL**
Design Process	Principles of service application	Principle of granualrity classification	Design TOOL
Outputs	Interface definition Process	Interface parameter definition	Management Tools
RASIC	Naming convention	Service software deployment	UML TOOL
Process Change	Arbitration mechanism	functional safety measures for services	
Deviation from Application Process	Information Security Measures for services	Service status Management	
	Exception Handling for Services	Usecase design principles	
	Service Authority Management		

2 Definition and basic principles of service

2.1 Concept Definition

2.1.1 SOA

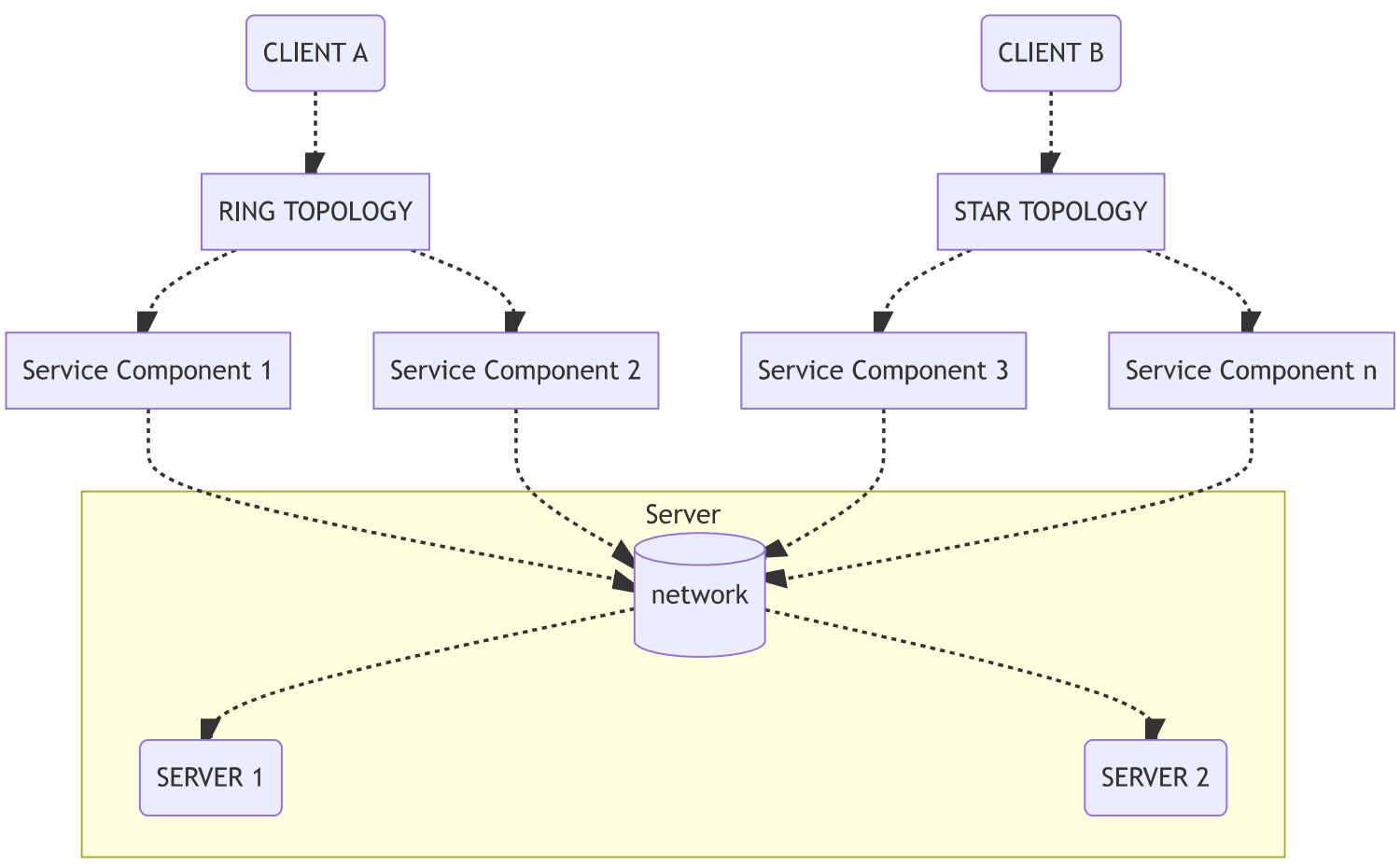
➢ SOA is a service-oriented software architecture and an architectural design idea;

➢ SOA is independent of operating system, programming language and software architecture;

➢ Meet the following principles:

- **Has Service contract and Service Discovery (e.g. SOME/IP SD)**
- **Loose coupling: There is a certain encapsulation boundary, and the service call interaction is realized through the interface**
- **Location transparency: the consumer of the service does not have to care where the service is located**
- **Services are stateless**

Reusable between heterogeneous platforms



2.1.2 Services

➢ A service is an independently executable software component of a service with an accurately described functional scope;

➢ Realize a complex function through the interaction of different services;

➢ Provide functions as services to other software components through precisely defined service interfaces

➢ Services may use other services to perform their functions

2.1.3 Service Interface

A service interface refers to the attributes and methods of a service that can be directly provided to the outside world. Generally, a service has multiple service interfaces.

2.1.4 Server (server)

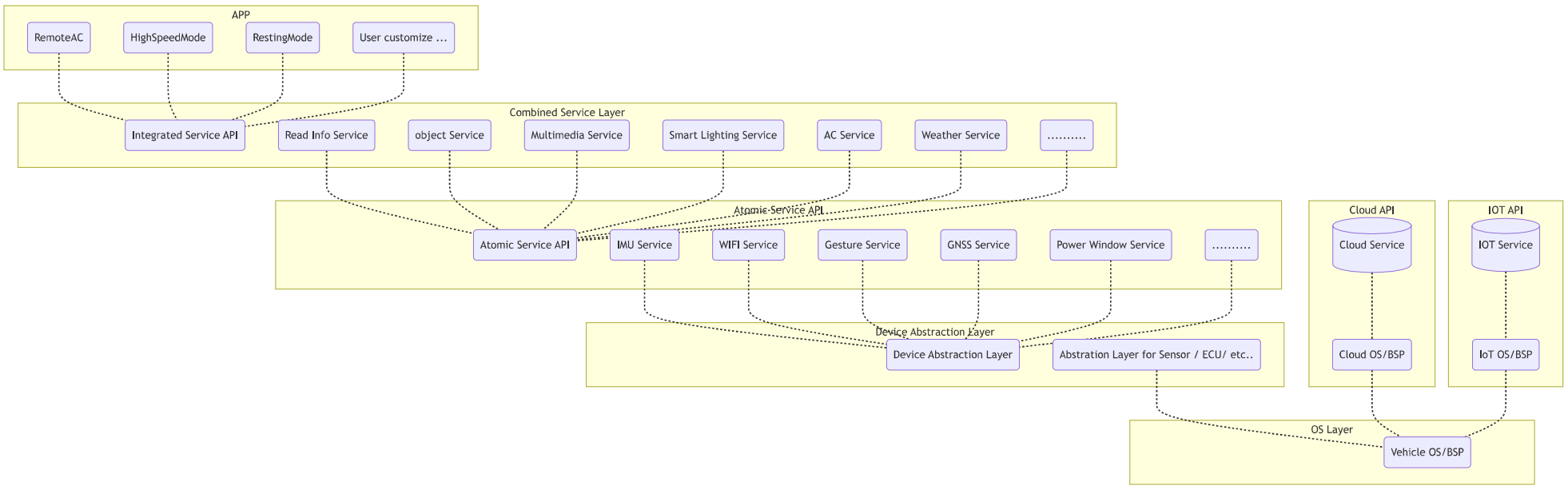
The server refers to a software entity that implements a service, that is, the provider of the service.

2.1.5 Client (client)

A client refers to a software entity that invokes a service to implement a functional logic or another service, that is, the consumer of the service.

2.2 Hierarchical structure of services

The hierarchical relationship of services is shown in the following figure:



Service Hierarchy Diagram

2.2.1 Hierarchy Definition:

➢ Application: by invoking multiple combined services, to realize product functions in specific scenarios, such as realizing a new ambient light mode;

➢ Combination service: Based on multiple services or other capabilities, through a certain logic, a type of service interface with universal capabilities is formed after combination, and this type of service also has strong reusability;

➢ Atomic service: Atomic service layer: a functional module that implements certain data fusion or control logic. As the smallest unit and single execution entity of the service, it provides basic services that can be arranged on demand for the application upward through the API, enabling one-time development and multiple reuse, and maximizing the development efficiency;

➢ Device abstraction layer: abstracts hardware resources such as sensors, actuators, and Legacy ECUs, provides device access interfaces for services upward through APIs, shields device function implementation differences (hardware differences & manufacturers differences), and reduces customization and duplication of labor;

2.2.2 Basic Principles of Service Layering

➢ Atomic services cannot call other services;

➢ The lower-level service cannot call the upper-level service;

➢ The application should call the composite service first, in principle, the atomic service cannot be called directly (if necessary, it needs to be approved by the architect);

➢ The API interface of the atomic service layer should be standardized as much as possible to ensure the reusability of the upper-layer composite services and applications;

➢ The API interface of the device abstraction layer should be standardized as much as possible, compatible with the diversity of the sensor execution layer, and reduce the impact of hardware changes on atomic services;

2.3 Design Goals of SOA

GOAL

1. Realize the hierarchical decoupling of the architecture through the design of standard service interfaces between layers to meet the needs of rich scene changes;
2. Encapsulate according to the capability unit to realize service interface reuse;
3. Break through the inter-domain, inter-vehicle-cloud protocol, language and other heterogeneous platform restrictions through standard communication protocols, and realize resource sharing and interconnection.
4. Standard interfaces can be used for application development to achieve smooth and seamless deployment of multiple platforms
5. Realize the standardization of vehicle data and the standardization of cloud storage data
   2.4 How services are designed
   Service design generally has two methods: Top-Down and Bottom-Up:

2.4.1 Top-down

The top-down approach is called forward design. In the service-based architecture, most complex cross-domain functions (such as cloud-pipe-terminal linkage), newly introduced system functions, etc., need to be designed in this way during the development process. The gist of it is as follows:

➢ Starting from requirements, carry out logical disassembly, forward design of system and software architecture, and services designed for function realization.

➢ Service design follows logical architecture design, so this type of service should be generated synchronously during system design(SRD). During logic design, FO is responsible for functional requirement transformation and logic architecture design, defining the boundaries and interaction of logic modules, and generating service requirements (such as data flow usercase) during LC deployment.

➢ The service designer checks in the service library according to the requirements. If the existing service in the library can meet the requirements, the service will be used. If not, a new service will be designed.

2.4.2 Bottom-up

It is applicable to the existing systems inherited in the platform architecture design, or the deployment of functions has a clear conclusion. In this case, we can abstract some services according to the capabilities of existing domains, ECUs, and subsystems. Mainly from the following aspects:

➢ More traditional and relatively independent systems, such as windows, seats, air conditioners, etc., can have corresponding ECUs to provide interface capabilities, and the corresponding domain controllers can reveal services;

➢ The power and chassis domains involve relatively high safety, and generally less control interfaces are designed.The CAN and LIN capabilities are sorted out, and the status signals are represented, which is suitable for bottom-up;

➢ Hardware abstraction, modules with independent hardware capabilities such as sensors and actuators, the general interface is relatively simple and fixed, and the abstraction of the service interface can be realized from the bottom up;

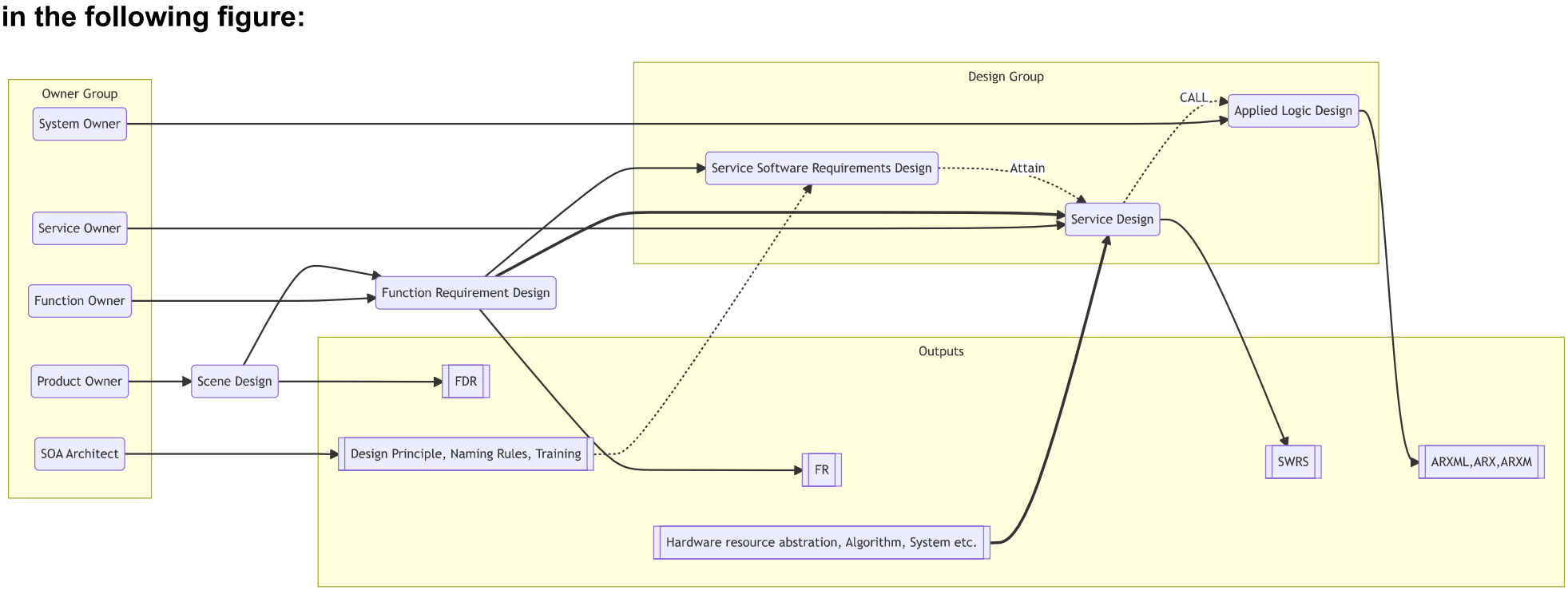
➢ The independent algorithm module is also abstracted like the hardware module. For example, the fatigue detection module outputs the fatigue level, and theThe aviation module outputs road information, etc.Therefore, in the early stage of architecture design, the basic service can be abstracted in a bottom-up manner, and then the service library can be supplemented from the top down according to the functional requirements.

3 Development process

3.1 Service Design Process

➢ In the development process of SOA architecture, it is generally used to extract atomic services from the bottom up, and then define the process of combining services from top to bottom according to the business relationship.

➢ The general process of service development is combined with GEELY's original development process, as shown in the following figure:



In the implementation of a specific project, the service development process strategy should be defined by the architect according to the actual state of the project.

➢ Definition of relevant roles in the process

ROLE

ϟ Product Manager: Service definition and acceptance of vehicle function product form, output product definition documents and

Usecase;
- Function Owner: Responsible for undertaking specific Usecase requirements, formulating implementation plans, decomposing them to the system, completing the handshake, and outputting functional requirements documents;
- System Owner: Responsible for undertaking the functional requirements and service requirements of the system, designing software requirements, and outputting software requirements documents;
- Service Owner: defines all the attributes of the service itself, such as the interface, parameters, and instructions of the service;
- Architect: Define design principles, design specifications, etc., propose and review SOA implementation plans.

➢ Service output

- ARXML file, including service interface design, parameter design, communication-related parameters, etc.;
- Service detailed design document (can be placed in SWRS), including service description, Usecase, interface description, sequence diagram, etc.;

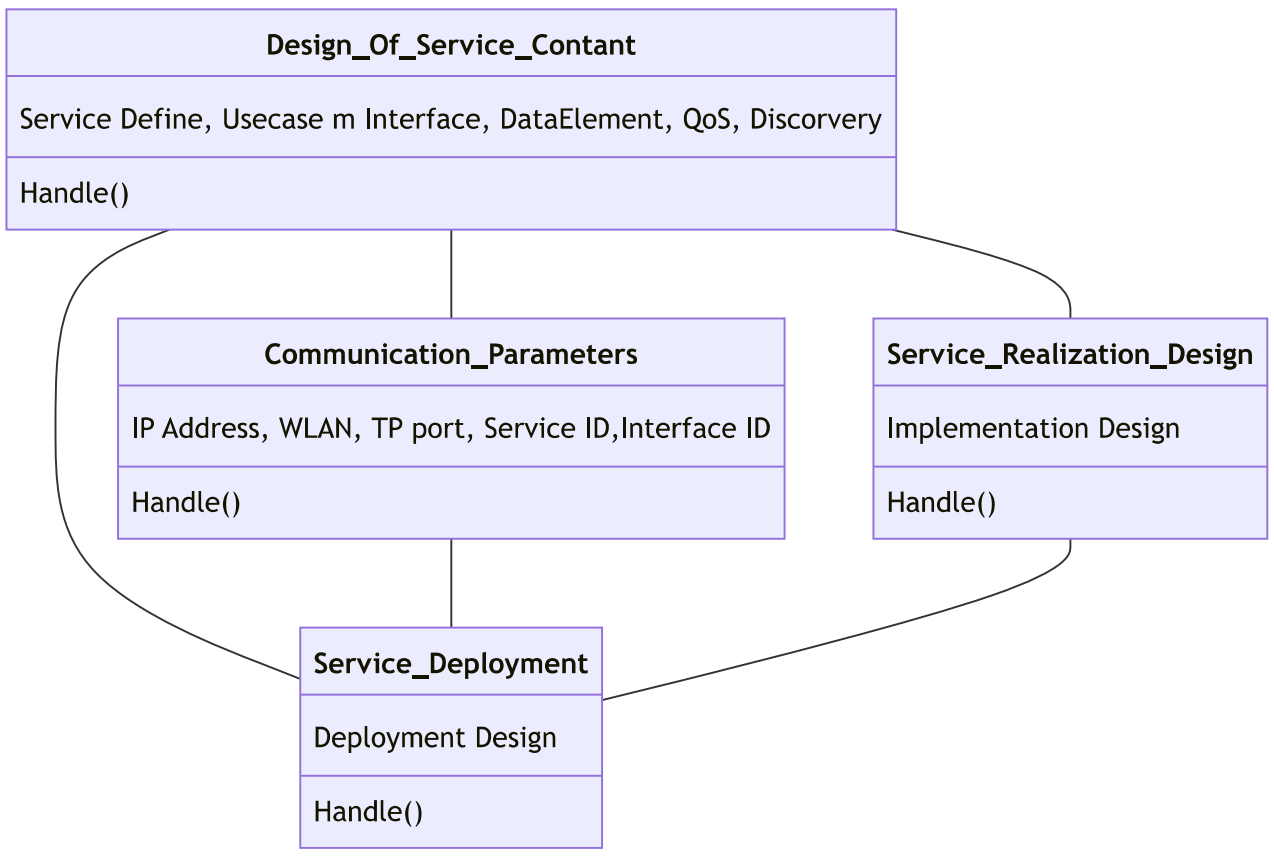
3.2 Change Process of Services

See document "SCR Process Description"

3.3 Specification Deviation Process of Service Design

If there is a situation that violates the definition of this specification during service design, it needs to be reported to the architect for approval before it can be valid.

4. Service Design Principles



➢ **Service name design: Service Name defines the name of the service according to the naming rules of the service interface below, and the name reflects the key capabilities of the service;**

➢ **Service use case design: According to the basic principles of use case design below, extract the basic capability boundary and interaction granularity of the service to form a Service Usecase diagram;**

➢ **Service interface design: According to the following interface design principles, define the data exchange method between the service provider and the consumer through different Service interfaces;**

➢ **Parameter design: According to the following service parameter design principles, different DataElement definitions are used to meet the data transmission requirements of different interfaces;**

➢ **Service interface timing: define the timing relationship between interfaces through UML interface timing diagram;**

➢ **Service QoS design: service reliability measures, such as failure Retry mechanism, etc.;**

➢ **Design of service release and stop strategy, define in what state this service can be provided, such as default power-on, or define other conditions to provide;**

➢ **Design of service interface failure handling strategy: such as timeout of response in R-R method mechanism For details, please refer to the service interface failure handling policy below.**

4.1.2 Communication parameter design

➢ **The parameter design of the service itself such as service ID, interface ID, time group ID, etc.;**

➢ **Definition of network communication related parameters, VLAN, IP/PORT, etc.;**

➢ **Selection of transport layer protocol: UDP is selected by default, and is selected when the total length of the payload exceeds 1440 bytes TCP;**

4.1.3 Service Deployment Design

In the system design phase, each service should be deployed on a SC (Service Component) for logic Implementation, and then follow the SC to deploy on the corresponding hardware and implement software.

4.2 Extraction of Usecase for Services

➢ The use case of the service defines how the service is used by the consumer, and describes the dialogue between the consumer and the service in order to use the functions provided by the service;

➢ The boundary definition of service Usecase should be within the scope of business, without system restrictions or technical constraints;

➢ The granularity of service Usecase extraction should be based on business activities rather than specific implementations.

4.3 Application principles of services

➢ Consider using the service for cases that do not require high transmission timeliness (end-to-end transmission delay tolerance > 100ms, this value is an empirical value);

➢ Depends on the function implemented by the service, and the functional safety level of this part of the link is required to be less than or equal to ASIL B;

4.4 Priority Principles for Service Extraction

➢ For resources with strong reusability, that is, the same resource may be used in multiple places, and priority should be given to the use of services;

➢ Independent capability units (hardware, algorithms, etc.) are preferentially packaged into services;

➢ The resource types are the same, but are provided in multiple places (such as cameras, door locks, etc.), which can be prioritized as services and deployed in multiple places through multiple instantiations;

➢ In the vehicle-cloud integration architecture, since the cloud interfaces restful and Http are also service-based, it is recommended that some cross-vehicle cloud information transfer services, such as Fota, account management, etc., should be used first;

➢ If SOME/IP communication is used, the service interface should be based on the status and control interface. The access address of large data (audio and video, large files, etc.) can use the service transmission, but the transmission of big data itself cannot use the service.

4.5 Principles of Service Granularity Division

The service granularity division should follow the following two principles:

➢ The granularity of the service should consider the reusability, flexibility and performance of the service as a whole;

➢ The key point of atomic service is to consider its reusability. It should take the service extraction object as the unit, and provide the service with the smallest granularity

➢ Composite service is the combination of different atomic services and business, and should be divided according to the principle of business autonomy, and provide services with corresponding granularity;

4.6 Principles of Service Interface Definition

4.6.1 Interface type (take SOME/IP as an example)

Field (Setter/Getter/Notifier): Represents a property that can be set/read/subscribed;
Method: Represents a method that is invoked by request/response;
Fire and forget Method: Indicates a method, only a request, no response;
Event: Represents an event. Different events can form an event group and complete the call through subscription/notification;

4.6.2 Selection principle of interface type

4.6.2.1 Control interface design

The control class interface usually selects the RR-Method class interface for transmission. Generally, there are two forms: synchronous and asynchronous:

➢ Synchronous interaction: refers to sending a request, you need to wait for the return, and then you can send the next request, there is a waiting process; you can directly return parameters through the return value of Method

➢ Asynchronous interaction: refers to sending a request without waiting for the return, and can send the next request at any time, that is, without waiting. The asynchronous interactive service provider can delay providing the final result data to the caller. During this period, the caller can release the occupied thread and other resources to

avoid blocking, and wait until the result is generated and re-acquire the thread for processing. The asynchronous interface can use the RR+Event method, that is, the initial result data is returned first through Response, and when the execution is completed, the final result data is returned through Event.

➢ Generally, F&F type interfaces are not selected.

4.6.2.2 State class interface design

The state class interface can use two forms of RR-Method and Event interfaces as needed:

➢ For those who need periodic feedback or event broadcast from Provider, use the Event interface;

➢ For the state that needs to be actively obtained by the Consumer, the Method interface is used;

➢ The interface of the state class needs to support the above two capabilities at the same time.

4.6.2.3 Use of Field

To simplify the interface, Some/IP defines an interface of type Field, which can be used to represent interface types with multiple functions.

➢ The function of Field-Setter is equivalent to the function of the interface used to control the class in RR-Method, the main difference is that the form of carrying parameters is different, field can only carry one parameter;

➢ Field-Setter is not applicable to asynchronous interface, Method should be used;

➢ The function of Field-Getter is equivalent to the function of the interface for state class in RR-Method, the main difference is that the form of carrying parameters is different, field can only carry one parameter;

➢ The Field-Notifier function is equivalent to the function of the Event interface, the main difference is whether the initial value needs to be set;

4.6.2.4 Composition principle of EventGroup

In order to use resources in a targeted manner, several Event/Field-Notifier interfaces in a service should be divided into several EventGroups according to their functional purposes.
EventGroup design principles are as follows:
DESIGN PRINCIPLES

1. When there is Event or Notify in the service, you need to define an evnetgroup for each Event and Notify
2. There can be multiple eventgroups for each service, and an eventgroup containing all Events and Nofity needs to be defined
3. For the custom subscription requirements of a specific user, a corresponding eventgroup can be created separately. This design requires the authorization of the architect. For example, if a client only cares about WipingLevel and WipingMode, you can consider packaging the two into one eventgroup.

4. For the same physical category, it is generally necessary to obtain and use at the same time. Consider forming an eventgroup, for example, the states of X, Y, and Z axes can be combined.

4.6.2.5 Instance usage principles

In the whole vehicle system, there are often different resources that provide a capability at the same time, such as multiple cameras of the same model, multiple glass lifts, etc. In this way, when servitizing, for versatility and saving development resources, the same one will be considered. Services are deployed in multiple places through multiple instantiations.

➢ The ID of the instance is only bound to the deployment location and is not affected by other factors;

➢ Instances of the same service are distinguished by instance ID, and multiple instances can be deployed on the same ECU or on multiple ECUs;

➢ ECUs deploying multiple service instances should listen to different ports of each service instance (because instanceID is not reflected in Some /IP Header, it can be distinguished by the combination of service ID+Socket (IP, TP type, port));

➢ If the service is deployed on multi-core MPU or multi-operating system, it should be allocated according to the virtual host and considered according to multiple ECUs.

4.7 Principles of Interface Transmission Parameters

4.7.1 Basic parameter types

Several basic data types commonly found in services are as follows:

➢ Uint/Int

It is only necessary to consider whether the range of shaping meets the requirements. It is relatively simple and common to use. In Ethernet, it is not necessary to define the precision and deviation values like CAN signals. Int8.

➢ Float

If you want to describe non-integer physical quantities, such as temperature, length, etc., you can use floating point. It is mainly necessary to consider the range of the data and whether the length is sufficient; the physical value is directly transmitted, and it does not need to be converted like the CAN signal;

➢ String

If you need to describe a piece of text, numbers or a file in Json format, String is generally used. Because C++ supports the definition of dynamic-length strings, whether it is fixed or not, the default is variable-length strings;

➢ Structure

Structure is a common data type design method, and all data types can be freely combined in a structure. This method can solve the problem that some service interface types only support one parameter.

➢ Array

When multiple data elements of the same type need to be represented, the form of an array can be used, and the data elements only need to be
It can be the same type, or it can be a set of structures. If the length is fixed, you can use Fix Length
Array, if the length is not fixed, use Dynamic Array, Dynamic Array needs to be defined on the array limit, as far as possible to ensure that the limit is within 1440 bytes;

➢ Union

Deformables are mainly used to indicate that in some cases, you want to use a data to represent different meanings according to different scenes.
Because this data type has very small usage scenarios and is cumbersome to implement at the application layer, it is not recommended to use it.

4.7.2 Principles of Data Type Selection

➢ When the service interface carries the application data type, it is necessary to consider the characteristics of the interface. It is stipulated in AUTOSAR that, except Method, both Event and Field can only carry one parameter. If multiple parameters are required, they can be designed in the form of Struct;

➢ Do not use Boolean value, use Uint8 instead;

➢ RR Method, etc. need to have a unified return value RetVal. For the definition of standard parameters, see Appendix 2 of the checklist.

➢ The return value of all interface calls must have an error code, and the return value of the get type can be made into a structure together with RetVal;

➢ When enumeration is defined, the abnormal value Error or Unknown state should be defined as the maximum value. For example, for the parameter type of Uint8, the abnormal value should be set from 255 down, that is, when there are two abnormal values, take 255 and 254;;

➢ In order to increase readability, the return value of Method should be defined as the structure RetValStruct;

➢ Common parameter types should be unified, for example, time (year, month, day, hour, minute, second, millisecond) should be defined as Uint64;

➢ The parameter type of latitude and longitude coordinates should be defined as Float, etc.;

➢ For fixed-length array FixArray type, the specific length should be defined;

➢ For the dynamic array type DynamicArray, the upper limit of its length needs to be defined;

➢ For the parameters of the list type, the length often changes, consider using a dynamic array;

4.8 Service and Interface Naming Guidelines

When designing SOA services, the following naming conventions need to be followed

4.8.1 Basic Specifications

➢ Try to use the full name of the word, and refer to the "GEELY SOA Abbreviation Table" for abbreviation after the name is too long;

➢ The name does not contain special characters "," "." "/" "" "~", etc., and can be separated by "_";

➢ Considering the reusability of services and parameters, no topology node information should be included in various names;

➢ Service name, interface name and parameter name cannot be repeated;

➢ The following system keywords cannot be used in naming:

naming

- skeleton
- proxy
- internal
- resources
- method
- event
- field
- input/output
- amsr
- ara
- com
- someip
- base
- vac
- std
- serialization

4.8.1.1 Service Name Service Name

The service instance and the service implementation that refers to the definition, if the service is written separately later, it means the service instance

➢ The service name shall be capitalized and the name shall be in English;

➢ The service name cannot have an underscore;

➢ The service name suffix needs to end with Srv;

eg. LedControlSrv

4.8.1.2 Service Interface Name Service Interface Name

The communication interface between services on the SOA platform has three forms: Event, Method and Field. ServiceInterface is used to define the Event/Method/Field message type and specific namespace, regardless of the specific communication protocol.

➢ The service interface name should be capitalized with the first letter of the word (camel case), the full English name, the gerund structure, and be limited to 3 words as much as possible;

➢ The service interface name cannot have an underscore;

● Event

The Event interface represents the actual transmitted data, and takes the data as the operation object. As long as the meaning of the data can be clearly expressed, the naming specification follows the basic specification, and the suffix should end with Evt; for example, the Event corresponds to the Method, and the Event represents the asynchronous call of the Method. Feedback, the name should be unified.

eg. CurrentVIeEvt : current value
eg. GetWiperState and WiperStateEvt : asynchronous call of wiper state

● Method

The Mehtod interface represents a certain control, and the communication method adopts RPC remote call, usually with verb behavior, such as control, status query, transmission, registration, setting, etc. Among them, Method is further divided into F&F, and
There are two types of R&R, FF is a single call, no feedback is required, and RR is a request response, which requires feedback.
The interface name needs to clearly express the meaning of the method. It is recommended to use a gerund for naming, and use camel case.
The nomenclature, based on the CURD/REST reference, should end with the suffix Mtd. The following basic naming paradigm is designed:
get get the status set set the status rpt pass the information jdg judge the event
crt creates threads/processes/dynamic services/files/events, etc. del deletes threads/processes/files, etc.
eg.
setSoundOnMtd : sound the horn
jdgVehMovingMtd : Determine if the vehicle is moving getCarCfgMtd : Get car config value rptComponentStsMtd : Report component status information
crtXTaskUsrLightShowSrvMtd : Create user-defined light show service delDebugClass0MsgMtd : Delete all msg debug files with debug level 0

● Field

Field represents an attribute, usually refers to a state value or some kind of information, and the name should clearly express the content of the attribute righteous.
Field contains the following three types of information: getter : read-only interface, whose prototype is method, obtains server information notifier : read-only interface, whose prototype is event, receives data from the server
setter : write interface, the prototype is method, set/modify server-side related information
Field name should be: noun + Fld, the noun is the attribute name.
eg. VehSpeedFld : Vehicle Movement Control

4.8.2 EventGroup name Event group name

In order to clearly express the meaning of the event group, EventGroup should be named according to the function characteristics of the event group, ending with _EG. Such as MapUpdateSts_EG.

4.8.3 Parameter name

➢ The parameters of the structure type should be suffixed with Struct, such as CellStatusStruct;

➢ Array type parameters should be suffixed with Arry, such as BasicStationInfoArry;

➢ Simple data types can be named with the nouns of their own characteristics;

➢ To increase readability, you can choose to add words such as "status", "info", "opt", "ctrl" as suffixes;

4.9 Service Deployment Principles

The deployment of services mainly refers to the implementation of services on the application layer SWC. Service deployment is closely related to the software architecture of the application layer, so it needs to be completed by the service designer and the software architecture designer of the corresponding controller. The following are general requirements only:

➢ In general, services should be deployed in loosely coupled systems such as Adaptive AUTOSAR, and different services should be deployed in corresponding APPs according to the definitions of their providers and consumers to achieve decoupling between application layers;

➢ Atomic services can also be deployed in systems with strong coupling at the application layer such as Classic AUTOSAR. You can choose a dedicated SWC to centrally process the provision and consumption of services, which can be implemented in the form of Sender-Receiver Port;

➢ If the same service is deployed on different hardware or APPs, you need to use instance to distinguish.

4.10 Service Arbitration Mechanism

➢ The priority control of general services should be completed by the corresponding modules in the middleware, such as IAM in the AP;

➢ If the priority control is not implemented in the protocol stack, or the special needs of the business, the application processing method can also be used.

formula, the scheme is as follows:

● Basic principles of arbitration:

ARBITRATION PRINCIPLES

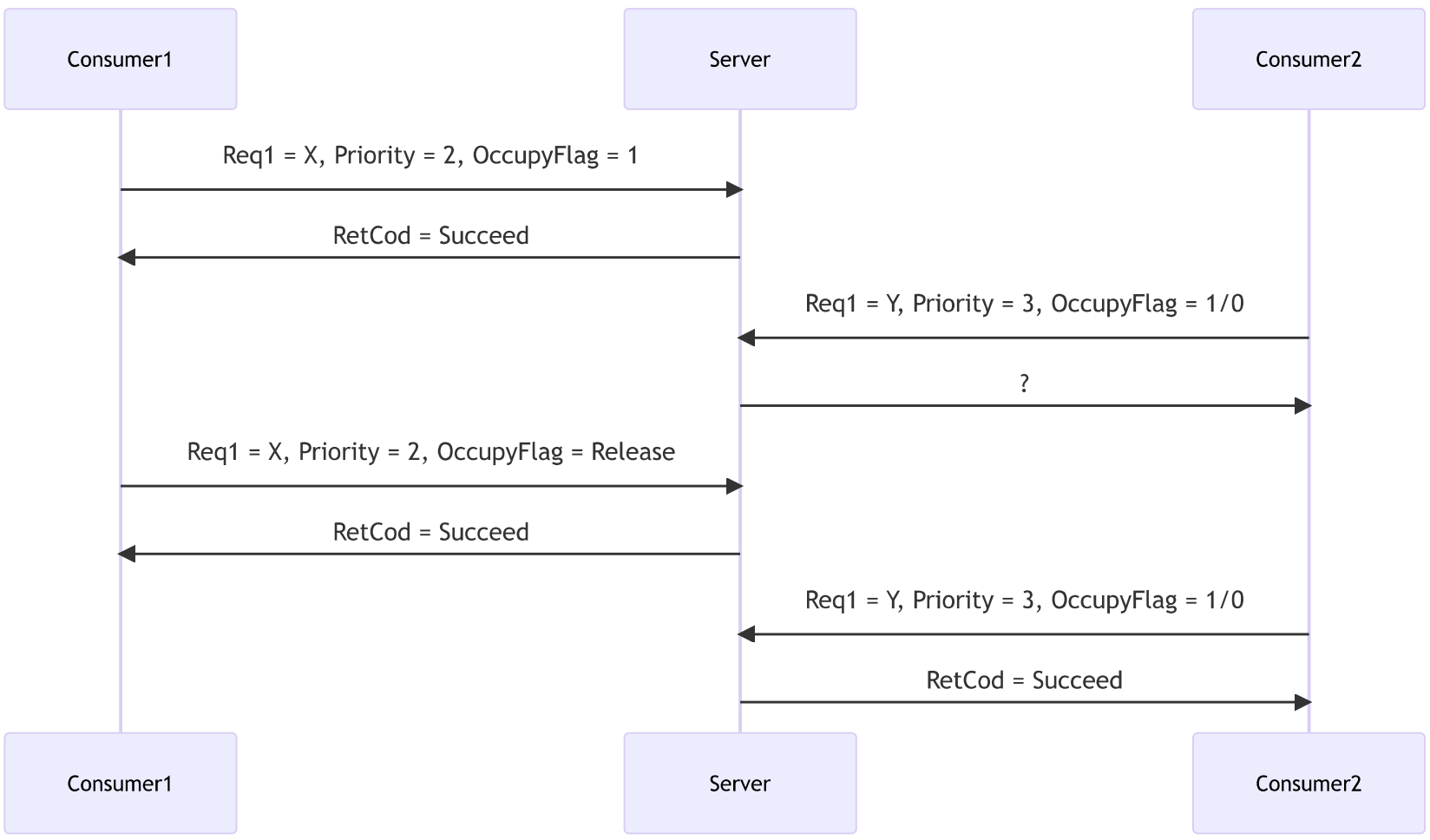
1. Arbitration is conducted by the provider of the service;
2. Only the control interfaces of some services need to be arbitrated (continuous actions or actions that need to be maintained);
3. Arbitration is based on a class of attributes of the accused object.

   ● Arbitration implementation:

   ARBITRATION IMPLEMENTATION
4. The server divides the service interface capabilities into several levels according to the importance of the required business (such as security, comfort, entertainment, etc., and several sub-categories can be set in each category), and the higher the level PriorityID can be set, higher priority;
5. For different types of actions, the parameter OccupyFlag needs to be passed in at the same time (0x0=false, 0x1=true), indicating whether the resource needs to be occupied;
6. The presence of OccupyFlag or the "continuous execution period of the action" is regarded as occupied, and the occupancy period needs to be considered

   PriorityID, no need to consider PriorityID;
7. For different consumers of the same priority, the later one has priority and interrupts the former;
8. High priority interrupts low priority, low priority waits for high priority, and returns the status;
9. The priority level ID can be obtained from the priority level table and passed from the consumer to the provider;
10. The priority level table is maintained by the corresponding subsystem.

Example of Arbitration



4.11 Functional Safety Definition of Services

The functional safety of a service is uniformly defined by the functional safety team.

4.12 Exception Handling of Services

4.12.1 Consumer

➢ For the R-R method type interface, after the Req information is sent, if no Resp feedback is received within a period of time (if there is no special business requirement, the default is 2S), it is defined as a communication timeout, and the consumer should start the next sequence (retransmission). or an error). If this phenomenon occurs for N consecutive times (defined by the service diagnosis policy) within a single power-on cycle, the timeout fault is recorded;

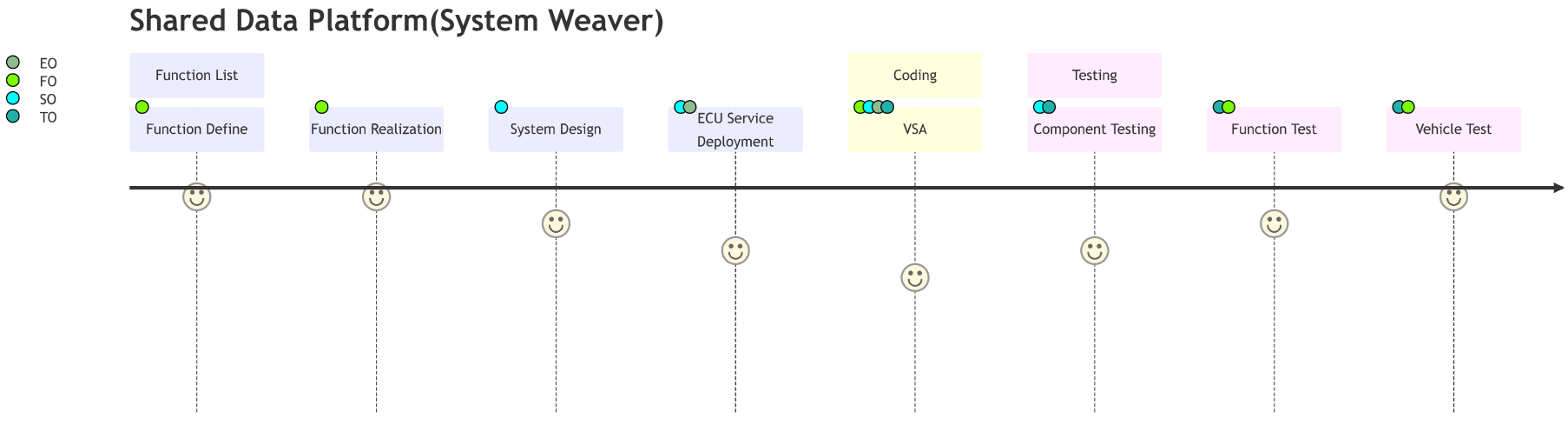
➢ For the Event interface, if it is sent periodically, the communication loss can be judged according to the CAN signal, that is, if the Event message is not received for N consecutive cycles, the failure of communication loss is recorded;

4.12.2 Server

The server cannot dire#### ctly diagnose service exceptions based on the relevant information of service communication, and should record the buried points at the key points of service implementation. The specific scheme should be determined by the business link of the specific service and the log scheme of the server software.

5 Development Toolchain for Services

5.1 Design Tool Service design is designed in SystmWeaver, and System Weaver generates software requirement specification SWRS and service interface file Arxml.



5.2 Data Structure Service development The overall data structure is consistent with GEEA_2.0, service design is integrated in the subsystem design level, Service, ServiceComponent and ServiceInterface module design is added, and service deployment is integrated in the ECU design.

In the planning level, add Machine and Process designs to undertake the deployment of service designs;

Overall data structure diagram

- SOA POC
  - Attribute
    - Function List
  - functions
    - visibility
- SOA Architecture
  - Function Design

        - visibility
    - SOA Design
        - Function Design
            - Body Domain
              [+] ExteriorLight Subsystem
              [+] Audio SubSystem
              [+] VMM SubSystem
              [+] APP SubSystem
            - ECU Allocated Design
              [+] ECU System
            - Service List

★ Service and Service Interface

Define the interface data of Service and Service, the interface data includes Method, Event and Field data types;

Service Data Structure Diagram

- PowerWindowService Service
    - PowerWindowService Service Interface
      [+] GetWinCurrentStatus Method
      [+] CtrlWinPosition
      [+] NotifyWinCurrentStatus Event
      [+] WinLiftPosition
      [+] WinLockSwitchStatus Field

● ServiceComponent

Define the service application component that implements the function, ProvideService is the service capability provided by the service application component, and ConsumerService is other services that the service application component needs to call to realize the function.
ServiceComponent data structure diagram

- FrontWiperSWSrv Service Component
  [+] Requirement Service Requirement
    - [+] FrontWiperSWSrv ProviderService
        - [-] FrontWiperSWSrv
    - [+] FrontWiperSrv ConsumerService
        - [-] FrontWiperSrv

● Machine and Process

Machine defines an entity that the APAUTOSAR software stack runs with the operating system for Deployment implementation Service application components, service components are deployed to different processes.

Machine and Process data structure diagram

- CCCMachine
  [-] WiperProcess
    - [-] FrontWiperSWSrv
        - [+] Requirement
        - [+] FrontWiperSWSrv
        - [+] FrontWiperSrv

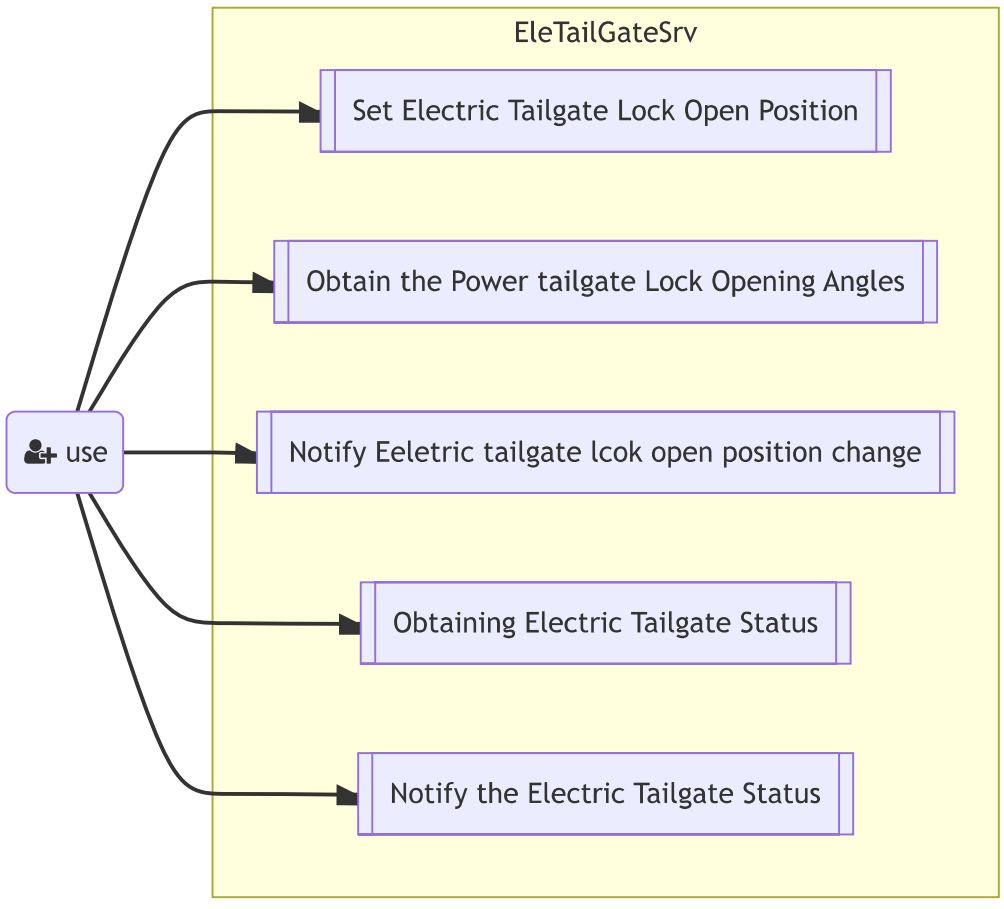
5.3 Service data output

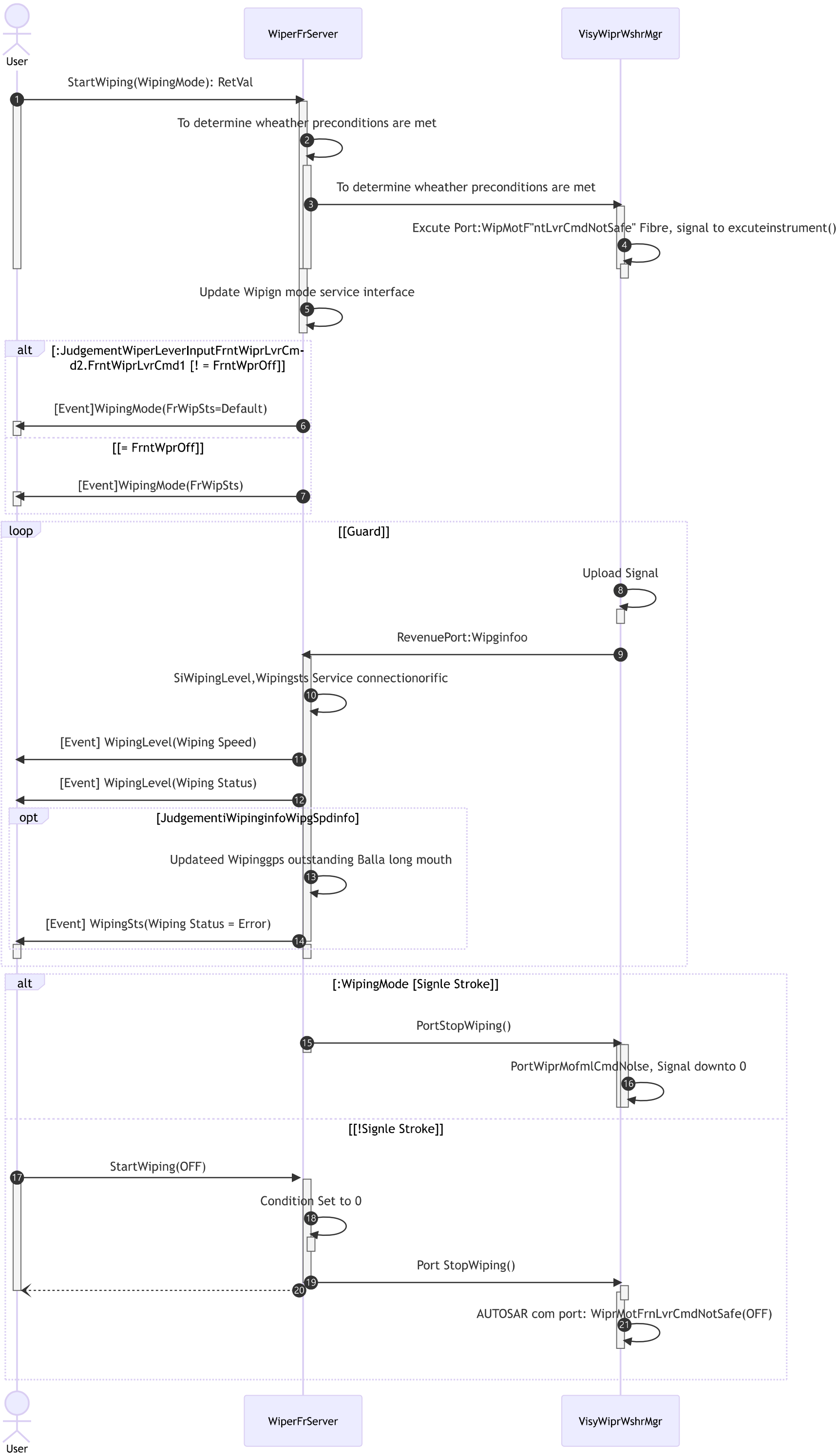
The output of the service design is the requirement specification document SWRS and the interface file Arxml, which are output through the SystemWeaver tool after receiving and deploying at the ECU layer.

5.4 UML Design

In service design, in order to express the activity relationship between services and consumers and the time sequence relationship between interfaces, sequence diagrams and use case diagrams need to be used. UML design tools are used, and Astah or (draw.io) software is recommended. The specific design method refers to the standard UML modeling method, which is no longer specifically defined in this specification.

The schematic diagram of the design case is as follows.


EleTailGateSrv
Set Electric Tailgate Lock Open Position
Obtain the Power tailgate Lock Opening Angles
use
Notify Eeletric tailgate lcok open position change
Obtaining Electric Tailgate Status
Notify the Electric Tailgate Status


User
WiperFrServer
VisyWiprWshrMgr
StartWiping(WipingMode): RetVal
1
To determine wheather preconditions are met
2
To determine wheather preconditions are met
3
Excute Port:WipMotF"ntLvrCmdNotSafe" Fibre, signal to excuteinstrument()
4
Update Wipign mode service interface
5
alt
[:JudgementWiperLeverInputFrntWiprLvrCm-
d2.FrntWiprLvrCmd1 [! = FrntWprOff]]
[Event]WipingMode(FrWipSts=Default)
6
[[= FrntWprOff]]
[Event]WipingMode(FrWipSts)
7
loop
[[Guard]]
Upload Signal
8
RevenuePort:Wipginfoo
9
SiWipingLevel,Wipingsts Service connectionorific
10
[Event] WipingLevel(Wiping Speed)
11
[Event] WipingLevel(Wiping Status)
12
opt
[JudgementiWipinginfoWipgSpdinfo]
Updateed Wipinggps outstanding Balla long mouth
13
[Event] WipingSts(Wiping Status = Error)
14
alt
[:WipingMode [Signle Stroke]]
PortStopWiping()
15
PortWiprMofmlCmdNolse, Signal downto 0
16
[[!Signle Stroke]]
StartWiping(OFF)
17
Condition Set to 0
18
Port StopWiping()
19
20
AUTOSAR com port: WiprMotFrnLvrCmdNotSafe(OFF)
21
User
WiperFrServer
VisyWiprWshrMgr

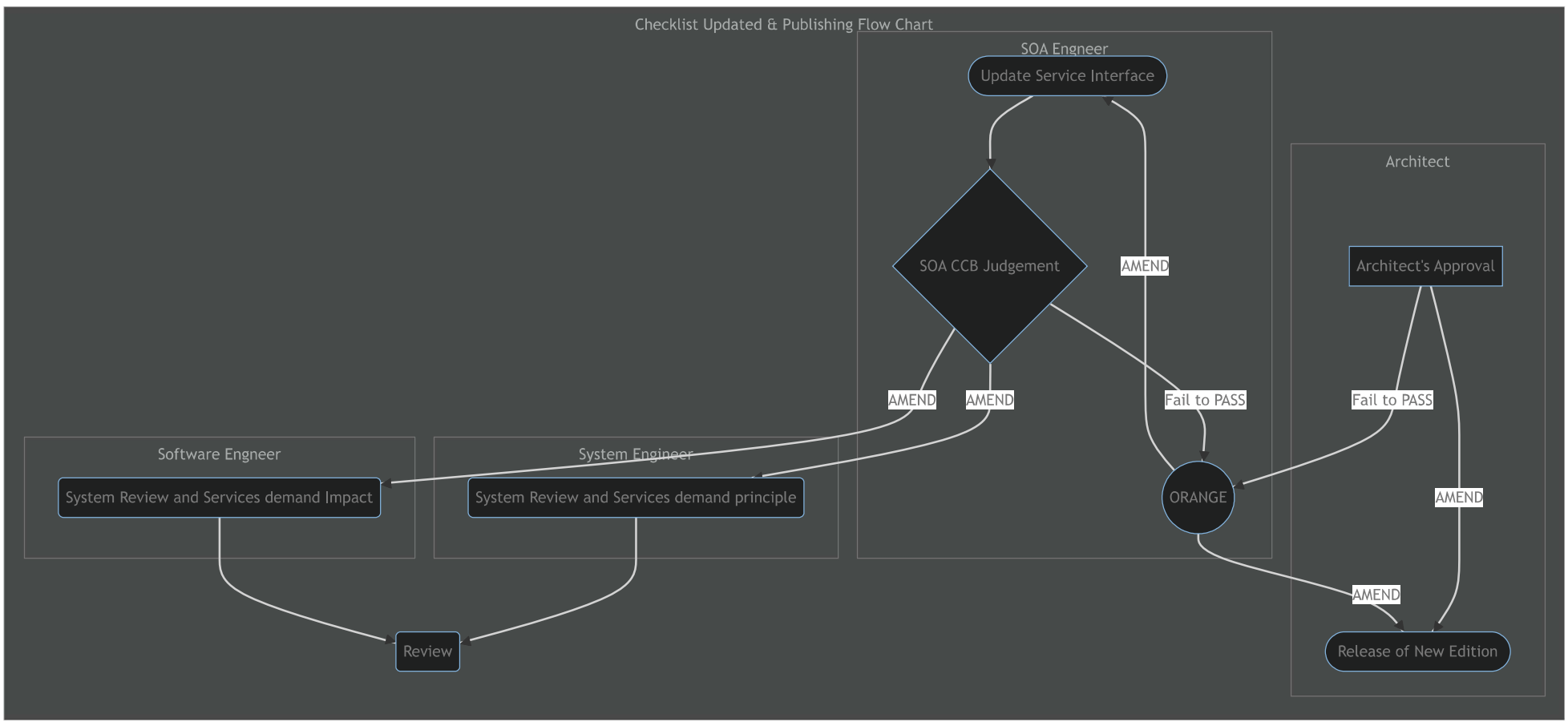
6 Design Verification

Service design acceptance includes service architecture principles, service interface definition principles and service requirements definition principles. In addition, the criteria for software acceptance of SOA services as a sub-item of the service design acceptance criteria will form a separate software acceptance checklist.

When SOA engineers and system engineers complete the design of service interface and service requirements and conduct design review, they need to check the content according to the specific version of the checklist. The checklist needs to indicate the name of the checked object,

Version information, author, date, applicable checklist version, and recorded comments should also be included in the checklist.

Checklist update and release process:



Similarly, changes in service requirements and changes in software acceptance standards must be approved by the corresponding system CCB and software CCB.

In other cases, the other two parties will be transferred for review and analysis, and after the final approval, the architect will approve and approve.

For the specific content of the Checklist, see the attached document "SOA Service Development Checklist".