applied sciences

MDPI

Article

Building Traceability Between Functional Requirements and Component Architecture Elements in Embedded Software Using Structured Features

Insun Yoo [1], Hyoseung Park [1], Seok-Won Lee [1,2,3] and Ki-Yeol Ryu [1,2,*]

[1] Department of Computer Engineering, Ajou University, Suwon 16499, Republic of Korea; dlsnfl18@ajou.ac.kr (I.Y.); bravephs88@ajou.ac.kr (H.P.); leesw@ajou.ac.kr (S.-W.L.)
[2] Department of Software and Computer Engineering, Ajou University, Suwon 16499, Republic of Korea
[3] Department of Artificial Intelligence, Ajou University, Suwon 16499, Republic of Korea
[*] Correspondence: kryu@ajou.ac.kr

Abstract: In embedded software for critical domains such as medical devices and defense, requirement traceability is essential for ensuring quality attributes. Standards and regulations mandate traceability between requirements and artifacts such as design elements and code. However, existing methods often overlook the hardware-dependent nature of embedded systems or conduct traceability retroactively, which may affect consistency. This study introduces a structured feature-based approach to component architecture design, bridging the gap between requirements and design to ensure traceability. The structured feature model supports traceability between functional requirements, software components, and hardware elements in embedded systems. A case study demonstrates that structured features can effectively map the requirements to design artifacts, helping to visualize relationships through a traceability matrix. Although the process is manual, structured features improve efficiency in the early stages of design and create traceable links between requirements and architectural elements.

Keywords: embedded software; functional requirement; requirements traceability; component architecture; structured feature

check for updates

Citation: Yoo, I.; Park, H.; Lee, S.-W.; Ryu, K.-Y. Building Traceability Between Functional Requirements and Component Architecture Elements in Embedded Software Using Structured Features. *Appl. Sci.* **2024**, *14*, 10796. https://doi.org/10.3390/app142310796

Academic Editor: Alexander Barkalov

Received: 24 October 2024
Revised: 6 November 2024
Accepted: 18 November 2024
Published: 21 November 2024

1. Introduction

Embedded software plays a critical role in various application domains, including medical devices and defense, where requirements traceability has become an indispensable aspect of software development and management processes [1,2]. In particular, ensuring traceability between requirements and design is crucial for guaranteeing the reliability and safety of software. Requirements traceability aids in effectively managing the changes and updates throughout the software lifecycle, contributing to project flexibility.

Embedded software must operate in real time under constrained hardware resources, necessitating close integration between software and hardware. In such complex environments, clear traceability between requirements and design becomes even more critical. However, the process of maintaining traceability can be resource intensive, and its importance is often overlooked during development [3]. Requirements traceability offers numerous benefits even in general software development, leading to the proposal of methods that can perform traceability after development. Notably, post-development traceability is often less effective than methods that integrate traceability throughout the developmental lifecycle. Moreover, unlike general-purpose software, embedded software with hardware dependencies requires specialized traceability methods that reflect these characteristics.

This study demonstrates that the structured feature-based component architecture design method can apply structured features to trace requirements in component design. Structured features serve as tools for bridging the gap between requirements analysis and design, effectively reflecting the requirements in the component architecture [4]. We have analyzed how structured features can support traceability not only between functional requirements and components but also between hardware elements affected by embedded software and components.

This paper proposes a method for applying structured features to ensure traceability within a component-based embedded software development process. The proposed method supports the component-based design process by enabling traceability between requirement analysis and component design, thereby ensuring traceability among functional requirements, components, and hardware elements affected by embedded software.

The rest of this paper is structured as follows: Section 2 discusses the background of requirements traceability analysis, covering component-based software architecture design processes, structured features, and the structured feature-based design process; Section 3 presents a review of existing requirements traceability methods; Section 4 analyzes the application of structured features at each step of the design process, helping to visualize the relationships between artifacts; Section 5 explains the traceability of structured features through a case study showing the intermediate outputs that reflect the requirements in the design; finally, Sections 6 and 7 summarize the findings, discuss their limitations, present our conclusions, and propose future research directions.

2. Preliminaries

This section provides an overview of the key concepts and methodologies that form the foundation of the proposed approach. Component-based development and structured features lay the groundwork for understanding how the proposed Structured Feature-based Component Architecture Design (SFCAD) integrates these concepts to achieve effective traceability between requirements and design.

2.1. Component-Based Development

A system can be constructed based on the interaction of components and interfaces, which exposes the system's functionality [5]. These components are developed independently and can be reused in other software systems. The goal of component-based development is to simplify system development, maintenance, and extension while improving productivity through reuse [6].

The component-based software development methodology has characteristics that can be applied to other existing software development process models. As shown in Figure 1, this methodology consists of two main processes, namely, the component development process and the component-based system development process, in which components are selected and assembled to suit the system under development. The component development process involves developing the components that make up the system and includes requirements analysis, design, implementation, testing, and maintenance. The components are identified and defined in the design phase [7].

The component-based development methodology can also be effectively applied to embedded software development [8–10]. For example, Electronic Control Units (ECUs) in vehicles consist of multiple software components that perform various functions, and each component can be developed independently and replaced or updated as required. This allows automotive manufacturers to manage and maintain systems more efficiently, thereby reducing development costs and time through component reuse in resource-constrained environments. AUTomotive Open System Architecture (AUTOSTAR) promotes the reuse and independent development of software components by ensuring interoperability between various ECUs and software modules through a standardized software architecture [11]. AUTOSAR's architecture comprises Basic Software (BSW), a Runtime Environment (RTE), and application software consisting of independent components. This structure enables flexible and efficient system upgrades and new feature additions while integrating components from various suppliers to reduce development costs and improve quality [12,13].

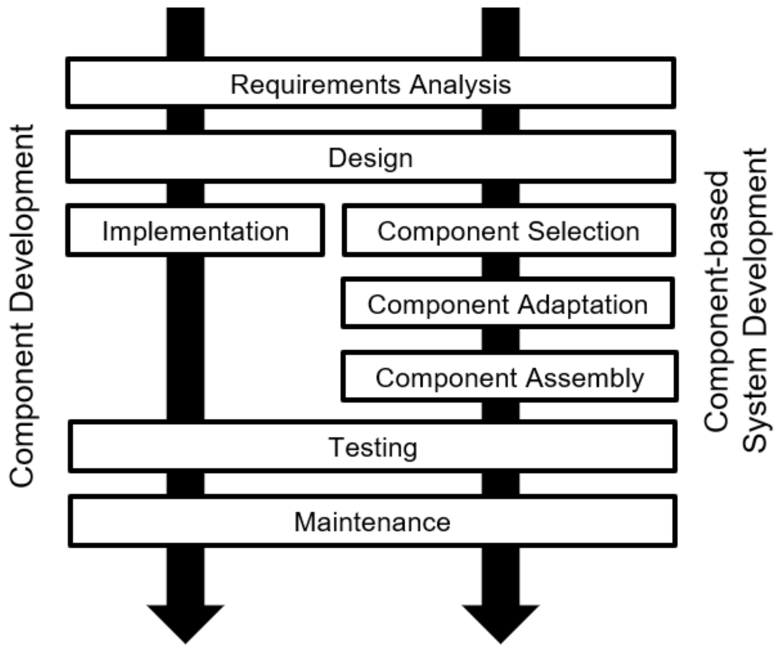


Figure 1. Component-based development process.

The defined components are specified for reuse. Component specification is a technique used to ensure that others can reuse the component accurately and appropriately. Thus, component specifications should be structured efficiently to match the purpose of the component. Component specification following the software contract approach includes elements such as functionality (describing the component's functional specifications), environment (infrastructure required for the component to run as intended), interface (specifications for how the component is invoked and how it interacts with other components), and nonfunctional properties (quality attributes such as performance, security, and reliability) [14,15].

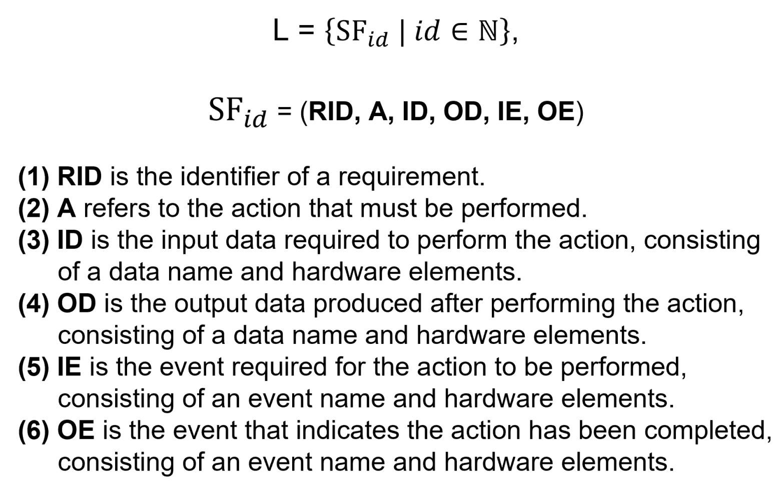
In component-based development, the components are designed and developed to construct the target system. This research focuses on the component development stage, and in particular on the component design phase, during which components are identified and the component-based software architecture (CSA) (referred to in this paper as component architecture) is constructed.

2.2. Structured Features

A feature is defined as a "Unit of functionality of a software system that satisfies a requirement, represents a design decision, and provides a potential configuration option" [16]. It encompasses the functionality and performance of the software as perceived from the user's perspective, bridging the problem space with the solution space [17]. The function block feature is an early model of the structured features used in considering both hardware and requirements during design [18].

Structured features are built upon the function block feature by organizing hardware and software elements into patterns, making it easier for developers to understand and design them. Structured features are used in embedded software development to explicitly express the relationships between hardware components and software functions and between requirements and design [19]. Structured features act as intermediaries between the requirement and design domains, possessing a lower level of abstraction than requirements but a higher level of abstraction than design artifacts. Therefore, structured features are designed to contain both requirements and design elements.

As shown in Figure 2, the structured feature list L, which is a collection of structured features, comprises the structured features. The attributes of the structured feature model are designed to organically connect the requirements with the design. The structured feature model, denoted as SF_{id}, is defined as a 6-tuple; as shown in Figure 2, it consists of a requirement ID (RID), action (A), input data (ID), output data (OD), input event (IE), and output event (OE). An attribute represents the action that the structured feature intends to perform, whereas the flow of data and event attributes such as ID, OD, IE, and OE can be represented as attributes of the structured feature.



$$\mathrm{L} = \{\mathrm{SF}_{id} \mid id \in \mathbb{N}\},$$

$$\mathrm{SF}_{id} = (\mathbf{RID, A, ID, OD, IE, OE})$$

(1) RID is the identifier of a requirement.
(2) A refers to the action that must be performed.
(3) ID is the input data required to perform the action, consisting of a data name and hardware elements.
(4) OD is the output data produced after performing the action, consisting of a data name and hardware elements.
(5) IE is the event required for the action to be performed, consisting of an event name and hardware elements.
(6) OE is the event that indicates the action has been completed, consisting of an event name and hardware elements.

Figure 2. Attributes of structured feature model.

The SFCAD method formalizes the connection between the problem space and solution space through structured features [20]. As shown in Figure 3, the SFCAD process is divided into component identification, where the components comprising the CSA are identified; component interaction, which defines the interfaces between the identified components; and component specification, which specifies the components. The connections between structured features and design elements in each process were confirmed in [4]. In this study, we further analyze the mechanisms connecting these processes.

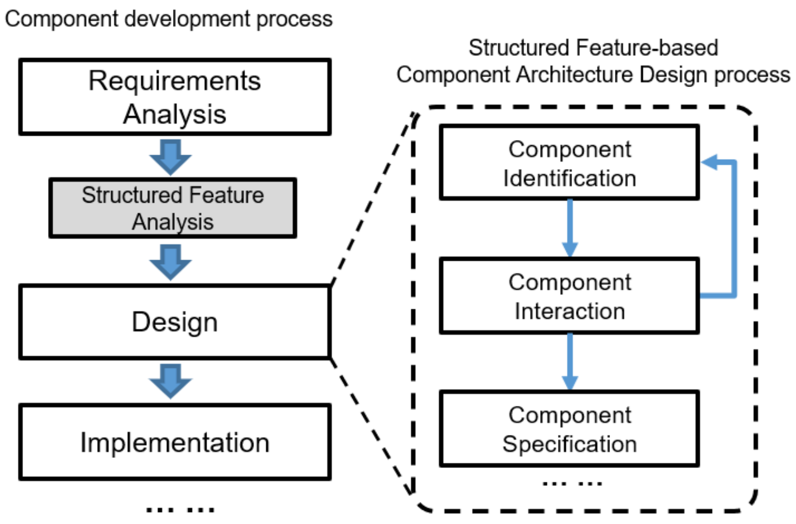
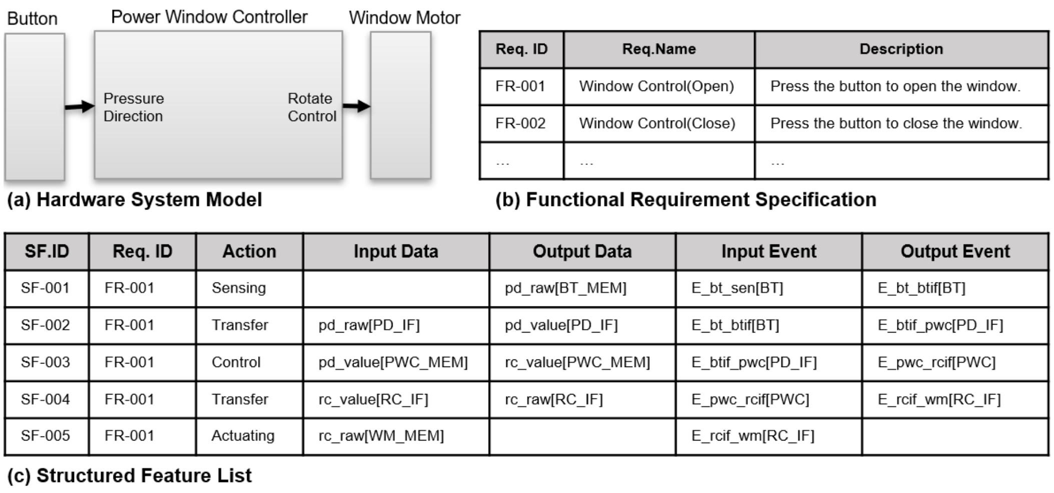

Component development process
Requirements Analysis
Structured Feature Analysis
Design
Implementation
… …
Structured Feature-based Component Architecture Design process
Component Identification
Component Interaction
Component Specification
… …

Figure 3. Structured feature-based component architecture design process.

Structured feature analysis is derived from the hardware system model and the functional requirements specification. The scope of the hardware system model encompasses those elements of hardware systems that are affected by embedded software. The format used to describe functional requirements may vary depending on the project or group. Below is an example of the hardware system model, the functional requirements specification, and the list of structured features produced from these sources.

Figure 4 presents an example of the materials used to generate a structured feature list and the resulting outputs. In (a), the hardware system model receives directional input through a button, which sends data for pressure direction to the power window controller. These data are then converted into rotate control data to open or close the window. The converted rotate control data are sent to the window motor, where they are transformed into a signal, allowing the window to open or close. In (b), the functional requirements specification is described in a brief format, with the designer arbitrarily adding identifiers to the requirements for structured feature analysis. In (c), the structured feature list derived from analyzing (a) and (b) is shown. In the case study presented in this research, the hardware system model and functional requirements specification are provided as shown and we demonstrate that design traceability can be maintained through the structured feature list as the design progresses.


Button
Power Window Controller
Window Motor
Pressure Direction
Rotate Control
(a) Hardware System Model
(b) Functional Requirement Specification
(c) Structured Feature List

Req. ID	Req.Name	Description
FR-001	Window Control(Open)	Press the button to open the window.
FR-002	Window Control(Close)	Press the button to close the window.
…	…	…

SF.ID	Req. ID	Action	Input Data	Output Data	Input Event	Output Event
SF-001	FR-001	Sensing		pd_raw[BT_MEM]	E_bt_sen[BT]	E_bt_btif[BT]
SF-002	FR-001	Transfer	pd_raw[PD_IF]	pd_value[PD_IF]	E_bt_btif[BT]	E_btif_pwc[PD_IF]
SF-003	FR-001	Control	pd_value[PWC_MEM]	rc_value[PWC_MEM]	E_btif_pwc[PD_IF]	E_pwc_rcif[PWC]
SF-004	FR-001	Transfer	rc_value[RC_IF]	rc_raw[RC_IF]	E_pwc_rcif[PWC]	E_rcif_wm[RC_IF]
SF-005	FR-001	Actuating	rc_raw[WM_MEM]		E_rcif_wm[RC_IF]	

Figure 4. An example of a structured feature produced from the structured feature analysis phase.

The structured features serve as elements that link the problem space to the solution space. In this research, structured features are used as a means to connect the functional requirements with the component architecture. To address the functional requirements derived from the problem space, we have applied a component architecture design method based on structured features. The component design process is then further analyzed using structured features, to examine how the structured features correspond to each stage.

3. Related Works

This section reviews existing research on requirements traceability, highlighting its significance in software engineering and the various methods developed to support it. In addition, the challenges and approaches specific to embedded software traceability are discussed, providing the necessary context for understanding the motivations behind the development of the SFCAD method.

3.1. Requirements Traceability

In software development, the requirements must first be analyzed, followed by design and implementation based on these requirements. Requirements traceability refers to the ability to trace and manage outputs produced during the development process in both directions across all stages [21]. Software with high requirements traceability offers several advantages, including increased reusability, maintainability, and reliability of software artifacts [22,23].

One of the most common methods for visualizing traceability is the use of a traceability matrix [22,24]. A traceability matrix is a traditional technique that helps to visualize how requirements are reflected in the design and implementation stages using a tabular format. This allows developers to effectively manage changes in requirements and verify test coverage. The matrix can be adjusted to focus on specific areas of interest by rearranging the rows and columns. Although this technique has been in use for a long time, it continues to be adapted and employed in various domains and contexts [25–27]. In this study, we use a traceability matrix to visualize the relationships between functional requirements and components, demonstrating how our proposed method supports traceability.

Approaches using Information Retrieval (IR) and Natural Language Processing (NLP) have been applied to requirements traceability because of their advantage in generating a large number of traceability links more quickly than manual expert-driven tracing [28–31].

However, general software traceability methods often do not account for the hardware dependencies inherent in embedded software, thereby limiting their direct applicability to embedded systems. Additionally, although the IR and NLP approaches offer the advantage of automating traceability, they still require preprocessing to filter relevant information and overcome language barriers, meaning that expert involvement is required to ensure accuracy [30,32,33]. This study proposes a design method in which structured features act as traceability links for the hardware dependencies of the embedded software. As this is a manual method, it can flexibly accommodate a wide range of inputs.

3.2. Requirements Traceability in Embedded Software

Requirements traceability plays a critical role in embedded software development by helping to identify the origin of safety-related requirements, clarify these requirements, and make it easier to understand the interrelationships between software artifacts, thereby facilitating the management of requirements changes [34].

For these reasons, standards related to embedded software development mandate requirements traceability. In the automotive software domain, relevant standards include ISO 26262 and the Automotive Software Process Improvement and Capability Determination (A-SPICE) [35]. ISO 26262 is an international standard for ensuring the functional safety of automotive electronic systems requiring traceability of embedded software requirements. This standard provides requirements related to requirements management, traceability, and safety analyses. A-SPICE is a process capability evaluation model aimed at improving the quality of automotive software development and processes [36]. A-SPICE uses traceability to demonstrate the implementation of requirements, managing risks and helping identify the impact of changes in requirements. Although these standards provide a framework for improving the processes and requirements necessary for the safe development of embedded software, they do not offer detailed guidelines on how to achieve traceability.

Embedded software design involves unique characteristics because systems must interact with the physical environment and because factors such as real-time performance, reliability, and hardware constraints must be considered. In this design process, requirements traceability is essential for verifying that all requirements are consistently reflected from the early design stages to the final implementation [37,38].

Numerous approaches for supporting traceability in embedded software development are based on model-driven methods. Wang et al. proposed a method using Model-Driven Development (MDD) to generate traceability between Natural Language Requirements (NLRs) and Architecture Analysis and Design Language (AADL) models [39]. Abdelahad et al. proposed a traceability approach that integrates SysML with Business Process Model and Notation (BPMN) and Decision Model and Notation (DMN) to support decision-making requirements [40]. In their approach, SysML was used to model certain aspects of the system, while process and decision-making activities were defined using the BPMN and DMN standards, respectively, providing traceability between different modeling methods. Intrigila et al. proposed a method for managing requirements in critical software, focusing on providing requirements specifications for software development, verification, and maintenance activities [41]. This method employs an integrated model using SysML, BPMN, and DMN, offering efficient requirements management techniques such as V&V (verification and validation) and traceability. Alenazi et al. proposed a mutation-driven method that generates mutants of state-machine diagrams and uses them to identify accurate traceability links in automated requirements traceability [42]. Their process includes model checking of safety requirements and identifying trace links by ensuring that the attributes of safety requirements are preserved in the mutants, which results in higher accuracy. Ahmadiyah et al. modeled the traceability between requirements and code through a property-listing task [43]. Their SeFea-Trace Conceptual Model (STCM) uses mathematical notation and a metamodel to enrich the information in the conceptual model based on software artifact properties, thereby establishing links between implementations.

Although advantageous for automation and intuitive understanding, model-based traceability for embedded software is often limited in its applicability because of the diverse methods used to model the requirements, as demonstrated by the results in [44,45]. Requirements are often modeled using text, goal-oriented models, or unspecified methods that restrict the context in which such approaches can be applied.

This study analyzes SFCAD, which is a method for designing embedded software architecture from a functional perspective, and demonstrates how SFCAD supports traceability from a requirements viewpoint. To illustrate this, we visualize the traceability matrix in a case study. After analyzing the structured features and generating a structured feature list, SFCAD proceeds with designer mapping components and structured features based on direct sensor-actuator patterns. During the generation of the structured feature list, the designer analyzes various types of requirements.

4. Proposed Method

The SFCAD method utilizes sensor–actuator patterns in the component architecture generated based on action attributes in the structured feature list. The structured features are then mapped to the component architecture. These are remapped in response to changes in the architecture, driving the connection between the requirements and component architecture elements. In this section, we analyze how structured features serve as intermediaries between requirements and design elements in SFCAD by using them to identify and map elements of both requirements and component architecture, thereby demonstrating traceability through structured features.

4.1. Location of Structured Features in the Process

Structured features consist of a requirement ID, action attributes, data attributes, and event attributes, each associated with software requirement elements and hardware elements. The software requirement elements are extracted from refined software requirements specifications, whereas hardware elements are extracted from the hardware system model. Traceability relationships can be established by reflecting the attributes of structured features in the design of components.

Structured features are derived by analyzing the relationship between the software requirements specification (an output of the requirements analysis phase) and the hardware system model, which represent the hardware systems that the software influences. The set of structured features produced iin this manner takes the form of a structured feature list, which helps to define the scope of the software system by modeling the necessary data flow for software development. During the design phase, the attributes of the structured features in the structured feature list are reflected in the design, thereby establishing a connection between the requirements and design through the structured features.

Figure 5 shows the component development process and the required outputs in this study. In the component development process applying SFCAD, the structured feature analysis phase is positioned between the requirements analysis phase and the design phase. Structured feature modeling is performed during the structured feature analysis phase, during which the attribute values of the structured feature model that constitute the structured feature list are extracted from the requirements specification and the hardware system model.

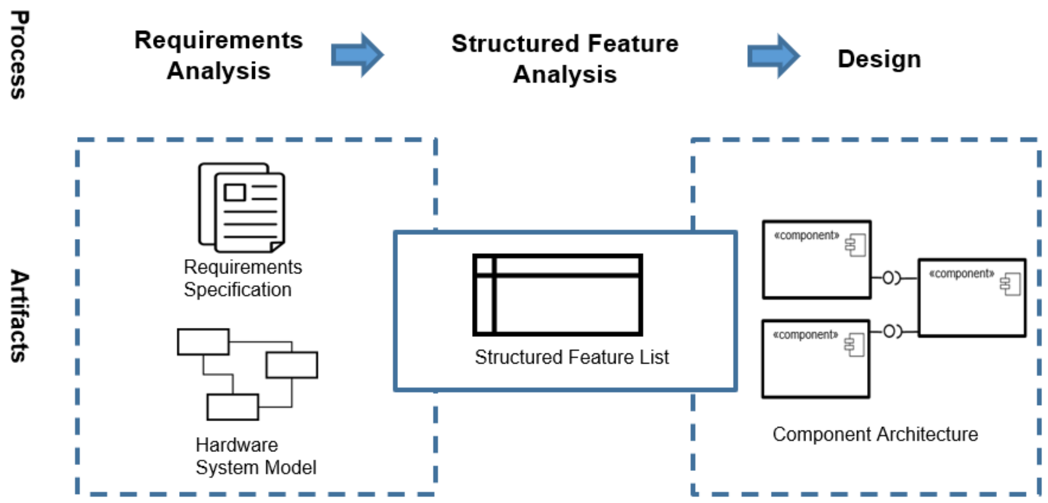


Figure 5. Location of the structured feature list in the component-based development process.

4.2. Correspondence Between Components and Structured Features

In SFCAD, there is a component identification phase that first identifies the initial components by applying the sensor–actuator pattern and then remaps the structured features that were previously associated with the old components to the newly designed components in order to reflect quality attributes.

4.2.1. Initial Component Identification by Structured Feature

In SFCAD, the component identification phase first identifies the initial components by applying the sensor–actuator pattern. The set of components C that makes up the component architecture is defined as $C = \{c_1, c_2, \ldots\}$, where each element c denotes an individual component in set C.

The types of components C_0 that constitute the initial component architecture A_0 comprise the initial sensor c_{sensor}, initial actuator $c_{actuator}$, initial controller $c_{controller}$, and initial interface $c_{interface}$ components, which are defined as follows:

$$C_0 \in \left\{c_{sensor}, c_{controller}, c_{actuator}, c_{interface}\right\} \tag{1}$$

- c_{sensor}: InitialSensorComponent—the component that receives external input.
- $c_{controller}$: InitialControllerComponent—the component that processes logical commands.
- $c_{actuator}$: InitialActuatorComponent—the component that executes control commands.
- $c_{interface}$: InitialInterfaceComponent—the component that converts data.

Each of these components is categorized based on a component architecture that follows a sensor–actuator pattern. The roles of the components are as follows: c_{sensor} receives data into the system; $c_{controller}$ processes the received data; $c_{actuator}$ executes the processed data; finally, $c_{interface}$ connects the data flow among the three components and is placed between them.

Structured features are mapped to the roles of initial components as follows.

$$M_0 = map(L, C_0)\begin{cases} c_{sensor} & \text{if } SF_{id.Action} == \text{Sensing} \\ c_{controller} & \text{if } SF_{id.Action} == \text{Control} \\ c_{actuator} & \text{if } SF_{id.Action} == \text{Actuating} \\ c_{interface} & \text{if } SF_{id.Action} == \text{Transfer} \end{cases} \tag{2}$$

The mapping information between the components, interfaces, and architecture features in the component architecture is represented by M. Here, M is a set of pairs $\langle SF_{id}, c_m \rangle$ between structured features and components, or pairs $\langle SF_{id}, I_n \rangle$ between structured features and interfaces. In this case, the number id of SF ranges from zero to $\leq$

(the number of structured features in L), m ranges from zero to $\leq$ (number of components in C), and n ranges from zero to $\leq$ (number of interfaces in CSA).

The initial mapping information M_0 is defined by a function $map(L, C_0)$. The function $map(SF, C_0)$ is calculated from the structured feature list L and initial set of components C_0. The action attribute of the structured features is SF_i in L, and can take four values that represent the actions performed by the components categorized by patterns. Based on the action attribute value, a structured feature is mapped to its corresponding initial component. If the value of $SF_{id.Action}$ is "Sensing", then the structured feature is mapped to the sensor component c_{sensor}, as it is responsible for reading data. If the value is "Control", then it is mapped to the controller component $c_{controller}$, which processes the data. If the value is "Actuating", then it is mapped to the actuator component $c_{actuator}$, which executes the processed data. Finally, if the value is "Transfer", then it is mapped to the interface component $c_{interface}$, which transfers data. The mapping information M_0 can be used to provide traceability for future analyses.

The method of identifying the initial components and mapping structured features establishes a connection between the initial component architecture and the requirements through the sensor–actuator pattern. The action attribute of a structured feature represents the performed action, while the unit of action corresponds to the unit of action performed by each component in the pattern. Similarly, the initial components are divided into individual components, each corresponding to a specific action as defined by the action attribute of the structured feature.

4.2.2. Remapping of Structured Features and Components

As shown in Figure 3, the component identification phase in SFCAD can be performed iteratively. As the component-based software architecture is modified, the structured features previously mapped to the initial components are remapped to the components of the updated component-based software architecture. This iterative component decomposition and integration process includes modifications to the initial component-based software architecture. These modifications can be carried out by applying the Attribute-Driven Design (ADD) method [46] multiple times based on important quality attributes in the target embedded system.

The mapping between structured features and components is redefined as the ADD process progresses. Remapping of the structured features occurs by redefining the relationship between component C_i in the existing component architecture and components C_{i+1} in the updated architecture after an ADD iteration. The remapping function $remap(M_i, C_i, C_{i+1})$ is defined as follows:

$$M_{i+1} = remap(M_i, C_i, C_{i+1}). \tag{3}$$

The value of i in the above equation ranges from the initial component identification described in Section 4.2.1 to the end of ADD. The function $remap(M_i, C_i, C_{i+1})$ is executed by checking the component decomposition conditions and remapping the structured features using the following steps:

(1) **Check component decomposition conditions.** During the ADD process, the designer produces the i-th and i + 1-th design results and checks the relationships between the components in the existing and new results, defined as follows:

$$\forall C_{ic} \in C_i, \exists C_{i+1c} \in C_{i+1} \mid C_{ic} \xrightarrow{decomposedinto} C_{i+1c}. \tag{4}$$

For component set C_i in the i-th component architecture and updated component set C_{i+1} in the i + 1-th architecture, it is determined whether component C_{ic} in C_i has been decomposed into components C_{i+1c} in C_{i+1}.

(2) **Remapping of structured features.** If a component is determined to have been decomposed, the structured feature $SF_{C(i)_{c_j}}$ mapped to the j-th component in the i-th

component architecture is remapped to the k-th component $C_{(i+1)_{c_k}}$ in the updated i + 1-th component architecture using the algorithm shown below.
First, in line 4 of Algorithm 1, Equation (4) is applied to determine $SF_{C(i)_{c_j}}$ mapped to $C_{(i)_{c_j}}$ in M_i. The structured feature is then remapped to the decomposed component. The relationship between the remapped structured feature and the component is added to M_{i+1} and the remapping is returned, allowing the designer to maintain the traceability information.

Algorithm 1 `Update mapping info into` M_{i+1}

```
Require: i-th mapping information $M_i$, i-th component set $C_i$, (i+1)-th component set $C_{i+1}$
Ensure: i-th mapping information $M_{i+1}$
1: for all j-th $C_{(i)_{c_j}}$ in $C_i$ do
2:   for all k-th $C_{(i+1)_{c_k}}$ in $C_{i+1}$ do
3:     if $C_{(i)_{c_j}} \xrightarrow{decomposedinto} C_{(i+1)_{c_k}}$ then
4:       $SF_{C(i+1)_{c_k}} \leftarrow SF_{C(i)_{c_j}}$
5:       Put $SF_{C(i+1)_{c_k}}$'s mapping information into $M_{i+1}$
6:     end if
7:   end for
8: end for
9: return $M_{i+1}$
```

Through this process, the structured features are remapped to the updated components, allowing the initial relationship between the structured features and initial components to be redefined. By redefining the relationships between the components and structured features, natural traceability between requirements and components is achieved. Both the initial components and structured features share a common foundation based on the actions performed within the sensor–actuator pattern. The initial components are defined according to the actions performed by the components divided through the sensor–actuator pattern, whereas the structured features define actions based on the interactions between hardware elements within the sensor–actuator pattern to satisfy the requirements. The structured features concretize the functions necessary to meet the system requirements, while the initial components are the basic units that implement these functions. Thus, the initial components are adjusted to accommodate the detailed requirements and actions specified by the structured features, thereby ensuring functional consistency and integration across the entire system. Because the actions defined by the structured features are directly linked to the operations of the initial components, the interactions of the system can be clearly understood.

4.3. Structured Features for Allocating Interface Operations

In SFCAD, components are identified during the component identification phase, then operations are assigned to the interfaces during the component interaction phase to define the data flow between components. The process of assigning operations to the component interfaces is performed by analyzing the data-related attributes of the structured features. This assignment occurs between component c_P, with a provided interface representing the methods or functionalities offered to the outside, and component c_R, with a required interface representing the methods or functionalities that the component needs from the outside to operate correctly.

(1) **Check whether the data inputs and outputs match between component pairs.** The following equation determines whether the Input Data (ID) and Output Data (OD) of the structured features SF_{c_P} and SF_{c_R} mapped to components $\mathrm{c_P}$ and $\mathrm{c_R}$ in the

component architecture C_i match. In case of a match, data exchange occurs between the two components.

$$OD(SF_{c_P}) == ID(SF_{c_R}) \implies exchange(c_P, c_R) \tag{5}$$

To assign operations related to data flow via interfaces between components, the ID and OD attributes of the structured features are compared. Structured features categorized by the sensor–actuator pattern either receive or output data. Based on this mechanism, data flow operations are assigned to the interfaces between the mapped components. If $OD(SF_{c_P})$ and $ID(SF_{c_R})$ match, this signifies a data exchange between the two components.

(2) **Operation Allocation.**The function that sends data from c_P through its provided interface and receives data in c_R through its required interface is defined as $exchange(c_P, c_R)$. This function describes the process by which the data provided by c_P are received by c_R.
Equation (5) is applied to line 6 of the algorithm in Algorithm 2. In order for data exchange to occur, both the sent and received data must be defined. In this case, the exchanged data are referred to as *ProvidedData*, which are the data sent from c_P and received by c_R. This function is implemented using the $exchange(c_P, c_R)$ function and assigned to the interface Interface(c_P, c_R) between c_P and c_R.

Algorithm 2 `Operation allocation into Interface betwee n` c_P `and` c_R

```
Require: $c_P, c_R$
Ensure: An operation is assigned to the interface between $c_P$ and $c_R$, Updated $M_i$ with
    interface operation's allocation information
1: Function $exchange(c_P, c_R)$ :
2:    ProvidedData $= c_P$.provideData()
3:    $c_R$.receiveData(ProvidedData)
4: if $OD(SF_{c_P}) == ID(SF_{c_R})$ then
5:    Interface$(c_P, c_R) \xleftarrow{allocate} exchange(c_P, c_R)$
6:    Put $SF_{c_P}$ and $SF_{c_R}$ mapping information into $M_i$
7: end if
```

Operations can be assigned to the interfaces between the components in the same manner, even for components that are re-identified during the component identification phase. Thus, the actions of structured features affect not only the components that make up the competent architecture but also the assignment of operations to the interfaces that enable data interaction between components.

SFCAD uses the structured feature list generated via structured feature analysis to identify the initial components and allocate the necessary operations to the interfaces that connect the components. These structured features provide the foundation for the initial component architecture design during the component identification phase and contribute to the natural derivation of data flow between the components during the component interaction phase.

4.4. Construction of a Traceability Matrix

In this study, the artifacts to be connected are the functional requirements and the components and interfaces that comprise the component architecture. We use the traceability matrix method to demonstrate the traceability between these two elements. The columns of the traceability matrix used to visualize the traceability between the two elements are structured as shown in Table 1.

Table 1. Columns of the traceability matrix.

Column Name	Description
Req. ID	Identifier of functional requirement
SF. ID	Identifier of structured feature
Component	Name of component-forming component architecture
Interface	Name of the interface forming the component architecture

Additional columns related to the designer's interests can be added or removed. In this study, the traceability matrix shown in Table 1 focuses on the traceability between two elements: the functional requirements, represented by the 'Req. ID' column, and the components and interfaces that make up the component architecture, represented by the 'Component' and 'Interface' columns.

To generate the traceability matrix TM_i corresponding to the i-th design result, a structured feature list L and mapping information M_i are required. We generate a traceability matrix using T and L as inputs with the following algorithm shown below.

In lines 2–4 of the algorithm in Algorithm 3, the attributes RID and SF_{id} of SF_{id} are entered into the 'Req. ID' and 'SF. ID' rows of TM_i, respectively, whereas the mapping information pairs in M_i are added to the component and interface columns corresponding to each structured feature. The difference is that line 2 extracts information from L, whereas lines 3–4 extract information from M_i.

Algorithm 3 `Generating` TM_i

```
Require: Structured Feature List $L$, $i$-th mapping information $M_i$
Ensure: $i$-th Traceability Matrix $TM_i$
1: for all id-th $SF_{id}$ in $L$ do
2:     $(Req.ID, SF.ID) \leftarrow \langle SF_{id.RID}, SF_{id} \rangle$
3:     $(SF.ID, Component) \leftarrow \langle SF_{id}, c_m \rangle$
4:     $(SF.ID, Interface) \leftarrow \langle SF_{id}, I_n \rangle$
5: end for
6: return $TM_i$
```

The structured features serve as intermediaries that connect the requirements and component elements. By adding a column for the identifiers of the structured features in the traceability matrix, designers can visually confirm the connections between two elements. More detailed relationships can be identified by referencing the structured feature identifiers in the structured feature list, enabling a link between the traceability matrix and structured feature list. The list of structured features also includes attributes related to actions and hardware elements, allowing designers to verify the actions or hardware elements of the components in the component architecture. By linking the traceability matrix with the structured feature list, designers can effectively trace the relationships between requirements and component architecture elements.

4.5. Ensuring Traceability of Requirements and Component Architecture Elements

4.5.1. Tracing Between Structured Features and Requirements

Software requirement specifications and hardware system models are necessary to generate the structured features. Figure 6 shows the relationship between requirements analysis artifacts and the attributes of the structured feature model.

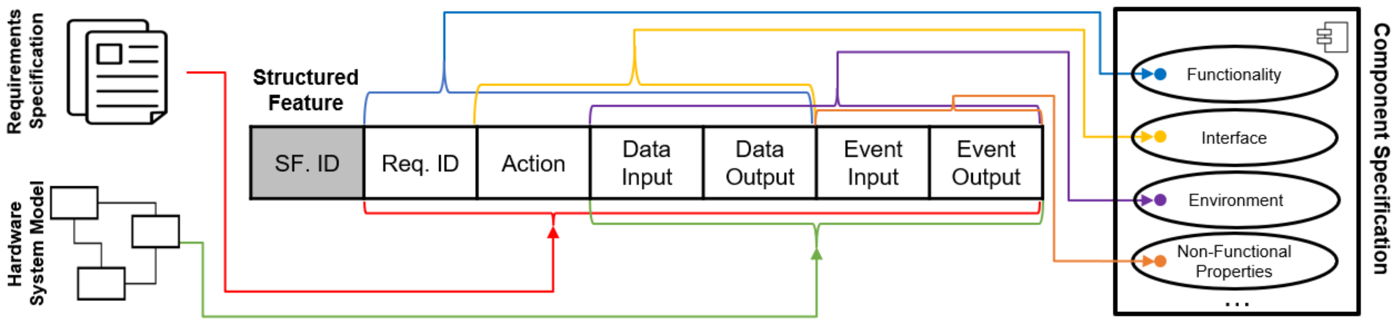


Figure 6. Association between a structured feature and artifacts.

The software requirements specification and hardware system model together directly or indirectly compose the attributes of the structured features. Figure 6 shows the parts of the structured feature model composed of these elements. The requirements specification affects all attributes of the structured feature model, and contains not only functional requirements but also non-functional requirements, as well as foundational information such as the scope of the software being designed and developed. This information influences the entire development process, including design and implementation. To satisfy the requirements, structured features can represent this information either directly, such as in Req. ID attributes that specify the scope of the software, or indirectly, such as through input and output data and events. The hardware system model is explicitly represented by the input and output data and the events of the structured feature model. The existing information is expressed as attribute values within the structured feature model, creating a connection between the artifacts of the requirements analysis phase and the outputs of the structured feature analysis phase.

A structured feature SF is defined as $SF_{id} = (RID, A, ID, OD, IE, OE)$, as shown in Figure 2. The attributes of a structured feature can be classified into those related to requirements (RID), actions (A), data (ID and OD), and events (IE and OE).

During the structured feature analysis phase, the designer refines the software requirement specifications into software requirement elements to enable the extraction of those elements required for the structured features. These software requirement elements are then assigned as values to each structured feature attribute. The hardware system model is refined into hardware elements and assigned as values to the action-, data-, and event-related attributes of the structured feature. Through this assignment, both the software requirements and the hardware system model are reflected in the resulting structured feature. This creates links between the attributes of the structured feature and the requirements, which can then be used to establish traceability between the requirement RID and structured feature, as the RID is assigned to the attributes of the structured feature.

4.5.2. Tracing Between Structured Features and Component Architecture

To proceed with the structured feature analysis phase in the SFCAD process, outputs from the requirements analysis phase such as the functional requirements specification and hardware system model are required. Based on these outputs, the structured feature list is analyzed and produced during the structured feature analysis phase.

The component architecture generated by SFCAD in the component-based development process is specified for component reuse. To ensure that the component can be reused to reflect the developer's intent, the component specifications must include the information necessary for reuse, such as the functionality and interfaces of the component. Figure 6 shows the relationship between the attributes of the structured features and the elements of component specification.

During the component identification and component interaction phases of SFCAD, the designer uses the action attributes of the structured features from the structured feature list to generate the initial component architecture by applying a sensor–actuator pattern. Both the initial component architecture and the modified component architecture may change based on the quality attributes or business requirements identified during the requirements analysis phase. When a change occurs, the existing structured features and elements of the component architecture are remapped accordingly. By including structured features in the generation of the component architecture, traceability from the requirements to the component architecture is ensured.

The attributes of the structured feature model provide the essential information for specifying each component. The requirements specification affects all the attribute values of the structured features, and these values directly influence the construction of the component architecture. The functionality attribute represents the functionality of the component, providing information about the actions the component performs. The component identity-structured features establish a connection between the structured features and components whereby the attributes of the structured features can be associated with the functionality of the components. The action and data attributes of the structured features influence the assignment of operations to interfaces during the component interaction phase, which is based on the analysis of data and transmission flows.

Embedded software in hardware systems operates with a specific purpose. The environment attribute refers to the environment in which the component is expected to operate, reflecting the hardware dependencies of embedded software. The hardware elements represented by the data and event attributes of the structured features describe this environment and link the structured features to component specifications. The event attribute of the structured feature is used to represent its states or events. Non-functional properties such as the performance or execution order of the component can be associated with the event attributes of structured features. Thus, each attribute of the structured feature model influences various aspects of the component architecture design and plays a crucial role in connecting requirements to the final component architecture.

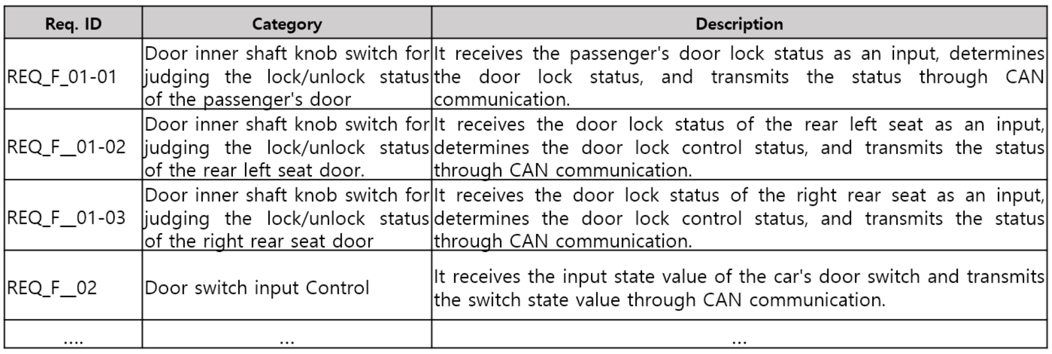
5. Case Study

In this section, we demonstrate the SFCAD process and verify its application to structured features throughout the process. To validate the capability of structured features in establishing connections between software functional requirements and component architecture, we conducted a case study and present the resulting artifacts. The case study addresses the design of a component architecture for a vehicle-door control system. Although the requirements specification comprised various requirements, in this case the designer assumed that the functional requirements and hardware system modeling from the requirements specification had been completed and that structured feature analysis had been conducted based on this information.

5.1. Target System for Structured Feature Analysis

The target system for the application of structured features was a vehicle door control system that included functional requirements such as checking whether the vehicle doors were locked or unlocked. To demonstrate the traceability of functional requirements and hardware elements to component architecture elements within SFCAD, we present the software functional requirements necessary for system development along with the hardware system model that influences the software development process. For this case study, 15 functional requirements were selected from among 31, focusing on those related to the hardware elements identified in the system specifications as well as those impacted by the hardware system.

Figure 7 shows a subset of the functional requirements related to the knob switch. The functional requirements are described alongside the hardware system necessary for the proper operation of the requirements, differentiating between the two. Among these, hardware elements that were ambiguous or unrelated to the selected functional requirements were excluded from the hardware system model and elements related to security were partially renamed.



Req. ID	Category	Description
REQ_F_01-01	Door inner shaft knob switch for judging the lock/unlock status of the passenger's door	It receives the passenger's door lock status as an input, determines the door lock status, and transmits the status through CAN communication.
REQ_F__01-02	Door inner shaft knob switch for judging the lock/unlock status of the rear left seat door.	It receives the door lock status of the rear left seat as an input, determines the door lock control status, and transmits the status through CAN communication.
REQ_F__01-03	Door inner shaft knob switch for judging the lock/unlock status of the right rear seat door	It receives the door lock status of the right rear seat as an input, determines the door lock control status, and transmits the status through CAN communication.
REQ_F__02	Door switch input Control	It receives the input state value of the car's door switch and transmits the switch state value through CAN communication.
....	...	...

Figure 7. Functional requirements.

Based on the overall system requirements specification, we analyzed the relationships between the hardware elements associated with the software functional requirements and derived the hardware system model by applying the sensor–actuator pattern, as shown in Figure 8. Centered around the Integrated Control Unit (ICU), the hardware system model consists of an H/W switch section, Controller Area Network (CAN), and H/W actuator section. In the hardware system model, the hardware switch section represents the components that send signals to the ICU, which is equipped with software, the CAN section serves as the communication channel that receives the data processed by the ICU, and the hardware actuator section represents the components that control the external environment. These elements are interconnected to control the system. The hardware system model is composed of hardware switch sensors, actuators, and network communication-related elements, making it a suitable example for illustrating the structure of software embedded in vehicles.

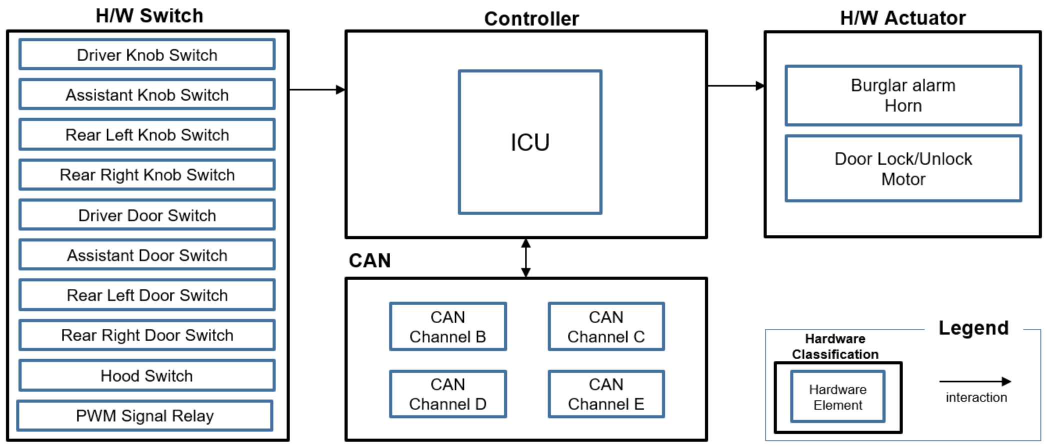
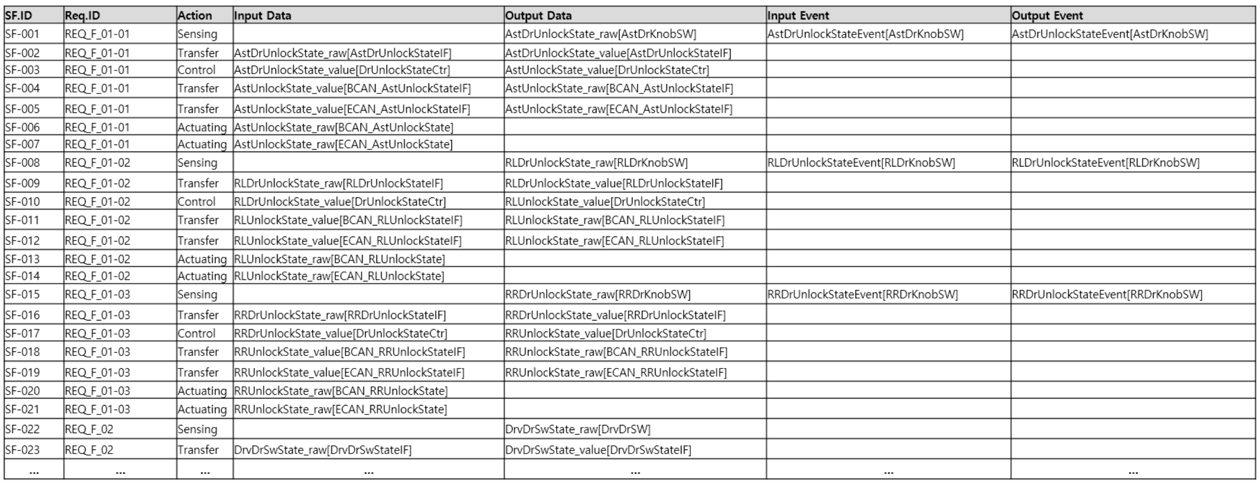


Figure 8. Hardware system model.

Using the provided requirements and hardware system model, we conducted a structured feature analysis and generated a structured feature list. The structured feature list from the case study consisted of 106 structured features, a subset of which is shown in Figure 9. As in Figure 2, each attribute of the structured feature is represented by a column in the structured feature list. In Figure 6, the software functional requirements are linked to RID and the hardware system elements are connected to ID, OD, IE, and OE, indicating that the analysis has been conducted.



SF.ID	Req.ID	Action	Input Data	Output Data	Input Event	Output Event
SF-001	REQ_F_01-01	Sensing		AstDrUnlockState_raw[AstDrKnobSW]	AstDrUnlockStateEvent[AstDrKnobSW]	AstDrUnlockStateEvent[AstDrKnobSW]
SF-002	REQ_F_01-01	Transfer	AstDrUnlockState_raw[AstDrUnlockStateIF]	AstDrUnlockState_value[AstDrUnlockStateIF]		
SF-003	REQ_F_01-01	Control	AstDrUnlockState_value[DrUnlockStateCtr]	AstUnlockState_value[DrUnlockStateCtr]		
SF-004	REQ_F_01-01	Transfer	AstUnlockState_value[BCAN_AstUnlockStateIF]	AstUnlockState_raw[BCAN_AstUnlockStateIF]		
SF-005	REQ_F_01-01	Transfer	AstUnlockState_value[ECAN_AstUnlockStateIF]	AstUnlockState_raw[ECAN_AstUnlockStateIF]		
SF-006	REQ_F_01-01	Actuating	AstUnlockState_raw[BCAN_AstUnlockState]			
SF-007	REQ_F_01-01	Actuating	AstUnlockState_raw[ECAN_AstUnlockState]			
SF-008	REQ_F_01-02	Sensing		RLDrUnlockState_raw[RLDrKnobSW]	RLDrUnlockStateEvent[RLDrKnobSW]	RLDrUnlockStateEvent[RLDrKnobSW]
SF-009	REQ_F_01-02	Transfer	RLDrUnlockState_raw[RLDrUnlockStateIF]	RLDrUnlockState_value[RLDrUnlockStateIF]		
SF-010	REQ_F_01-02	Control	RLDrUnlockState_value[DrUnlockStateCtr]	RLUnlockState_value[DrUnlockStateCtr]		
SF-011	REQ_F_01-02	Transfer	RLUnlockState_value[BCAN_RLUnlockStateIF]	RLUnlockState_raw[BCAN_RLUnlockStateIF]		
SF-012	REQ_F_01-02	Transfer	RLUnlockState_value[ECAN_RLUnlockStateIF]	RLUnlockState_raw[ECAN_RLUnlockStateIF]		
SF-013	REQ_F_01-02	Actuating	RLUnlockState_raw[BCAN_RLUnlockState]			
SF-014	REQ_F_01-02	Actuating	RLUnlockState_raw[ECAN_RLUnlockState]			
SF-015	REQ_F_01-03	Sensing		RRDrUnlockState_raw[RRDrKnobSW]	RRDrUnlockStateEvent[RRDrKnobSW]	RRDrUnlockStateEvent[RRDrKnobSW]
SF-016	REQ_F_01-03	Transfer	RRDrUnlockState_raw[RRDrUnlockStateIF]	RRDrUnlockState_value[RRDrUnlockStateIF]		
SF-017	REQ_F_01-03	Control	RRDrUnlockState_value[DrUnlockStateCtr]	RRUnlockState_value[DrUnlockStateCtr]		
SF-018	REQ_F_01-03	Transfer	RRUnlockState_value[BCAN_RRUnlockStateIF]	RRUnlockState_raw[BCAN_RRUnlockStateIF]		
SF-019	REQ_F_01-03	Transfer	RRUnlockState_value[ECAN_RRUnlockStateIF]	RRUnlockState_raw[ECAN_RRUnlockStateIF]		
SF-020	REQ_F_01-03	Actuating	RRUnlockState_raw[BCAN_RRUnlockState]			
SF-021	REQ_F_01-03	Actuating	RRUnlockState_raw[ECAN_RRUnlockState]			
SF-022	REQ_F_02	Sensing		DrvDrSwState_raw[DrvDrSW]		
SF-023	REQ_F_02	Transfer	DrvDrSwState_raw[DrvDrSwStateIF]	DrvDrSwState_value[DrvDrSwStateIF]		
...	...	...	...	...	...	...

Figure 9. Structured feature list.

In this case study, the application of structured features assumes that the structured feature analysis has already been completed and that a structured feature list has been generated. Thus, the pairs $SF_{id.RID}, SF_{id}$, that is, REQ_F_01-01, SF-001 and REQ_F_01-01, SF-002 in Figure 9, have already been derived. Therefore, the information for mapping requirements and structured features is already available in the structured feature list, which is used to create the traceability matrix.

5.2. Connection Between Structured Features and Component Architecture

This case study illustrates the process of connecting structured features with a component architecture visualized through a traceability matrix. Mapping between the structured features and initial components is performed by identifying the action value of the structured feature and linking it to the initial components. Figure 10 shows the results of mapping the structured features SF-001 to SF-021 derived from the functional requirement REQ_F_01 to their corresponding initial components.

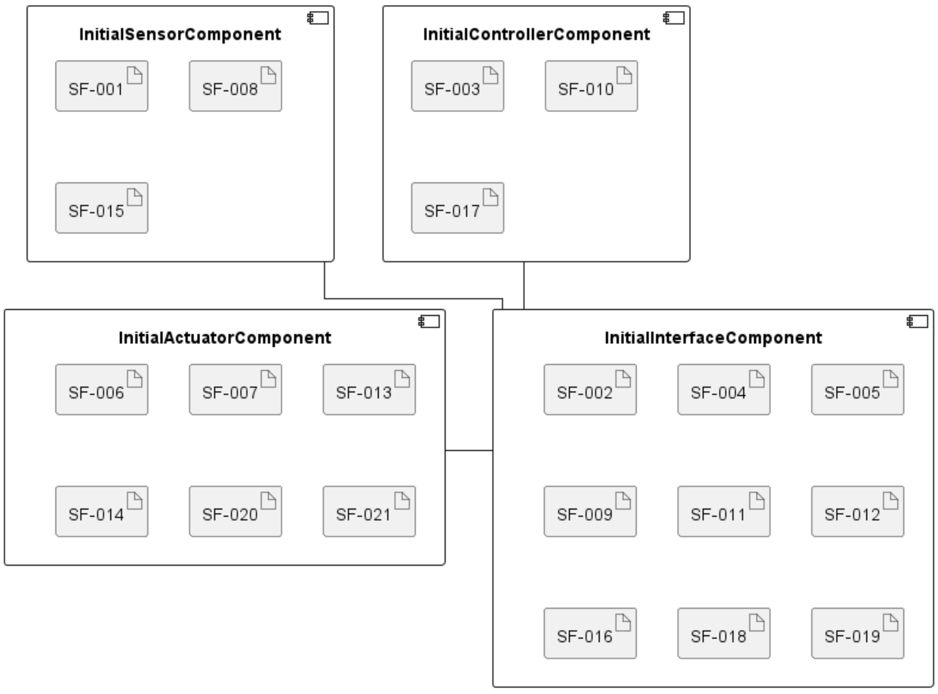


Figure 10. Mapping of initial components and structured features related to REQ_F_01.

'InitialSensorComponent' is a component that receives external stimuli into the system, and the structured features with the action value of "Sensing" are mapped to it. In the structured features related to REQ_F_01 in Figure 9, the features with "Sensing" values are SF-001, SF-008, and SF-015. These features are mapped to 'SensorComponent', whereas structured features with other action values are mapped to their respective initial components.

In the case study, the software architecture was designed to facilitate software system modifications based on changes in the hardware elements that constitute the system [47]. 'InitialSensorComponent' is responsible for receiving all the stimuli for the target system. However, as shown in Figure 11, hardware elements that receive external stimuli exist in various forms, such as switches and CAN channels. In order to efficiently accommodate changes in hardware elements and improve modifiability, software functionality should be separated according to hardware functionality. Figure 11 depicts the portion of the architecture related to splitting REQ_F_01 to reduce the cost and time required to implement changes when modification requirements arise in the software system.

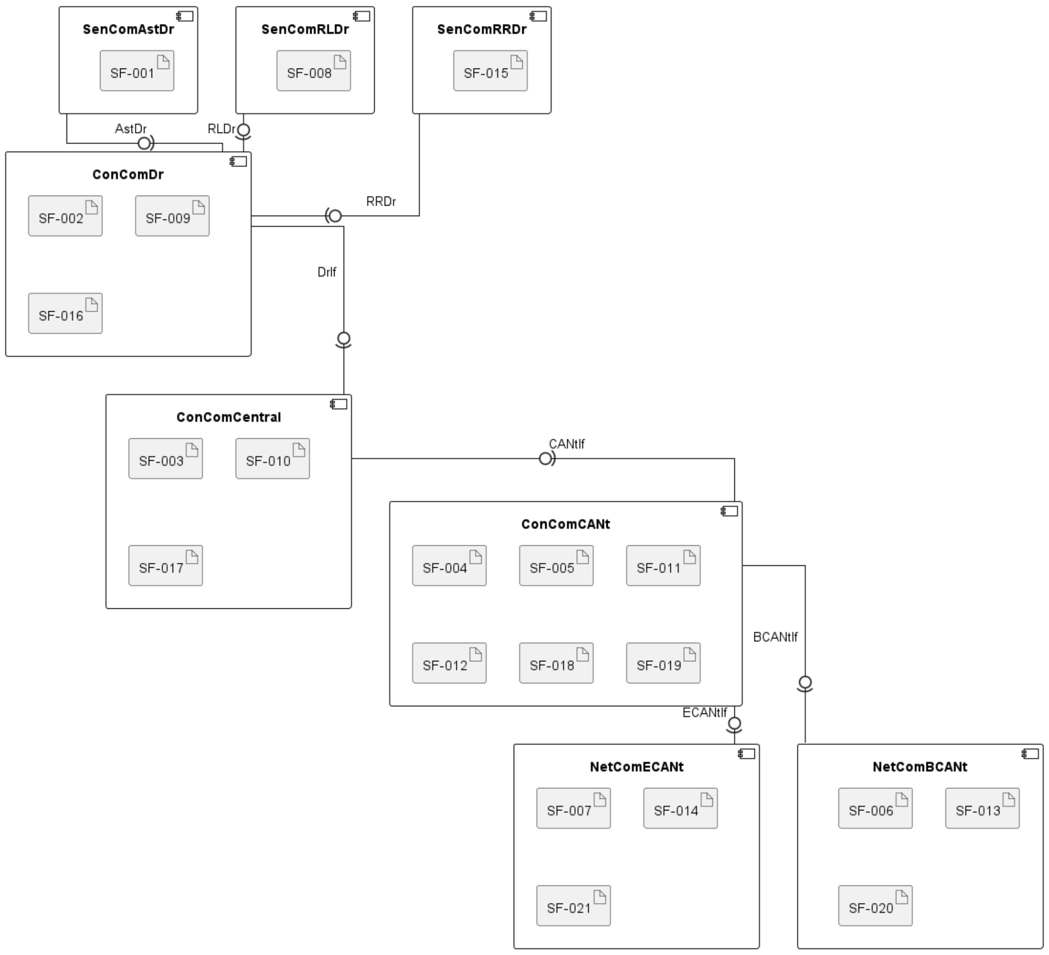


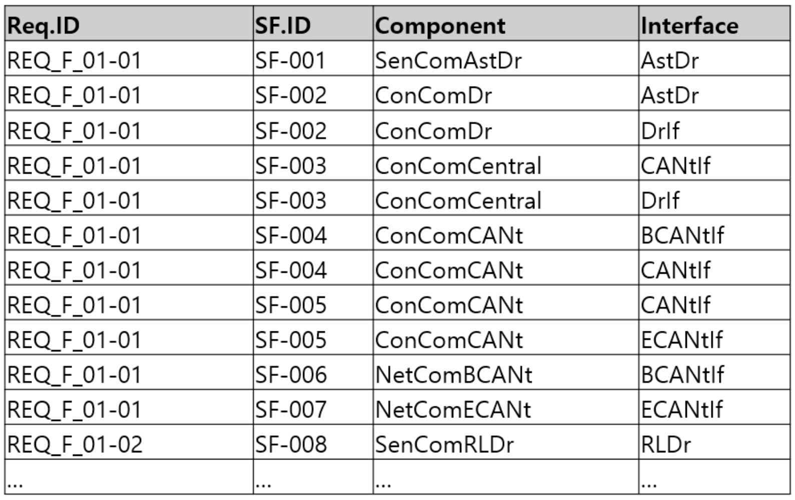
Figure 11. Mapping of components and structured features related to REQ_F_01.

The structured features initially mapped in Figure 10 are remapped to the components in the modified architecture as the component architecture is updated. As a result, 'InitialSensorComponent' is divided according to the position of the vehicle's door sensors, as follows: 'SenComAstDr' handles the function of the assistant knob switch; 'Sen-ComRLDr' handles the function of the rear-left knob switch; and 'SenComRRDr' handles the function of the rear-right knob switch. Each component receives data from the knob switch at the respective position in the vehicle. 'InitialInterfaceComponent' transforms the data received by 'InitialSensorComponent' into a format that 'InitialControllerComponent' can process and that other components such as 'InitialActuatorComponent' can receive. Therefore, it is divided according to the types of data that are processed. Structured features previously mapped to the original components are remapped using the structured feature-to-component remapping method. For example, because the output data value of SF-001 mapped to 'SenComAstDr' (Astunlockstate_raw) matches the input data value of SF-002 mapped to 'ConComDr', an operation is assigned to the interface Astdr between 'SenComAstDr' and 'ConComDr' using the method for assigning operations to the interfaces in SFCAD. This operation facilitates the transfer of data corresponding to 'Astunlockstate_raw' from 'SenComAstDr' to 'ConComDr'.

5.3. Traceability Matrix

The design results from SFCAD are represented using the traceability matrix described in Section 4.4 to illustrate the relationships between artifacts. Figure 12 shows the traceability matrix depicting the relationships between the structured features, components, and interfaces related to requirement REQ_F_01-01 from the design process conducted in the case study.

In Figure 12, SF-001 is mapped to 'SenComAstDr'. It can be observed in Figure 9 that an operation needs to be assigned to the AstDr interface. These trace relationships are visualized using the traceability matrix, which has of four columns: Req.ID, SF.ID, Component, and Interface. Using the traceability matrix, it was possible to visually confirm the traceability between the elements.

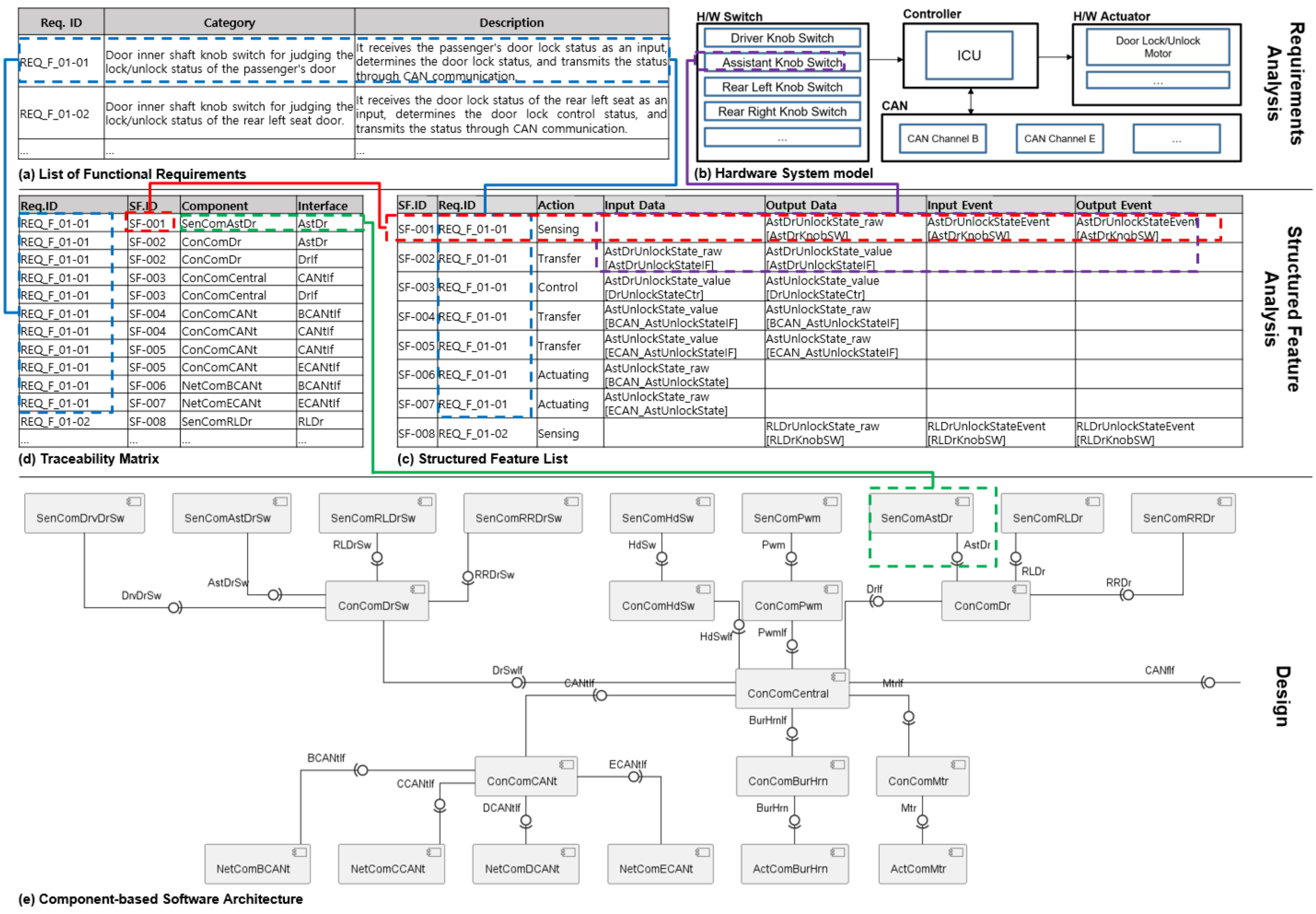


Req.ID	SF.ID	Component	Interface
REQ_F_01-01	SF-001	SenComAstDr	AstDr
REQ_F_01-01	SF-002	ConComDr	AstDr
REQ_F_01-01	SF-002	ConComDr	DrIf
REQ_F_01-01	SF-003	ConComCentral	CANtIf
REQ_F_01-01	SF-003	ConComCentral	DrIf
REQ_F_01-01	SF-004	ConComCANt	BCANtIf
REQ_F_01-01	SF-004	ConComCANt	CANtIf
REQ_F_01-01	SF-005	ConComCANt	CANtIf
REQ_F_01-01	SF-005	ConComCANt	ECANtIf
REQ_F_01-01	SF-006	NetComBCANt	BCANtIf
REQ_F_01-01	SF-007	NetComECANt	ECANtIf
REQ_F_01-02	SF-008	SenComRLDr	RLDr
...	...	...	...

Figure 12. Traceability matrix related to REQ_F_01-01.

5.4. Connection Between Artifacts

The functional requirements for the automotive door control software and related hardware elements form the foundational elements for design through structured feature analysis. Figure 13 shows the artifacts used in the case study, including the functional requirements list, hardware system model, structured feature list, traceability matrix, and component-based software architecture related to REQ_F_01-01. The artifacts in Figure 13 consist of (a) the functional requirements list, (b) the hardware system model, which is an output of the requirements analysis phase, (c) the structured feature list, which is the output from the structured feature analysis phase, and (d) the component-based software architecture, which is the output from the design phase. Additionally, the trace relationships between the outputs of each phase are shown in (d) in the traceability matrix.



Req. ID	Category	Description
REQ_F_01-01	Door inner shaft knob switch for judging the lock/unlock status of the passenger's door	It receives the passenger's door lock status as an input, determines the door lock status, and transmits the status through CAN communication.
REQ_F_01-02	Door inner shaft knob switch for judging the lock/unlock status of the rear left seat door.	It receives the door lock status of the rear left seat as an input, determines the door lock control status, and transmits the status through CAN communication.
...	...	...

(a) List of Functional Requirements

(b) Hardware System model

SF.ID	Req.ID	Action	Input Data	Output Data	Input Event	Output Event
SF-001	REQ_F_01-01	Sensing		AstDrUnlockState_raw [AstDrKnobSW]	AstDrUnlockStateEvent [AstDrKnobSW]	AstDrUnlockStateEvent [AstDrKnobSW]
SF-002	REQ_F_01-01	Transfer	AstDrUnlockState_raw [AstDrUnlockStateIF]	AstDrUnlockState_value [AstDrUnlockStateIF]		
SF-003	REQ_F_01-01	Control	AstDrUnlockState_value [DrUnlockStateCtr]	AstUnlockState_value [DrUnlockStateCtr]		
SF-004	REQ_F_01-01	Transfer	AstUnlockState_value [BCAN_AstUnlockStateIF]	AstUnlockState_raw [BCAN_AstUnlockStateIF]		
SF-005	REQ_F_01-01	Transfer	AstUnlockState_value [ECAN_AstUnlockStateIF]	AstUnlockState_raw [ECAN_AstUnlockStateIF]		
SF-006	REQ_F_01-01	Actuating	AstUnlockState_raw [BCAN_AstUnlockState]			
SF-007	REQ_F_01-01	Actuating	AstUnlockState_raw [ECAN_AstUnlockState]			
SF-008	REQ_F_01-02	Sensing		RLDrUnlockState_raw [RLDrKnobSW]	RLDrUnlockStateEvent [RLDrKnobSW]	RLDrUnlockStateEvent [RLDrKnobSW]

(c) Structured Feature List

Req.ID	SF.ID	Component	Interface
REQ_F_01-01	SF-001	SenComAstDr	AstDr
REQ_F_01-01	SF-002	ConComDr	AstDr
REQ_F_01-01	SF-002	ConComDr	DrIf
REQ_F_01-01	SF-003	ConComCentral	CANtIf
REQ_F_01-01	SF-003	ConComCentral	DrIf
REQ_F_01-01	SF-004	ConComCANt	BCANtIf
REQ_F_01-01	SF-004	ConComCANt	CANtIf
REQ_F_01-01	SF-005	ConComCANt	CANtIf
REQ_F_01-01	SF-005	ConComCANt	ECANtIf
REQ_F_01-01	SF-006	NetComBCANt	BCANtIf
REQ_F_01-01	SF-007	NetComECANt	ECANtIf
REQ_F_01-02	SF-008	SenComRLDr	RLDr
...	...	...	...

(d) Traceability Matrix

(e) Component-based Software Architecture

Figure 13. Association between artifacts in SFCAD.

The requirement REQ_F_01-01 is related to eight structured features, as indicated by the values in the 'Req.ID' column in (c). Among the structured features related to REQ_F_01-01, SF-001 and SF-002 are influenced by the Assistant Knob Switch, which is part of the hardware system model, as confirmed by the attribute values in the structured feature list in Figure 13b). In the traceability matrix, SF-001 is derived from the functional requirement REQ_F_01-01, which affects the configuration of 'SenComAstDr' components. Additionally, it can be seen that the AstDr interface must transmit and receive the data corresponding to the data-related attribute AstDrUnlockState_raw of SF-001.

6. Discussion

In our case study, we applied structured features to integrate functional requirements and hardware system model elements into the CSA design. To achieve this, we utilized SFCAD, analyzing outputs such as the functional requirement list, hardware system model, structured feature list, traceability matrix, and CSA, which are all related to the functional requirement REQ_SW_01-01.

The findings from our case study demonstrate that structured features can effectively bridge the gap between requirements analysis and design artifacts. The attributes of the structured features, which are derived from data and events involving both requirements and hardware system elements, directly influence the CSA design process through SFCAD. These attributes are reflected in key design stages, including component identification and interface operation allocation. Additionally, by positioning the SF.ID column between the ReqID and Component/Interface columns in the traceability matrix, the relationships between the two artifacts are clearly visualized, resulting in enhanced traceability.

Despite the increasing complexity of the system, SFCAD applies a sensor–actuator pattern that addresses common challenges in the software design process. This pattern allows for a flexible and efficient embedded software design and can be extended to other software systems. Although the traceability process in SFCAD is manual, it provides a practical methodology for initiating design and integrating traceability into the development workflow.

However, this study also has some limitations. First, the software architecture can vary depending on the designer's perspective, meaning that the designer's intent and viewpoint significantly impact the final design. SFCAD currently focuses only on the functional aspects related to data flow in CSA. While structured features include both data- and event-related attributes, SFCAD does not yet fully utilize event attributes. Incorporating event attributes to account for component execution timing could further enhance the tool's ability to model and track the dynamic characteristics of embedded software.

Second, although SFCAD promotes requirement traceability in parallel with the design process, the manual creation and management of the traceability matrix imposes an additional burden on designers. This manual effort contributes to the tendency to conduct requirement traceability retrospectively or ignore it altogether, despite its critical importance. Automating or partially automating this process would require the development of tools that support design based on the mathematical models presented in Section 4.

Third, although SFCAD primarily supports traceability between requirements and design during the component development process, requirements must be traceable throughout the entire software lifecycle, from analysis and design to implementation and maintenance. Currently, SFCAD specializes in the early stages of design traceability. Thus, extending its scope to support lifecycle-wide traceability, such as converting design outputs into code traceability, is necessary. Without establishing early traceability between the requirements and design, the burden on designers and developers may increase in later stages, making early traceability a critical challenge.

Future research should address these limitations by exploring the applicability of structured features and SFCAD across diverse system environments, and should also analyze their impact on the overall development process. In addition, optimizing the application of structured features and developing automated tools represent ways of further enhancing traceability and efficiency.

In conclusion, this study has demonstrated that structured features effectively narrow the gap between requirements analysis and design, leading a more efficient CSA design process in its early stages. These findings suggest that structured features can play an active role in future CSA designs.

7. Conclusions

This study analyzed the requirement traceability method in SFCAD, which is a structured feature-based approach for designing component-based software architectures. SFCAD performs initial component identification, designing the components based on the attributes of the structured features. SFCAD supports the connection between components by assigning operations to the interfaces in order to define the data flow between components. Following this design flow, designers can effectively reflect the functional requirements in the component architecture design, which enables traceability using hardware elements based on the characteristics of the structured features. Ultimately, traceability visualization techniques such as a traceability matrix allow for an intuitive understanding of the relationships between artifacts.

Our case study results show that structured features help to bridge the gap between the artifacts of requirements analysis and design. The attributes of the structured features are based on data and events that include requirements and hardware system elements, and these attributes directly influence the CSA design through SFCAD. The attribute values of the structured features, such as component identification and interface operation allocation, are then reflected in the CSA design stages.

However, because SFCAD is applied after the structured feature analysis has already been conducted, the quality of the design outcomes is dependent on the quality of the structured features. Furthermore, the current process is manual, which represents another limitation. Despite these limitations, SFCAD has the potential to enhance the efficiency of the early stages of CSA design through the use of structured features.

To address the limitations of this study, future research should explore ways to extend SFCAD's support for traceability to better reflect the characteristics of embedded software and investigate the development of automated tools for requirement traceability. These efforts are expected to enable a broader application of structured features and SFCAD for traceability.

Author Contributions: Methodology, I.Y. and H.P.; Writing—original draft, I.Y.; Writing—review & editing, S.-W.L.; Project administration, S.-W.L. and K.-Y.R. All authors have read and agreed to the published version of the manuscript.

Funding: Seok-Won Lee's work was supported by the BK21 FOUR program of the National Research Foundation of Korea funded by the Ministry of Education(NRF5199991014091) and the Institute of Information& communications Technology Planning & Evaluation (IITP) under the Artificial Intelligence Convergence Innovation Human Resources Development (IITP-2024-RS-2023-00255968) grant funded by the Korea government(MSIT).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The original contributions presented in the study are included in the article, further inquiries can be directed to the corresponding author.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Peraldi-Frati, M.A.; Albinet, A. Requirement traceability in safety critical systems. In Proceedings of the 1st Workshop on Critical Automotive Applications: Robustness & Safety, Valencia, Spain, 27 April 2010; pp. 11–14.
2. Ramesh, B.; Powers, T.; Stubbs, C.; Edwards, M. Implementing requirements traceability: A case study. In Proceedings of the 1995 IEEE International Symposium on Requirements Engineering (RE'95), York, UK, 27–29 March 1995; pp. 89–95.
3. Kannenberg, A.; Saiedian, H. Why software requirements traceability remains a challenge. *Crosstalk J. Def. Softw. Eng.* **2009**, *22*, 14–19.
4. Yoo, I.; Ryu, K.Y. Structured Feature-Based Component Architecture Design from a Traceability Perspective. In Proceedings of the 2024 Fifteenth International Conference on Ubiquitous and Future Networks (ICUFN), Budapest, Hungary, 2–5 July 2024; pp. 250–252.
5. Szyperski, C.; Gruntz, D.; Murer, S. *Component Software: Beyond Object-Oriented Programming*; Pearson Education: London, UK, 2002.
6. Crnkovic, I. Component-based software engineering—New challenges in software development. *Softw. Focus* **2001**, *2*, 127–133. [CrossRef]
7. Lau, K.K.; Cola, S.D. *An Introduction to Component-Based Software Developement*; World Scientific: Singapore, 2018.
8. Yen, I.L.; Goluguri, J.; Bastani, F.; Khan, L.; Linn, J. A component-based approach for embedded software development. In Proceedings of the Fifth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing, ISIRC, Washington, DC, USA, 29 April–1 May 2002; pp. 402–410.
9. Panunzio, M.; Vardanega, T. A component-based process with separation of concerns for the development of embedded real-time software systems. *J. Syst. Softw.* **2014**, *96*, 105–121. [CrossRef]
10. Campeanu, G.; Carlson, J.; Sentilles, S. Component-based development of embedded systems with GPUs. *J. Syst. Softw.* **2020**, *161*, 110488. [CrossRef]
11. AUTOSAR (AUTomotive Open System ARchitecture). Available online: https://www.autosar.org (accessed on 10 October 2024).
12. Fürst, S.; Mössinger, J.; Bunzel, S.; Weber, T.; Kirschke-Biller, F.; Heitkämper, P.; Kinkelin, G.; Nishikawa, K.; Lange, K. AUTOSAR—A Worldwide Standard is on the Road. In Proceedings of the 14th International VDI Congress Electronic Systems for Vehicles, Baden-Baden. Citeseer, Baden-Baden, Germany, 7–8 October 2009; Volume 62.
13. Martínez-Fernández, S.; Ayala, C.P.; Franch, X.; Nakagawa, E.Y. A Survey on the Benefits and Drawbacks of AUTOSAR. In Proceedings of the First International Workshop on Automotive Software Architecture, Montreal, QC, Canada, 4–8 May 2015; pp. 19–26.
14. Meyer, B. Applying'design by contract'. *Computer* **1992**, *25*, 40–51. [CrossRef]

15. Geisterfer, C.M.; Ghosh, S. Software component specification: A study in perspective of component selection and reuse. In Proceedings of the Fifth International Conference on Commercial-off-the-Shelf (COTS)-Based Software Systems (ICCBSS'05), Orlando, FL, USA, 13–16 February 2006; p. 9.
16. Apel, S.; Kästner, C. An overview of feature-oriented software development. *J. Object Technol.* **2009**, *8*, 49–84. [CrossRef]
17. Turner, C.R.; Fuggetta, A.; Lavazza, L.; Wolf, A.L. A conceptual basis for feature engineering. *J. Syst. Softw.* **1999**, *49*, 3–15. [CrossRef]
18. Yoo, I.; Lee, J.; Ryu, K.Y. Function Block Features to improve Traceability in Embedded Software Architecture Design. In *Proceedings of the Korea Software Congress 2019*; Korean Institute of Information Scientists and Engineers: Seoul, Republic of Korea, 2019; pp. 275–277.
19. Park, H.; Yoo, I.; Ryu, K.Y. Structured Feature Model for Feature Engineering of Embedded Software. In *Proceedings of the Korea Software Congress 2021*; Korean Institute of Information Scientists and Engineers: Seoul, Republic of Korea, 2021; pp. 248–250.
20. Yoo, I.; Park, H.; Ryu, K.Y. Component-based Embedded Software Architecture Design based on Structured Feature. *J. KING Comput.* **2023**, *19*, 36–48.
21. Gotel, O.C.; Finkelstein, C. An analysis of the requirements traceability problem. In Proceedings of the IEEE International Conference on Requirements Engineering, Colorado Springs, CO, USA, 18–21 April 1994; pp. 94–101.
22. Lamsweerde, A.V. *Requirements Engineering: From System Goals to UML Models to Software Specifications*; John Wiley & Sons, Ltd.: Hoboken, NJ, USA, 2009.
23. Wiegers, K.E.; Beatty, J. *Software Requirements*; Pearson Education: London, UK, 2013.
24. Madaki, A.A.; Zainon, W.M.N.W. A review on tools and techniques for visualizing software requirement traceability. In *Lecture Notes in Electrical Engineering, Proceedings of the 11th International Conference on Robotics, Vision, Signal Processing and Power Applications: Enhancing Research and Innovation through the Fourth Industrial Revolution*; Springer: Berlin/Heidelberg, Germany, 2022; pp. 39–44.
25. Suteeca, K. Requirement Traceability Matrix for SaaS Development. In Proceedings of the 2023 Joint International Conference on Digital Arts, Media and Technology with ECTI Northern Section Conference on Electrical, Electronics, Computer and Telecommunications Engineering (ECTI DAMT & NCON), Phuket, Thailand, 22–25 March 2023; pp. 183–187.
26. Hidayati, N.N.; Rochimah, S. Requirements traceability for detecting defects in agile software development. In Proceedings of the 2020 10th Electrical Power, Electronics, Communications, Controls and Informatics Seminar (EECCIS), East Java, Indonesia, 26–28 August 2020; pp. 248–253.
27. Jeong, S.; Cho, H.; Lee, S. Agile requirement traceability matrix. In Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings, Gothenburg, Sweden, 27 May–3 June 2018; pp. 187–188.
28. Zhang, S.; Wan, H.; Xiao, Y.; Li, Z. IRRT: An Automated Software Requirements Traceability Tool based on Information Retrieval Model. In Proceedings of the 2022 IEEE 22nd International Conference on Software Quality, Reliability, and Security Companion (QRS-C), Guangzhou, China, 5–9 December 2022; pp. 525–532.
29. Lyu, Y.; Cho, H.; Jung, P.; Lee, S. A systematic literature review of issue-based requirement traceability. *IEEE Access* **2023**, *11*, 13334–13348. [CrossRef]
30. Pauzi, Z.; Capiluppi, A. Applications of natural language processing in software traceability: A systematic mapping study. *J. Syst. Softw.* **2023**, *198*, 111616. [CrossRef]
31. Laliberte, C.D.; Giachetti, R.E.; Kolsch, M. Evaluation of Natural Language Processing for Requirements Traceability. In Proceedings of the 2022 17th Annual System of Systems Engineering Conference (SOSE), Rochester, NY, USA, 7–11 June 2022; pp. 21–26.
32. Zhao, L.; Alhoshan, W.; Ferrari, A.; Letsholo, K.J.; Ajagbe, M.A.; Chioasca, E.V.; Batista-Navarro, R.T. Natural language processing for requirements engineering: A systematic mapping study. *ACM Comput. Surv. (CSUR)* **2021**, *54*, 1–41. [CrossRef]
33. Wang, B.; Wang, H.; Luo, R.; Zhang, S.; Zhu, Q. A Systematic Mapping Study of Information Retrieval Approaches Applied to Requirements Trace Recovery. In Proceedings of the SEKE, Virtual, 1–10 July 2022; pp. 1–6.
34. Katta, V.; Raspotnig, C.; Karpati, P.; Stålhane, T. Requirements management in a combined process for safety and security assessments. In Proceedings of the 2013 International Conference on Availability, Reliability and Security, Regensburg, Germany, 2–6 September 2013; pp. 780–786.
35. *ISO 26262*; Road Vehicles—Functional Safety. International Organization for Standardization: Geneva, Switzerland, 2011.
36. VDA Working Group 13. *Automotive SPICE Process Assessment/Reference Model*, 4th ed.; VDA Working Group 13: Berlin, Germany, 2023.
37. Sikora, E.; Tenbergen, B.; Pohl, K. Requirements engineering for embedded systems: An investigation of industry needs. In Proceedings of the Requirements Engineering: Foundation for Software Quality: 17th International Working Conference, REFSQ 2011, Essen, Germany, 28–30 March 2011; pp. 151–165.
38. Sikora, E.; Tenbergen, B.; Pohl, K. Industry needs and research directions in requirements engineering for embedded systems. *Requir. Eng.* **2012**, *17*, 57–78. [CrossRef]
39. Wang, F.; Yang, Z.B.; Huang, Z.Q.; Liu, C.W.; Zhou, Y.; Bodeveix, J.P.; Filali, M. An approach to generate the traceability between restricted natural language requirements and AADL models. *IEEE Trans. Reliab.* **2019**, *69*, 154–173. [CrossRef]
40. Abdelahad, C.; Riesco, D.; Kavka, C. Requirements Traceability using SysML Diagrams and BPMN. *Int. J. Adv. Softw.* **2020**, *13*, 129–138.

41. Intrigila, B.; Della Penna, G.; D'Ambrogio, A.; Campagna, D.; Grigore, M. Process-Oriented Requirements Definition and Analysis of Software Components in Critical Systems. *Computers* **2023**, *12*, 184. [CrossRef]
42. Alenazi, M.; Niu, N.; Savolainen, J. A novel approach to tracing safety requirements and state-based design models. In Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering, Seoul, Republic of Korea, 27 June–19 July 2020; pp. 848–860.
43. Ahmadiyah, A.S.; Rochimah, S.; Siahaan, D. Modeling Traceability between Requirements and Coding Using the Property Listing Task. *IEEE Access* **2024**, *12*, 129274–129289. [CrossRef]
44. Souza, E.; Moreira, A.; Goulão, M. Deriving architectural models from requirements specifications: A systematic mapping study. *Inf. Softw. Technol.* **2019**, *109*, 26–39. [CrossRef]
45. Ahmed, S.; Ahmed, A.; Eisty, N.U. Automatic transformation of natural to unified modeling language: A systematic review. In Proceedings of the 2022 IEEE/ACIS 20th International Conference on Software Engineering Research, Management and Applications (SERA), Las Vegas, NV, USA, 25–27 May 2022; pp. 112–119.
46. Bass, L.; Clements, P.; Kazman, R. *Software Architecture in Practice: Software Architect Practice_c3*; Addison-Wesley: Boston, MA, USA, 2012.
47. Bengtsson, P.; Lassing, N.; Bosch, J.; van Vliet, H. Architecture-level modifiability analysis (ALMA). *J. Syst. Softw.* **2004**, *69*, 129–147. [CrossRef]