Traceability Analysis Of A High-Level Automotive System Architecture Document

Dennis Hild
Individual Standard GmbH
Berlin, Germany
dennis.hild@individual-standard.com

Martin Beckmann
Technische Universität Berlin
Berlin, Germany
martin.beckmann@tu-berlin.de

Andreas Vogelsang
Technische Universität Berlin
Berlin, Germany
andreas.vogelsang@tu-berlin.de

***Abstract*—More and more functions in automotive systems are enabled and controlled by software that is distributed over a large number of systems and components. In addition, the systems are more and more interconnected to implement the desired vehicle functions. Creating and maintaining a high-level overview of the relations between vehicle functions, systems, and components in a complete and consistent way requires a lot of effort. On the other hand, such a high-level architecture is beneficial for planning the development, analyzing the architecture, or defining product variants. In this paper, we analyze a real-world document that was manually created to document all vehicle functions, systems, and components of one car series and relations between these. We formalized the content of this model by providing a model of the concepts and a set of consistency rules. By evaluating the consistency rules on the given document, we found 213 contradictory relations and 547 missing relations. Based on these results, we conclude that manually maintaining such high-level architectures is highly error-prone and should thus be supported by automation and appropriate tooling.**

***Index Terms*—Traceability, Trace Link Recovery, Automotive System Architecture Analysis**

I. Introduction

Innovation in the automotive industry is still primarily driven by functions that are implemented in software [1], [2]. Since automotive systems encompass a wide variety of different domains (e.g., infotainment or control engineering of mechanical and electronic components) which at the same time are highly-connected, it is a challenging task to maintain an overview of all the dependencies of the involved systems. Studies have shown that developers are unaware of a large fraction of these dependencies [3]. The concept of relating entities that exhibit dependencies or other forms of connections is called traceability [4] and has been considered an important factor in the design of complex software systems (as in automotive software engineering) for quite some time [5]. Such connections are implemented by the use of trace links [4]. These trace links may appear in different directions, for example, horizontally (from system to system) as well as vertically (from system to function). Moreover, these dependencies also occur across multiple dimensions (e.g., a system may rely on the output of another system's function). This means, one system may depend on the function provided by another system. But keeping such an overview in a correct, consistent, and complete state involves even more aspects. One must also keep track how all the systems and their functions are mapped to components (i.e., Electronic Control Units (ECUs) of a vehicle). This challenge is further aggravated by the fact that modern vehicles consist of numerous systems and components. As a result, the size of a document that contains a high-level architectural overview of functions, systems, and components has reached an enormous extent. As the number of systems in modern vehicles continues to grow [6], so does the number of dependencies and thus also the required effort to maintain them properly. In practice, such documents are oftentimes created and maintained manually in general-purpose tools such as MS Excel.

On the other hand, a high-level overview of functions, systems, components, and their relations is essential to plan development activities [7], to define product variants, or to assess and optimize the systems' architecture [8]. This is especially important when components and sometimes even whole systems are developed by (oftentimes multiple different) suppliers [2], [9]. But the trace links not only facilitate the identification of dependencies, they are also needed even aside from organizational issues. The existence of trace links is mandated by regulations for safety-critical systems (see ISO26262 [10]) and process improvement standards (see Automotive SPICE [11]) of the automotive industry.

In this paper, we analyze a high-level architecture document of a vehicle series from practice. We aim at two things: First, we analyze the underlying structure of the document to define a model that represents its concepts (functions, systems, components, and relations between them). In addition, we define a set of consistency rules. Based on the consistency rules, we are able to uncover missing connections and to correct contradictory connections. Second, we apply the set of rules to the document to find out how many false and missing connections are existent.

The remainder of this paper is organized as follows. The next section provides details on the background. We explain what information the document contains, how the document is composed, and how trace links are realized. The third section presents the underlying structure of the document and introduces a set of rules that aim to ensure correctness and completeness of the connections. In the fourth section, we present the results of applying the set of rules to the document and emphasize a number of situations that occur most frequently. The fifth section provides insights into related work in the area of trace link recovery. The last section concludes this work and gives an outlook on future work.

II. Background

A. High-Level Architectural Description Of Automotive E/E Systems

As stated in the introduction, the relations between vehicle functions, systems, and components is often manually maintained in a high-level architecture document. The main content of this document is displayed in a table-like manner. Fig. 1 shows an abstract excerpt of this document.

It contains information on:

- names of systems in a vehicle series
- names of functions and the system they are associated with
- names of subfunctions and by which functions they are used
- information on whether functions and subfunctions are provided by or contribute to a system
- a mapping of systems, functions, and subfunctions to components

The first column in the table in Fig. 1 defines the type of an entity in the table[1]. It may be a system, a function, or a subfunction. The second column *System* assigns a system to each entity to indicate its membership. If the entity is a function, the third column specifies its name. The fourth column does the same for subfunctions. Function and subfunction entries of a system may reference a function or subfunction that is provided by that system or by another system. In the latter case, the other system must contain a corresponding entry for that function. To indicate this difference, the column *Type* specifies whether a function or subfunction is an own function of the system (denoted by the *o* in the appropriate cell) or an external function (denoted by the *e* in the appropriate cell). In case a function is an external function, the columns *Provided By* and *Provided To* give information which system provides the function and which system is using it. This information is redundantly provided for both the origin (e.g., *F_2* in system *S_n*) and for the target (e.g. *F_2* in system *S_1*). A function may be provided for multiple systems, while it only has one origin. In the example in Fig. 1, function *F_2* from system *S_n* is used as a subfunction in function *F_1* of system *S_1*. The rest of the columns represent the components of the vehicle series. An *x* in a cell maps an entity to a component, which means that the entry is deployed on the component. It has to be noted that subfunctions might intentionally not have a mapping to a component and functions might not have any subfunctions.

The whole document is managed in a commonly used spreadsheet program. Although the possibility exists, the spreadsheet does not make use of macros or other programmable routines. It is managed in a complete manual manner.

[1]The indentation does not exist in the original document and is used to improve clarity on which entity belongs to which entity

B. Realization Of Trace Links

The rows in the document contain three types of references to other entities:

Vertical Traceability: Systems consist of functions, which themselves may consist of subfunctions (vertical traceability). This *belongs to*-relation is expressed redundantly in the document in two ways. First, a function belongs to a system if it is listed in a row below the system entity but before another system. The same applies for the relation of subfunctions to functions. This results in a hierarchical decomposition. Second, the column *System* contains the name of the system to which the function or subfunction belongs to.

Horizontal Traceability: Systems may use functionality provided by functions of other systems. Such a function or subfunction is denoted as an external function in the column *Type*. The relation between systems (horizontal traceability) is expressed using the columns *Provided By* and *Provided To*. For the affected function (or subfunction), these columns contain the name of the origin and the target respectively. This information is stored both for the system using the function and for the system providing the function. The providing system lists all of the systems using the function in the column *Provided To*. Similar to what is explained in (1), external functions also appear in a row below the system using it. As a result, this relation is also expressed redundantly.

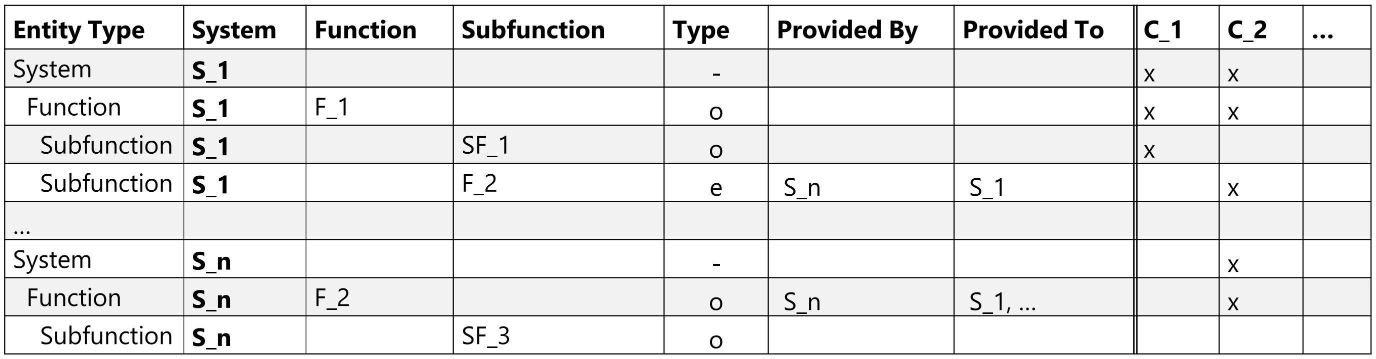
Deployment Traceability: Systems, functions, and subfunctions are mapped to components to express that they are deployed on these. This is expressed by an *x* in the column of the respective component column. If a subfunction is deployed on a component, the function it belongs to and its system must also have a relation to this component (i.e., the deployment relations are aggregated).

C. Key Figures

To give an overview over the size of such a high-level architecture document, we provide some figures for the analyzed document. The document consists of 3,214 rows and 208 columns. It contains the basic architectural information for 169 systems and encompasses a mapping to 180 individual components. For the systems, 2,111 functions and 935 subfunctions are listed. Functions may be used multiple times by other systems as functions or as subfunctions. Both numbers include such multiple appearances.

D. Research Design

The objective of our research effort was to gain insights in the practical realization of traceability in an automotive architecture document. The provided document was analyzed manually by the first author of this paper and peculiarities were discussed with the contributing authors of this paper. This has lead to the identification of a number of problems concerning the existing traceability information in the document. Based on these findings a set of rules was created to automatically improve the completeness and consistency of the traceability in the document. This set of rules is presented in a conceptual manner in the next section. The rules were then formalized and



Entity Type	System	Function	Subfunction	Type	Provided By	Provided To	C_1	C_2	...
System	**S_1**			-			x	x	
Function	**S_1**	F_1		o			x	x	
Subfunction	**S_1**		SF_1	o			x		
Subfunction	**S_1**		F_2	e	S_n	S_1		x	
...									
System	**S_n**			-				x	
Function	**S_n**	F_2		o	S_n	S_1, ...		x	
Subfunction	**S_n**		SF_3	o					

Fig. 1: Example of the structure of the architecture document (o: own function, e: external function)

implemented. This way we were able to gather information on what kind of problems appear how frequently. The results of this analysis are explained and discussed in Section IV.

III. Trace Link Recovery

A. *Underlying Structure Of The Document*

The underlying structure of the document as explained in the previous section is visualized in Fig. 2. The boxes represent the three entities *System*, *Function*, and *Component*. The arrows display the connections between them. For our approach, we distinguish between two different kinds of connections:

1) Connections that are existing explicitly in the document (represented by dotted lines)
2) Connections that we can derive from the existing relations (represented by dashed lines)

For the connections represented by enumeration item 1) we aim to improve the completeness by finding missing connections and adding these to the document. The connections represented by enumeration item 2) only exist implicitly in the document. For these dependencies, it is necessary to follow other existing (explicit) connections to make them visible. For instance, to find dependencies between systems, it is necessary to follow an external function of a system back to its originating system. Although such connections are not yet documented explicitly in the document, developers of the OEM mentioned that these connections would provide valuable information to them. Hence, we also intent to find these connections.

Besides, we also attempt to find and correct inconsistent and incorrect connections. Incorrect and inconsistent connections are characterized by missing information about the origin or target of the trace link. E.g., a function is provided to a certain system, but that system does not contain the function.

Overall, we want to improve the quality of the traceability links in the document based on the extracted structure. We try to achieve this goal by proposing a set of rules that find missing links and correct false links.

B. *Set Of Rules*

The rules we propose can be categorized into two sets. The first set (consisting of 8 rules) aims to identify and repair incorrect links (correcting rules) while the second set (consisting of 18 rules) aims to recover missing trace links (recovering rules). The second set of rules addresses both kinds of connections in the document as mentioned in the enumeration in the previous subsection. A number of such cases where links are either missing or incorrectly implemented as well as a solution are depicted in the Fig. 3 to Fig. 8. Besides, a short description and the number of rules for both sets is displayed in TABLE I.

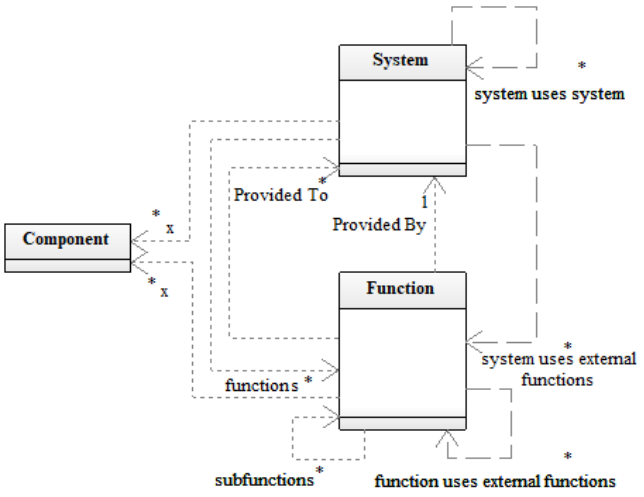
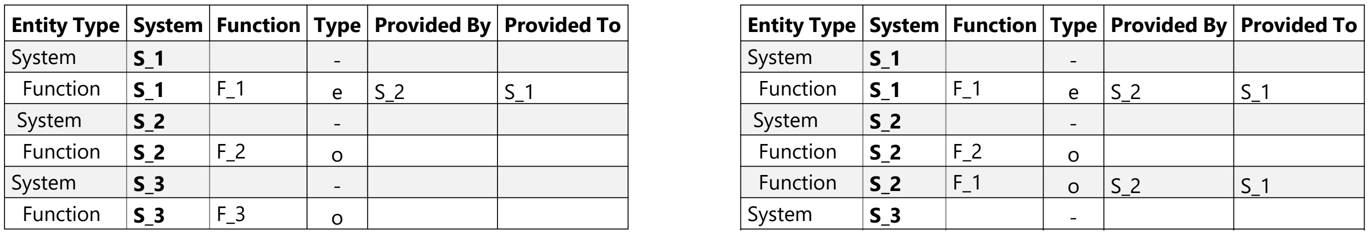


Fig. 2: Underlying structure of the architecture document

TABLE I: Overview of the set of rules

Description	**#Rules**
Correcting Rules	8
Identification of function type	3
Correction of faults	5
Recovering Rules	18
Recovering connections from functions ...	8
... to functions	1
... to systems	2
... to components	5
Recovering connections from systems ...	10
... to functions	2
... to systems	2
... to components	6



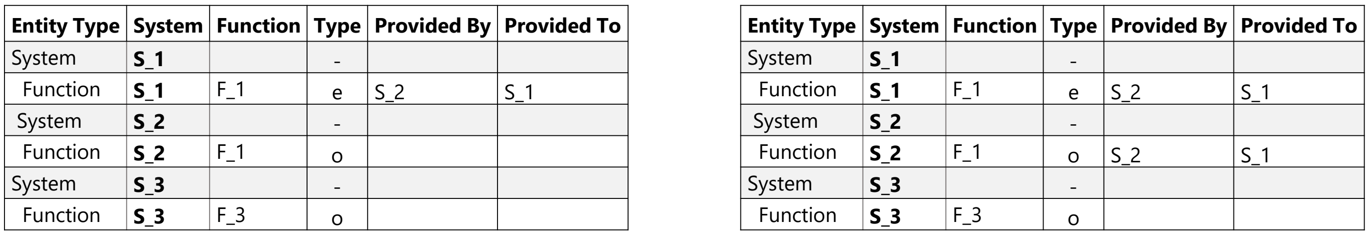
Entity Type	System	Function	Type	Provided By	Provided To
System	**S_1**		-		
Function	**S_1**	F_1	e	S_2	S_1
System	**S_2**		-		
Function	**S_2**	F_2	o		
System	**S_3**		-		
Function	**S_3**	F_3	o		

(a) existing

Entity Type	System	Function	Type	Provided By	Provided To
System	**S_1**		-		
Function	**S_1**	F_1	e	S_2	S_1
System	**S_2**		-		
Function	**S_2**	F_2	o		
Function	**S_2**	F_1	o	S_2	S_1
System	**S_3**		-		

(b) expected

Fig. 3: Case 1 - Function missing in originating system



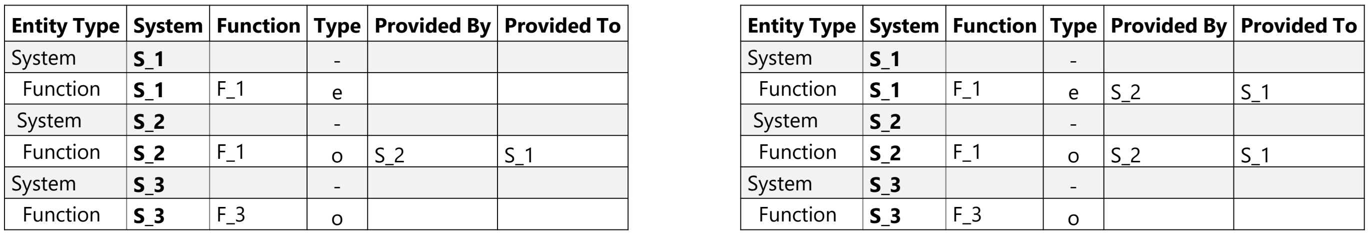
Entity Type	System	Function	Type	Provided By	Provided To
System	**S_1**		-		
Function	**S_1**	F_1	e	S_2	S_1
System	**S_2**		-		
Function	**S_2**	F_1	o		
System	**S_3**		-		
Function	**S_3**	F_3	o		

(a) existing

Entity Type	System	Function	Type	Provided By	Provided To
System	**S_1**		-		
Function	**S_1**	F_1	e	S_2	S_1
System	**S_2**		-		
Function	**S_2**	F_1	o	S_2	S_1
System	**S_3**		-		
Function	**S_3**	F_3	o		

(b) expected

Fig. 4: Case 2 - Missing start of trace link



Entity Type	System	Function	Type	Provided By	Provided To
System	**S_1**		-		
Function	**S_1**	F_1	e		
System	**S_2**		-		
Function	**S_2**	F_1	o	S_2	S_1
System	**S_3**		-		
Function	**S_3**	F_3	o		

(a) existing

Entity Type	System	Function	Type	Provided By	Provided To
System	**S_1**		-		
Function	**S_1**	F_1	e	S_2	S_1
System	**S_2**		-		
Function	**S_2**	F_1	o	S_2	S_1
System	**S_3**		-		
Function	**S_3**	F_3	o		

(b) expected

Fig. 5: Case 3 - Missing end of trace link

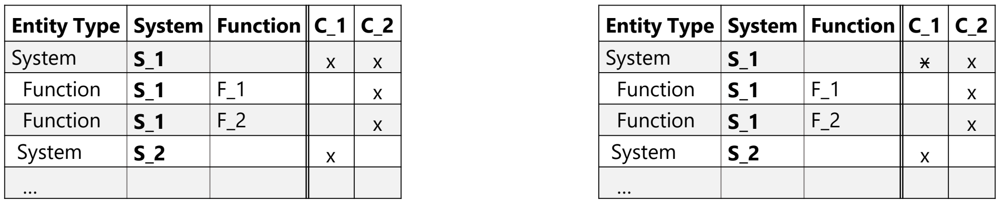
Entity Type	System	Function	C_1
System	**S_1**		
Function	**S_1**	F_1	x
...			

(a) existing

Entity Type	System	Function	C_1
System	**S_1**		x
Function	**S_1**	F_1	x
...			

(b) expected

Fig. 6: Case 4 - Missing mapping of system to component



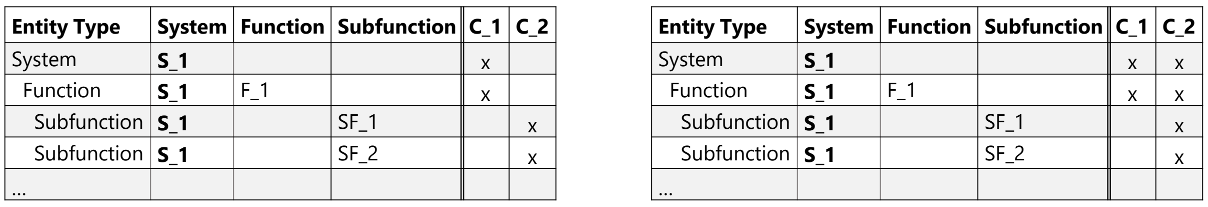
Entity Type	System	Function	C_1	C_2
System	**S_1**		x	x
Function	**S_1**	F_1		x
Function	**S_1**	F_2		x
System	**S_2**		x	
...				

(a) existing

Entity Type	System	Function	C_1	C_2
System	**S_1**		~~x~~	x
Function	**S_1**	F_1		x
Function	**S_1**	F_2		x
System	**S_2**		x	
...				

(b) expected

Fig. 7: Case 5 - Superfluous mapping of system to component



Entity Type	System	Function	Subfunction	C_1	C_2
System	**S_1**			x	
Function	**S_1**	F_1		x	
Subfunction	**S_1**		SF_1		x
Subfunction	**S_1**		SF_2		x
...					

(a) existing

Entity Type	System	Function	Subfunction	C_1	C_2
System	**S_1**			x	x
Function	**S_1**	F_1		x	x
Subfunction	**S_1**		SF_1		x
Subfunction	**S_1**		SF_2		x
...					

(b) expected

Fig. 8: Case 6 - Missing mapping of function to component

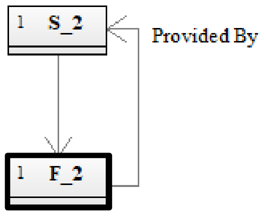


Fig. 9: Structure for completion of *Provided By*

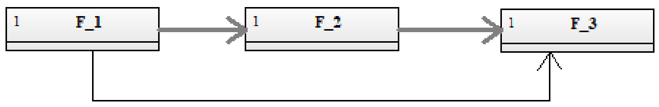
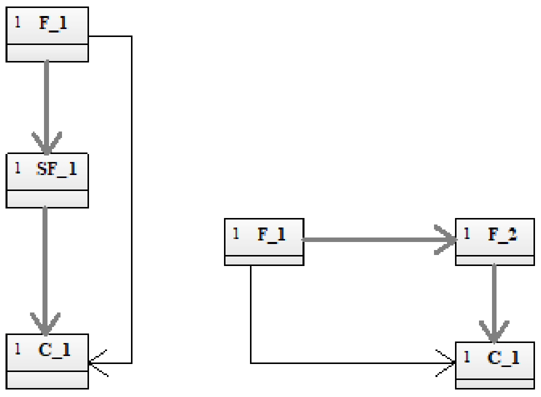


Fig. 10: Structure for a trace link between functions



(a) Link via used subfunction

(b) Link via used function

Fig. 11: Structure examples for trace links from functions to components

C. Correcting Rules

There are three basic ideas behind the correcting rules:

1) Information at the origin must also exist at the target and vice versa (e.g., an external function is listed in the originating system as in Fig. 3).
2) The information must be complete (the columns *Provided By* and *Provided To* of functions used by other systems must not omit system names as in Fig. 4 and Fig. 5).
3) The type of a function must be correct (e.g., an external function is indicated by an *e* in the column *Type*).

Rule 1) assures that a function provided to another system actually exists in the originating system. Rule 2) assures that every time a function is used externally, the function contains information on the originating system and the using systems. Rule 3) assures that the type is correct. Hence, the correcting rules try to identify functions appearing in other systems and check whether all information on that function across the entire document is correct. An instance, where one of the rules applies, is depicted in the model-notation in Fig. 9. Assuming function *F_2* of system *S_n* in Fig. 1 does not have the value *S_n* in the column *Provided By*, this value is complemented. This realizes the (deliberately redundant) backward connection *Provided by* in Fig. 9.

D. Recovering Rules

The rules for recovering missing connections are created by transitively deducting connections from existing connections. These rules create trace links between functions, systems, and components. The main intention of these rules is to make connections explicit. Examples for such connections are displayed in Fig. 10, Fig. 11, Fig. 12, and Fig. 13. The thicker arrows between the entities symbolize existing connections. The narrow arrows are the connections that are created by our approach.

Hence, Fig. 10 depicts the dependency of a function *F_1* to a function *F_3*, resulting from a dependency of *F_1* to *F_2* which in turn depends on *F_3*.

Fig. 11 depicts a dependency of a function *F_1* to a component *C_1*. This dependency results from a subfunction (Fig. 11a) or another function (Fig. 11b) which has a dependency to component *C_1* and is used by the function *F_1*.

Fig. 12 depicts the dependencies of a system *S_1* to another system *S_2* that results from a function *F_1* using the output of a function *F_2* of system *S_2*.

Fig. 13 is similar to the situation displayed in Fig. 11a but propagates the dependencies further up to the system.

These rules only showcase a part of all possible rules. The total number of 18 rules were systematically gathered by going through all possible combinations of connections that might exist and assessing them towards their necessity in practice. That also includes recursive structures that appear when, for instance, multiple functions are involved (as in Fig. 10). The

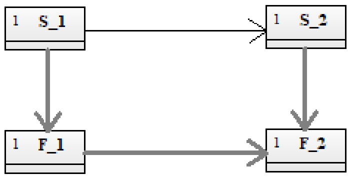


Fig. 12: System to system structure example

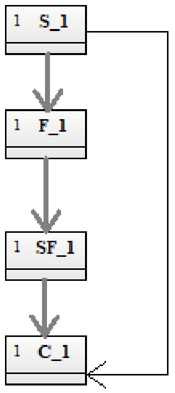


Fig. 13: Example structure for trace links between systems and components

next section presents the results from applying the rules to the described data set.

IV. Analysis / Evaluation

In this section we present the results of applying the rules to the described data set. The process of the evaluation started with the implementation of the approach and its rules. This implementation was then used on the already described data set. In this we regard the number of trace links before the application of the rules, after the application of the correction rules (first rule set) and after the application of the recovering rules (second rule set). These numbers are also visualized in Fig. 14. The objective of the evaluation was to find out how often certain faulty situations occur. We chose certain deficiencies because, in our view, these reflected the ones with the biggest impact on the document quality. This left us with a total of four question we aimed to answered. Moreover, we wanted to know how many trace links existed before and after the application of our approach and how many corrections were made. Hence, the question we wanted to answer are the following:

Q1) How many external functions are found that were not defined? We describe an external function as not defined if the function does not exist in the system that is supposed to be the originating system. Hence, we look for an external function which is not listed in the specified originating system. This situation is also displayed Fig. 3.

Q2) How many components are connected to a function but not to the system of the function? The circumstances

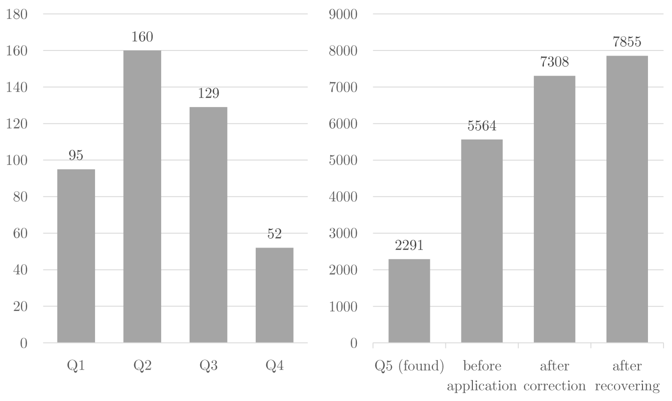


Fig. 14: Results

reflected by this question are the same as displayed in Fig. 13 and Fig. 6.

Q3) How many components are connected to a system but not to any function of the system? In the context of this question, we look for the number of components which are connected to systems but not to any of the functions of the system (also displayed in Fig. 7). Such a situation indicates that there is no existing rationale for the dependency between the system and the specified component.

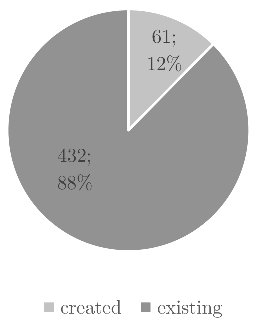
Q4) How many components are connected to subfunctions but not to the function using the subfunction? This question is similar to Q3. But instead of functions and systems, it aims at subfunctions and functions (also displayed in Fig. 8).

Q5) How many trace links exist before the application of the approach and how many exist after the application of the approach? With this question we aim to find out how many trace links existed before and after the application of our approach.

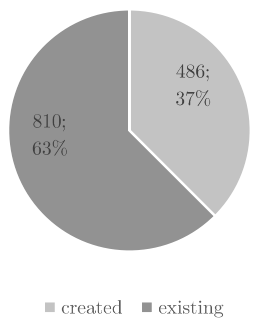
Q6) How many changes were made by the correcting rules? By answering this question, we try to find out how many mistakes could be automatically fixed by our approach.

A. *Results*

After the application of the approach, we analyzed the results to answer the previously stated questions. Fig. 14 visualizes the answers of the questions. Considering question *Q1)*, the approach recovered 95 external functions which did not exist in the specified originating system. It is possible that such a situation arises then a function, which is used by another function or system does not exists anymore, or an external function is needed by the considered system and another system is responsible for this functionality. So the function also has to be created in the corresponding system. Between components and systems 160 trace links were recovered (*Q2)*). Considering question *Q3)*, we identified 129 cases in which components are connected to systems for no reason. Similarly for *Q4)*, we identified 52 missing trace links between components and functions. To answer question *Q5)*: the analysis shows 5,564 initially existing connections. After the application of the first set of rules 7,308 connections exists. This number of



(a) functions as source



(b) systems as source

Fig. 15: Number of recovered trace links

connections results from filling in the *Provided By* column for each function. After the application of the approach 7,855 connections exist. Hence, the method recovered 2,291 implicit existing trace links. These trace links consists of trace links, which exists through filling *Provided By* and *Provided To* columns, and trace links recovered by the application of the second set of rules. For the last question *Q6)* we found that the correcting rules were applied 213 times.

B. Discussion

In this subsection we discuss whether the links added by our rules provide additional benefit. For this purpose, we compare how many trace links existed before the application of our approach and how many after. We do this for both trace links from functions and from systems.

The results for functions are displayed in Fig. 15a and for systems in Fig. 15b. For functions initially 432 trace links existed. Our approach was able of recovering 61 more trace links. Hence, we were able to improve the trace link coverage for functions by about 12%.

For systems initially 810 trace links existed. Here, we found 486 additional trace links that did not exist before. This represents 37% of all trace links that should have existed.

In more detail, we observed that 11 out of 18 recovering rules are relevant for the examined document. We derive this conclusion from the fact that these rules recover trace links that already existed in many places between entities. The remaining 7 of the 18 rules could not create any trace links because the corresponding structures did not appear in the document. Hence, these 7 rules are more of a theoretical nature.

All in all, our results confirm related findings that maintaining traceability requires a significant effort for even moderately sized systems [12], let alone large systems as in our analysis.

V. Related Work

Automatically complementing trace links between or within software engineering artifacts falls into the area of trace link recovery [13]. There are already a number of approaches in this field. They differ in the underlying idea of how to find connections.

Text-based information retrieval approaches (e.g. [14]) use textual artifacts such as textual requirements. Event-based approaches (e.g. [14, p. 173-194]) are (oftentimes) interactive approaches that trigger the creation of connections when artifacts are created. Rule-based approaches create connections based on predefined rules (e.g. syntactic rules [15]). Model-based approaches rely on an underlying model of the involved artifacts and create trace links based on detected changes [14, p. 215-240] or require the user to apply the approach from the very start [16].

The method presented in this paper makes use of the structure of the artifacts and is usable at any time. Unlike the mentioned and other related approaches, it is independent of textual information (which as a source of ambiguity, cannot achieve full reliability [17], [18]) and does not require to be used in a continuous manner. To the best of our knowledge there is no approach in literature that fits these criteria.

VI. Conclusion and Future Work

In this paper, we presented an analysis of trace links in a high-level architecture document of an automotive OEM. We extracted the underlying structure of the document and derived a set of rules that ensures the correctness and completeness of trace links. The rules, we proposed, were implemented in order to improve the quality of the document. An evaluation of this method shows that 547 missing trace links were created and 213 incorrect or inconsistent trace links were corrected.

In future, it may prove to be fruitful to implement such rules in the program the document is maintained. This may encourage a better quality from the start and prevent occurrences of defects and inconsistencies whenever it is manually edited. But we think, it would even be more promising to use modeling tools instead of a spreadsheet program to represent such information. Although it remains doubtful whether this can be accomplished in industrial practice on a wide scale. Future work should also address the impact on other developments artifacts that are derived of the document analyzed in this paper. More generally, it needs to be assessed how the rules can be adapted to documents other than the one investigated in this work.

References

[1] K. Grimm, "Software technology in an automotive company: Major challenges," *International Conference on Software Engineering*, 2003.
[2] A. Haghighatkhah, A. Banijamali, O.-P. Pakanen, M. Oivo, and P. Kuvaja, "Automotive software engineering: A systematic mapping study," *Journal of Systems and Software*, vol. 128, 2017.
[3] A. Vogelsang and S. Fuhrmann, "Why feature dependencies challenge the requirements engineering of automotive systems: An empirical study," in *21st IEEE International Requirements Engineering Conference (RE)*, 2013. [Online]. Available: https://arxiv.org/pdf/1708.08660
[4] O. Gotel and C. Finkelstein, "An analysis of the requirements traceability problem," *International Conference on Requirements Engineering*, 1994.
[5] B. Ramesh and M. Jarke, "Toward reference models for requirements traceability," *IEEE transactions on software engineering*, vol. 27, no. 1, 2001.
[6] A. Pretschner, M. Broy, I. H. Kruger, and T. Stauner, "Software engineering for automotive systems: A roadmap," *Future of Software Engineering*, 2007.
[7] F. Pettersson, M. Ivarsson, and P. Öhman, "Automotive use case standard for embedded systems," *ACM SIGSOFT Software Engineering Notes*, vol. 30, no. 4, 2005.

[8] A. Vogelsang, H. Femmer, and M. Junker, “Characterizing implicit communal components as technical debt in automotive software systems,” in *13th Working IEEE/IFIP Conference on Software Architecture (WICSA)*. IEEE Computer Society, 2016, pp. 31–40. [Online]. Available: http://www.aset.tu-berlin.de/fileadmin/fg331/Publications/WICSA16.pdf

[9] G. L. Ragatz, R. B. Handfield, and T. V. Scannell, “Success factors for integrating suppliers into new product development,” *Journal of Product Innovation Management*, vol. 14, no. 3, 1997.

[10] International Organization for Standardization, “ISO/DIS 26262 - road vehicles — functional safety,” 2009.

[11] VDA QMC Working Group 13 / Automotive SIG, “Automotive SPICE Process Assessment / Reference Model,” 2015.

[12] B. Ramesh, C. Stubbs, and M. Edwards, “Lessons learned from implementing requirements traceability,” *Journal of Defense Software Engineering*, vol. 8, no. 4, 1995.

[13] J. Cleland-Huang, O. Gotel, and A. Zisman, *Software and Systems Traceability*. Springer, 2012, vol. 2, no. 3.

[14] J. Cleland-Huang, B. Berenbach, S. Clark, R. Settimi, and E. Romanova, “Best practices for automated traceability,” *Computer*, vol. 40, no. 6, 2007.

[15] G. Spanoudakis, A. Zisman, E. Pérez-Minana, and P. Krause, “Rule-based generation of requirements traceability relations,” *Journal of Systems and Software*, vol. 72, no. 2, pp. 105–127, 2004.

[16] J. Cleland-Huang, J. H. Hayes, and J. Domel, “Model-based traceability,” *ICSE Workshop on Traceability in Emerging Forms of Software Engineering*, 2009.

[17] D. M. Berry, E. Kamsties, and M. M. Krieger, “From contract drafting to software specification: linguistic sources of ambiguity,” http://se.uwaterloo.ca/ dberry/handbook/ambiguityHandbook.pdf, University of Waterloo, Tech. Rep., 2003.

[18] M. Robeer, G. Lucassen, J. M. E. van der Werf, F. Dalpiaz, and S. Brinkkemper, “Automated extraction of conceptual models from user stories via NLP,” *International Requirements Engineering Conference*, 2016.