Assessment of ITS Architectures

Manuel Fünfrocken[1]*, Andreas Otte[1], Jonas Vogt[1], Niclas Wolniak[1], Horst Wieker[1]

[1] htw saar, University of Applied Sciences, Saarbrücken, Germany,
Goebenstr. 40, 66117 Saarbrücken, +49 681 5867 668,
*manuel.fuenfrocken@htwsaar.de, Germany

Abstract: Worldwide intelligent transportation systems (ITS) are at the edge from research and development to deployment and commercial operation. Japan has already introduced ITS technology in the market and the USA and the European Union have elaborated planes for deployment. ITS in this context means communication and services based on communication technology between vehicles and other traffic participants on one hand and traffic and road infrastructure, vehicle manufacturer and other mobility service provider on the other hand. For a successful introduction, a reliable and secure exchange of mobility related information is a key factor. To provide such an exchange an overall architecture for ITS and for all participants und users is necessary. With the introduction in the market, the assessment of an ITS architecture without an existing reference system arises. Because most assessment method currently existing, measure how 'good' an architecture is in comparison to another architecture. In this paper, we describe ways and approaches how existing methods can be extended and combined to provide means for the assessment of ITS architectures in the pre-deployment phase. As a result, deploying parties should be enabled to assess an architecture before introducing it to the public.

1. Introduction

Automated and connected driving functions are a keystone for future mobility. Efficient electric driving is important to reach the global environmental goals. Multimodal mobility concepts with all kinds of transportation systems (e.g. cars, bus, robot taxi, train, motorcycle, etc.) are necessary to fulfil the mobility demand of the growing number of people living on this planet, especially in big megacities but also in the sparsely inhabited rural areas. All those future scenarios have one thing in common: they will only be possible through communication. Missing and non-transparent communication structures and inconsistent information quality complicate or even prevent the introduction of new services and the partaking of new stakeholders. An ITS architecture that connects those services, communication networks, traffic infrastructure and the mobility users has to provide an easy, reliable, saleable, secure and privacy friendly environment. In this context such an architecture is called cooperative intelligent transportation system (C-ITS).

Many industrial companies and consortia (e.g. classical vehicle manufacturers and suppliers, IT Companies like Google or Apple but also new players like Tesla or Uber) and a huge number of research projects around the globe have worked and are still working on this topic and created a multitude of ITS architectures.

The question arises what is the most suitable system for deployment. Moreover, how can the quality of such an architecture be measured and rated. Many assessment solutions only cope with the comparison of different solutions; however, for ITS no exiting currently in-use solutions exist. Nonetheless, to compare developed architectures such a solution would be necessary. In this paper, we describe methods how such an assessment is possible without the need of a system for comparison. For a full view on the architecture a technical, an economic, and an organizational assessment is necessary. We focus on the technical part of the assessment, based on the experiences we gathered in different research projects.

This paper is structured as follows: the next section "Assessment Methodologies" gives an overview about different available assessment methods. Based on those methods two variants of our assessment approach in different projects is presented in the section "Approach". The paper concludes with an outlook regarding the next steps.

2. Assessment Methodologies

A large number of assessment methods exist to verify different models and architectures, which however are not equally suitable to verify ITS communication architectures. Especially in the field of research where sometimes no comparable architectures exist, and complex requirement structures and abstract concepts are in consideration conventional assessment methods do not meet the requirements of measuring, as their focus is mostly product and business driven.

The following six methodologies form various fields are very common and often used to assess software architectures. Therefore, they shall be examined about their suitability and applicability.

2.1. Cost–utility analysis (CUA)

In the CONVERGE research project a cost-utility analysis, sub-analysis of the multiple-criteria decision analysis (MCDA) has been used [1]. This is a comparably simple qualitative approach to distinguish between two or more decision alternatives. It is easy to calculate and well suited for 'soft criteria'. However, as it relies on the presence of two or more alternatives, it is hardly useable as there are currently no alternatives to the architecture deployed in the ITS context. For CONVERGE two hypothetical architectures, an ideal solution and a feasible optimum of the existing architecture were defined for comparison.

2.2. Field operational test support action (FESTA)

The FESTA methodology, as described in the FESTA-Handbook [2], is widely used in common C-ITS projects. However, it is intended for so-called field operational tests (FOT). According to FOT-Net, a FOT is "A study undertaken to evaluate a function, or functions, under normal operating conditions in road traffic environments typically encountered by the participants using study design so as to identify real-world effects and benefits." Therefore resulting in large-scale user tests, intended to examine the behaviour of a system and its impact under real and everyday conditions. As an operational field test is a basic requirement of a FESTA assessment it is not suitable for the assessment of an abstract architecture, which is not tested in such a way. It is especially worse, if no implementation of the architecture is present and the assessment should be based only on an architectural description.

2.3. Scenario-Based Architecture Analysis (SAAM)

The scenario-based architecture analysis method (SAAM) has been proposed by Kazman *et al.* [3] as early as 1996. It was developed to provide a "way to express and analyse context-dependent quality attributes" [3] in software architectures. The principle behind SAAM is that the quality of an architecture can be assessed by evaluating how much changes are needed to adapt the software to specific use cases. For example, if easy portability is a relevant quality criterion for an architecture it is evident to examine for a few examples, how easy the architecture could be ported.

The basic process for a scenario-based architecture analysis [3] is split into six steps. Step (1), **Describe the architecture,** obtains a more or less formal description of the architecture, where it is made sure that every participant understands the architecture definition. In the next step, (2) **Develop scenarios,** scenarios used to evaluate the architecture are specified. Kazman et al. [3] recommend using descriptions of one or two sentences each, briefly describing a task or action, which the architecture should be capable of performing. In step (3), **Evaluate each scenario**, each scenario is assigned to one of two groups: direct or indirect scenarios. Direct scenarios are scenarios, which can be performed by the architecture without any changes. In contrast to direct scenarios, performing indirect scenarios - also described as *change cases* or *growth cases* – need specific adjustment of the architecture. In addition, the number of components, which should be modified, as well as the expected effort should also be noted. In the fourth step, (4) **Reveal scenario interactions**, the number of changes for each component is identified. Step (5), **Weight scenarios and scenario interactions,** provides a prioritization, by weighting scenarios against each other. This depends highly on the circumstances under which the SAAM is performed. In the final step (6), **Interpret results**, the results are interpreted in context of the environment in which the SAAM was performed.

2.4. Architecture Trade-off Analysis Method (ATAM)

The Architecture Trade-off Analysis Method (ATAM) is the successor of SAAM, also developed by Kazman et al. [4]. Its main goal is to choose a suitable architecture by highlighting trade-offs.

ATAM defines nine steps [5], where in step (1) **Present the ATAM**, the evaluation method is described to the assembled participants. In the second step, (2) **Present business drivers**, a project spokesperson (ideally the project manager or system customer) describes what business goals are motivating the development effort and hence what will be the primary architectural drivers (e.g., high availability or time to market or high security). In step (3) **Present architecture**, an architect will describe the architecture, focusing on how it addresses the business drivers presented in step 2. Following this, step (4) **Identify architectural approaches,** identifies new and existing architectural approaches, led by the architect, but does not analyse them. Before analyzation, step (5) **Generate quality attribute utility tree**, examines quality factors and requirements that comprise system "utility" (performance, availability, security, modifiability, usability, etc.), specified down to the level of scenarios, annotated with stimuli and responses, and prioritizes them. The analyzation of the approaches is done in step (6) **Analyse architectural approaches**. Based on the high-priority factors identified in Step 5, the architectural approaches that address those factors are investigated and analysed here (for example, an architectural approach aimed at meeting performance goals will be subjected to a performance analysis). During this step, architectural risks, sensitivity points, and trade-off points are identified. In step (7), **Brainstorm and prioritize scenarios**, a larger set of scenarios is elicited by the entire group of stakeholders. This set of scenarios is prioritized via a voting process involving the entire stakeholder group. In the eight step, (8) **Analyse architectural approaches**, the activities of Step 6 are reiterated, but the highly ranked scenarios from Step 7 are used. Those scenarios are considered to be test cases to confirm the analysis performed thus far. This analysis may uncover additional architectural approaches, risks, sensitivity points, and trade-off points, which are then documented. In the final step, (9) **Present results**, the ATAM team presents the findings to the assembled stakeholders. This is based on the information collected in the ATAM (approaches, scenarios, attribute-specific questions, the utility tree, risks, non-risks, sensitivity points, trade-offs.

2.5. Architecture Level Modifiability Analysis (ALMA)

Bengtsson et al. [6] created the Architecture Level Modifiability Analysis (ALMA). It is intended to provide a repeatable metric to assess the modifiability of software architectures. It is based on the methods developed by Lassing et al. [7] and Bengtson and Bosch. [8]

ALMA consists of five different steps. It is intended to serve three different overall goals: prediction of future maintenance cost, identification of system inflexibility and comparison of two or more alternative architectures. In the first step, (1) **Goal setting**, one of the three overall intended goals is selected. In the second step, (2) **Architecture description,** a more or less formal definition of the architecture is created. ALMA does not specify a specific

kind of architecture definition, but emphases, that the description should provide information of the decomposition of the system in components, the relationships between components and the relationship to the system's environment. In the third step, (3) **Change scenario elicitation,** scenarios to assess the architecture are created and elicited. This process is similar to other scenario selection processes, e.g. for requirement analysis. However, as to limit the number of scenarios to investigate for ALMA, the scenarios are grouped into so-called *equivalence classes*, so just one scenario from each equivalence class needs to be considered. To further reduce the number of scenarios, classification of change categories is used. Here, the relevance classes, which are not relevant to evaluation for the intended goal, are left out of the analysis. Afterward follows step (4), **Change scenario evaluation.** This is the main step of ALMA where an architecture level impact analysis is performed. This analysis itself consists of three steps: identification of affected components, determination of effect on the components, and determination of ripple effects. Those steps are performed together by system architects, designers, and analysts, as especially the last step, determination of ripple effects is hard to do without access to actual source code. Therefore, estimations need to be performed based on the knowledge and experience of the partaking stakeholders. Finally, for step (5) **Interpretation**, Bengtsson et al. cite Lindvall and Sandahl [9], stating that impact analysis tends to predict only half of the necessary changes, due to bad estimations of software engineers and architects. This needs to be considered by the analysts when interpreting the results.

2.6. Performance Assessment of Software Architecture (PASA)

Williams and Smith have developed the Performance Assessment of Software Architecture (PASA) method. [10] As the methods described above, it is a scenario-based process. However, the scenarios used in PASA are different from those used in SAAM or ATAM. Whereas SAAM and ATAM use scenarios which describe the contexts in which a system is used, scenarios in PASA describing processing steps for a particular use of the software. The type of scenarios used in SAAM/ATAM are called *performance studies* in PASA. In contrast to SAAM/ATAM, PASA is more concerned with evaluating the performance aspect of an architecture.

An assessment performed with PASA consists of the steps described below. Usually, they are performed in the order given, but iteration or adaptions to fit the context in which they are performed are common. PASA starts with (1) **Process overview**, where the process is described to all partakers. In this step, the reasons for performing the process, as well as a description of the process and the expected outcome is presented. Then, in (2) **Architecture overview**, the architecture is presented to the assessors. This is not done by means of formal documents, but by an actual presentation held by the developers of the architecture. Typically, the assessors already have reviewed the available documentation, so they can interact in a question-and-answer fashion with the development team during this phase. This process is often necessary, as most architecture descriptions are only available in an informal way. Afterwards, step (3) **Identification of critical use cases** is performed. A use cases describes the behaviour of the system, as they are visible to the end-user. Critical use-cases are those, who are required for the system to work correctly. In addition, use cases with a performance risk associated to them are also critical use cases. A performance risk may be that the system will fail, if certain performance requirements are not meet. Those use cases are used as input for (4) **Selection of key performance scenarios**. Each of the use cases identified in the previous process step consist of several actions executed in sequence to fulfil the use case. Key performance scenarios are those scenarios, which are performed frequently, as they influence the overall system performance the most. Additionally, some scenarios might be included, which are not performed frequently, but are have also critical performance requirements, like crash recovery procedures. Scenarios are described as annotated UML Sequence diagrams. [11, 12] In the next step, (5) **Identification of performance objectives**, it is defined what a 'good' and 'bad' system, or in the case of PASA, a 'fast' and 'slow' system is. This is necessary to be able to assess something, especially abstract entities like software architecture. Therefore, clear, quantitative, and measurable objectives for each scenario are defined. As the architecture is only seldom specified in detail assessors and developer discuss in step (6) **Architecture clarification and discussion** the key elements identified in the previous process steps together. The goal is to understand component relations and the impact on the performance of those. To analyse the architecture in step (7) **Architectural analysis**, several techniques may be used, like identification of software architectural styles and patterns [13, 14] identification of performance antipatterns [15], or performance modelling and analysis [10]. "Antipatterns [15] are conceptually similar to patterns [16] in that they document recurring solutions to common design problems. They are known as antipatterns because their use (or misuse) produces negative consequences. Antipatterns document common mistakes made during software development. They also document solutions for these mistakes. Thus, antipatterns tell you what to avoid and how to fix a problem when you find it. Performance antipatterns document common performance problems and how to fix them. [16] [17] They capture the knowledge and experience of performance experts by providing a conceptual framework that helps analysts to identify performance problems and suggesting ways of solving them. Antipatterns are refactored (restructured or reorganized) to overcome their negative consequences. A refactoring is a correctness-preserving transformation that improves the quality of the software. For example, the interaction between two components might be refactored to improve performance by sending fewer messages with more data per message. This transformation does not alter the semantics of the application, but it may improve overall performance. Refactoring may also be used to enhance other quality attributes including reusability, modifiability, or reliability." [10] The next step is (8) **Identification of alternatives**. If performance problems are found, alternatives may be identified to circumvent those problems. This can be done, e.g. by deviations from architectural style, alternative interactions between components, or refactoring to remove an antipattern. Williams et al. state, that "it is important that the PASA client receive a document containing the mission,

findings, specific steps to take, the priority of the steps, and their relative importance" [10]. This is done in process step (9) **Presentation of results**. The final step (10) is **Economic analysis.** If the PASAM was successful, failures and shortcomings in the architecture have been identified in the design phase and could be removed by facilitating alternatives. However, as this cannot be seen easily, it is recommended to perform an analysis how much resources have been spent on the PASAM and how much resources would have been needed to fix those problems in a later process step.

3. Approach

Based on the results of the evaluation of the various methodologies described above, a mixed evaluation method has been chosen for our research projects iKoPA [16] and C-MobILE [17]. This method combines cost-utility analysis with scenario-based evaluation methods. Those methods are used for a functional assessment, which is enhanced by an additional evaluation of the implementation based on FESTA field test, as well as an ATAM analysis for a non-functional assessment. The results of the both assessments are combined by the use of cost utility analysis again.

The cost-utility analysis is used to evaluate requirement fulfilment. It provides an easy way to see the indication, if all requirements have been fulfilled. The scenario-based architecture assessment will evaluate the ability of the architecture to pass various, hypothetical scenarios.

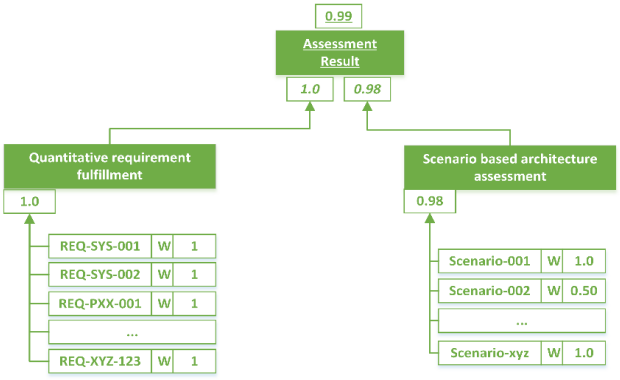

Fig. 1. *Overview of methodology. (Exemplary values)*

With those two sub-assessments, as can be seen in Fig. 1, the overall assessment can be calculated as the weighted sum of overall requirement fulfilment and the overall scenario fulfilment. This is formally defined as:

$$\boldsymbol{F_A = W_{A_R} * F_R + W_{A_S} * F_S} \quad (1)$$

With

$$\boldsymbol{0.0 \leq F_R \leq 1.0, F_R \in \mathbb{Q}} \quad (2)$$

$$\boldsymbol{0.0 \leq F_S \leq 1.0, F_S \in \mathbb{Q}} \quad (3)$$

Whereas the sum of the weights is equal to one:

$$\boldsymbol{W_{A_R} + W_{S_R} = 1, with\ W_{A_R}, W_{S_R} \in \mathbb{Q}} \quad (4)$$

Taken alone, those values are hardly useful without further explanation. However, they may be used by future initiatives who want to compare their architectures against architectures already assessed. In the following sub sections, the detailed calculation of the various fulfilment numbers is shown.

3.1. Quantitative requirement fulfilment

The overall requirement fulfilment is calculated from the fulfilment values of the various requirements, as shown in Fig. 2.

The overall degree of fulfilment for the requirements F_R can be calculated as

$$\boldsymbol{F_R = \sum_{k=0}^{n} W_{R_k} * F_{R_k}} \quad (5)$$

Where W_{R_k} is the relative weight of requirement *k* and $\boldsymbol{F_{R_k}}$ is the degree of fulfillment of requirement *k*. Under the following two conditions,

First, that the sum of all weights is equal to one

$$\boldsymbol{\sum_{k=0}^{n} W_{R_k} = W_{R_G} = 1.0 \leq W_{R_k} \leq 1.0, W_{R_k} \in \mathbb{Q}} \quad (6)$$

And second, that the fulfilment is measured as binary value

$$\boldsymbol{F_{R_k} \in \{0,1\}, F_{R_k} \in \mathbb{N}, 0 = not\ fulfilled; 1 = fulfilled} \quad (7)$$

F_R will also be between zero and one, or more formally,

$$\boldsymbol{0.0 \leq F_R \leq 1.0, F_R \in \mathbb{Q}} \quad (8)$$

The fulfilment $\boldsymbol{F_{R_k}}$ of a specific requirement is evaluated based on its `Means-of-Verification'. The means-of-verification describe, how an assessor needs to verify, that the requirement has been fulfilled. For most of the requirements, this is done by performing an expert rating based on the architecture.

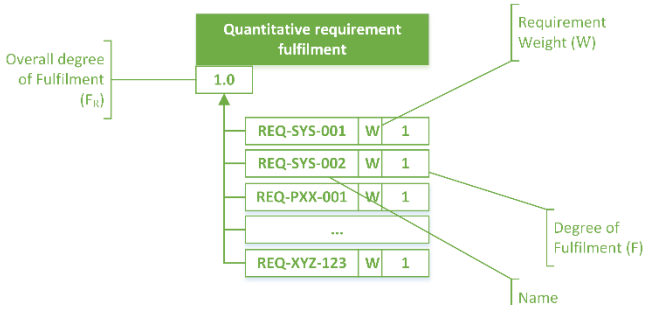

Fig. 2. *Requirement fulfilment*

3.2. Definition of Means-of-verification

As defined earlier, expert assessors need to rate whether each individual requirement has been fulfilled by the architecture or not. In order to perform this binary assessment,

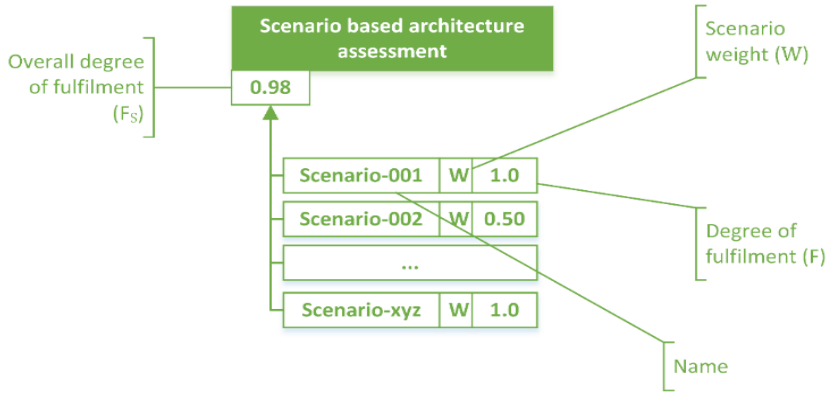

Fig. 3. *Scenario-based architecture assessment*

we describe a clear process supporting the assessor in the following.

The foundation for the assessment has been set by the creator of each requirement. The crator has also provided a means of verification (MoV) statement, which is stored in the same database. The MoV statement shall be used by the assessor to check the fulfilment of a requirement, as it provides precise conditions of the specific requirement. .

Once the MoV is defined by the requirement author, a reviewer checks the MoV, to make sure it is understandable and plausible. If this is not the case, the reviewer writes a comment on the requirement, thus starting a discussion with the author. Once a consensus is reached, the requirement MoV is marked as finished in a tracking sheet.

3.3. Requirement weighting

Not all requirements have the same importance with respect to the project's goals. As some requirements are more important than others are, we propose four different weights, as shown in Fig. 4, which are used to weight requirements in relation to each other.

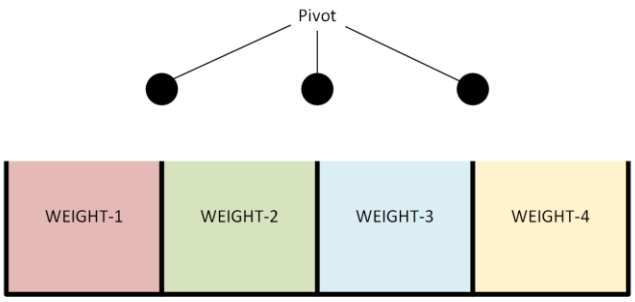

Fig. 4: Weight level and pivot requirements

Fig. 4 shows the four weight-level and the so called pivot requirements. The pivot requirements representing border/transition points between two weight-level and will be defined globally in the assessment process. Therefore, a pivot requirement can be taken as orientation to the assessor on where to assign a specific requirement. Every requirement can be compared in the matter of significance to the three pivot requirements. For example, if an item is more important than the pivot requirement 3 but less important than pivot 2, it will be assigned to WEIGHT-3. The three pivot requirements are chosen from the gathered requirement pool in an assessment meeting.

The weight-level themselves refer to a numeric factor between 0.5 and 3. This factor is used as a tool to describe how much more important e.g. WEIGHT-1 is compared to WEIGHT-4.

WEIGHT-1 shall be used for most significant requirements that have to be fulfilled in any case. It refers to the factor 3.0. WEIGHT-2 is assigned to the factor 1.5. It is used for important requirements that can be neglected in a few special cases but constrain the functionality if not fulfilled. The weight-level WEIGHT-3 is neutral and have the factor 1.0. WEIGHT-4 is used for less important items, the factor 0.5 is chosen for this level.

3.4. Scenario-based architecture assessment

The scenario-based architecture assessment follows a similar approach to the previous described quantitative requirement fulfilment. Individual degrees of scenario fulfilment are aggregated into an overall scenario fulfilment degree, as shown in Fig. 3.

The overall fulfilment degree of the various scenarios is summed up to an overall fulfilment degree as shown in Fig. 3.

The overall degree of fulfilment for the architecture assessment F_S can be calculated as

$$\boldsymbol{F_S} = \sum_{\boldsymbol{k=0}}^{\boldsymbol{n}} \boldsymbol{W_{S_k}} * \boldsymbol{F_{S_k}} \qquad (9)$$

where W_{S_k} is the relative weight of the scenario k and F_{S_k} is the degree of fulfillment of requirement k. The following two conditions apply:

first, the sum of all weights W_{S_G} is equal to one

$$\sum_{\boldsymbol{k=0}}^{\boldsymbol{n}} \boldsymbol{W_{S_k}} = \boldsymbol{W_{S_G}} = \boldsymbol{1,0.0} \leq \boldsymbol{W_{S_k}} \leq \boldsymbol{1.0}, \boldsymbol{W_{S_k}} \in \mathbb{Q} \qquad (10)$$

and second, the fulfilment degree of each scenario $\boldsymbol{F_{S_k}}$ is express as percent value

$$\boldsymbol{0.0} \leq \boldsymbol{F_{S_k}} \leq \boldsymbol{1.0}, \boldsymbol{F_{S_k}} \in \mathbb{Q} \qquad (11)$$

Where a value of 1.0 means, that the architecture is able to fully support the scenario, and a value of 0.0 means, that the architecture cannot be used to fulfil the scenario at all.

Scenarios will be evaluated by their feasibility. Therefore, five level of feasibility are defined and described in Table 1. Each level is assigned to a numeric value, allowing the calculation of a degree of fulfilment.

Table 1 Feasibility level

Level	Description	Value
Feasible	The scenario is already fully supported without any adjustment of the architecture.	1.0
Conditionally feasible	The scenario is conditionally feasible, if an interface and its protocol has to be assimilated.	0.75
Adaptably feasible	The scenario is adaptably feasible, if a component needs to be extended with an additional interface to itself or another component.	0.5
Hardly feasible	The scenario is hardly feasible, if a new component is needed to realize it.	0.25
Not feasible	The scenario is not feasible at all without heavy changes like an insertion of a new layer.	0.0

3.5. Scenario creation process

Known scenario-based architecture assessments like SAAM as described above do not define how scenarios are created. Therefore, the methodology defines a tool to facilitate this process.

Each scenario shall describe a hypothetical apply or use-case, which is described in a short text. Afterwards an assessor needs to examine if a scenario can be handled by the architecture. Such use-cases arises from a topic, e.g. security or privacy and affects one or more components of the architecture. Every project partner is assigned to a topic or expertise and shall determine each component if there could be a use-case that could be applied to the architecture, which has not already been described as a requirement or a use-case before.

3.6. Scenario Weighting

Every scenario needs to be weighted as well. A direct comparison of two or more scenarios can succeed by comparing the scenarios affiliation to one or more goals the architecture is used to achieve. The weight levels introduced in 3.3 can also be used for the scenario weighting.

4. Outlook

Our approach described above has been defined in two different research projects, but has only partially been tried yet.

The German project iKoPA [16] develops the basic design for a system, which serves as an open integrated platform for future intelligent transportation services. These services for automated driving will be connected in an innovative, future-proof, secure, privacy-friendly and comprehensive way. The iKoPA project and the assessment of its architecture will be finished end of 2018.

C-MobILE [17] aims to help local authorities in Europe to deploy C-ITS services. The reference architecture developed in this project will be assessed partially by the approach present in this paper. As this project also contains a field operational test in 2019/2020, the assessment can be enhanced by measuring performed according to FESTA [2].

It would be interesting to compare the assessment results achieved in those projects and see, if general rules for assessment of C-ITS based architectures can be obtained from those. For the future users of C-ITS architectures, it is crucial to possess a possibility to assess architectures before implementing them.

5. Acknowledgment

The results presented in this paper have been taken from work done in the iKoPA and C-MobILE projects.

The project iKoPA is funded by the German Federal Ministry of Education and Research. The results presented in this paper were developed jointly by the iKoPA project partners.

The project C-MobILE has received funding from the European Union's Horizon 2020 research and innovation program under grant agreement No 723311.

6. References

[1] CONVERGE, "Deliverable D6 - Final Assessment," online, http://converge-online.de/doc/download/D6-AP8-Final-Assessment.pdf, 2016.

[2] FOT-Net, "FESTA-Handbook - Version 6," online, http://fot-net.eu/wp-content/uploads/sites/7/2017/01/FESTA-Handbook-Version-6.pdf, 2016.

[3] R. Kazman, A. Gregory, L. Bass and P. Clements, "Scenario-based analysis of software architecture," *IEEE software*, pp. 47-55, 1996.

[4] R. Kazman, M. Klein and P. Clements, "ATAM: Method for architecture evaluation," DTIC Document, 2000.

[5] S. E. Institute, "Architecture Tradeoff Analysis Method," Carnegie Mellon Univerity, [Online]. Available: http://www.sei.cmu.edu/architecture/tools/evaluate/atam.cfm. [Accessed 28 03 2017].

[6] P. Bengtsson, N. Lassing, J. Bosch and H. van Vliet, "Architecture-level modifiability analysis (ALMA)," *Journal of Systems and Software*, vol. 69, no. 1-2, pp. 129-147, 2004.

[7] N. Lassing, D. Rijsenbrij and H. van Vliet, "Towards a broader view on software architecture analysis of flexibility," in *IEEE Comput. Soc*, 1999.

[8] J. Bosch and P. Bengtsson, "Architecture level prediction of software maintenance," in *IEEE Comput. Soc*, 1999.

[9] M. Lindvall and K. Sandahl, “How well do experienced software developers predict software change?,” *Journal of Systems and Software*, vol. 43, no. 1, pp. 19-27, 1998.

[10] L. G. Williams and C. U. Smith, “PASA-SM: a method for the performance assessment of software architectures,” in *ACM Press*, 2002.

[11] J. Rumbaugh, I. Jacobson and G. Booch, The unified modeling language reference manual, Boston: Addison-Wesley, 2005.

[12] K. Fakhroutdinov, “UML Diagrams,” [Online]. Available: http://www.uml-diagrams.org/sequence-diagrams.html. [Accessed 28 03 2017].

[13] E. Gamma, R. Helm, R. Johnson and J. Vlissides, Design patterns: elements of reusable object-oriented software, Reading, Mass: Addison-Wesley, 1995.

[14] F. Buschmann, Pattern-oriented software architecture: a system of patterns, Chichester: Wiley, 1996.

[15] W. J. Brown, AntiPatterns: refactoring software, architectures, and projects in crisis, New York: Wiley, 1998.

[16] R. H. R. J. a. J. V. E. Gamma, *Design Patterns: Elements of Reusable Object-Oriented Software*, massachusetts: Addison-Wesley, 1995.

[17] C. U. S. a. L. G. Williams, Performance Solutions: A Practical Guide to Creating Responsive, Scalable Software, Reading, Massachusetts: Addison-Wesley, 2002.

[18] C. U. S. a. L. G. Williams, “Software Performance Antipatterns,” in *Software performance antipatterns*, Ottawa, Canada, 2000.

[19] iKoPA, “iKoPA Project Website,” 03 01 2018. [Online]. Available: https://ikopa.de. [Accessed 03 01 2018].

[20] C-MobILE, “Accelerating C-ITS Mobility Innovation and depLoyment in Europe,” 2017. [Online]. Available: http://c-mobile-project.eu.

[21] R. C. M. H. W. M. I. a. T. J. M. W. J. Brown, AntiPatterns: Refactoring Software, Architectures, and Projects in Crisis, New York: John Wiley and Sons, Inc., 1998.