

Model-Based Prototype Development to Support Early Validation of Cyber-Physical System Specifications

Jennifer Brings, Philipp Bohn, Torsten Bandyszak, Felix Föcker, and Marian Daun

paluno – The Ruhr Institute for Software Technology, University of Duisburg-Essen, Germany
{jennifer.brings, philipp.bohn, torsten.bandyszak,
marian.daun}@paluno.uni-due.de

Abstract. [Motivation] In the engineering of cyber-physical systems, attention is given to early validation of requirements artifacts. The use of prototypes is one known technique to identify incorrectness and aberrations from the stakeholder intentions early. [Problem] Software prototypes of cyber-physical systems' software, however, often need to be adapted to the prototype's hardware, which may differ from the system's hardware. This leads to differences between the system's and the implemented prototype's specification, impeding the applicability of validation results to the system's requirements. [Solution Idea] To support the validation of requirements artifacts for cyber-physical systems, we propose the generation of an explicit prototype specification. Traceability relations aid the requirements engineer in deciding whether a finding actually corresponds to a defect in the requirements or whether it results from changes due to the prototype's different hardware. [Contribution] In this paper, we propose a model-based prototyping approach for validating requirements artifacts of cyber-physical systems. The approach relies on the generation of an explicit prototype specification and on a categorization scheme for validation results. The latter allows determining the impact of the validation results on the system's requirements, and facilitates the correction of errors.

Keywords: Early validation, model-based development, prototype engineering, cyber-physical systems

1 Introduction

A major challenge in system development is specifying requirements that correctly reflect underlying stakeholder needs. Relevant stakeholder intentions are often unclear, ambiguous, or simply unknown, which leads to the development of systems that fail to fulfill their purposes. The later these problems are discovered, the higher the cost for fixing them [1]. Hence, it is important to validate requirements as early as possible [2]. One commonly acknowledged technique for validating requirements is the development of prototypes (e.g., [3], [4]). Prototypes allow stakeholders to experience and interact with the system, which helps them decide if a property is desired or not.

Copyright © 2016 for this paper by its authors. Copying permitted for private and academic purposes.

To validate existing requirements and elicit further stakeholder intentions by prototyping, it is important to evaluate a system's behavior and its respective functional interdependencies in a sufficiently realistic environment. However, this is often not feasible in the engineering of cyber-physical systems (CPS) software. Since CPS consist of software and hardware parts, either the hardware needs to be simulated or the software needs to be deployed on some hardware that is different from the hardware that the final system will use (e.g., an electronic control unit of a previous version). Hence, there may be significant differences between the system's and the prototype's hardware, which impedes the use of the prototype for requirements validation. The prototype hardware, for example, may not offer a specific sensor that the system needs to gain information about its environment. In this case, the required information must be obtained from other sources, e.g., by using a combination of different sensors or by simulation. This causes a difference between the prototype's requirements and the requirements for the actual system. Hence, some validation results (e.g., those concerning the missing sensor) are only applicable to the prototype but not to the system itself.

To help determine if a validation result is applicable to the system requirements and, consequently, to discover incorrect and missing requirements, we propose a model-based approach for deriving a prototype specification from the system specification, as well as for analyzing and transferring the validation results. To this end, we propose a technique and a supporting classification scheme for categorizing validation results, which allow for determining, which validation results are applicable to the system specification. Furthermore, this model-based approach supports the correction of the system specification based on a corrected prototype specification.

This paper is structured as follows: Section 2 introduces our solution concept. Subsequently in Section 3, we demonstrate the solution concept using an avionics collision avoidance system as case example. Section 4 discusses related work and Section 5 concludes the paper.

2 Solution Concept

Developing prototypes has proven to be an effective way for validating requirements (see [5]). Yet the development of prototypes for cyber-physical systems faces several challenges due to their typically close interaction with their environment (cf. e.g., [6]). To overcome these challenges, we propose a model-based approach for the development of prototypes to aid in validating requirements specifications of cyber-physical systems. Our approach provides semi-automated support for the early validation of cyber-physical systems specifications by providing guidance on deriving prototype specifications and analyzing the obtained validation results w.r.t. their applicability to the original system. Fig. 1 illustrates the steps and artifacts involved in the approach, which are:

1. A prototype specification is derived from the system specification. While doing so, traceability links and rationales for changes are documented. Changes can for instance, result from the use of a hardware component different from the final system's hardware. This step will be explained in more detail in Section 2.1.

2. The prototype is developed according to the prototype specification in order to obtain an executable prototype of the cyber-physical system.
3. The prototype is demonstrated so that stakeholders can assess and evaluate its behavior. Additionally, common inspection and other validation techniques might be applied. Validation results are documented as output of this step (see Section 2.2).
4. The validation results are checked with respect to their applicability to the system specification (further details in Section 2.3). To this end, trace links and rationales for identifying differences between system and prototype specification need to be considered.
5. A corrected system specification is created semi-automatically based on the validation results. Some corrections can be incorporated automatically into the system specification while others need to be revised manually (Section 3.5 will give examples for both cases)

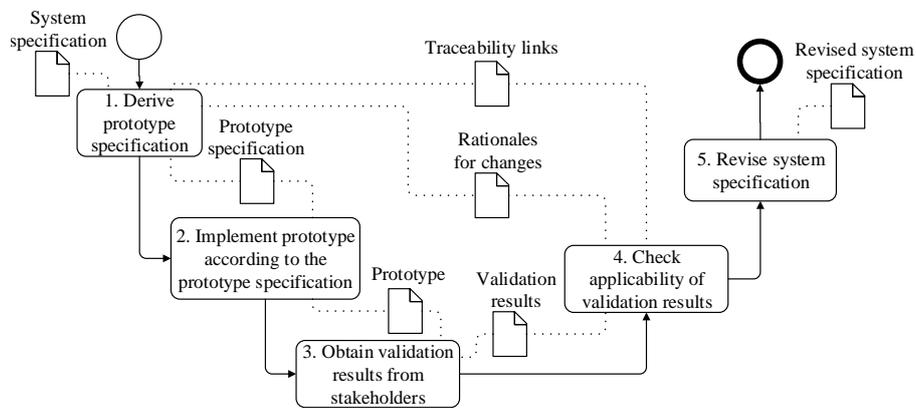


Fig. 1. Solution approach

In the following subsections, we elaborate on the main parts of our contribution. First, we will describe how to derive the prototype specification from the system specification and how to document the changes made. Then we elaborate on the validation step of our approach, including the attribute scheme, which provides attributes to categorize validation results. Last, we explain how this categorization helps determining whether a validation result is applicable to the system specification or not.

2.1 Deriving the Specification of the Prototype

Due to limited capabilities of the prototype's hardware, it is often not feasible to implement the system requirements specification as is and deploy it onto the prototype's hardware. Instead, a specific prototype specification has to be created to ensure structured prototype development. Our approach relies on the idea to derive this prototype specification from the original system specification. In doing so, changes from the system specification are incorporated in the prototype specification. These changes take the specific capabilities of the prototype's hardware into account, which most likely

differs from the originally intended CPS hardware. For example, in the absence of a specific sensor, the prototype might estimate context measurements.

A crucial task that goes along with these changes is the documentation thereof. As model-based engineering of cyber-physical systems can be seen as a de-facto standard [7], automated traceability techniques can be used to document relationships between originating model elements in the system specification and the adaptations in the prototype specification (cf. [8]). Traceability- or trace-links (cf. e.g., [9]) allow for documenting relationships between development artifacts. Typed trace links are used to provide additional information regarding the type of relationship between two model elements [10]. Several taxonomies for traceability relationship types have been suggested in the literature (cf. e.g., [11], [8]). For documenting changes between the system's requirements specification and the prototype's requirements specification, we use trace links to trace the evolution of software artifacts, such as those suggested by [12], for example. In particular, we distinguish different kinds of traceability links (see Table 1) to ensure proper documentation of the relation between system specification and prototype specification. At the same time, rationales for the traced changes are documented as well.

Table 1. Traceability link types used to document differences between the system specification and the prototype specification

Traceability link type	Description
Replaces	This type of trace link connects one model element from the system specification with one model element from the prototype specification if they both specify the same underlying functionality, yet some changes had to be made.
Refines	This type of trace link connects one model element from the system specification with several model elements from the prototype specification, which specify the same functionality.
Add	This type of trace link indicates that a model element is added to the prototype specification, which does not correspond to any model element in the system specification.
Delete	This type of trace link indicates that a model element in the system specification does not correspond to any model element in the prototype specification.

2.2 Validating the Prototype

For validating the system under development, the stakeholders experience the prototype's behavior and assess each of the intended system's functionality with respect to its presence and correctness in the implemented prototype. In doing so, stakeholders determine if a functionality is a necessary feature of the system and if further functionalities are missing. Each validation result is documented in a structured way and categorized according to an attribute scheme.

Our classification of validation results is based on the Orthogonal Defect Classification scheme (ODC) [13]; in particular, on the defect qualifier attribute of the ODC. This attribute allows for classifying the reasons for defects in software artifacts to support

requirements document inspections [14]. Table 2 shows the possible values for each attribute. For example, a functionality, which is wanted but implemented incorrectly in the prototype is classified as *present*, *incorrect*, and *necessary*. While this scheme allows for several combinations of attribute values, some value combinations are not possible. It is, for instance, impossible to assess the correctness of a functionality that is not present at all.

Table 2. Attribute scheme for validation results

Attribute	Values	Description
Presence	present / not present	Does the prototype provide the functionality?
Correctness	correct / incorrect	Is the functionality implemented correctly in the prototype?
Relevance/ Acceptance	necessary / arbitrary / unwanted	Is the functionality necessary (i.e., desired by the stakeholders), arbitrary (i.e., neither desired nor unwanted by the stakeholders), or unwanted (i.e., it must not implemented)?

2.3 Applying the Prototype's Validation Results to the System Specification

Since the stakeholders can only validate the prototype's behavior and not the intended system's behavior, each validation result obtained from the stakeholder's validation activities needs to be checked with respect to its applicability to the original system specification itself. The previously documented traceability information in combination with the classification of validation results determine for which validation results this check can be automated, and which validation results have to be checked manually. For those validation results that can be checked automatically, defects as well as correct functionalities can be identified in the system specification in a fully automated manner.

Fig. 2 illustrates the steps involved in checking if a validation result is applicable to the system specification: (For a coherent example please refer to Section 3)

- First, for each validation result, the prototype specification is checked if the respective functionality is documented correctly therein (Step 4.1 in Fig. 2). Whether a certain functionality is documented correctly can be assessed from its respective attributes (cf. Section 2.2). Two combinations of attributes indicate a correctly documented functionality in the prototype specification:
 1. The functionality is *necessary* or *arbitrary*, it is *correctly* implemented w.r.t. the stakeholder intentions, and it is *present* in the prototype specification
 2. The functionality is *not necessary* and *not present* in the prototype specification. This case, however, is unlikely to be relevant as stakeholders are unlikely to identify any non-implemented behavior as unnecessary.
- If a functionality has been identified as correctly documented in the prototype specification, the documented traceability information (see Section 2.1) is used to identify corresponding parts in the system specification. Analyzing respective trace links facilitates checking whether functionality is correctly specified in the system specification (Step 4.2 in Fig. 2). This check leads to one of the following cases:

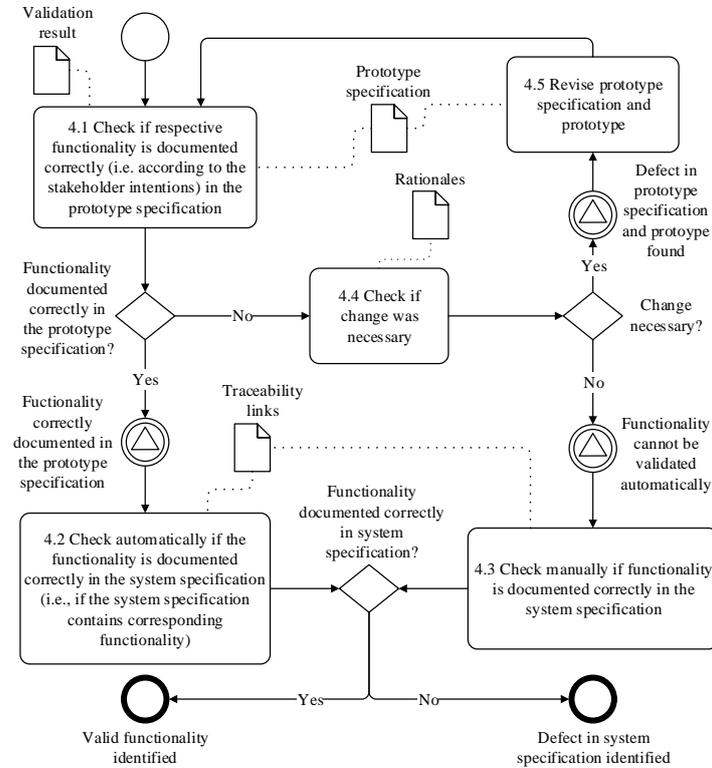


Fig. 2. Applicability check for validation results

- Case 1. The corresponding parts in the system specification can now also be considered correct with respect to the stakeholder intentions. In this case, a valid functionality has been identified.
- Case 2. The system specification contains a defect if the documented traceability information shows no corresponding part in the system specification. For unwanted functionality, which has been identified, corresponding parts in the system specification can be removed automatically and necessary functionality that is missing in the system specification can be added automatically if it is correctly documented in the prototype specification.
- If a functionality has not been documented correctly in the prototype specification, there could be two reasons for this. Either the prototype and its specification contain a defect, or the prototype and its specification had to be altered for a specific reason. Hence, it is necessary to check if there are reasons for changes (Step 4.4 in Fig. 2). Since all alterations, which were made when deriving the prototype specification, and the reasons for them were documented, it is easy to determine which of the following is the case:

Case 3. If a functionality is not documented correctly in the prototype specification and subsequently not implemented as desired in the prototype because the prototype's hardware is incapable of supporting this functionality, the rea-

son has been documented. Here, automated validation of the system specification is not possible, and the validation has to be done manually, i.e., the correctness of the functionality represented in the system specification needs to be checked manually (Step 4.3 in Fig. 2). Nevertheless, the insights gained from the stakeholders during the demonstration of the prototype may offer valuable support in these cases as well.

- Case 4. If there is no reason for a functionality not being documented correctly, a defect in the prototype, the prototype specification, and thus in the corresponding part of the system specification has been discovered. Instead of remedying this defect manually in the system specification and thereby risking introducing new defects, a manual correction of the prototype specification and the prototype (Step 4.5) enables the later validation of the correction in another iteration.

In addition to Fig. 2, Table 3 briefly summarizes these four distinct cases that can be identified in applicability checking. As can be seen, the validation results obtained can also aid in the correction of the system specification if defects have been discovered. In some cases, this correction can even be done automatically.

Table 3. Cases in applicability checking of validation results according to Fig. 2

Case	Functionality documented correctly?	No changes traceable?	Rationale documented?	Applicable to system specification?	Correct?
1	✓	✓	n/a	✓	✓
2	✓	✗	n/a	✓	✗ (Automated correction applicable)
3	✗	n/a	✓	✗	Manual review necessary
4	✗	n/a	✗	Revision of prototype and prototype specification necessary	

3 Evaluation Example: Avionic Collision Avoidance System

In this section, we will discuss the application of our approach to an avionic collision avoidance system. Therefore, we will provide details on each step of the approach as described in Section 2 and outlined in Fig. 1. We evaluated the applicability of the approach using an avionic proactive collision avoidance system (PCAS) as a case example. The PCAS shall exchange comprehensive flight data (i.e., complete flight plans), and propagate these flight information among the different aircraft forming a CPS network. This shall allow airplanes to prevent collisions proactively, i.e., long before a potential collision may occur. Some more details on the PCAS can be found in [15]. As hardware for the prototype, a quadcopter was chosen.

Based on the system specification of the collision avoidance system PCAS, we derived an adapted prototype specification for the chosen technology. Subsequently, we

implemented and deployed the software prototype to several quadcopters. Hence, multiple prototypes consisting of hardware and software parts were used to demonstrate the behavior of a collision avoidance system in a cyber-physical system network. The demonstration of the prototype's behavior allowed for validating parts of the system specification and revealed exemplary defects in other parts of the system specification, some of which have been corrected automatically.

3.1 Step 1: Derivation of the Prototype Specification

While the collision avoidance system was meant to be deployed in aircraft, the prototype was implemented using a quadcopter. These hardware differences necessitated several adaptations to the system specification. Fig. 3 shows an excerpt of the system specification for the PCAS and the corresponding part from the prototype specification. The differences between them are due to the fact that aircraft send out their ID via their secondary radar to recognize each other, while quadcopters can only recognize other quadcopters via tags using their cameras. For each change, we explicitly documented the traceability information using trace links (cf. Section 2.1).

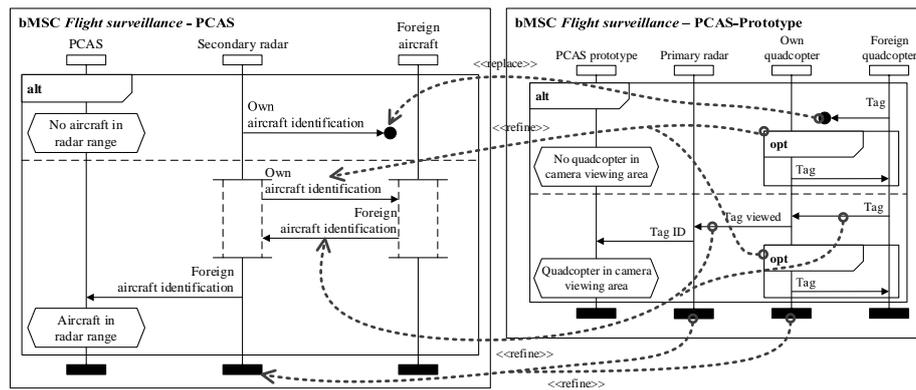


Fig. 3. System specification and prototype specification of the PCAS including trace links

3.2 Step 2: Development of the Prototype

We developed the prototype for the collision avoidance system according to the prototype specification, and tested it extensively to ensure correctness of the prototype w.r.t. the prototype specification. Beside the development of the software prototype for the collision avoidance system, it was also necessary to develop further software prototypes for several other avionic systems, which interact with the PCAS. The collision avoidance system relies on information provided by, e.g., the flight management system and the radar. Additionally, the system relies on other avionic systems, such as the flight control system to execute necessary flight maneuvers, for example. Hence, these necessary contextual systems had to be prototypically implemented as well, to ensure proper functioning of the collision avoidance system prototype.

The prototype as well as these other systems were implemented using the OSGI framework [16]. Fig. 4 illustrates the architecture of the embedded controller, which comprises the prototype's software of the PCAS and the other needed avionic systems, and thus simulates these interconnected embedded systems.

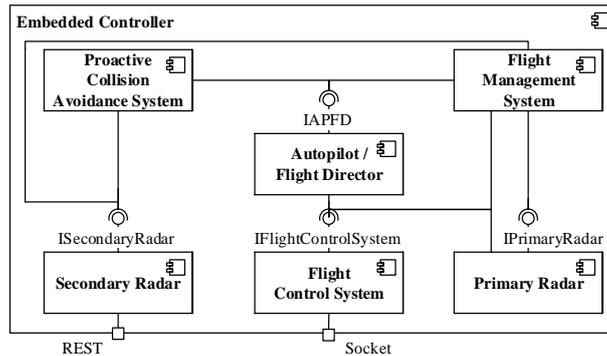


Fig. 4. Architecture of the embedded controller

Since the software could not be deployed on the quadcopters directly, for each quadcopter an instance of the embedded controller ran on an ordinary computer, which sends commands to their respective quadcopter via Wi-Fi. The control center monitors and controls the simulation scenarios. The resulting hardware architecture is depicted in Fig. 5.

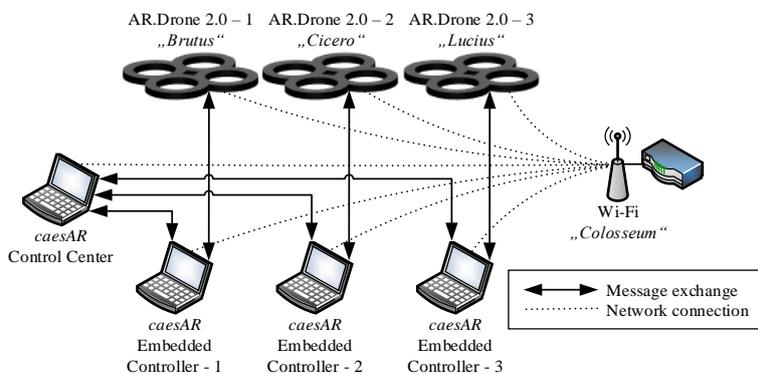


Fig. 5. Network architecture of the prototype system network

As can be seen, each quadcopter communicates with its own control logic on its embedded controller via Wi-Fi. These embedded controllers are interconnected and simulate the communication between the different quadcopters. In case of a detection of another quadcopter, the detecting quadcopter informs its embedded controller. Subse-

quently, the embedded controller negotiates adapted routes with the embedded controller of the detected quadcopter. Finally, both embedded controllers inform their quadcopters about necessary route changes.

3.3 Step 3: Classification of Validation Results

The next step is to obtain, document, and categorize results from the validation activities conducted using the prototype (cf. Section 2.2). An excerpt from the validation results obtained from the prototype’s demonstration is shown in Table 4. For example, the functionality evaluated in row 1 (*Aircraft Identification*) shows that the prototype is capable of identifying other aircraft (in this case other quadcopters). Hence, this functionality is present in the prototype. Since it is desired for the PCAS to identify aircraft in its vicinity, this functionality was furthermore categorized as necessary. However, as the prototype does not exchange aircraft IDs via the secondary radar, the stakeholders deemed the implementation of this functionality incorrect.

Table 4. Excerpt from the validation results for the collision avoidance system

#	Functional-ity	Attribute values	Description
1	Aircraft Identification	present incorrect necessary	The prototype uses cameras and tags for identification. This is incorrect as the stakeholders desire the aircraft to use the secondary radar for identification. This is due on the one hand to the presence of secondary radar in all aircraft over a certain size and on the other hand to the limitations associated with camera use such as unreliable results in case of bad visibility.
2	Collision Avoidance Trigger	present correct necessary	When the prototypical collision avoidance system identifies another quadcopter (prototype) / aircraft (system), it triggers the collision avoidance procedure, as is desired by the stakeholders.
3	Priority Parameter	(not) present correct necessary	The exchange of a parameter that determines the right of way between two potentially colliding quadcopter (prototype) / aircraft (system) ensures that the collision avoidance system in both aircraft arrive at the same conclusion as to which aircraft will give way.

For another example, the validation result concerning the collision avoidance trigger functionality (row 2 in Table 4) was categorized as present, correctly implemented in the prototype, and necessary.

As for the result referring to the functionality of exchanging a *Priority Parameter* (row 3 in Table 4), it became obvious during the development of the prototype that so far no decision had been made regarding the course adaptation. In particular, it had not been determined which aircraft should have to adapt its course and which could stay its course in case of a collision threat. Hence, the exchange of a parameter, which determines priority and thus right of way, was added to the prototype and its specification.

Consequently, this added functionality was categorized as present, correct, and necessary, although it is not present in the original system specification.

3.4 Step 4: Applicability Checking of Validation Results

To see if a validation result is applicable to the system specification, first it needs to be checked if the respective functionality is documented correctly in the prototype specification (see Section 2.3). If the functionality is documented incorrectly therein, it further needs to be checked whether there is a reason that prevents the correct implementation of the respective functionality in the prototype.

As can be seen in Table 4, the functionality for validation result 1 (*Aircraft Identification*) is not documented correctly in the prototype specification. However, as this functionality is not the same as in the system specification (cf. Fig. 3), but has been replaced due to hardware constraints, this does not indicate a defect in the system specification. Instead, this validation result is not applicable to the system specification and this functionality has to be validated manually (cf. Case 3 in Table 3).

The functionality for validation result 2 (*Collision Avoidance Trigger*) is documented correctly in the prototype specification. As there is no documented trace information that would indicate a difference between this functionality in the prototype specification and in the system specification, the documentation of this functionality in the system specification can be validated automatically (cf. Case 1 in Table 3).

The functionality for validation result 3 (*Priority Parameter*) is documented correctly as well. However, as previously described, this functionality has been added to the prototype specification and does not exist in the system specification. Since it corresponds to a functionality that was found to be a valuable extension to the original requirements, a defect in the system specification has been discovered here (cf. Case 2 in Table 3).

3.5 Step 5: Correction of the System Specification

The automated correction of the system specification is illustrated in Fig. 6 using the missing exchange of a priority parameter defect. To correct this missing functionality, the model elements referring to this functionality (highlighted in Fig. 6), which were documented in the prototype specification, are added to the system specification. Based on a correct prototype specification, this correction can be made automatically using model transformation techniques (cf. e.g. [17]).

4 Related Work

Since early validation of requirements is commonly considered of vital importance to avoid defective software and higher costs later on [1], several validation techniques have been proposed. In general, validation in the early phases deals with checking requirements against stakeholder intentions, e.g., to evaluate whether the desires of the stakeholders have been sketched correctly, to check whether all assumptions are valid,

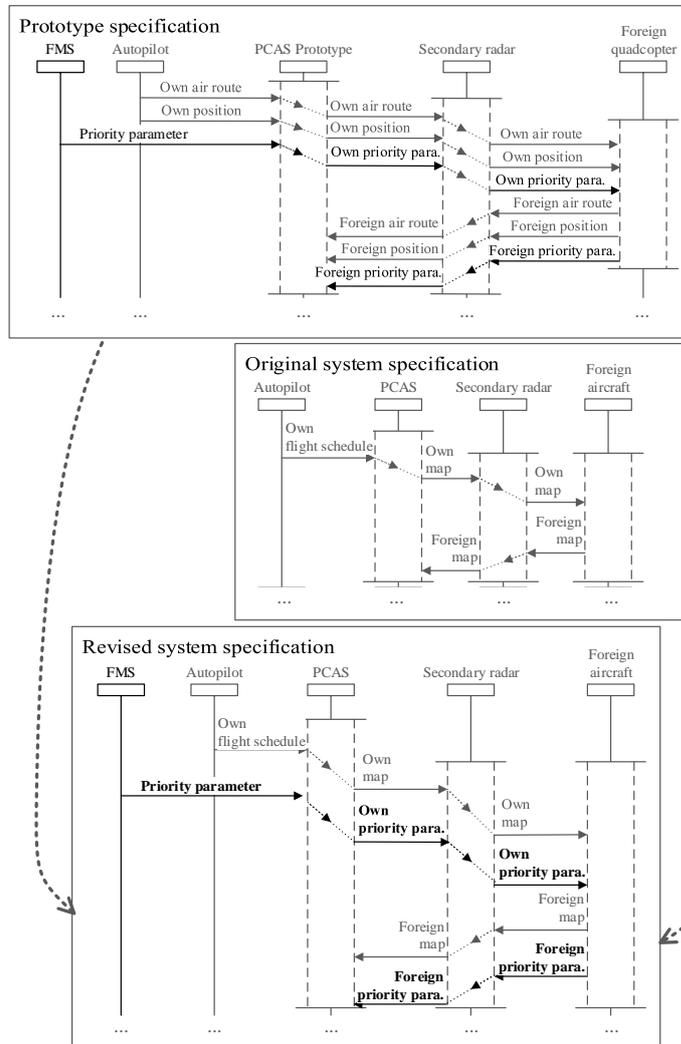


Fig. 6. Correction of the system specification

and the intended laws and regulations are considered. As automated verification is usually not applicable in early phases, attention is paid to techniques such as inspections [18], simulation [19], and prototyping [10].

Rapid prototyping is often suggested for early validation of first drafts and ideas as well as for eliciting new requirements (cf. [21], [4]). The latter is of particular importance for requirements engineering as it is well known that knowledge gained from the stakeholders (i.e. by experiencing technological possibilities from early system versions or prototypes) leads to changing requirements and new additional requirements [22]. Hence, prototypes can be used to iteratively evolve requirements specifications

(cf. [21]). Therefore, research was conducted on the use of prototypes for evolution of specification artifacts during prototype-based development (e.g., [23], [24], [25]). In these existing approaches, not the relationships between two specifications (i.e. the system specification and the prototype specification) are under investigation but the relation between increments of the system specification. While other works focus on the use of prototyping for evolution of a specification, the approach proposed in this paper uses explicit relations to a dedicated prototype specification for validating and correcting the original specification.

In general, prototyping aids in the validation of requirements as it enables stakeholders to experience the system's look and feel and its behavior. Thus, it not only allows stakeholders to express their opinions about existing requirements but also to identify missing requirements. To support the systematic evaluation of the prototype, pre-determined usage scenarios can be executed for prototype demonstration (cf. [26], [27]). Furthermore, the prototyping approach can be used to complement formal verification techniques with user feedback [28]. These kind of works focus on the systematic use of prototypes for validation. In doing so, they can complement our proposed solution approach, which does not guide the review but focuses on the systematic analysis and incorporation of review results.

Although prototyping is a well-known approach for validating requirements [29] it is mainly used for evaluating user interfaces of information systems (cf. e.g., [30]). Early validation of behavioral requirements for embedded and cyber-physical systems often relies on simulation, sometimes also called virtual prototyping (cf., e.g. [5], [31].), which not only allows for validating functional requirements, but also certain non-functional requirements, such as timing requirements (cf. e.g. [32]). In contrast to the validation of physical prototypes (e.g., prototype software for a new engine control unit deployed on an older hardware component), simulation does not take the real physical environment of the system into account, which is a strong demand for the validation of cyber-physical systems.

5 Discussion & Conclusion

In this paper, we presented a model-based approach to aid validation of requirements artifacts for cyber-physical systems. As shown, validation results for the prototype specification can aid in the identification of defects in the system specification. The extent to which validation results are applicable to the system specification depends on how many changes need to be made to the system specification when deriving the prototype specification due to differences in the technology used.

We evaluated our approach using a collision avoidance system from the avionics domain. For such safety-critical systems a prototypical validation approach is not sufficient on its own. In particular, it is still of utmost importance to validate the system itself and not only a prototype of the system. Nevertheless, the prototypical evaluation can aid the validation and the exploration of solution options in early phases of development. The approach is of particular interest for those domains (like the embedded

domain) which have a strong interest in early validation to decrease the number development cycles for highly complex software.

Beyond the validation of functional requirements, the prototypical implementation provides additional benefits for the development. For example, the prototypical implementation provided further insights about aspects that are usually not specified in requirements specifications but are important for the system's development. For instance, in the presented case example, the decision needed to be made as to what constitutes a collision threat. For quadcopters as well as aircraft, collisions can occur even if their flight paths never cross, but they only get too close to each other. Therefore, it was not sufficient to check for crossing flight paths and determine when each aircraft will reach this intersection. A reasonable safety margin around each aircraft also needed to be taken into account.

Acknowledgements.

Thanks to Stefan Beck, Arnaud Boyer, and Jürgen Meilinger from Airbus for their support in application of the approach to an industry example. The research serving as basis for this paper was funded by the *German Federal Ministry of Education and Research* (support code: 01IS12005C).

References

1. Boehm, B., Basili, V.R.: Software Defect Reduction Top 10 List. *Computer* 34, 135–137 (2001)
2. IEEE Standard Adoption of ISO/IEC 15026-4--Systems and Software Engineering--Systems and Software Assurance--Part 4: Assurance in the Life Cycle. (2013)
3. Ogata, S., Matsuura, S., Sakai, R., Sato, H., Kobayashi, T.: Enhancement of Requirements Specification Traceability by Model Driven Requirements Analysis Employing Automatic Prototype Generation. In: *IASTED Int. Conf. on Softw. Eng.*, pp. 55–63. (2011)
4. Kordon, F., Luqi: An Introduction to Rapid System Prototyping. *IEEE Trans. Softw. Eng.* 28, 817–821 (2002)
5. Aceituna, D., Do, H., Lee, S.-W.: Interactive Requirements Validation for Reactive Systems through Virtual Requirements Prototype. In: *MoDRE*, pp. 1–10. IEEE (2011).
6. Lee, E.A.: Cyber Physical Systems: Design Challenges. In: *11th IEEE Int. Symp. on Object and Component-Oriented Real-Time Distributed Computing*, pp. 363–369. IEEE (2008)
7. Jedlitschka, A., Jung, J., Lampasona, C.: Evaluation Summary. In: *Model-Based Engineering of Embedded Systems. The SPES 2020 Methodology*, pp. 231–239. Springer (2012)
8. Spanoudakis, G., Zisman, A.: Software Traceability: A Roadmap. In: *Handbook of Software Engineering and Knowledge Engineering*, pp. 395–428. World Scientific Publishing (2004)
9. ISO/IEC/IEEE: Systems and software engineering—Vocabulary. (2010)
10. Gotel, O.C.Z., Cleland-Huang, J., Hayes, J.H., Zisman, A., Egyed, A., Grünbacher, P., Dekhtyar, A., Antoniol, G., Maletic, J., Mäder, P.: Traceability Fundamentals. In: *Software and Systems Traceability*, pp. 3–22. Springer (2012)
11. Ramesh, B., Jarke, M.: Toward reference models for requirements traceability. *IEEE Trans. Softw. Eng.* 27, 58–93 (2001)

12. Cote, I., Heisel, M.: A UML Profile and Tool Support for Evolutionary Requirements Engineering. In: 15th European Conf. on Software Maintenance and Reengineering (CSMR), pp. 161–170. IEEE (2011)
13. Chillarege, R., Bhandari, I.S., Chaar, J.K., Halliday, M.J., Moebus, D.S., Ray, B.K., Wong, M.-Y.: Orthogonal Defect Classification - A Concept for In-Process Measurements. *IEEE Trans. Softw. Eng.* 18, 943–956 (1992)
14. Freimut, B., Denger, C.: A Defect Classification Scheme for the Inspection of QUASAR Requirements Documents. IESE-Report No. 076.03/E, Version 1.0. Fraunhofer IESE (2003)
15. Daun, M., Brings, J., Bandyszak, T., Bohn, P., Weyer, T.: Collaborating Multiple System Instances of Smart Cyber-physical Systems: A Problem Situation, Solution Idea, and Remaining Research Challenges. In: IEEE/ACM 1st Int. WS on Softw. Eng. for Smart Cyber-Physical Systems (SEsCPS), pp. 48–51. IEEE (2015)
16. Specifications – OSGi™ Alliance, <https://www.osgi.org/developer/specifications/>
17. Milicev, D.: Automatic Model Transformations Using Extended UML Object Diagrams in Modeling Environments. *IEEE Trans. Softw. Eng.* 28, 413–431 (2002)
18. Fagan, M.E.: Design and code inspections to reduce errors in program development. *IBM Syst. J.* 15, 182–211 (1976)
19. Brandstetter, V., Froese, A., Tenbergen, B., Vogelsang, A., Wehrstedt, J.C., Weyer, T.: Early Validation of Automation Plant Control Software using Simulation Based on Assumption Modeling and Validation Use Cases. *CSIMQ* 4, 50–65 (2015)
20. Andrews, B.A., Goeddel, W.C.: Using Rapid Prototypes for Early Requirements Validation. In: 13th AIAA/IEEE Digital Avionics Systems Conference, pp. 70–75. IEEE (1994)
21. Luqi: Software Evolution Through Rapid Prototyping. *Computer* 22, 13–25 (1989)
22. Nuseibeh, B.: Weaving together requirements and architectures. *Computer* 34, 115–119 (2001)
23. Berzins, V., Luqi, Yehudai, A.: Using Transformations in Specification-Based Prototyping. *IEEE Trans. Softw. Eng.* 19, 436–452 (1993)
24. Berzins, V.: Recombining changes to software specifications. *Journal of Systems and Software* 42, 165–174 (1998)
25. Luqi, Chang, C.K., Zhu, H.: Specifications in software prototyping. *J. of Syst. and Softw.* 42, 125–140 (1998)
26. Sutcliffe, A.: A Technique Combination Approach to Requirements Engineering. In: 3rd IEEE International Symposium on Requirements Engineering, pp. 65–74. IEEE (1997)
27. Kotonya, G., Sommerville, I.: Requirements Engineering: Processes and Techniques. John Wiley & Sons; J. Wiley (1998)
28. Siddiqi, J., Morrey, I., Hibberd, R., Buckberry, G.: Towards a System for the Construction, Clarification, Discovery and Formalisation of Requirements. 1st Int. Conf. on Requirements Eng., pp. 230–238. IEEE (1994)
29. Andriole, S.J.: Fast, cheap requirements prototype, or else! *IEEE Softw.* 11, 85–87 (1994)
30. Kamalrudin, M., Grundy, J.: Generating Essential User Interface Prototypes to Validate Requirements. In: 26th IEEE/ACM Int. Conf. on Automated Softw. Eng. (ASE), pp. 564–567. IEEE (2011)
31. Thompson, J.M., Heimdahl, Mats P. E., Miller, S.P.: Specification-Based Prototyping for Embedded Systems. In: ESEC/FSE '99, LNCS, vol. 1687, pp. 163–179. Springer (1999)
32. Zimmermann, J., Stattelmann, S., Viehl, A., Bringmann, O., Rosenstiel, W.: Model-Driven Virtual Prototyping for Real-Time Simulation of Distributed Embedded Systems. In: 7th IEEE Int. Symp. on Industrial Embedded Systems (SIES'12), pp. 201–210. IEEE (2012)