Certification as a service

Sebastian Copei[1⋆] and Manuel Wickert[2⋆] and Albert Zündorf[3]

[1] Kassel University, Kassel, Germany,
sco@uni-kassel.de
[2] Frauenhofer IEE, Kassel, Germany,
manuel.wickert@iee.frauenhofer.de
[3] Kassel University, Kassel, Germany,
zuendorf@uni-kassel.de

Summary. The development of industry 4.0 and smart energy IT-Components relies on highly standardized communication protocols to reach vendor-independent interoperability. The interoperability is required to build complex systems without vendor lock-in and stay in operation for many years without frequent hardware changes. However, the classical standardization and certification processes for such communication protocols are typically incompatible with modern agile software development processes. This increases the complexity of building microservice-based systems or cloud-native solutions in these domains. To cope with that, we propose an agile method for the standardization of communication protocols. We show that this approach will be compatible with agile development processes and present a cloud-native certification service architecture that works within a continuous integration and deployment pipeline, seamlessly. This will support the efficiency of building microservices and cloud-native applications for the industry 4.0 and smart energy domain.

Key words: microservices, certification, saas, agile, software as a service, platform as a service

1 Motivation

Vendor independent scaling of distributed systems is often very complex. One of the key challenges is using the same interfaces, protocols, and data models for inter-service communication. For industrial processes as well as for energy system technology this is very often the motivation for standardization of protocols such as (OPC UA [13], IEC 61850 [9], IEC 60870-5-104 [8], etc.). These standardized protocols enable interoperability between different vendors of software systems or integrated devices. The use of a standard is also helpful as a sales argument for the vendors. For the customers, buying a product with standard interfaces reduces the dependency to a certain company. Especially for industrial processes or in the energy domain where integrated devices stay in operation for years or decades, the independence of a certain company is crucial.

⋆ These two authors contributed equally

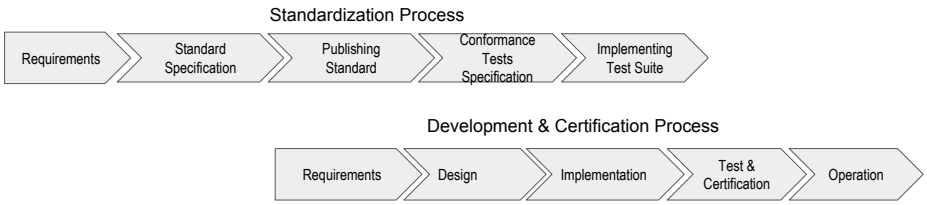

Fig. 1. Classic Standardization and Product Development Processes

Figure 1 shows a classical standardization and certification scenario divided into two processes. The first (upper) process shows a view on the development of a new version of a standard. The second process shows a classical waterfall model for applications where the certification is part of the testing phase. Note, both process views are very coarse overviews and do not provide a detailed look at a certain complex standardization or certification scenario.

The standardization typically begins with a collection of requirements by a standardization organization. For these requirements, a communication standard is developed. Usually, the communication standard only describes the communication of a particular layer of the ISO/OSI communication stack. When a new version of the standard is completed, the standard is published. After publishing, conformance tests may be specified and test suites may be implemented. E.g. The OPC Foundation offers a conformance test suite for its members [14]. The development and certification of actual applications and components starts after the publishing of the standard. Applications and components may be designed after a requirement phase. Afterward, the implementation of standard communication interfaces starts. In order to claim conformance to the standard, new applications and components may need to be tested according to the conformance tests and get certified by an authorized certification body.

The important message of Figure 1 is that the development process can start only after (a new version of) the standard has been published. If the software development starts earlier, there is the risk that the resulting product is not compatible with the new version of the standard. Waiting for the standard to be published increases the time to market for interoperable software solutions that use the communication standard. Thus, the whole process from the emergence of the need for a new protocol to the availability of certified applications and components may easily take several years. The reason is that classical standardization processes rest on a linear software development model like the waterfall or V model. As a result, new standard versions are not often published e.g., IEC 61850-1 is published in 2003 in version 1.0 and in 2013 in version 2.0. Therefore agile development processes such as SCRUM, eXtreme Programming, or Kanban are not supported by theses standardization processes.

Smart Energy Applications, as well as Industry 4.0, are connecting classical industrial monitoring and control solutions with modern IoT-based technologies. Thereby modern software development processes are applied to address fast-changing requirements in both sectors. To be innovative, the development

and deployment cycles have to be very short. This knowledge is based on our experience with more than 15 different research and application projects in the energy sector. Therefore we reconsidered how standardization and certification processes can be integrated into an agile product development process.

This paper presents a new standardization and certification approach for communication protocols. This approach primarily addresses the definition of communication standards and the certification of software systems to prove their compliance with such standards. In contrast to Figure 1, we focus here on the certification part of the development process. In the following, we will use the term standardization to describe the process to develop and publish a new standard, including the specification of conformance tests and the implementation of test suites. The term certification is used to describe the relevant parts in the development process of applications to test a new version of an application for compliance to a standard and get a certification for this new version.

2 Related work

Agile standardization and certification processes have already been examined in various domains. Examples are high security system certification for aviation[3] and railways [1]. The authors of [3] present a way to certify security-critical components in a transportation system. They focus on high-level certificates. To provide the credibility of the certificates, the authors use a semi-formal description language. [1] shows a way to certify security-critical aerospace components. The authors use UML as a modeling tool to provide an incrementally changeable model description to achieve an agile certification process. However, the solutions presented in both papers are very domain-specific and focused on security certification. The given solutions only fit into their use cases and can not be used as a general approach. Furthermore, the solutions only cover the certification process in a client-side. Our solution want to cover the whole process from developing a standard to certifying implementations of it.

An evolutionary standardization approach for file based data is presented in [4]. The considered standardization focus is the engineering of automation systems. The basic idea is to start from an existing proprietary file format of one vendor and change it evolutionary to a neutral and later on to a common format, apparently often XML in that context. Similar to our process, this approach proposes a stepwise standardization. Nevertheless, the evolutionary approach is not intended to support agile development processes in the development of tools. In contrast to our approach, it focuses on file-based communication.

In [2] an agile standardization was performed for Process Control Equipment (PCE). The domain is close to the considered domain of this work. The authors require that standardization has to be done agile and "should proceed stepwise". However, the focus of [2] is the concrete standardization of PCE Requests, not the standardization process itself.

This paper presents a new process for standardization and certification of communication standards. We tried to specify a very generic approach that

should be transferable to other domains as well. This is achieved through the "Everything as a service" principle[5]. Furthermore, our solution is designed to run easily in a public or private cloud, which allows very fast adaption.

3 The Agile Standardization and Certification Processes

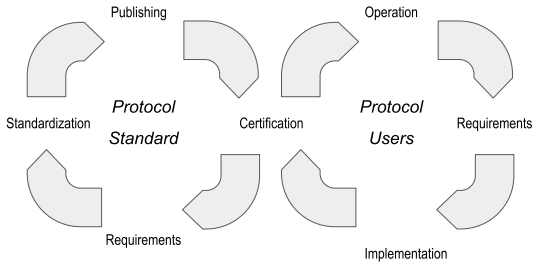

Fig. 2. Agile Standardization Process

We propose an agile standardization and certification process that has two interwined development cycles, cf. Figure 2. As in other agile approaches, standardization and certification should be performed in small increments. The basic idea is to start with a minimal set of communication protocol features (e.g., establish a connection or login to a server) and add feature by feature in several iterations. Every iteration ends with a minor version change in the standard. The corresponding part of the overall standard is published e.g., via Github or some other configuration management service. Based on the publication of the standard for some features, the standard conformance tests that certify compliance with these features are extended or adapted and again published via a configuration management service. The standardization process runs iteratively, i.e., as soon as one feature has been completed, subsequent application development may start while the standardization continues with the next features.

The product development cycle, including the certification of a product, is shown on the right side of Figure 2. The development of standard-compliant products may start with the requirements definitions for certain product features. The implementation of these product features may follow this. As soon as an implementation of some feature is available, this feature implementation may try to pass the corresponding protocol conformance tests for a specific communication standard version. As soon as the new product version is certified, it may be released and operated in production.

Each time a new version of the standard is exposed, and the corresponding conformance tests are deployed a test-driven development iteration of the products is ready to begin. Obviously, the conformance test will not be able to provide a complete test set for a product. However, these tests will support the product

development relating to the communication interfaces. This approach has the advantage that first conformance tests will be available soon after the first iterations of the standardization process have completed. Thus, product development and standard development may be interwined. Thereby, standard-compliant products will be available soon after the standard has reached a sufficient level of completeness. In addition, product development may provide feedback to the standardization process. Product development may e.g., point to overly complex conformance tests or inconvenient APIs or missing details, etc. This feedback may be used by the standardization process to enhance the standardization of the corresponding features and to come up with improved versions of the conformance tests. The importance of such feedback is also discussed in [4].

On the other hand, new versions of conformance tests that have already been passed by some product may require to run these performance tests again and perhaps to adapt already completed features to the new requirements. Such changes to already defined conformance tests may also happen when following features or later standardization iterations require previously standardized features to evolve. This is an infrequent problem inherent in agile software development. If a product development team wants to avoid such issues, it may wait with its work until the standardization process has reached a sufficient level of completeness and stability. One can argue that this may be a drawback of our approach since stability is an important requirement for communication devices in the field. However since we have also consider security for such field devices we have to provide easy mechanisms to provide software updates in operation. In addition there are a lot of regulatory changes in the energy sector that have implications to the communication interfaces. Therefore adapting the communication interfaces very often will come to order within the next years.

4 Certification as a Service Architecture

For a certain standard, a certification service will support the agile standardization and certification process. Here we propose a microservice [12, 7] based certification as a service architecture. This architecture should support the understanding of our agile standardization and certification process on the one hand. On the other hand, we built some basic prototypes based on this architecture to evaluate the standardization. However, we will focus on the architecture here because the implementation and evaluation of a certain standard are still work in progress.

Continuous integration and continuous deployment are methods to support fast feedback during agile development. A Certification as a service implementation extends a typical continuous deployment pipeline, as shown in Figure 3. The certification step should be performed after the integration phase (which includes integration testing). The certification step consists of the execution of the conformance tests and the creation of a certificate. An implementation of our certification as a service platform will perform this step. This allows the deployment of certified products in every continuous deployment cycle. If conformance

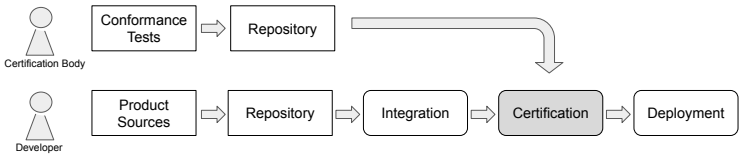

Fig. 3. Continuous Certification Pipeline

tests fail, the pipeline stops at the certification step, just like a failure during integration tests will stop the pipeline.

Each certification pipeline certifies a product according to a particular standard version. Whenever a new standard version is published, the respective conformance tests will be adapted or extended for the new version of the standard. The certification bodies will add the standard to a repository. As soon as the new tests have uploaded, a product can be certified for the new standard version.

The certification service itself should be hosted as a service by the standardization or depending on organizational aspects, a certification body. As software as a service (SaaS), it should be compatible with a typical build pipeline software such as Jenkins. That allows an independent certification of products even with fast development cycles.

Our proposal for the certification service architecture is shown in Figure 4. We defined five microservices, two repositories, and an event broker.

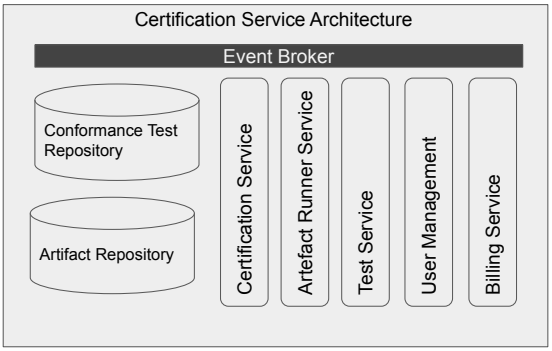

Fig. 4. Certification Service Architecture

The repositories are responsible for storing a product for certification (artifact repository) and the conformance tests (conformance test repository). Both artifacts and conformance tests should be available in different versions. To perform the conformance tests, an instance of the artifact should be up and running for certification. The "Artifact runner Service" is responsible for running this artifact and configure it correctly. The execution of the conformance tests will be done by the "Test Service". This service will provide test results for the "Certification Service". The certification service will create a certificate for the artifact if

all tests are passed successfully. The "User Management" and "Billing Service" have administrative responsibilities. Since the business model of a certification body is to issue certificates, it is necessary to implement user management and billing functionalities. The communication to the product development should be done by RESTful HTTP. RESTful HTTP is supported by typical build pipeline products. For internal communication, event sourcing should be used. Therefore we suggest making use of an event broker like Apache Kafka.

Our architecture aims to provide a proposal for a certification as a service solution. Typical container orchestration tools can support implementations. Therefore an implementation of our service should be cloud-native [15, 11, 6].

5 Conclusion and Future work

We presented a new way to achieve a more agile process during the standardization and certification steps. We provide a certification as a service architecture that should support the affected stakeholders during the whole process. This means, on the one hand, that a standardization organization should have the possibility to provide fast incremental updates of their standards. On the other hand, we enable companies to use agile development processes for their certified implementation of standardized communication interfaces.

The paper is part of our work in the European research project InterConnect EU [10]. In the next steps, we will implement the certification as a service architecture for a new communication standard in the context of e-mobility. We will examine how agile standardization approaches will work in that context. Furthermore, we will evaluate how this approach will support the agile development of prototypes for e-mobility use cases.

Acknowledgement

This work is part of the interconnect project [10] which has received funding from the European Union's Horizon 2020 research and innovation program under grant agreement No 857237.

References

1. Sergio Bezzecchi, Paolo Crisafulli, Charlotte Pichot, and Burkhart Wolff. Making agile development processes fit for v-style certification procedures. *CoRR*, abs/1905.06604, 2019. URL: `http://arxiv.org/abs/1905.06604`, http://arxiv.org/abs/1905.06604 `arXiv:1905.06604`.
2. P. G. Bigvand, R. Drath, A. Scholz, and A. Schüller. Agile standardization by means of pce requests. In *2015 IEEE 20th Conference on Emerging Technologies Factory Automation (ETFA)*, pages 1–8, 2015.

3. David J Coe and Jeffrey H Kulick. A model-based agile process for do-178c certification. In *Proceedings of the International Conference on Software Engineering Research and Practice (SERP)*, page 1. The Steering Committee of The World Congress in Computer Science, Computer . . . , 2013.
4. R. Drath and M. Barth. Concept for managing multiple semantics with automationml — maturity level concept of semantic standardization. In *Proceedings of 2012 IEEE 17th International Conference on Emerging Technologies Factory Automation (ETFA 2012)*, pages 1–8, 2012.
5. Y. Duan, G. Fu, N. Zhou, X. Sun, N. C. Narendra, and B. Hu. Everything as a service (xaas) on the cloud: Origins, current and future trends. In *2015 IEEE 8th International Conference on Cloud Computing*, pages 621–628, June 2015. http://dx.doi.org/10.1109/CLOUD.2015.88 doi:10.1109/CLOUD.2015.88.
6. C. Fehling, F. Leymann, R. Retter, W. Schupeck, and P. Arbitter. *Cloud Computing Patterns*. Springer Vienna, Vienna, 2014.
7. Martin Fowler and James Lewis. Microservices. 2014. URL: http://martinfowler.com/articles/microservices.html.
8. IEC 60870-5-104 - Telecontrol equipment and systems. Standard, International Electrotechnical Commission, Geneva, CH, 2006.
9. IEC 61850 Standard Series, Communication networks and systems in substations. Standard, International Electrotechnical Commission, Geneva, CH, 2020.
10. Interconnect project - homepage. Last viewed 20.04.2020. URL: https://interconnectproject.eu/.
11. Nane Kratzke. A brief history of cloud application architectures. *Applied Sciences*, 8, 08 2018. http://dx.doi.org/10.3390/app8081368 doi:10.3390/app8081368.
12. Sam Newman. *Building Microservices*. O'Reilly Media, Inc., 1st edition, 2015.
13. OPC Unified Architecture, IEC 62541, Standard Series. Standard, OPC Foundation, International Electrotechnical Commission, Scottsdale, USA, 2008.
14. Opc foundation test tools. Last viewed 20.04.2020. URL: https://opcfoundation.org/developer-tools/certification-test-tools/opc-ua-compliance-test-tool-uactt.
15. Claus Pahl, Pooyan Jamshidi, and Olaf Zimmermann. Architectural principles for cloud software. *ACM Transactions on Internet Technology*, 18, 06 2017. http://dx.doi.org/10.1145/3104028 doi:10.1145/3104028.