Towards a Taxonomy for Eliciting Design-Operation Continuum Requirements of Cyber-Physical Systems

Jon Ayerdi*, Aitor Garciandia†, Aitor Arrieta*, Wasif Afzal‡, Eduard Enoiu‡, Aitor Agirre†,
Goiuria Sagardui*, Maite Arratibel§ and Ola Sellin¶
University of Mondragon*, Ikerlan †, Mälardalen University‡, Orona§, Bombardier Transportation¶,
*{jayerdi,aarrieta,gsagardui}@mondragon.edu, †{agarciandia, aagirre}@ikerlan.es, ‡{wasif.afzal,
eduard.paul.enoiu}@mdh.se, §marratibel@orona-group.com, ¶ola.sellin@rail.bombardier.com

***Abstract*—Software systems that are embedded in autonomous Cyber-Physical Systems (CPSs) usually have a large life-cycle, both during its development and in maintenance. This software evolves during its life-cycle in order to incorporate new requirements, bug fixes, and to deal with hardware obsolescence. The current process for developing and maintaining this software is very fragmented, which makes developing new software versions and deploying them in the CPSs extremely expensive. In other domains, such as web engineering, the phases of development and operation are tightly connected, making it possible to easily perform software updates of the system, and to obtain operational data that can be analyzed by engineers at development time. However, in spite of the rise of new communication technologies (e.g., 5G) providing an opportunity to acquire Design-Operation Continuum Engineering methods in the context of CPSs, there are still many complex issues that need to be addressed, such as the ones related with hardware-software co-design. Therefore, the process of Design-Operation Continuum Engineering for CPSs requires substantial changes with respect to the current fragmented software development process. In this paper, we build a taxonomy for Design-Operation Continuum Engineering of CPSs based on case studies from two different industrial domains involving CPSs (elevation and railway). This taxonomy is later used to elicit requirements from these two case studies in order to present a blueprint on adopting Design-Operation Continuum Engineering in any organization developing CPSs.**

***Keywords*-DevOps, Design-Operation, Requirements Elicitation, Cyber-Physical Systems**

I. Introduction

Cyber-Physical Systems (CPSs) integrate computation with physical processes whose behavior is defined by both physical and software parts of the system [1]. While the cyber-physical controller consists of discrete software, the physical layer is composed of parallel physical processes running in continuous time. The cyber layer is composed of computational platforms and networks that are in charge of monitoring and controlling physical processes [31]. These systems are part of many products we use in our daily life, such as vehicles, airplanes, elevators and trains. As the lifecycle of these systems is rather long, all their components require maintenance, including their software components. Given that the software of these systems is usually extremely complex, the software constantly evolves during the CPS lifecycle based on several aspects [18], such as (1) new functional and non-functional requirements, (2) hardware obsolescence and/or system degradation, and (3) correction of bugs detected while the system is operating.

In the last few years, there have been several improvements in terms of modeling and simulation techniques [2], [15], [20] to develop and validate complex CPSs from the early development stages. However, when the software is deployed in the CPS, the methods used during operation and maintenance do not have synergies with the methods used in development. In other contexts, such as web-engineering, there are Design-Operation Continuum Engineering methods such as DevOps that permit software development methods to be streamlined with methods for operation time. DevOps practices efficiently integrate development and operations, aiming at shortening the lead time between a change request and the deployment in production using automation, agile software development and continuous delivery (CD) pipelines. Yet, for CPSs, traditional Design-Operation Continuum Engineering methods require substantial changes in order to be dependable enough. More specifically, Design-Operation Continuum methods must provide solutions in order to have a more efficient process which guarantees that (1) software updates are performed safely and securely, (2) most of the faults are detected in the design phase before the software is deployed in the CPS and (3) problems that can emerge in operation can be reproduced in development in order to analyse and propose potential solutions.

In order to start developing Design-Operation Continuum Engineering methods supported by the appropriate tools for CPSs, a taxonomy of relevant concepts is expected to ease the understanding of their rather complex development and maintenance process. In this paper, we build and instantiate such a taxonomy in order to assist requirements analysts with the identification and categorization of the requirements related to different aspects of the CPS Design-Operation Continuum Engineering. The main purpose of this classification is supporting the elicitation of new requirements and the easier identification of problems such as omissions, ambiguity, vagueness, conflicts or duplication in the requirements. Furthermore, this classification is also helpful for determining the organisational roles responsible for each requirement, as well as for the management and reuse of the elicited requirements in later stages of the development lifecycle. This taxonomy is inspired by case

studies from two different industrial domains: the elevation domain and the railway domain. Both case studies are provided by companies that are leaders in their sectors. By analyzing the data provided by these companies through their documentation (e.g., internal technical documents, repositories, code, etc), as well as through interviews with their engineers, we were able to develop a general purpose taxonomy for Design-Operation Continuum Engineering methods for CPSs. With this taxonomy, organizations can instantiate their domain-specific categorization and classification of requirements.

The remainder of this paper is structured as follows: Section II describes the taxonomy development process. Section III presents the two industrial case studies that inspired this work, which are also the first systems where this taxonomy has been applied. Section IV describes the developed taxonomy in detail. Section V describes the process of requirements elicitation using the taxonomy. Section VI reviews related work and Section VII concludes the paper.

II. Taxonomy Definition Method

To develop the taxonomy, we followed the guidelines proposed by Ralph [24], which provide a set of steps to follow. The first step refers to *choosing the strategy*. We opted for using the "grounded theory and interpretative case study" approach as our main strategy, in addition to personal experiences acquired by the long-term collaboration between the industrial and academic partners involved in this paper. In this case, we analyzed two case studies from different domains, which permitted us to identify their commonalities and differences. We do not expect that our taxonomy can be generalized to all CPSs by using only two industrial case studies as a basis, but at least, we believe that it provides fundamental evidence that it could be adopted for many complex and industry-relevant cases. On the other hand, we unavoidably made use of personal experience to an extent in order to develop this taxonomy, so a certain degree of bias can be expected.

The second step is the *site selection*. Two sites were selected for developing the taxonomy, which are two organizations of CPS developers: Orona (elevation domain) and Bombardier (railway transportation domain). We chose these sites due to several reasons. Firstly, there is a long-standing collaboration between the researchers and practitioners that work for these organizations. Secondly, both sites are developers of complex CPSs. Thirdly, the domains are sufficiently different (i.e., elevation and railway) to ensure a minimum degree of heterogeneity for the development of a general taxonomy. Lastly, and most importantly, both sites are relevant target users for the taxonomy, they produce rich data and detailed explanations for their requirements, and are accessible to the authors of this paper.

As for the *data collection*, which is the third step proposed in the guidelines [24], two processes were followed. On the one hand, the direct observation methodology [24] was employed. On the one hand, we reviewed internal documentation provided by both Orona and Bombardier. On the other hand, we interviewed participants from the companies involved. To this end, we prepared a set of relevant questions carefully selected by the researchers. We later interviewed practitioners from Orona with various positions and experience levels, who would directly benefit from adopting Design-Operation Continuum methods into their development processes. As we were creating a taxonomy from scratch, similar to [13], the interview questions had to be as generic and open-ended as possible. Therefore, we opted for semi-structured interviews [28], which combine open-ended questions with specific questions. Thus, the interviewer had to improvise new questions based on the interviewee's response. Additionally, we used internal documentation from both industrial companies, including test plans, comments from the code repository, standards, etc. More information related to the data collection from each of the case studies is given in Section III.

The fourth step is related to the *data analysis* [24]. We took notes based on (1) the interviews to engineers working in Orona, and (2) by accessing internal documentation at Orona and Bombardier. We then used an iterative approach to code our notes and build the taxonomy. We initially developed a first structure of the taxonomy by having reviewed the state-of-the-art on Design-Operation Continuum methods. We later evolved this initial taxonomy with the information extracted from the interviews as well as the internal documentation of the case companies.

The last step refers to the *conceptual evaluation*. Similar to [13], in order to ensure that the final taxonomy was comprehensive and representative, we validated the taxonomy by involving both researchers and industrial participants. These were different from those involved in the interviews. The participants were asked to (1) identify potential weaknesses of the theory, as suggested by Ralph [24], and (2) provide evaluation criteria based on the credibility and transferability of the taxonomy.

III. Case Studies

In this section, we describe the two industrial case studies used to extract the taxonomy. One of the case studies is from the vertical transport (elevation) domain, whereas the other one is from the railway domain. Both companies that provide the case studies are leaders in their domain, and therefore, the technology that they use is cutting-edge. In this section, we explain the subsystems considered to build the taxonomy of the paper and its current software development process, the specific methodology followed in each of the case studies for developing it, and how both companies expect to improve their software development process by adopting Design-Operation Continuum methods.

A. Case study from the elevation domain

Orona's activities are focused on the design, manufacturing, installation, maintenance, and modernisation of elevators, escalators, and moving ramps and walkways. An elevator installation is a complex CPS composed by a set of elevators that interact to provide service to passengers with the goal

of minimising the Average Waiting Time (AWT) and, more recently, also taking into account other criteria such as energy consumption, transport capacity, or overall transit time. Nowadays, over 250.000 elevators worldwide use Orona's technology. As most of the new functionality in elevators' installations is provided by software, Orona has a systematic and well established process for the development and release of new software versions.

The traffic master manages the passenger flow. It is composed by several software modules such as the dispatcher, which executes the traffic algorithm to allocate calls to elevators, the signalling to guide passengers (e.g., by communicating the assigned elevator), or the access control which disables specific floors for unauthorized passengers. The traffic master is constantly evolving in order to improve the service by including new functionalities or adapting to the building requirements. In conclusion, this system is a good candidate for the adoption of Design-Operation Continuum methods.

Interviews to the dispatcher manager, two software engineers and a system validation engineer were carried out by the researchers. The dispatcher manager defines new functionalities and analyzes poor performance in installations. The software engineers develop and validate the software. Lastly, the validation engineer tests the dispatcher in the Elevator.

Figure 1 illustrates the process (current tasks, roles and tools) extracted from the analysis of the data collected during the interviews. Within the requirement elicitation of a new release (1), a rigorous validation plan is defined comprising three main validation phases. The *Software-in-the-Loop (SiL)* phase usually encompasses most of the development work for a new functionality. The software produced in this step (2) (depicted as system under test (SUT)) is validated (3) in a purely virtual environment using a domain specific simulator (i.e., Elevate™).[1] At the SiL phase, tests ensure the quality of service requirements (e.g., AWT over time). The *Hardware-in-the-Loop (HiL)* phase (4-5-6) follows the previous SiL phase. In this phase, both virtual and real components are mixed together to compose an integrated scenario that is very close to the real one. Simulators that are used at the SiL phase are substituted by real hardware (e.g., elevator controllers) and communication networks, enabling integration tests of the entire system. Nevertheless, some parts may still be simulated (e.g., elevator shaft simulator, passenger demand, etc.). At the HiL phase, test cases check the functional correctness of the release. Finally, the software is deployed into the real system, *operational phase* (7-8), and eventually monitored by maintenance staff (9). Some installations require a deep analysis in order to understand the perception of the customers (10). This analysis is performed by trying to reproduce the situations observed in reality in simulations at the SiL phase.

Executing the validation plan both at SiL and HiL phases follows a similar sequence: (a) deploy to validation infrastructure; (b) configure the context (i.e., the type of building, number of floors, etc.); (c) define and configure data to monitor; (d) execute the test cases; (e) analyze the data; (f) decide whether the version is ready for the next phase. Configuration, deployment, analysis and decision are mainly manual steps, while the execution of test cases at HiL is semi-manual. Therefore, these tasks are error-prone, require significant effort and are dependent on specific knowledge and skills. This is especially exacerbated at the HiL phase, where a configuration of the validation infrastructure requires special knowledge and can take hours to ensure a proper configuration and deployment. Besides, test cases are executed in real-time, and thus, on top of the test execution times being potentially long, the availability of an engineer is required during this process.

Once in operation, feedback for new requirements or bug fixes is received from (a) customers, (b) monitors of the CPS and (c) regulation changes. Checking customers' feeling of poor performance about the installation (i.e., the feeling that some passengers are waiting too long) by reproducing the scenario in the domain-specific simulator is always a time consuming and cumbersome task. Sometimes, an in-situ monitoring process of the installation is required for a limited period of time, which is extremely costly. Besides, it is not always possible to reproduce the situation, and therefore, a deep analysis to identify differences has to be performed.

B. Case study from the railway domain

Bombardier Transportation (BT) is one of the leading companies in the rail industry, providing rolling stock and associated services of system maintenance, signalling, fleet management and asset life management. It has a broad and innovative product portfolio in rolling stock, consisting of e.g., light rail vehicles, metros, commuter and regional trains. For the definition of Design-Operation Continuum requirements, we have selected BT's Train Control and Management System (TCMS). This system is the center of the distributed system that controls the train. It is involved in almost all train functions, either in a controlling or a supervisory capacity. Examples of train functions controlled and managed by the TCMS are collecting line voltage, controlling the train engines, opening and closing the train doors and uploading diagnostic data. For collecting data on BT's current status of the development of TCMS as well as finding out opportunities of Design-Operation Continuum methods for testing and deployment, we made use of archival data as one of our first steps. We had access to internal documentation of a relevant BT project. Documentation such as test plans were read and analyzed. We used document analysis [4] as a systematic procedure for reviewing or evaluating these documents. Two experienced researchers, with extensive experience in research projects with BT, were involved in the analysis.

At BT, a test plan documents the scope, approach, resources and schedule of the testing activities per project. The deployment plan is also partly reflected in the test plan. The test plan covers the detailed planning regarding the three levels of tests for TCMS: software component test, function test and system test. The test plan also mentions the PASS/FAIL criteria for the

[1] https://www.peters-research.com/index.php/elevate

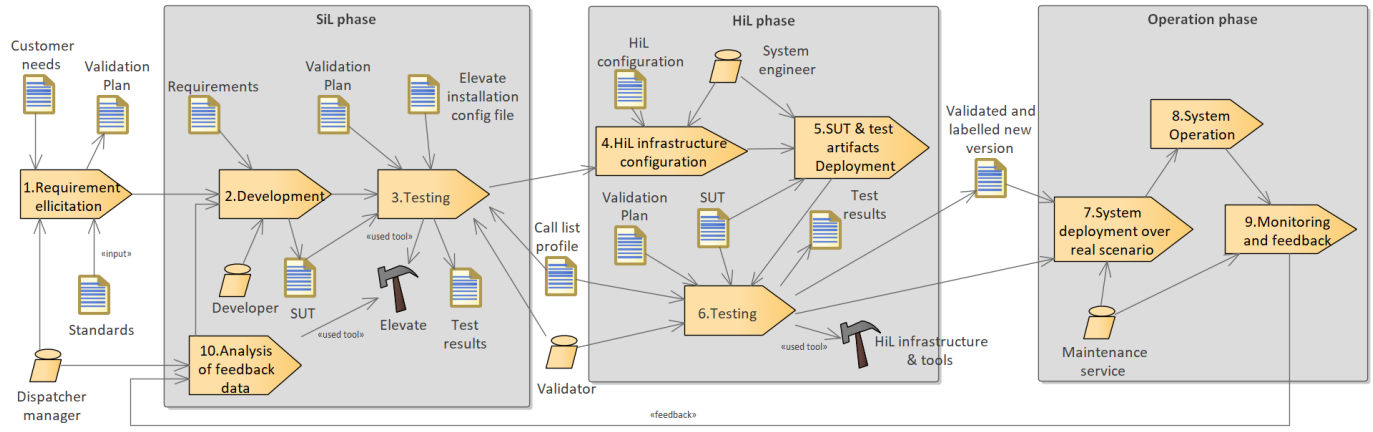


Fig. 1. Current software development process at Orona

three levels of tests. The plan touches upon the deployment in terms of environmental/infrastructure needs. The three levels of tests are performed in MiL, SiL and HiL simulators. Component and functional tests are typically performed in the domain-specific MiL/SiL simulator, while system tests are typically performed in the HiL simulator. Fig. 2 shows the development process at TCMS with corresponding simulation levels. The execution of test plan at MiL, SiL and HiL follows a somewhat similar sequence: 1) prepare software component/function/system test infrastructure; 2) develop software components/features; 3) develop component/function/system test cases; 4) build and deploy on test bench; 5) execute tests; 6) record defects (if any); 7) generate test report; 8) release software when no critical defects remaining. This is shown in Fig. 3. Many activities in this process require manual intervention, such as setting up the test environment and activities around testing at different simulation levels.

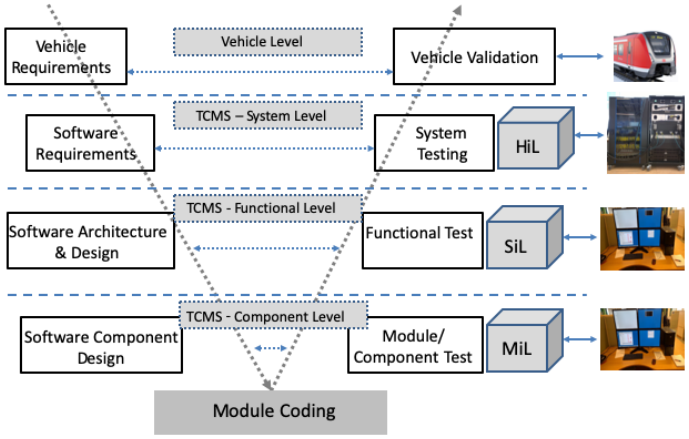


Fig. 2. Development process for TCMS at BT.

In testing TCMS, the engineering processes of software development (including requirement engineering and testing) are performed according to safety standards and regulations [6]. Specification-based testing is mandated by the EN 50128 standard to be used to design test cases. Each test case should contribute to the demonstration that a specified requirement has indeed been covered and satisfied. Executing test cases on TCMS is supported by a test framework that includes the comparison between the expected output with the actual outcome. Testing at the functional level is done against TCMS design requirements. The created test suites are based on functional specifications expressed in a natural language. Stimulation and responses are checked at the interfaces at the functional level. The test cases are composed by test steps, which define a stimulation and the expected output. Test case design is done in a scripting language and in a test management tool. Alternatively, if the test management tool is not available, the test case design can be documented with comments in the test scripts or in separate word documents.

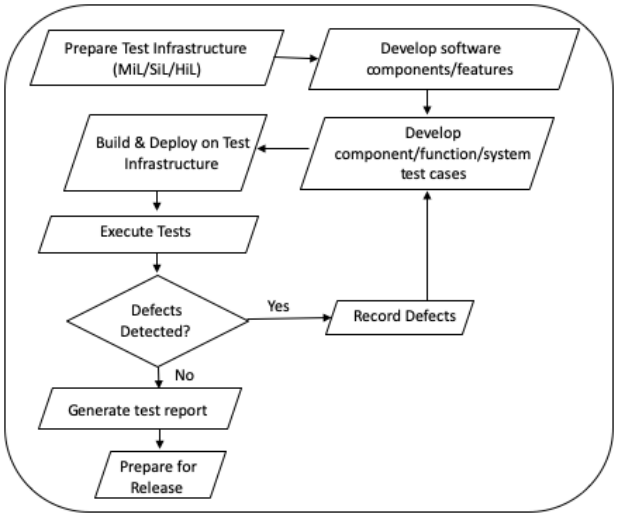


Fig. 3. Execution of test plan at MiL/SiL/HiL levels at BT.

C. Expected benefits

Design-Operation Continuum methods could automate several tasks of a software release. The goals of this approach

can be summarized in the following points: (1) Efficiently manage product variants and configurations throughout the testing process, reducing the number of errors and the time to configure a validation context at the MiL, SiL and HiL stages; (2) Reuse requirements at different levels of abstractions for validating system designs, ensuring that they are enforced consistently; (3) Automate the deployment of new software releases to the SiL, HiL and Operation, considerably reducing the costs; (4) Automate the execution of test cases at all phases; (5) Automate the validation of software releases by using streamlined test oracles that can be re-used across the whole lifecycle (i.e., MiL, SiL, HiL and Operation); (6) Automatically collect data during operation, which will enable reproducing real-life scenarios at design-time (i.e., MiL, SiL and HiL); (7) Enable the generation of test cases by using information collected from the whole lifecycle.

Achieving these objectives would result in an improvement of the software development practice and a reduction of the overall cost of releasing a new software version. The automated management of requirements, variants and configurations across all levels of abstraction and lifecycle stages would not only reduce the cost of the process, but also reduce the number of errors. Furthermore, the software quality would be improved by using real data from operation to identify realistic situations at design-time and to detect potential bottlenecks by monitoring the quality of service across the different software releases. This methodology would also allow the detection of defects earlier in the development lifecycle, further reducing the cost of testing. In addition, the quality would also benefit from increasing the likelihood and frequency of detecting bugs both at design-time and at operation-time by using streamlined test oracles at all the levels.

IV. Taxonomy for Design-Operation Continuum Methods of CPSs

Taxonomies can be organized following one of the two main classification approaches: enumerative or faceted [27]. *Enumerative approaches* utilize a fixed set of predefined classes, which results in simple and easy to apply classification schemes, but may not be appropriate for unexplored domains where the knowledge base is still unstable or incomplete. *Faceted approaches*, on the other hand, allow the classification of entities based on multiple perspectives, which provides a flexibility that is useful when developing taxonomies for immature domains [32]. In this work, we followed a faceted classification structure, since the Design-Operation Continuum, particularly when applied to CPSs, is an emerging domain that has not been fully defined yet. The final taxonomy contains four facets which are orthogonal to each other, three of which contain a single level of categories and the last of which contains several levels of sub-categories. The full taxonomy is shown in Figure 4.

A. Lifecycle stage

This facet represents the X-in-the-loop system execution level, which is an aspect specific to CPSs development processes. Requirements may be applicable to one or more of these classes. In this work, we define only the four classes which we identified as relevant for both of the analyzed case studies. For CPSs, although we believe that this classification is general, there could be cases where other levels might need to be defined. For instance, to the knowledge of the authors, there are companies where the MiL phase is split into several sub-phases with different fidelity levels. A few years ago, there was a step between SiL and HiL named Processor-in-the-Loop, which had the goal of detecting potential inconsistencies that the compiler could introduce [29]. However, to the best of our knowledge, this step is not commonly used any longer. Furthermore, as this phase was not used in the industrial case studies used to build our taxonomy, we did not include it.

At the **MiL** stage, the CPS, including the software, hardware and environment, is executed as a model by a model execution software. This setup allows the early and easy detection of failures in a controlled environment, but there are several types of errors that cannot be observed at this level, e.g., communication errors.

At the **SiL** stage, the CPS software is run on a simulated environment. The use of the real software allows the detection of several errors that could not be detected in MiL, such as those related with arithmetic precision. Nevertheless, not all the software errors can be observed yet, since the processor where the SiL runs is often different from the processor in the real hardware.

At the **HiL** stage, the CPS software is deployed on the real hardware, but within a controlled environment, such as a test bench. Since physical hardware is involved, the CPS execution must be real-time, which makes it much more costly than the previous stages. This execution level allows the detection of many new classes of errors, such as timing or hardware interaction issues, which were not observable in previous stages.

The **Operation** is the stage where systems are deployed in real environments, possibly in production. We distinguish this stage from HiL because usually intrusive testing can no longer be performed at this level, since the CPS is already running in real scenarios. Nevertheless, some non-intrusive validation techniques such as Runtime Verification can still be performed.

B. Scope

We distinguish three different scope classes depending on the applicability of the requirement. The significance of this facet is that it enables the reuse of the requirements throughout their applicable scope. Depending on the product strategy of a company, the categorization we provide can be refined or extended beyond the three classes we define. For example, we could have a family of products that share some common requirements, in which case a more fine-grained classification, such as a feature-level scope, would be useful.

Organization refers to the requirements that will be reused for the Design-Operation Continuum of all the products of the organization. For example, this category may include

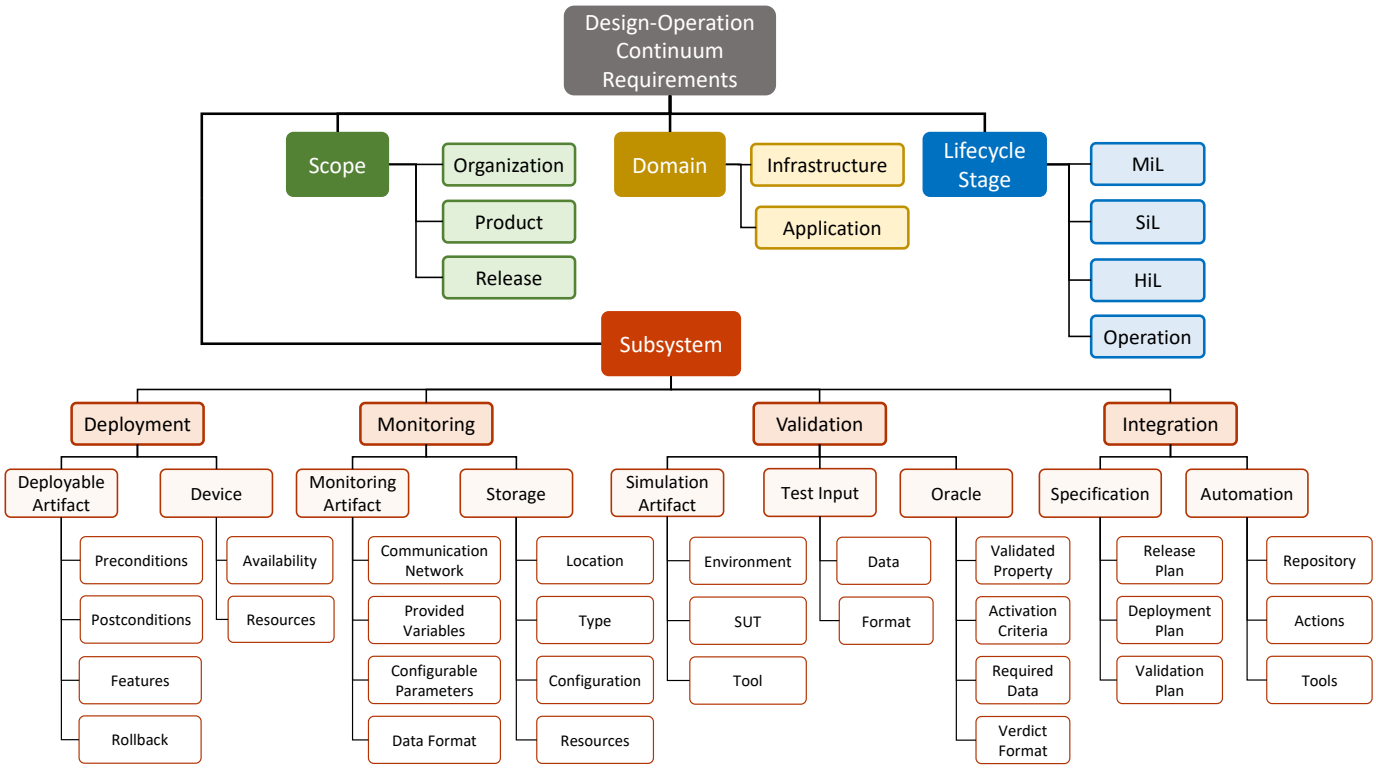


Fig. 4. Taxonomy of Design-Operation Continuum Requirements for CPSs

requirements such as the deployment subsystem being able to copy files to a target device running an SSH server.

Product refers to the requirements that are specific of a particular product, which can therefore be reused across all of its releases. For instance, this would include being able to automatically launch the simulators used for the test execution of an elevator dispatcher.

Release refers to the requirements specific to a particular software release of a product, which we differentiate from requirements applicable to all its releases. For instance, the verification of an optional feature of a product belongs to this category, as it is only applicable to some specific releases.

C. Domain

This facet categorizes the requirement by the domain in which it belongs, which we divide into two sub-categories. This categorization facilitates the assignment of the requirements within the organization between two different roles, usually IT department and development team.

- **Infrastructure**. On the one hand, we identify the requirements related with the Design-Operation Continuum infrastructure itself, which address concerns such as the monitoring of the infrastructure elements (e.g., deployment progress, status of the deployed components, etc.). Within the organization, the roles responsible for these requirements may not be directly related with the development of CPS products, since they only concern the development infrastructure.
- **Application**. On the other hand, we consider the requirements related with the particular applications being developed (i.e., the CPS), such as the monitoring of the application itself (e.g., tracking the status of the elevators based on CAN messages, etc.). These requirements must be managed by the organizational roles working directly on the CPS products.

D. Subsystem

This facet classifies a requirement by the Design-Operation Continuum subsystem for which it is relevant. Our taxonomy considers the subsystems of Deployment, Monitoring, Validation and Integration.

1) Deployment: Automating deployment means providing the infrastructure that allows automated CI server to connect to the designated production/validation machine and upload executable and configuration files [21]. The continuous deployment subsystem allows the automatic deployment of a new software release in the virtual infrastructure for validation purposes. Afterwards, the new release is deployed in the real CPS in operation. In this subsystem, requirements that are necessary to deploy artifacts at the MiL/SiL/HiL/Operation of the system are specified. It is important to mention that in this category, the requirements for the Operation stage are the most demanding ones, since aspects such as heterogeneous platforms or the status of the CPS before the deployment need to be considered. Examples of requirements obtained by our industrial case studies in this category include *"The deployment service shall provide support for ARMV7 boards"*,

"The deployment service shall provide support for Linux and Windows", "The dispatcher down time during deployment shall be less than 15 sec" or "The system shall allow the deployment of artifacts by defining the allocation or by defining the memory requirements". Nowadays, releasing and deploying new software versions is a time-consuming and error-prone activity. Requirements in this category facilitate the automation of the continuous deployment for new releases. Two subcategories have been defined: Devices and Deployable Artifacts. Note that this category is closely related to the Continuous Integration category.

- *Devices*. **Description.** A CPS is composed by heterogeneous platforms. Automation of the deployment process in CPSs is highly complex due to the number of heterogeneous platforms, models and interfaces necessary to deploy software releases. The goal of this subcategory is to collect requirements related to the variety and heterogeneity of hardware, software and communications for which the deployment subsystem must provide support. To gather requirements, an analysis of the platforms of the CPSs shall be performed. These type of requirements will have influence on the deployment architecture that must be designed to provide support for all the devices in which an automatic deploy will be performed. This category also has impact on the techniques and methods used for the deployment, e.g., container based deployment mechanisms that are valid for Linux based devices are not for embedded bare metal devices. **Subcategories.** There are different aspects to be specified: (1) *Resources* of the devices. Hardware, software (e.g., installed OS) and communication networks (e.g., CAN, Ethernet) that the deployment subsystem is going to deal with. (2) *Availability* of the device during deployment (e.g., maximum downtime of the device to perform the deploy).
- *Deployable Artifacts*. **Description.** A CPS is composed of different software components distributed in heterogeneous devices. Deployable artifacts are "soft" components which are part of the CPS, such as software of new releases and configuration files. When using Design-Operation Continuum methods, test oracles, monitors, etc. can also be considered deployable artifacts. This subcategory includes the specification of the features of the artifacts to be deployed. To gather requirements, an analysis of the software elements of the CPS shall be performed. Requirements in this category define the deployment rules, and are useful to ensure the pre and post deployment conditions and to design the rollback mechanisms. **Subcategories.** There are different aspects to be specified: (1) *Deployment conditions*: Pre-conditions specify criteria to be met before starting the deployment, e.g., *"the Elevator shall be out-of-service"*. Post-conditions are verified after the deployment is completed, e.g., *"the device reboots correctly"* (2) *Features of the artifacts*: hardware requirements, e.g., minimum CPU or RAM requirements to execute the artefact, software requirements, e.g., supported OS, communication interface requirements, e.g., to be deployed in a device with access to CAN or/and the allocation of the artifact, e.g., in which device shall be deployed, (3) *Rollback policy* in case of deployment failure, e.g., *"The system shall support the remote rollback to a previous version"*.

2) Monitoring: Continuous monitoring is a key process in Design-Operation Continuum. The goal of this process is to extract data from a system so that it can be analyzed [17]. Monitoring in the deployment ensures that certain conditions are met before and after deploying. In the validation process, it provides data to the oracles so that they can provide a verdict. Besides, it can also be useful to observe and record the status of the infrastructure/application and later reproduce real scenarios in simulation. Examples of requirements in this category include *"Monitoring data from MiL/SiL/HiL test executions shall be available through logs", "Monitors shall provide connectors for CAN and Ethernet", "Monitoring data for the last day shall be persisted for further analysis"*.

This category facilitates gathering monitoring requirements at different lifecycle stages and levels of a CPS. Two subcategories have been defined:

- *Monitoring artifacts*. **Description.** Continuous monitoring can be done (1) at the infrastructure level, e.g., to control the CPU or memory usage or (2) at the application level, to monitor, for instance, the position and speed of an elevator. The goal of this subcategory is to collect monitoring needs of both the infrastructure and the application. To gather requirements, an analysis of the application data lifecycle and the infrastructure features (e.g., CPU usage) shall be performed. Requirements in this subcategory have impact on the design of the monitoring infrastructure. **Subcategories.** There are different aspects to be specified: (1) *Communication Network*, the source from which data must be collected, e.g., *"the monitor must gather the data from the CAN bus"*, (2) *Data fields* that will be provided, e.g., *"the monitor must provide the elevator positions"*, (3) *Format* in which the data will be provided by the monitor, e.g., *"the monitor will provide the current elevator position periodically via MQTT"*, and (4) *Configurable parameters* for the monitor,e.g., *"the update period for the elevator positions may be configured with a value between 50 and 500 milliseconds"*.
- *Storage*. **Description.** Storage of the monitored data is essential to analyze and reproduce scenarios in simulation. The storage strategy may be different depending on the data being monitored. Some data could be more critical and other may need more memory resources. These requirements might include, for example, dumping data on a local file, storing it on the edge of the network, or sending it to a cloud database. The goal of Storage subcategory is to describe how the data shall be stored in order to be accessible from other services. To gather requirements, an analysis of the application data usage shall

be performed. Requirements in this subcategory have impact on the design of the storage strategy for the data that is being monitored. **Subcategories.** There are different aspects to be specified: (1) *Location* describes where the data is to be persisted, e.g., a shared folder on a NAS or a database endpoint, (2) *Type* relates to the database format, either a relational database, an object oriented database or even text file based, (3) *Configuration* includes attributes such as duration of the saved data, backup replicas or even availability aspects), (4) *Resources* specifies the type of device used for persistence, as well as the disk space size.

3) Validation: Testing, verification and validation activities are important in any kind of domain when developing software. In the case of CPSs, this is particularly important because most of the functionality of these systems is driven by software. Furthermore, this functionality is often safety or mission-critical, and a failure could lead to severe consequences. Both industrial case studies used to build the taxonomy rely on simulation-based testing for the validation of software. This technique allows raising the level of abstraction of complex CPSs in which testing is performed [5]. It allows (1) executing larger test suites and (2) building test oracles that can automate verification and validation tasks [5]. Furthermore, simulation-based testing allows modelling the environment in which the CPS operates (e.g., in the case of the elevators, the interaction of the elevators with passengers). Test, verification and validation in Design-Operation Continuum methods for CPSs needs to be practised from MiL phases through the Operation. This is because failures that could not be observed in previous stages can be identified in Operation. To this end, oracles need to be re-used across all these test levels to allow full automation. Examples of requirements obtained by focusing on the industrial needs of the case studies in this category include *"The SUT shall be the relevant version of the project-specific TCMS software", "The input to the test cases at the functional level shall be the stimuli triggering the execution of a defined functionality", "The oracles shall be activated by a test input or by identifying a precondition in operation"*. Note that the elicited requirements in this category shall provide the validation to be continuous and as automated as possible. To this end, three sub-categories were identified:

- *Simulation Artifact*. **Description.** This category concerns the artifacts that are necessary in order to enable simulation-based testing, which we divide in three main categories: **Subcategories.** (1) *Environment* refers to the conditions under which the system runs, which are usually expressed in the form of simulator parameters (e.g., the number of floors in the building); (2) the *SUT* is the component of the system that is being tested, which must usually comply with certain interfaces in order to be usable for simulation-based testing; and (3) the *Tool* is the simulator used to execute the SUT (e.g., Simulink). An example of a simulation artifact requirement for one of the industrial case studies is *"Test cases shall be executed by using the Elevate simulator"*.
- *Test Input*. **Description.** In order to drive the execution of the selected test cases, test inputs must be injected into the SUTs before or during their execution. We divide the requirements for these test inputs into two main categories. **Subcategories.** (1) The input data itself, which is determined by the test cases that need to be executed (e.g., must test having multiple passenger calls at the same time), and (2) The format that is used to define the test inputs (e.g., test inputs must be provided in a XML file which follows a specific structure).
- *Oracle*. **Description.** Test oracles are components in charge of emmitting a verdict (e.g., PASS/FAIL) based on the conformance of the system towards a specified property (e.g., for the Orona's dispatching system, the daily average waiting time per passenger shall be less than 30 seconds). Note that although monitoring the system is required for the validation, we classify monitoring as a separate subsystem, since monitoring is often performed beyond the context of system validation. The purpose of the oracles is to determine whether the observed behaviour of the system is correct or incorrect, which is usually done by verifying properties specified by a domain expert. An example of an elicited requirement for a test oracle is "test oracles shall be re-used across all levels (i.e., MiL, SiL, HiL and Operation)", or "test oracles shall be capable of validating 100% of functional requirements". **Subcategories.** Four sub-categories were identified based on the industrial case studies. (1) Validated properties are system's requirements themselves (e.g., $AWT < 30sec.$); (2) activation criteria are preconditions that trigger a test oracle to validate a specific property; (3) required data refers to the monitoring data needed by the oracle in order this to be able to check certain property; (4) the verdict format refers to the semantics provided by the verdict (e.g.,a quantitative value (e.g., a quantitative value from 0 to 1, with 1 meaning full compliance and the value becoming closer to 0 as the degree of compliance decreases).

4) Integration: Continuous Integration encompasses the subsystems to automate the pipeline from the development environment to the continuous deployment, monitoring, and validation subsystems. Automation is achieved by chaining different tasks together. The process involves software repositories, usage of adequate build tools, automated testing environments and testing tools, and deployment to operation. Examples of requirements in this category include *"The deployment specification shall provide support to link an artifact to a device", "A validation specification shall allow to specify test cases at SiL and HiL level", "The source code shall be available from outside the company"*. This category facilitates gathering requirements related with the integration of all of the subsystems (deployment, validation and monitoring) into an automated pipeline. Two sub-categories have been defined:

- *Specification*. **Description.** A pipeline is a sequence of

actions that have to be performed from the initial build of the project to the deployment of the real system. For this, different aspects must be specified, such a as the validations to be performed. In this category, requirements related to the specification of the sequence to be automated for the CI/CD scenario are provided. These requirements will be used to select or develop the CI/CD tool and the languages to specify the pipeline. **Subcategories.** There are different aspects to be specified: (1) *Release plan*: requirements for the language to specify the configuration and build process of the software. (2) *Deployment plan*: requirements for a language to specify all the steps related to the deployment. (3) *Validation plan*. Requirements for the language to specify the validation of the CPS, which is a critical step in the continuous integration process. This plan will describe the configuration, coordination and management of all the verification artifacts. For our case studies, for instance, we identified validation plan requirements such as being able to execute multiple instances of a system concurrently in order to compare their behaviour. All of these plans could be integrated into a single CI/CD plan. However, we have decided to classify these specification languages separately because different roles might be involved in the requirements elicitation for each of them.

- *Automation*. **Description.** Implementation of continuous integration or continuous deployment mechanisms depends on a series of tools that facilitate functions necessary to achieve fully automated operation. In this category, requirements that should be considered for automating the pipeline are defined.These requirements will be used to select the CI/CD tool and define the pipeline. **Subcategories.** There are different aspects to be specified: (1) *Repositories*: requirements for the artifact storage, which may be, for instance, a Git repository hosted on the cloud. The deployment subsystem will pull the artifacts from this repository when a deployment is executed. Availability, security, storage capacity, etc are defined in this category. (2) *Actions*: Actions that must be executed in the CI/CD pipeline, such as automatically initiating a deployment plan when a new commit is pushed to the master branch of the repository. (3) *Tools*: there might be requirements for using specific tools for some tasks of the pipeline. For instance, and organization might decide that the continuous integration process will be automated with Jenkins.

V. Requirements Elicitation Process

The requirements elicitation process we followed is iterative and and is summarized in Figure 5. In a first phase, requirements are elicited by considering the four categories. The initially elicited requirements should be general to any kind of CPS (or at least, any CPS that includes the categories that we identified in the taxonomy). Every requirement belongs to at least one of the sub-categories of each of the four facets we extracted in the taxonomy (i.e., Subsystem, Scope, Domain and Lifecycle). It is important to note that one requirement might affect more than one of the sub-levels of each category (e.g., both MiL and SiL).

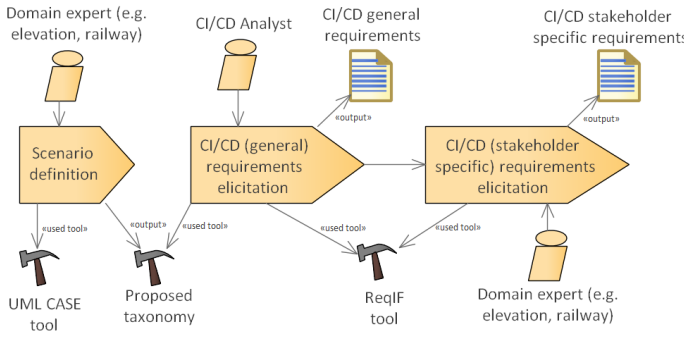


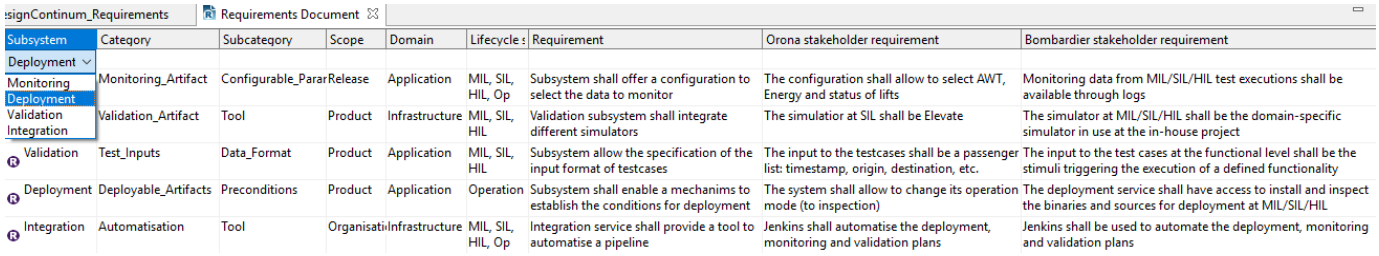
Fig. 5. Requirements elicitation process overview

In a second phase, for each of the elicited requirements, the industrial companies (i.e., Orona and Bombardier) instantiated the proposed requirement to their domain and application (e.g., in the Orona's use-case, there are some specific requirements for adapting the Design-Operation Continuum methods for the dispatcher, such as the simulator used for SiL). The tool used to document requirements was ReqIf studio[2], because the researchers involved had previous experience with this tool. A template with the facets of the taxonomy has been developed in order to classify each of the requirements, which includes (1) the general requirement, (2) how the requirement is instantiated for the case of the Dispatching algorithm in Orona, and (3) how the requirement is instantiated for the TCMS of BT. A screenshot of the developed template along with five elicited requirements is shown in Figure 6.

VI. Related Work

White and Edwards [33] proposed a taxonomy (RE-Views) to classify system views, subviews and their interdependencies. The requirements are classified into operational environment, system capabilities, system constraints, development requirements, verification & validation requirements and specification of system growth and change. The authors also mention a classification of requirements specification approaches, ranging from informal (natural language) to formal (mathematical) approaches. A more thorough classification of requirements specification techniques was given by Roman [26] in terms of formal foundation, scope, level of formality, degree of specialization, specialization area and development method. More recently, Hasan et al. [11] provide a classification of specification approaches for non-functional requirements. Hughes et al. [12] provide a two dimensional taxonomy for requirements analysis; one dimension corresponds to the set of viewpoints of different stakeholders (*concerns*) and the second dimension (*frames*) represents the views of technical specialists. Examples of a concern and a frame are functional requirements and behavioral model respectively. Jarke et al. [14] describe an ontology of requirements engineering of an information system by dividing

[2]https://reqif.academy/software/reqif-studio/



signContinum_Requirements | Requirements Document

Subsystem	Categoory	Subcategory	Scope	Domain	Lifecycle s	Requirement	Orona stakeholder requirement	Bombardier stakeholder requirement
Deployment								
Monitoring	Monitoring_Artifact	Configurable_Parar	Release	Application	MIL, SIL, HIL, Op	Subsystem shall offer a configuration to select the data to monitor	The configuration shall allow to select AWT, Energy and status of lifts	Monitoring data from MIL/SIL/HIL test executions shall be available through logs
Deployment	Validation_Artifact	Tool	Product	Infrastructure	MIL, SIL, HIL	Validation subsystem shall integrate different simulators	The simulatior at SIL shall be Elevate	The simulator at MIL/SIL/HIL shall be the domain-specific simulator in use at the in-house project
Validation	Test_Inputs	Data_Format	Product	Application	MIL, SIL, HIL	Subsystem allow the specification of the input format of testcases	The input to the testcases shall be a passenger list: timestamp, origin, destination, etc.	The input to the test cases at the functional level shall be the stimuli triggering the execution of a defined functionality
Validation								
Deployment	Deployable_Artifacts	Preconditions	Product	Application	Operation	Subsystem shall enable a mechanims to establish the conditions for deployment	The system shall allow to change its operation mode (to inspection)	The deployment service shall have access to install and inspect the binaries and sources for deployment at MIL/SIL/HIL
Integration	Automatisation	Tool	Organisati	Infrastructure	MIL, SIL, HIL, Op	Integration service shall provide a tool to automatise a pipeline	Jenkins shall automatise the deployment, monitoring and validation plans	Jenkins shall be used to automate the deployment, monitoring and validation plans

Fig. 6. Screenshot of a set of elicited requirements using our taxonomy

it into three 'worlds': *subject world* to represent properties such as timeliness, accuracy; *usage world* to represent user interface and functions; *development world* to represent development time, cost and consistency with standardized procedures. Nuseibeh et al. [22] describe a viewpoint interaction model to represent heterogeneity in requirements of software systems. A viewpoint is composed of five *slots*: style (to show representation knowledge), work plan (to show development process knowledge), domain (to show area of system under development), specification (to show system description) and work record (to show development history).

Several different domain-specific requirement taxonomies are also found in literature, e.g., for: safety requirements [7], security requirements [8], trust-related requirements [30], mobility-related requirements [10], usability requirements [3] and web-based enterprise systems [9]. Recently, automatic requirements categorization techniques have also been proposed. Knauss et al. [16] present a tool-supported approach based on Bayesian classifiers to identify security-relevant requirements. Ott [23] uses a similar approach to automatically classify and extract requirements with related information using text classification algorithms.

We were also able to find some fragmented evidence on requirements elicitation approaches for cyber-physical systems. Reza et al. [25] generated a set of quality attribute scenarios using pre-defined templates to document key non-functional requirements of a small spacecraft (CubeSat). The templates had the following fields: source, stimulus, environment, artifact, response, and response measure. Wiesner et al. [34] present a gamified approach for eliciting stakeholder requirements for a cabin video surveillance system of an aircraft. Though lacking in details, the approach works with storytelling and mutual agreement on requirements from different stakeholders. Loucopoulus et al. [19] report on the e-CORE (early Capability Oriented Requirements Engineering) approach that utilizes modeling for enterprise capabilities, goals, actors and information objects. This model-driven approach suggests different models such as capability model, goal model, actor-dependency model and information model.

Differently to all these studies, the taxonomy that we propose is related to Design-Operation Continuum Engineering methods in the context of CPSs. CPS is an emerging domain, and there is no clear path for adapting Design-Operation Continuum practices to it, because the challenges are inherently different from the domains discussed in the existing literature, such as web development. This taxonomy is used to elicit the requirements for Design-Operation Continuum Engineering methods from two different industrial domains developing CPSs. This sets an example for instantiating a taxonomy for any other organization developing CPSs.

VII. Conclusions, Limitations and Future Work

This paper proposes a taxonomy for Design-Operation Continuum methods applied to the context of CPSs, which has been systematically developed by following the guidelines proposed by Ralph [24]. To this end, two industrial case studies have been used, interviews have been performed with industrial experts, and we have been provided access to internal documents from both companies. The last phase of the taxonomy has been the validation with CPSs engineering experts that were not involved in the development of the taxonomy. By using this taxonomy, requirements can be elicited in two steps: Firstly, generic requirements are derived, which can be applied to any CPSs. Secondly, these generic requirements are instantiated for specific systems.

While the proposed taxonomy is applied to two different case studies, the applicability of it needs further examination, both for similar and different contexts. The taxonomy is based on fairly general categories, but we nevertheless foresee revisions, particularly to cater for the domain-specific requirements of other types of CPSs. In the future, we would like to perform a more comprehensive taxonomy considering (1) other sources from a systematic literature review and (2) other industrial CPSs. On the other hand, the taxonomy could also be extended to cover Design-Operation Continuum tasks beyond the ones identified for our two case studies, such as fault recovery automation and unforeseen situations detection.

Acknowledgment

This publication is part of a project that has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 871319. Jon Ayerdi, Aitor Arrieta and Goiuria Sagardui are part of the Software and Systems Engineering research group of Mondragon Unibertsitatea (IT1326-19), supported by the Department of Education, Universities and Research of the Basque Country.

This work has been partially supported by the Basque Government through the Elkartek program under the DIGITAL project (Grant agreement no. KK/2019-00095).

REFERENCES

[1] Rajeev Alur. *Principles of cyber-physical systems*. MIT Press, 2015.

[2] Sara Abbaspour Asadollah, Rafia Inam, and Hans Hansson. A survey on testing for cyber physical system. In *IFIP International Conference on Testing Software and Systems*, pages 194–207. Springer, 2015.

[3] H. Belani. Towards a usability requirements taxonomy for mobile aac services. In *Proceedings of the First International Workshop on Usability and Accessibility Focused Requirements Engineering*. IEEE Press, 2012.

[4] G. A. Bowen. Document analysis as a qualitative research method. *Qualitative research journal*, 9(2):27, 2009.

[5] Lionel Briand, Shiva Nejati, Mehrdad Sabetzadeh, and Domenico Bianculli. Testing the untestable: model testing of complex software-intensive systems. In *Proceedings of the 38th international conference on software engineering companion*, pages 789–792, 2016.

[6] CENELEC. 50128: Railway Application: Communications, Signaling and Processing Systems, Software For Railway Control and Protection Systems. In *Standard Official Document*. European Committee for Electrotechnical Standardization, 2001.

[7] D. Firesmith. A taxonomy of safety-related requirements. https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=29419, 2004. Online; accessed 27 Feb 2020.

[8] D. Firesmith. A taxonomy of security-related requirements. https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=30108, 2005. Online; accessed 27 Feb 2020.

[9] A. Ghazarian. Characterization of functional software requirements space: The law of requirements taxonomic growth. In *2012 20th IEEE International Requirements Engineering Conference (RE)*, 2012.

[10] S. Gopalakrishnan and G. Sindre. A revised taxonomy of mobility-related requirements. In *2009 International Conference on Ultra Modern Telecommunications Workshops*, 2009.

[11] M. M. Hasan, P. Loucopoulos, and M. Nikolaidou. Classification and qualitative analysis of non-functional requirements approaches. In I. Bider, K. Gaaloul, J. Krogstie, S. Nurcan, H. A. Proper, R. Schmidt, and P. Soffer, editors, *Enterprise, Business-Process and Information Systems Modeling*, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.

[12] K. J. Hughes, R. M. Rankin, and C. T. Sennett. Taxonomy for requirements analysis. In *Proceedings of IEEE International Conference on Requirements Engineering*, 1994.

[13] N. Humbatova, G. Jahangirova, G. Bavota, V. Riccio, A. Stocco, and P. Tonella. Taxonomy of real faults in deep learning systems. *International Conference on Software Engineering (ICSE)*, 2020.

[14] M. Jarke, J. Bubenko, C. Rolland, A. Sutcliffe, and Y. Vassilou. Theories underlying requirements engineering: an overview of nature at genesis. In *[1993] Proceedings of the IEEE International Symposium on Requirements Engineering*, 1993.

[15] Siddhartha Kumar Khaitan and James D McCalley. Design techniques and applications of cyberphysical systems: A survey. *IEEE Systems Journal*, 9(2):350–365, 2014.

[16] E. Knauss, S. Houmb, K. Schneider, S. Islam, and J. Jürjens. Supporting requirements engineers in recognising security issues. In D. Berry and X. Franch, editors, *Requirements Engineering: Foundation for Software Quality*, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[17] Richard Lai, S Mahmood, R Lai, and Y S Kim. Survey of component-based software development. *The Institution of Engineering and Technology*, 3(May 2007):58–64, 2014.

[18] Edward Ashford Lee and Sanjit A Seshia. *Introduction to embedded systems: A cyber-physical systems approach*. Mit Press, 2016.

[19] P. Loucopoulos, E. Kavakli, and N. Chechina. Requirements engineering for cyber physical production systems. In P. Giorgini and B. Weber, editors, *Advanced Information Systems Engineering*, pages 276–291, Cham, 2019. Springer International Publishing.

[20] Reza Matinnejad, Shiva Nejati, Lionel C Briand, and Thomas Bruckmann. Test generation and test prioritization for simulink models with dynamic behavior. *IEEE Transactions on Software Engineering*, 45(9):919–944, 2018.

[21] Phu H. Nguyen, Nicolas Ferry, Gencer Erdogan, Hui Song, Stéphane Lavirotte, Jean Yves Tigli, and Arnor Solberg. A systematic mapping study of deployment and orchestration approaches for iot. In *IoTBDS 2019 - Proceedings of the 4th International Conference on Internet of Things, Big Data and Security*, pages 69–82. SciTePress, 2019.

[22] B. Nuseibeh, J. Kramer, and A. Finkelstein. A framework for expressing the relationships between multiple views in requirements specification. *IEEE Transactions on Software Engineering*, 20(10):760–773, 1994.

[23] D. Ott. Automatic requirement categorization of large natural language specifications at mercedes-benz for review improvements. In *Proceedings of the 19th International Conference on Requirements Engineering: Foundation for Software Quality*, Berlin, Heidelberg, 2013. Springer-Verlag.

[24] P. Ralph. Toward methodological guidelines for process theories and taxonomies in software engineering. *IEEE Transactions on Software Engineering*, 45(7):712–735, 2018.

[25] H. Reza, C. Korvald, J. Straub, J. Hubber, N. Alexander, and A. Chawla. Toward requirements engineering of cyber-physical systems: Modeling cubesat. In *2016 IEEE Aerospace Conference*, 2016.

[26] G. C. Roman. A taxonomy of current issues in requirements engineering. *Computer*, 18(4):14–23, 1985.

[27] J. Rowley and R. Hartley. *Organizing knowledge: an introduction to managing access to information*. Routledge, 2017.

[28] C. B. Seaman. Qualitative methods in empirical studies of software engineering. *IEEE Transactions on software engineering*, 25(4):557–572, 1999.

[29] Hesham Shokry and Mike Hinchey. Model-based verification of embedded software. 2009.

[30] G. Sindre. Trust-related requirements: A taxonomy. In W. Wojtkowski, W. G. Wojtkowski, J. Zupancic, G. Magyar, and G. Knapp, editors, *Advances in Information Systems Development*, Boston, MA, 2007. Springer US.

[31] Martin Törngren and Paul T Grogan. How to deal with the complexity of future cyber-physical systems? *Designs*, 2(4):40, 2018.

[32] M. Usman, R. Britto, J. Börstler, and E. Mendes. Taxonomies in software engineering: A systematic mapping study and a revised taxonomy development method. *Information and Software Technology*, 85:43–59, 2017.

[33] S. White and M. Edwards. A requirements taxonomy for specifying complex systems. In *Proceedings of First IEEE International Conference on Engineering of Complex Computer Systems*, 1995.

[34] S. Wiesner, J. B. Hauge, F. Haase, and K-D. Thoben. Supporting the requirements elicitation process for cyber-physical product-service systems through a gamified approach. In I. Nääs, O. Vendrametto, J. Mendes R., R. F. Gonçalves, M. T. Silva, G. von Cieminski, and D. Kiritsis, editors, *Advances in Production Management Systems. Initiatives for a Sustainable World*, Cham, 2016. Springer International Publishing.