Adeptness

ADEPTNESS – Design-Operation Continuum Methods for Testing and Deployment under Unforeseen Conditions for Cyber-Physical Systems of Systems

EUROPEAN COMMISSION

Horizon 2020

H2020-ICT-01-2019

GA No. 871319

Deliverable No.	ADEPTNESS D3.1	
Deliverable Title	Report on the application of test oracles at the operational time of CPSoS	
Deliverable Date	2021-12-31	
Deliverable Type	Report	
Dissemination level	Public	
Written by	MGEP	2021-12-3
Checked by	MGEP SRL	2021-12-15
Approved by	Executive board	2021-12-20
Status	Final	2021-12-21

H2020-ICT-01-2019 – 871319 – ADEPTNESS: Design-Operation Continuum Methods for Testing and Deployment under Unforeseen Conditions for Cyber-Physical Systems of Systems

Acknowledgement

The author(s) would like to thank the partners involved with the project for their valuable comments on previous drafts and for performing the review.

Project partners

1 – MGEP – Mondragon Goi Eskola Politeknikoa – ES

2 – ORO – Orona S. Coop – ES

3 – UES – Ulma Embedded Solutions S. Coop – ES

4 – SRL – Simula Research Laboratory S. Coop – NO

5 – BT – Bombardier Transportation Sweden – SE

6 – IKL – Ikerlan S. Coop – ES

7 – EGM – Easy Global Market SAS – FR

8 – MDH – Maelardalens Hoegskola – SE

9 – TUW – Technische Universitaet Wien – AT

Disclaimer:

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 871319.

Document Information

Additional author(s) and contributing partners

Name	Organisation
Maialen Otaegi Aitor Arrieta Jon Ayerdi Ibai Roman	MGEP MGEP MGEP MGEP
Shaukat Ali Liping Hang	SRL SRL

Document Change Log

Name	Date	Comments
V0.1	2021-07-21	Initial draft
V1.0	2021-12-03	First version to be reviewed by Executive Board
V1.1	2021-12-15	Version including the changes proposed by the Executive Board

Exploitable results

Exploitable results	Organisation(s) that can exploit the result
Tool for generation of test oracles	MGEP

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. Purpose of the document

1.1. Document structure

This document presents the research and experiments done on the application of test oracles at the operational time of CPSoS.

Section 3 introduces some background on Test Oracles, as well as the state-of-the-art.

Section 4 is focused on Organic Oracles at operation. The section starts with a detailed description of the formalisation of Organic Test Oracles, to then explain how Uncertainty is handled within Organic Test Oracles, and finishes presenting the integration with Adeptness Microservice Template.

Section 5 introduces Pre-Specification Test Oracles. First, preliminary experimentation in this regard is introduced. Next, the approach used to fit Pre-specification Oracles in the Adeptness Context is described. And finally, the integration of this type of oracles with the Adeptness Microservice Template is explained.

Section 6 is dedicated to Metamorphic-based Test Oracles, where the Testing Interface and the Metamorphic Relations are described. Next, the experiment carried out is presented, starting with the setup, and concluding with some results.

The deliverable culminates with some conclusions on the experiments carried out regarding the application of test oracles at the operational time of CPSoS.

1.2. Deviations from the original Description in the Grant Agreement Annex 1 Part A

1.2.1. Description of work related to deliverable in GA Annex 1 – Part A

There are no deviations with respect to work of this deliverable.

1.2.2. Time deviations from original planning in GA Annex 1 – Part A

There are no deviations with respect to work of this deliverable.

1.2.3. Context deviations from the original plan in GA Annex 1 – Part A

There are no deviations from the Annex 1.

2. Introduction

Design-operation continuum methods require seamless integrations between the design-time approaches and operation-time approaches. These include the case of test oracles. Test oracles aim at determining whether the system is behaving as expected or not. Traditionally, test oracles have been applied for design-time testing, while its operation-time has been limited to simple run-time verification functions. However, determining whether a Cyber-Physical Systems (CPS) is behaving as expected is nontrivial. In some cases, such systems suffer from the test oracle problem (i.e., determining the test outcome is non-feasible). Furthermore, CPSs are inherent to different kinds of uncertainty (e.g., uncertainty in the environment, uncertainty of networks, etc), thus, the oracle needs to deal also with it. In this deliverable we analyse three different types of test oracles: (1) organic test oracles, (2) pre-specification test oracles and (3) metamorphic test oracles. We analyse how each of these can be applied on the design-operation continuum of CPSs, solving the aforementioned limitations.

3. Background and State of the art on test oracles

3.1. Background

A test oracle is the source in charge of determining whether a system complies with the specified properties or not. Four types of test oracles are typical for testing software systems: (1) Organic test oracles, (2) Pre-specification test oracles, (3) Internal test oracles and (4) Metamorphic test oracles. Figure 1 shows an example of an organic test oracle. These oracles take as input both, the input of the SUT as well as the output of the SUT. The oracle is structured with all the necessary sources to determine which the verdict of the inputs should be; Figure 2 High level structure of an organic oracle shows a high-level structure of these kind of oracles proposed for the context of Cyber-Physical Systems (CPSs). Typically, these oracles check specific system requirements, for which each requirement is composed by the pre-condition and the assertion. The pre-condition checks whether the system has been activated. If the requirement has been activated, then the assertion is computed (i.e., it is checked whether the system holds the specified requirement).

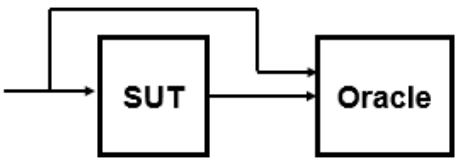

FIGURE 1 EXAMPLE OF AN ORGANIC TEST ORACLE

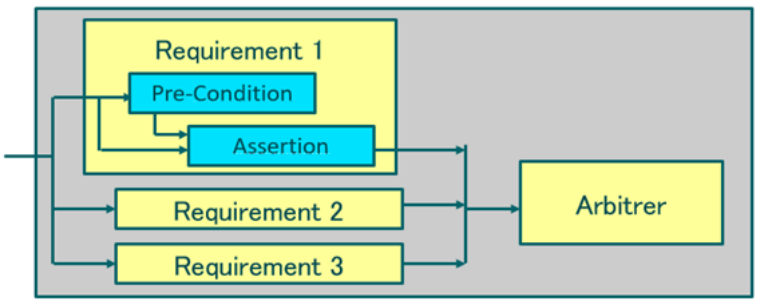

FIGURE 2 HIGH LEVEL STRUCTURE OF AN ORGANIC ORACLE

Other kinds of test oracles are those related to pre-specification test oracles. Figure 3 shows the high-level structure of these kinds of oracles. Unlike organic test oracles, these oracles get only as input the inputs of the SUT. As output, they provide the reference values that the SUT should have, which are compared against the SUT by the checker, returning a verdict (e.g., PASS, FAIL). There are different kinds of these test oracles. The most typical ones are related to the regression test oracles, for which the test oracle replicates a previous version of the SUT, which is considered as a "baseline". There might also be an "implementation approximation", for which expected results are obtained from a reference work. In re-engineering of systems, there might also be what is known as a "golden implementation", where existing applications can be used to generate expected results (e.g., this might happen when re-engineering a SUT, for which white-box access is not available). The oracle might even be manual, where the expected results are determined by hand.

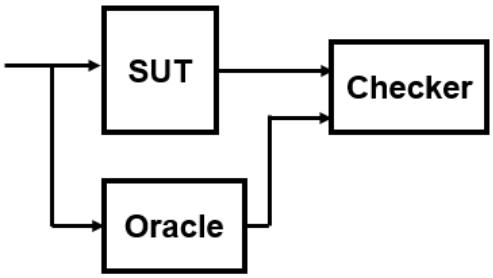

FIGURE 3HIGH LEVEL STRUCTURE OF PRE-SPECIFICATION TEST ORACLES

Internal oracles are those that are programmed inside the SUT. Typically, these oracles are developed by the developers, which insert assertions in the internal program to check the run-time behaviour of the SUT. A drawback of such oracles is that they are intrusive, therefore, they can have an impact on the running time of the implementation. These kind of test oracles are not analysed in this Deliverable.

Metamorphic testing~\cite{1998-chen-tr} is a technique aiming to alleviate the oracle problem (i.e., when it is not possible to determine which the outcome of the SUT should be). It is based on the intuition that often it is simpler to reason about relations between the inputs/outputs of multiple, related test executions, rather than the relations between inputs/outputs of each individual test execution [1]. Given two test cases defined by their corresponding inputs and outputs TC_1 = {I_1, O_1}, TC_2 = {I_2, O_2}, whenever a given input relation r_i holds between the two inputs, a corresponding output relation r_o is expected to hold between the outputs. Hence, the metamorphic oracle can be expressed as r_i(I_1, I_2) ➔ r_o(O_1,O_2).

The goal of Adeptness is to (1) investigate these kinds of test oracles in the context of CPSs and (2) adapt such test oracles for testing CPSs both at design-time as well as at operation-time. The former has different challenges which are specific to the context of CPSs, such as, the impossibility of determining which the test output should be (tackled by metamorphic test oracles), or the time-continuous behaviour of CPSs. The latter has other challenges, such as those related to interoperability of test oracles for being tested at different Design-Operation continuum test levels, as well as the issue of needing multiple test executions in the context of metamorphic or pre-specification test oracles or dealing with the uncertainty that the CPSoS are exposed to. In such cases, we leveraged the use of Artificial Intelligence (AI) algorithms to substitute the execution of the system, which, in the case of CPSoS are highly time-consuming.

3.2. Related work

In the context of CPSs and CPSoS, different test oracles have been proposed. Zander [2] used organic test oracles for embedded systems from the automotive domain. Kane et al., specified test oracles based on requirements [3]. Nevertheless, all of them assume that the tests are going to be executed at design-time. The type of systems they are testing have well stablished requirements since they come from the safety domain. However, most CPSs are untestable, which means that their test outcome is not easy to determine. Furthermore, if the test oracles need to be re-used at operation-time, these need to consider the inherent uncertainty at which CPSs are exposed to.

[1] Ayerdi, J., Terragni, V., Arrieta, A., Tonella, P., Sagardui, G., & Arratibel, M. (2021, August). Generating metamorphic relations for cyber-physical systems with genetic programming: an industrial case study. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (pp. 1264-1274).

[2] *Zander-Nowicka, J. (2009). Model-based testing of real-time embedded systems in the automotive domain.*

[3] Kane, A., Fuhrman, T., & Koopman, P. (2014, June). Monitor based oracles for cyber-physical system testing: Practical experience report. In *2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks* (pp. 148-155). IEEE.

To overcome the first limitation, i.e., the test outputs not being feasible to be obtained, several works have relied on metamorphic testing. Specially, this technique has been applied to test autonomous vehicles [4],[5]. This technique has also been applied in other contexts, such as autonomous unmanned aerial vehicles [6]. However, to the date there are no contexts related to real industrial CPSs where such technique has been applied on. Furthermore, this technique is solely designed for design-time testing, not being possible to later use it for operation-time testing.

Another option for solving the test oracle problem which is envisioned in Adeptness is the use of Machine-learning as a surrogate of regression test oracles. The use of machine-learning algorithms to alleviate the test oracle problem is not new, although it has receieved significantly less attention than other software testing activities (e.g., test selection).

A recent systematic survey performed by Durelly et al. identified a total of 10 studies where machine learning algorithms were used to construct oracles[7]. As a substitute of regression test oracles, machine-learning algorithms were used to predict the expected output of SUTs in two studies [8],[9]. One of them used artificial neural networks, while the other used regression trees and neural networks. Nevertheless, their approach is applied for unit testing, while in Adeptness we are focusing on system-level tests. Furthermore, their evaluation is performed in a toy example involving the triangle type problem, where determining the test outcome is trivial.

When testing CPSs, simulation-based testing is the main driving technology. In such contexts, recent studies tackle the test oracle problem. Menghi et al. proposed a test oracle generation tool for Simulink models, which was based on a Domain Specific Language [10]. Stocco et al. proposed a technique for testing autonomous vehicles whose control algorithm is based on deep neural networks [11]. Their oracle employs simulation-based testing and determines a confidence value for the system at each step of the execution. While this is a CPS, their approach is not generalizable to any kind of CPS but limited to the domain of autonomous driving.

In Adeptness we not only want to explore the test oracle problem for design-time testing, but also be able to re-use such oracles in a seamless way for operation-time testing. For such purpose, it is highly important to (1) deal with the inherent uncertainty at which CPSs are exposed to and (2) solve the issue of requiring multiple test executions by those test oracles based on that (i.e., pre-specification and metamorphic test oracles).

---

[4] *Zhi Q Zhou and Liqun Sun. Metamorphic testing of driverless cars. 2019.*

[5] *Yuchi Tian, Kexin Pei, Suman Jana, and Baishakhi Ray. Deeptest: Automated testing of deep-neural-network-driven autonomous cars. In Proceedings of the 40th international conference on software engineering, pages 303–314. ACM, 2018.*

[6] *Mikael Lindvall, Adam Porter, Gudjon Magnusson, and Christoph Schulze. Metamorphic model-based testing of autonomous systems. In 2017 IEEE/ACM 2nd International Workshop on Metamorphic Testing (MET), pages 35–41. IEEE, 2017*

[7] *V. H. Durelli, R. S. Durelli, S. S. Borges, A. T. Endo, M. M. Eler, D. R. Dias, and M. P. Guimaraes, "Machine learning applied to software testing: A systematic mapping study," IEEE Transactions on Reliability, vol. 68, no. 3, pp. 1189–1212, 2019.*

[8] *H. Jin, Y. Wang, N.-W. Chen, Z.-J. Gou, and S. Wang, "Artificial neural network for automatic test oracles generation," in 2008 International Conference on Computer Science and Software Engineering, vol. 2. IEEE, 2008, pp. 727–730.*

[9] *A. Singhal and A. Bansal, "Generation of test oracles using neural network and decision tree model," in 2014 5th International ConferenceConfluence The Next Generation Information Technology Summit (Confluence). IEEE, 2014, pp. 313–318.*

[10] *C. Menghi, S. Nejati, K. Gaaloul, and L. C. Briand, "Generating automated and online test oracles for simulink models with continuous and uncertain behaviors," in Proceedings of the ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/SIGSOFT FSE 2019, Tallinn, Estonia, August 26-30, 2019, 2019, pp. 27–38.*

[11] *A. Stocco, M. Weiss, M. Calzana, and P. Tonella, "Misbehaviour prediction for autonomous driving systems," in Proceedings of 42nd International Conference on Software Engineering, ser. ICSE '20. ACM, 2020, p. 12 pages.*

4. Organic oracles at operation

A large number of CPSoS properties can be validated and tested by employing organic oracles. A special effort was done to formalise such kind of test oracles that are reusable at different stages of software development. As a result, the designed test oracles are stage independent enough to be executed at Model-in-the-Loop (MiL), Software-in-the-Loop (SiL) and Hardware-in-the-Loop (HiL) levels, but also at operation.

This fact is particularly interesting as software development requires to be tightly connected with the operational stage as the software is being executed in a real environment. An important aspect to consider when a CPS is in operation is the uncertainty at which the CPS is exposed. To tackle this, the organic test oracles that we present provide means to handle uncertainty.

4.1. Formalisation of organic test oracles

As stated in the deliverable *4.1. DSL for continuous validation of CPSoS*, an industrial case study from the elevation domain was taken as a starting point to formalise the test oracles for the Adeptness infrastructure. Then, with the aim of developing a case-study agnostic test oracles, the results were generalised using open-source benchmarks.

The resulting model is illustrated in Figure 4. This model was presented in *4.1. DSL for continuous validation of CPSoS*, however, we consider that the work done is also part of the task we are presented in this deliverable.

The model is divided into two main parts, the *MonitoringFile* and the *CPS*. These are located at the top of the figure, inherited from the *Type* component, which, in turn, is at the same level as *PackageDeclaration* and *Imports* components. Each part could be developed in a separate file. The *CPS* section is intended for the description of the oracles, and the *MonitoringFile* section is intended for the declaration of the signals required by the oracles. The latter, also known as the monitoring plan, could be imported from an oracle definition file, and as can be seen in the figure, is referenced from the *CPS* component. Therefore, if a monitoring plan is defined in a separate file, it could be used in several oracles' definition files.

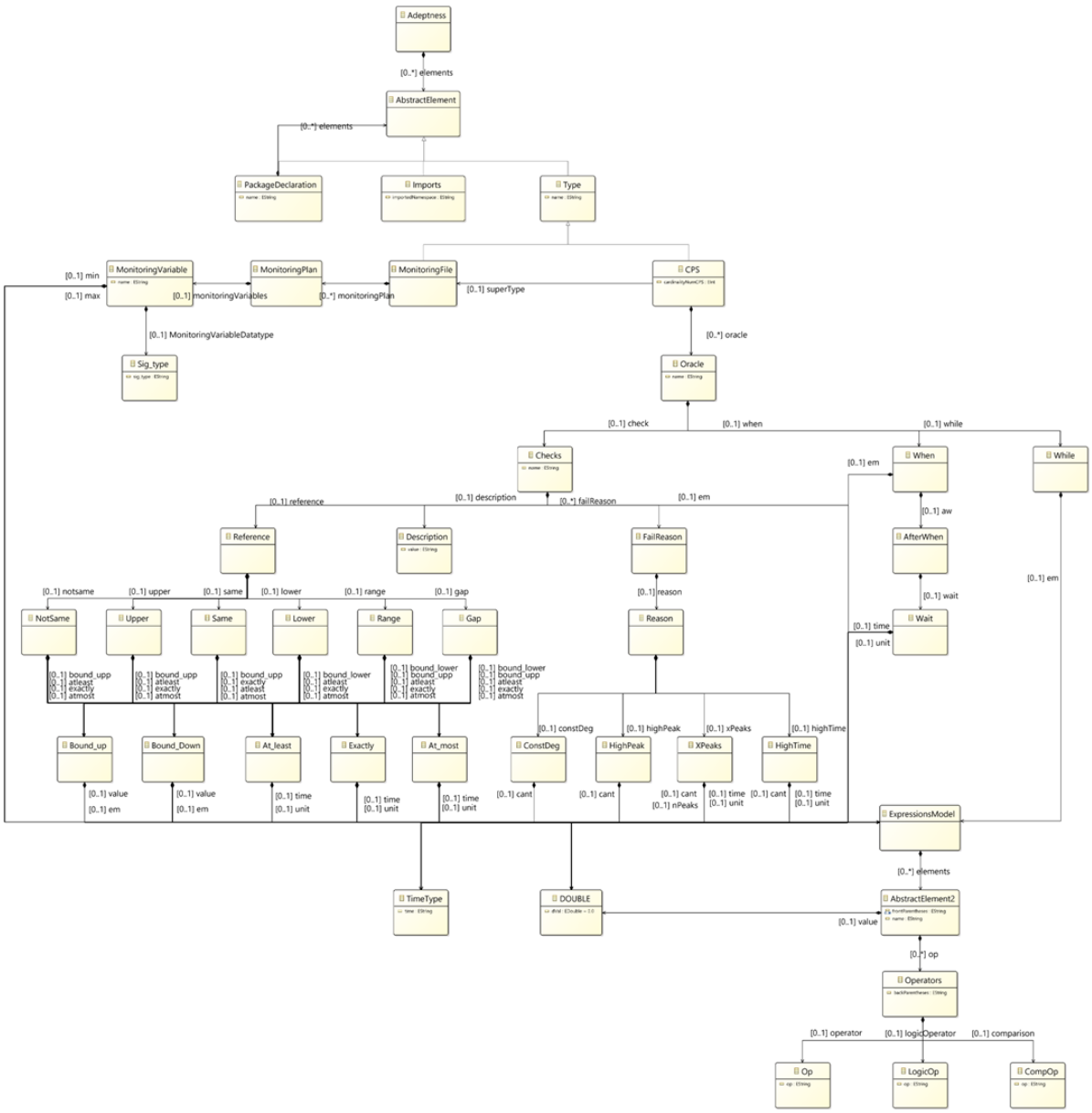

FIGURE 4 ORGANIC ORACLES MODEL

In the following sections we are going to explain in detail the organic oracle's model.

4.1.1. Monitoring plan

To define organic test oracles, it is necessary to identify and extract available CPS monitoring variables. As mentioned earlier, these monitors are defined in a separate file: the monitoring plan file. Within the file, first, the monitoring plan is given a name and then, for each of the monitors declared, a name, data type and maximum and minimum values are specified. The maximum and minimum values are only necessary if the specified data type is numeric (i.e., double and Boolean). The following figure gathers the section of the model that represents the monitoring plan file:

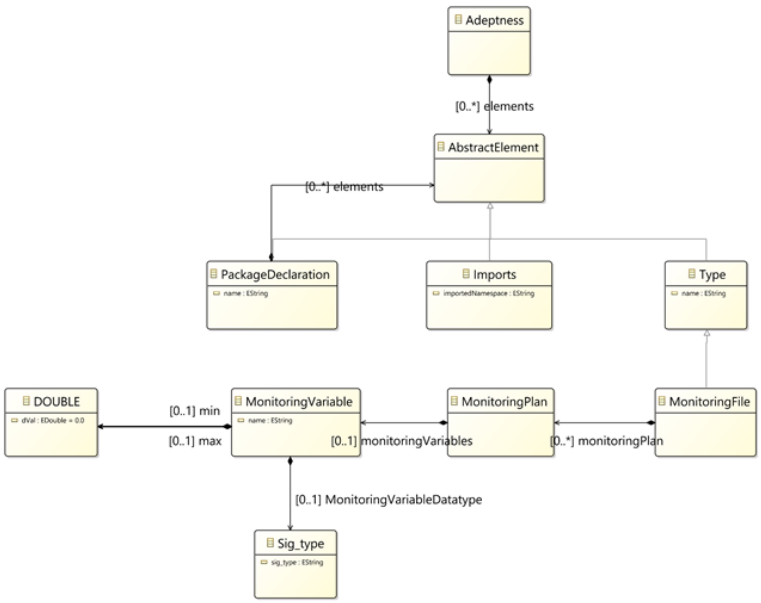

FIGURE 5 MONITORING PLAN MODEL

4.1.2. Oracle definition

After defining the monitoring plan, oracles for each CPS within a CPSoS are defined. Oracles are defined at CPS level instead of CPSoS because, despite the interaction of another CPS can be influenced, the monitoring data is associated and obtained from a single CPS. A CPS might have information of other CPSs (e.g., for Orona's use-case the traffic master always monitors the status of the doors to better estimate serving times). Each CPS implements a monitoring plan, which means that it has accessible all data defined in the monitoring plan (previous figure). Additionally, it is possible to include cardinality at this level, meaning that CPS that behave equally could reuse the same oracles, changing the monitoring plan. Below, the oracle's definition section of the model can be found. As can be seen, the *CPS* component inherits the name attribute from *Type*, includes a *MonitoringFile*, and may be composed of several *Oracle*s.

As for the *Oracle* definition, on the right side, there are the components for specifying a precondition, which could be *When* or *While* and, on the left side, there is the *Check* component. The latter gathers the assertion definition, which is a conjunction of the signal to be asserted (modelled by the component *ExpressionModel*) and an assertion pattern and a reference signal (modelled by the component *Reference*), a failure reason (modelled by the *FailReason* component) and, finally, a *Description*. In the following paragraphs further details of the preconditions, the assertion and the failure reason are going to be provided.

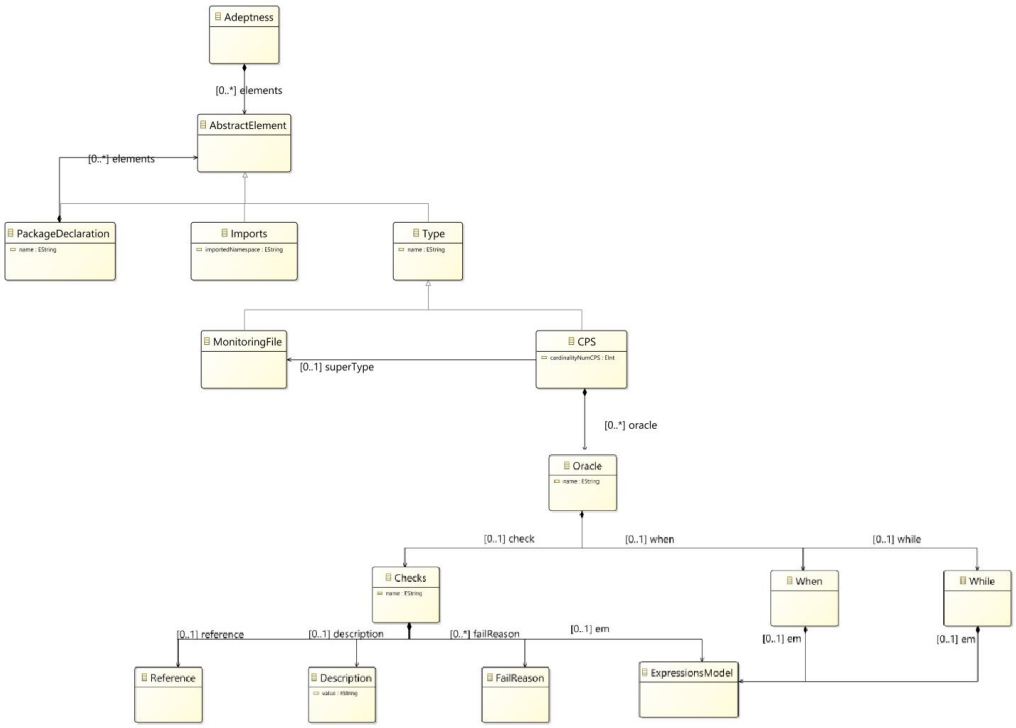

FIGURE 6 MODEL OF THE ORACLE DEFINITION RELATED SECTION

Further details in the organic test oracle's definition model are going to be provided in the following sections.

Precondition

Specifying a precondition allows defining test oracles that are not continuously assessing the status of a CPS and giving verdicts accordingly. A precondition aims to insert a previous step that must be fulfilled in order to evaluate the assertion. A precondition is represented by operations among signals and values, reflecting a status of a CPS, and should evaluate to true or false.

Within the analysed scenarios, in some cases, the need to wait for a certain period of time before assessing the status of a CPS was observed. With this in mind, the Adeptness organic test oracles could include waiting clauses that specify the period of time between the precondition is met and the assessment takes place.

Figure 7 gathers the section regarding the precondition of an organic test oracle. A precondition is optional, as there can be oracles that constantly assert data. If it is set, it is the starting point of the testing process. A precondition is a Boolean condition expressed through a *While* or a *When* component. As can be seen, both preconditions are composed by an *ExpressionsModel*, which in turn, enables the creation of an expression that can be evaluated to true or false. Although the model allows for different types of expressions, this constraint should be enforced afterwards. The difference between these two conditions is that the *While* precondition specifies system states, and therefore the assert must be performed during the same cycle(s) in which the assertion is checked. On the contrary, the *When* precondition specifies an event, and when this event is given, there is no need to remain asserting it while the assertion is being checked. Although the two preconditions act in a similar way, the difference becomes obvious when the assertion specifies a temporal condition (covered in the following paragraphs). Additionally, a *When* precondition allows for expressing an *After* temporal-logic expression through a *Wait* component and pauses the oracle for the specified time and then, the assertion is checked. The *Wait* component is composed by a time value and a time unit,

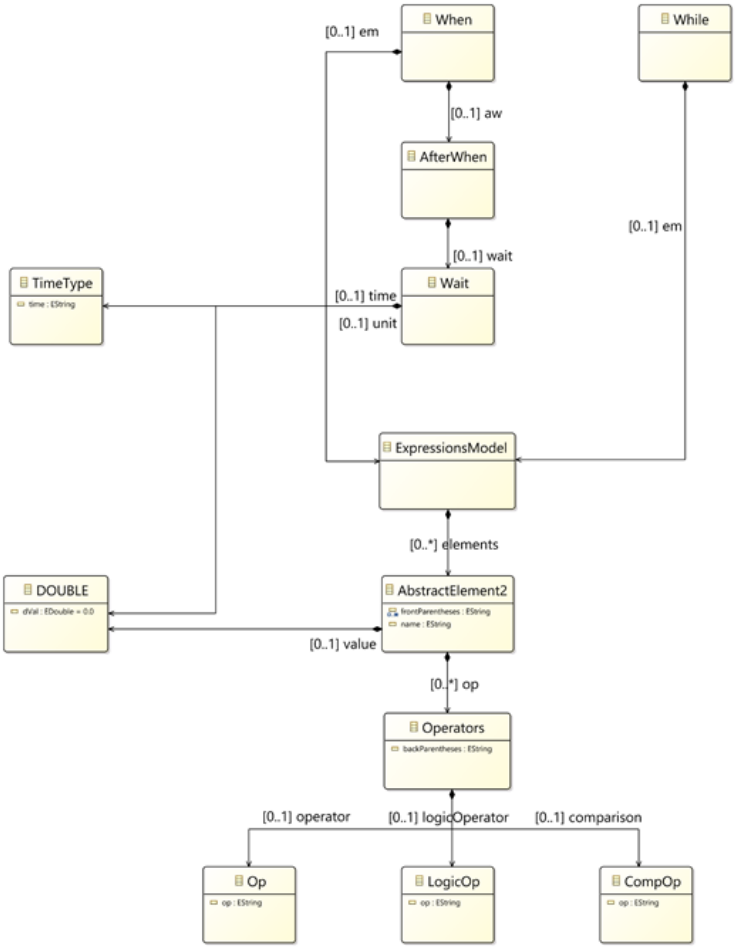

FIGURE 7 PRECONDITION RELATED SECTION OF THE MODEL

Assertion

An assertion is decisive for a test oracle to fail or succeed. The compliance or non-compliance of this condition indicates if a certain property of a CPS, modelled with the test oracle, is fulfilled or not. As previously stated, if a precondition is set, the evaluation of the assertion completely depends on the precondition. CPS properties were analysed to define the assertion. As a result, the main assertion of the oracle is composed by a signal to be evaluated, a comparison pattern, a signal to compare (we refer to this in the remainder of the deliverable as the reference signal) and, optionally, a temporal condition that determines the duration at which the assertion must be checked. The latter only makes sense if a precondition was specified for the oracle.

Figure 8 illustrates the six different patterns we identified through our conversation with industrial CPS practitioners (including Orona's engineers and companies from ADEPTNESS's advisory board) and by reviewing the state-of-the-art on CPSs testing. The red lines stand for a reference signal and denote the boundaries of the grey area. The signal being tested should be placed inside this area in order for the oracle to pass the test.

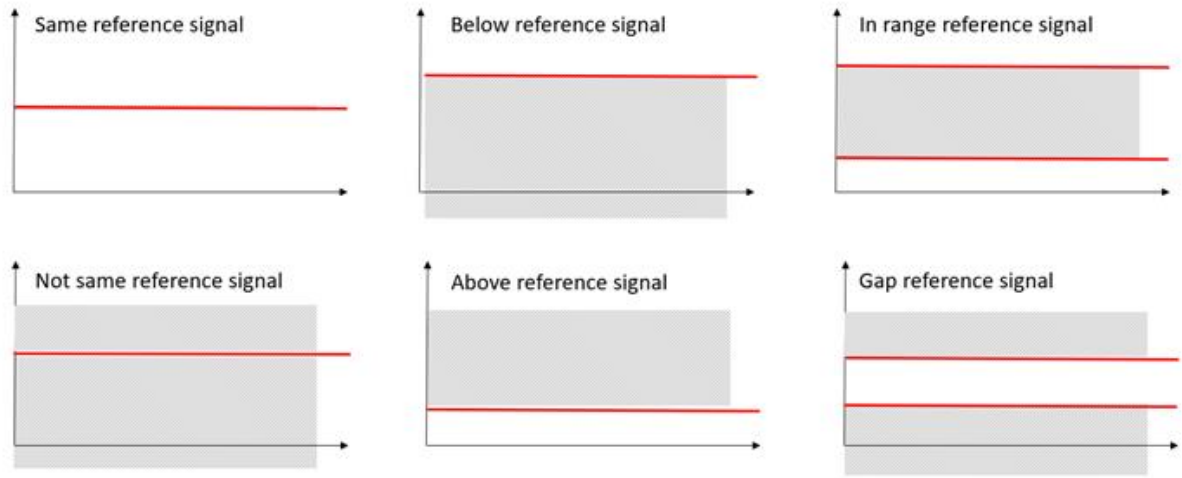

FIGURE 8 ORACLE ASSERTION PATTERNS

The following figure illustrates the section of the model where the assertion is defined. On the one hand, the signal to be assessed is defined through the *ExpressionModel*, which must be constrained afterwards to express a signal. On the other hand, we defined the comparison pattern and the reference signal. The patterns presented above are modelled through the *Same*, *NotSame*, *Upper*, *Lower*, *Range* and *Gap* components, *Upper* corresponding to the below pattern and *Lower* to the above pattern. The reference signal is modelled through the *Bound_up* or *Bound_down* component, however, in the case of the in range or gap pattern both reference signals are necessary. Finally, temporal conditions for the assertion could be established through the *At_least*, *At_most* and *Exactly* components of the model.

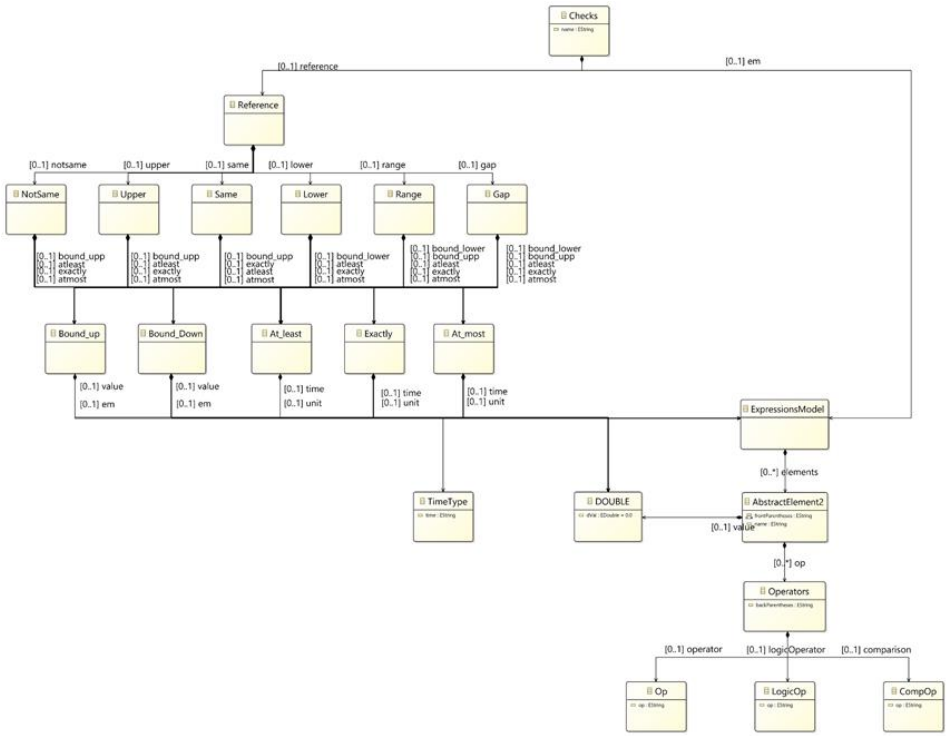

FIGURE 9 ASSERTION RELATED SECTION OF THE MODEL

Failure reason

The test oracles, in addition to delivering a verdict on whether an assertion is met or not, it was considered interesting to also indicate the degree of compliance. With this in mind, data assertions are converted into a confidence level value each cycle an assertion is evaluated. The confidence level value ranges from -1 to 1 and 2. A positive value means that the property defined within the checks is asserted as "PASS", whereas a negative value means that the property is being violated. The 2 confidence value refers to an oracle not fulfilling the specified precondition, and therefore the property is not being checked (i.e., inconclusive). The following table gathers the equations used to calculate the confidence level value for each pattern. *Sign* stands for the signal under check, *min* and *max* stand for the minimum and maximum values of the signal, respectively, and *ref*, *upperRef* and *lowerRef* stand for a reference signal of the pattern; in cases where there is only one reference signal, *ref* is used, otherwise, *upperRef* is used for the upper bound reference signal, and *lowerRef* for the lower bound reference signal.

TABLE 1 EQUATIONS FOR THE CALCULATION OF THE CONFIDENCE LEVEL VALUE FOR EACH ASSERTION PATTERN

Same	Not same
$conf(t) = \begin{cases} \frac{sign-ref}{ref-min} & \text{if } (sign < ref), \\ \frac{ref-sign}{max-ref} & \text{if } (sign > ref), \\ 1 & \text{if } (sign == ref) \end{cases}$	$conf(t) = \begin{cases} \frac{ref-sign}{ref-min} & \text{if } (sign < ref), \\ \frac{sign-ref}{max-ref} & \text{if } (sign > ref), \\ -1 & \text{if } (sign == ref) \end{cases}$
Below	**Above**
$conf(t) = \begin{cases} \frac{ref-sign}{ref-min} & \text{if } (sign < ref), \\ \frac{ref-sign}{max-ref} & \text{if } (sign \geq ref) \end{cases}$	$conf(t) = \begin{cases} \frac{sign-ref}{max-ref} & \text{if } (sign > ref), \\ \frac{sign-ref}{ref-min} & \text{if } (sign \leq ref) \end{cases}$
In range	**Gap**
$conf(t) = \begin{cases} \frac{upperRef-sign}{(upperRef-lowerRef)/2} & \text{if } (sign<upperRef \ \&\& \\ & sign>lowerRef+(upperRef-lowerRef)/2), \\ \frac{sign-lowerRef}{(upperRef-lowerRef)/2} & \text{if } (sign<lowerRef \ \&\& \\ & sign<lowerRef+(upperRef-lowerRef)/2), \\ \frac{sign-lowerRef}{lowerRef-min} & \text{if } (sign < lowerRef), \\ \frac{upperRef-sign}{max-upperRef} & \text{if } (sign > upperRef) \end{cases}$	$conf(t) = \begin{cases} \frac{sign-upperRef}{(upperRef-lowerRef)/2} & \text{if } (sign<upperRef \ \&\& \\ & sign>lowerRef+(upperRef-lowerRef)/2), \\ \frac{lowerRef-sign}{(upperRef-lowerRef)/2} & \text{if } (sign>lowerRef \ \&\& \\ & sign<lowerRef+(upperRef-lowerRef)/2), \\ \frac{lowerRef-sign}{lowerRef-min} & \text{if } (sign < lowerRef), \\ \frac{sign-upperRef}{max-upperRef} & \text{if } (sign > upperRef) \end{cases}$

However, by analysing the industrial case studies it was noticed that certain violations could be accepted, especially those related to QoS measures. To tackle this issue, two measures were taken, (1) add the possibility to adjust the confidence level, so that the user can specify a negative confidence value that it is yet acceptable and (2) define a total of four failing reasons, so that different failure patterns can be specified, each accepting certain deviations from the property to be asserted. An example for each failing reason for a below assertion pattern can be found in the figure below.

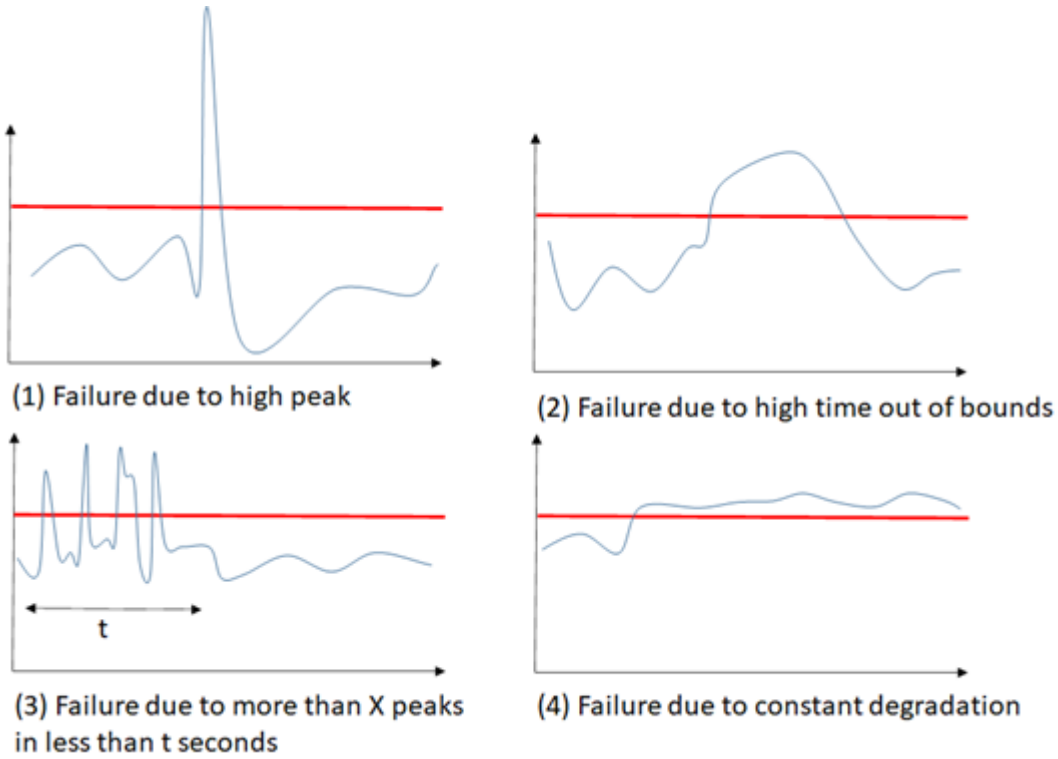

FIGURE 10 THE FOUR FAILURE REASONS FOR A BELOW PATTERN

When a below pattern is defined, values above the reference signal (drawn with a red line) result in a negative confidence value, i.e., the property defined in the oracle has been violated. This violation is then analysed against the failure patterns to decide whether it is acceptable or not. (1) is the least flexible failure pattern, a single violation makes the oracle fail. (2) accepts violations for a specific duration but fails afterwards. (3) accepts a determined number of violations, in a specified time window. (4) aims to detect constant degradation, allowing some violations as long as the overall is remains within the limits. At least one failing reason shall be specified, but the four types could be used.

The concepts introduced above are modelled as illustrated in the figure below:

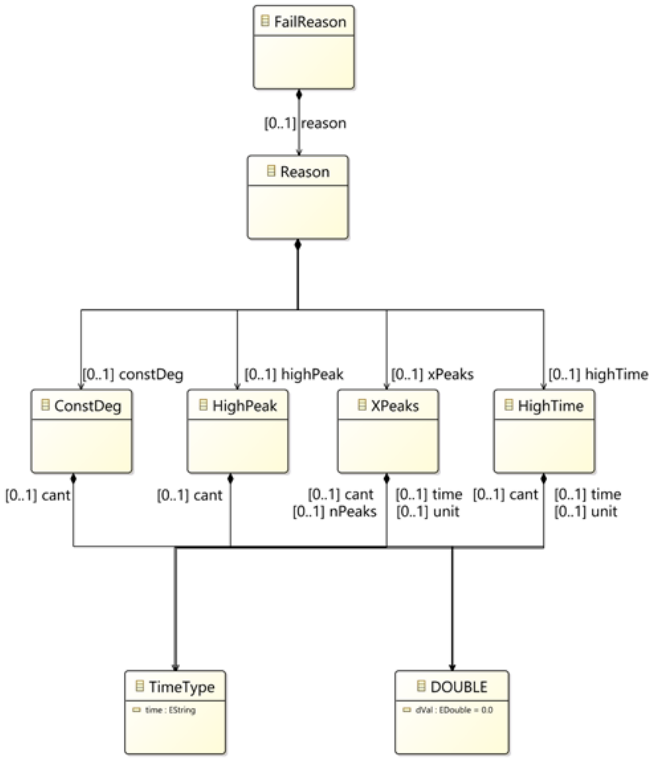

FIGURE 11 FAILURE PATTERN RELATED SECTION OF THE MODEL

4.2. Handling uncertainty in organic test oracles

One core problem of having organic oracles at operation-time for CPSoS is that CPSoS are inherent to uncertainty. For instance, in the context of Orona's use-case, several uncertain problems need to be considered, such as, the weight that passengers have, the time they take to enter into the elevators or wrong installation usages (e.g., calling an elevator but later not entering). To handle uncertainty-related problem in organic test oracles, we employed uncertainty datatype libraries from SRL's previous work UncerTum[12], which implemented an uncertainty conceptual model named U-Model[13] including three uncertainty libraries (i.e., Probability library, Vagueness library and Ambiguity library) that can facilitate specification of most uncertainties in CPS. This enables the usage of organic oracles at operation-time.

The Probability library contains data types like Percentage and various probability distributions e.g *Normal Distribution*, which can specify uncertainties in CPSoS especially related to sensor measurement, and *Poisson Distribution*, which can specify the uncertain arrivals of passengers[14]. In some cases, it is not possible to specify uncertainties as probabilities. For example, QoS-based passenger's satisfaction defined in CIBSE Guide D[15] is usually measured with fuzzy logic. Thus, data types related to fuzzy logic such as Fuzzy set and Fuzzy Interval in Vagueness library can be used. Similarly, the Ambiguity library was used for cases when probability and

[12] *Zhang M, Ali S, Yue T, et al. Uncertainty-wise cyber-physical system test modeling[J]. Software & Systems Modeling, 2019, 18(2): 1379-1418*

[13] Zhang M, Selic B, Ali S, et al. Understanding uncertainty in cyber-physical systems: a conceptual model[C]//European conference on modeling foundations and applications. Springer, Cham, 2016: 247-264

[14] *Sorsa J, Ehtamo H, Kuusinen J M, et al. Modeling uncertain passenger arrivals in the elevator dispatching problem with destination control[J]. Optimization Letters, 2018, 12(1): 171-185.*

[15] Barney G. Transportation systems in buildings: CIBSE Guide D: 2010. London: Chartered Institution of Building Services Engineers, 2010

vagueness libraries are not adequate. For example, a specific traffic profile with a specific building configuration has an acceptable distribution of AWTs over a specific period (e.g., every 5 minutes) to achieve better user satisfaction. This means that AWTs per 5 minutes should not have too much uncertainty (e.g., AWTs per 5 minutes shouldn't change frequently and drastically), which can be measured with Shannon Entropy datatype in Ambiguity library.

4.3. Integration with Adeptness Microservice template

The organic oracles presented in this section must be able to be fully integrated into the Adeptness ecosystem. The test oracles require the values of the signals of the corresponding CPS in order to perform the assessment and, additionally, test oracles must provide the verdicts back to the infrastructure. As stated in the deliverable D1.2 *Microservices interface definition* these communications are carried out through the MQTT protocol, and the same approach will be used at all stages of the software development: SiL, HiL and operation. The figure below shows these communications.

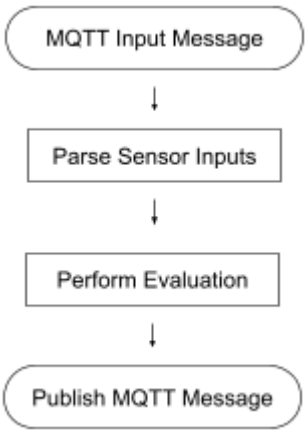

FIGURE 12 COMMUNICATIONS BETWEEN THE ADEPTNESS INFRASTRUCTURE AND THE ADEPTNESS MICROSERVICE

First, the inputs for the test oracles are published through MQTT messages by the monitors. The microservice where this test oracle is executing, gets and parses the messages and provides the inputs to the organic test oracle that is executing. The oracle, in turn, performs the evaluation, and returns the verdict back to the microservice. Finally, the verdict is published through an MQTT message to the rest of the subsystems.

The microservice is also responsible for providing the means of binding the inputs defined on the oracle to uniquely identifiable URNs and topics, as well as to externally configure the unique publication topics for the verdicts generated by oracles

5. Pre-specification Test Oracles

Another type of oracles used for testing software systems, as well as CPSs, is the so-called pre-specification test oracles. In such context, the SUT and the oracle have the same inputs, and provide an output. While the SUT provides the real test output, the test oracle provides the expected test output. Both outputs are later compared by using an organic test oracle (e.g., both outputs should be the same, the output of the SUT should be below the oracle output). Different versions of the pre-specification test oracles are available. Among them, in a DevOps context, where the software continuously evolves, the regression test oracles are the most well-known ones. In such cases, the pre-specification oracle is a previous version of the SUT. While this practice is well known, it poses significant challenges in the context of CPSs. The first one is that in the

context of CPSs, executing a test takes a long time. In these cases, the execution needs to be done twice (on the SUT and on a previous version of the SUT). The second one is that at operation-time, this is not possible unless a methodology similar to a digital twin, which should run in the cloud, is enabled.

To solve this problem, in Adeptness we have used machine-learning algorithms as substitutes of the SUT, which make the execution of tests faster, and therefore scalable both at design-time as well as operation-time. However, to ensure that such technology is capable of substituting the SUT, we have performed a preliminary experiment by using Orona's use-case.

5.1. Preliminary experimentation

In this section we explain the preliminary experiment we carried out to assess the feasibility of using machine-learning algorithms as substitutes of previous SUT versions, which would enable the algorithm to act as a regression oracle. This preliminary experimentation has been published in AST 2021[16].

The software development process of Orona's dispatching algorithm is shown in Figure XX. There are a total of three Design-Operation testing levels: Software-in-the-Loop (SiL), Hardware-in-the-Loop (HiL) and Operation. For testing at the SiL and HiL test level, two kinds of tests are carried out: (1) short-scenario tests and (2) full-day tests. The former validates specific functional properties of the system. The expected outcome of a test in this case is obtained by implementing some assertions. The latter mimic normal full-day scenarios. The expected test outcomes in this case relate to certain Quality of Service (QoS) measures over time obtained by re-executing the test in a regression test oracle.

A test case in the context of Orona for testing a dispatching algorithm version refers to the following fields:

- Building installation: It configures the environment of where the SUT is executed, and encompasses different fields (e.g., number of floors, number of elevators, elevators' characteristics).
- Test input (call list profile): a test input in this context refers to a file that includes a list of passengers. For each passenger, this file includes (1) the arrival time (i.e., when the passenger requests an elevator), (2) arrival floor, (3) destination floor, (4) weight of the passenger, (5) capacity factor by mass, (6) the loading time, (7) the unloading time and (8) information related to the behaviour of the passenger when not all elevators serve all floors.
- Expected output: based on the input, what the test outcome should be. At unitary functional level tests, the expected output is typically related to a functional behaviour of the elevator (e.g., elevator number 1 attends calls from passengers 1 and 2, elevator number 2 attends a call from passenger 3). For long full-day tests, this is related to certain QoS metrics, which is addressed in this section.

In this section we focus on the oracles applied for, what is known in Orona internally as "long full-day tests". For these kinds of tests, the inputs encompass full-day passenger data simulating the passenger flow in a building. The expected output refers to a time series of QoS values over time. Typical QoS values refer to the Average Waiting Time (AWT) or the Average Time to Destination (ATTD). The most used QoS measure in Orona is the AWT because it is the most sensitive measure for a passenger to determine whether a system of elevators performs well or not[17]. For such full-day tests, these can be differentiated into two main groups: theoretical and real.

[16] A. Arrieta, J. Ayerdi, M. Illarramendi, A. Agirre, G. Sagardui and M. Arratibel. *Using Machine Learning to Build Test Oracles: an Industrial Case Study on elevators Dispatching Algorithms*. Automated Software Testing conference. 2021.

[17] G. Barney and L. Al-Sharif, *Elevator traffic handbook: theory and practice*. Routledge, 2015.

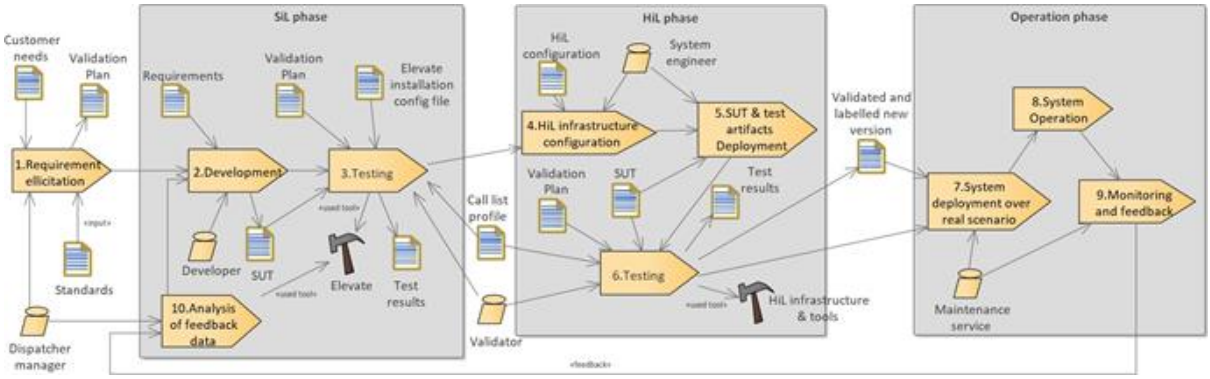

FIGURE 13 SOFTWARE DEVELOPMENT PROCESS OF ORONA'S DISPATCHING ALGORITHMS[18]

On the one hand, theoretical passenger data-based test cases provide test inputs based on theoretical studies of passenger flows in buildings. An example of such test cases is shown in Figure 2, where a graph with the number of up calls, down calls and inter-floor calls in a time window of five minutes for a simulation of 13 hours based on the Siikonen theory for a building of offices can be seen. On the other hand, Orona uses data obtained from real installations to test their dispatching algorithms in more realistic conditions. This helps with the validation of dispatching algorithms from several perspectives, e.g., the identification of certain patterns that were not considered in theoretical traffic profiles.

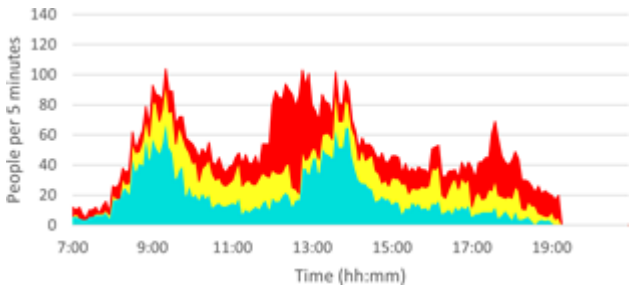

FIGURE 14 A GRAPH SHOWING THE NUMBER OF UP CALLS (BLUE), DOWN CALLS (RED) AND INTER-FLOOR CALLS (YELLOW) IN A THEORETICAL TRAFFIC PROFILE

5.1.1. Pre-specification test oracle for elevators dispatching algorithms based on machine-learning algorithms

Figure 15 Overview of the approach at the SiL test level shows the overall architecture of the architecture we have developed to validate whether machine-learning algorithms are a good substitute of regression test oracles.

[18] J. Ayerdi, A. Garciandia, A. Arrieta, W. Afzal, E. P. Enoiu, A. Agirre, G. Sagardui, M. Arratibel, and O. Sellin. *Towards a taxonomy for eliciting design-operation continuum requirements of cyber-physical systems.* in 28th IEEE International Conference on Requirements Engineering, RE 2020, Zurich, Switzerland, 2020, 2020.

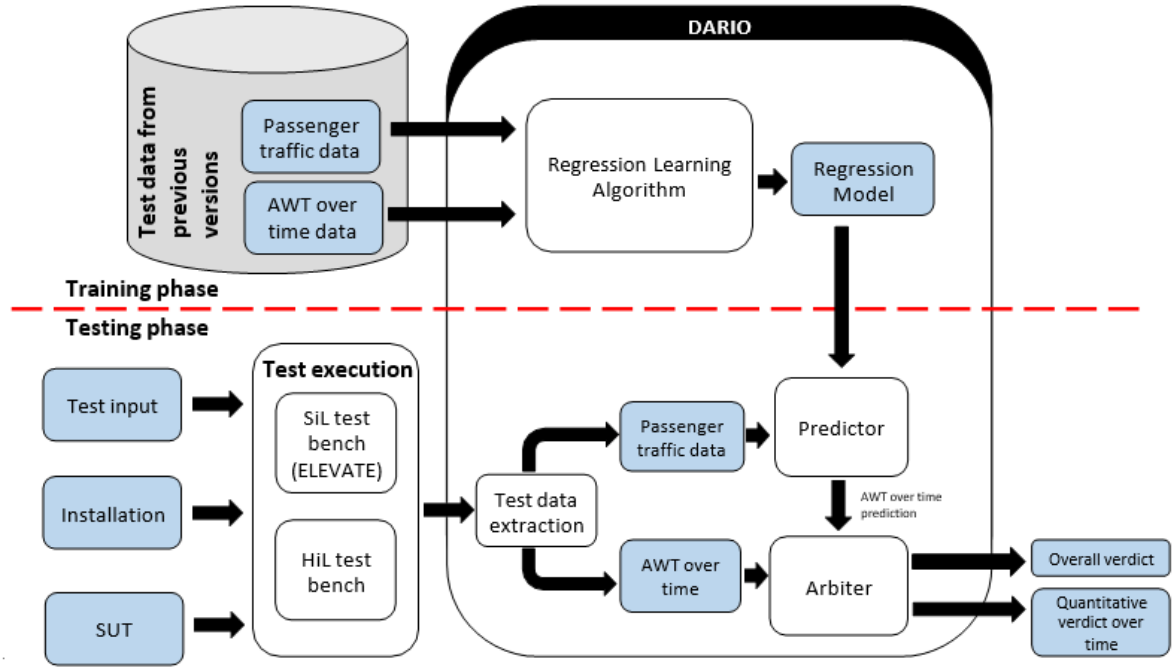

FIGURE 15 OVERVIEW OF THE APPROACH AT THE SIL TEST LEVEL

It has two main phases: (1) the training phase and (2) the testing phase. During the training phase, a machine learning algorithm is trained by using data from previous software versions, which is used by Orona to validate other versions of the software. The algorithm yields a model, which is used by DARIO (Dispatching Algorithm Oracle) in the testing phase. During the testing phase, we use Elevate to execute test cases. When the test has finished, Elevate provides a set of files, which are treated by DARIO to extract (1) the passenger traffic profile and (2) the AWT over the simulation time. The passenger traffic profile is used by the model predictor to predict the AWT over the simulation time. The AWT over the simulation time is used by the arbiter of DARIO to yield the overall verdict and a quantitative verdict over the time.

Training phase

The training phase in our approach aims at providing a machine-learning algorithm with labelled data to train it. During such phase, a machine-learning algorithm adapts some internal parameters based on training data so that it performs well on future unseen input data [6]. In Orona, the Verification and Validation activities are well documented, thus, obtaining the required data to train an algorithm is straightforward.

We used the AWT to label a test as PASS or FAIL. The AWT can be a global value that measures the overall AWT for all passengers in the test input or a signal over the simulation time, indicating the AWT of the passengers in the test input for a specific time period. Elevate provides information of both. DARIO uses the AWT for a specific time period to determine a verdict.

For training a machine-learning algorithm, we categorised the data into different domain-specific features related to passengers traffic data. As for the output feature, the AWT QoS metric is considered, as it is the measure that the dispatching algorithm under test used in this paper targets. All of them, for a time window of five minutes. We chose a five-minute time window based on the information provided by Elevate. A script was developed to automatically extract this data from a database where Orona saves all the test history. When the data was extracted, the training phase was launched by the script, using the MATLAB machine-learning toolbox. The regression learning algorithm yields a trained regression model, which can later be used in the testing phase to predict the AWT.

Usually, the passenger traffic data in the historical test database is not the same as the test input in the testing phase because when changes are made in the dispatching algorithm, these changes typically include new functionalities or bug corrections. Subsequently, in the test inputs used during the testing phase, testing the new functionality or a scenario that aims to trigger the fault is usually implemented. In addition, at the HiL test level, tests also might include scenarios where the test engineer tests the Human Machine Interface (HMI) of the system. In those cases, as the testing is manual, where by the system interacts with the tester, having the exact same test case is impossible.

Testing phase

When the regression learning algorithm is trained, it yields a trained machine-learning model, which is used in the testing phase. For the current implementation, this phase has four steps:

The first step refers to test execution, where the dispatching algorithm is tested by using simulation-based testing, The second step refers to test data extraction, where the test results and other necessary data is extracted. The third step refers to prediction based on the regression model, which yields the expected AWT result. Lastly, the fourth step refers to the arbitration process, which compares the AWT obtained by the regression algorithm with the AWT estimated by the regression model by using an organic oracle (explained in Section 4). We now explain all these steps in further detail.

a. Test execution: To execute a test, simulation-based testing is employed. As previously mentioned, the test can be executed at two distinct levels: (1) at the Software-in-the-Loop test level, where Elevate might be used and (2) at the Hardware-in-the-Loop test level, where the HiL test bench from Orona is used.

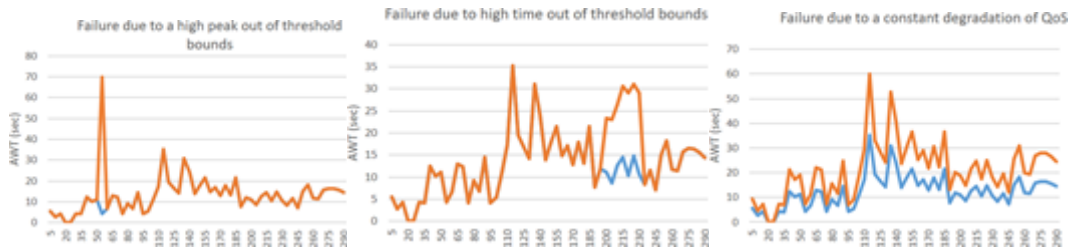

FIGURE 16 THE THREE REASONS WHY A TEST CAN BE CATALOGUED AS FAIL (BLUE SIGNAL REFERS TO THE REFERENCE AWT AND ORANGE SIGNAL REFERS TO THE AWT OBTAINED BY THE SOFTWARE VERSION UNDER TEST)

b. Test data extraction: After the test has been executed, the tool extracts the necessary data from the testing files yielded by the test execution tools. At both test levels, i.e., SiL and HiL, both files are the same, which allows better reusability of the implemented test data extraction functionality.
c. Prediction: The data is sent to the regression model yielded during the training phase. This model makes the inference, which means it predicts for the test data, which is the reference AWT. This reference is later compared by the test arbiter in the fourth Step.
d. Arbitration: The arbitration process compares the AWT obtained by the algorithm with the AWT predicted by the inference. The arbiter uses an organic oracle defined in the previous section. Basically, the oracle determines that the AWT of the new version of the algorithm should be below the yielded AWT. Furthermore, by discussing with engineers from Orona, we determined three failing reasons to catalogue a test as FAIL or as PASS, as discussed in 4.1.2. Figure 16 The three reasons why a test can be catalogued as FAIL (blue signal refers to the reference AWT and orange signal refers to the AWT obtained by the software version under test) shows the three examples.

The first reason might be that at certain point, the software version under test shows a high peak on the AWT measure. This is because at a certain point, probably due to a bug, at least one passenger was unattended for a long period of time. The second reason is because the AWT measure for the software version under test exhibits a value higher than the specified threshold for a long period of time. The last scenario is related to a constant degradation of the AWT value throughout all the steps of the execution. As explained in Deliverable

4.1 *DSL for continuous validation of CPSoS*, and in 4.1.2 *Oracle definition*, a numerical value of the verdict is yielded, and the test is catalogued as PASS or FAIL based on this value and the specified failing reasons.

Implementation

Since the goal was to perform a fast validation of the potential of machine-learning algorithms to assess their effectiveness, we implemented it in MATLAB. This tool also provides support for a wide variety of algorithms. Additionally, although the approach is generalisable to any regression machine-learning algorithm, our implementation was on top of the following ones: (1) Support Vector Machines (SVM), (2) Regression Decision Trees, (3) Ensemble, (4) Regression Gaussian Process (RGP) and (5) Stepwise Regression. The reason why these algorithms were chosen was (1) availability within the MATLAB framework and (2) appropriateness for our context in terms of prediction speed, training speed, memory usage and required tuning.

The selected algorithms have a fast prediction and training speed (unlike other algorithms such as neural networks). In addition, these algorithms have a small memory usage, something important when deploying the oracles in operation integrated with the Adeptness microservice. Lastly, the selected algorithms require minimal tuning, something that is paramount to ease the transfer of the approach to practitioners.

5.1.2. Empirical evaluation

We now explain how we evaluated our approach. Our evaluation aims to answer the following Research Questions (RQs):

- RQ1 – Training with theoretical data: Which machine learning algorithm yields the most accurate test verdicts when trained with theoretical passenger test data?
- RQ2 – Training with real passenger data: Which machine learning algorithm yields the most accurate test verdicts when trained with real passenger test data?

Experimental setup

Case study: We used a real dispatching algorithm provided by Orona, named as Conventional Group Control (CGC) algorithm, which is the most used algorithm in Orona. Therefore, there were several versions of this algorithm available in Orona, which allowed us to have access to several sets of relevant test data for performing the experiments. In the evaluation, we used a complex building installation that Orona uses to validate dispatching algorithms, which is related to a real installation named the communication city, in Madrid. The building has a total of 10 floors and six elevators, each having a capacity of 1250 Kg weight and 16 passengers. Another reason for choosing this building is that Orona has relevant data obtained from the real installation while in operation, which allowed us to answer RQ2. To train the machine learning algorithms of our oracles, we used available test data for testing a previous version of the CGC algorithm within the specific building. This data included ten theoretical passenger list test inputs and four real passenger list test input data.

Evaluation metrics: Mutation testing was used to inject faults through the dispatching algorithm under test. This technique has been found to be a good substitute for real faults in the past [8]. The dispatching algorithms are programmed in C, thus traditional mutation operators for the C programming language were used. These mutations were introduced in a uniform manner throughout the different sections of the source code that are relevant in the simulation environment. A total of 99 mutants were generated. It is important to note that as simulation-based testing was used, what means that executing each mutant takes a long time. This number is similar or larger to other studies where simulation-based testing was used to evaluate testing approaches

[19], [20], [21]. From these 99 mutants, 18 were removed from the evaluation, because they crashed or passengers because passengers were not attended. In practice, both types of failures are easily detectable without the need of an oracle. Thus, we ended up with 81 mutants in our evaluation. The 81 mutants were reviewed by a domain expert to check that they were not semantically equivalent to the original program.

When tests were executed, we obtained the AWT over the time using Elevate, the simulation tool used by Orona to validate elevators dispatching algorithms. Based on a similar work [22], we selected four measures to evaluate the quality of the test oracles: precision, recall, f1 and accuracy. Accuracy is especially important in our study as it is the only measure that considers True Negatives. In our context, classifying faults well is as important as classifying correct behaviour as correct.

For each mutant and each test case, we considered the overall verdict returned by DARIO, catalogued either as "PASS" or "FAIL". Additionally, we used the same passenger list with a regression test oracle (i.e., an original previous version under test), which is the current practice to determine if a test passes or fails by Orona. Similar to other works tackling the test oracle problem [23], the verdict provided by DARIO was considered a true negative (TN), a true positive (TP), a false negative (FN) or a false positive (FP) as defined below:

- TN: Both the test oracle (i.e., DARIO) and the regression test oracle returned a "PASS" verdict.
- TP: Both the test oracle (i.e., DARIO) and the regression test oracle returned a "FAIL" verdict.
- FN: The test oracle (i.e., DARIO) returned a "PASS" and the regression test oracle returned a "FAIL".
- FP: The test oracle (i.e., DARIO) returned a "FAIL" and the regression test oracle returned a "PASS".

The tests were executed in Elevate instead of in the HiL due to practicality (i.e., if the tests were executed using the HiL test bench, the experiments would take around 2 years).

A total of four experimental scenarios were designed for answering the two RQs:

- Scenario 1: To answer the first RQ, we first used test cases that involve theoretical test inputs. These test inputs were automatically generated by Elevate and are based on a study performed by Siikonen [24]. In total, 10 of these test cases were used. We employed the 10-fold cross validation to validate DARIO along with all the selected machine-learning algorithms.
- Scenario 2: We used an additional scenario to answer the first RQ. The same type of test cases were used for training, but for testing, test cases obtained from the real installation were used. This scenario would emulate (1) how the theoretical passenger data performed in order to train the algorithms during validation when using data extracted from operation and (2) how the theoretical passenger data perform for training the algorithms when the oracle is used in the real installation.
- Scenario 3: To answer the second RQ, we used test cases obtained with data extracted from the real installation in operation. In total, four of these test cases were used. We thus employed the 4-fold cross validation to validate DARIO along with all the selected machine-learning algorithms.

---

[19] *R. Matinnejad, S. Nejati, L. C. Briand, and T. Bruckmann, "Test generation and test prioritization for simulink models with dynamic behavior," IEEE Trans. Software Eng., vol. 45, no. 9, pp. 919–944, 2019. [Online]. Available: https://doi.org/10.1109/TSE.2018.2811489*

[20] *A. Arrieta, S. Wang, U. Markiegi, A. Arruabarrena, L. Etxeberria, and G. Sagardui, "Pareto efficient multi-objective black-box test case selection for simulation-based testing," Information & Software Technology, vol. 114, pp. 137–154, 2019. [Online]. Available: https://doi.org/10.1016/j.infsof.2019.06.009*

[21] *A. Arrieta, S. Wang, A. Arruabarrena, U. Markiegi, G. Sagardui, and L. Etxeberria, "Multi-objective black-box test case selection for cost-effectively testing simulation models," in Proceedings of the Genetic and Evolutionary Computation Conference, ser. GECCO '18. New York, NY, USA: ACM, 2018, pp. 1411–1418. [Online]. Available: http://doi.acm.org/10.1145/3205455.3205490*

[22] *A. E. Genc¸, H. Sozer, M. F. Kırac¸, and B. Aktemur, "Advisor: An ad- ¨justable framework for test oracle automation of visual output systems," IEEE Transactions on Reliability, 2019.*

[23] *W. K. Chan, J. C. Ho, and T. Tse, "Finding failures from passed test cases: Improving the pattern classification approach to the testing of mesh simplification programs," Software Testing, Verification and Reliability, vol. 20, no. 2, pp. 89–120, 2010.*

[24] *M.-L. Siikonen,"On traffic planning methodology," Elevator technology, vol. 10, pp. 267–274, 2000.*

- Scenario 4: Similar to Scenario 2, we used an additional scenario to answer the second RQ. In this case, we used the same test cases as in Scenario 3 to train the algorithms, but for testing we used the ten theoretical passenger-based test cases.

Analysis of the results and discussion

The Table below summarizes the obtained results for the four scenarios designed to answer the RQs of our preliminary evaluation. The first RQ aimed at assessing the performance of the machine-learning algorithm when trained with theoretical passengers. Scenario 1 used a 10-fold cross validation with 10 theoretical passenger data. In terms of precision, SVM was the technique outperforming the remaining algorithms, followed by regression tree and stepwiselm. As for the recall measure, regression tree and ensemble performed best, followed by SVM. In terms of accuracy and F-measure, SVM performed best, although the results by regression tree were close in both cases, unlike the remaining three machine learning algorithms, which dropped below 0.8 in terms of both accuracy and F-measure.

Scenario	Metrics	SVM	Regression tree	Ensemble	RGP	Stepwiselm
Scenario 1	Precision	0.89	0.83	0.68	0.76	0.82
	Recall	0.88	0.89	0.89	0.78	0.78
	Accuracy	0.89	0.87	0.69	0.71	0.79
	F-1	0.86	0.83	0.70	0.68	0.74
Scenario 2	Precision	0.74	0.80	0.41	0.61	0.41
	Recall	0.80	0.75	0.86	0.86	0.85
	Accuracy	0.74	0.75	0.39	0.58	0.38
	F-1	0.70	0.69	0.45	0.59	0.44
Scenario 3	Precision	0.60	0.76	0.29	0.59	0.70
	Recall	0.94	0.98	1.00	0.88	0.83
	Accuracy	0.59	0.79	0.37	0.58	0.74
	F-1	0.64	0.80	0.44	0.62	0.70
Scenario 4	Precision	0.25	0.25	0.25	0.30	0.25
	Recall	1.00	1.00	1.00	0.99	1.00
	Accuracy	0.25	0.25	0.25	0.37	0.28
	F-1	0.39	0.39	0.39	0.45	0.40

TABLE 2: SUMMARY OF RESULTS FOR THE FOUR EXPERIMENTAL SCENARIOS

In the second scenario, ten theoretical passenger data were used for training, whereas four real-world operational data for testing. Regression tree performed best in terms of average precision, followed by SVM. The results for ensemble RGP and Stepwise algorithms were quite low in terms of the average precision. While these three algorithms slightly outperformed SVM and regression tree for the recall metric, their accuracy and F-measure were low as compared with SVM and Regression tree.

Overall, the results for all the four measures in Scenario 2 were lower than those shown in Scenario 1. We conjecture that this is because the difference between the types of passenger traffic flow in the test cases that are based on theoretical traffic profiles and the ones obtained from the real installation. Our hypothesis is that the theoretical passenger profiles do not explore areas which the real passenger profiles actually do and vice-versa.

The second RQ aimed at answering how the performance of the different machine-learning algorithms when trained with real passenger data-based test cases was. Scenario 3 in Table 2 shows the results for the 4-fold cross validation when using the real passenger data-based test cases both, for training and for testing. In this case, regression tree performed best in terms of precision, accuracy, and F-measure. In addition, with a recall of 0.98, the regression tree algorithm was the second best, after ensemble. Nevertheless, the precision, accuracy and F-measure values for ensemble were all below 0.5, which means that this algorithm had a high number of false positives.

Scenario 4 also used real passenger data-based test cases to train the machine learning algorithms. Although all algorithms performed well in terms of recall, meaning that they produced none or a low number of false negatives, their results in terms of precision, recall and accuracy were below 0.3. This is due to a high number of false positives.

The hypothesis in this case is similar to the one for RQ1. The traffic profiles obtained from the real building installation might not exercise areas or produce situations that are considered in the theoretical traffic profiles. This makes it difficult for the regression algorithms to accurately predict the reference AWT value.

Conclusion of the preliminary experiment

This preliminary experiment was carried out to assess whether machine learning algorithms are appropriate substitute of traditional regression oracles. The results suggest that they are at least for one of the use-cases from the Adeptness project, especially if they are trained with the same type of passenger files (i.e., real, or theoretical). We called this Oracle DARIO because it was developed thinking on the Dispatching algorithms from Orona. However, the algorithm is generalizable to any kind of Cyber-Physical System. Compared with the traditionally used regression oracles, which have several disadvantages, DARIO trains machine-learning algorithms with previous test data. This training takes only a few seconds (always less than 3 seconds), whereas executing the regression test oracle takes minutes or hours at SiL (depending on the length of the test case), and hours or days at HiL (not being possible to correctly perform some tests, such as those involving HMI). In our evaluation, where an industrial dispatching algorithm from Orona was used, the accuracy of the proposed test oracle when labelling tests as PASS or FAIL ranged between 0.79 and 0.87, which is competent enough to conclude that machine-learning algorithms are appropriate to substitute regression test oracles, and therefore, be a potential solution of such kind of oracles for operational testing of CPSoS.

5.2. Pre-specification oracles in the ADEPTNESS context

The aforementioned studies have provided valuable insight into pre-specification oracles, which has been applied to the ADEPTNESS context. Two design principles have been followed to do so: To implement the pre-specification oracles as an extension of organic oracles, and to separate the computation of the training and the inference phases. In the current section we explain these design principles.

5.2.1. Machine Learning to learn unobserved signals

Pre-specification oracles are designed to take advantage of the work done in organic ones, so that expert designed rules can be combined with rules learnt from data. ML models are used to predict unobserved signals by using supervised learning methods, based on which, the assertion rules can be applied. This design allows the best of both worlds, the reliability (based on the explainability) of expert designed assertions and the flexibility of ML methods.

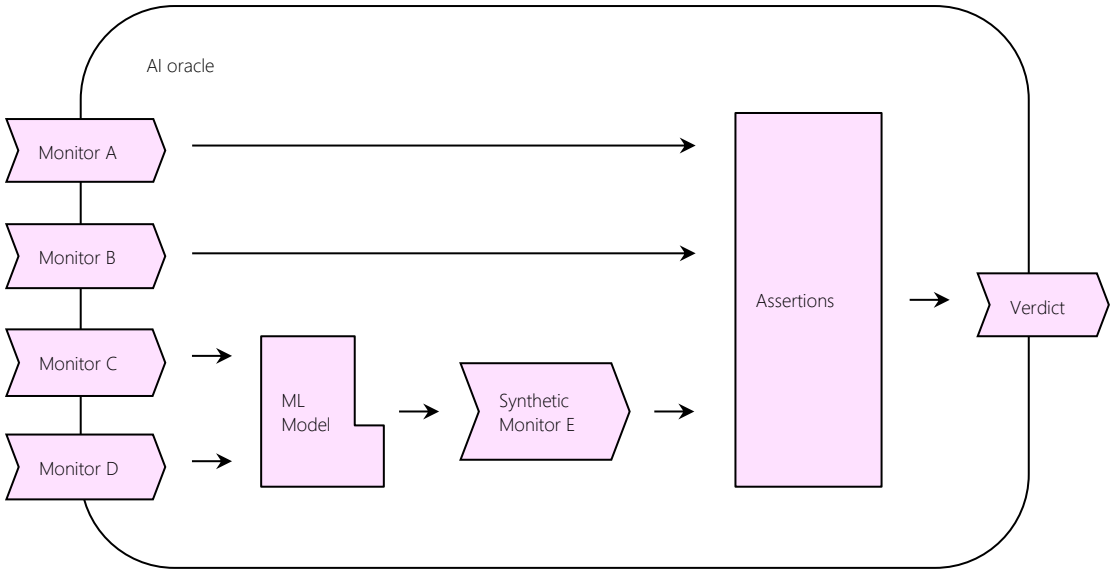

FIGURE 17 USING ML TO INFER UNOBSERVED SIGNALS

As shown in 4 Organic oracles at operation, organic oracles rely on several monitors that reflect actual physical measurements made by the CPS in order to reach their verdicts. On the other hand, the assertions made by pre-specification oracles, in addition to physical monitors, may also depend on regression monitors. As can be seen in Figure 17, ML models are used to infer regression monitors from other monitors. These monitors can be used to represent unobserved variables of the CPS, or to model complex relationships between variables.

5.2.2. Differentiating the training phase from the inference phase

As in other supervised learning scenarios, ML models must be learned from previous observations, in the so-called training phase. Depending on the ML model used this training phase requires more or less calculations, although in most scenarios is considered a computationally demanding task. Nevertheless, once the ML model is trained, the inference process is less resource intensive.

Being one of the main limitations of edge devices the lack of computational power, we propose a two-step process, where, first, we learn the model from the training data in computationally capable resource, and then, we use this model in the edge devices in order to infer a regression monitor, and, based on the defined assertions, deliver the verdict of a possible malfunction of the CPSoS.

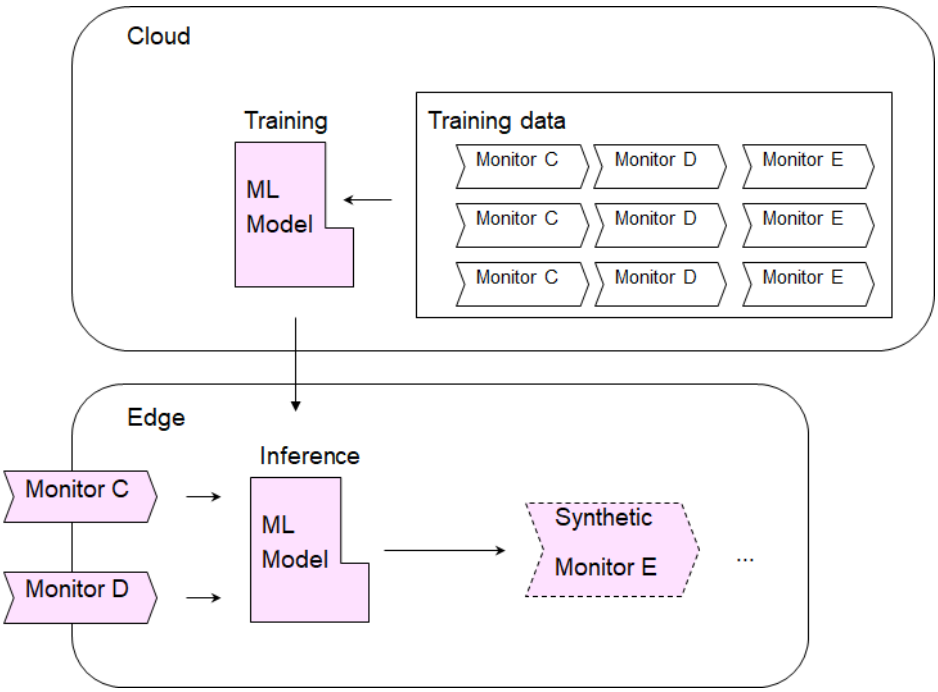

FIGURE 18 TRAINING AND INFERENCE PHASES OF THE SUPERVISED LEARNING PROCESS

Figure 18 shows this two-step computing process. In this example, there are two physical monitors on the edge device, C and D, from which the regression monitor E is being inferred. The ML model that is used to perform the inference has been trained using previously gathered data in a computationally capable resource. When collecting these data, it must be labelled with the desired output of the regression monitor E. If the regression monitor E represents a signal of a sensor present in the SiL or HiL phases of the testing, it can be used to label the training data, to then, at operational time, mimic the output of the sensor, essentially eliminating the need for a sensor or other direct observation method.

For the training phase, as a computationally capable resource the Adeptness cloud infrastructure is provided as well as means to parametrize and configure the training phase. However, if the provided infrastructure is not enough for the particular requirements of the ML model due to its complexity, any previously trained model, compatible with the Adeptness inference process, is allowed.

5.2.3. Pre-specification oracle's Model

As stated previously, pre-specification oracles take advantage of the organic oracles. The following is the section of the model that gathers the part regarding the pre-specification oracles. As can be seen, the *InferMonitoringFile* and the *ModelFile* components are added at the same level as the *Oracle* definition file and *MonitoringFile*. The *InferMonitoringFile* is composed of the *MonitoringInferVariables* components and a *ModelFile*. *MonitoringInferVariables* has the following attributes: a name, a model, a maximum and a minimum value, and a *Sig_type*. The *ModelFile* component imports a *MonitoringFile* and could be composed by various *NonTrainableModel*s as well as *TrainableModel*s. The former has the following attributes: a name, a variable's list, and a model. The latter, on the other hand, has also a name and a variable's list attributes, but also, a *dataFile* and is composed of a *Layer*, and this, in turn is composed by a *DenseType* component. This model enables the definition of trainable models, as well as the application of previously trained models.

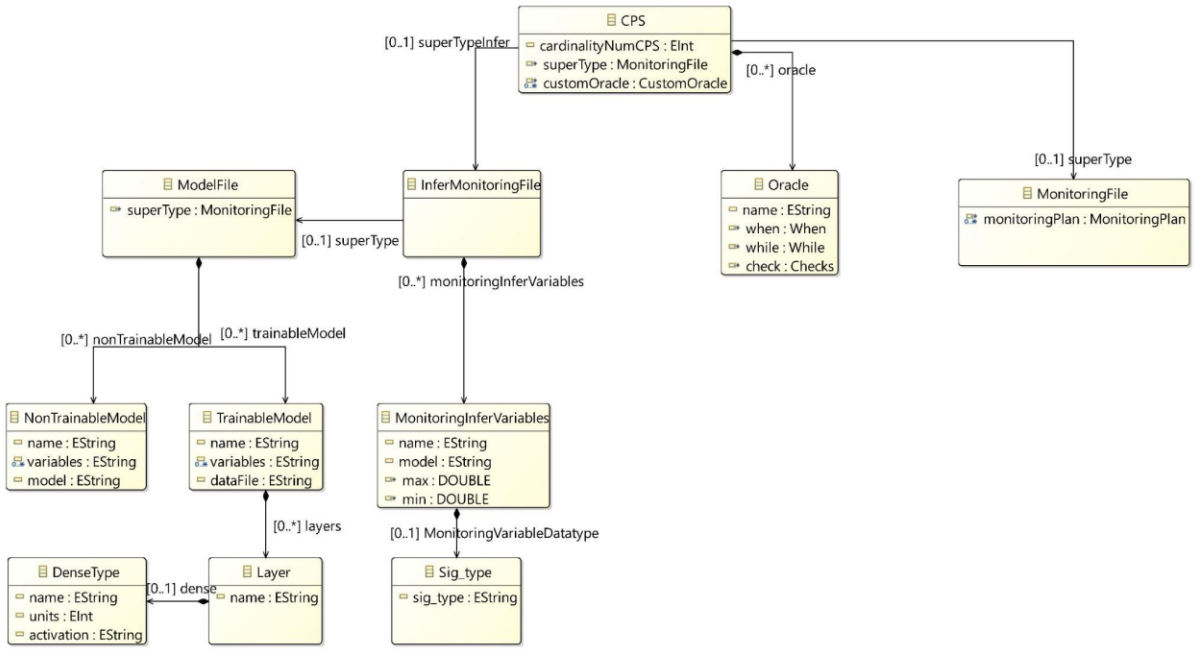

FIGURE 19 THE SECTION OF THE MODEL CORRESPONDING TO THE PRE-SPECIFICATION ORACLES

5.3. Integration with Adeptness microservice template

The communications among the organic test oracle and the Adeptness ecosystem were presented in the 4.3 *Integration with Adeptness Microservice template*. Pre-specification oracles, as an extension of organic test oracles, behave similarly. The only difference is that, once the sensor inputs are parsed, the inference is performed to obtain all the regression monitors before any evaluation is carried out.

The figure below shows how pre-specification oracles are integrated into the Adeptness ecosystem. The stage in pink is the only difference with regards to the organic test oracles.

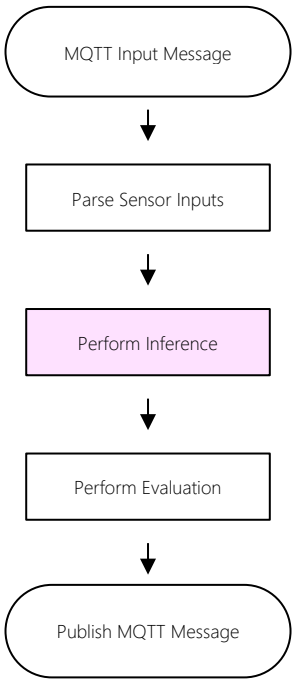

FIGURE 20 INTEGRATION OF THE PRE-SPECIFICATION TEST ORACLES WITH THE ADEPTNESS ECOSYSTEM

6. Metamorphic-based test oracles

In order to evaluate the feasibility and effectiveness of metamorphic testing, similar to the previous section, we performed some preliminary experiments on Orona's CGC elevator dispatching algorithm, which is the algorithm used in most of the company's multi-elevator installations. We published these preliminary experiments in our ISSRE 2020 article.[25]

6.2. Testing interface

For the metamorphic oracle, we decide to abstract the scenarios from the elevation domain as test cases with two complex inputs and a single output. The inputs of a test case are:

- The list of *available elevators* (variable number of elevators) and their initial position (floor).
- The *passengers list*, which is a variable length list in which each passenger has an arrival time, source floor, destination floor, and weight.

Every other parameter from the elevation scenarios is always kept at its default value for simplicity.

Our MRs are defined based on non-functional properties (the QoS metrics) of the system, which in practice makes this approach similar to performance metamorphic testing.

However, in contrast to previous work on performance metamorphic testing, our aim is detecting functional failures (i.e., incorrect, or inefficient choices from the dispatching algorithm) rather than performance bugs.

Each Metamorphic Relation (MR) we define will only consider a single QoS metric obtained by the system during the test execution as an output. Our MRs use either of the following QoS measures:

- *Average Waiting Time (AWT)*: The average time from the moment a landing call is issued until an elevator stops to attend the call, measured in seconds.
- *Total Distance (TD)*: The sum of the distances traversed by all the elevators of the building, measured in floors.
- *Total Movements (TM)*: The count of all the movements (i.e., engine start-ups) of all the elevators of the building.

6.3. Metamorphic Relations

With the specified interface, we proposed the following Metamorphic Relation Input Patterns (MRIPs) for our MRs:

- *MRIP1 Additional calls*: This MRIP consists in appending an additional passenger call to the follow-up test case, resulting in generally worse QoS measures due to the increased workload (additional passenger).
- *MRIP2 Additional elevators*: This MRIP consists in enabling one or more additional elevators in the follow-up test case, resulting in generally better QoS measures due to the system having more resources (elevators) to work with for the same workload (passengers list).
- *MRIP3 Initial position change*: This MRIP consists in changing the initial positions of the elevators without changing their number, resulting in generally similar QoS measures due to the initial positions

[25] Ayerdi, J., Segura, S., Arrieta, A., Arratibel, G. S., & Arratibel, M. (2020, October). *QoS-aware metamorphic testing: An elevation case study*. In 2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE) (pp. 104-114). IEEE.

having a limited impact on the scenario (specifically, we define an upper bound on the difference between the QoS measures obtained in the source and follow-up executions).

The following figure depicts a source test case on the left, and a follow-up test case for *MRIP3 (Initial position change)* on the right. Every parameter not mentioned in the MRIPs is always kept identical in the source and follow-up test cases.

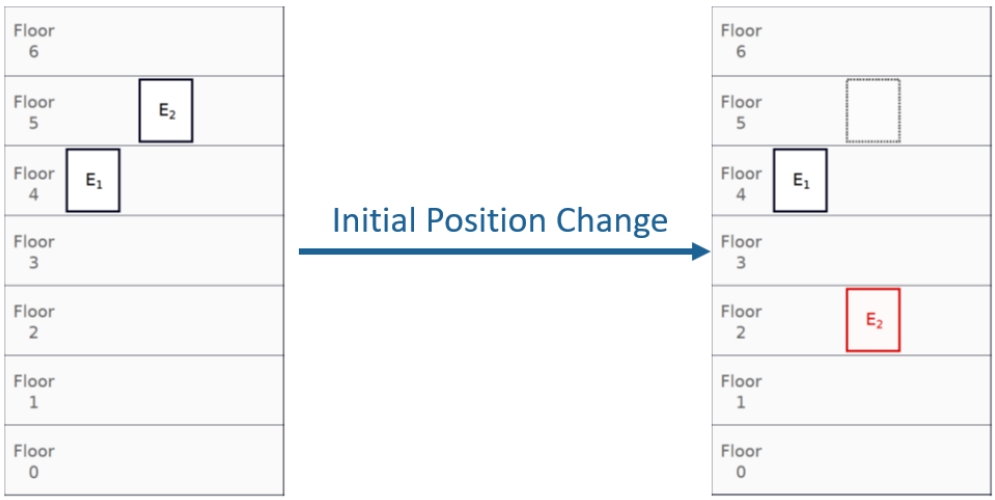

FIGURE 21 MRIP3 INITIAL POSITION CHANGE INPUT TRANSFORMATION

The specific MRs were defined with the help of a domain expert in order to ensure their soundness. In practice, however, these MRs could still yield false positives. This is due to the nature of performance testing oracles such as these MRs: The inherent non-determinism of the obtained QoS measures (e.g., simulation/sensor inaccuracies), as well as the approximate nature of some of the system's algorithms (multi-elevator dispatchers cannot always issue optimal dispatches).

In order to mitigate this issue, we define some *tolerance thresholds* that allow small deviations from the QoS measure expected in the follow-up test execution. The following image depicts the actual threshold of QoS values from the follow-up test execution that an MR would consider correct (in green, passing verdict) and incorrect (in red, failing verdict). The orange vertical lines show the exact frontier values when evaluating the MR strictly, and the red vertical lines show the actual frontier values after adding a tolerance threshold to the MR.

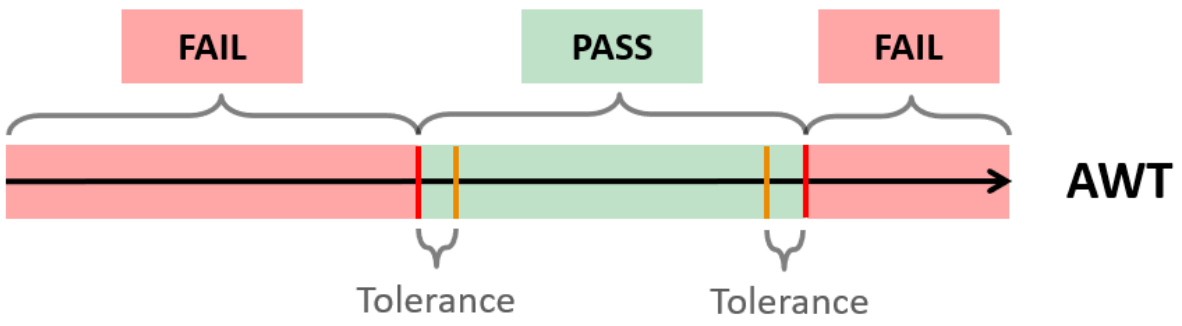

FIGURE 22TOLERANCE THRESHOLDS FOR METAMORPHIC ORACLES

Furthermore, in order to differentiate slight deviations and huge differences between the expected value of a QoS metric and the actual value, all of our MRs provide a quantitative verdict which indicates the severity of the failures. The quantitative verdict can be calculated as the difference between the actual and closest acceptable QoS measure obtained by the follow-up test case. These quantitative verdicts can be used to

direct the attention of the test engineers towards the most severe failures, which generally should be prioritized.

6.4. Experimental setup

In order to evaluate the effectiveness of *GAssertMRs*, we performed an experiment which uses Orona's Conventional GroupControl (CGC) elevator dispatching algorithm as the system under test.

In order to perform this evaluation, we generated **89 mutants** of the algorithm by injecting operator mutations to its C source code, which we used to obtain faulty test executions for the tool. In turn, the correct executions were obtained by executing the original dispatching algorithm, which has already been thoroughly validated in the building which we use to execute our evaluation.

The test cases were generated based on a template project of a real building with 10 floors and up to 6 elevators. For each test case, the number of available elevators, their initial positions, and the passengers which arrive at the building are randomly generated. In total, we generated 1340 distinct test cases, each of which was executed on all the 90 system variants (the original system and the 89 mutants).

6.5. Experimental results

After executing the test cases, all of the proposed MRs combined killed 74 out of 89 mutants, which resulted in a **mutation score of 83%**. Nevertheless, the analysis of each individual Metamorphic Relation (MR) revealed that there is a great disparity between their performances. The original dispatcher was also verified with the proposed MRs and the same test cases, and **none of the metamorphic tests yielded any false positives** for it.

The following table shows the mutation scores of each individual MR, as well as the aggregate results for each MRIP and the global score:

TABLE 3 SUMMARY OF THE EXPERIMENTAL RESULTS FOR THE METAMORPHIC ORACLES

Metamorphic Relation		Mutation Score		
MRIP1	$MR1_{AWT}$	55.1%	78.8%	83.1%
	$MR1_{TD}$	74.2%		
	$MR1_{TM}$	56.2%		
MRIP2	$MR2_{AWT}$	57.3%	59.6%	
	$MR2_{TD}$	14.6%		
	$MR2_{TM}$	5.6%		
MRIP3	$MR3_{AWT}$	14.6%	46.1%	

	$MR3_{TD}$	40.4%		
	$MR3_{TM}$	4.5%		

These results show that $MR1_{TD}$ obtained the highest mutation score by a wide margin, while others $MR2_{AWT}$, $MR1_{TM}$, $MR1_{AWT}$ and $MR3_{TD}$ also obtained good scores. These results seem to indicate that only a few MRs are needed in order to obtain the best results, while others seem to be almost useless.

Regardless, we conclude that the metamorphic-based test oracles we developed in these experiments are sufficiently cost-effective for them to be used in practice. The main drawback of this approach is its **high cost** due to having to execute multiple test cases for the oracle. However, in our preliminary experiments, we discovered that the MRs we defined could obtain a **high (83%) mutation score** with **no false positives**. Furthermore, these oracles have the additional advantage of being **highly reusable**, since they tend to be less sensitive to configuration or environmental changes than regular oracles because the outputs of follow-up test cases are evaluated against those observed in the source test cases[26].

7. Summary

In this Deliverable we have shown how the most typical test oracles can be applied at the design-operation continuum of CPSs. Specifically, we have tackled the following aspects:

1) We have analysed how each of these oracles can be integrated with different adeptness microservices.
2) We have analysed how the test oracles can deal with the inherent uncertainty at which CPSs are exposed to. Specifically, we have incorporated three kind of uncertainties in the organic test oracles.
3) Pre-specification test oracles need to execute the test both in the version under test and in a previous version. Since this is not feasible at operation-time, we have explored the use of machine-learning algorithm to tackle such problem.
4) Lastly, we have explored the application of metamorphic testing for those situations at which determining the outcome of the CPS is unfeasible.

To validate the proposed approaches, we have carried out a series of experimental scenarios in a small-scale experiment using Orona's use-case. During other Work Packages, we will explore the application of such test oracles in both industrial use-cases and by carrying out a larger scale experimental scenarios.

[26] Segura, S., Troya, J., Durán, A., & Ruiz-Cortés, A. (2017, May). *Performance metamorphic testing: Motivation and challenges*. In 2017 IEEE/ACM 39th International Conference on Software Engineering: New Ideas and Emerging Technologies Results Track (ICSE-NIER) (pp. 7-10). IEEE.

8. Risk Register

Major risks were not identified.

9. Quality Assurance

The executive board is the body for quality assurance. The procedure for review and approval of deliverable is described in Deliverable Report D8.1 – "Project handbook". The quality will be ensured by checks and approvals by WP Leaders as part of the executive board (see front pages of all deliverables).

10. Acknowledgments

This project has received funding from the European Union's Horizon 2020 research and innovation programme under Grant Agreement no. 871319.

Disclaimer

This document reflects the views of the author(s) and does not necessarily reflect the views or policy of the European Commission. Whilst efforts have been made to ensure the accuracy and completeness of this document, the ADEPTNESS consortium shall not be liable for any errors or omissions, however caused.

Mondragon Unibertsitatea

Goi Eskola Politeknikoa
Escuela Politécnica Superior