Testing Cyber-Physical Systems under Uncertainty: Systematic, Extensible, and Configurable Model-based and Search-based Testing Methodologies

D 2.3 - Report on Uncertainty Modelling Framework V.3

Project Acronym	U-Test	**Grant Agreement Number**		H2020-ICT-2014-1. 645463	
Document Version	1.0	**Date**	2017-10-27	**Deliverable No.**	2.3
Contact Person	Phu Hong Nguyen	**Organisation**		Simula Research Laboratory	
Phone	+47 90025581	**E-Mail**		phu@simula.no	

Document Version History

Version No.	Date	Change	Author(s)
0.1	2017-01-19	Initial document outline	Phu Nguyen
0.2	2017-03-08	Detailed document outline	Phu Nguyen
0.3	2017-04-17	Section 1	Phu Nguyen
0.4	2017-04-18	New Section 3.3	Luca Berardinelli
0.5	2017-04-19	Section 2.1	SRL
0.6	2017-04-21	Section 2.2, Section 3.1	SRL
0.7	2017-05-05	New TUW Sections	Luca Berardinelli
0.8	2017-05-05	Sections related to Application level	FF
0.9	2017-05-08	Integration	SRL
0.9 Inter	2017-05-30	Intermediate version	SRL
0.95	2017-09-06	A revised version without the models of pilots	SRL
0.96	2017-10-19	Updated by SRL and TUW	SRL, TUW
0.97	2017-10-20	Last updates from FF	FF
0.98	2017-10-26	Revised according to the reviews	SRL, TUW, FF
1.0	2017-10-27	Final version	SRL, TUW, FF

Contributors

Name	Partner	Part Affected	Date
Shaukat Ali	SRL	Sections 2.1, 3.1, 4.1, 5.1	
Man Zhang	SRL	Sections 2.1, 3.1, 4.1, 5.1	
Tao Yue	SRL	Sections 2.1, 3.1, 4.1, 5.1	
Phu Hong Nguyen	SRL	All	
Marc-Florian Wendland	FF	Sections 3.2, 4.2, 5.2	
Luca Berardinelli	TUW	Sections 3.3, 4.3, 5.3	

Reviewers

Name	Partner	Part Affected	Date
Fabien Peureux	EGM	All	
César Cuevas	IKL	All	

Executive Summary

The Uncertainty Modelling Framework (UMF) provides a systematic way for standard-based holistic modelling as a Unified Modelling Language (UML) profile for the specification of uncertainty concepts in Cyber-Physical Systems (CPS). Following an Agile-like approach, this deliverable reports on the third version of the UMF (UMF V3), which is an upgrade on the two previous versions (UMF V1 and UMF V2, reported in the deliverables D2.1 [3] and D2.2 [4]). Specifically, the UMF V3 improves from the UMF V2 by 1) improving the modelling profiles; and 2) finalizing the modelling methodology such as the guidelines for applying the UML Testing Profile (UTP), modelling indeterminacy sources, integrating formal fitness factor provider, and extending library for supporting rule-based evolution strategy. We have showed how the UMF V3 is an upgrade on V2 for uncertainty modelling at the application, infrastructure, and integration levels of CPS.

Keywords: Cyber-Physical Systems, Uncertainty Modelling, Modelling Framework, UML, UTP

Table of Contents

Abbreviations

CPS	Cyber-Physical System
Dx	Deliverable number x
EGM	Easy Global Market
FPX	Future Position X
FF	Fraunhofer FOKUS
IEEE	Institute of Electrical and Electronics Engineers
IoT	Internet of Things
IKL	Ikerlan
MARTE	Modelling and Analysis of Real-Time and Embedded Systems
MBT	Model-Based Testing
Mx	Milestone
NMT	Nordic Medtest
OCL	Object Constraint Language
SBSE	Search-Based Software Engineering
SRL	Simula Research Laboratory
SUT	System Under Test
TR	Technical Report
TUW	Technische Universität Wien
T4UME	Tool for Uncertainty Modelling and Evaluation
U-Taxonomy	Uncertainty Taxonomy
UHS	ULMA Handling Systems
UME	Uncertainty Modelling and Evaluation
UMF	Uncertainty Modelling Framework
UTF	Uncertainty Testing Framework
UTP	UML Testing Profile
UUP	UML Uncertainty Profile
WP	Work Package

1 Introduction

This document presents the work done for the WP2's Task 2.3 in developing the third version of the Uncertainty Modelling Framework (UMF V3). The UMF provides a methodology to create and specify test-ready models based on existing modelling and testing standards. The models are based on the U-Test specific uncertainty profile providing the relevant concepts to describe uncertainty at the Application level, Infrastructure level, and Integration level of CPS. This report presents the UMF V3, achieved through iterative improvements over the UMF V2 reported in D2.2 [4]. This is the final deliverable of the WP2, which documents the complete UMF V3.

In Section 1.1, we revisit the position of UMF and its V3 in the whole U-Test workflow. The specific objectives of this deliverable are presented in Section 1.2. Then, we give in Section 1.3 an overview of the main content of this document.

1.1 U-Test Workflow

Figure 1 shows the general U-Test workflow, the position of the UMF, and its relationship with other U-Test components. We would like to recall that the UMF comprises models and profiles for uncertainty in CPS, based on the concepts introduced in U-Taxonomy [2], and Modelling and Testing Standards. The application of the UMF produces test-ready models as output. Next, the application of the Uncertainty Testing Framework (UTF) (WP4) on these models generates uncertainty-wise test cases.

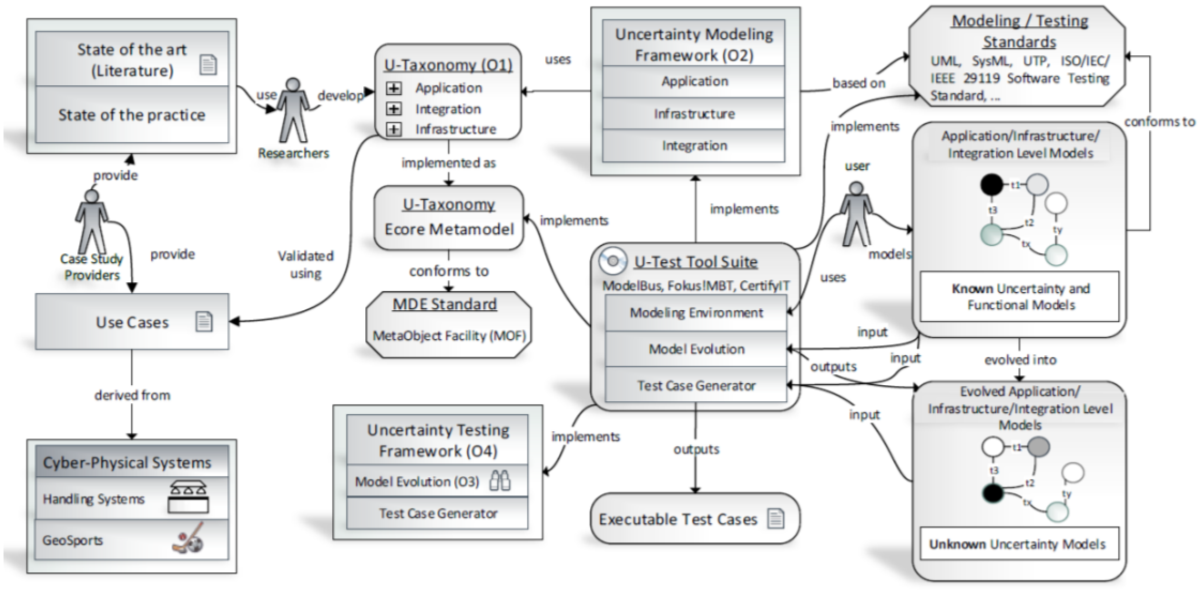

Figure 1. U-Test workflow

The UTF offers corresponding uncertainty test case generators to generate and execute adequate test cases for the U-Test use cases. Finally, the test execution results are used to evolve unknown uncertainty information (before the U-Test workflow) into known uncertainty functional models (after a walk-through across the U-Test workflow). This means that formerly unknown uncertainty behaviours go into known (uncertainty) behaviours.

In the UMF V3, we have made several improvements for better supporting uncertainty modelling and testing such as the guidelines for applying the Uncertainty Testing Profile (UTP). The UMF V3 also incorporates more libraries for facilitating the discovery of unknown uncertainties to the model evolution algorithms.

1.2 Objectives of the Deliverable

The main objectives of this deliverable are two-fold. On one hand, we report the improvements that we have made from the UMF V2 in terms of modelling methodology and supporting libraries. These improvements result in the new version of UMF, namely UMF V3. On the other hand, we show in the companion D2.4 [5] how we have used this latest version of UMF to completely model all the use cases of the two pilot systems. The test-ready models specify 100% of the Geo Sports and Warehouse Management System (WMS) scenarios, as well as the use cases described for them [1].

1.3 Structure of the Deliverable

The remainder of this document is structured as follows: Section 2 presents an overview of the UMF. We briefly describe the updates in the UMF V3, compared to the UMF V2 that we have reported in the D2.2 [4]. Then, we present in Section 3 the detailed updates in the UML Uncertainty Profile (UUP) of the UMF V3. Section 4 shows the updates of modelling methodology for supporting uncertainty modelling at three CPS levels, i.e., application, infrastructure, and integration. Section 5 concludes this document by summarizing the achievements of the UMF V3.

2 Uncertainty Modelling Framework V3

In this section, we present an overview of the UMF V3 and its updates based on the UMF V2 reported in the D2.2 [4]. Specifically, we recall in Section 2.1 the architectural overview of the UMF. Then, in Section 2.2, we present an overview of the updates to the UML Uncertainty Profile, Model Libraries and Modelling Methodology compared to the UMF V2.

2.1 An overview of the UMF

We recall in Figure 2 the architectural overview of the UMF, together with its inputs and outputs. The UMF is built on well-established and widely accepted modelling standards such as UML, UTP, and MARTE. The UMF uses as input updated requirements, together with the U-Taxonomy. The requirements for both the Geo Sports and Warehouse Management System scenarios have been updated with supplementary details. Furthermore, in the Geo Sports scenario, for supporting the test execution, in some of the use cases the X4 device has been replaced with the Quuppa device. These changes are due to use case requirements. This resulted of course in supplementary updates of the requirements detailing the behaviour of the Quuppa device.

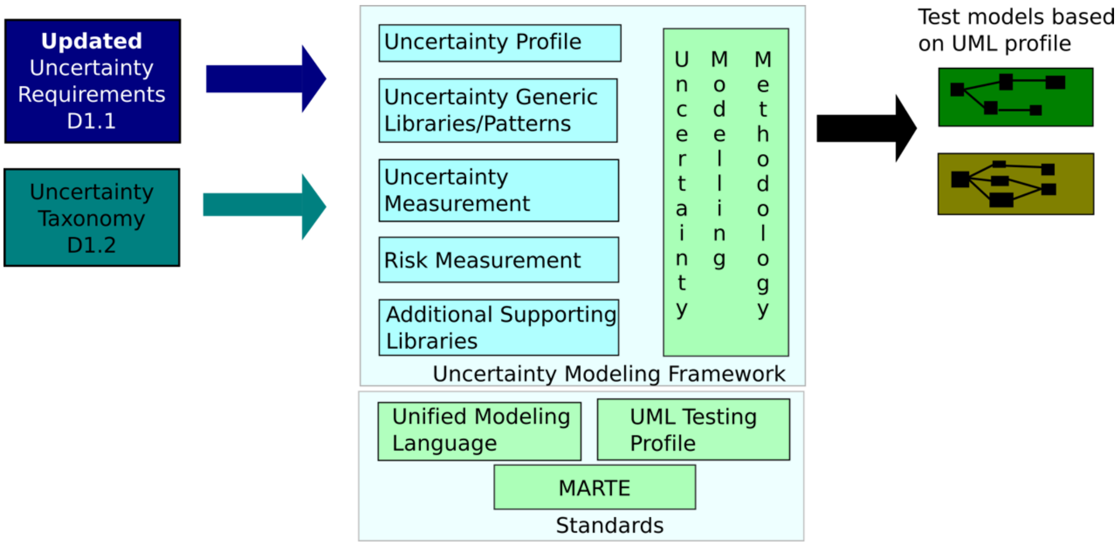
Updated Uncertainty Requirements D1.1
Uncertainty Taxonomy D1.2
Uncertainty Profile
Uncertainty Generic Libraries/Patterns
Uncertainty Measurement
Risk Measurement
Additional Supporting Libraries
Uncertainty
Modeling
Methodology
Uncertainty Modeling Framework
Unified Modeling Language
UML Testing Profile
MARTE
Standards
Test models based on UML profile

Figure 2: UMF Architecture

Our UMF defines a UML profile that provides support for uncertainty definition, management and specification. The UMF also provides methodologies for easing the usage and application of uncertainty profiles. Based on the UMF, we have developed test-ready models, for both pilot scenarios, which are described in the Deliverable 2.4 [5].

2.2 An Overview of the Updates in the UMF V3

The UMF V3 improves the UMF V2 by finalizing the modelling methodology such as the guidelines for applying UTP, for modelling indeterminacy sources, integrating formal fitness factor provider, and extending library for supporting rule-based evolution strategy.

In the following sections we describe the updates to the UML Uncertainty Profile (UUP, Section 3), Model Libraries and Modelling Methodology (Section 4) with respect to the UMF V2 reported in D2.2 [4].

3 Updates on UML Uncertainty Profile

The U-Test UML Uncertainty Profile (UUP) is divided into (1) Core Profile, (2) Application Level Profile, and (3) Infrastructure Level Profile.

3.1 Core (Integration Level) Profile

This section only highlights the updates as compared to D2.2 [4] to avoid unnecessary repetition. More specifically, there are a few updates in the core profile for modelling *Belief* and *Uncertainty*.

Since we focus on testing a CPS in the presence of environmental uncertainties, we need to introduce uncertainties in the physical environment that lead to the uncertain behaviour of the CPS. To achieve this, we have further extended the *indeterminacy source* in the profile to enable the modelling of indeterminacy sources.

Figure 3 shows the updated profile for modelling indeterminacy sources, whereas the rest of the profile is unchanged. As shown in Figure 3, we provide a set of options to model indeterminacy sources, e.g., as a UML Behaviour (e.g., State Machine) or as a constraint formulated in the Object Constraint Language (OCL). An indeterminacy source always has 1..* indeterminacy specifications, i.e. «IndeterminacySpecification» (conditions) that must be true for an indeterminacy source to occur. The «IndeterminacySourceInput» specifies the action that triggers/releases the occurrence of «IndeterminacySource». The recommendation for applying indeterminacy source profile is presented in Section 4.1. In addition, we have developed a set of options to enable indeterminacy source during testing, e.g., EnablePattern, SelectSpecification and FindPosition.

The detailed specification of these profile updates can be found in the online specification of the Core Profile available at [9] or in the technical report **TR9.pdf** accompanying this deliverable.

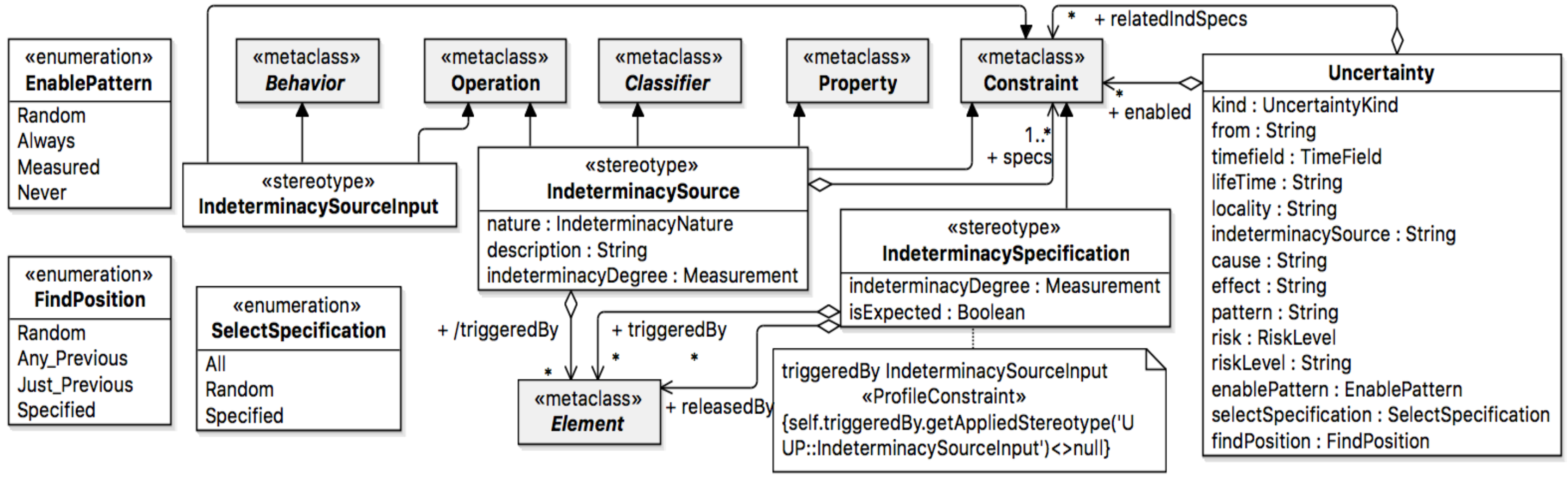
«enumeration» EnablePattern
Random
Always
Measured
Never
«metaclass» Behavior
«metaclass» Operation
«metaclass» Classifier
«metaclass» Property
«metaclass» Constraint
* + relatedIndSpecs
Uncertainty
kind : UncertaintyKind
from : String
timefield : TimeField
lifeTime : String
locality : String
indeterminacySource : String
cause : String
effect : String
pattern : String
risk : RiskLevel
riskLevel : String
enablePattern : EnablePattern
selectSpecification : SelectSpecification
findPosition : FindPosition
* + enabled
1..* + specs
«stereotype» IndeterminacySourceInput
«stereotype» IndeterminacySource
nature : IndeterminacyNature
description : String
indeterminacyDegree : Measurement
«stereotype» IndeterminacySpecification
indeterminacyDegree : Measurement
isExpected : Boolean
«enumeration» FindPosition
Random
Any_Previous
Just_Previous
Specified
«enumeration» SelectSpecification
All
Random
Specified
+ /triggeredBy
+ triggeredBy
«metaclass» Element
+ releasedBy
triggeredBy IndeterminacySourceInput «ProfileConstraint» {self.triggeredBy.getAppliedStereotype('UUP::IndeterminacySourceInput')<>null}

Figure 3. Profile Diagram of Indeterminacy Source

3.2 Application Level Profile

No changes with respect to D2.2 [4].

3.3 Infrastructure Profile

This section introduces the new version of Infrastructure CPS and Infrastructure Uncertainty Profiles, their relationship with the Uncertainty Modelling and Evaluation (UME) approach and tool (T4UE), presented in the deliverable D3.3 [7] (see Figure 4). More details about these profiles can be found in the TR **tuwien-mobiquitous2017.pdf** [8] accompanying this deliverable.

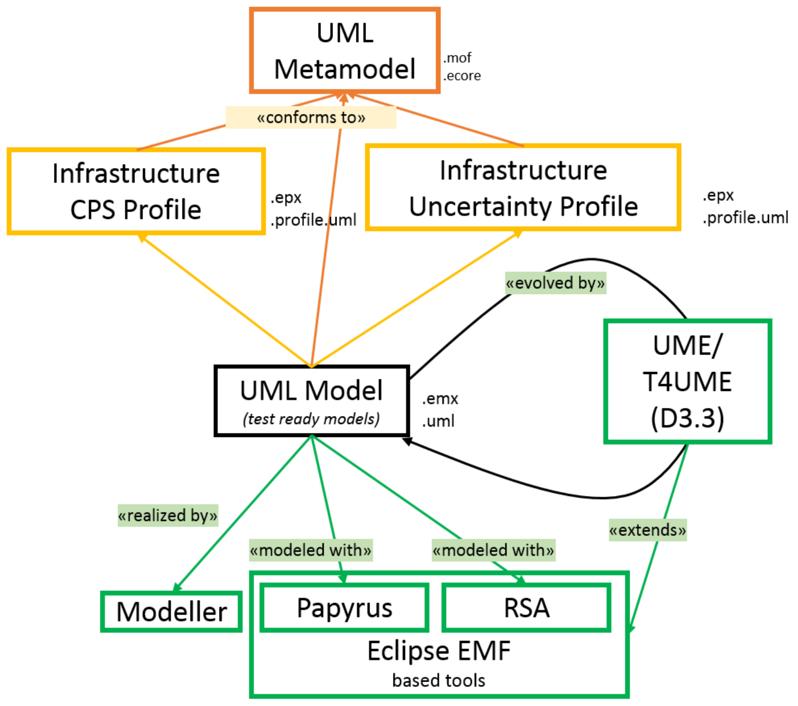

Figure 4. The Infrastructure CPS and Infrastructure Uncertainty Profile and overview.

Figure 4 shows the roles played by the Infrastructure CPS and Uncertainty Profiles with respect to U-Test UML Models and the Uncertainty Modelling and Evaluation (UME) approach further detailed in WP3.

The updated profile refines the Infrastructure CPS Profile description given in D2.2 [4] by highlighting its design rationales and its applicability on different UML diagrams (Class, Component, Composite Structure and Deployment)[1].

The aforementioned profiles are required by the UME methodology and the companion tool (T4UME) to enable uncertainty detection and model refactoring actions at design time[2].

[1] It does not prescribe the adoption of all the mentioned UML Diagram types. It is a modelling choice to represent the model content on different diagram types since valid UML Models can be obtained without the creation of any UML Diagram. Diagrams are only an effective mean to ease the communication between modellers.

[2] We replaced the Infrastructure CPS Library with modelling wizards provided by the UME methodology introduced in deliverable d3.3. Wizards allow the on demand generation of all the

The following subsections describe the Infrastructure Uncertainty and Infrastructure CPS Profiles and in detail.

3.3.1 The Infrastructure Uncertainty Profile

Based on U-Taxonomy, we updated the Infrastructure Uncertainty Profile shown in Figure 5. The Infrastructure Uncertainty profile extends the core Uncertainty stereotype. In particular, we introduce the *Uncertainty* concept as stereotype in addition to profile types (i.e., defining Class at the profile level). The «`InfrastructureUncertainty`» stereotype is characterized by the following properties, modelled as UML `Enumeration` types, namely:

- `TemporalManifestationType`,
- `LocationType`,
- `NonFunctionalDimensionalityType`,
- `CauseType`,
- `ObervationTimeType`,
- `FunctionalDimensionalityType`,
- `EffectPropagationType`.

We then identify different specialization of the «*UncertaintyFamily*» stereotype namely:

- `DataDeliveryUncertainty`,
- `ActuationUncertainty`,
- `ExecutionEnvironmentUncertainty`,
- `GovernanceUncertainty`,
- `ElasticityUncertainty`,
- `StorageUncertainty`.

Each family is characterized by a particular set of values assigned to infrastructural uncertainty properties that determine whether an uncertainty belongs to a particular family. For example, the functional dimensionality property of «StorageUncertainty» stereotype is set by default to `storage`.

stereotyped model elements originally planned to be part of static Infrastructure CPS Library that can rapidly become obsolete due to changes to profiles.

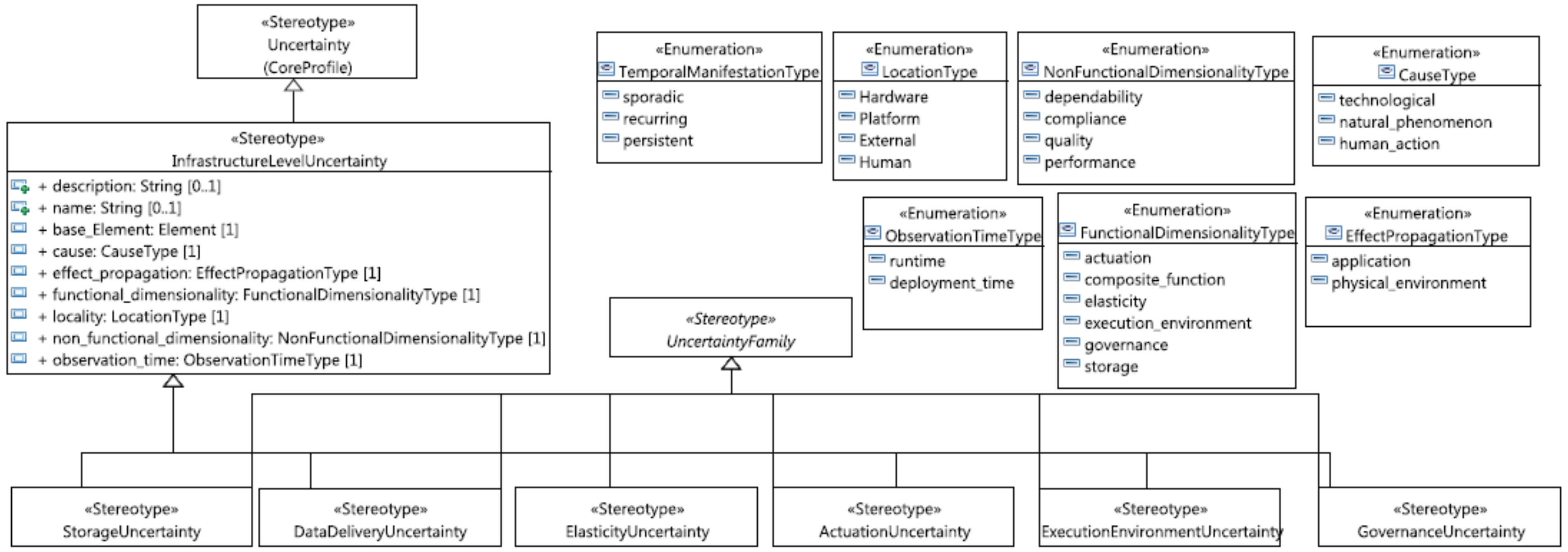

Figure 5 The up-to-date Infrastructure Uncertainty Profile.

3.3.2 The Infrastructure CPS Profile

The previous version of the CPS profile was presented in D2.2 [4]. An excerpt of the new version of the Infrastructure CPS Profile with *Stereotypes*[3] and their relationships is depicted in Figure 6.

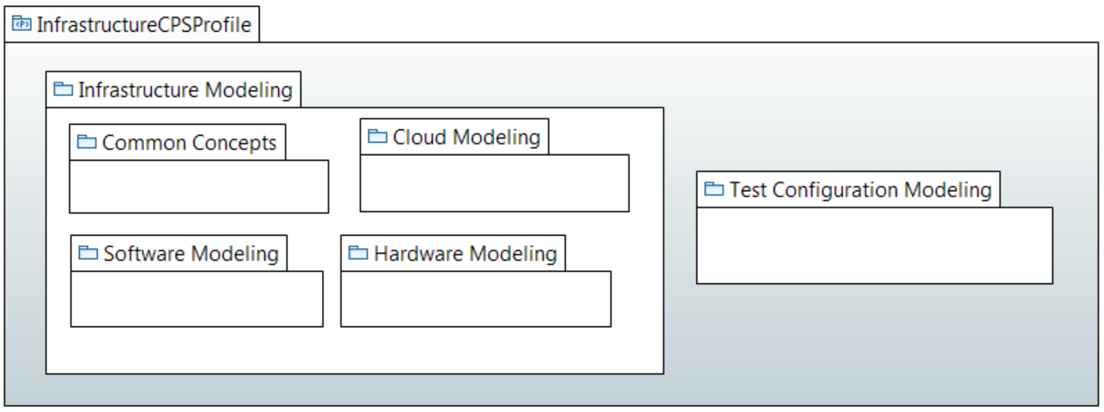

Figure 6. Package Diagram of the Infrastructure CPS Profile.

The goals of this new version are:

- To highlight the design rules behind the profile definition and to refine the extension relationships between stereotypes and UML meta-classes (e.g., replace Element meta-class with Class one) to suitably narrow the scope of stereotype applications (e.g., from any model element on any diagram to Class depicted on Class Diagrams) thus taming the complexity of modelling activity in UMF.
- To structure the profile in packages to group stereotypes with respect to the purposes (see Figure 6) they are modelling (i) edge and cloud infrastructures (`Infrastructure Modelling` package) and (ii) test configurations (`Test Configuration Modelling` package).

The next subsections detail the `Infrastructure Modelling` and `Test Configuration Modelling` packages of the new version of the Infrastructure CPS Profile.

3.3.2.1 Infrastructure Modelling Package

One of the main goals of the Infrastructure CPS Profile is representing the `Infrastructure` and its constituting `InfrastructureElements`, both physical and virtual ones, which are part of the cloud-based CPS.

Figure 7 shows an excerpt of the stereotypes and their relationships defined within the Infrastructure CPS Profile. The envisaged Infrastructure CPS includes both software and hardware `Units`. We aim at representing both kinds of units in a specular manner to provide the same modelling expressiveness, with the only exception of communication devices and protocols, as detailed later.

[3] When referring to Stereotype model element, we write the term in italics and capital letter.

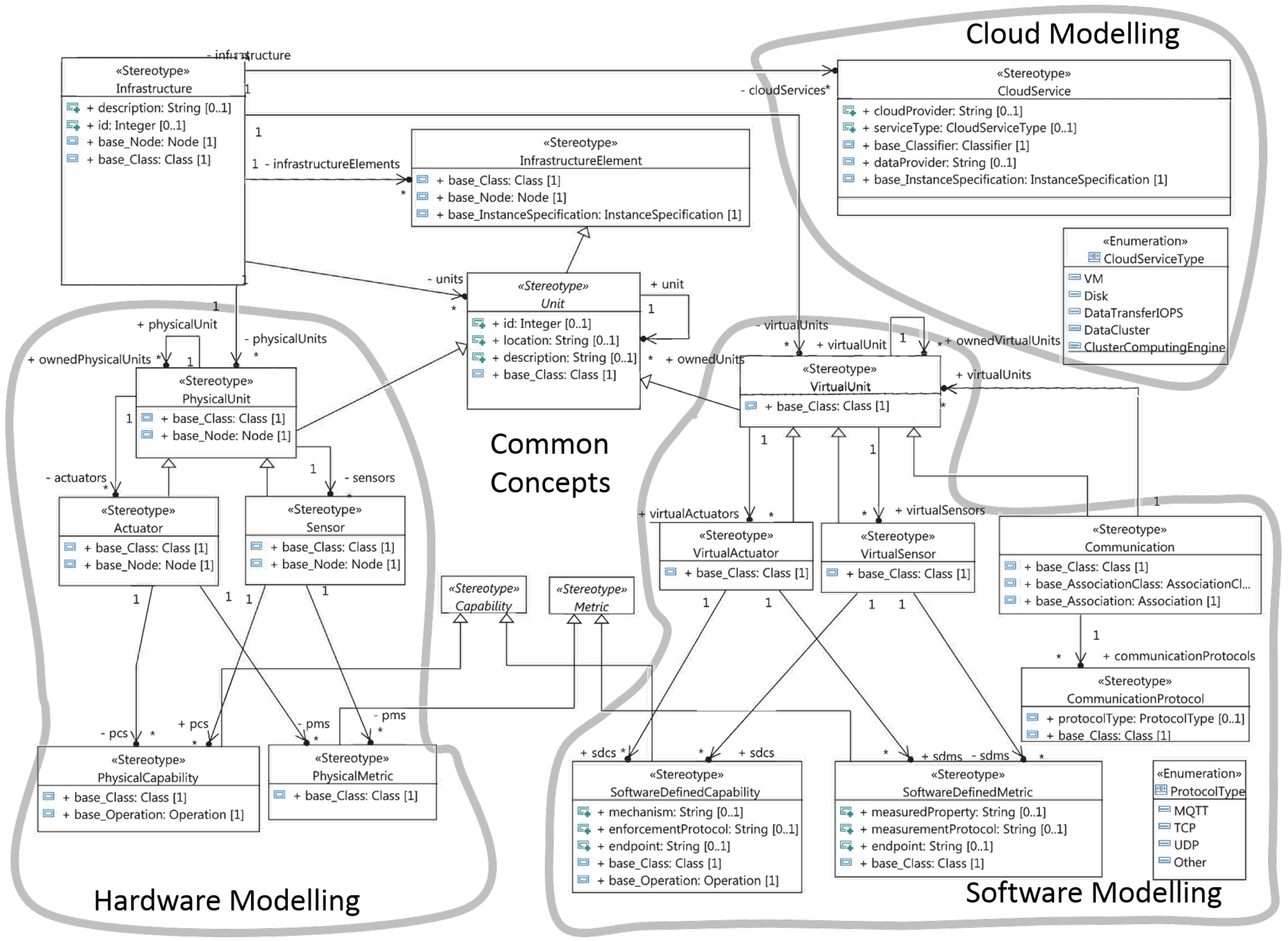
Cloud Modelling
Common Concepts
Hardware Modelling
Software Modelling
«Stereotype» Infrastructure
+ description: String [0..1]
+ id: Integer [0..1]
+ base_Node: Node [1]
+ base_Class: Class [1]
- infrastructure
- cloudServices*
1 - infrastructureElements
«Stereotype» InfrastructureElement
+ base_Class: Class [1]
+ base_Node: Node [1]
+ base_InstanceSpecification: InstanceSpecification [1]
«Stereotype» CloudService
+ cloudProvider: String [0..1]
+ serviceType: CloudServiceType [0..1]
+ base_Classifier: Classifier [1]
+ dataProvider: String [0..1]
+ base_InstanceSpecification: InstanceSpecification [1]
«Enumeration» CloudServiceType
VM
Disk
DataTransferIOPS
DataCluster
ClusterComputingEngine
- units
+ unit
«Stereotype» Unit
+ id: Integer [0..1]
+ location: String [0..1]
+ description: String [0..1]
+ base_Class: Class [1]
+ ownedUnits
- virtualUnits
+ virtualUnit
+ ownedVirtualUnits
+ virtualUnits
«Stereotype» VirtualUnit
+ base_Class: Class [1]
+ physicalUnit
- physicalUnits
+ ownedPhysicalUnits
«Stereotype» PhysicalUnit
+ base_Class: Class [1]
+ base_Node: Node [1]
- actuators
- sensors
«Stereotype» Actuator
+ base_Class: Class [1]
+ base_Node: Node [1]
«Stereotype» Sensor
+ base_Class: Class [1]
+ base_Node: Node [1]
«Stereotype» Capability
«Stereotype» Metric
- pcs
+ pcs
- pms
- pms
«Stereotype» PhysicalCapability
+ base_Class: Class [1]
+ base_Operation: Operation [1]
«Stereotype» PhysicalMetric
+ base_Class: Class [1]
+ virtualActuators
+ virtualSensors
«Stereotype» VirtualActuator
+ base_Class: Class [1]
«Stereotype» VirtualSensor
+ base_Class: Class [1]
«Stereotype» Communication
+ base_Class: Class [1]
+ base_AssociationClass: AssociationCl...
+ base_Association: Association [1]
+ communicationProtocols
«Stereotype» CommunicationProtocol
+ protocolType: ProtocolType [0..1]
+ base_Class: Class [1]
+ sdcs
+ sdcs
+ sdms
- sdms
«Stereotype» SoftwareDefinedCapability
+ mechanism: String [0..1]
+ enforcementProtocol: String [0..1]
+ endpoint: String [0..1]
+ base_Class: Class [1]
+ base_Operation: Operation [1]
«Stereotype» SoftwareDefinedMetric
+ measuredProperty: String [0..1]
+ measurementProtocol: String [0..1]
+ endpoint: String [0..1]
+ base_Class: Class [1]
«Enumeration» ProtocolType
MQTT
TCP
UDP
Other

Figure 7. The Infrastructure CPS Profile. Excerpt of the packages Common Concepts, Software Modelling, Hardware Modelling, and Cloud Modelling

Therefore, we have applied the following design rule in the definition of the Infrastructure CPS Profile, yielding four sets of stereotypes, collected in the following sub-packages:

- `Common Concepts`. This sub-package introduces generic, abstract stereotypes for concept crosscutting software and platform representations.
- `Software Modelling` and `Hardware Modelling` packages. These sub-packages define concepts to represent the software and hardware elements that build up an infrastructure. We introduce new concrete stereotypes by adding Virtual- and Physical- prefixes to the name of the generic, abstract stereotypes defined in the `Common Concepts` sub-package.
- `Cloud Modelling` package. This sub-package introduces stereotypes to represent cloud-based elements.

For the sake of explanation, we represent an excerpt of the Infrastructure CPS Profile as a flat Package Diagram where Common Concepts' stereotypes are connected to other stereotypes graphically grouped in areas named after the containing package.

The `Infrastructure` is composed of multiple, generic `Units`, each one with its own identifier, location, description, and configuration properties. In particular, a configuration represents the settings associated to the `Unit`.

`Units` can be divided in `PhysicalUnits` and `VirtualUnits` that represent hardware and software resources, respectively. Both physical and virtual units are complex elements and can be composed of other physical and virtual units, respectively.

A `PhysicalUnit` has associated `Actuators` and `Sensors`[4], which, in turn, are themselves particular kinds of `PhysicalUnits`.

An `Actuator` represents a hardware component that changes the status of the surrounding environment. Each `Actuator` realizes one or more `PhysicalCapabilities`. A `Sensor` is a component through which a `PhysicalUnit` monitors its environment (e.g., location tracker, temperature sensor, humidity sensor). Each `PhysicalUnit` has associated `Metrics` that it is capable to collect. For example, a thermostat physical unit can include both a sensor to collect temperature and humidity (i.e., the physical capability to collect two metrics), and an actuator that has the capability to modify the temperature and the humidity of the surrounding environment.

Each `PhysicalUnit` has associated one or more `VirtualUnits` that run on top of it (e.g., PLC code running and governing machines within a production system).

As anticipated, we assume a specular set of concepts to describe the software architecture of the Infrastructure CPS. A `VirtualUnit` has associated `VirtualActuators` and a `VirtualSensors`, which, in turn, are themselves particular kinds of `VirtualUnits`.

A `VirtualActuator` represents a software component through which the owning `VirtualUnit` controls the hardware platform elements that interact with the environment. Each `VirtualActuator` realizes one or more `VirtualCapabilities`.

A `VirtualSensor` is a software component through which physical sensors are controlled. Each `VirtualUnit` has associated one or more `SoftwareDefinedMetrics` that it is capable to collect (e.g., logical representations of physical measures like temperatures are float variables). The `SoftwareDefinedMetric` has an `id, name, description, endpoint, period, measuredProperty`, and `measurementProtocol`. These are the attributes needed for accessing the sensor information.

[4] In this case, terms like Actuators and Sensors are definitely hardware-specific terms and we choose to not add Physical- prefix to the corresponding stereotypes.

Concerning the modelling of communication units and protocols, it is worth noting that we did not introduce a specific stereotype for communication devices, like routers or cables, but we plan to model them as `PhysicalUnits`. On the contrary, we introduce a specific `Communication` stereotype to model interactions between `VirtualUnits`. Each communication realizes a particular `ProtocolType` between different infrastructure elements. The supported protocols values are MQTT, HTTP, TCP, UDP, AMQP, and STOMP.

We then map stereotypes to UML meta-classes to determine which model elements they can be applied to and, then, in which UML diagrams they can appear. In Figure 7, the extended UML meta-class is referred by the «*base_class*» stereotype property. All the aforementioned stereotypes define structural modelling concepts. For this reason, the chosen UML meta-classes are `Class`, `Component`, and `InstanceSpecification` for both software and platform related concepts. Therefore, the UML modeller can (i) define new infrastructural element types via `Classes`, `Components` and `Nodes`, (ii) instantiate typed objects via `InstanceSpecification`, and (iii) depict them on `Class`, `Component`, and `Deployment` diagrams.

3.3.2.2 Test Configuration Modelling Package

Figure 8 shows a set of stereotypes from the Infrastructure CPS Profile defined to represent, in UML, test configurations. These stereotypes complement those included in the *Infrastructure Modelling* package depicted in Figure 7.

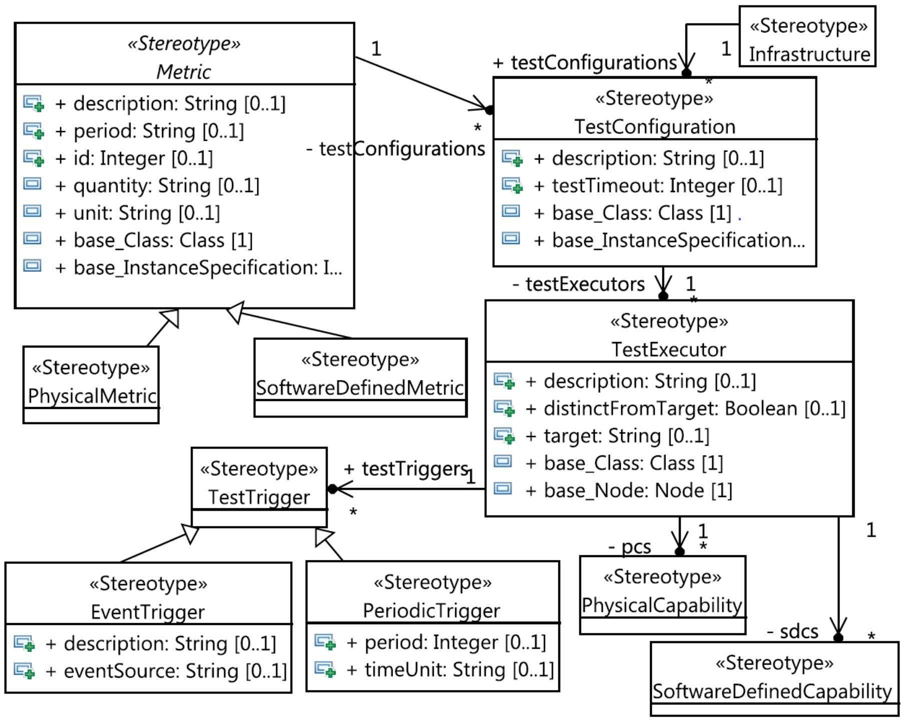
«Stereotype» Metric
+ description: String [0..1]
+ period: String [0..1]
+ id: Integer [0..1]
+ quantity: String [0..1]
+ unit: String [0..1]
+ base_Class: Class [1]
+ base_InstanceSpecification: I...
«Stereotype» PhysicalMetric
«Stereotype» SoftwareDefinedMetric
«Stereotype» Infrastructure
+ testConfigurations
- testConfigurations
«Stereotype» TestConfiguration
+ description: String [0..1]
+ testTimeout: Integer [0..1]
+ base_Class: Class [1]
+ base_InstanceSpecification...
- testExecutors
«Stereotype» TestExecutor
+ description: String [0..1]
+ distinctFromTarget: Boolean [0..1]
+ target: String [0..1]
+ base_Class: Class [1]
+ base_Node: Node [1]
+ testTriggers
«Stereotype» TestTrigger
«Stereotype» EventTrigger
+ description: String [0..1]
+ eventSource: String [0..1]
«Stereotype» PeriodicTrigger
+ period: Integer [0..1]
+ timeUnit: String [0..1]
- pcs
«Stereotype» PhysicalCapability
- sdcs
«Stereotype» SoftwareDefinedCapability
1
*

Figure 8. Test Configuration Modelling package

For testing CPS, we associate a `TestConfiguration` to `Metric`, being an extension point for further types of tests. The `TestConfiguration` has a name, description and a `testTimeout`. The `testTimeout` gives the maximum amount of time in which the associated `TestExecutor` should answer the test. The `TestExecutor` has an association to `Capability` or to `SoftwareDefinedCapability`, for the case in which certain settings need to be done on the CPS before the test is executed. The `TestExecutor` owns a description, a Boolean stating whether the unit executing this test is different from the target of the test, and a target describing the target of the test.

Each TestConfiguration has associated a TestTrigger, which describes when the test should be executed. The TestTrigger is of two types, either EventTrigger or PeriodicTrigger. The EventTrigger is used for event-based testing (e.g., when, during system runtime, the quality is too low). The EventTrigger has two attributes: the description of the event, and the eventSource. The PeriodicTrigger is used for tests executed in specific periods described under various units of time. The PeriodicTrigger has two attributes: the period and the timeUnit.

Finally, any *CPS* is equipped with CloudServices of different types (see CloudServiceTypes enumeration including VM, Disk, StorageService, and DataAnalyticsEngine) corresponding to cloud offerings by cloudProvider and dataProvider.

It is worth noting that we have designed the stereotypes belonging to the Test Configuration Modelling package to be applicable on both Classes and InstanceSpecification model elements. Therefore, the UMF user can specify tests both referring to SUT architectural types (i.e., Classes, Operations and Associations depicted on Class Diagrams) or use case specific configurations made of graph of connected architectural instances (i.e., InstanceSpecification, Links and Slot values[5]).

4 Updates on Modelling Methodology

This section presents the updates in the modelling methodology at the integration level (Section 4.1), application level (Section 4.2), and infrastructure level (Section 4.3) of CPS.

4.1 Updates in the Methodology at the Integration Level

The main updates of the methodology at the integration level are represented as orange parts in Figure 9: Validation Guidelines, and Design Decisions & Recommendations. For the complete details, please see the **TR6.pdf [10]** accompanying this deliverable.

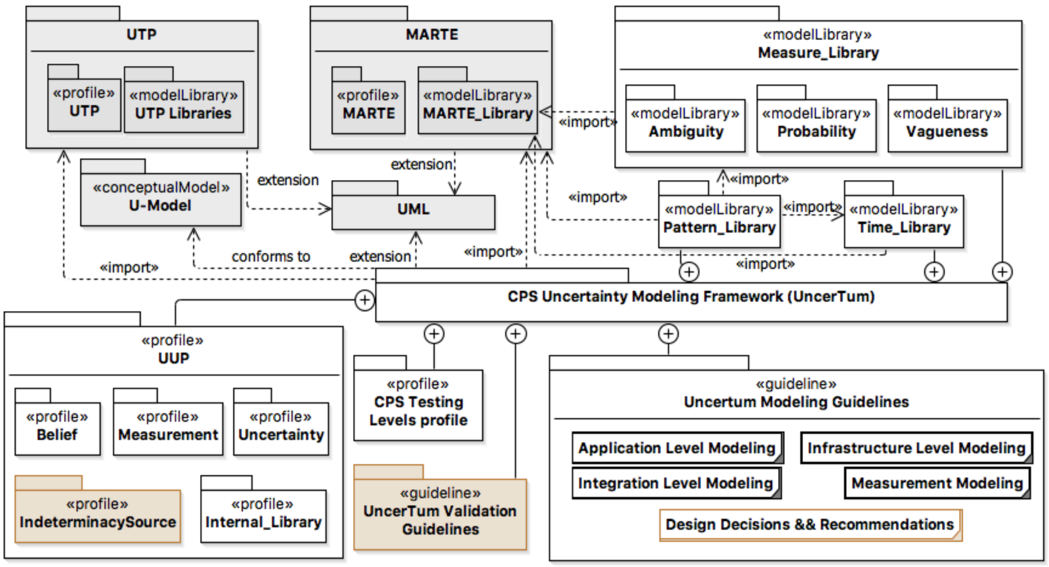
UTP
«profile» UTP
«modelLibrary» UTP Libraries
MARTE
«profile» MARTE
«modelLibrary» MARTE_Library
«modelLibrary» Measure_Library
«modelLibrary» Ambiguity
«modelLibrary» Probability
«modelLibrary» Vagueness
«import»
extension
«conceptualModel» U-Model
extension
UML
«import»
«modelLibrary» Pattern_Library
«import»
«modelLibrary» Time_Library
conforms to
extension
«import»
«import»
«import»
CPS Uncertainty Modeling Framework (UncerTum)
«profile» UUP
«profile» Belief
«profile» Measurement
«profile» Uncertainty
«profile» IndeterminacySource
«profile» Internal_Library
«profile» CPS Testing Levels profile
«guideline» UncerTum Validation Guidelines
«guideline» Uncertum Modeling Guidelines
Application Level Modeling
Infrastructure Level Modeling
Integration Level Modeling
Measurement Modeling
Design Decisions && Recommendations

Figure 9. The Overview of UncerTum (orange is updated)

[5] See the U-CertifyIt modelling methodology, where test configurations are specified at the instance level

To facilitate the construction of test-ready models with our methodology, we have listed a set of design decisions and recommendations as shown in Figure 9. They are summarized, along with the rationales behind in Table 4 of the aforementioned **TR6.pdf [10]**.

To ensure that test-ready models are syntactically correct and communication across state machines of various physical units constituting a CPS takes place correctly, we have developed the validation process (Figure 9) with step-wise guidelines and a set of recommendations to fix problems in test-ready models. Such validation is aimed at finding modelling errors that may have been introduced by a test modeller accidentally. Once test-ready models have been successfully validated, test cases can be then generated from them. Since the execution of test-ready models requires data to execute triggers, we generate data as follows: 1) if a trigger (Call Event/Signal Event) is guarded by a guard condition, we generate random values for all the variables involved in the guard condition that satisfy the guard condition and use these values to fire the trigger, and generate random values for all the other parameters of the call event/signal event, 2) if a trigger (Call Event/Signal Event) is not guarded, we generate random values for all the parameters of the Call Event/Signal Event to fire the trigger, 3) if a trigger corresponds to a Change Event, we randomly generate values that satisfy the change condition, 4) if a trigger corresponds to a Time Event, we ensure that the specified period of time in the event is elapsed. For more details, see Section 8 of the **TR6.pdf [10]** accompanying this deliverable.

To ease the modeling indeterminacy source, we summarize our recommendations for applying the indeterminacy source profile (Table 1) and update the activity diagram for modelling indeterminacy source (Figure 10). For more details, see Section 2.1.3 and 4.5.2.B of the **TR9.pdf** [9] accompanying this deliverable.

Table 1. Recommendations for applying the Indeterminacy Source part of the UUP

#	Stereotype Applied		Base Element
S1: States of the environment of the CPS are indeterminate, such as the batteryStatus.			
R1	«IndeterminacySource»		Property
	«IndeterminacySpecification»		Constraint
	Op1	«IndeterminacySourceInput»	Operation
	Op2	«IndeterminacySourceInput»	Operation, Constraint
R2	«IndeterminacySource»		Constraint
	«IndeterminacySpecification»		FALSE (default)
	Op1	«IndeterminacySourceInput»	Operation
	Op2	«IndeterminacySourceInput»	Operation, Constraint
S2: Input data is indeterminate.			
R1	«IndeterminacySource»		Operation
	«IndeterminacySpecification»		Constraint
	«IndeterminacySourceInput»		Constraint
S3: Occurrences of an event from the environment (e.g., "pressing the button") are indeterminate.			
R1	«IndeterminacySource»		Property
	«IndeterminacySpecification»		Constraint
	Op1	«IndeterminacySourceInput»	Operation
	Op2	«IndeterminacySourceInput»	Operation, Constraint
R2	«IndeterminacySource»		Constraint
	«IndeterminacySpecification»		FALSE (default)
	Op1	«IndeterminacySourceInput»	Operation
	Op2	«IndeterminacySourceInput»	Operation, Constraint

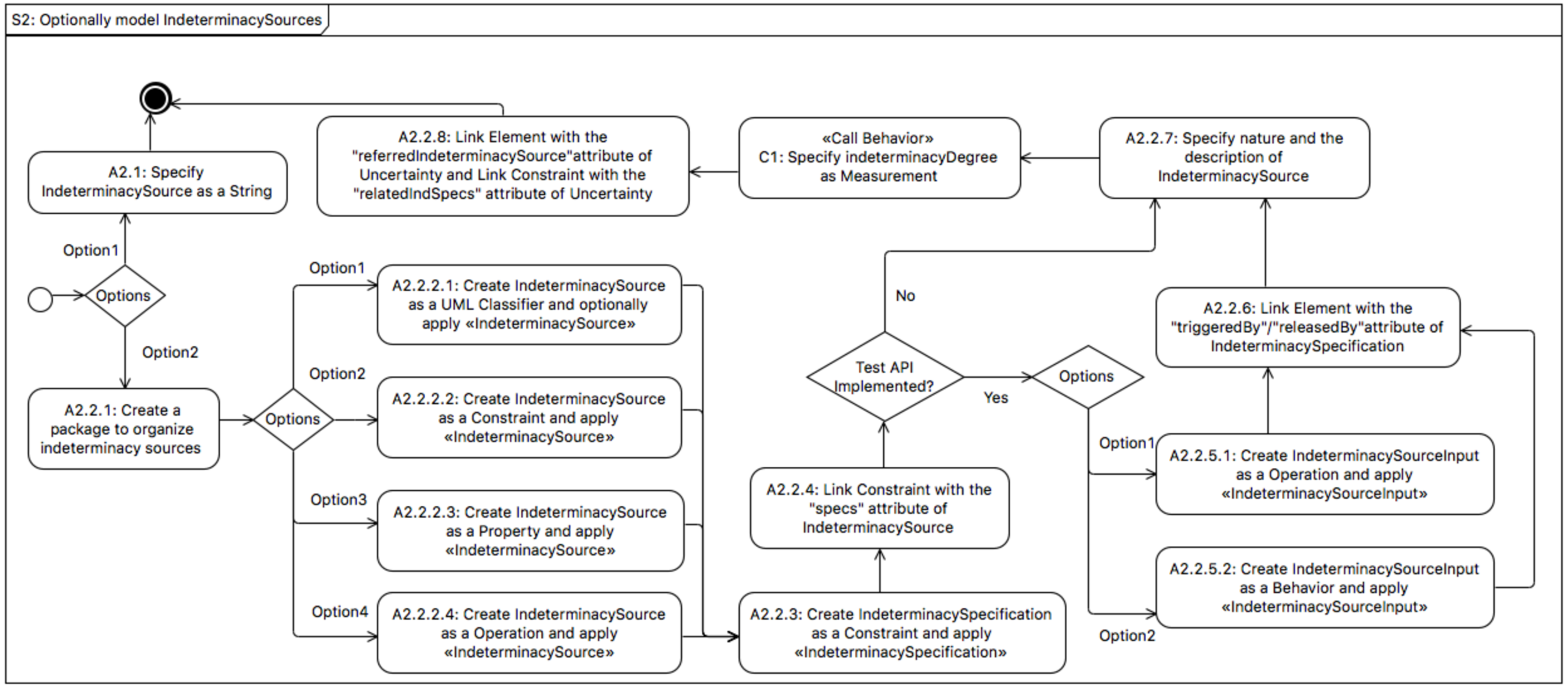
S2: Optionally model IndeterminacySources
A2.1: Specify IndeterminacySource as a String
Option1
Options
Option2
A2.2.1: Create a package to organize indeterminacy sources
Options
Option1
Option2
Option3
Option4
A2.2.2.1: Create IndeterminacySource as a UML Classifier and optionally apply «IndeterminacySource»
A2.2.2.2: Create IndeterminacySource as a Constraint and apply «IndeterminacySource»
A2.2.2.3: Create IndeterminacySource as a Property and apply «IndeterminacySource»
A2.2.2.4: Create IndeterminacySource as a Operation and apply «IndeterminacySource»
A2.2.3: Create IndeterminacySpecification as a Constraint and apply «IndeterminacySpecification»
A2.2.4: Link Constraint with the "specs" attribute of IndeterminacySource
Test API Implemented?
No
Yes
Options
Option1
Option2
A2.2.5.1: Create IndeterminacySourceInput as a Operation and apply «IndeterminacySourceInput»
A2.2.5.2: Create IndeterminacySourceInput as a Behavior and apply «IndeterminacySourceInput»
A2.2.6: Link Element with the "triggeredBy"/"releasedBy"attribute of IndeterminacySpecification
A2.2.7: Specify nature and the description of IndeterminacySource
«Call Behavior» C1: Specify indeterminacyDegree as Measurement
A2.2.8: Link Element with the "referredIndeterminacySource"attribute of Uncertainty and Link Constraint with the "relatedIndSpecs" attribute of Uncertainty

Figure 10. Model *IndeterminacySource*

Table 2 summarizes the updates in the UMF at the Integration level.

Table 2. Overall Updates in the UMF

Category	**Name**	**Update Status**	**Details in Section #**
Profile	`IndeterminacySource`	Updated	Section 3.1 and details in Section 2.1.1 - 2.1.3 of the **TR9.pdf** or online specification [9].
Profile	`IndeterminacyInput`	New	
Profile	`IndeterminacySpecification`	New	
Profile	`SelectSpecification`	New	
Profile	`FindPosition`	New	
Profile	`EnablePattern`	New	
Methodology	15 design decisions and recommendations	New	Section 4.1 and details in Table in Table 4 of the **TR6.pdf [10]**
Methodology	The activity diagram to enable validate process	New	Section 4.1 and details in Section 8 of the **TR6.pdf [10]**
Methodology	9 recommendations to fix problems in test-ready model.	New	Section 4.1 and details in Section 8 of the **TR6.pdf [10]**
Methodology	The recommendations of applying indeterminacy source profile	New	Section 4.1 and details in Section 4.5.2.B of the **TR9.pdf** or online specification [9].
Methodology	The activity diagram to model indeterminacy source	Updated	Section 4.1 and details in Section 4.5.2.B of the **TR9.pdf** or online specification [9].

4.2 Updates of the modelling methodology at the Application level

The application level modelling methodology remains almost stable with respect to D2.2 [4]. Still, there are minor updates to the pilot modelling process and deployment modelling process, which are summarized in the subsequent sections.

4.2.1 Pilot modelling process

Update of Transition's effects

Until D2.2 [4], the effects of transitions and the body of auxiliary operations[6] were defined by means of fUML-compliant Activities and visualized as activity diagrams. Although this approach was very adequate from a methodological, semantic and/or technical perspective, it failed in terms of usability. Usability is, however, particularly critical with respect to the development of effects (or, in general, any other behavioural statement that needs to be executed by a test case generator). At the beginning of the U-Test project, when first

[6] Operations that help in the generation of test cases by executing some re-occurring functionality but do not reflect any external interface of the actual SUT implementation.

considerations about the U-Test UMF where done, it was expected to utilize the Papyrus Alf editor, which produces fUML-Activities from textual Alf statements. Back in those days, this was considered (and it still is) the easiest, yet most elegant and usable way to develop transition effects and auxiliary methods. Unfortunately, the Papyrus Alf editor did not mature in time for a reliable usage within the U-Test project (it was still not fully-fledged and mature in the Eclipse Mars release, i.e., after M18 of the U-Test duration).

After reconsidering how to circumvent this pure technical issue, we decided to go for a direct modelling of the corresponding activity diagrams – corresponding in terms of the activity diagrams that would have been generated by the Papyrus Alf editor. Although, from a semantic point of view, this was exactly what was required for the application level process, changes to the use cases and transitively to the affected activity diagrams required a huge and ineffective maintenance effort. Furthermore, the end users would have to be experts in modelling the executable activity diagrams on which we relied. Since this would impose a too high technical barrier on the industrial adoption of the U-Test application level, we decided, after the experiences of D2.2 [4] with activity modelling, to even go one-step back and directly include the required C# code. This C# snippet was intended to be generated from the executable activity diagrams beforehand in order to generate test cases from the underlying C#-based engine for test generation.

Due to missing stability of the Papyrus Alf Editor at first, and the infeasible activity modelling barrier at second, the final update of the pilot modelling process has shifted towards the direct usage of C# code to express transition effects and auxiliary behaviour.

4.2.2 Uncertainty modelling

No changes with respect to D2.2 [4].

4.2.3 Deployment modelling

The specification of test directives was already described in D2.2 [4] to a certain degree. Since then, the test directive-based modelling methodology was further developed in order to simplify the entire deployment and automation process. There are now three different kinds of test directives defined for the UMF, technically represented as sub-types of the UTP stereotype «*TestDesignDirective*». These are:

- `UTFTestDirective;`
- `TestCaseGenerationDirective;`
- `TestCodeGenerationDirective;`
- `TestExecutionDirective;`
- `UTFModelEvolutionConfiguration and GenerationSizeStrategy.`

These test directives serve different purposes in the UMF test automation architecture (Fokus!MBT U-Test). They aim at a completely automated test process without any manual intervention.

The `UTFTestDirective` is a test directive specially tailored for U-Test that acts as a container for the test automation process required directives. The `UTFTestDirective` acts as the executable entry point into the automated uncertainty testing process. It combines the directives for both automated test generation and automated test execution. When the user executes the `UTFTestDirective`, the entire automated tool chain will be setup and configured automatically. The result of the test generation sub-process (if successful) is a number of UTP test scripts.

Technically, a `UTFTestDirective` is defined as an extension of the UTP 2 test design facility, i.e. a stereotype that specializes the UTP stereotype «*TestDesignDirective*». The corresponding profile specification is shown in Figure 11.

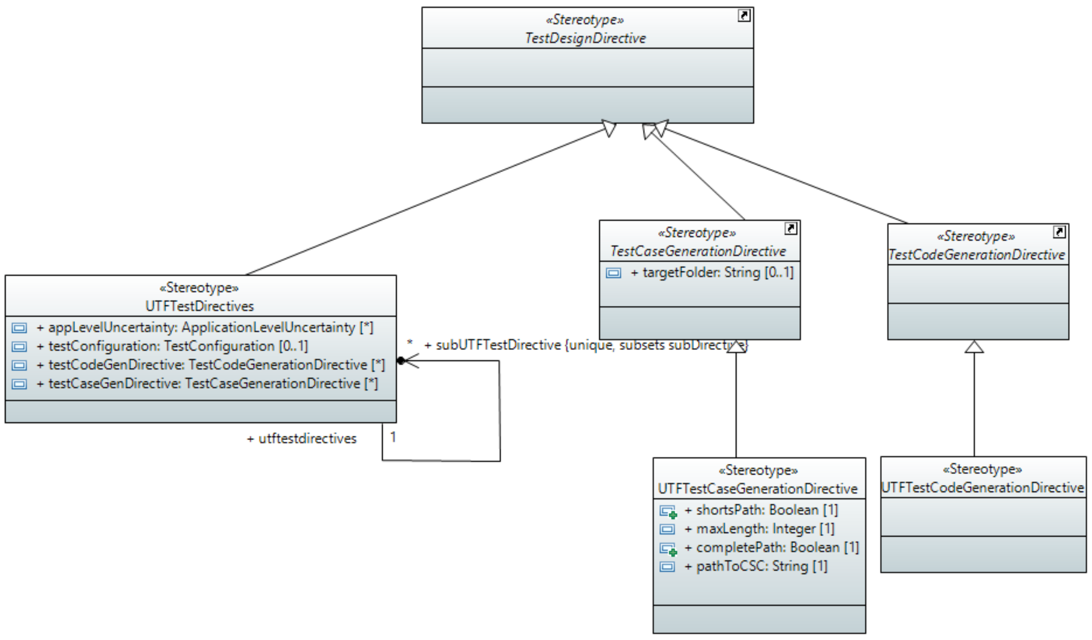
«Stereotype» TestDesignDirective
«Stereotype» UTFTestDirectives
+ appLevelUncertainty: ApplicationLevelUncertainty [*]
+ testConfiguration: TestConfiguration [0..1]
+ testCodeGenDirective: TestCodeGenerationDirective [*]
+ testCaseGenDirective: TestCaseGenerationDirective [*]
* + subUTFTestDirective {unique, subsets subDirective}
+ utftestdirectives 1
«Stereotype» TestCaseGenerationDirective
+ targetFolder: String [0..1]
«Stereotype» TestCodeGenerationDirective
«Stereotype» UTFTestCaseGenerationDirective
+ shortsPath: Boolean [1]
+ maxLength: Integer [1]
+ completePath: Boolean [1]
+ pathToCSC: String [1]
«Stereotype» UTFTestCodeGenerationDirective

Figure 11. Specification of the UTFTestDirective

The stereotypes «*UTFTestCaseGenerationDirective*» and «*UTFTestCodeGenerationDirective*» are similar concepts but only for the purpose of test case generation and generation of executable test scripts. Both parts represent the corresponding manual activities in generating logical test cases and implementing executable test cases.

The `UTFModelEvolutionConfiguration` serves the purpose of configuring the model evolution in terms of population size, crossover activation (see D3.3 [7] for further details) and the generation size strategy. The generation size strategy determines how many generations shall be generated by search-based algorithm for revealing new uncertain behaviour of the system under test. The application level UMF supports currently a fix generation size strategy (represented by the stereotype «`FixGenerationSize`») that when certain number of generations has been generated, used for test case generation and test case execution. The long-term goal in this regard is to replace the fix generation size strategy with a dynamic generation size strategy. This one would then use the outcome of each test case execution of each generation in order to check whether the desired result (in terms of a fitness factor threshold) has been found. This would allow an optimized automation process based on the search-based problem as opposed to finishing testing after a fix number of generations.

The `TestExecutionDirective` abstracts from concrete deployment modelling and creates a deployment specification for automated test case execution (automated start of the execution of the test cases that have been generated). Furthermore, the entire test execution system (JUnit in case of U-TEST project) is automatically setup and the JUnit engine gets started. After test execution, the test logs are feedback and evaluated for eventual verdict calculation.

Furthermore, the `UTFTestDirective` couples the known application level uncertainties (which serve as the basis for the model evolution algorithm in order to detect further unknown uncertainties) with a concrete U-Test test strategy (the different strategies are described in detail in D3.2 [6]). The underlying test automation engine exploits this information to configure and steer the model evolution implementation later on. The UML profile for the application level test strategies is shown in Figure 12.

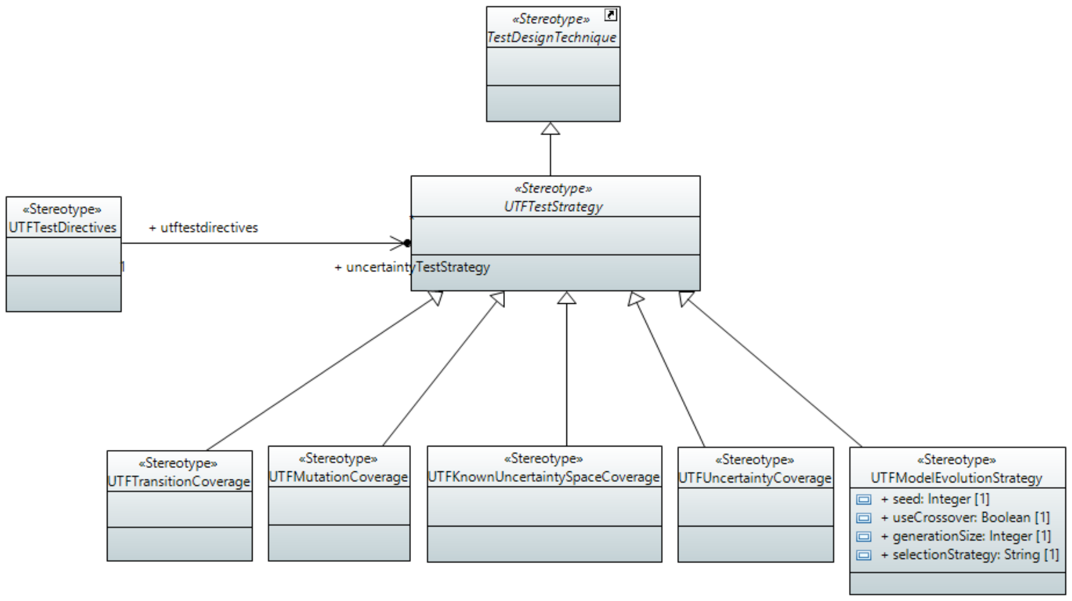

Figure 12. UML profile for application level UMF

Summary

With respect to the deployment of eventually executable test cases, a `UTFTestDirective` encapsulates all information and generates the respective UML deployment specification automatically, instead of imposing these subtle modelling steps on the test engineer. These changes in the definition of the deployment modelling process are a major step towards the simpler application, increase usability of the UMF. For these reasons, it has been implemented in the recent version of the Fokus!MBT UTEST.

4.3 Updates of the Infrastructure level modelling methodology

The infrastructure level profile has been further extended to model different kinds of infrastructural elements (IoT hardware/software components and cloud services) as types (by extending the UML Class meta-class) and instances (by extending the UML `InstanceSpecification` meta-class). Since the evolution process of non-standard profiles is supposed to continue to accommodate different needs of users and tool vendors, we introduce in D3.3 [7] a methodology and tool that adapt to changes to profiles applied to UML Model.

4.4 Test-Ready Models for Pilots

The application of the UMF yields test-ready models as outputs, which are based on UML and the UTP and UUP profiles. In general, test modellers should be guided by the provided modelling methodology in order to create these test-ready models. The goal is to produce models that are defined at a sufficient level of detail to generate adequate test cases. The test-ready models for both pilots are described in the Deliverable D2.4 [5].

5 Summary

In general, we have made significant improvements from UMF V2 to UMF V3 in terms of completing the modelling methodology, and the modelling support such as the UUP for better supporting modelling indeterminacy source. Moreover, we have also shown the application of UMF V3 in completing the test-ready models for the two industrial case studies (D2.4 [5]). The detailed achievements in UMF V3 at the CPS's integration level, application level, and infrastructure level are summarized as follows.

5.1 UMF for Integration Level

We have reached the milestone Mx4 regarding the UMF V.3 (final version) for uncertainty modelling at the integration level of CPS. Comparing with UMF V.2, the main improvements, as shown in Figure 9, (page 18) for uncertainty modelling at the integration level include:

1) The update of the UUP for supporting modelling indeterminacy source, which leads to occurrence of the uncertainties.
2) The implementation of the validation process to ensure the syntactically correctness of test-ready models.
3) The summarized design decisions and recommendations to provide modellers with options to construct test-ready models using UMF V.3 at the integration level.

We present in the accompanying D2.4 [5] the implementation of 100% use cases for the two case studies at the integration level.

5.2 UMF for Application Level

Milestone Mx4 (UMF V.3) has been successfully reached with respect to uncertainty modelling at the application level. The major updates in UMF V.3 targeted the following aspects:

- Pilot modelling process: Use of C# action code instead of fUML-compliant Activities due to technical reasons.
- Deployment modelling process: Integration of U-Test-specific test directives to drive the entire dynamic test process

5.3 UMF for Infrastructure Level

We have reached the milestone Mx4 regarding the UMF V.3 for uncertainty modelling at the infrastructure level of CPS. When compared to UMF V.2, the main improvements for infrastructure level modelling at the infrastructure level include:

- Updated Infrastructure Uncertainty Profile. We introduced stereotype definitions in addition to profile types (i.e. Classes defined in UML profiles), and new enumerated types to distinguish among different kinds of infrastructure uncertainty families.
- Updated InfrastructureCPS Profile with clear distinction among IoT and cloud infrastructural elements.

It is worth noting that we introduce wizards as part of the Uncertainty Modelling and Evaluation methodology and tool in D3.3 [7] to automatically adapt infrastructure level modelling guidelines to changes in profiles applied to UML models.

6 Bibliography

[1] *D1.1: U-Test Deliverable Report on Requirements Collection.*

[2] *D1.2: U-Test Deliverable Report on Uncertainty Taxonomy.*

[3] *D2.1: U-Test Deliverable Report on Uncertainty Modelling Framework (UMF) V1.*

[4] *D2.2: U-Test Deliverable Report on Uncertainty Modelling Framework (UMF) V2.*

[5] *D2.4: U-Test Deliverable Report on Uncertainty Modelling Framework (UMF) V3: The Test-Ready Models of Pilots.*

[6] *D3.2: U-Test Deliverable Report on Uncertainty Testing Framework V.2.*

[7] *D3.3: U-Test Deliverable Report on Uncertainty Testing Framework V.3.*

[8] Truong, H.-L., L. Berardinelli, I. Pavkovic, and G. Copil. *Modeling and Provisioning IoT Cloud Systems for Testing Uncertainties*. in *The 14th EAI International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services (MobiQuitous)*. 2017. Melbourne, Australia: ACM.

[9] Zhang, M., S. Ali, T. Yue, and P.H. Nguyen, *Uncertainty Modeling Framework for the Integration Level V.3*. Technical Report, 2016. 2016-01. Available from: https://www.simula.no/publications/uncertainty-modeling-framework-integration-level-v2.

[10] Zhang, M., S. Ali, T. Yue, R. Norgren, and O. Okariz, *Uncertainty-Wise Cyber-Physical System test modeling.* Software & Systems Modeling, 2017.